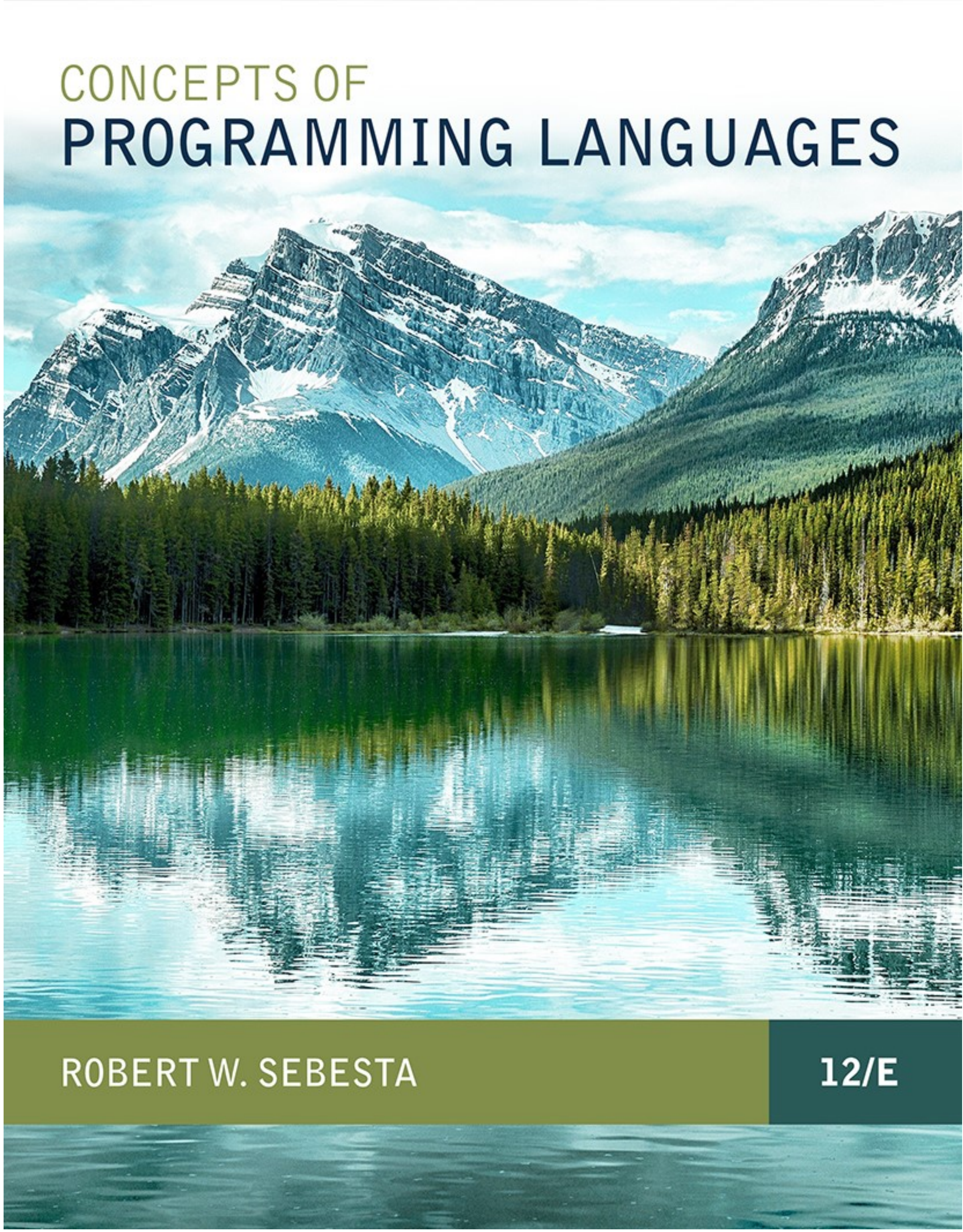
Chapter 16

Logic Programming Languages



ISBN 0-321-49362-1

Chapter 16 Topics

- Introduction
- A Brief Introduction to Predicate Calculus
- Predicate Calculus and Proving Theorems
- An Overview of Logic Programming
- The Origins of Prolog
- The Basic Elements of Prolog
- Deficiencies of Prolog
- Applications of Logic Programming

Copyright © 2018 Pearson. All rights reserved.

Introduction

Logic (declarative) Programming Language

- Express programs in a form of symbolic logic
- Use a logical inferencing process to produce results
- *Declarative* rather that *procedural*:
  - Only specification of *results* are stated
  - Not detailed *procedures* for producing them.

Copyright © 2018 Pearson. All rights reserved.

Proposition

- Proposition: a logical statement that may or may not be true:
  - e.g., man(jake)   likes(bob, steak)
  - Consists of objects and relationships of objects to each other
    - Operands (Truth or Falsity)
    - Operations (and, or)
    - Parentheses (aid determining order evaluation).
    - e.g.,
      - T and F
      - Identifier (a sequence of one or more digits and letters, with the first of which is a letter), e.g., abc
      - If "abc" is a proposition, then so is "¬ abc" .

Copyright © 2018 Pearson. All rights reserved.

Symbolic Logic

- Symbolic Logic used for the basic needs of formal logic:
  - Express propositions
  - Express relationships between *propositions*
  - Describe how new propositions can be inferred from *other* propositions
- Particular form of symbolic logic used for logic programming called (*first-order*) *predicate calculus*
  - use quantified variables over non-logical objects
  - allow the use of sentences that contain variables
  - E.g., *For all x, if x is a man then x is mortal*.

Copyright © 2018 Pearson. All rights reserved.

Object Representation

- Objects in propositions are represented by simple terms: either constants or quantified variables
  - *Constant*: a symbol that represents an object
    - e.g., likes(bob, steak); *likes*, *bob*, *steak* are **constants**
  - *Quantified Variable*: a symbol that can represent different objects at different times
    - e.g., $\forall$ X. P (For all X, P is true), where **X** is a quantified variable; $\forall$ is a **universal** quantifier
    - Different from variables in imperative languages
    - e.g., $\forall$x. P*x* -> Q*x* [P*x*: x is a man; Q*x*: x is mortal].

Copyright © 2018 Pearson. All rights reserved.

Compound Terms

- *Atomic propositions*
  - simplest proposition; e.g., likes(bob, trout)
  - consist of compound terms
  - Compound term composed of two parts
    - Functor: function symbol that names the relationship: e.g., student, like
    - Ordered list of parameters (tuple)

e.g.,

```
student(jon)
like(seth, OSX)
like(nick, windows)
like(jim, linux).
```

Copyright © 2018 Pearson. All rights reserved.

Forms of a Proposition

- Propositions can be stated in two forms:
  - *Fact*: proposition is assumed to be true
  - *Query*: truth of proposition is to be determined
- Compound proposition:
  - Have two or more atomic propositions
  - Propositions are connected by logical operators.

Copyright © 2018 Pearson. All rights reserved.

Logical Operators

Name	Symbol	Example	Meaning
negation	$\neg$	$\neg$ a	not a
conjunction	$\cap$	a $\cap$ b	a and b
disjunction	$\cup$	a $\cup$ b	a or b
equivalence	$\equiv$	a $\equiv$ b	a is equivalent to b
implication	$\supset$ $\subset$	a $\supset$ b a $\subset$ b	a implies b b implies a

Copyright © 2018 Pearson. All rights reserved.

Quantifiers

Name	Example	Meaning
universal	$\forall X.P$	For all X, P is true
existential	$\exists X.P$	There exists a value of X such that P is true

Copyright © 2018 Pearson. All rights reserved.

Clausal Form

- Too many ways to state the same thing
- *Clausal form*:
  - standard form for propositions
  - $B_1 \cup B_2 \cup \ldots \cup B_n \subset A_1 \cap A_2 \cap \ldots \cap A_m$
    - means if all the As are true, then at least one B is true
- *Antecedent*: right side (i.e., $A_1 \ldots A_m$)
- *Consequent*: left side (i.e., $B_1 \ldots B_n$).

Copyright © 2018 Pearson. All rights reserved.

Resolution

- *Resolution*:
  - process of inferring an proposition from given propositions
  - allows inferred propositions to be *computed* from given propositions.
  - e.g.,
    - older(joanne, jake) $\subset$ mother(joanne, jake)
    - wiser(joanne, jake) $\subset$ older(joanne, jake)
    - wiser(joanne, jake) $\subset$ mother(joanne, jake).

Copyright © 2018 Pearson. All rights reserved.

Unification & Instantiation

- *Unification*: finding values for variables in propositions that allows matching process to **succeed**
- *Instantiation*: assigning ***temporary*** values to variables to allow unification to succeed
  - After instantiating a variable with a value, if matching fails, may need to *backtrack* and instantiate with a different value.

Copyright © 2018 Pearson. All rights reserved.

Theorem Proving

- Basis for logic programming
- When propositions are used for resolution, only *restricted* form can be used
- *Horn clause* – can have only two forms
  - *Headed*: single atomic proposition on left side
    - e.g., likes(bob, trout) $\subset$ likes(bob, fish) $\cap$ fish(trout)
  - *Headless*: ***empty*** *left side* (used to state facts)
    - e.g., father(bob, jake)
- Most propositions can be *stated* as Horn clauses.

Copyright © 2018 Pearson. All rights reserved.

Overview of Logic Programming

- Declarative semantics
  - There is a simple way to determine the meaning of each statement
  - Simpler than the semantics of imperative languages
- Programming is nonprocedural
  - Programs do not state *how* a result is to be computed, but rather the **form** of the **result**.

Copyright © 2018 Pearson. All rights reserved.

Example: Sorting a List

- Describe the characteristics of a sorted list, not the process of rearranging a list

❖ sorted (list) $\subset$

$\forall_j$ such that $1 \leq j < n$, $list(j) \leq list(j+1)$

❖ sort(old_list, new_list) $\subset$

permute (old_list, new_list) $\cap$ sorted (new_list)

Notes:

1. permute is a predicate returned *true* if its 2^{nd} parameter—new_list, is a permutation of its 1^{st} parameter—old_list
2. sort the items in old_list and put them into new_list.

Copyright © 2018 Pearson. All rights reserved.

The Origins of Prolog

- University of Aix-Marseille, France (Alain Calmerauer & Phillippe Roussel)
  - Natural language processing
- University of Edinburgh, Scotland (Robert Kowalski)
  - Automated theorem proving.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Terms

- uses the *Edinburgh* syntax of Prolog
- *Term*: a constant, variable, or structure
- *Constant*: an atom or an integer
  - ***Atom***: symbolic value of Prolog, consists of either:
    - a string of letters, digits, and underscores beginning with a lowercase letter
    - a string of printable ASCII characters delimited by apostrophes.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Terms

- *Variable*: any string of letters, digits, and underscores beginning with an uppercase letter
  - *Instantiation*:
    - binding of a variable to a value
    - Lasts as long as it takes to satisfy one complete goal
- *Structure*: represents atomic proposition

  e.g., functor (*parameter list*) .

Copyright © 2018 Pearson. All rights reserved.

Prolog: Fact Statements

- Used for the hypotheses
- Headless Horn clauses

```
female(shelley).
male(bill).
father(bill, jake).
```

- Prolog statement is **terminated** by a period "."

Prolog: Rule Statements

- Used for the hypotheses
- Headed Horn clause
- Right side: *antecedent* (*if* part)
  - May be single term OR conjunction
- Left side: *consequent* (*then* part)
  - Must be **single** term

e.g., `ancestor(mary,shelley):- mother(mary,shelley).`

- *Conjunction*: multiple terms separated by logical AND operations (implied).

Copyright © 2018 Pearson. All rights reserved.

Prolog: Example Rules

- Can use variables (*universal objects*) to generalize meaning:

```
parent(X,Y):- mother(X,Y).
parent(X,Y):- father(X,Y).
grandparent(X,Z):- parent(X,Y), parent(Y,Z).

sibling(X,Y):- mother(M,X), mother(M,Y),
                    father(F,X), father(F,Y).
```

Copyright © 2018 Pearson. All rights reserved.

Prolog: Goal Statements

- For theorem proving, theorem is in form of proposition that we want system to prove or disprove.
- In Prolog, these propositions are called *goals (statements) or queries*; same format as headless Horn clause: e.g., `man(fred)`
- Conjunctive propositions and propositions with variables also legal goals

```
father(X, mike)
```

  - System will then attempt, through unification, to find an instantiation of X that results in a true value.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Inferencing Process

- If a goal is a compound proposition, each of the facts is a subgoal.
- To prove a goal is true, must find a chain of inference rules and/or facts. For **goal Q**:

  P_2 :- P_1

  P_3 :- P_2

  …

  Q :- P_n
- Process of proving a subgoal called matching, satisfying, or **resolution**.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Approaches

- *Bottom-up resolution, forward chaining*
  - Begin with **facts** and **rules** of database and attempt to find sequence that leads to goal
  - Works well with a large set of possibly correct answers
- *Top-down resolution, **backward** chaining*
  - Begin with **goal** and attempt to find sequence that leads to set of facts in database
  - Works well with a small set of possibly correct answers
- Prolog implementations use **backward** chaining.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Subgoal Strategies

- When goal has more than one subgoal, can use either
  - Depth-first search: find a complete proof for the first subgoal before working on others
  - Breadth-first search: work on all subgoals in parallel
- Prolog uses depth-first search
  - Can be done with fewer computer resources.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Backtracking

- *Backtracking:*
  - with a goal with multiple subgoals, if fail to show truth of one of subgoals, reconsider previous subgoal to find an alternative solution
- Begin search where previous search left off
- Can take lots of time and space because may find all possible proofs to every subgoal.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Trace

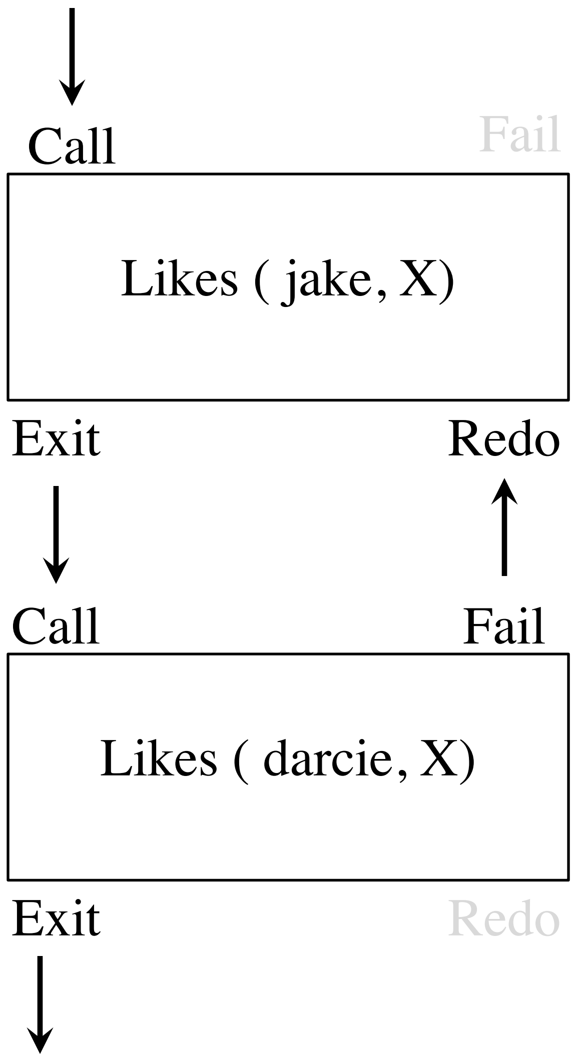
- **Built-in** structure that displays instantiations at each step
- *Tracing model* of execution - four events:
  - *Call* --beginning of attempt to satisfy goal
  - *Exit* --when a goal has been satisfied
  - *Redo* --when backtrack occurs
  - *Fail* --when goal fails.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Example of Trace

```
likes(jake, chocolate).
likes(jake, apricots).
likes(darcie, licorice).
likes(darcie, apricots).

trace.
likes(jake, X), likes(darcie, X).
(1)  1 Call: likes(jake, _0)?
(1) 1 Exit: likes(jake, chocolate)
(2) 1 Call: likes(darcie, chocolate)?
(2) 1 Fail: likes(darcie, chocolate)
(1)  1 Redo: likes(jake, _0)?
(1) 1 Exit: likes(jake, apricots)
(3) 1 Call: likes(darcie, apricots)?
(3) 1 Exit: likes(darcie, apricots)
X = apricots
```



Note: _0 : **internal** variable used to store instantiated values.

Copyright © 2018 Pearson. All rights reserved.

Prolog: Simple Arithmetic

- Prolog supports integer variables and integer arithmetic
  - e.g., +(7, x)
- `is` operator: takes an arithmetic expression as right operand and variable as left operand
  - e.g., `A is B / 17 + C.`

    /* B and C are instantiated, but A is **not** */
  - e.g, 1 is sin(pi/2). /* false; as sin(pi/2) = 1.0 */
- Not the same as an assignment statement!
  - The following is illegal:

```
Sum is Sum + Number.
```

```
Ref: http://www.swi-prolog.org/pldoc/man?predicate=is/2
```

Copyright © 2018 Pearson. All rights reserved.

Prolog: Example

```
speed(ford,100).
speed(chevy,105).
speed(dodge,95).
speed(volvo,80).
time(ford,20).
time(chevy,21).
time(dodge,24).
time(volvo,24).
distance(X,Y) :-   speed(X,Speed),
                   time(X,Time),
                   Y is Speed * Time.
```

A query: `distance(chevy, Chevy_Distance).`

`105 * 21 = 2205.`

Copyright © 2018 Pearson. All rights reserved.

Prolog: List Structures

- Other basic data structure (besides atomic propositions we have already seen): list
- *List* is a sequence of *any* number of elements
- Elements can be atoms, atomic propositions, or other terms (including other lists)

e.g.,

```
[apple, prune, grape, kumquat]
[]           /* (empty list) */
[X | Y]      /* head X, tail Y; head : CAR, tail: CDR */
[ X | Y ] = [ a, b, c]     /* x = a; y = [b , c] */.
```

Copyright © 2018 Pearson. All rights reserved.

Prolog: Append

```
append([], List, List).
append([Head | List_1], List_2, [Head | List_3]) :-
            append (List_1, List_2, List_3).
e.g.,
 append([bob, jo], [jake, darcie], Family).
  Family = [ bob, jo, jake, darcie]
```

Notes:

1. **Head** the **first** element of the list
2. List_1 and List_2 are lists to be appended;
   List_3 is the **resulting** list
3. **append** is a **predicate** that returns **yes** or **no**.

Copyright © 2018 Pearson. All rights reserved.

Prolog: reverse, member

```
reverse([], []).
reverse([Head | Tail], List) :-
  reverse(Tail, Result),
           append(Result, [Head], List).
```

e.g., Q = [c, b, a]

reverse([a, b, c], Q). ➔ yes

```
member(Element, [Element | _ ]).
member(Element, [ _ | List]) :-
                           member(Element, List).
```

e.g., member(a, [b, a, c]). ➔ yes

Note: The underscore character (_) means an **anonymous** variable—it means we do not care what instantiation it might get from unification.

Copyright © 2018 Pearson. All rights reserved.

Deficiencies of Prolog

- Resolution order control
  - In a pure logic programming environment, the order of attempted matches is **nondeterministic** and all matches would be attempted *concurrently*
- The closed-world assumption
  - The only knowledge is what is in the **database**
- The negation problem
  - Anything not stated in the database is assumed to be **false**
- Essential limitations
  - It is easy to state a sort process in logic, but difficult to actually do—it doesn't know how to sort.

Copyright © 2018 Pearson. All rights reserved.

Applications of Logic Programming

- Relational database management systems
- Expert systems
- Natural language processing.

Copyright © 2018 Pearson. All rights reserved.