Logic Programming: An Overview

Saumya Debray
Department of Computer Science
University of Arizona
Tucson, AZ 85721, USA

Outline

- declarative *vs.* imperative programming
- what is logic programming?
- logic programming: the landscape today
- implementation issues:
  - sequential
  - parallel
- connections with other areas
  - functional languages
  - deductive databases
  - constraint programming languages

Imperative Programming

Programming Model : The user specifies *how* a problem is to be solved, i.e., the sequence of actions to be carried out.

Execution Model : computation proceeds by updating the state, i.e., reading and writing to a shared store.

Advantages :

- control of machine-level entities
- efficiency

Disadvantages :

- less support for high-level abstractions
- program behavior defined only in terms of the sequence of machine level operations – affects understandability and portability.

Declarative Programming

Programming Model : The user specifies *what* constitutes a solution.

Execution Model : The system figures out *how* the computation should proceed.

There may not be a notion of state, i.e., a shared store.

Advantages :

- users think in terms of high-level abstractions: programs become easier to read, write, maintain.
- the system takes care of implementing the abstractions: programs become more portable.

Disadvantages :

- efficient implementation of high-level primitives may not be easy
- the lack of state may be awkward.

An Example of Declarative Programming

Symbolic Differentiation

```
% deriv(Y, X, Z) is true iff Z is the derivative
% of Y w.r.t. X, i.e., Z = dY/dX.

deriv(var(X), var(Y), 1) :- X == Y.
deriv(var(Y), var(X), 0) :- X \== Y.
deriv(num(N), _, 0).
deriv(U+V, X, DU+DV) :-
     deriv(U, X, DU), deriv(V, X, DV).
deriv(U-V, X, DU-DV) :-
     deriv(U, X, DU), deriv(V, X, DV).
deriv(U*V, X, V*DU + U*DV) :-
     deriv(U, X, DU), deriv(V, X, DV).
deriv(U/V, X, (DU*V - U*DV)/V^2) :-
     deriv(U, X, DU), deriv(V, X, DV).
```


What is Logic Programming?

"*Procedural Reading*" :

Generalizations of "traditional" languages in certain ways:

- Allow nondeterminism, i.e., define relations rather than functions:
  - A procedure can be defined by multiple bodies.
  - These alternative definitions can be executed in various ways, e.g. sequentially by backtracking, or in parallel.
  - Parameter passing is via a generalized pattern matching procedure.
- do not require that representable objects be "fully defined", i.e., allow partially defined values (data structures with "holes").

  This allows novel programming techniques. E.g.:
  - In a parallel system, a process may send a message with a "hole" for the reply;
  - The recipient can "fill in" this hole with the reply without ever having to know who the sender was.

What is Logic Programming? (cont'd)

"*Logical Reading*" :

- programs are statements in some logic, e.g.:

  "If $d/dx(u) = u_1$ and $d/dx(v) = v_1$ then $d/dx(u+v) = u_1 + v_1$."

  ```
  deriv(U+V, X, DU+DV) :-
       deriv(U, X, DU), deriv(V, X, DV).
  ```
- execution corresponds to making inferences using the proof rules of the logic.
- computation is viewed as (controlled) deduction.
- Gives rise to new terminology, e.g.:

  procedure $\Rightarrow$ "predicate"
  procedure definition $\Rightarrow$ "clause"
  execution $\Rightarrow$ "derivation"

Prolog : A Logic Programming Language

Data Objects (*terms*) :

- a variable or a constant is a term;
- if f is an n-argument "function symbol" and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

Examples : `X`, `27`, `person(Name, dob(Month, Date, Year))`.

Types :

- Dynamic types: no type declarations.
- Types usually supported:
  - numeric (integers, float), symbolic constants, terms.
  - terms give arrays, records, and lists.

Unification : a generalized pattern matching procedure for terms. E.g.:

unify(`f(X, g(X, Y)), f(a, Z)`) = `f(a, g(a, Y))`

with unifier $\mathtt{X} \mapsto \mathtt{a}, \mathtt{Z} \mapsto \mathtt{g(a, Y)}$.

Prolog : A Logic Programming Language (cont'd)

Program Objects :

literal : a procedure call.

If p is an n-argument procedure and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a literal.

clause : a procedure definition.

If $H, B_1, \ldots, B_n$ are literals then

$$H \;:-\; B_1, \ldots, B_n$$

is a clause. E.g.:

```
deriv(U+V, X, DU+DV) :-
     deriv(U, X, DU), deriv(V, X, DV).
```

The *head* is H, the *body* is a sequence of procedure calls $B_1, \ldots, B_n$.

predicate : a procedure, defined by a collection of clauses.

Prolog : A Logic Programming Language (cont'd)

Program Execution :

environment : interactive; the user enters top-level procedure calls to be executed.

parameter passing : by unification of the actuals with the formals.

executing a procedure :

- execute the clauses in their top-to-bottom order.
- save information where necessary to allow backtracking if unification fails.

executing a clause :

- if unification of actuals with formals fails, backtrack.
- otherwise, execute the body literals in their left-to-right order.

Prolog Programs: Logical Reading

clauses : a clause '$Head \;:-\; Body$' denotes the statement

if $Body$ **then** $Head$.

predicates : a predicate defined by the clauses

'$H_1 \;:-\; B_1$', ..., '$H_n \;:-\; B_n$'

denotes

(H_1 **if** B_1**) and . . . and (**H_n **if** B_n**)**.

goals : a goal $G_1, \ldots, G_n$ with variables $x_1, \ldots, x_k$ denotes the statement

(there exist $x_1, \ldots, x_k$**)[**G_1 **and . . . and** G_n**]**.

that must be proved.

Prolog Programs: Logical Semantics

Programming language semantics :

- the meaning of a program is a function from input values to output values;
- this function is typically defined as the limit of a series of successive approximations;
- the mathematics can be quite complicated:
  - need to show that the limit exists (domain, functions must have right properties);
  - need to model the effects of destructive updates, **goto**s, etc.

Logical Semantics of Prolog :

- The meaning of a program is the set of "facts" that follow logically from it.
- semantic function: *immediate consequence*:

  Given a set of facts known to be true, what other facts can be inferred from the program?

Logic Programming vs. Prolog

- "logic programming" $\not\equiv$ Prolog.
- We can independently vary both the "*logic*" and the "*programming*":
  - Varying the *logic* :
    - higher order LP: λ-Prolog; logic + functional programming;
    - logic + OOP: LOGIN, LIFE;
    - constraint logic programming: CLP($\mathcal{R}$), CHIP;
    - programming using linear logic, concurrent constraint programming.
  - Varying the *programming* :
    - bottom-up processing, deductive DBs;
    - committed choice languages : CP, GHC, Parlog, Janus, ...
    - various AND- and OR-parallel execution models;
    - Andorra-based languages.

Prolog: Implementation Issues

Data Management :

- efficient unification:
  - compiler generates specialized unification code where possible;
  - general unification routine invoked rarely.
- reset variables on backtracking:
  - maintain a log of variables that need to be reset (the *trail*);
  - no need to log variables assigned since the last backtrack point.

Prolog: Implementation Issues (cont'd)

Control Management : analogous to Pascal + backtracking:

- Indexing: use a hash table to find "candidate" clauses quickly;
- Efficient management of backtracking:
  - create a backtrack point only if there is some clause to try on backtracking;
  - save only enough information to restore state on backtracking:
    - actual parameters;
    - stack, heap tops;
    - address of code to execute.
- Efficient procedure calls and returns:
  - parameters passed in registers;
  - stack frames created by callee, only if necessary;
  - tail recursion optimization.

Prolog: Performance

The fastest Prolog implementations are now approaching (sometimes surpassing) the performance of optimized C:

Some performance figures from Aquarius Prolog [Van Roy, 1991]:

Program	Aquarius	MIPS C unopt	MIPS C opt
tak(24,16,8)	1.2s	2.1s	1.6s
fib(30)	1.5s	2.0s	1.6s
hanoi	1.3s	1.6s	1.5s
qsort	2.8s	3.3s	1.4s

Performance : cont'd

Performance of the Janus system (a "committed-choice" LP language implemented at University of Arizona) :

Program	J/gcc -O2	J/cc -O2	J/cc -O4
aquad	1.238	0.712	1.119
bessel	0.978	0.599	0.614
binomial	1.127	0.643	0.938
chebyshev	1.179	0.470	0.470
e	1.047	0.969	0.968
fib	0.997	1.025	1.028
log	0.984	0.491	0.491
mandelbrot	1.234	0.305	0.518
muldiv	1.193	1.087	1.089
nrev	1.635	1.636	1.877
pi	1.012	0.529	0.529
sum	1.055	1.055	4.172
tak	1.218	1.298	1.303
Geom. Mean :	1.134	0.752	0.940

Bottom-Up Processing

- Prolog's execution strategy:
  - top-down, or backward chaining;
  - one solution at a time, backtrack to find alternatives.
- Alternative execution strategy:
  - bottom-up, or forward chaining;
  - propagate sets of solutions; no backtracking.

Bottom-Up Processing and Deductive Databases

- Logic programs with bottom-up evaluation generalize relational DB query languages:
  - no need for a separate embedded query language;
  - relational operators such as *select*, *project*, *join*, *cartesian product* fall out naturally;
  - recursion allows much more powerful queries.
- Implementation challenges:
  - goal ordering;
  - finiteness inference;
  - query optimization in the presence of recursion.
- Implementations:
  - NAIL! (Stanford U.)
  - LDL, LDL++ (MCC, Austin);
  - Coral (U. Wisconsin-Madison)

Parallel Evaluation of Logic Programs

Three main flavors of parallelism:

And Parallelism : parallel execution of (independent) computations that are both needed for the final answer.

Example (divide & conquer) : solve subtasks of a task in parallel.

Or Parallelism : parallel exploration of multiple alternative execution paths.

Example (n-queens) : search alternative placements in parallel.

Stream Parallelism : parallel execution of producers and consumers.

Example (compiler) : execute the scanner and parser in parallel.

Parallel Evaluation of Logic Programs (cont'd)

- These notions of parallelism are high level, and not tied to machine level features (implies portability).
- The best form(s) of parallelism for a problem depend on the problem:
  - nondeterministic search problems (AI) : Or parallelism
  - deterministic computations : And parallelism
  - "phased" computations : Stream parallelism.
- The technical challenge is not to *discover* parallelism, but to *control* it:
  - benefits from parallelism vs. overheads from managing parallel tasks;
  - communication costs.

Connections with Functional Programming

- Functional programming languages:
  - *deterministic*, i.e., compute functions
  - require "fully defined" values
  - *higher order*, i.e., function objects can be passed around as values.
- Logic programming languages:
  - *nondeterministic*, i.e., compute relations
  - do not require "fully defined" values
  - *first order*, i.e., program and data objects are distinct.
- In some ways, logic programming generalizes functional programming.

  In other ways, functional languages generalize logic languages.
- There have been a variety of proposals to combine logic and functional languages.

Constraint Logic Programming

- In Prolog, a unification of the form

  $X = f(Y)$

  imposes a constraint on the values X may take on.
- The idea can be extended to domains other than terms. Tradeoff:
  - A rich constraint system is more expressive, but constraint solving incurs more runtime overhead.
  - A simple constraint system is more efficient, but may not be as expressive.
- A computation can now be thought of as a process of refining the possible values for its variable by imposing constraints.
- The language $CLP(\mathcal{R})$ is based on constraints over real numbers.

Current Research Areas

Theory :

- connections with non-monotonic, hypothetical reasoning (AI);
- connections with theory of concurrency;
- connections with object-oriented languages.

Practice :

- Prolog optimization;
- parallel systems;
- "higher order" systems : programs as data.