Introduction to PROLOG

PROgramming in LOGic

Dr. Ganesh Bhutkar
VIT, Pune INDIA
ganesh.bhutkar@vit.edu

TY BTech Comp - 2020-21

What is Prolog?

- Programming in Logic
  - Declarative language
    - Focus on *describing* the problem and desired solution
    - Use a subset of First Order Logic (Horn clauses)
  - Characteristics
    - Knowledge is represented by facts and rules
    - The system applies *logical deduction* to find answers for the problem
    - Depth-first search engine

Prolog Programs

- Terms
  - The data objects of the language
  - Either constant (atom or number), variable or compound term
- Facts and Rules
  - Predicates: "Generalized functions", allowing multiple return values, used in multiple directions
  - Facts: Predicates assumed to be true
  - Rules: `P(..) :- P`$_1$`(..),P`$_2$`(..),…,P`$_n$`(..).`

Prolog Terms

- Constant
  - Denotes a known entity/object/thing
  - Includes **numbers** (integers, floats), **atoms**
  - Must *begin with a lowercase letter*
  - E.g. `john` (atom), `123` (integer), `‘hello world’` (atom), `-0.05e10` (floating point), `[]` (atom)

Prolog Term - Atom

- An atom is identified by its name.
- No special syntax. However,
  - Atoms containing spaces or certain other special characters must be surrounded by single quotes.
  - Atoms beginning with a capital letter must also be quoted, to distinguish them from variables.
- Atoms can be constructed in 3 ways:
  - Strings of letters, digits & the underscore, starting with a lower-case letter: anna x_25 nil
  - String of special characters: <----> ::== .:.
  - Strings of characters enclosed in single quotes: ‘Tom’ ‘x_>:’ 'some atom’

Prolog Terms

- The empty list, written [], is also an atom.
- Atoms are definite elementary objects, and correspond to proper nouns in natural language.
- The name of an atom has NO inherent meaning to the computer, but is just a symbol.

- Numbers:
  - Reals: 3.14 -0.573
  - Integers: 23 5753 -42

Prolog Terms

- Variable
  - Represents an unknown object
  - Corresponds to improper nouns
  - A string consisting of letters, numbers and underscore characters
  - Must *begin with an uppercase letter or an underscore*
  - E.g. `Name`, `_type`, `X`, `Value`, `_3`, `_Result`, `_`
  - '`_`' is the anonymous variable. It means 'don't care'.

Prolog Terms

- Variable
  - Scope restricted to one clause. i.e. variables with the same name in different clauses are unrelated.
  - The anonymous variable _ is special
    - `getsEaten(X) :- eats(_,X).`
    - Multiple occurrences of _ within the SAME clause are UNRELATED
  - The variables in Prolog are VERY different from those in imperative languages (eg. C)
  - Variable are not assigned but instantiated.

Prolog Terms

- Compound Term
  - Consists of
    - A function symbol called **functor**
    - Term(s) in parentheses separated by commas
  - Can represent a structured data like tree, list
  - E.g. tree(tree(a,nil),tree(b,X))
  - Special cases of compound terms:
    - **Lists** are defined inductively:
      - The atom [] is a list.
      - A compound term with functor . (dot) and arity 2, whose second argument is a list, is itself a list.

Prolog Terms

- Special syntax for denoting lists: .(`A, B`) is equivalent to `[A|B]`.
- `.(a,.(b,.(c,[])))` same as `[a|[b,c]]` same as `[a,b,c]`

- **Strings**: A sequence of characters surrounded by quotes is equivalent to a list of (numeric) character codes.
  - String is just a list of ASCII codes.
  - `“Humpty”` same as `[72,117,109,112,116,121]`

Prolog Programs

- Programming in Prolog is very different from programming in a procedural language.
- Prolog programs describe relations, defined by means of **clauses: facts and rules**.
- In Prolog, you supply a database of **facts and rules**; you can then perform queries on the database.
- The basic unit of Prolog is the **predicate** entering into the database.
- Run the program by making some queries.
- The system tries to deduce the **query** from the facts and rules.
- The answer is either true or false and the instantiated value of variables.
- Sometimes, it is the **side-effects** that are wanted, e.g. Printing something on the screen.

Prolog Facts and Rules

- Predicate
  - Defines a relation among elements or properties of elements
  - Consists of a predicate name (head), term(s) in parentheses separated by commas
  - e.g. `mother(susan, ann)`, `factorial(4,24)`
  - A predicate is either true or false
  - No inherent meaning for the computer, just relations between objects

Prolog Facts and Rules

- Predicate
  - Can be regarded as generalized function.
  - E.g. `append(X,[a,b],Z)` may mean `X` appended to `[a,b]` gives `Z`.
    - Can treat `X` and `[a,b]` as input, `Z` as an output
    - Can also treat `[a,b]` and `Z` as input, `X` as output, which is asking what list appened to `[a,b]` gives `Z`
  - Can “return” multiple values easily, and the “function” can be used in different directions.

Prolog Facts and Rules

- Fact
  - Represents what is assumed to be true
  - Consists of a predicate ended with a full stop
  - E.g.
    - `colour(red).`
    - `company(theIBM).`
    - `course(csc3230,'Fundamentals of AI').`
    - `equal(X,X).`
    - `non_leaf(tree(_,_)).`
  - Similar to what is stored in a relational database

Prolog Facts and Rules

- Rule
  - Represents a conditional assertion
  - The head is a predicate, the body is one or more predicates – Horn clause
  - Tells how does the truth of a predicate depends on the truth of other predicates
  - Can be regarded as the body of a function
    - `light(on) :- switch(on).`
    - `father(X,Y) :- parent(X,Y), male(X).`
    - `between(X,Y,Z):- before(X,Y),before(Y,Z).`

Prolog Facts and Rules

- Rule

$$\underbrace{H(\ldots)}_{\text{Head Goal (Conclusion)}} \text{ :- } \underbrace{B_1(\ldots),\ B_2(\ldots),\ \ldots,\ B_n(\ldots).}_{\text{Body Goals (Conditions)}}$$

  - Meaning : H(...) is true, if $B_1(\ldots)$, $B_2(\ldots)$, ..., $B_n(\ldots)$ are all true.
  - Commas in the body can be read as the logical 'AND'.
  - When there are more than 2 rules with the same head, they have the meaning of logical 'OR'.

Queries

- Ask the program whether a predicate (or conjunction of predicates) is true based on the facts and rules
- Similar to function calls in other languages
- Similar to queries in database

Queries - Examples

- `?- father(tom,john).`
  - Asking whether the atom `tom` is related to the atom `john` by the predicate `father`, either by a fact or through rules
  - May mean: is tom father of john?
- `?- tutor(csc3230,X).`
  - Ask the system to find an `X` such that `csc3230` is related to `X` by the predicate `tutor`
  - May mean: who are the tutors of csc3230?

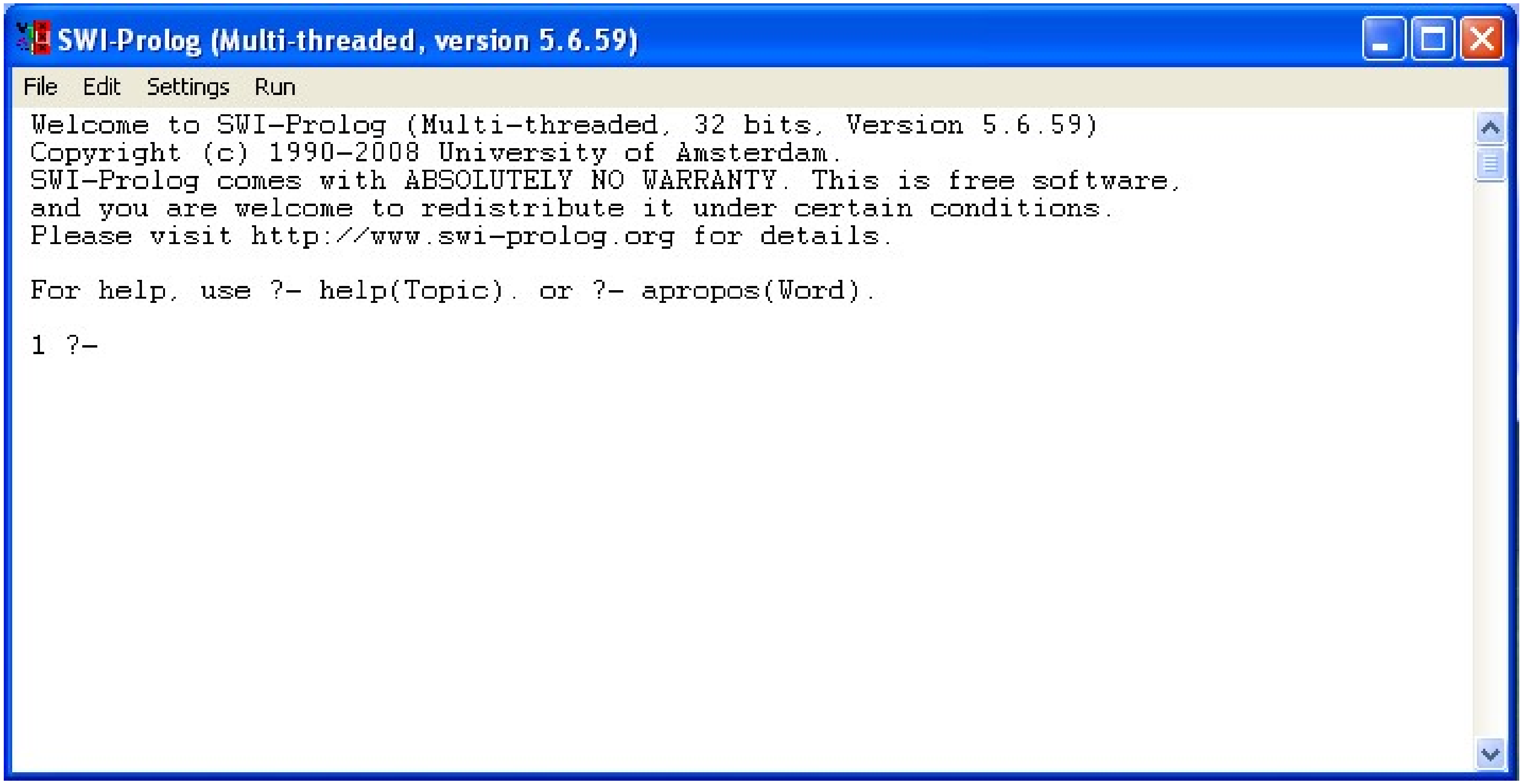
Queries - Examples

- `?-tutor(csc3230,X),supervisor(Y,X).`
  - Find `X` and `Y` such that both predicates are true
  - May mean: who is the supervisor of the tutor of csc3230?
- `?-append([a,b],[c],Z).`
  - May mean: what is `[a,b]` appended to `[c]`?
- `?-takes(X,csc3230),age(X,A),A>20.`
  - May mean: who takes csc3230 and is above 20?

What is SWI-Prolog?

- SWI-Prolog offers a comprehensive **Free Software Prolog environment**.
- Started in 1987 and has been driven by the needs for **real-world applications**.
- These days, SWI-Prolog is widely used in research and education as well as for commercial applications

Launch SWI-Prolog



Queries - Examples

- assert(before(a,b)).
- assert(before(b,c)).
- assert(before(a,d)).
- assert(before(b,d)).

- ?- before(a,b).
- true
- ?- before(b,a).
- false
- ?- before(a,X).
- X = b /* press ; */
- X = d /* press ; */

Queries - Examples

```
before(a,b).
before(b,c).
before(a,d).
before(b,d).
```

- Save as a .pl file

- ?- before(X,d).
- X = a /* press ; */
- X = b /* press ; */

- ?- before(a,c).
- false /* !? */

Notes: If Prolog answers “no”, it doesn’t mean that answer is definitely false. It means that the system cannot deduce that it is true given its database – Closed World Assumption

Running Prolog

- To load a prolog program
  - ?- [filename]
  - Or simply double-click the file
- Type "help" to get online help.

Activity

- Write the following in Prolog
- Facts:
  - Bear eats honey
  - Bear eats salmon
  - Rat eats salmon
  - Salmon eats worm
- Queries:
  - Who eats salmon?
  - Who eats both honey and salmon?

Activity

- eats(X,salmon) , eats(X,honey).
- Rules:
  - For all X and Y, X is in Y's food chain if Y eats X
  - `food_chain(X,Y) :- eats(Y,X).`
  - For all X and Y: X is in Y's food chain if Y eats X, Or, Y eats some Z and X is in Z's foodchain.
  - `food_chain(X,Y) :- eats(Y,Z), food_chain(X,Z).`
- Queries:
  - What is in rat's food chain?
  - Whose food chain contains worm?

Queries - Unification

- Try to match two predicates or terms by suitably instantiating variables
- Rules

Term	Another Term	Condition
Uninstantiated variable X	Any term	The term does not contain X
Atom or Number	Atom or Number	They are equal
Compound Term	Compound Term	Same functors, same arity, and the corresponding terms unify

Queries – Unification Examples

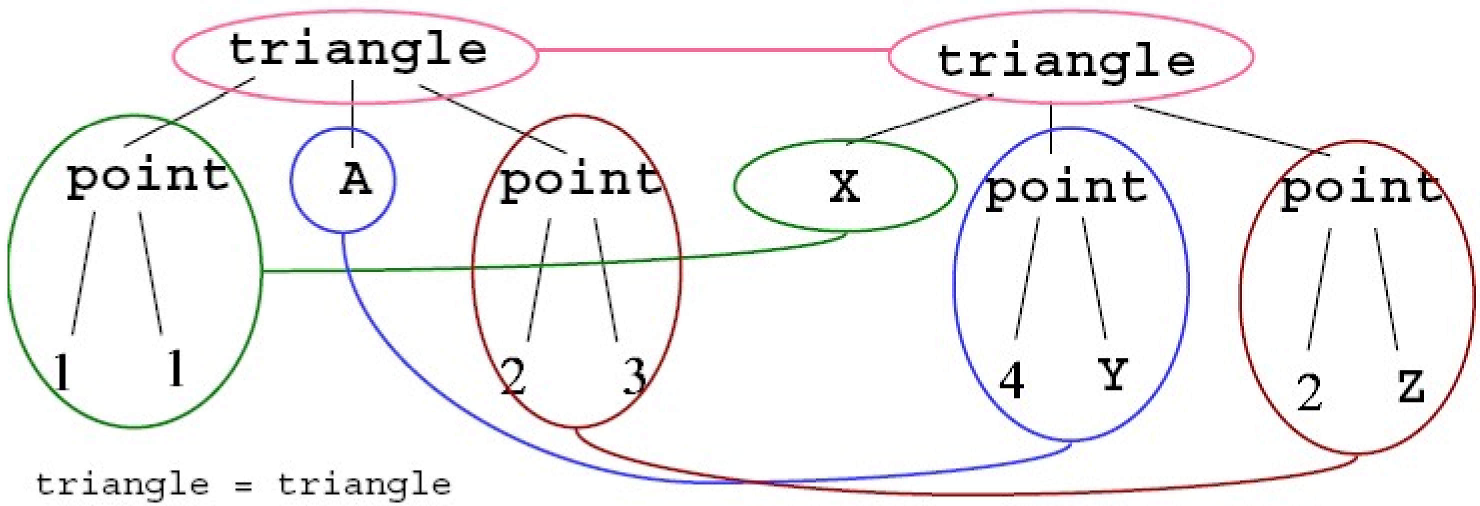
1st Term	2nd Term	Unified?	Variable Instantiation
abc	xyz	no	
X	Y	yes	X→Y
Z	123	yes	Z→123
f(A)	f(234)	yes	A→234
f(A)	f(1,B)	no	
f(g(A),A)	f(B,peter)	yes	A→peter, B→g(peter)
t(L,t(X,b))	t(t(c,d),t([],b))	yes	L→t(c,d), X→[]
[H|T]	[a,b,c,d]	yes	H→a, T→[b,c,d]

Geometric Example

- Use structures to represent simple geometric shapes.
  - point - two numbers representing X and Y coordinates.
  - seg - a line defined by two points.
  - triangle - defined by three points.
    - `point(1,1).`
    - `seg( point(1,1), point(2,3) ).`
    - `triangle( point(4,2), point(6,4), point(7,1) ).`

Geometric Example

- triangle(point(1,1), A, point(2,3)) = triangle(X, point(4,Y), point(2,Z)).




```
triangle = triangle
point(1,1) = X
A = point(4,Y)
point(2,3) = point(2,Z)
```

Arithmetics

- Predefined operators for basic arithmetic:
  - +, -, *, /, mod
- If not explicitly requested, the operators are just like any other relation
- Example:

```
X = 1 + 2.
X=1+2
```

Arithmetics

- The predefined operator '*is*' forces evaluation.

  ```
  ?- X is 1 + 2.
  X=3
  ```

- A **is B (A and B here can be anything) means**
  - Evaluate B to a number and perform matching of the result with A
- The comparison operators also force evaluation.

  ```
  ?- 145 * 34 > 100.
  true
  ```

Comparison Operators

- X > Y X is greater than Y.
- X < Y X is less than Y.
- X >= Y X is greater than or equal to Y.
- X =< Y X is less than or equal to Y.
- X =:= Y the values of X and Y are equal.
- X =\= Y the values of X and Y are not equal.

= and =:=

- X = Y causes the matching of X and Y and possibly instantiation of variables.
- X =:= Y causes an arithmetic evaluation of X and Y, and cannot cause any instantiation of variables.
  - `1 + 2 =:= 2 + 1.`
  - **`true`**
  - `1 + 2 = 2 + 1.`
  - **`false`**

Activity: The Greatest Common Devisor

- Write a Prolog program that calculates the GCD of two integers.
- Given X and Y, the gcd D can be found by:
  - If X and Y are equal then D is equal to X.
  - If X < Y then D is equal to the gcd of X and (Y-X).
  - If Y < X then do the same as in (2) with X and Y interchanged.

GCD

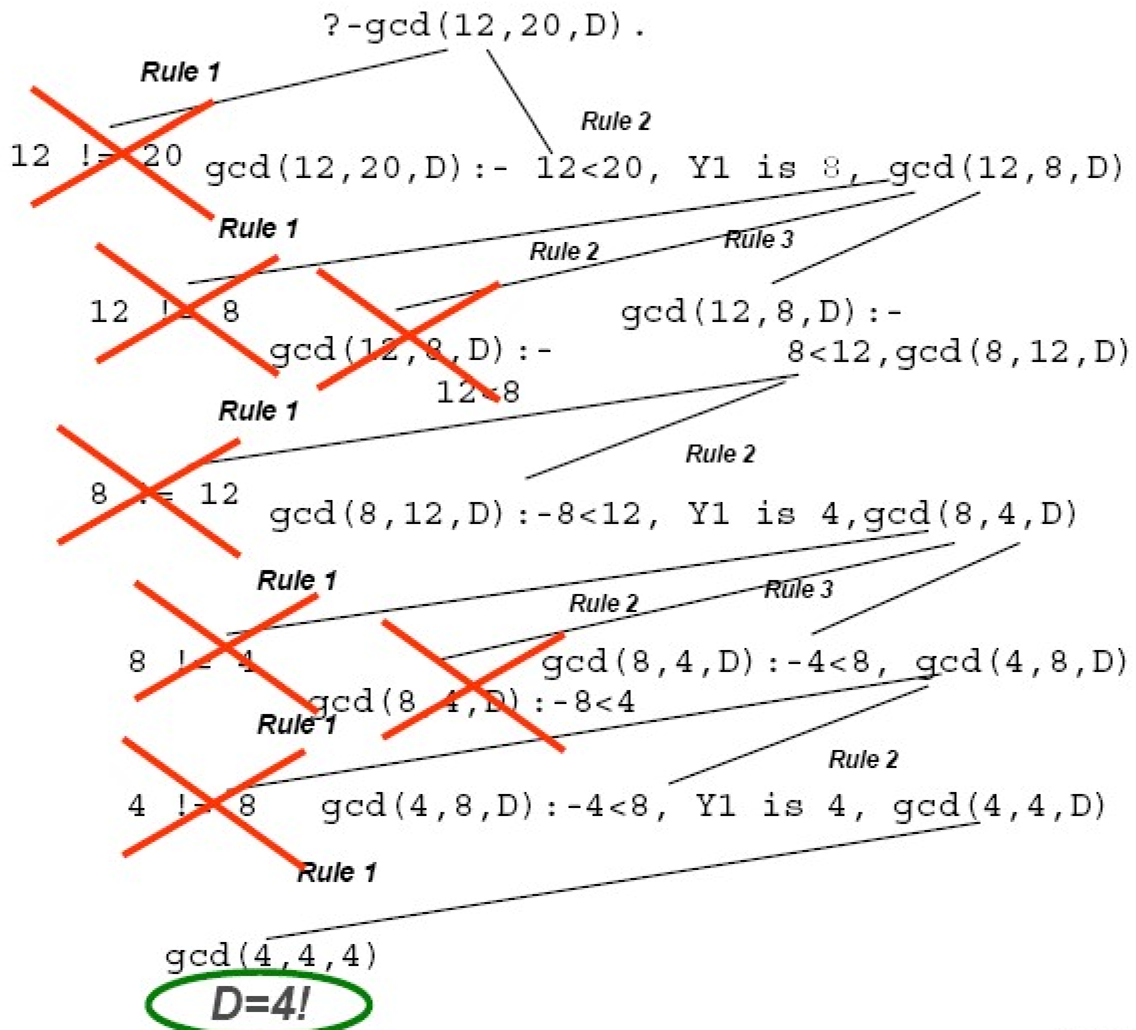
```
gcd(X,X,X).
gcd(X,Y,D) :-
    X < Y,
    Y1 is Y - X,
    gcd(X,Y1,D).
gcd(X,Y,D) :-
    Y < X,
    gcd(Y,X,D).
```

How does it work?

```
gcd(X,X,X).

gcd(X,Y,D) :-
    X < Y,
    Y1 is Y - X,
    gcd(X,Y1,D).

gcd(X,Y,D) :-
    Y < X,
    gcd(Y,X,D).
```



Queries - Backtracking

- When asked `P`$_1$`(..),P`$_2$`(..),…,P`$_n$`(..).`
- Most Prolog will attempt the following
  - Unify P_1 with a fact or rule, instantiate variables if needed
  - If P_1 unifies with more than one fact or rule, the first one is chosen
  - If succeed, do the same for P_2, and so on from left to right
  - If all predicates succeed, the whole goal succeeds
  - If anyone fails, say P_i, Prolog backtracks, and try an alternative of P_{i-1}
  - The predicates are tried in a Depth-First manner
  - After a successful query, if user presss '`;`', backtrack and try alternatives

Queries - Backtracking

- before(a,b). << Not match
- before(b,c). << Not match
- before(c,d). << Not match
- before(A,C) :- before(A,B), before(B,C).

- ?- before(a,c).

Queries – Backtracking Example

- before(a,b).
- before(b,c).
- before(c,d).
- before(A,C) :- before(A,B), before(B,C). << Unifed, with A→a,C→c

- ?- before(a,c).

before(a,c) :- before(a,B), before(B,c).

⇓

Call : before(a,B). << Put B=b

⇓

yes Exit : before(a,b). << Match Fact 1.

Queries – Backtracking Example

- before(a,b).
- before(b,c).
- before(c,d).
- before(A,C) :- before(A,B), before(B,C). << Unifed, with A→a,C→c

- ?- before(a,c).

before(a,c) :- before(a,B), before(B,c).

⇩

Call : before(b,c). << As B=b

⇩

yes Exit : before(b,c). << Match Fact 2.

Queries – Backtracking Example

- before(a,b).
- before(b,c).
- before(c,d).
- before(A,C) :- before(A,B), before(B,C).

- ?- before(a,c). << succeeds, use the rule with A$\rightarrow$a,B$\rightarrow$b,C$\rightarrow$c

```
before(a,c) :- before(a,b), before(b,c).
```

yes yes

yes

- See “AI through Prolog” ch 3 for a more elaborate explanation

References

- Artificial Intelligence through Prolog by Neil C. Rowe
    - http://www.cs.nps.navy.mil/people/faculty/rowe/book/book.html
- http://en.wikipedia.org/wiki/Prolog
- SICStus Prolog (Summary) prepared by Dr. Jimmy Lee
    - http://appsrv.cse.cuhk.edu.hk/~csc3230/reference/prolog_primer.ps