CS2136: Paradigms of Computation

Class 07:

Creating & Altering the Database
Prolog & Logic Programming
First-Order Predicate Calculus

Creating and Altering the Database

Creating the Database

- Some Prolog implementations let you type in facts and rules.
- SWI-Prolog compiles facts and rules when you load in a Prolog source file.

Creating the Database II

- Clocksin & Mellish:
  - consult(filename)
    - Adds clauses to the in-memory database.
  - reconsult(filename)
    - Supersedes clauses for same predicate.
    - Adds clauses for new predicates.
- SWI-Prolog:
  - consult(filename)
    - Acts like C & M reconsult.

Altering the Database

- Clocksin & Mellish:
  - asserta(clause).
    - Inserts clause at beginning of database.
  - assertz(clause).
    - Inserts clause at end of database.
  - retract(clause).
    - Removes clause from database.

Altering the Database II

- SWI-Prolog:
  - asserta, assertz, retract
    - Same as previous slide.
  - assert(clause)
    - Same as assertz.

Logic Programming

Logic Programming

- Basic Idea:
  - Don’t tell the program what to do when.
  - Tell it what is true and let it draw conclusions.
- Looked at another way:
  - Don’t say how to solve the problem.
  - Say what a solution looks like.
- Yet another way:
  - Declarative semantics.
  - Not procedural semantics.

Relating Prolog to Logic Programming

- Consider the Prolog statement:
  - mother(X,Y) :- parent(X,Y),female(Y).
- This...
  - Expresses a proposition we hypothesize is true.
  - Says how to show that someone is a mother.

Predicate Calculus

- Not directly related to the differential or integral calculus you already know.

What is "Calculus"?

- Calculus (from www.dictionary.com):
  1. Pathology. An abnormal concretion in the body, usually formed of mineral salts and found in the gallbladder, kidney, or urinary bladder, for example.
  2. Dentistry. Tartar.

  (continued...)

What is "Calculus"?

- Calculus (from www.dictionary.com):
  3. Abbr. calc. Mathematics
     a. The branch of mathematics that deals with limits and the differentiation and integration of functions of one or more variables. [What we usually think of.]
     b. A method of analysis or calculation using a special symbolic notation. [What we will talk about today.]
     c. The combined mathematics of differential calculus and integral calculus. [What we usually think of.]
  4. A system or method of calculation: " [a] dazzling grasp of the nation's byzantine budget calculus" (David M. Alpern).

Predicate Calculus: First- and Second-Order

- predicate calculus - an extension of the propositional calculus that allows the use of predicates, logical statements which may include variables. First-order predicate calculus only allows simple variables. Second-order predicate calculus allows variables that may themselves be predicates. The language Prolog is, for the most part, a first-order predicate calculus system.
  - http://www.ai.usma.edu:8080/overview/GLOSSARY.HTM

Predicate Calculus (the 50-minute version)

- We have “Terms”, which can be either:
  - Constants (like Prolog atoms).
  - Variable symbols (like in Prolog).
  - Compound terms (function symbol + arguments), e.g.:
    - wife(henry)
    - distance(point1,X)
- “Atomic Propositions” express relationships between objects.
  - Predicate symbols + arguments, e.g.:
    - human(mary)
    - likes(Man,wine(chianti))

Differences between PC and Prolog

- In Prolog, a structure can be:
  - a goal
  - an argument in another structure
  - both
- In PC:
  - Function symbols are functors used to construct arguments.
  - Predicate symbols are functors used to construct propositions.

Making Compound Propositions from Atomic

- Logical connectives:
  - α, β are any proposition.

Connective	PC Syntax	Book Syntax	Meaning
Negation	$\neg\alpha$	$\sim\alpha$	"not α"
Conjunction	$\alpha\wedge\beta$	$\alpha\&\beta$	"α and β"
Disjunction	$\alpha\vee\beta$	$\alpha\#\beta$	"α or β"
Implication	$\alpha\supset\beta$	α->β	"α implies β"
Equivalence	$\alpha\equiv\beta$	α<->β	"α is equivalent to β"

Implication: What does "α->β" (pronounced "α implies β") mean?

- Whenever α is true, β is true.
- This is equivalent to (~α) # β
- Does not imply causality.
- Truth table:

a	b	a->b
false	false	true
false	true	true
true	false	false
true	true	true

Equivalence: What does "$\alpha \equiv \beta$" (pronounced "α is equivalent to β") mean?

- Whenever α is true, β is true, and vice versa.
- This is equivalent to $(\alpha \rightarrow \beta)$ & $(\beta \rightarrow \alpha)$
- Also equivalent to $(\alpha \& \beta)$ # $(\sim\alpha \& \sim\beta)$
- Truth table:

a	b	a->b
false	false	true
false	true	false
true	false	false
true	true	true

Quantifiers

- Meaning of these propositions is only defined when variables are introduced by quantifiers.
- v is a variable, P is a proposition.

Quantifier	PC Syntax	Book Syntax	Meaning
Universal	∀v.P	all(v,P)	“P is true whatever v stands for”
Existential	∃v.P	exists(v,P)	“There is something v can stand for, such that P is true”

Universal Quantifier:
What does “∀v.P” or “all(v,P)” mean?

- For all v, P is true.
- Example:
  - all(X, man(X) -> human(X))
  - “For all X, if X is a man, X is human.”
  - “All men are human.”

Existential Quantifier:
What does “∃v.P” or “exists(v,P)” mean?

- There is a v, such that P is true.
- Example:
  - exists(Z, father(john,Z) & female(Z))
  - “There exists a Z, such that john is the father of Z and Z is female.”
  - “John has a daughter.”

Quantifiers in Prolog I

- In Prolog rules:
  - For any variable in the head of the rule, a universal quantifier.
  - For any variable in the body of the rule that doesn't appear in the head, existential quantifier.

Quantifiers in Prolog II

- Example:
  - mom(X) :- mother(X, Y).
  - In English: X is a mom if X is the mother of Y
  - With quantifiers:

    for all X, X is a mom is there exists a Y so that X is the mother of Y.
  - Symbolically:

$\forall X$ mom(X) :- $\exists Y$ mother(X, Y).

Now What?

- There are ways to manipulate the propositions into standard forms, and do things with them.
- Beyond the scope of this course, but we can look at some things.

Resolution

- Suppose we take certain things to be true, what follows logically from them?
- J. Alan Robinson discovered the Resolution Principle, which can determine this mechanically.
- Fit two clauses together, then drop out the same atomic formula from both sides.

Resolution Example

- In this notation, “;” means “or”.
- Clauses:
  - sad(chris);angry(chris) :- workday(today), raining(today).
  - unpleasant(chris) :- angry(chris),tired(chris).
- Merge to get:
  - sad(chris);angry(chris);unpleasant(chris) :- workday(today),raining(today), angry(chris),tired(chris).
- Simplify to:
  - sad(chris);unpleasant(chris) :- workday(today),raining(today),tired(chris).

Problems With Resolution

- Resolution may generate many clauses.
  - How do you get them all?
  - How do you know which ones you want?
- Resolution does not guarantee that you can prove some clause is true.
- Resolution can show if a set of clauses is consistent or inconsistent.
  - Inconsistent if they resolve to the empty clause ":- ."

What Good Does This Do?

- You can assert something is true by putting it on the left side of an “:-”.
  - loyal(rover) :- .
- You can assert something is false by putting it on the right side of an “:-”.
  - :- loyal(rover).
- So...

Proving Through Resolution

- Start with a set of clauses which are consistent.
- Add the negation of the clause you are trying to prove.
- See if they resolve to the empty clause, meaning they are inconsistent.
- If so, the clause is true.

Next Time

- More Prolog & FOPC!