IKI30820 – Logic Programming
Logic Programming Paradigm
Slide 01

Adila A. Krisnadhi Ari Saptawijaya

Fakultas Ilmu Komputer
Universitas Indonesia

2009/2010 • Semester Gasal

Programming Paradigm

- Programming paradigm:
  - fundamental style of computer programming (enforced by the language);
  - pattern that serves as a school of thoughts for programming of computers
- Major programming paradigms:
  - Imperative:
    - ★ Procedural/imperative paradigm
    - ★ Object-oriented paradigm
  - Declarative:
    - ★ Functional paradigm
    - ★ Logic paradigm

Imperative/procedural paradigm

- Computation as a sequence of action
  - “first do this and next do that”
- Stress more on **how** the computation takes place than on **what** is being computed
  - When solving a problem, put concern to method than meaning of the problem
- Example: Pascal, C/C++, Basic, Fortran, Cobol, etc.

Object-oriented paradigm

- Data as well as operations are encapsulated in objects
- Objects interact by means of message passing
- Objects are grouped in classes
  - Allows programming of the classes (as opposed to programming of individual object)
- Examples: C++, Java,

Functional Paradigm

- Originates from purely mathematical discipline: the theory of functions
  - Computation is based on functions
  - Functions have the same status as others (numbers, lists, ...), i.e., functions are first-class values
- Example: LISP, Haskell, Erlang, ...

Logic programming paradigm

- based on mathematical logic
- uses logical facts to create a model that can prove consistency deduce further truths, answer questions about the model
  - basic unit: clauses.
- Example: Prolog

LP Paradigm – Example 1

- Setting up a database of flight connections
  - Is there a direct flight from A to B?
  - Can I fly from C to D?
  - What are possible destinations I can reach from E?
  - etc.
- To answer these questions:
  - design and write a program;
  - run program with queries related to questions.

LP Paradigm – Example 1

- List of direct flights

```
direct(jakarta,yogyakarta).
direct(jakarta,surabaya).
direct(surabaya,makassar).
direct(makassar,manado).
```

- Q1: Is there a direct flight from Jakarta to Surabaya?

```
?- direct(jakarta,surabaya).
Yes.
```

- Q2: What are possible destinations of direct flights from Jakarta?

```
?- direct(jakarta,X).
X = yogyakarta;
X = surabaya.
```

LP Example 1

To find connections between two cities:

- There is a connection from X to Y, if there is a direct flight from X to Y

```
connection(X,Y) :- direct(X,Y).
```

- There is a connection from X to Y, if there is a direct flight from X to Z and a connection from Z to Y.

```
connection(X,Y) :- direct(X,Z), connection(Z,Y).
```

LP Paradigm – Example 1

- Q1: Is there a flight from Jakarta to Makassar?

```
?- connection(jakarta,makassar).
Yes.
```

- Q2: Where can one fly from Surabaya?

```
?- connection(surabaya,X).
X = makassar;
X = manado;
No.
```

LP Paradigm – Example 1

- Q3: Can someone fly from Manado?

```
?- connection(manado,X).
No.
```

- Q4: From where can one fly from Manado?

```
?- connection(X,manado).
X = makassar;
X = jakarta;
X = surabaya;
No.
```

LP Paradigm – Example 1

Two aspects of Prolog:

- Same program to compute answers to different problems (or queries)
- Program can be used much like a database
  - Knowledge is stored in the form of **facts** and **rules**, i.e., deductive database
  - Prolog models query processing in deductive database

LP Paradigm – Example 2

Finding all elements which are members of two given lists

- List: `[a1,a2,...,an]` or `[a1 | [a2,...,an] ]`
- `a1` is called **head** of `[a1,a2,...,an]`
- `[a2,...,an]` is called **tail** of `[a1,a2,...,an]`
- Ex: `[1,2,3,4,5] = [1 | [2,3,4,5]]`

LP Paradigm – Example 2

- X is a member of both L1 and L2 if X is a member of L1 and X is a member of L2.

```
member_both(X,L1,L2) :- member(X,L1),
                        member(X,L2).
```

- Now, when is X a member of a list L?

LP Paradigm – Example 2

- X is a member of both L1 and L2 if X is a member of L1 and X is a member of L2.

```
member_both(X,L1,L2) :- member(X,L1),
                        member(X,L2).
```

- Now, when is X a member of a list L?
  - if X is the head of L, then the answer is positive (a fact)

    ```
    member(X, [X | List]).
    ```

  - or otherwise, the answer is positive if X is a member of the tail of L.

    ```
    member(X,[Y | List]) :- member(X,List).
    ```

LP Paradigm – Example 2

- X is a member of both L1 and L2 if X is a member of L1 and X is a member of L2.

```
member_both(X,L1,L2) :- member(X,L1),
                        member(X,L2).
```

- Now, when is X a member of a list L?
  - if X is the head of L, then the answer is positive (a fact)

    ```
    member(X, [X | List]).
    ```

  - or otherwise, the answer is positive if X is a member of the tail of L.

    ```
    member(X,[Y | List]) :- member(X,List).
    ```

- Run the program to solve the problem:

```
?- member_both(X,[1,2,3],[2,3,4,5]).
```

- How do we solve this with imperative programming style (e.g., C)?

```
#define SIZE1 3
#define SIZE2 4

void memberBoth(int a[],int b[],int c[]) {
  int i, j, k=0;
  for (i=0;i<SIZE1;i++) {
    for (j=0;j<SIZE2;j++) {
      if (a[i] == b[j]) {
        c[k] = a[i]; k = k+1;
      }
    }
  }
}

int main() {
  int list1[SIZE1] = {1,2,3};
  int list2[SIZE2] = {2,3,4,5};
  int result[SIZE2];
  memberBoth(list1,list2,result);
  ...
}
```

LP Paradigm – Example 2

Other aspects of Prolog

- Searching mechanism need not be explicitly specified (it's implicitly given):
  - Generate all elements of the first list, which are then tested for membership in the second rule (cf. the rule).
- Prolog solutions can be used in a number of ways (cf. C solution)
  - Testing membership

```
?- member_both(2,[1,2,3],[2,3,4,5]).
```

  - Instantiating an element of a list

```
?- member_both(2,[1,2,3],[X,3,4,5]).
```

- Prolog constructs list dynamically
  - No size of list needs to be specified in advance (cf. use of array to represent list in a C program).

LP Paradigm - Example 3

- Strange squares
  - $45^2 = 2025$
  - Split 2025 into 20 and 25: $20 + 25 = 45$
  - Find other numbers with four digit squares that exhibit the same peculiarity.

LP Paradigm - Example 3

- Strange squares
  - $45^2 = 2025$
  - Split 2025 into 20 and 25: $20 + 25 = 45$
  - Find other numbers with four digit squares that exhibit the same peculiarity.

```
strange_square(N,Z) :-
    between(10,99,N),
    Z is N*N,
    Z >= 1000,
    Z//100 + Z mod 100 =:= N.
```

LP Paradigm – Example 3

Other aspects of Prolog:

- Stress on **what** is being computed rather than on **how** the computation takes place:
  - Declarative interpretation is enough to obtain the desired program
  - Procedural interpretation is not explicitly needed when designing the program (cf. example 2)
    - ★ Pick up every natural number between 10 and 99
    - ★ Test it, whether it is a solution of the problem
- The same program can be used for computing or testing or mixture between the two
  - Testing

```
?- strange_square(55,3025)
```

  - Computing (and testing as well)

```
?- strange_square(55,Z)
Z=3025
Yes
?- strange_square(44,Z)
No
```

LP Applications: Reasoning Agents

- Agent: perceive environment through sensors and act upon environment through actuators (Russel & Norvig, AI: A Modern Approach)
- Reasoning agents:
  - Capabilities are characteristisc of human-like intelligence:
    - ★ Mental representation of the world
    - ★ Correct reasoning with this representation
  - LP to encode (incomplete) world models, continuously update model upon the performance of an action, reason and draw logical conclusions based on world model
- References:
  - Fluent calculus (FLUX): `www.fluxagent.org`
  - Computational Logic Agents
  - Prospective Logic Programming

LP Applications: Semantic Web

- Current web content is for humans to read, *not* for computer programs to manipulate meaningfully.
- "For the semantic web to function, computers must have access to **structured collections** of information and sets of **inference rules** that they can use to conduct automated reasoning." (Tim Berners-Lee, 2001)
- LP is used to represent knowledge (in the form of **rules**) and reason on them (by making inferences with the rules)
- References:
  - The Semantic Web, Tim Berners-Lee, et.al., Scientific American – May 2001 (available online)
  - Semantic Web Logic Programming Tools, Alferes, et.al. Workshop on Principles and Practice of Semantic Web Reasoning, at 19th Int. Conf. on Logic Programming (ICLP03) (available online)
  - International Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services. `http://events.deri.at/alpsws2006`

LP Applications: Natural Language Processing (Computational Linguistics)

- LP is used to implement grammars
  - Analyze the syntax
  - Define the meaning (of a *fragment* of a natural language)
- Applications:
  - Natural language queries to databases
  - Natural language system specification
- References:
  - Attempto Controlled English (ACE): `http://www.ifi.unizh.ch/attempto/description/index.html`
  - Natural Language Processing Techniques in Prolog: `http://www.coli.uni-saarland.de/~kris/nlp-with-prolog/html`

LP Applications: Security protocol analysis

- Security protocols are designed to meet security properties
- Security protocols and security properties are specified using logic program
- Analysis of security protocols
  - Model attackers' capabilities using logic program
  - Outputs traces of attacks (if exist) for specified bounded number of sessions
- References:
  - CoProVe: `http://130.89.144.15/cgi-bin/psltl/show.cgi`
  - ProVerif: `http://www.di.ens.fr/~blanchet/crypto-eng.html`
  - NRL Protocol Analyzer: `http://chacs.nrl.navy.mil/projects/crypto.html`

Molecular biology

- Using inductive logic programming
  - Understand relationship between chemical structure and activity of drugs
  - Predicting mutagenesis to understand and predict carcinogenesis
  - Predicting protein secondary structures
- Using constraint-based methods
  - Workshops on constraint-based methods in bioinformatics 2005: `http://www.dimi.uniud.it/dovier/WCB05`
- References:
  - Inductive Logic Programming: `http://www.doc.ic.ac.uk/~shm/ilp.html`

A brief history

- Logic programming
  - Introduced by Robert Kowalski in 1974
  - Algorithm = Logic + Control
- **Prolog**
  - **Pro**gramming in **log**ic
  - Programming language that uses logic programming for computing
  - Introduced by Alain Colmerauer in 1970s

Prolog

- Prolog implementation used in this course is SWI Prolog
- Download free at `http://www.swi-prolog.org/`
- Developed by Jan Wielemaker, University of Amsterdam
- There are some other implementations (e.g. SICStus Prolog, XSB, etc.)