Why LISP and Prolog?

✔ All reasonable programming languages are “equivalent”

✗ Any computable function can be computed by a program written in any reasonable language

✔ But some languages have different features that make them better for some purposes than other languages

✔ So what’s special about LISP and Prolog?
LISP and Prolog: Languages for Artificial Intelligence

✔ Intelligence involves manipulating symbolic representations of the world (the “physical symbol system hypothesis”, A. Newell and H. Simon)

✔ Both LISP and Prolog provide built-in support for manipulating symbols and structured collections of symbols (for example, lists)

✔ Conventional languages concentrate on supporting the manipulation of numerical values of symbols

✗ Note that LISP and Prolog can also be used for numerical computing

✗ Note that conventional languages can also be used for symbolic computing
LISP and Prolog are Functional/Descriptive, not Procedural/Imperative languages

✔ The central difference:

✗ Procedural/Imperative languages use assignment statements
  • an assignment changes the value of a variable

✗ Functional/Descriptive languages do not use assignment statements
  • variables keep their values for the duration of a program
Procedural or imperative languages

✔ FORTRAN, C, Pascal, ALGOL, Ada, ...

✗ assignment statements are the basic construct

✗ a program is a series of instructions to modify the contents of machine memory
  • programmer describes what is to be computed
  • programmer organizes the computation into a sequence of small steps
  • programmer ensures proper management of machine memory during the computation
Functional or descriptive languages

✔ LISP, Prolog, ML, fp, SETL, ...

✗ descriptions of desired values are the basic construct

✗ a program is a relatively abstract description of what is to be computed

  • programmer describes what is to be computed, without specifying instruction sequence or memory management details

  • language interpreter or compiler takes care of detailed sequencing of operations and memory management

  • no assignment statements!

✗ (also called “applicative” or “declarative” languages)
Example: programs for mergesort

✔ Basic idea of mergesort:

✗ merging two sorted lists is an easy way to produce a longer sorted list

✗ sorting a very short list (zero or one element) is very easy

✗ so, sort a list by merging successively larger sublists

table 2,
p.165,
knuth vol 3
Mergesort in an imperative language

algorithm S,
p164,
knuth vol 3
Mergesort in LISP

(defun merge-sort (list-of-numbers)
  (if (small list-of-numbers) list-of-numbers
      (merge-lists-of-numbers
        (merge-sort (one-half list-of-numbers))
        (merge-sort (other-half list-of-numbers))))

(defun small (L)
  (or (eq (length L) 0) (eq (length L) 1)))

(defun one-half (L)
  (last L (ceiling (/ (length L) 2))))

(defun other-half (L)
  (ldiff L (one-half L))

(defun merge-lists-of-numbers (L1 L2)
  (merge ’list L1 L2 #’< ))

----------------------------------------------------------

USER(1): (merge-sort ’(503 87 512 61 908))

(61 87 503 512 908)
**Mergesort in Prolog**

```prolog
merge-sort([],[]).
merge-sort([X],[X]).
merge-sort(List,SortedList) :-
    divide(List, List1, List2),
    merge-sort(List1, Sorted1),
    merge-sort(List2, Sorted2),
    merge(Sorted1, Sorted2, SortedList).

divide([],[],[]).
divide([X], [X], []).
divide([X, Y | L], [X | L1], [Y | L2]) :-
    divide(L, L1, L2).
merge([],[],[]).
merge([X|L],[],[X|L]).
merge([],[X|L],[X|L]).
merge([X|L1],[Y|L2],[X,Y|L3]) :-
    X < Y, merge(L1,[Y|L2],L3).
merge([X|L1],[Y|L2],[Y,X|L3]) :-
    Y >= X, merge(L2,[X|L1],L3).

?- merge-sort([503 87 512 61 908], X).
X = [61, 87, 503, 512, 908]
```
Advantages of functional/descriptive languages

✔ If there are no assignment statements, values of variables cannot change during program execution

✔ If variables cannot change their values, expressions involving variables can be evaluated in any order

✔ If expressions can be evaluated in any order, the compiler can more easily optimize and/or parallelize

✗ LISP is the assembly language for Tera Inc.’s supercomputer

✗ Prolog was the language for Japan’s Fifth Generation supercomputer project
Advantages of functional/descriptive languages, cont’d.

✔ If there are no assignment statements, function calls cannot have obscure side effects
  ❌ example of obscure side effect: what is printed by this program?

```c
int a=3, b=4;

main()
{
    print("%d", a + F(b));
}

int F(int x)
{
    a = a + 1; /* assignment changing global variable */
    return (a * x);
}
```

✔ If function calls cannot have obscure side effects, the meaning of expressions is apparent from their written form

✔ If the meaning of expressions is apparent from their written form, programs are easier to understand
Disadvantages of functional/descriptive languages

✔ If the language’s interpreter or compiler is poorly designed, functional programs may be inefficient

✔ Low-level control of the machine can be difficult

✔ Use of assignment statements may be more natural for some problems
LISP

✔ LISP is a *functional* language
  ✗ programming = describing relations between data values by defining and applying functions
  ✗ use strategy of top-down, stepwise refinement for program development
  ✗ “pure LISP” has no assignment statements...

✔ LISP is based on the mathematical formalism of the lambda calculus
  ✗ supports a precise semantics
  ✗ makes reasoning about programs, proofs of correctness easier

✔ LISP is an *implicitly typed* language
  ✗ you don’t need to declare the type of any variable
  ✗ variable type is determined at run time
  ✗ good for rapid development of programs

✔ LISP supports operations on symbols and lists: (a (b c) (d (e f g)))
  ✗ after all, LISP stands for LISt Processing
  ✗ lists are an extremely powerful data type
  ✗ LISP supports many other data types also

✔ LISP programs are themselves lists: (defun foo (x y) (cons x y))
  ✗ it is easy for LISP programs to create other LISP programs
  ✗ this self-referentiality of LISP is an extremely powerful feature
  ✗ (LISP also stands for Lots of Irritating Silly Parentheses)))
Prolog

✓ Prolog is a *logic language*
  ✗ programming = describing relationships between data values by logical assertions
  ✗ use strategy of top-down, stepwise refinement for program development

✓ Prolog is based on the mathematical formalism of first-order predicate logic
  ✗ after all, Prolog stands for *Programming with logic*
  ✗ supports a precise semantics
  ✗ makes reasoning about programs, proofs of correctness easier

✓ Prolog is an *implicitly typed* language
  ✗ you don’t need to declare the type of any variable
  ✗ variable type is determined at run time
  ✗ good for rapid development of programs

✓ Prolog supports operations on symbols and lists

✓ Prolog relations are themselves lists