

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

Text T. Nelson.

---

# CSE341

## Programming Languages

Lecture 12.1 – December 11, 2012

Prolog

© 2012 Yakup Genç

Slides are taken from C. Li & W. He

# Today

---

- Prolog

# Midterm – Answers

---

Question 1: Answer the following short questions.

1.1. Why do programs define partial functions?

Because functions domain cannot be expressed exactly by the typing system.

1.2. What are the four major programming paradigms?

They are imperative/procedural, functional/applicative, logic, and object oriented.

1.3. What does DFA stand for and what components does it have?

DFA stands for deterministic finite automaton. A DFA has a unique start state and one or more final (accepting) states.

1.4. What does a configuration in a DFA represent?

A configuration in a DFA represents a state and the remaining input.

# Midterm – Answers

---

Question 1: Answer the following short questions.

1.5. What form do all production rules in a left regular grammar have? Explain the symbols you might use to answer this question.

All production rules in a left regular grammar has the form “ $A \rightarrow w$ ” or “ $A \rightarrow B w$ ” where “ $A$ ” is a non-terminal and “ $w$ ” is a terminal symbol.

1.6. What do two Ls in LL parsing stand for? Explain what you mean.

The first “L” in LL parsing stands for left to right scan of the input and the second “L” means that leftmost derivation is generated.

1.7. How is “dangling else” ambiguity resolved in C/C++.

By associating each else with the closest if or by using {}.

# Midterm – Answers

---

Question 1: Answer the following short questions.

1.8. What are attribute grammars?

Attribute grammars are augmented CFGs where each grammar symbol has an associated set of attributes.

1.9. What is the referencing environment of a statement?

Referencing environment of a statement is all the names that are visible in the statement.

1.10. What is a discriminated union? What other two names does it have?

A discriminated union is a union with a discriminant. It is also called free unions, disjoint unions or variant type.

1.11. How is a type system defined? (Hint: It has two parts).

A type system is defined by type constructors and type checking.

# Midterm – Answers

---

Question 1: Answer the following short questions.

1.12. Please fill in the blank: Haskell is a strongly typed language.

1.13. What are the two basic kinds of expressions in Scheme?

Atoms and lists.

1.14. What is programming in a “purely functional style”?

Programming without side effects.

1.15. What is the evaluation rule for lists in Scheme programming language?

Recursively evaluate the elements.

# Midterm – Answers

**Question 3:** Give a context free grammar for the language  $\{ a^i b^j c^k \mid i=k, i,j,k \geq 0 \}$ ?  
Show the parse tree for the input string “aabbcc”?

$\langle S \rangle \rightarrow \langle B \rangle \mid a \langle S \rangle c$

$\langle B \rangle \rightarrow \langle B \rangle b \mid \epsilon$

$S \Rightarrow a S c$

$\Rightarrow a a S c c$

$\Rightarrow a a B b c c$

$\Rightarrow a a B b b c c$

$\Rightarrow a a B b b b c c$

$\Rightarrow a a b b b c c$

# Midterm – Answers

**Question 4:** Show that the following CFG is ambiguous? Note that  $\langle . \rangle$  represents a non-terminal symbol while all others represents terminals. Use the input string “a+a\*a” in your answer.

$$\langle S \rangle \rightarrow \langle S \rangle + \langle S \rangle \mid \langle S \rangle * \langle S \rangle \mid ( \langle S \rangle ) \mid a$$

We show two distinct derivations for the input string. Either by direct derivation or parse tree.

$$\begin{aligned} S &\Rightarrow S + S \\ &\Rightarrow a + S \\ &\Rightarrow a + S * S \\ &\Rightarrow a + a * S \\ &\Rightarrow a + a * a \end{aligned}$$

$$\begin{aligned} S &\Rightarrow S * S \\ &\Rightarrow S + S * S \\ &\Rightarrow a + S * S \\ &\Rightarrow a + a * S \\ &\Rightarrow a + a * a \end{aligned}$$



# Midterm – Answers

**Question 5:** Consider the following code segment on the left below. Answer the questions on the right.

```
int n = 1;
print_plus_n(int x) {
    cout << x + n;
}
increment_n() {
    n = n + 2;
    print_plus_n(n);
}
test() {
    int n;
    n = 200;
    print_plus_n(7);
    n = 50;
    increment_n();
    cout << n;
}
```

What would test() print if static scoping is used?

8 6 50

What would it print if dynamic scoping is used?

207 104 52

# Midterm – Answers

**Question 6:** What is the result of the following in Scheme & Haskell?

<code>((lambda (x) (* x 50)) 1)</code>	50
<code>(\x -&gt; x + 50) 1</code>	51
<code>map (+ 12) [1,2,3,4,5]</code>	<code>[13,14,15,1,17]</code>

Write Haskell functions/expressions for the following:

A function that filters a given list returning all the elements bigger than 10 and smaller than 50. You can use any existing list functions you know in Prelude.hs.

`myfilter :: [Int] -> [Int]`

`myfilter x = filter (\y -> y>10 && y<50) x`

Write the above as a lambda expression.

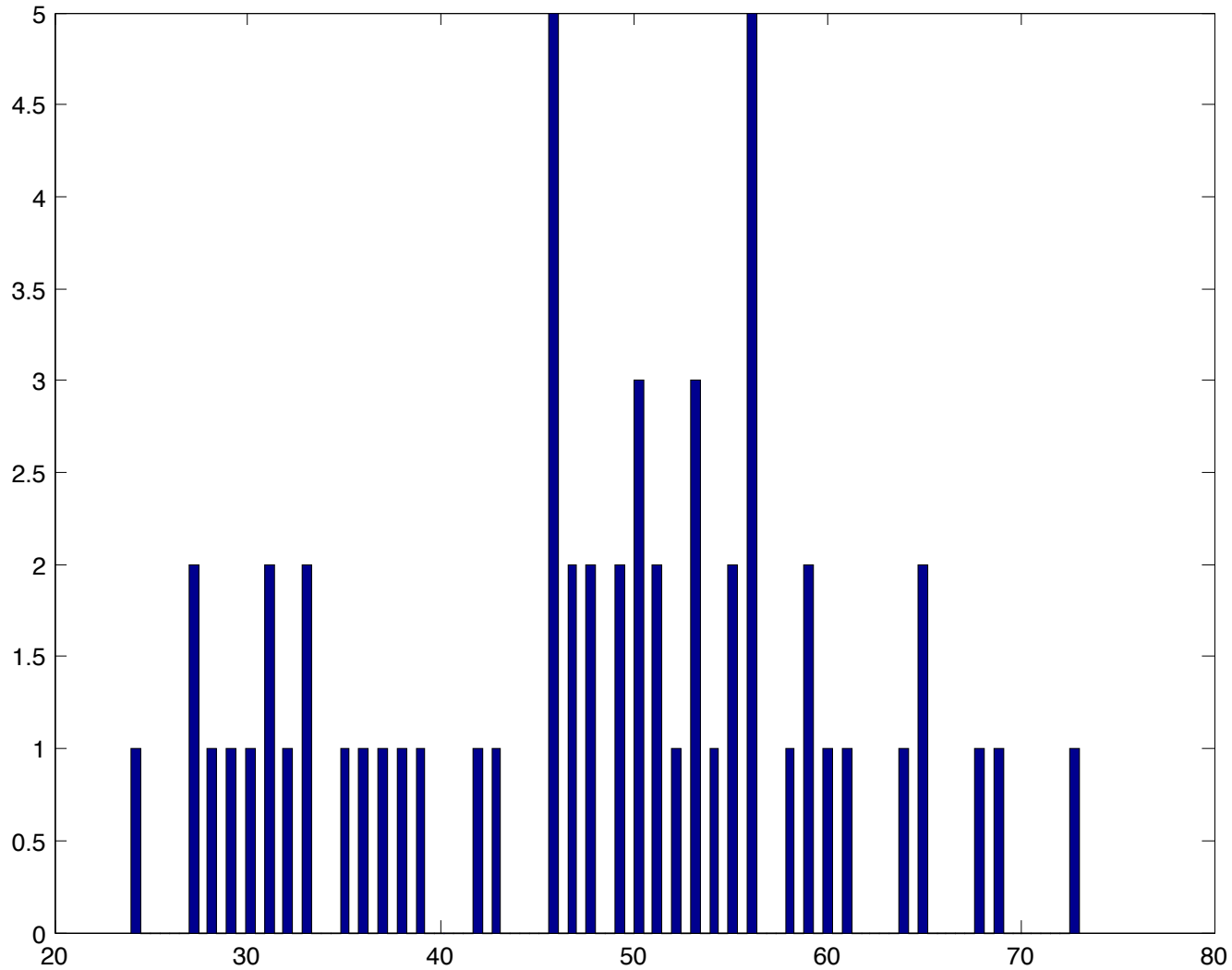
`\x -> filter (\y -> y>10 && y<50) x`

Using the map function, write an expression that subtracts 10 from all the entries of the given list [11,12,13].

`map (\x -> x-10) [11,12,13]`

# Midterm Results

---



# SWI-Prolog

---

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

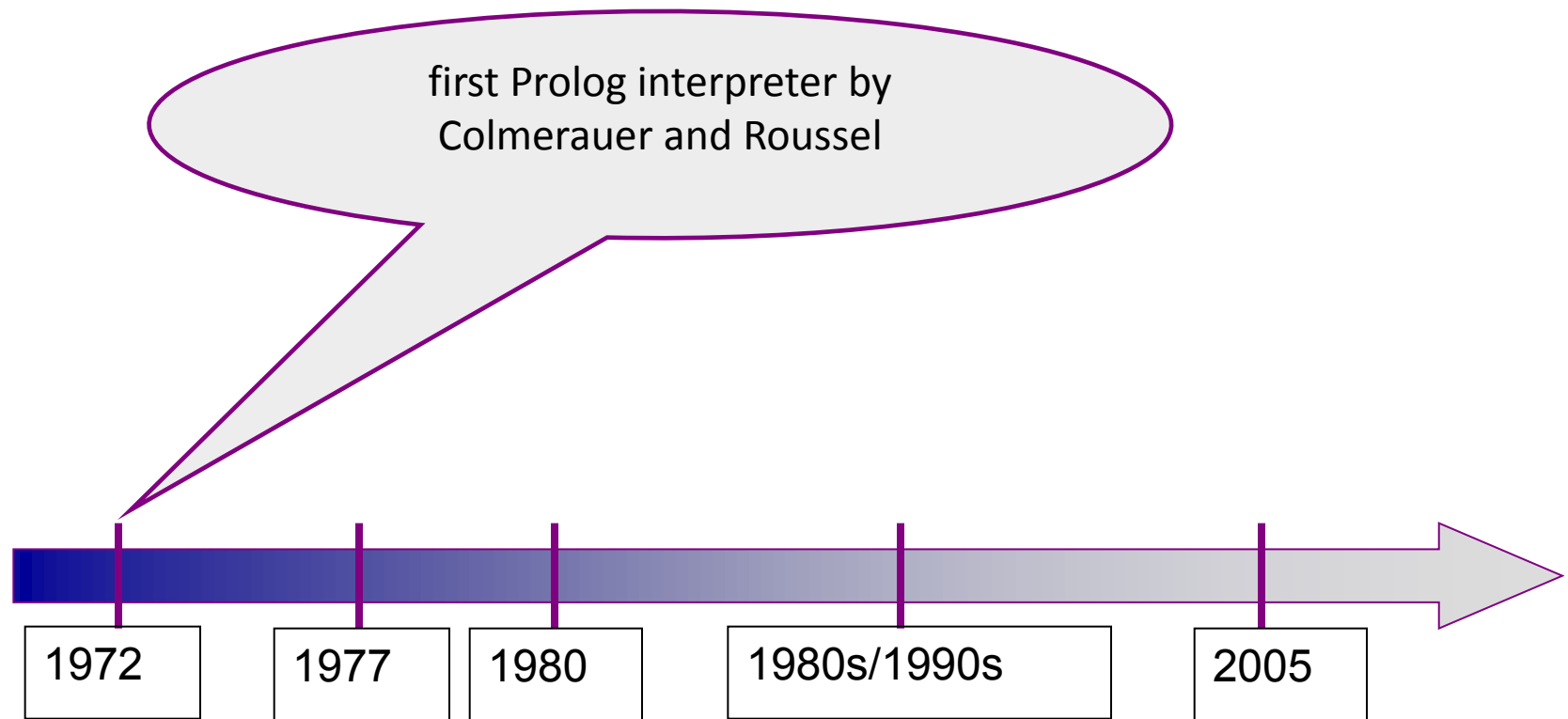
# Prolog

---

- 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”

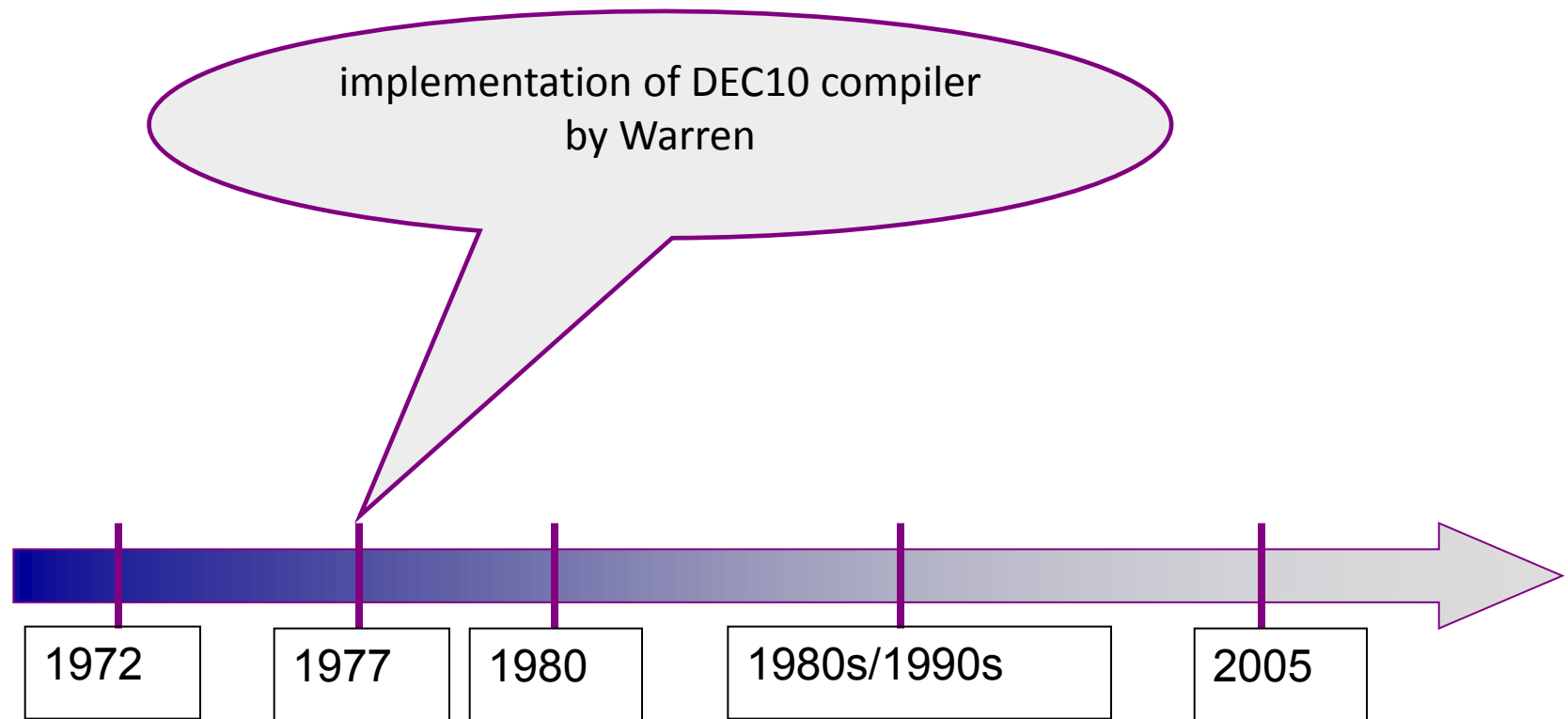
# History of Prolog

---



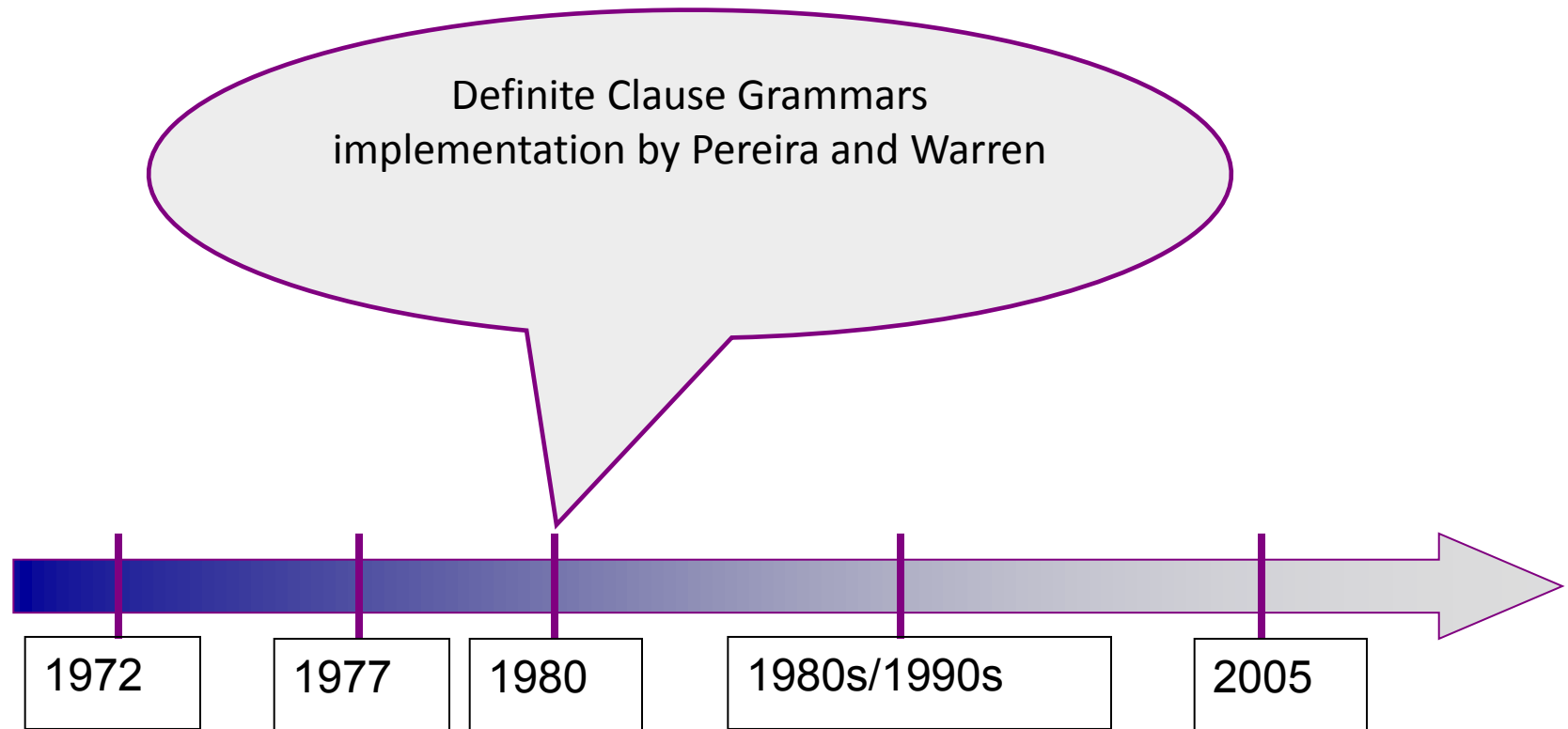
# History of Prolog

---



# History of Prolog

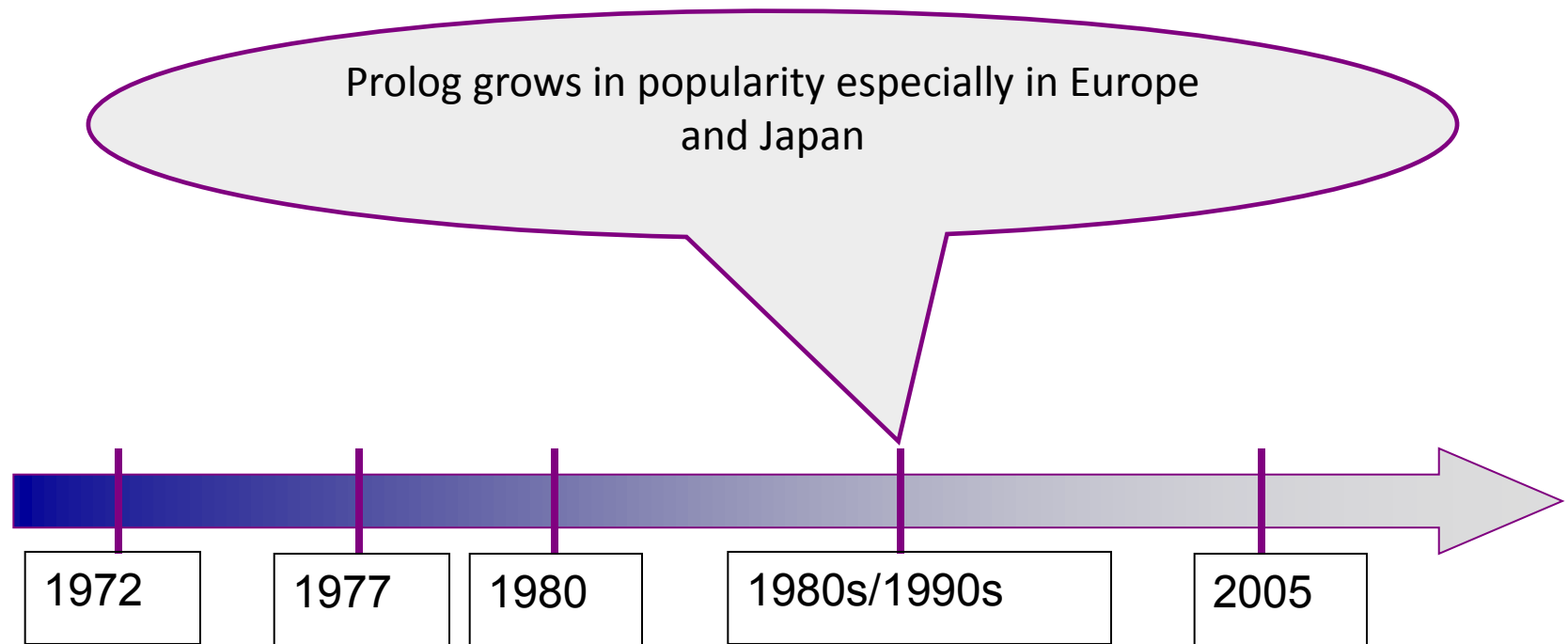
---





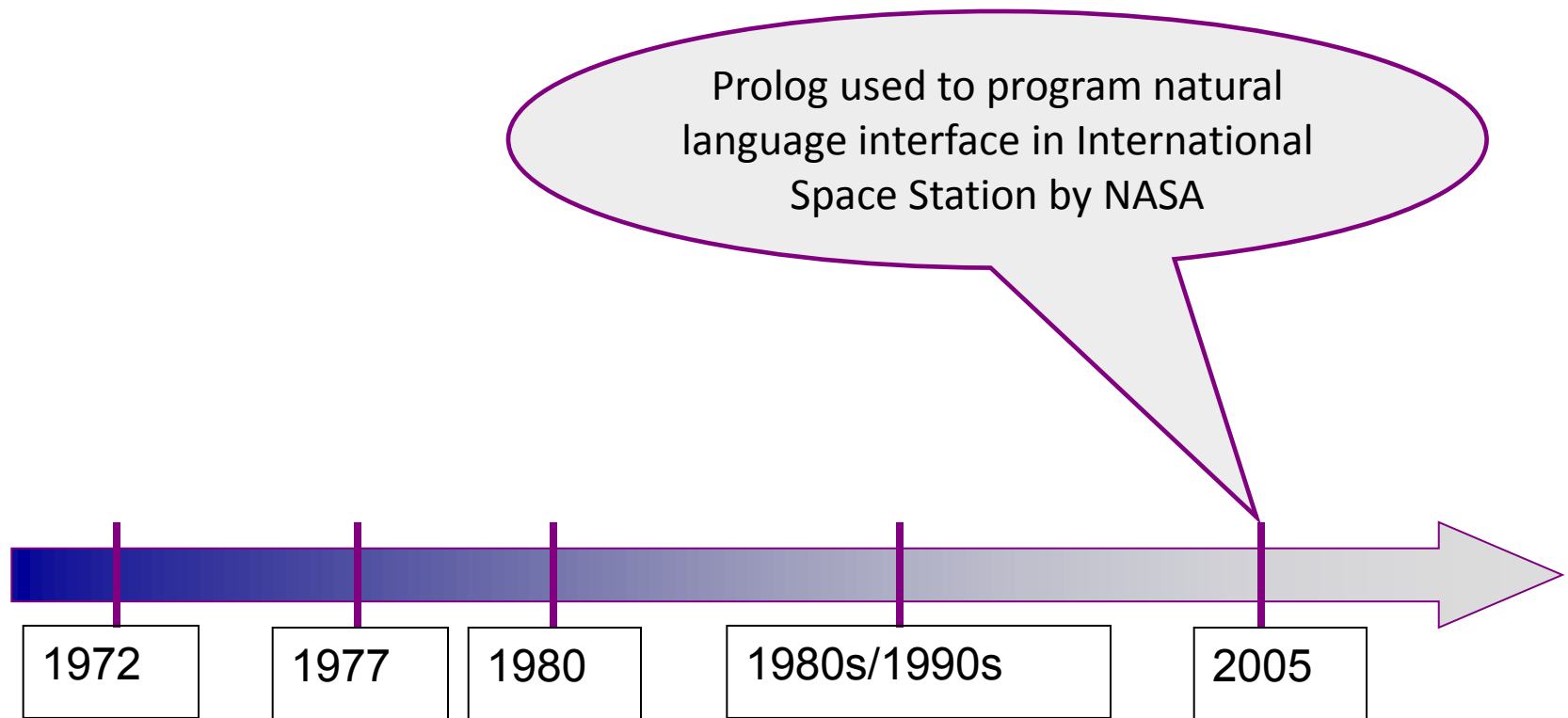
# History of Prolog

---



# History of Prolog

---



# Logic Programming

---

- Program

Axioms (facts): true statements

- Input to Program

query (goal): statement true (theorems) or false?

- Thus

Logic programming systems = deductive databases

datalog

# 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)

# 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?

# 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

$$\forall S \forall x (x \in S \vee x \notin S)$$

---

# 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.,  $x$ ,  $y$ ,  $\text{foo}$
- **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  $f(x_1, \dots, x_n)$  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.

# 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" | ... ;
```

# 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.,  $(\forall x) \text{dolphin}(x) \rightarrow \text{mammal}(x)$

- **Existential quantification**

- $(\exists x)P(x)$  means that  $P$  holds for **some** value of  $x$  in the domain associated with that variable
- E.g.,  $(\exists x) \text{mammal}(x) \wedge \text{lays-eggs}(x)$
- Permits one to make a statement about some object without naming it

# Translating English to FOL

**Every gardener likes the sun.**

$$\forall x \text{ gardener}(x) \rightarrow \text{likes}(x, \text{Sun})$$

**You can fool some of the people all of the time.**

$$\exists x \forall t \text{ person}(x) \wedge \text{time}(t) \rightarrow \text{can-fool}(x, t)$$

**You can fool all of the people some of the time.**

$$\forall x \exists t (\text{person}(x) \rightarrow \text{time}(t) \wedge \text{can-fool}(x, t))$$

$$\forall x (\text{person}(x) \rightarrow \exists t (\text{time}(t) \wedge \text{can-fool}(x, t)))$$

← Equivalent

**All purple mushrooms are poisonous.**

$$\forall x (\text{mushroom}(x) \wedge \text{purple}(x)) \rightarrow \text{poisonous}(x)$$

**No purple mushroom is poisonous.**

$$\neg \exists x \text{ purple}(x) \wedge \text{mushroom}(x) \wedge \text{poisonous}(x)$$

$$\forall x (\text{mushroom}(x) \wedge \text{purple}(x)) \rightarrow \neg \text{poisonous}(x)$$

← Equivalent

**There are exactly two purple mushrooms.**

$$\begin{aligned} \exists x \exists y \text{ mushroom}(x) \wedge \text{purple}(x) \wedge \text{mushroom}(y) \wedge \text{purple}(y) \wedge \neg(x=y) \wedge \forall z \\ (\text{mushroom}(z) \wedge \text{purple}(z)) \rightarrow ((x=z) \vee (y=z)) \end{aligned}$$

**Clinton is not tall.**

$$\neg \text{tall}(\text{Clinton})$$

# First-Order Predicate Calculus: Example

---

- `natural(0)`  
 $\forall X, \text{natural}(X) \rightarrow \text{natural}(\text{successor}(x))$
- $\forall X \text{ and } Y, \text{parent}(X,Y) \rightarrow \text{ancestor}(X,Y).$   
 $\forall A, B, \text{ and } C, \text{ancestor}(A,B) \text{ and } \text{ancestor}(B,C) \rightarrow \text{ancestor}(A,C).$   
 $\forall X \text{ and } Y, \text{mother}(X,Y) \rightarrow \text{parent}(X,Y).$   
 $\forall X \text{ and } Y, \text{father}(X,Y) \rightarrow \text{parent}(X,Y).$   
`father(bill,jill).`  
`mother(jill,sam).`  
`father(bob,sam).`
- `factorial(0,1).`  
 $\forall N \text{ and } M, \text{factorial}(N-1,M) \rightarrow \text{factorial}(N,N*M).$

# 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 ( $\rightarrow$ ): `a $\rightarrow$ b` (`b or not a`)  
equivalence ( $\leftrightarrow$ ): `a $\leftrightarrow$ b` (`a $\rightarrow$ b and b $\rightarrow$ a`)



# First-Order Predicate Calculus: Statements

---

- *Quantifiers*

*universal quantifier* "for all"  $\forall$

*existential quantifier* "there exists"  $\exists$

*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**:

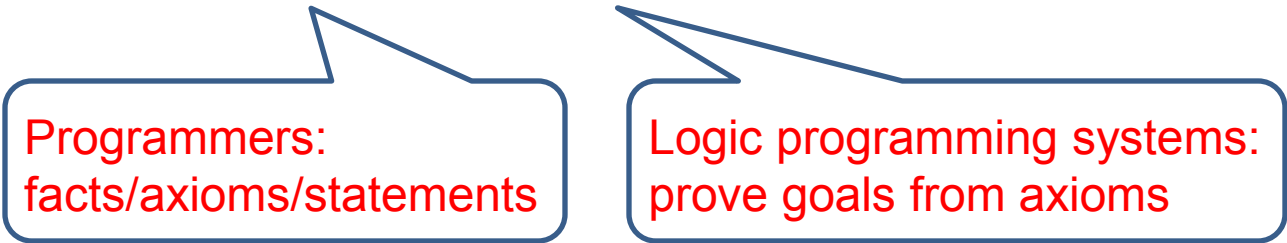
- Contain constants, variables, and functions.

- Cannot have predicates, qualifiers, or connectives.

# 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

# 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 $Hu \rightarrow Ho$	Premise	“If it is humid, it is hot”
3 Ho	Modus Ponens(1,2)	“It is hot”
4 $(Ho \wedge Hu) \rightarrow R$	Premise	“If it’s hot & humid, it’s raining”
5 $Ho \wedge Hu$	And Introduction(1,3)	“It is hot and humid”
6 R	Modus Ponens(4,5)	“It is raining”

# Horn Clause

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

$b \leftarrow a_1 \text{ and } a_2 \text{ and } a_3 \dots \text{ and } a_n.$

head

body

no “or” and no quantifier

$b \leftarrow .$  fact

$\leftarrow b.$  query

- Variables in head: universally quantified  
Variables in body only: existentially quantified
- Need “or” in head? Multiple clauses

# Horn Clauses: Example

---

- First-Order Logic:

`natural(0) .`

`$\forall X, \text{natural}(X) \rightarrow \text{natural}(\text{successor}(x)) .$`



- Horn Clause:

`natural(0) .`

`$\text{natural}(\text{successor}(x)) \leftarrow \text{natural}(X) .$`

# Horn Clauses: Example

---

- First-Order Logic:

`factorial(0,1).`

$\forall N \text{ and } \forall M, \text{ factorial}(N-1, M) \rightarrow \text{factorial}(N, N * M).$



- Horn Clause:

`factorial(0,1).`

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

# 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) .
```

# Horn Clauses: Example

---

- First-Order Logic:

$\forall X, \text{mammal}(X) \rightarrow \text{legs}(X, 2) \text{ or } \text{legs}(X, 4).$



- Horn Clause:

$\text{legs}(X, 4) \leftarrow \text{mammal}(X) \text{ and not } \text{legs}(X, 2).$

$\text{legs}(X, 2) \leftarrow \text{mammal}(X) \text{ and not } \text{legs}(X, 4).$



# Prolog syntax

---

- `:-` for  $\leftarrow$   
`,` 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) .
```

# 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>
```

# Next Class

---

- Object Oriented programming