Outline

- Midterm
- Homework #5
- Pattern matching
- Unification
- Logic programming
Midterm

- Midterm grades
  - UG = 50% of points
  - G = 35% of points
- Midterm solutions
Homework #5

- Solutions
Pattern matching

- **Slade's version**
  - \((A \ast\text{WILD}\ast B)\) matches \((A B C B)\)
  - return \((B C)\)

- **More general**
  - \((A \ ?x\ B \ ?y\ B)\) matches \((A B B C B)\)
    - \(?X\) => \(B\)
    - \(?Y\) => \(C\)
  - \((A \ ?x\ B \ ?x\ B)\) matches \((A B B B B)\)
Bindings

- Pattern matcher maintains a binding list
  - `((?X . B) (?Y . C))`

- Once bound
  - a pattern variable never changes in value

- The rest of the target must be consistent with the bindings

- Matching against a variable
  - extends bindings, or
  - confirms consistency, or
  - fails
Norvig code

- Must load "auxfns.lsp" first
  - Problem with trial version?
  - try defpackage / in-package

- Pattern matcher
  - variables for single values not segments
  - different syntax for extended patterns
  - patmatch.lsp
Unification

- Variables on both sides
  - (foo ?a x) matches (foo x ?a)
- There is a binding that unifies the expressions
  - ?a = x
Unify function

- Tree-based
  - matches both car and cdr of pattern
- Attempts to unify variables
  - regardless of where they appear
Issues

- Must allow variables bound to variables
  - ?X -> ?Y
- Must not have circular evaluation
  - ?X -> ?Y
  - ?Y -> ?X
- Must not have recursive bindings
  - ?X -> (F ?X)
  - "occurs in" check
Examples
Prolog

- Assertions
  - statements
- Rules
  - implications
- Queries
  - statements of unknown truth
Example

- (<- (mortal ?x) (human ?x))
- (<- (human socrates))
- (?- (mortal socrates))
- or
- (? (mortal ?y))
Brief detour: macros

- What are
  - `<-
  - `?-`
- Obviously not a function
  - arguments are not evaluated
Macro

(defmacro <- (&rest clause)
  "add a clause to the data base."
  `(add-clause ',(replace-?-vars clause)))

- Arguments are not evaluated
- Can be used to build code to be evaluated
Macro expansion

- A macro is an arbitrary piece of Lisp code
  - result is an expression to be evaluated
- Make your own syntax
- Examples
  - expand <-
  - expand OR
- More about macros next week
Logic Review

- **Predicates**
  - has-professor(csc358, burke)

- **Variables**
  - if has-professor(course, professor) and subject-of (course, topic) then knows(professor, topic)

- **Quantification**
  - for all $x > 0$, there exists $s$ such that $s \times s = x$

- **Implication**

- **Equivalence**
  - daughter $(A, B)$ iff mother $(B, A)$ or father $(B, A)$ and female $(A)$
Terminology

- Left-hand side
  - IF part
- Right-hand side
  - THEN part
- Clause
  - a statement headed by a predicate
- Assertion
  - a true statement
Prolog

- Handles only implication
  - no equivalence
- Handles only a single implicated clause
  - No A => B and C
- Handles only AND clauses
  - no OR
  - no NOT
- Handles only universally-quantified variables
  - no "there exists"
- Called "Horn clauses"
Backward chaining

- If we want to prove something is true
  - we want to prove that is follows from what we already know
- Is socrates mortal?
  - follows from rules and assertions
- We follow the rules backwards
Algorithm

- Start with item to prove
  - if matching assertion exists, bind variables and return
  - if not, select rules that could prove it
  - try to prove each of these
Proof Tree

- Root $S = \text{statement to prove}$
- $S$ is the consequent of some rule
- The rule may have several precedent parts
  - each of these can be considered something to prove
Search problem

- Horn clause theorem proving is effectively search
  - we are searching the tree of rules and assertions
  - for something that will prove what we want to know
- Prolog does this depth-first
  - order of rule definition
- Interestingly
  - Horn clause theorem proving is Turing-complete
Backtracking

- Involves
  - making choices
  - testing them
  - when they fail, re-doing the choice differently

- Must keep track of
  - what choices have been tried
  - what to do differently
  - where the last choice was
In Prolog

- To prove Q
  - we may have several rules with Q is a consequent
  - if one rule fails to resolve, we try the next

- Recursive formulation
  - since resolve a rule requires more clauses to satisfy
  - etc.
Complexity

- Large rule base = more stuff to search
- Crucial "fan out" factor
  - how many ways to prove something
Many basic Lisp functions can be defined
- member
- length

Relational rather than functional
Knowledge Representation

- Prolog requires that we be declarative
- We have to have a language for representing our problem
  - Prolog doesn't help you write a good language
  - Good language = difference between computable and not
- We have to have rules that will answer our queries
  - Easy to write bad rules
How to build a language

- Select primitives
  - the most basic facts to be represented
  - from which other useful things can be inferred
- Select more complex predicates
  - more like the queries you want to answer
- Write rules that relate complex to simple
Example

I married a widow who has a grown-up daughter. My father, who visited us often, fell in love with my step-daughter and married her. Hence my father became my son-in-law and my step-daughter became my mother. Some months later, my wife gave birth to a son, who became the brother-in-law of my father, as well as my uncle. The wife of father, that is, my step-daughter also had a son. I'm my own grandpa.
Frustrations

- **Numbers**
  - pure Prolog doesn't have math operations
  - no relational operators like >.

- **Control**
  - pure Prolog only has one programmer-defined control over evaluation: order of definition
  - badly written rules can create infinite loops

- **Evaluation**
  - pure Prolog has no notion of value beyond variable binding
Hard to derive primitives (abduction)

- Knowing that Beatrice is someone's aunt
  - can't infer that she is female
  - even though this information is in the rules, sort of
Only backward chaining

- Sometime it is desirable to make inference in both directions

- Example
  - Mammals
Real Prolog

- Is a modern programming language
- Has mechanisms for math operations
- Has data structures
- Has some control structures
  - Cut (max example)
Homework #5

1. Create assertions and rules for a set of facts