Notes for Chapter 12
Logic Programming

- The AI War
- Basic Concepts of Logic Programming
- Prolog
- Review questions
The AI War

• How machines should learn: inductive or deductive?
• **Deductive:** Expert => rules => knowledge, top-down approach, expert systems used LISP, Prolog, and shell languages CLIPS and JESS; programs suffered from: brittle and expensive to maintain.
• **Inductive:** knowledge <= rules <= Data, bottom-up, machine learning and data mining – extracts patterns from data and learns from examples, such as Decision Tree, Artificial NN, Genetic Algorithm; starting from 1980’s.
Logic Programming: Motivation

- **Logic** is used to represent program
- **Deductions** are used as computation
- A higher level language does more automatically – we can concentrate more on what is to be done and less on how to do it
- **Ideal**: Algorithm = logic (what) + Control (how) – only specify logic and let system take care of control
Logic Programming: Theoretical foundation

- *predicate calculus, Horn Clauses* – knowledge representations
- *Refutation system, unification, instantiation* – auto deduction methods
- *resolution principle* – inference engine behind Prolog
Differences between Procedural P. and Logic P.

- **Architecture:** Von Neumann machine (sequential steps)
- **Syntax:** Sequence of statements (a, s, I)
- **Computation:** Sequential statements execution
- **Control:** Logic and control are mixed together

- Abstract model (dealing with objects and their relationships)
- Logic formulas (Horn Clauses)
- Deduction of the clauses
- Logic and control can be separated
Basic Concepts

- A clause is a formula consisting of a disjunction of literals.
- Any formula can be converted into a set of clauses, for example:
  - $P \rightarrow Q \equiv \neg P \lor Q$
- Empty clause denoted by $[]$, always false.
Resolution

• An important rule of inference that can applied to
  – clauses (consisting of disjunction of literals)
  – a *refutation system*: prove by contradiction

• Idea: given two clauses, we can infer a new clause by taking the disjunction of the two clause & eliminating the complementary pair of literals
Resolution as A refutation system

Given a set of clauses $S \&$ and goal $G$,

* negate the goal $G$
* $\{S\} \cup \{\neg G\}$
* existence of contradiction $\Rightarrow$ derivation of empty clause

Based on $\{S\} \cup \{\neg G\}$ is inconsistent if $\{S\} \cup \{G\}$ is consistent
Resolution in a nutshell

- Represent knowledge and questions in terms of *Horn Clause* form of predicate logic
- Inconsistence checking: *refutation*
- The heart of the rule is the *unification* algorithm (the process of finding substitutions for variables to make arguments match – finding answers to questions)
Programming in Prolog

• Asserting some *facts* about objects and their relationships
• Representing general knowledge in terms of *rules*
• Asking *questions* about objects and their relations.
Forward/backward chaining

- A group of multiple inferences that connect a problem with its solution is called a **chain**.
- **Forward chaining**: inference starts from facts/rules.
- **Backward chaining**: inference starts from given problems.
Backtracking technique

• Inference backtracks to a previous step when a failure occurs.
• Naïve backtracking: backtracks mechanically to the most recent step when a failure occurs.
• Intelligent backtracking: analyze the cause of a failure & backtracks to the source of values causing the failure.
Prolog: sequence control

- Given a query, Prolog uses *unification* with *backtracking*.
- All rules have local context
- A query such as: $q_1, q_2, \ldots, q_n$
- *Unification implementation*: first evaluates $q_1$, then $q_2$, and so on (from *left to right*); database search (*top down*)
Deficiencies of Prolog

• Resolution order control
  • Ordering of pattern matching during resolution
  • Cut operator

• Closed world assumption
  • It has only the knowledge of its database
  • A true/fail system rather than a true/false

• The negation Problem
  • Prolog not operator is not equivalent to logical NOT operator
More on the negation problem

• The fundamental reason why logical NOT cannot be an integral part of Prolog is the form of the Horn clause.

• If all the B propositions are true => A is true. But it cannot be concluded that is false otherwise.
Negation as failure

- Example of page 565
  - parent(amy, bob).
  - ?- not(mother(amy, bob)).
  - The answer is yes, since the system does not know that amy is female and the female parents are mothers.
  - If we are to add these facts to our program, not(mother(amy, bob)) would no longer be true.
Concept Questions (1)

• What is backward chaining inference method?
• What is forward chaining inference method?
• Which inference method does each of the following languages use: Prolog, Clips?
Concept questions (2)

• What are the motivations for logic programming?
• What are the differences between procedural programming and logic programming?
• Execution of a Prolog program: knowledge representation and computation
Concept questions

(3)

• What is deductive analysis? Illustrate with an example.
• What is inductive analysis? Illustrate with an example.
• What is an expert system/rule based system? How does it work?
Concept Questions
(4)

- Use set notation to describe resolution as a refutation system.
- Construction of deduction tree of resolution.
- Programming in Prolog:
  - asserting facts,
  - representing knowledge in rules,
  - asking questions about objects and relations