Logic Programming

Introduction

Michael Genesereth
Computer Science Department
Stanford University

Lecture will begin at ~3:05 PDT.
Logic Programming is a style of programming in which programs take the form of sentences in the language of Symbolic Logic.

Logic Program is a collection of such sentences.

Logic Programming Language is a language for writing such programs.

Logic Programming System is a computer system that manages the creation, modification, and/or execution of logic programs.
Imperative Programming
A triangle is a polygon with 3 sides.

\[ e = mc^2 \]
Runnable Specifications

A logic program is basically a **runnable specification**.

**Specification**
- What we believe about the *application area*
- What we want to know or to achieve in *application area*
- With no arbitrary decisions
- With no concern for internal processing details

**Runnable**
- Can be directly **interpreted**
- Can be **compiled** into traditional programs
  (Think automatic programming.)
Programming Ease

Easier to create and modify than traditional programs.

Programmers can get by with little or no knowledge of the capabilities of systems executing those programs.

There is no need to make arbitrary choices.

Easier to learn logic programming than traditional programming.

Oddly, expert computer programmers often have more trouble with logic programming than novices.
Composability
Agility
**Definition**

A person \(X\) is the grandparent of a person \(Z\) if and only if there is a person \(Y\) such that \(X\) is the parent of \(Y\) and \(Y\) is the parent of \(Z\).

**Uses**

- Determine whether art is the parent of Cal.
- Determine all of the grandchildren of Art.
- Compute the grandparents of Cal.
- Compute all grandparent-grandchildren pairs.
McCarthy’s Example of Versatility
Why Logic

Language
General / Domain-independent - no built-in assumptions
+ Highly expressive

Other data languages are easier for humans to use
Other definitional languages exist
But all can be converted to logical statements

Interpreter
Automated Reasoners capable of drawing conclusions
Can take advantage of domain-dependent reasoners
Logic

\[ \text{init} (\text{cell}(1,1,b)) \]
\[ \text{init} (\text{cell}(1,2,b)) \]
\[ \text{init} (\text{cell}(1,3,b)) \]
\[ \text{init} (\text{cell}(2,1,b)) \]
\[ \text{init} (\text{cell}(2,2,b)) \]
\[ \text{init} (\text{cell}(2,3,b)) \]
\[ \text{init} (\text{cell}(3,1,b)) \]
\[ \text{init} (\text{cell}(3,2,b)) \]
\[ \text{init} (\text{cell}(3,3,b)) \]
\[ \text{init} (\text{control}(x)) \]

\[ \text{next} (\text{cell}(M,N,P)) :- \]
\[ \text{does}(P,\text{mark}(M,N)) \]
\[ \text{next} (\text{cell}(M,N,Z)) :- \]
\[ \text{does}(P,\text{mark}(M,N)) \& \]
\[ \text{true}(\text{cell}(M,N,Z)) \& Z \# b \]

\[ \text{legal} (P, \text{mark}(X,Y)) :- \]
\[ \text{true}(\text{cell}(X,Y,b)) \& \]
\[ \text{true}(\text{control}(P)) \]
\[ \text{legal} (x, \text{noop}) :- \]
\[ \text{true}(\text{control}(x)) \]
\[ \text{legal} (o, \text{noop}) :- \]
\[ \text{true}(\text{control}(o)) \]

\[ \text{terminal} :- \text{line}(P) \]
\[ \text{terminal} :- \neg \text{open} \]

\[ \text{goal} (x, 100) :- \text{line}(x) \]
\[ \text{goal} (x, 50) :- \text{draw} \]
\[ \text{goal} (x, 0) :- \text{line}(o) \]
\[ \text{goal} (o, 100) :- \text{line}(o) \]
\[ \text{goal} (o, 50) :- \text{draw} \]
\[ \text{goal} (o, 0) :- \text{line}(x) \]

\[ \text{row}(M,P) :- \]
\[ \text{true}(\text{cell}(M,1,P)) \& \]
\[ \text{true}(\text{cell}(M,2,P)) \& \]
\[ \text{true}(\text{cell}(M,3,P)) \]

\[ \text{column}(N,P) :- \]
\[ \text{true}(\text{cell}(1,N,P)) \& \]
\[ \text{true}(\text{cell}(2,N,P)) \& \]
\[ \text{true}(\text{cell}(3,N,P)) \]

\[ \text{diagonal}(P) :- \]
\[ \text{true}(\text{cell}(1,1,P)) \& \]
\[ \text{true}(\text{cell}(2,2,P)) \& \]
\[ \text{true}(\text{cell}(3,3,P)) \]

\[ \text{line}(P) :- \]
\[ \text{true}(\text{cell}(M,1,P)) \& \]
\[ \text{true}(\text{cell}(M,2,P)) \& \]
\[ \text{true}(\text{cell}(M,3,P)) \]

\[ \text{draw} :- \neg \text{line}(x) \& \]
\[ \neg \text{line}(o) \]
Applications
Circuit:

Premises:

\[ o \leftrightarrow (x \land \neg y) \lor (\neg x \land y) \]
\[ a \leftrightarrow z \land o \]
\[ b \leftrightarrow x \land y \]
\[ s \leftrightarrow (o \land \neg z) \lor (\neg o \land z) \]
\[ c \leftrightarrow a \lor b \]

Applications:
Simulation
Configuration
Diagnosis
Test Generation
## Constraint Satisfaction

### Table: Owners and Breeds

<table>
<thead>
<tr>
<th>Years</th>
<th>Owners</th>
<th>Breeds</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Anita</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Molly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Riley</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Samson</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shadow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Sudoku Puzzle

```
6 1 4 5
8 3 5 6
2 1 4 6
8 4 7 6
6 3 1 4
7 9 1 4
5 3 2 6
7 2 6 9
4 5 8 7
```
Deductive Databases

\[
q(X) :\neg p(X,Y) \land p(X,Z) \land Y \neq Z
\]

\[
g(X,Z) : p(X,Y) \land p(Y,Z)
\]

\[
\text{illegal} : p(X,Y) \land p(Y,X)
\]

Questions
Updates

Database Manager

Answers
Notifications

\[
p(a,b)
p(b,c)
p(a,b)
\]
Data Integration

Integrated Search

Side-by-side Comparison

Integration Engine

Supplier 1
Supplier 2
Supplier 3
Supplier 4

Manufacturer 1
Manufacturer 2

Marketplace Data

Satisfaction Ratings

Product analysis
Business Rules

http://logicprogramming.stanford.edu/examples/programsheets/demonstration.html
Computational Law is that branch of legal informatics concerned with the mechanization of legal reasoning.

Automated Legal Reasoning Systems
- Legal analysis of specific cases
- Planning for compliance in specific cases
- Analysis of regulations for overlap, consistency, etc.

http://logicprogramming.stanford.edu/examples/portico/demonstration.html
General Game Playing

http://logicprogramming.stanford.edu/examples/nineboard/demonstration.html
Natural Language Processing
Theorem Proving

PTTP

means

Prolog Technology Theorem Prover

by acronymsandslang.com
History
Assembly Language

mov ecx, ebx
mov esp, edx
mov edx, r9d
mov rax, rdx

Assembly Language

Assembler + Linker

100101011001
010011111011
111010101101
010101010100

Processor

Machine Language
Higher Level Languages

Fortran
for Scientists and Engineers

David S. Touretzky
COMMON LISP
A Gentle Introduction to Symbolic Computation
The main advantage we expect the advice taker to have is that its behavior will be improvable merely by making statements to it, telling it about its … environment and what is wanted from it.

- John McCarthy 1958
McCarthy's paper belongs in the Journal of Half-Baked Ideas... The gap between McCarthy's general programme and its execution... seems to me so enormous that much more has to be done to persuade me that even the first step in bridging the gap has already been taken.

- Yehoshua Bar-Hillel 1958
The potential use of computers by people to accomplish tasks can be “one-dimensionalized” into a spectrum representing the nature of the instruction that must be given the computer to do its job. Call it the what-to-how spectrum. At one extreme of the spectrum, the user supplies his intelligence to instruct the machine with precision exactly how to do his job step-by-step. ... At the other end of the spectrum is the user with his real problem. ... He aspires to communicate what he wants done ... without having to lay out in detail all necessary subgoals for adequate performance.

- Ed Feigenbaum 1974
This course
Types of Logic Programming:

- Database Programming
- Classic Logic Programming
- Dynamic Logic Programming
- Constraint Systems
- Answer Set Programming
- Inductive Logic Programming (i.e. Progol)

Languages:

- Datalog
- Prolog
- Epilog
- LPS
- Progol
<table>
<thead>
<tr>
<th>April</th>
<th>May</th>
<th>June</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Introduction</td>
<td>Action Definitions</td>
<td>Advanced Topics</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Datasets</td>
<td>Lists, Sets, Trees</td>
<td>View Definitions</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>30</td>
</tr>
<tr>
<td>Queries</td>
<td>Simple Examples</td>
<td>Query Optimization</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>28</td>
</tr>
<tr>
<td>Updates</td>
<td>Dynamic Logic Programs</td>
<td>Advanced Topics</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>30</td>
</tr>
<tr>
<td>Query Evaluation</td>
<td>Database Management</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td>Query Optimization</td>
<td>Worksheets</td>
<td>Project Reports</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>9</td>
</tr>
<tr>
<td>View Definitions</td>
<td></td>
<td>Project Reports</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>11</td>
</tr>
<tr>
<td>Query Optimization</td>
<td></td>
<td>Project Reports</td>
</tr>
</tbody>
</table>
Sets

\[ \{a, b, c\} \cup \{b, c, d\} = \{a, b, c, d\} \]

\[ a \in \{a, b, c\} \]

\[ \{a, b, c\} \subseteq \{a, b, c, d\} \]

Functions and Relations

\[ f(a, b) = c \]

\[ r(a, b, c) \]
CS 106 or equivalent
Grades

Numerical Score
  10% for each of Assignments 1, 2, 3, 4, 5
  50% for the Term Project

Reported Grade
  Based on numerical score (see above)
  *No* curve - independent of number of students
  Satisfactory = 70% and above

Extra Credit
  Added to score before determining Reported Grade
  Discretionary
Introduction to Logic Programming

Michael Genesereth, Stanford University
Vinay K. Chaudhri, Stanford University

“This is a book for the 21st century: presenting an elegant and innovative perspective on logic programming. Unlike other texts, it takes datasets as a fundamental notion, thereby bridging the gap between programming languages and knowledge representation languages; and it treats updates on an equal footing with datasets, leading to a sound and practical treatment of action and change.” – Bob Kowalski, Professor Emeritus, Imperial College London

“In a world where Deep Learning and Python are the talk of the day, this book is a remarkable development. It introduces the reader to the fundamentals of traditional Logic Programming and makes clear the benefits of using the technology to create runnable specifications for complex systems.” – Son Cao Tran, Professor in Computer Science, New Mexico State University

“Excellent introduction to the fundamentals of Logic Programming. The book is well-written and well-structured. Concepts are explained clearly and the gradually increasing complexity of exercises makes it so that one can understand easy notions quickly before moving on to more difficult ideas.” – George Younger, student, Stanford University

ABOUT SYNTHESIS
This volume is a printed version of a work that appears in the Synthesis Digital Library of Engineering and Computer Science. Synthesis books provide concise, original presentations of important research and development topics, published quickly, in digital and print formats.
The following syllabus lists all of the materials of the course. Note that there are interactive exercises at the ends of the chapters in the course textbook. (Click on the exercise numbers to go to the exercise pages.) These exercises are an essential part of the course, and you will benefit from tackling them. Some are easier than others, but you should attempt them all. Do the exercises! Do The Exercises!! DO THE EXERCISES!!!

**Color Code**
- Black - Lecture Slides
- Blue - Readings
- Red - Assignments
- Grey - Comment

**Introduction (Week 1)**
- Lecture 1 - Introduction
- Lecture 2 - Datasets
- Chapter 1 - Introduction
- Chapter 2 - Datasets
- Programs With Common Sense
- Logic Programming
- Assignment 1.1 - Datasets in Sierra
- Assignment 1.2 - Game State
- Assignment 1.3 - Triples
- Project