Logic Programming *week 2*

Facts and rules infer new facts by asking questions. The system searches the fact database to determine a question's answer by logical deduction, e.g. from "x > y" and "y > z", we can infer "x > z". Inference procedures tell which statements are valid, inferred from others.

Syntax is how something is written, while semantics is what something means.

The syntax of propositional logic consists of propositional symbols (e.g. A, B, C, D), and connectives (∧ (and), ∨ (or), → (then), ¬ (not), ≡ (is the same as)). Its semantics is the assignment of a truth value (true or false) to each statement. For example:

A = "it is night", B = "it is dark", C = "the light is on",

(A ∧ B) → C = "if it is night and it is dark, then the light is on",

B → A = "if it is dark, then it is night",

A proof is a sequence of statements (each being either a premise or derived from an earlier statement by on of the inference rules). A goal is the statement we are trying to prove. For example:

Premises*: if it's night, the room is dark. If the room is dark, the light turns on. The light is not on.*

Goal*: is it night?*

A = it is night, B = the room is dark, C = the light is on.

Given: A→B, B→C, ¬C,

The following truth table shows that A = false, therefore it is not night.

Variables			**Given**			**Trial conclusions**	
A	**B**	**C**	**A→B**	**B→C**	**¬C**	**A**	**¬A**
T	T	T	T	T	F	T	F
T	T	F	T	F	T	T	F
T	F	T	F	T	F	T	F
T	F	F	F	T	T	T	F
F	T	T	T	T	F	F	T
F	T	F	T	F	T	F	T
F	F	T	T	T	F	F	T
F	**F**	**F**	**T**	**T**	**T**	**F**	**T**

The problem with truth tables is that the number of rows grows exponentially as the number of propositional values increases (2^n).

A proof procedure is a method of proving statements using inference rules:

1. modus ponens: if A→B and A are true, infer B is true,

2. modus tollens: if A→B and ¬A are true, infer ¬B is true,

3. elimination: if both (A ∧ B) is true, then infer both A and B are true,

4. introduction: if both A and B are true, then infer (A ∧ B),

5. disjunctive syllogism: if (A ∨ B) and ¬A are true, then infer B is true,

For example:

Premises*: if it's night, the room is dark. If the room is dark, the light turns on. the light is not on.* Goal*: is it night?*

A = it is night, B = the room is dark, C = the light is on,

given: A→B, B→C, ¬C,

B→C *and* ¬C, *therefore* ¬B (*modus tollens*)

now, A→B and ¬B, *therefore* ¬A (*modus tollens*)

A is false, it is not night.

The limitation of propositional logic is that it cannot represent, e.g. "x > y" and "y > z" therefore "x > z". To resolve this, extend representation (add predicates), extend operators, and add unification.

In predicate calculus, symbols can consist of any letter, any digit, and underscores, representing constants, functions, predicates (all begin with lowercase), or variables (begin with uppercase). Constants, functions, and variables are known as terms.

Constants name specific objects or properties, variables assign general classes of objects or properties, and functions combine variables and classes, e.g. man(bob), woman(alice). Replacing a function with its value is called evaluation, e.g. plus(2,3) whose value is 5.

Predicates name relationships between objects, e.g. likes(bob, alice). They are special functions with true/false values. The same predicate name with different values is considered distinct.

e.g. ***constants***, ***variables***, ***functions***, ***predicates***
likes(bob, alice), ***likes(X, Y)***, ***likes(father_of(bob))***

Terms are mapped to objects in their domain. Predicate calculus' semantics determine an expression's truth value. The inference system must be able to determine when two expressions match. Two expressions match if they are syntactically identical.

Unification determines the substitution list to make two predicate expressions match. If p and q are logical expressions, unify(p, q) gives a substitution list that either makes p and q identical or fails. The notation X/Y indicates that X can be substituted for Y in the original expression. A substitution is assigned to values and variables. Two terms unify if there is a substitution that makes the two terms identical, e.g. unifying f(X, 2) and f(3, Y) can be written as 3/X, 2/Y.

The process is complicated by the existence of variables, which can be replaced by terms (other variables, constants, or function expressions), e.g.

Unify	**Substitutions**	**Outcome**
p(a, X) and p(a, b)	b/X	p(a, b)
p(a, X) and p(Y, b)	a/Y, b/X	p(a, b)
p(a, X) and p(Y, f(Y))	a/Y, f(a)/X	p(a, f(a))
p(a, X) and p(X, b)	a/X, b/X *(X can't be both)*	Fail
p(a, b) and p(X, X)	a/X, b/X	Fail

Logic Programming *week 3*

Prolog itself won't be in the exam, but the logic behind it (see above) will be.

Prolog proves goals by matching them with rules/facts. It tries to find variable bindings making expressions identical.

Backtracking is the process of tracing steps backwards to previous goals and re-satisfying them when a goal fails. Prolog goes through facts/rules top to bottom to try to find matches, keeping track of where it has got to.