Introduction of Programming Languages

Programming Language Concepts

- What is a programming language?
- Why are there so many programming languages?
- What are the types of programming languages?
- Does the world need new languages?

What is a Programming Languages

- A programming language is a set of rules that provides a way of telling a computer what operations to perform.
- A programming language is a set of rules for communicating an algorithm
- It provides a linguistic framework for describing computations

What is a Programming Language?

A programming language is a notational system for describing computation in a machine-readable and human-readable form.

A programming language is a tool for developing executable models for a class of problem domains.

What is a Programming Language

- English is a natural language. It has words, symbols and grammatical rules.
- A programming language also has words, symbols and rules of grammar.
- The grammatical rules are called syntax.
- Each programming language has a different set of syntax rules.

Why Are There So Many Programming Languages

- Why does some people speak French?
- Programming languages have evolved over time as better ways have been developed to design them.
  - First programming languages were developed in the 1950s
  - Since then thousands of languages have been developed
- Different programming languages are designed for different types of programs.

Levels of Programming Languages

High-level program

```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

Low-level program

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

```
0001001001000101
0010010011101100
10101101001...
```

What Are the Types of Programming Languages

- First Generation Languages
- Second Generation Languages
- Third Generation Languages
- Fourth Generation Languages
- Fifth Generation Languages

First Generation Languages

- Machine language
  - Operation code – such as addition or subtraction.
  - Operands – that identify the data to be processed.
  - Machine language is machine dependent as it is the only language the computer can understand.
  - Very efficient code but very difficult to write.

Second Generation Languages

- Assembly languages
  - Symbolic operation codes replaced binary operation codes.
  - Assembly language programs needed to be "assembled" for execution by the computer. Each assembly language instruction is translated into one machine language instruction.
  - Very efficient code and easier to write.

Third Generation Languages

- Closer to English but included simple mathematical notation.
  - Programs written in source code which must be translated into machine language programs called object code.
  - The translation of source code to object code is accomplished by a machine language system program called a compiler.

Third Generation Languages (cont'd.)

- Alternative to compilation is interpretation which is accomplished by a system program called an interpreter.
- Common third generation languages
  - FORTRAN
  - COBOL
  - C and C++
  - Visual Basic

Fourth Generation Languages

- A high level language (4GL) that requires fewer instructions to accomplish a task than a third generation language.
- Used with databases
  - Query languages
  - Report generators
  - Forms designers
  - Application generators

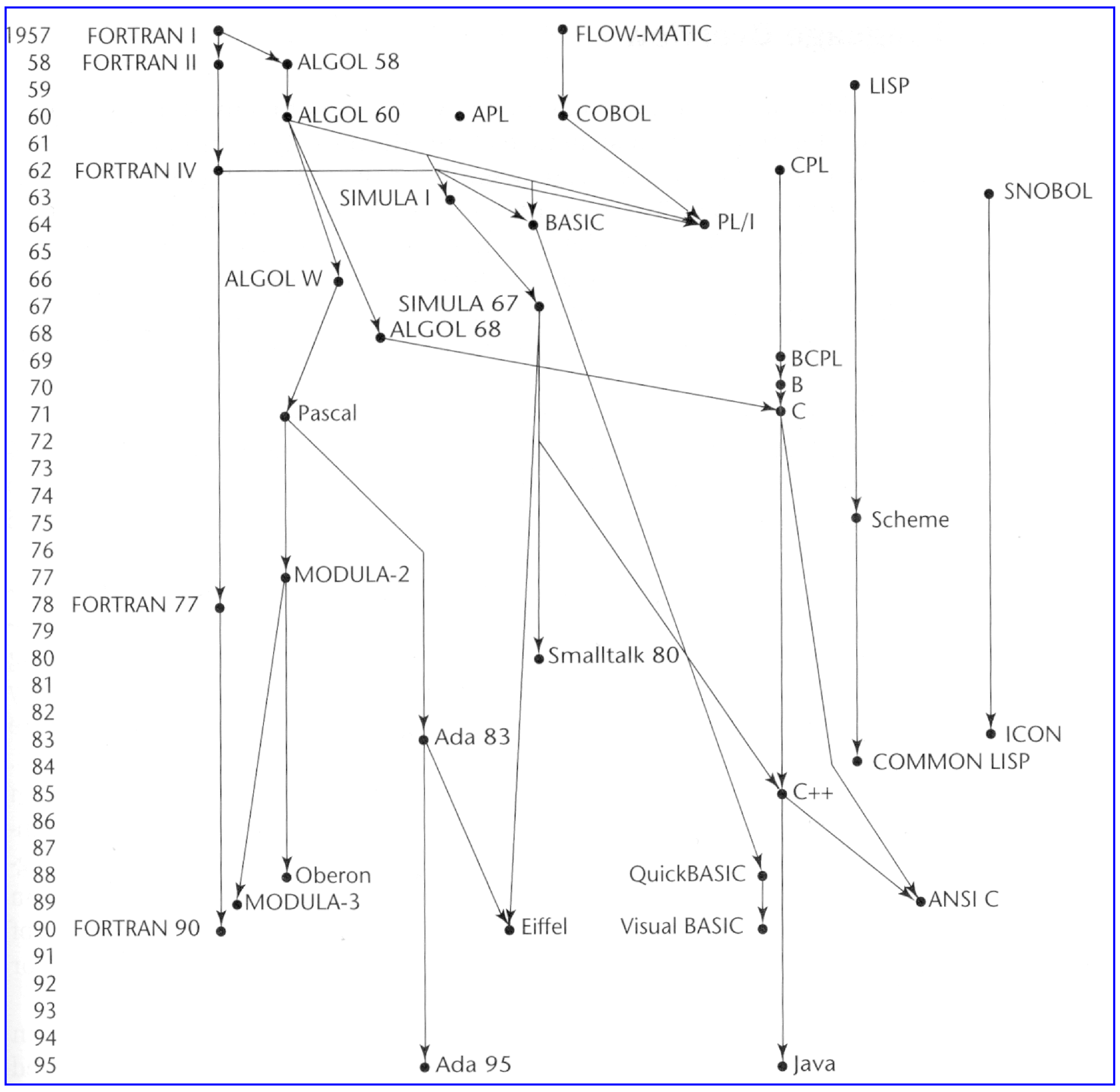
Fifth Generation Languages

- Declarative languages
- Functional(?): Lisp, Scheme, SML
  - Also called applicative
  - Everything is a function
- Logic: Prolog
  - Based on mathematical logic
  - Rule- or Constraint-based

Beyond Fifth Generation Languages

- Though no clear definition at present, natural language programs generally can be interpreted and executed by the computer with no other action by the user than stating their question.
- Limited capabilities at present.

Language Family Tree



The principal paradigms

- Imperative Programming (C)
- Object-Oriented Programming (C++)
- Logic/Declarative Programming (Prolog)
- Functional/Applicative Programming (Lisp)

Programming Languages

- Two broad groups
  - Traditional programming languages
    - Sequences of instructions
    - First, second and some third generation languages
  - Object-oriented languages
    - Objects are created rather than sequences of instructions
    - Some third generation, and fourth and fifth generation languages

Traditional Programming Languages

- FORTRAN
  - FORmula TRANslation.
  - Developed at IBM in the mid-1950s.
  - Designed for scientific and mathematical applications by scientists and engineers.

Traditional Programming Languages (cont'd.)

- COBOL
  - COmmon Business Oriented Language.
  - Developed in 1959.
  - Designed to be common to many different computers.
  - Typically used for business applications.

Traditional Programming Languages (cont’d.)

- BASIC
  - Beginner’s All-purpose Symbolic Instruction Code.
  - Developed at Dartmouth College in mid 1960s.
  - Developed as a simple language for students to write programs with which they could interact through terminals.

Traditional Programming Languages (cont’d.)

- C
  - Developed by Bell Laboratories in the early 1970s.
  - Provides control and efficiency of assembly language while having third generation language features.
  - Often used for system programs.
  - UNIX is written in C.

Object-Oriented Programming Languages

- Simula
  - First object-oriented language
  - Developed by Ole Johan Dahl in the 1960s.
- Smalltalk
  - First purely object-oriented language.
  - Developed by Xerox in mid-1970s.
  - Still in use on some computers.

Object-Oriented Programming Languages (cont'd.)

- C++
  - It is C language with additional features.
  - Widely used for developing system and application software.
  - Graphical user interfaces can be developed easily with visual programming tools.

Object-Oriented Programming Languages (cont'd.)

- JAVA
  - An object-oriented language similar to C++ that eliminates lots of C++'s problematic features
  - Allows a web page developer to create programs for applications, called applets that can be used through a browser.
  - Objective of JAVA developers is that it be machine, platform and operating system independent.

Special Programming Languages

- Scripting Languages
  - JavaScript and VBScript
  - Php and ASP
  - Perl and Python
- Command Languages
  - sh, csh, bash
- Text processing Languages
  - LaTex, PostScript

Special Programming Languages (cont'd.)

- HTML
  - HyperText Markup Language.
  - Used on the Internet and the World Wide Web (WWW).
  - Web page developer puts brief codes called tags in the page to indicate how the page should be formatted.

Special Programming Languages (cont'd.)

- XML
  - Extensible Markup Language.
  - A language for defining other languages.

A language is a language is a language

- Programming languages are <u>languages</u>
- When it comes to mechanics of the task, learning to speak and use a programming language is in many ways like learning to speak a human language
- In both kind of languages you have to learn new vocabulary, syntax and semantics (new words, sentence structure and meaning)
- And both kind of language require considerable practice to make perfect.

But there is a difference!

- Computer languages lack ambiguity and vagueness
- In English sentences such as *I saw the man with a telescope* (Who had the telescope?) or *Take a pinch of salt* (How much is a pinch?)
- In a programming language a sentence either means one thing or it means nothing

What determines a “good” language

- Formerly: Run-time performance
  - (Computers were more expensive than programmers)
- Now: Life cycle (human) cost is more important
  - Ease of designing, coding
  - Debugging
  - Maintenance
  - Reusability
- FADS

Criteria in a good language design

- **Writability:** The quality of a language that enables a programmer to use it to express a computation clearly, correctly, concisely, and quickly.
- **Readability:** The quality of a language that enables a programmer to understand and comprehend the nature of a computation easily and accurately.
- **Orthogonality:** The quality of a language that features provided have as few restrictions as possible and be combinable in any meaningful way.
- **Reliability:** The quality of a language that assures a program will not behave in unexpected or disastrous ways during execution.
- **Maintainability:** The quality of a language that eases errors can be found and corrected and new features added.

Criteria (Continued)

- **Generality:** The quality of a language that avoids special cases in the availability or use of constructs and by combining closely related constructs into a single more general one.
- **Uniformity:** The quality of a language that similar features should look similar and behave similar.
- **Extensibility:** The quality of a language that provides some general mechanism for the user to add new constructs to a language.
- **Standardability:** The quality of a language that allows programs written to be transported from one computer to another without significant change in language structure.
- **Implementability:** The quality of a language that provides a translator or interpreter can be written. This can address to complexity of the language definition.

Introduction to Logic Programming

A Little History

- Prolog was invented by Alain Colmerauer, a professor of computer science at the university of Aix-Marseille in France, in 1972
- The first application of Prolog was in natural language processing
- Prolog stands for programming in logic (PROgrammation en LOgique)
- Its theoretical underpinning are due to Donald Loveland of Duke university through Robert Kowalski (formerly) of the university of Edinburgh

Logic Programming

- Prolog is the only successful example of the family of logic programming languages
- A Prolog program is a theory written in a subset of first-order logic, called Horn clause logic
- Prolog is declarative. A Prolog programmer concentrates on *what* the program needs to do, not on *how* to do it
- The other major language for Artificial Intelligence programming is LISP, which is a functional (or applicative) language

Knight Moves on a Chessboard

- % This example is from unpublished (to the best of my knowledge) notes by Maarten
- % Van Emden.
- /* The extensional representation of the (knight) move relation follows. It
- consists of 336 facts; only a few are shown. In particular, all moves from
- position (5,3) on the chess board are shown. */

- move(1,1,2,3).
- move(1,1,3,2).
- ....
- move(5,3,6,5).
- move(5,3,7,4).
- move(5,3,7,2).
- move(5,3,6,1).
- move(5,3,4,1).
- move(5,3,3,2).
- move(5,3,3,4).
- move(5,3,4,5).
- ...
- move(8,8,7,6).
- move(8,8,6,7).

Intensional Representation of Moves

- /* The intensional representation of the (knight) move relation follows. It
- consists of facts (to define extensionally the relation succ/2) and rules (to
- define the relations move, diff1, and diff2. */

- move(X1,Y1,X2,Y2) :- diff1(X1,X2), diff2(Y1,Y2).
- move(X1,Y1,X2,Y2) :- diff2(X1,X2), diff1(Y1,Y2).

- diff1(X,Y) :- succ(X,Y).
- diff1(X,Y) :- succ(Y,X).

- diff2(X,Z) :- succ(X,Y), succ(Y,Z).
- diff2(X,Z) :- succ(Z,Y), succ(Y,X).

- succ(1,2).
- succ(2,3).
- succ(3,4).
- succ(4,5).
- succ(5,6).
- succ(6,7).
- succ(7,8).

Defining Relations by Facts

- parent(tom,bob).
- parent is the name of a relation
    - A relation of arity n is a function from n-tuples (elements of a Cartesian product) to {true, false}. (It can also be considered a subset of the n-tuples.)
- parent(pam, bob). parent(tom,bob). parent(tom,liz). parent(bob, ann). parent(bob,pat). parent(pat,jim).
- A relation is a collection of *facts*

Queries

?-parent(bob,pat).

yes

- A query and its answer, which is correct for the relation defined in the previous slide: this query *succeeds*

?-parent(liz,pat).

no

- A query and its answer, which is correct for the relation defined in the previous slide: this query *fails*

More Queries

?-parent(tom,ben). /* who is Ben? */

?-parent(X,liz). /* Wow! */

?-parent(bob,X). /* Bob's children */

?-parent(X,Y). /* The relation, fact by fact */

```
parent( pam,bob).
parent( tom,bob).
parent( tom,liz).
parent( bob,ann).
parent( bob,pat).
parent( pat,jim).
```

Composite Queries

- Grandparents:
  ?-parent(Y,jim), parent(X,Y).
  - the comma stands for "and"

  ?-parent(X,Y), parent(Y,jim).
  - order should not matter, and ***it does not!***
- Grandchildren:
  ?-parent(tom,X), parent(X,Y).
- Common parent, i.e. (half-)sibling:
  ?-parent(X,ann), parent(X,pat).

```
parent( pam,bob).
parent( tom,bob).
parent( tom,liz).
parent( bob,ann).
parent( bob,pat).
parent( pat,jim).
```

Facts and Queries

- Relations and queries about them
- Facts are a kind of *clause*
  - *Prolog programs consist of a list of clauses*
- The arguments of relations are atoms or variables (a kind of *term*)
- Queries consist of one or more *goals*
- Satisfiable goals succeed; unsatisfiable goals fail

Defining Relations by Rules

- The offspring relation:

  For all X and Y,
  Y is an offspring of X if
  X is a parent of Y
- This relation is defined by a rule, corresponding to the Prolog clause

  offspring(Y,X) :- parent(X,Y).
- Alternative reading:

  For all X and Y,
  if X is a parent of Y,
  then Y is an offspring of X

Rules

- Rules are clauses. Facts are clauses
- A rule has a condition and a conclusion
- The conclusion of a Prolog rule is its *head*
- The condition of a Prolog rule is its *body*
- If the condition of a rule is true, then it follows that its conclusion is true also

How Prolog Rules are Used

- Prolog rules may be used to define relations
- The offspring relation is defined by the rule offspring(Y,X) :- parent(X,Y):
  - if (X,Y) is in the parent relation, then (Y,X) is in the offspring relation
- When a goal of the form offspring(Y,X) is set up, the goal succeeds if parent(X,Y) succeeds
- Procedurally, when a goal matches the head of a rule, Prolog sets up its body as a new goal

Example

?-offspring(liz,tom).

- No fact matches this query
- The head of the clause
  offspring(Y,X) :- parent(X,Y) does
- Y is replaced with liz, X is replaced with tom
- The instantiated body *parent(tom,liz)* is set up as a new goal
- ?-parent(tom,liz) succeeds
- offspring(liz,tom) therefore succeeds too

```
parent( pam,bob).
parent( tom,bob).
parent( tom,liz).
parent( bob,ann).
parent( bob,pat).
parent( pat,jim).

offspring( Y,X) :- parent( X,Y).

% offspring(liz,tom).
```

More Family Relations

- *female* and *male* are defined *extensionally*, i.e., by facts; *mother* and *grandparent* are defined *intensionally*, I.e., by rules
- female(pam). … male(jim).
- mother(X,Y) :- parent(X,Y), female(X).
- grandparent(X,Z) :- parent(X,Y), parent(Y,Z).

Sister

- sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X).
- Try:

  ?-sister(X,pat).

  X = ann;

  X = pat /* Surprise! */
- (Half-)sisters have a common parent *and* are different people, so the correct rule is:
- sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X), different(X,Y).
  - (or: sister(X,Y) :- parent(Z,X), parent(Z,Y), parent(W,X), parent(W,Y), female(X), different(Z,W), different(X,Y).)

Clauses and Instantiation

- Facts are clauses without body
- Rules are clauses with both heads and non-empty bodies
- Queries are clauses that only have a body (!)
- When variables are substituted by constants, we say that they are *instantiated*.

Universal Quantification

- Variables are universally quantified, but beware of variables that only appear in the body, as in
- *haschild(X) :- parent(X,Y).*
- which is best read as:

  for all X,

  X has a child if

  there exists some Y such that X is a parent of Y

    - (I.e.: for all X and Y, if X is a parent of Y, then X has a child)

Ancestor

- ancestor(X,Z) :- parent(X,Z).
- ancestor(X,Z) :- parent(X,Y), parent(Y,Z).
- ancestor(X,Z) :- parent(X,Y1),
  parent(Y1,Y2,),
  parent(Y2,Z).

etc.

- When do we stop?
- The length of chain of people between the predecessor and the successor should not arbitrarily bounded.

A Recursive Rule

- For all X and Z,
  X is a predecessor of Z if
  there is a Y such that
  (1) X is a parent of Y and
  (2) Y is a predecessor of Z.
- predecessor(X,Z) :-
  parent(X,Y),
  predecessor(Y,Z).

The Family Program

- Comments
  - /* This is a comment */
  - % This comment goes to the end of the line
- SWI Prolog warns us when the clauses defining a relation are not contiguous.

```
% Figure 1.8   The family program.


parent( pam, bob).       % Pam is a parent of Bob
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).

female( pam).            % Pam is female
male( tom).              % Tom is male
male( bob).
female( liz).
female( ann).
female( pat).
male( jim).

offspring( Y, X)  :-     % Y is an offspring of X if
   parent( X, Y).        % X is a parent of Y

mother( X, Y)  :-        % X is the mother of Y if
   parent( X, Y),        % X is a parent of Y and
   female( X).           % X is female

grandparent( X, Z)  :-   % X is a grandparent of Z if
   parent( X, Y),        % X is a parent of Y and
   parent( Y, Z).        % Y is a parent of Z

sister( X, Y)  :-        % X is a sister of Y if
   parent( Z, X),
   parent( Z, Y),        % X and Y have the same parent and
   female( X),           % X is female and
   different( X, Y).     % X and Y are different

predecessor( X, Z)  :-   % Rule pr1: X is a predecessor of Z
   parent( X, Z).

predecessor( X, Z)  :-   % Rule pr2: X is a predecessor of Z
   parent( X, Y),
   predecessor( Y, Z).
```

Declarative Sorting

```
sort1(A, B) :- permutation(A,B), sorted(B).

permutation([],[]).
permutation(B, [A|D]) :- del(A,B,C),
   permutation(C,D).

sorted([]).
sorted([X]).
sorted([A, B | C]) :- A=<B, sorted([B|C]).

del(A, [A|B], B).
del(B, [A|C], [A|D]) :- del(B, C, D).
```

Declarative and Procedural Meaning of Prolog Programs

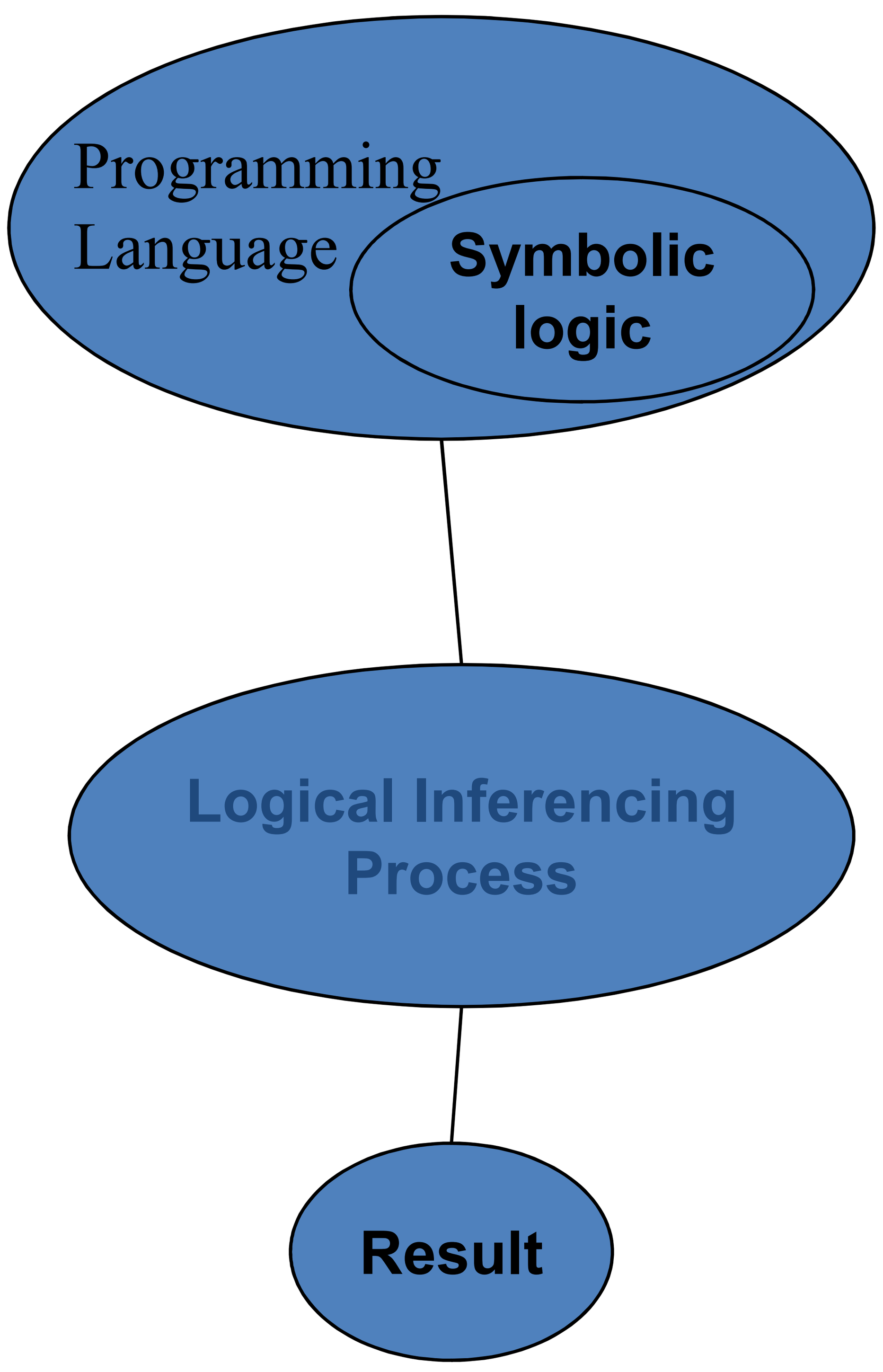
- The declarative meaning is concerned with the relations defined by the program: what the program states and logically entails
- The procedural meaning is concerned with how the output of the program is obtained, i.e., how the relations are actually evaluated by the Prolog system
- It is best to concentrate on the declarative meaning when writing Prolog programs
- Unfortunately, sometimes the programmer must also consider procedural aspect (for reasons of efficiency or even correctness

Logic Programming Languages

Objective

- To introduce the concepts of logic programming and logic programming languages
- To introduce a brief description of a subset of prolog

Introduction



- Logic programs are declarative programs
  - Specify the desired results - true
  - State the fact

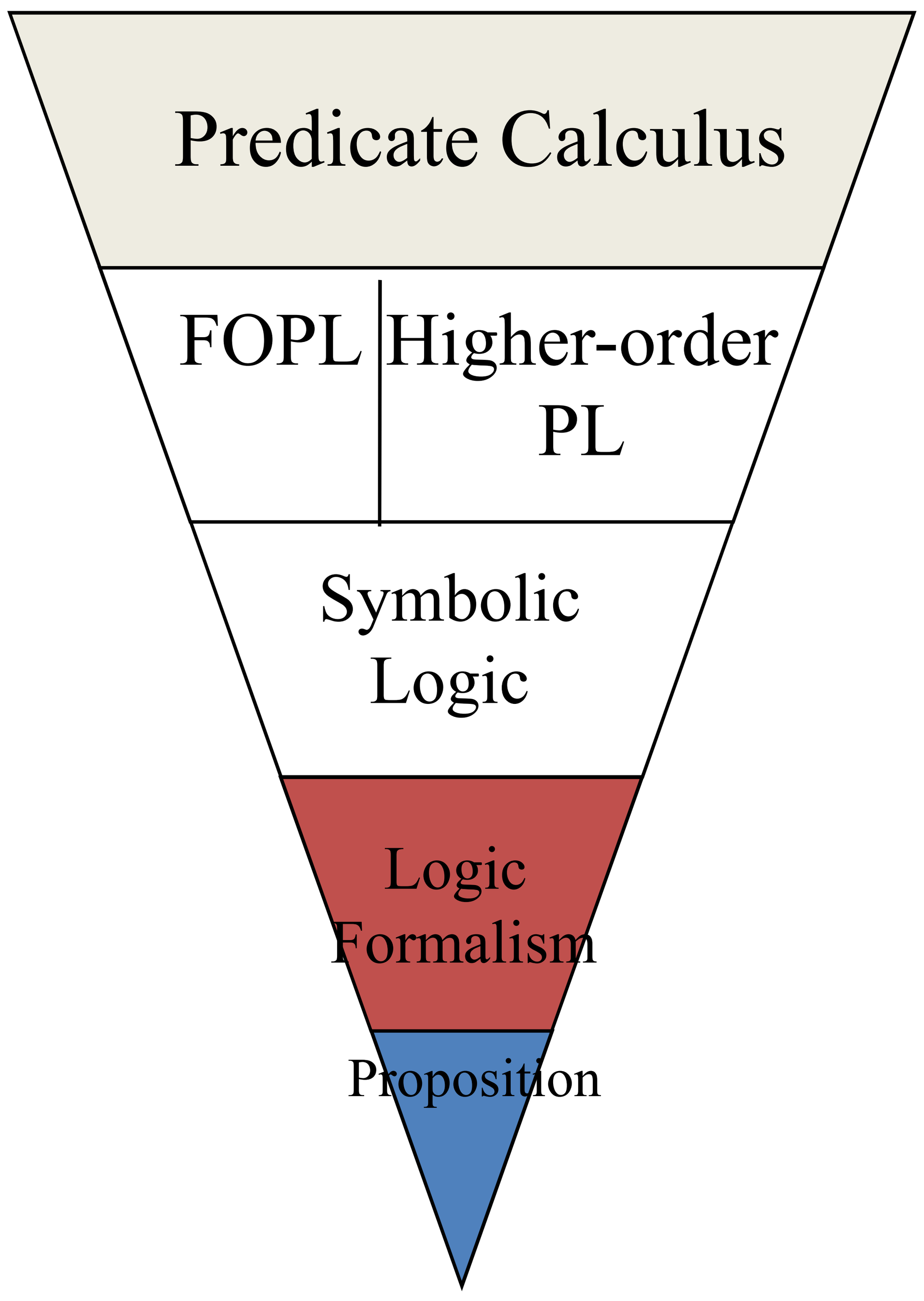
Introduction

- The major difference between logic programming and other programming languages (imperative and functional)
  - Every data item that exist in logic programming has written in specific representation (symbolic logic)
- Prolog is a logic programming that widely used logic language

Introduction

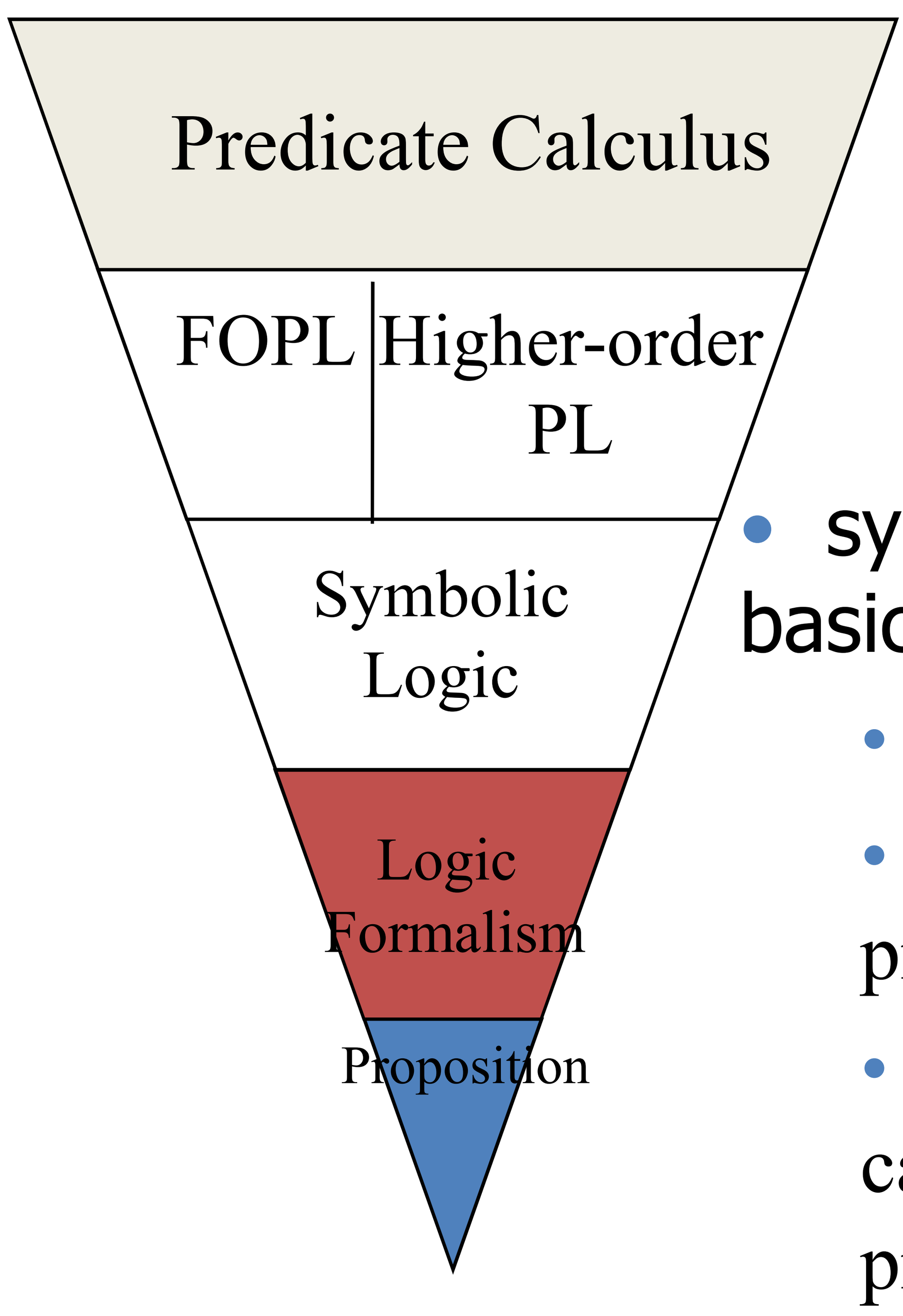
- Prolog specified the way of computer carries out the computation and it is divided to 3 parts:
  - logical declarative semantic of prolog
  - new fact prolog can infer from the given fact
  - explicit control information supplied by the programmer

Symbolic representation: Predicate Calculus



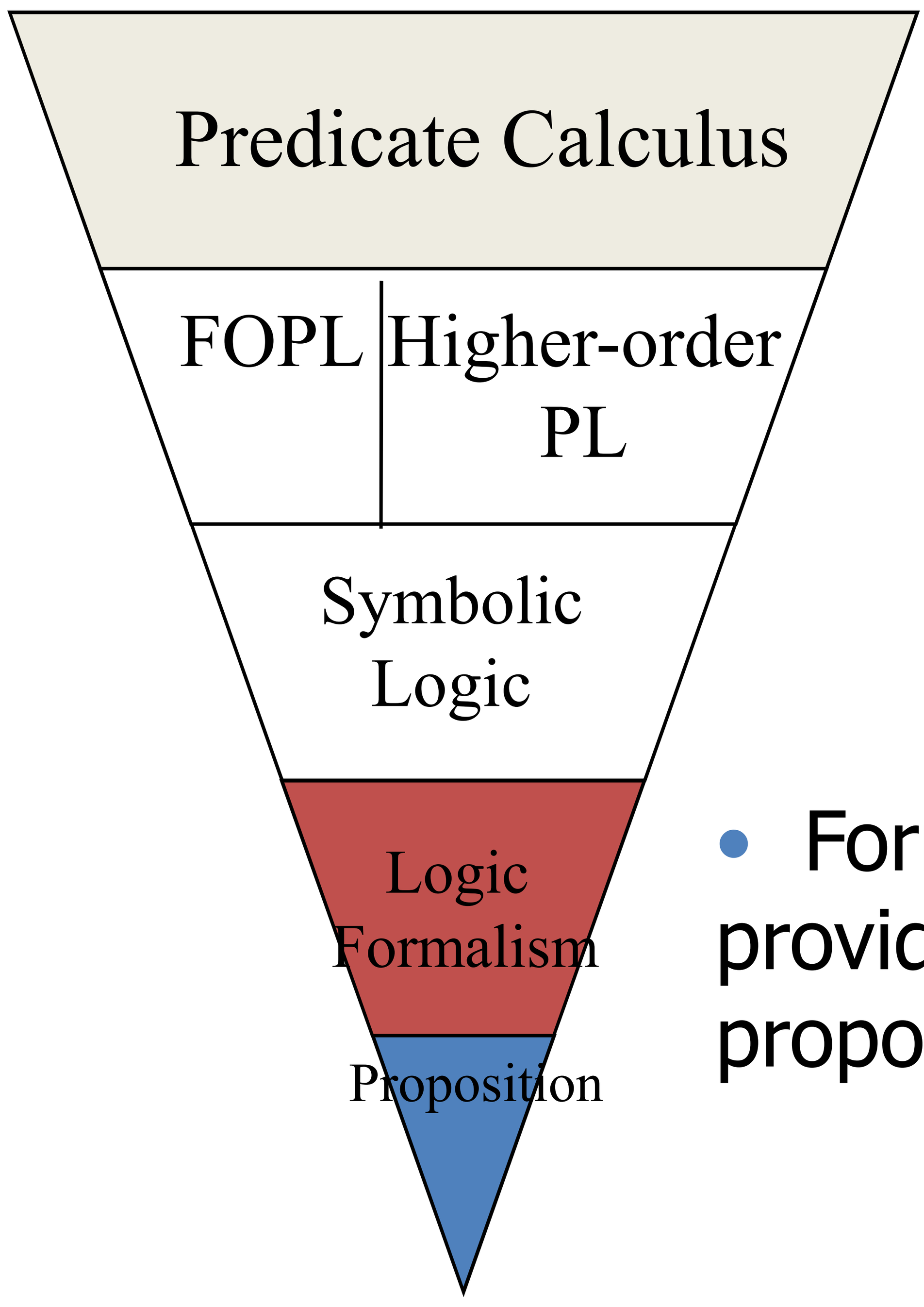
- mathematical representation of formal logic
  - is a particular form of symbolic logic that is used for logic programming

Symbolic representation: Predicate Calculus



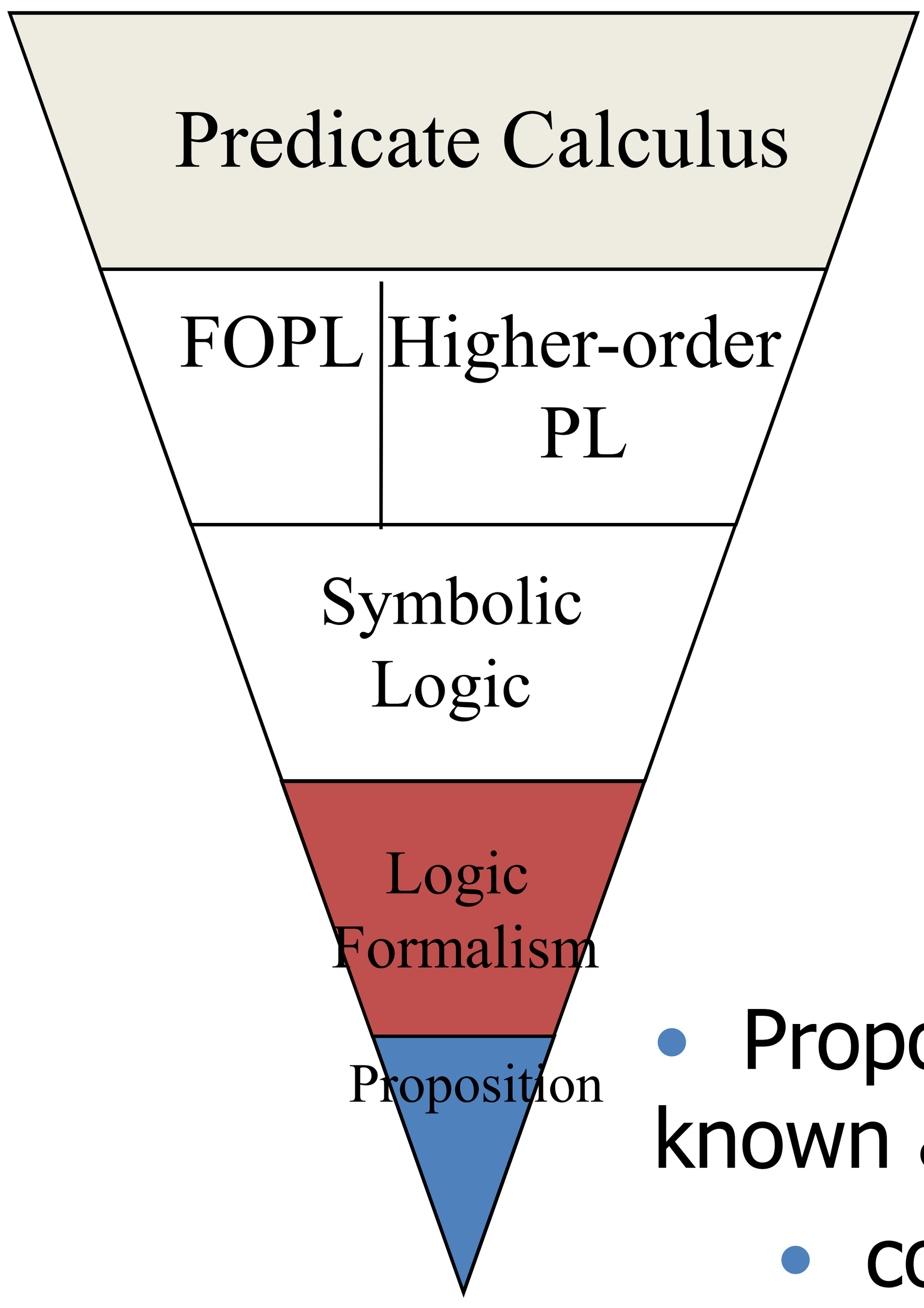
- symbolic logic used for the three basic need of formal logic
  - to express propositions
  - to express the relationships between propositions
  - to describe how new propositions can be inferred from other propositions that are assumed to be true

Symbolic representation: Predicate Calculus



- Formal logic was developed to provide a method for describing proposition.

Symbolic representation: Predicate Calculus



- Proposition is a logical statement also known as fact
  - consist of object and relationships of object to each other

Proposition

- Object:
  - Constant represents an object, or
  - Variable represent different objects at different times
- Simple proposition called as atomic propositions, consist of compound terms – one element of mathematic relation which written in a form that has the appearance of mathematical function notation.

Example (constants):

single parameter (1-tuple): man(jake)

double parameter (2-tuples): like(bob, steak)

Proposition

- Object:
  - Constant represents an object, or
  - Variable represent different objects at different times
- Simple proposition called as atomic propositions, consist of compound terms – one element of mathematic relation

Example:

single parameter (1-tuple): man(jake)

double parameter (2-tuples): like(bob, steak)

functor shows the names the relation

Proposition

- Object:
  - Constant represents an object, or
  - Variable represent different objects at different times
- Simple proposition called as atomic propositions, consist of compound terms – one element of mathematic relation

Example:

single parameter (1-tuple): man(jake)

double parameter (2-tuples): like(bob, steak)

list of parameter

Proposition

- Two modes for proposition:
  - proposition defined to be true (fact), and
  - the truth of the proposition is something that is to be determined (queries)
- Compound propositions have two or more atomic proposition, which are connected by logical operator (is the same way logic expression in imperative languages)

Logic 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	$\supset$	a $\supset$ b	a implies b
	$\subset$	a $\subset$ b	b implies a

Compound propositions

Example:

$a \cap b \supset c$

$a \cap \neg b \supset d$

$(a \cap (\neg b)) \supset d$

Precedence:

$\neg$ higher

$\cap \; \cup \; \equiv$

$\supset \; \subset$

lower

Variables in Proposition

- Variable known as *quantifiers*
- Predicate calculus includes two quanifiers, X – variable, and P – proposition

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

Variables in Proposition

Example

$\forall$X. (woman(X) $\supset$ human(X))

$\exists$X. (mother(mary, X) $\cap$ male(X))

Variables in Proposition

Example

$\forall X.(woman(X) \supset human(X))$

$\Rightarrow$ for any value of X, if X is a woman, then X is a human (NL: woman is a human)

$\exists X.(mother(mary, X) \cap male(X))$

Variables in Proposition

Example

$\forall X.(woman(X) \supset human(X))$

$\Rightarrow$ for any value of X, if X is a woman, then X is a human (NL: woman is a human)

$\exists X.(mother(mary, X) \cap male(X))$

$\Rightarrow$ there exist a value of X such that mary is the mother of X and X is a male (NL: mary has a son)

Clausal Form

- Simple form of proposition, it is a standard form for proposition without loss of generality
- Why we need to transform PC into CF?
  - too many different ways of stating propositions that have the same meaning

Example:

$\forall X.(woman(X) \supset human(X))$

$\forall X.(man(X) \supset human(X))$

Clausal Form

- General syntax for CF

  $B_1 \cup B_2 \cup \ldots \cup B_n \subset A_1 \cap A_2 \cap \ldots \cap A_m$

  $\Rightarrow$ if all the As are true, then at least one B is true

Example:

human(X) $\subset$ woman(X) $\cap$ man(X)

likes(bob, trout) $\subset$ likes(bob, fish) $\cap$ fish(trout)

Clausal Form

Example:

likes(bob, trout) $\subset$ likes(bob, fish) $\cap$ fish(trout)

consequent antecedent

- Characteristics of CF:
  - Existential quantifiers are not required
  - Universal quantifiers are implicit in the use of variables in the atomic propositions
  - No operator other than conjunction and disjunction are required

Clausal Form

Example:

likes(bob, trout) $\subset$ likes(bob, fish) $\cap$ fish(trout)

$\Rightarrow$ if bob likes fish and trout is a fish, then bob likes trout

Clausal Form

Example:

```
father(louis, al) ∪ father(louis, violet) ⊂
father(al,bob) ∩ mother(violet, bob) ∩
grandfather(louis, bob)
```

⇒ if al is bob's father and violet is bob's mother and louis is bob's grandfather, louis is either al's father or violet's father

Proving Theorems

- Method to inferred the collection of proposition
  - use a collections of proposition to determine whether any interesting or useful fact can be inferred from them
- Introduced by Alan Robinson (1965)

Proving Theorems

- Alan Robinson introduced resolution in automatic theorem proving
  - resolution is an inference rule that allows inferred proposition to be computed from given propositions
  - resolution was devised to be applied to propositions in clausal form

Proving Theorems

- Idea of resolution:

  $P1 \subset P2$ and $Q1 \subset Q2$

which given

  P1 is identical to Q2

  $\therefore\ Q1 \subset P2$

Proving Theorems

Example:

older(joanne, jake) $\subset$ mother(joanne, jake)
wiser(joanne, jake) $\subset$ older(joanne, jake)

$\therefore$ wiser(joanne, jake) $\subset$ mother(joanne, jake)

Proving Theorems

Example:

father(bob, jake) $\cup$ mother(bob, jake) $\subset$ parent(bob, jake)
gfather(bob, fred) $\subset$ father(bob, jake) $\cap$ father(jake, fred)

$\therefore$ gfather(bob, fred) $\cup$ mother(bob, jake)
$\subset$ parent(bob, jake) $\cap$ father(jake, fred)

Proving Theorems

- Process of determining useful values for variables during resolution – unification

- Unification
  - Hypotheses : original propositions
  - Goal: presented in negation of the theorem
  - Proposition in unification must be presented in Horn Clauses

Proving Theorems

- Horn Clauses:
  - Headed Horn Clauses

  Example:

  likes(bob, trout) $\subset$ likes(bob, fish) $\cap$ fish(trout)

  - Headless Horn Clauses

  Example:

  father(bob, jake)

Applications of Symbolic Computation

- Relational databases
- Mathematical logic
- Abstract problem solving
- Understanding natural language
- Design automation
- Symbolic equation solving
- Biochemical structure analysis
- Many areas of artificial intelligent