



Introduction to Logic Programming (Prolog)

Tutorial for

Programming Languages Laboratory (CS 431)

Samit Bhattacharya

Indian Institute of Technology Guwahati

Programming Paradigms

➤ Broadly two types

- Imperative
 - Declarative
-

Imperative Programming

- We are familiar with this paradigm (procedural/OOP)
 - Main concern – ‘how’ to do things

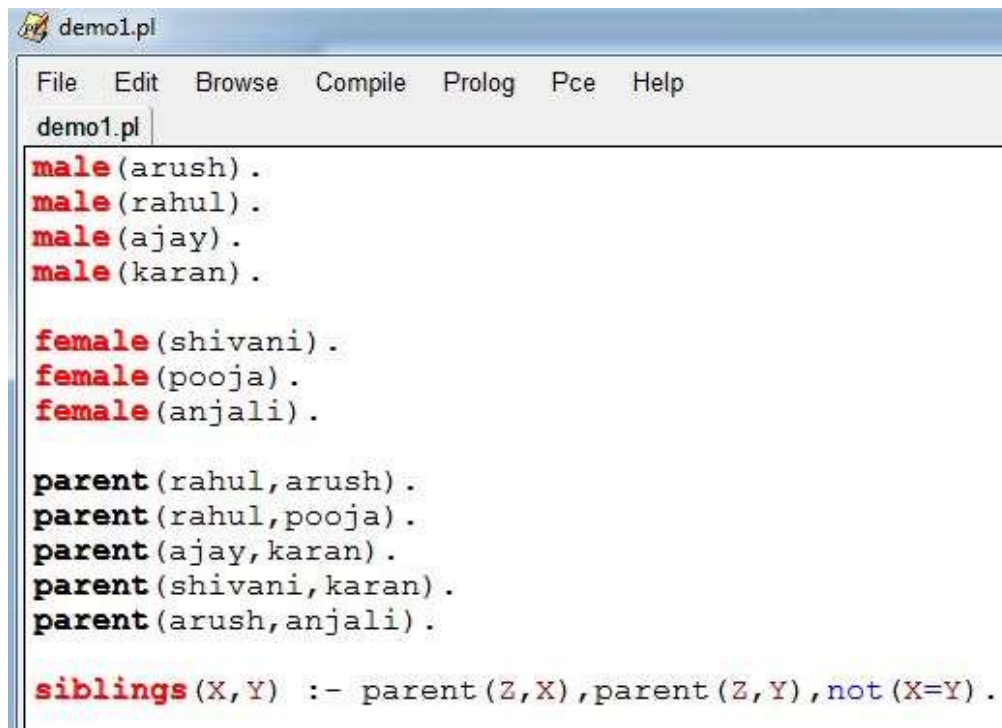
Declarative Paradigm

➤ Programming is done with 'expressions' or 'declarations' rather than statements

- Main concern – 'what' to do, not 'how'

Q: Who takes care of the "How" part?

Example



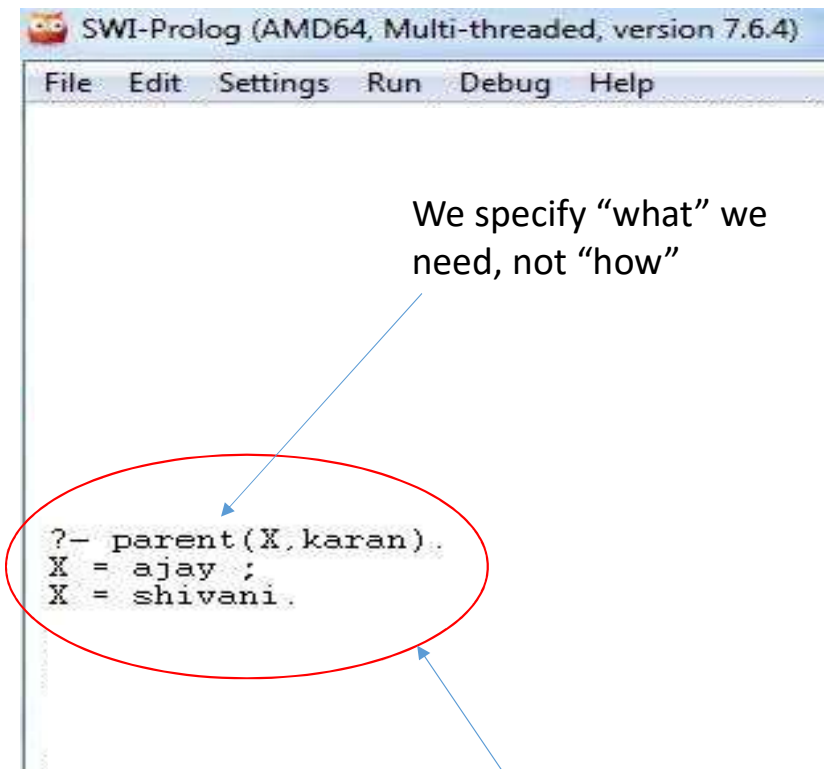
```
demo1.pl
File Edit Browse Compile Prolog Pce Help
demo1.pl
male(arush).
male(rahul).
male(ajay).
male(karan).

female(shivani).
female(pooja).
female(anjali).

parent(rahul,arush).
parent(rahul,pooja).
parent(ajay,karan).
parent(shivani,karan).
parent(arush,anjali).

siblings(X,Y) :- parent(Z,X),parent(Z,Y),not(X=Y).
```

Program



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help

?- parent(X,karan).
X = ajay ;
X = shivani.
```

We specify “what” we need, not “how”

Output

Declarative Paradigm

- Logic programming – going to discuss
 - Functional programming – next topic
-

LP - Basic

- Logic programming – centered around the idea of “formal logic”
 - Propositional logic
 - First order predicate logic
 - Higher order logic

Propositional Logic

- Simplest of all

Proposition (premise): If it is raining it is cloudy

Proposition (premise): It is raining

Proposition (conclusion): It is cloudy

- Conclusion obtained by applying *modus ponens* (inference rules) on the premises ($P \rightarrow Q$; P is asserted to be true, so therefore Q must be true) - uses propositional calculus

First Order Predicate Logic (FOPL)

- Allows quantification with variables (variable values can range over a *domain of discourse*)

PL: Socrates is a man

FOPL: X is a man

- We are concerned about the deductive system using FOPL formulae (well-formed formulae)
 - ✓ Inference using resolution refutation

FOPL

All mangoes are fruits

Alphonso is a mango

Therefore, Alphonso is fruit

More generally, in terms of wffs using FOPL notation

$\forall x, P(x) \Rightarrow Q(x)$

$P(a)$

Therefore, $Q(a)$

FOPL – Inferencing (RR)

- We can recast the inferencing process as follows to establish the same using *resolution refutation*

$\neg P(x) \vee Q(x)$

$P(a)$

Therefore, $Q(a)$

Requires conversion to CNF (Conjunctive Normal Form: “or” relation)

$P(x) \Rightarrow Q(x) : \text{CNF } \neg P(x) \vee Q(x)$

$\neg(P(x) \wedge Q(x)) : \text{CNF } \neg P(x) \vee \neg Q(x)$

You can check with Truth Tables

FOPL – Inferencing (RR)

- Rules

- ✓ Find two *clauses* containing the same *predicate*, where it is negated in one clause but not in the other.
- ✓ Perform a *unification* on the two predicates.
- ✓ If any *unbound* variables which were bound in the unified predicates also occur in other predicates in the two clauses, replace them with their bound values (terms) there as well.
- ✓ Discard the unified predicates, and combine the remaining ones from the two clauses into a new clause, also joined by the "V" operator.

FOPL – Inferencing (RR)

- Apply rules to the example

- ✓ Predicate P in the first clause in negated form and non-negated in the second clause; X is unbound variable and a is a bound value

- ✓ **Unification** on the two predicates resulting in *X substituted by a*

- ✓ Discard the unified predicate

- ✓ Apply substitution on the remaining clause (Q(X)) results in Q(a) [the conclusion] – hence proved

$\neg P(x) \vee Q(x)$
 $P(a)$
Therefore, $Q(a)$

Note

- There are higher order logic also (need not bother about those here)
- We shall do some assignments in Prolog, which works on the idea of FOPL and resolution refutation

LP Perspectives

- There are many (overlapping) perspectives on logic programming
 - ✓ Computations as deduction
 - ✓ Theorem proving
 - ✓ Non-procedural programming
 - ✓ Algorithms minus control
 - ✓ A very high level programming language
 - ✓ A procedural interpretation of declarative specifications
-

Computation as Deduction

- Logic programming offers a slightly different paradigm for computation

COMPUTATION IS LOGICAL DEDUCTION

- It uses the language of logic to express data and programs

For all X and Y , X is the father of Y if

X is a parent of Y and the gender of X is male.

Theorem Proving

- Logic Programming uses the notion of an *automatic theorem prover* as an interpreter
 - ✓ The theorem prover derives a desired solution from an initial set of axioms
- Note that the proof must be a "constructive" one so that more than a true/false answer can be obtained

E.g. The answer to

exists x such that $x = \text{sqrt}(16)$

should be

$x = 4$ or $x = -4$

rather than

true

Non-procedural Programming

- Logic Programming languages are non-procedural programming languages
 - ✓ One specifies **WHAT** needs to be computed but not **HOW** it is to be done
- That is, one specifies
 - ✓ The set of objects involved in the computation
 - ✓ The relationships which hold between them
 - ✓ The constraints which must hold for the problem to be solved
- And leaves it up to the language interpreter or compiler to decide **HOW** to satisfy the constraints

Algorithms Minus Control

- Nikolas Wirth (architect of Pascal) used the following slogan as the title of a book
 - ✓ Algorithms + Data Structures = Programs
 - Bob Kowalski offers a similar one to express the central theme of logic programming
 - ✓ Algorithms = Logic + Control
 - We can view the LOGIC component as
 - ✓ A specification of the essential logical constraints of a particular problem
 - And CONTROL component as
 - ✓ Advice to an evaluation machine (e.g. an interpreter or compiler) on how to go about satisfying the constraints)
-

A Very High Level Language

- A good programming language should not encumber the programmer with non-essential details
 - The development of programming languages has been toward freeing the programmer of more and more of the details...
 - ✓ ASSEMBLY LANGUAGE: symbolic encoding of data and instructions.
 - ✓ FORTRAN: allocation of variables to memory locations, register saving, etc.
 - ✓ JAVA: Platform specifics
 - ✓
 - Logic Programming Languages are a class of languages which attempt to free us from having to worry about many aspects of explicit control.
-

A Procedural Interpretation of Declarative Specifications

- One can take a logical statement like the following
 - ✓ *For all X and Y , X is the father of Y if X is a parent of Y and the gender of X is male.*
 - Which would be expressed in an LP language as
 - ✓ *`father(X,Y) :- parent(X,Y), gender(X,male).`*
 - And interpret it in two slightly different ways
 - ✓ **Declaratively** - as a statement of the truth conditions which must be true if a father relationship holds.
 - ✓ **Procedurally** - as a description of what to do to establish that a father relationship holds.
-

LP Languages

- Work initiated to deal with representational issues in AI (1960s and 70s)
- Planner (MIT) – earliest language (procedural paradigm)
 - ✓ Gave rise to many languages (e.g. Popler, Conniver, QLISP, Ether)
- Prolog (one of the popular languages)
 - ✓ First system by Alain Colmerauer & Philippe Roussel (1972)
- Prolog gave rise to many new languages
 - ✓ Fril, Gödel, Mercury, Oz, Visual Prolog, λ Prolog ...

SWI-Prolog

- SWI-Prolog is a good, standard Prolog for Windows and Linux
- It's licensed under GPL, therefore free Downloadable from:

<http://www.swi-prolog.org/>

Syllogisms (Logical Reasoning) in Prolog

Syllogism

Socrates is a man.

All men are mortal.

Is Socrates mortal?

Prolog

man(socrates).

mortal(X) :- man(X).

?- mortal(socrates).

Facts, Rules and Queries



- Fact: Socrates is a man.
man(socrates).
- Rule: All men are mortal.
mortal(X) :- man(X).
- Query: Is Socrates mortal?
mortal(socrates).
- Queries have the same form as facts

Prolog Basics

- Pure Prolog based on Horn clause (Alfred Horn, 1951)
 - Execution of a Prolog program is initiated by the user's posting of a single goal, called the query
 - Prolog engine tries to find a resolution refutation of the negated query (Selective Linear Definite or SLD resolution method - Robert Kowalski)
-

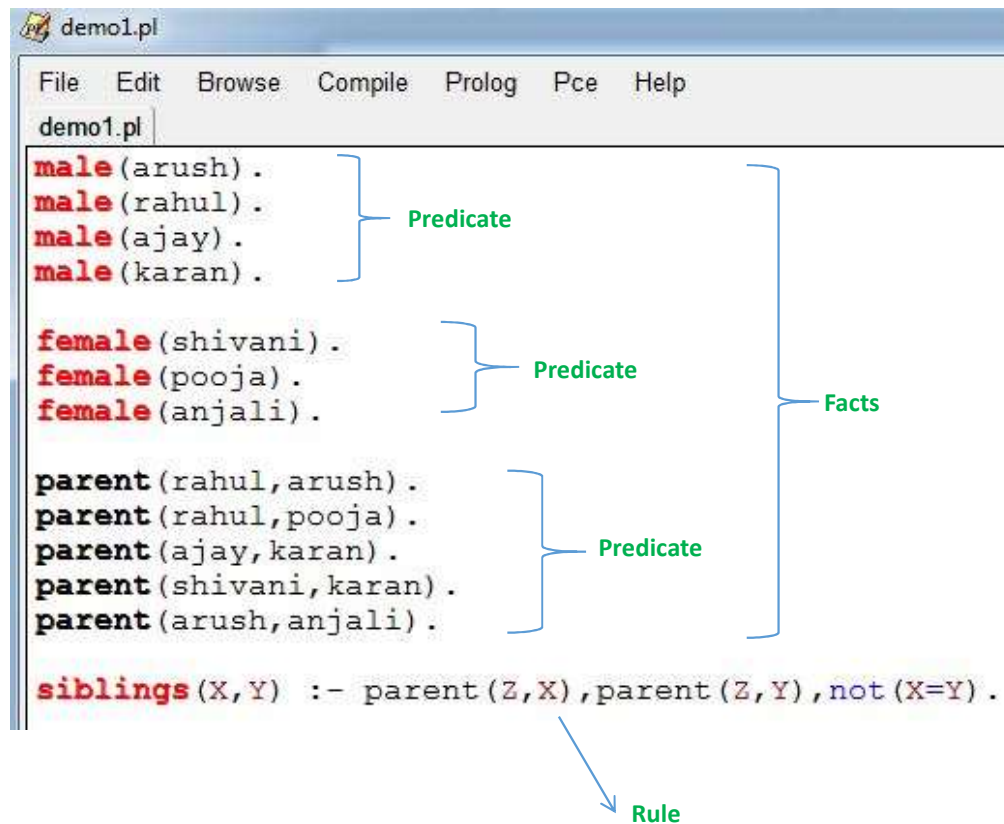
Prolog as Theorem Prover

- Prolog's "Yes" means "I can prove it" --
Prolog's "No" means "I can't prove it"
?- mortal(plato).
No
- This is the closed world assumption: the Prolog program knows everything it needs to know
- Prolog supplies values for variables when it can
?- mortal(X).
X = socrates

Structure of Programs

- Programs consist of procedures (also called predicates)
 - Procedures consist of clauses
 - Each clause is a fact or a rule
 - Programs are executed by posing queries
-

Prolog: A Basic Program



```
demo1.pl
File Edit Browse Compile Prolog Pce Help
demo1.pl
male(arush).
male(rahul).
male(ajay).
male(karan).

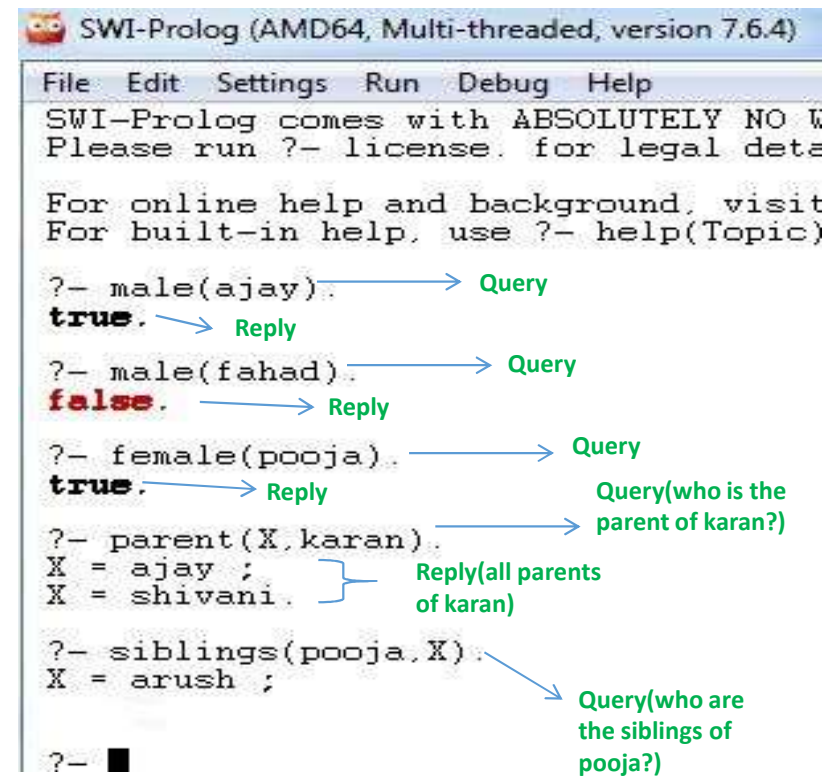
female(shivani).
female(pooja).
female(anjali).

parent(rahul,arush).
parent(rahul,pooja).
parent(ajay,karan).
parent(shivani,karan).
parent(arush,anjali).

siblings(X,Y) :- parent(Z,X),parent(Z,Y),not(X=Y).
```

Annotations:

- Predicate**: Points to `male`, `female`, and `parent`.
- Facts**: Points to the list of `male`, `female`, and `parent` facts.
- Rule**: Points to the `siblings` rule.



```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)
File Edit Settings Run Debug Help
SWI-Prolog comes with ABSOLUTELY NO WARRANTY.
Please run ?- license. for legal details.

For online help and background, visit
http://www.swi-prolog.org
For built-in help, use ?- help(Topic)

?- male(ajay).
true.

?- male(fahad).
false.

?- female(pooja).
true.

?- parent(X,karan).
X = ajay ;
X = shivani.

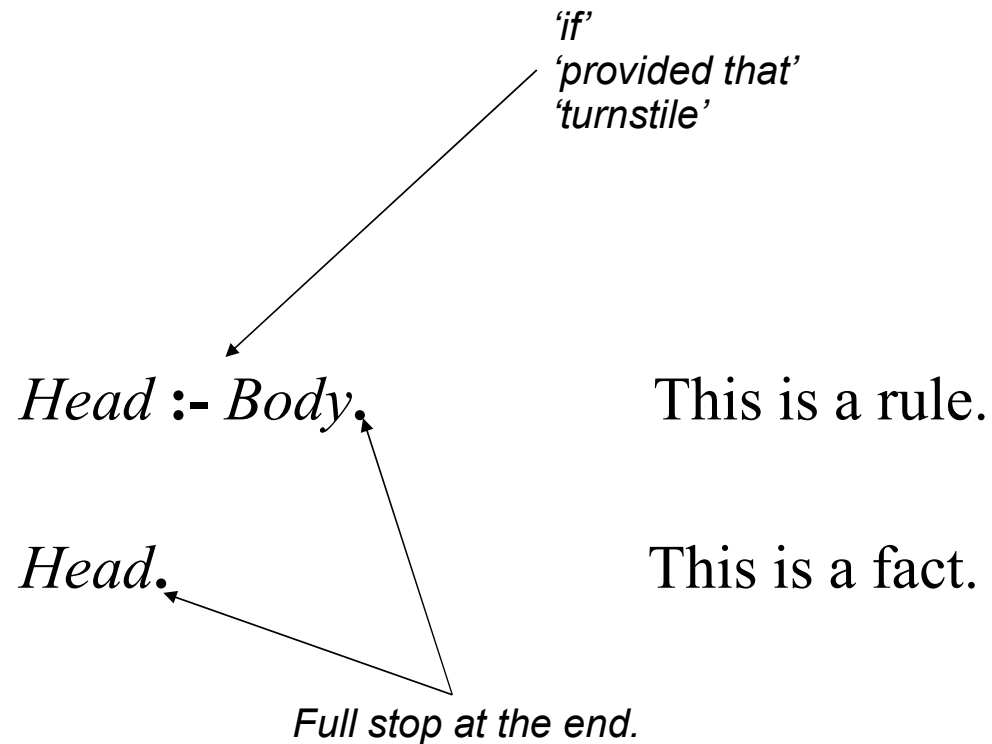
?- siblings(pooja,X).
X = arush ;

?-
```

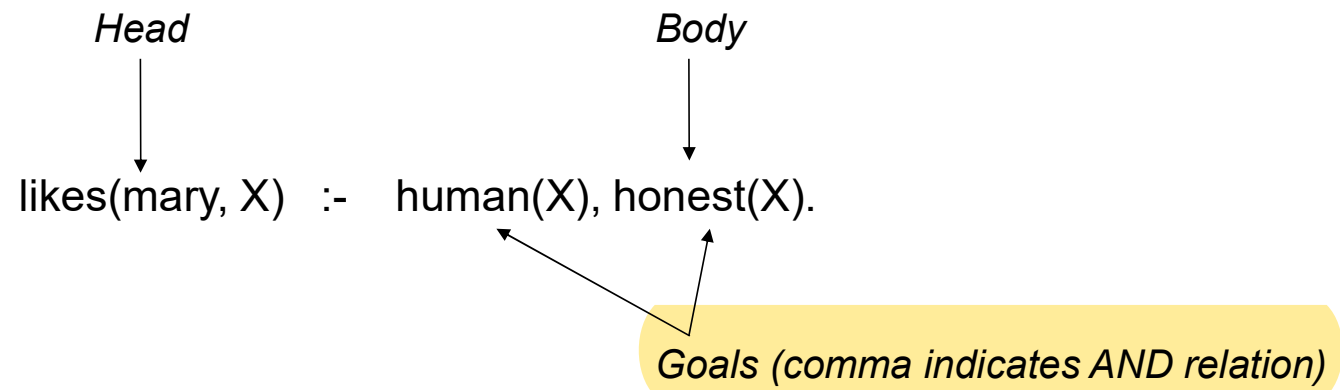
Annotations:

- Query**: Points to `?- male(ajay).`
- Reply**: Points to `true.`
- Query**: Points to `?- male(fahad).`
- Reply**: Points to `false.`
- Query**: Points to `?- female(pooja).`
- Reply**: Points to `true.`
- Query(who is the parent of karan?)**: Points to `?- parent(X,karan).`
- Reply(all parents of karan)**: Points to the list of results `X = ajay ; X = shivani.`
- Query(who are the siblings of pooja?)**: Points to `?- siblings(pooja,X).`

Clauses: Facts and Rules



Body of a (Rule) Clause Contains Goals



Interpretation of Clauses

Clauses can be given a declarative reading or a procedural reading.

Form of clause:

$$H \text{ :- } G_1, G_2, \dots, G_n.$$

Declarative reading:

“That H is provable follows from goals G_1, G_2, \dots, G_n being provable.”

Procedural reading:

“To execute procedure H , the procedures called by goals G_1, G_2, \dots, G_n are executed first.”

Another Example

`mother_child(trude, sally).`

`father_child(tom, sally).`

`father_child(tom, erica).`

`father_child(mike, tom).`

`sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).`

`parent_child(X, Y) :- father_child(X, Y).`

`parent_child(X, Y) :- mother_child(X, Y).`

?- sibling(sally, erica).

Yes

How it Works

- Initially, the only matching clause-head for `sibling(sally, erica)` is the first rule head, so proving the query is equivalent to proving the body of that clause
- Clause proved with appropriate variable bindings, i.e., the conjunction (`parent_child(Z,sally)`, `parent_child(Z,erica)`).
- The next goal to be proved is the leftmost one of this conjunction, i.e., `parent_child(Z, sally)`.
- Two clause heads match this goal. The system creates a choice-point and tries the first alternative, whose body is `father_child(Z, sally)`. (possible backtracking point)**
- This goal can be proved using the fact `father_child(tom, sally)`, so the binding `Z = tom` is generated, and the next goal to be proved is the second part of the above conjunction: `parent_child(tom, erica)`.
- Again, this can be proved by the corresponding fact. Since all goals could be proved, the query succeeds.

Search

- Logic Programming 'procedure' can either fail or succeed
 - If it succeeds, it may have computed some additional information (conveyed by instantiating variables)
 - ✓ Question: What if it fails.....? Answer: find another way to try to make it succeed
 - ✓ Most logic programming languages use a simple, fixed search strategy to try alternatives
-

Search

- If a goal succeeds and there are more goals to achieve, then remember any untried alternatives and go on to the next goal
 - If a goal succeeds and there are no more goals to achieve, then stop with success
 - If a goal fails and there are alternate ways to solve it, then try the next one
 - If a goal fails and there are no alternate ways to solve it and there is a previous goal, then propagate failure back to the previous goal
 - If a goal fails and there are no alternate ways to solve it and no previous goal then stop with failure.
-

Cuts

- A cut prunes or “cuts out” and unexplored part of a Prolog search tree
 - Cuts can therefore be used to make a computation more efficient by eliminating futile searching and backtracking
 - Cuts can also be used to implement a form of negation
-

Note

- There will be one more tutorial on Prolog (syntax) – after your midsem
 - Assignments are likely to be posted by next weekend
-

END
