CT331 Programming Paradigms Week 9 – Lecture 1

Dr. Finlay Smith
Room 430, IT Building
Finlay.smith@UniversityofGalway.ie

Introduction to Logic Programming

Logic Programming

- Logic programming is a programming paradigm based on formal logic.
  - A logic program consists of a series of assertions written in the language of formal logic.
- A program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain.
  - Results are derived from logic programs by symbolic reasoning.
  - Logic programming systems solve goals by systematically searching for a way to derive the answer from the program.

Logic Programming

- Uses **logic** to express knowledge.
- Uses **inference** to manipulate knowledge and resolve problems.
- In general, most logic programming is usually based on:

Horn-clause logic + negation-as-failure + backward chaining

- Programmer is responsible for specifying the basic logical relationships and does not specify the manner in which the inference rules are applied.

Horn-clause Logic

- In logic, a **clause** is an expression formed from a finite collection of **literals** (variables or their negations).
- A clause is called a Horn clause if it contains at most one positive literal.
  - A **definite clause** is a Horn clause that has exactly one positive literal.
  - A **negative clause** is a Horn clause that has no positive literals
- A Horn clause without a positive literal is called a **goal**.

- Horn clauses express a subset of statements of first-order logic. Programming language Prolog is built on top of Horn clauses. Prolog programs are comprised of definite clauses and any question in Prolog is a goal.

Negation-As-Failure

- A feature of Prolog and other logic programming languages in which failure of unification is treated as establishing the negation of a relation.
  - For example: if Bono is not in our database and we asked if he is Irish, Prolog would answer "no".
- Negation as failure is based on the *closed world assumption*
- Since we assume that anything that cannot be deduced from the facts we already have is wrong, if we *fail* to prove something it means that the opposite of it is considered true.

Backward Chaining

- An algorithm that works backwards from the goal, chaining through rules to find known facts that support the proof
- Backward chaining follows the classical depth-first search algorithm.
- Can have problems with repeated states and incompleteness.

Forward Chaining

- Present to future
- Data Driven
- Bottom-up reasoning
- Work forward to find solution follow from the facts
- Breadth First Search

Backward Chaining

- Present to past
- Goal driven
- Top-down reasoning
- Work backward to find facts that support the hypothesis
- Depth First Search

Logic Programming

- Declarative style programming paradigm.
- Computation through logical deduction.
- Uses the language of logic to express data and programs.
- Most of current logic programming languages use first order logic (FOL).
- Prolog – the most popular logic programming language

Syntax and Semantics

- Has *syntax* and *semantics*.
- Also has inference rules.

- **Syntax**: the rules about how to form formulas; this is usually the easy part of a logic.
- **Semantics**: about the meaning carried by the formulas, mainly in terms of logical consequences.
- **Inference rules:** describe correct ways to derive conclusions.

Different Perspectives of Logical Programming

- Computation as Deduction
- Theorem Proving
- Non-procedural Programming
- Algorithms Minus Control

Computation as Deduction

- Logic programming offers a slightly different paradigm for computation: *computation as logical deduction*
- It uses the language of logic to express data and programs.

For all X and Y, X is the father of Y if

X is a parent of Y and the gender of X is male.

Theorem Proving

- Logic Programming uses the notion of an *automatic theorem prover* as an interpreter.
- The theorem prover derives a desired solution from an initial set of axioms.
  - Axiom: a statement or proposition which is regarded as being established, accepted, or self-evidently true.
- Note that the proof must be a "constructive" one so that **more than** a true/false answer can be obtained.

Non-procedural Programming

- Logic Programming languages are non-procedural programming languages.
- A non-procedural language is one in which one specifies **what** needs to be computed but not **how** it is to be done.
- That is, one specifies:
  - the set of objects involved in the computation
  - the relationships which hold between them
  - the constraints which must hold for the problem to be solved

- The language interpreter or compiler decides **how** to satisfy the constraints.

Algorithms = Logic + Control

- Nikolas Wirth (architect of Pascal) used the following slogan as the title of a book:
  - Algorithms + Data Structures = Programs
- Robert Kowalski offers a similar one to express the central theme of logic programming:
  - Algorithms = Logic + Control
- We can view the **LOGIC** component as:
  - A specification of the essential logical constraints of a particular problem
- We can view the **CONTROL** component as:
  - Advice to an evaluation machine (e.g. an interpreter or compiler) on how to go about satisfying the constraints

Applications of Logic Programming

- Relational Databases
- Natural Language Interfaces
- Expert Systems
- Symbolic Equation solving
- Planning
- Prototyping
- Simulation
- Programming Language Implementation

Prolog: **Pro**grammation en **log**ique

- **The first and most popular logic programming language**
  - Invented by Alain Colmerauer and Phillipe Roussel at the University of Aix-Marseille, France in the early 70s.
- Prolog has its roots in first-order logic and is a declarative programming language
- The program logic is expressed in terms of *relations*, represented as *facts* and *rules*. A computation is initiated by running a query over these relations

Characteristics of Prolog

- Weakly typed
- No data abstraction
- No functional abstraction
- Has no mutable state
- Has no explicit control flow

How do you program in Prolog?

- Load facts/rules into interpreter
- Make queries to see if a fact is:
  - In the knowledge-base
  - Can be implied from existing facts or rules

- **Prolog is really an engine to *prove theorems***