*"Learning to program has no more to do with designing interactive software than learning to touch type has to do with writing poetry."*
*- T. Nelson.*

# CSE341
# Programming Languages

Lecture 9 – November 2021

Prolog



Slides are taken from C. Li & W. He

1

## SWI-Prolog

- http://www.swi-prolog.org/
- Available for:
  Linux, Windows, MacOS



2

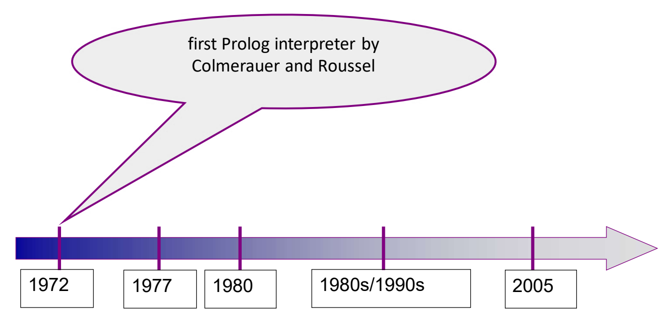
## Prolog

- Prolog:
  "Programming in Logic" (PROgrammation en LOgique)
- One (and maybe the only one) successful logic programming languages
- Useful in AI applications, expert systems, natural language processing, database query languages
- Declarative instead of procedural: "What" instead of "How"



3

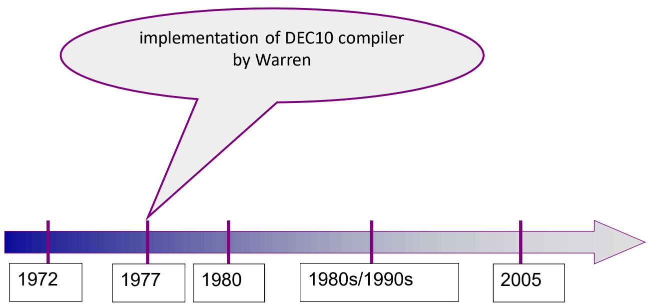
## History of Prolog






4

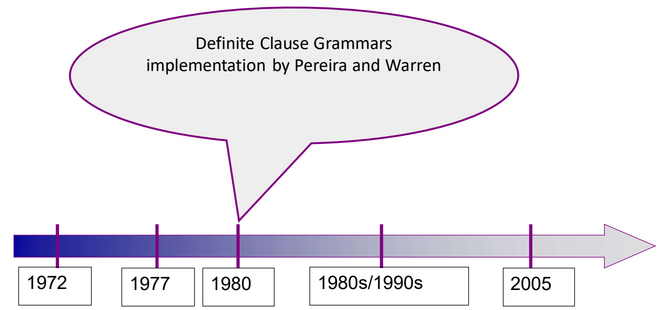
# History of Prolog






5

# History of Prolog

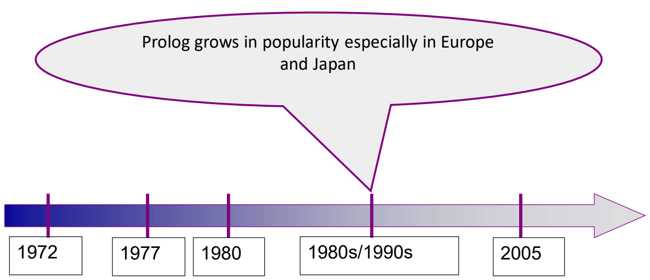





6

# History of Prolog

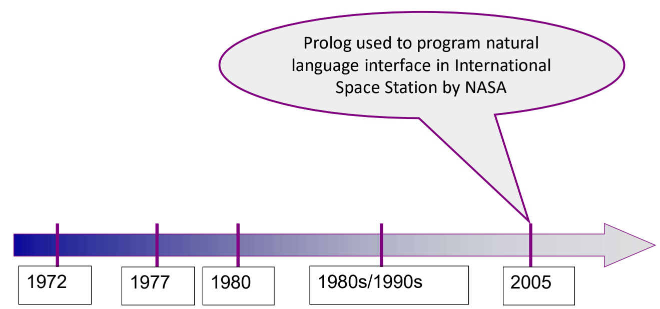
Fifth Generation Computing
Systems Project in Japan






7

# History of Prolog






8

## Logic Programming

- Program
  Axioms (facts): true statements
- Input to Program
  query (goal): statement true (theorems) or false?
- Thus
  Logic programming systems = deductive databases
  datalog



9

## Example

- Axioms:
  0 is a natural number. (Facts)
  For all x, if x is a natural number, then so is the successor of x.
- Query (goal).
  Is 2 natural number? (can be proved by facts)
  Is -1 a natural number? (cannot be proved)



10

## Another Example

- Axioms:
  The factorial of 0 is 1. (Facts)
  If m is the factorial of n - 1, then n * m is the factorial of n.
- Query:
  The factorial of 2 is 3?



11

## First-Order Predicate Calculus

- Logic used in logic programming:
  First-order predicate calculus
  First-order predicate logic
  Predicate logic
  First-order logic

  $\forall x\ (x \neq x+1)$
- Second-order logic
  $\forall S\ \forall\ x\ (x \in S \vee x \notin S)$



12

# First-Order Logic: Review

Slides from Tuomas Sandholm of CMU





## First-order Logic

- First-order logic (FOL) models the world in terms of
  - **Objects,** which are things with individual identities
  - **Properties** of objects that distinguish them from other objects
  - **Relations** that hold among sets of objects
  - **Functions,** which are a subset of relations where there is only one "value" for any given "input"
- Examples:
  - Objects: Students, lectures, companies, cars ...
  - Relations: Brother-of, bigger-than, outside, part-of, has-color, occurs-after, owns, visits, precedes, ...
  - Properties: blue, oval, even, large, ...
  - Functions: father-of, best-friend, second-half, one-more-than ...





## User Provides

- **Constant symbols,** which represent individuals in the world
  - Mary
  - 3
  - Green
- **Function symbols,** which map individuals to individuals
  - father-of(Mary) = John
  - color-of(Sky) = Blue
- **Predicate symbols,** which map individuals to truth values
  - greater(5,3)
  - green(Grass)
  - color(Grass, Green)





## FOL Provides

- **Variable symbols**
  - E.g., x, y, foo
- **Connectives**
  - Same as in PL: not ($\neg$), and ($\wedge$), or ($\vee$), implies ($\rightarrow$), if and only if (biconditional $\leftrightarrow$)
- **Quantifiers**
  - Universal $\forall$**x** or **(Ax)**
  - Existential $\exists$**x** or **(Ex)**

## Sentences built from Terms and Atoms

- A **term** (denoting a real-world individual) is a constant symbol, a variable symbol, or an n-place function of n terms.
  x and f($x_1$, ..., $x_n$) are terms, where each $x_i$ is a term.
  A term with no variables is a **ground term**
- An **atomic sentence** (which has value true or false) is an n-place predicate of n terms
- A **complex sentence** is formed from atomic sentences connected by the logical connectives:
  $\neg$P, P$\vee$Q, P$\wedge$Q, P$\rightarrow$Q, P$\leftrightarrow$Q where P and Q are sentences
- A **quantified sentence** adds quantifiers $\forall$ and $\exists$
- A **well-formed formula (wff)** is a sentence containing no "free" variables. That is, all variables are "bound" by universal or existential quantifiers.
  ($\forall$x)P(x,y) has x bound as a universally quantified variable, but y is free.



17

## A BNF for FOL

```
S := <Sentence> ;
<Sentence> := <AtomicSentence> |
          <Sentence> <Connective> <Sentence> |
          <Quantifier> <Variable>,... <Sentence> |
          "NOT" <Sentence> |
          "(" <Sentence> ")";
<AtomicSentence> := <Predicate> "(" <Term>, ... ")" |
                    <Term> "=" <Term>;
<Term> := <Function> "(" <Term>, ... ")" |
          <Constant> |
          <Variable>;
<Connective> := "AND" | "OR" | "IMPLIES" | "EQUIVALENT";
<Quantifier> := "EXISTS" | "FORALL" ;
<Constant> := "A" | "X1" | "John" | ... ;
<Variable> := "a" | "x" | "s" | ... ;
<Predicate> := "Before" | "HasColor" | "Raining" | ... ;
<Function> := "Mother" | "LeftLegOf" | ... ;
```



18

## Quantifiers

- **Universal quantification**
  - ($\forall$x)P(x) means that P holds for **all** values of x in the domain associated with that variable
  - E.g., ($\forall$x) dolphin(x) $\rightarrow$ mammal(x)
- **Existential quantification**
  - ($\exists$ x)P(x) means that P holds for **some** value of x in the domain associated with that variable
  - E.g., ($\exists$ x) mammal(x) $\wedge$ lays-eggs(x)
  - Permits one to make a statement about some object without naming it



19

## Translating English to FOL

**Every gardener likes the sun.**
$\forall$x gardener(x) $\rightarrow$ likes(x,Sun)

**You can fool some of the people all of the time.**
$\exists$x $\forall$t person(x) $\wedge$time(t) $\rightarrow$ can-fool(x,t)

**You can fool all of the people some of the time.**
$\forall$x $\exists$t (person(x) $\rightarrow$ time(t) $\wedge$can-fool(x,t))
$\forall$x (person(x) $\rightarrow$ $\exists$t (time(t) $\wedge$can-fool(x,t)) Equivalent

**All purple mushrooms are poisonous.**
$\forall$x (mushroom(x) $\wedge$ purple(x)) $\rightarrow$ poisonous(x)

**No purple mushroom is poisonous.**
$\neg\exists$x purple(x) $\wedge$ mushroom(x) $\wedge$ poisonous(x)
$\forall$x (mushroom(x) $\wedge$ purple(x)) $\rightarrow$ $\neg$poisonous(x) Equivalent

**There are exactly two purple mushrooms.**
$\exists$x $\exists$y mushroom(x) $\wedge$ purple(x) $\wedge$ mushroom(y) $\wedge$ purple(y) ^ $\neg$(x=y) $\wedge$ $\forall$z (mushroom(z) $\wedge$ purple(z)) $\rightarrow$ ((x=z) $\vee$ (y=z))

**Clinton is not tall.**
$\neg$tall(Clinton)



20

## First-Order Predicate Calculus: Example

- `natural(0)`
  `∀ X, natural(X) → natural(successor(x))`
- `∀ X and Y, parent(X,Y) → ancestor(X,Y).`
  `∀ A, B, and C, ancestor(A,B) and ancestor(B,C) → ancestor(A,C).`
  `∀ X and Y, mother(X,Y) → parent(X,Y).`
  `∀ X and Y, father(X,Y) → parent(X,Y).`
  `father(bill,jill).`
  `mother(jill,sam).`
  `father(bob,sam).`
- `factorial(0,1).`
  `∀ N and M, factorial(N-1,M) → factorial(N,N*M).`



21

## First-Order Predicate Calculus: Example

```
factorial(0,1).
factorial(1,1).
factorial(2,2).
factorial(3,6).
factorial(4,24).
factorial(5,120).
…
Factorial(100,…).
```



22

## First-Order Predicate Calculus: Statements

Symbols in statements:

- *Constants (a.k.a. atoms)*
  numbers (e.g., 0) or names (e.g., `bill`).
- *Predicates*
  Boolean functions (true/false) . Can have arguments. (e.g. `parent(X,Y)`).
- *Functions*
  non-Boolean functions (`successor(X)` ).
- *Variables*
  e.g., `X`.
- *Connectives (operations)*
  `and, or, not`
  `implication (→):a→b (b or not a)`
  `equivalence (↔): a↔b (a→b and b→a)`



23

## First-Order Predicate Calculus: Statements

- *Quantifiers*
  *universal quantifier* "for all" ∀
  *existential quantifier* "there exists" ∃
  *bound variable (a variable introduced by a quantifier)*
  *free variable*
- *Punctuation symbols*
  parentheses (for changing associativity and precedence.)
  comma
  period
- Arguments to predicates and functions can only be ***terms***:
  - Contain constants, variables, and functions.
  - Cannot have predicates, qualifiers, or connectives.



24

## Problem Solving

- Program = Data + Algorithms
- Program = Object.Message(Object)
- Program = Functions Functions
- Algorithm = Logic + Control

Programmers: facts/axioms/statements

Logic programming systems: prove goals from axioms

- We specify the logic itself, the system proves.
  - Not totally realized by logic programming languages. Programmers must be aware of how the system proves, in order to write efficient, or even correct programs.
- Prove goals from facts:
  - Resolution and Unification



25

## Proving things

- A **proof** is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference.
- The last sentence is the **theorem** (also called goal or query) that we want to prove.
- Example for the "weather problem"

| | | |
|---|---|---|
| 1 Hu | Premise | "It is humid" |
| 2 Hu→Ho | Premise | "If it is humid, it is hot" |
| 3 Ho | Modus Ponens(1,2) | "It is hot" |
| 4 (Ho∧Hu)→R | Premise | "If it's hot & humid, it's raining" |
| 5 Ho∧Hu | And Introduction(1,3) | "It is hot and humid" |
| 6 R | Modus Ponens(4,5) | "It is raining" |



26

## Horn Clause

- First-order logic too complicated for an effective logic programming system.
- Horn Clause: a fragment of first-order logic

  `b ← a1 and a2 and a3 … and an.`

  head

  body

  no "or" and no quantifier

  `b ←.` fact

  `←b.` query

- Variables in head: universally quantified

  Variables in body only: existentially quantified
- Need "or" in head? Multiple clauses



27

## Horn Clauses: Example

- First-Order Logic:

  `natural(0).`

  `∀X, natural(X) → natural(successor(x)).`

- Horn Clause:

  `natural(0).`

  `natural(successor(x)) ← natural(X).`



28

## Horn Clauses: Example

- First-Order Logic:

```
factorial(0,1).
∀ N and ∀ M, factorial(N-1,M) → factorial(N,N*M).
```

- Horn Clause:

```
factorial(0,1).
factorial(N,N*M) ← factorial(N-1,M).
```



29

## Horn Clauses: Example

- Horn Clause:

```
ancestor(X,Y) ← parent(X,Y).
ancestor(A,C) ← ancestor(A,B) and ancestor(B,C).
parent(X,Y) ← mother(X,Y).
parent(X,Y) ← father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```



30

## Horn Clauses: Example

- First-Order Logic:

```
∀ X, mammal(X) → legs(X,2) or legs(X,4).
```

- Horn Clause:

```
legs(X,4) ← mammal(X) and not legs(X,2).
legs(X,2) ← mammal(X) and not legs(X,4).
```



31

## Prolog syntax

- :- for ←
  , for and

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```



32

## Prolog BNF Grammar

```
<program> ::= <clause list> <query> | <query>
<clause list> ::= <clause> | <clause list> <clause>
<clause> ::= <predicate> . | <predicate> :- <predicate list>.
<predicate list> ::= <predicate> | <predicate list> , <predicate>
<predicate> ::= <atom> | <atom> ( <term list> )
<term list> ::= <term> | <term list> , <term>
<term> ::= <numeral> | <atom> | <variable> | <structure>
<structure> ::= <atom> ( <term list> )
<query> ::= ?- <predicate list>.
<atom> ::= <small atom> | ' <string> '
<small atom> ::= <lowercase letter> | <small atom> <character>
<variable> ::= <uppercase letter> | <variable> <character>
<lowercase letter> ::= a | b | c | ... | x | y | z
<uppercase letter> ::= A | B | C | ... | X | Y | Z | _
<numeral> ::= <digit> | <numeral> <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<character> ::= <lowercase letter> | <uppercase letter> |
            <digit> | <special>
<special> ::= + | - | * | / | \ | ^ | ~ | : | . | ? |  | # | $ | &
<string> ::= <character> | <string> <character>
```



33

## Resolution and Unification



34

## Resolution

- Resolution: Using a clause, replace its head in the second clause by its body, if they "match".
- $a \leftarrow a_1,\ \ldots,\ a_n.$
  $b \leftarrow b_1,\ \ldots,\ b_i,\ \ldots,\ b_m.$

  if $b_i$ matches $a$;
  $b \leftarrow b_1,\ \ldots,\ a_1,\ \ldots,\ a_n,\ \ldots,\ b_m.$



35

## Resolution: Another view

- Resolution: Combine two clauses, and cancel matching statements on both sides.
- $a \leftarrow a_1,\ \ldots,\ a_n.$
  $b \leftarrow b_1,\ \ldots,\ b_i,\ \ldots,\ b_m.$

  $\cancel{a},\ b \leftarrow a_1,\ \ldots,\ a_n,\ b_1,\ \ldots,\ \cancel{b_i},\ \ldots,\ b_m.$



36

## Problem solving in logic programming systems

- Program:
  - Statements/Facts (clauses).
- Goals:
  - Headless clauses, with a list of subgoals.
- Problem solving by resolution:
  - Matching subgoals with the heads in the facts, and replacing the subgoals by the corresponding bodies.
  - Cancelling matching statements.
  - Recursively do this, till we eliminate all goals. (Thus original goals proved.)



37

## Example

- Program:
  `mammal(human).`
- Goal:
  `← mammal(human).`
- Proving:
  ~~mammal(human)~~ ← ~~mammal(human)~~.
  ←.



38

## Example

- Program:

```
legs(X,2) ← mammal(X), arms(X,2).
legs(X,4) ← mammal(X), arms(X,0).
mammal(horse).
arms(horse,0).
```

- Goal:
  `← legs(horse,4).`
- Proving: ?



39

## Unification

- Unification: Pattern matching to make statements identical (when there are variables).
- Set variables equal to patterns: instantiated.
- In previous example:
  legs(X,4) and legs(horse,4) are unified.
  (X is instantiated with horse.)



40

## Unification: Example

- Euclid's algorithm for greatest common divisor
- Program:

```
gcd(U,0,U).
gcd(U,V,W) ← not zero(V), gcd(V, U mod V, W).
```

- Goals:

```
← gcd(15,10,X).
```



41

## Unification: Example

```
gcd(U,0,U).
gcd(U,V,W) ← not zero(V), gcd(V, U mod V, W).
```

1. gcd(15,10,X) does not match the first clause…
2. gcd(15,10,X) matches the second clause
   1. ← not zero(10), gcd(10, 15 mod 10, X)
   2. ← gcd(10, 5, X)
   3. ← not zero(5), gcd(5, 10 mod 5, X)
   4. ← gcd(5, 0, X)



42

## Things unspecified

- The order to resolve subgoals.
- The order to use clauses to resolve subgoals.
- Possible to implement systems that don't depend on the order, but too inefficient.
- Thus programmers must know the orders used by the language implementations. (Search Strategies)



43

## Example

- Program:

```
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).
ancestor(X,Y) :- parent(X,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
```

- Goals:

```
← ancestor(bill,sam).
← ancestor(X,bob).
```



44

## Prolog Search Strategy

- Applies resolution in strictly linear fashion
  - Replacing goals left to right
  - Considering clauses top to bottom order

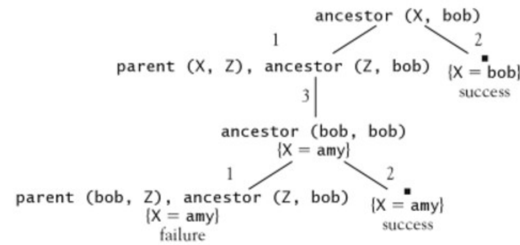
➔ A depth-first search on a tree of possible choices…



45

## Prolog Search Strategy

```
(1) ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
(2) ancestor(X, X).
(3) parent(amy, bob).
```






46

## Prolog Loops and Controls…

```
printpieces(L) :-append(X, Y, L),
                 write(X),
                 write(Y),
                 nl,
                 fail.
```

```
?- printpieces([1, 2]).
```

```
[][1,2]
[1][2]
[1,2] []
no
```

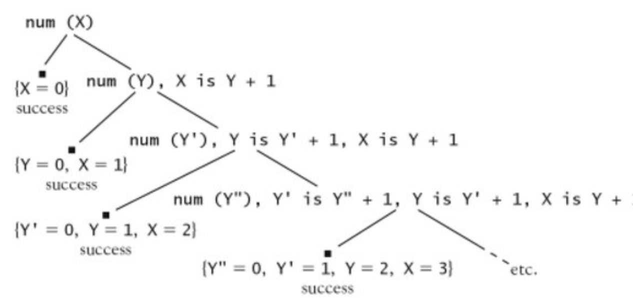
Backtracking…



47

## Prolog Loops and Controls…

```
(1) num(0).
(2) num(X) :- num(Y), X is Y + 1.
```






48