Programming Languages

Logic Languages – Chapter 12

Objective

Logic lecture's primary objective:

- Introduce you to Horn clauses and resolution
- Compare imperative, functional and logic languages

Logic Programming

Began early 1970s from work in automatic theorem proving and AI

AI – constructing automated deduction systems

1988 – Robinson introduced resolution rule which is well-suited to automation on a computer

PhD's related to logic programming: https://www.cs.nmsu.edu/ALP/phd-theses/phdtheses/

Logic Languages – all somewhat based on Prolog

ALF
Alma-0
CLACL-Langauge
Curry
Fril
Janus
LambdaProlog
Leda
Oz
Prolog
    Mercury
    Strawberry Prolog
    Visual Prolog
ROOP

https://en.wikipedia.org/wiki/List_of_programming_languages_by_type#Logic-based_languages

Language Paradigms

	Imperative	Functional	Logic
Example languages	Fortran, Pascal, C, Java, scripting most that we use	Scheme, LISP, ML Haskel, Single Assignment C	Prolog, Mercury (very few)
Basis	Turing machines (Alan Turing)	Lambda calculus (Alonzo Church)	Mathematical logic (Aristotle)
Computers Principally using	Iteration and side effects	Substitution of parameters into functions	Resolution of logical statements, driven by the ability to unify variables and terms

Resolution

Discrete Structures

Last inference rule

TABLE of Rules of Inference

Rule of Inference	*Tautology*	*Name*
p $p \rightarrow q$ $\therefore q$	$[p \wedge (p \rightarrow q)] \rightarrow q$	Modus ponens
$\neg q$ $p \rightarrow q$ $\therefore \neg p$	$[\neg q \wedge (p \rightarrow q)] \rightarrow \neg p$	Modus tollens
$p \rightarrow q$ $q \rightarrow r$ $\therefore p \rightarrow r$	$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	$[(p \vee q) \wedge \neg p] \rightarrow q$	Disjunctive syllogism
p $\therefore p \vee q$	$p \rightarrow (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \rightarrow p$	Simplification
p q $\therefore p \wedge q$	$[(p) \wedge (q)] \rightarrow (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	$[(p \vee q) \wedge (\neg p \vee r)] \rightarrow (q \vee r)$	Resolution

Prolog Resolution

Horn clause:

$H \leftarrow B_1, B_2, \ldots, B_n$

H can be gotten from B_1, and B_2, and …, and B_n

Resolution

If $C \leftarrow A, B$

$D \leftarrow C$

Then $D \leftarrow A, B$

Example Prolog Database

```
/* Database of  3 facts and 1 rule */
 rainy(seattle).
 rainy(rochester).
 cold(rochester).
 snowy(X):-rainy(X), cold(X).
```

Queries:

```
 ?-rainy(seattle).
 ?-cold(seattle).
 ?-snowy(rochester).
 ?-snowy(seattle).
 ?-snowy(C).
```

Example Prolog Database

```
/* Database of  3 facts and 1 rule */
 rainy(seattle).
 rainy(rochester).
 cold(rochester).
 snowy(X):-rainy(X), cold(X).
```

What do you get from ther queries?

```
 ?-rainy(seattle).            => true.
 ?-cold(seattle).             => false.
 ?-snowy(rochester).          => true.
 ?-snowy(seattle).            => false.
 ?-snowy(C).                  => C = rochester.
```

Language Paradigms (more)

	Imperative	Functional	Logic
Char-acter-istics	Mirrors underlying hardware and can be "tweaked" for high performance	Avoids the semantic complexity of side effects. Particularly good for symbolic manipulation. Since referentially transparent, easier to reasoning about, good for parallel processing.	Well suited for problems that emphasize relationships and search. Can be considered "runnable specifications" Naturally parallel Used for formal specification, expert systems, theorem proving, and sophisticated control systems

Language Paradigms (more)

	Imperative	Functional	Logic
Power (aside from hardware imposed restrictions on arithmetic precision, disk and memory space)	Full power of Turing machines (Turing complete)	Full power of Lambda calculus (Turing-complete)	Less than full generality of resolution theorem proving (Turing complete)
Programming constructs added that weren't in the basic model	Nothing needed	I/O and precision Some languages add assignment	I/O, true arithmetic, imperative control flow, high-order predicates for self-inspection and modification

Logic Language Strengths

Logic languages are good for:

- Theorem proving
- Executing specifications when those specifications are written formally
- Expert systems
- Sophisticated control systems
- Problems that emphasize relationships and search
- Naturally parallel