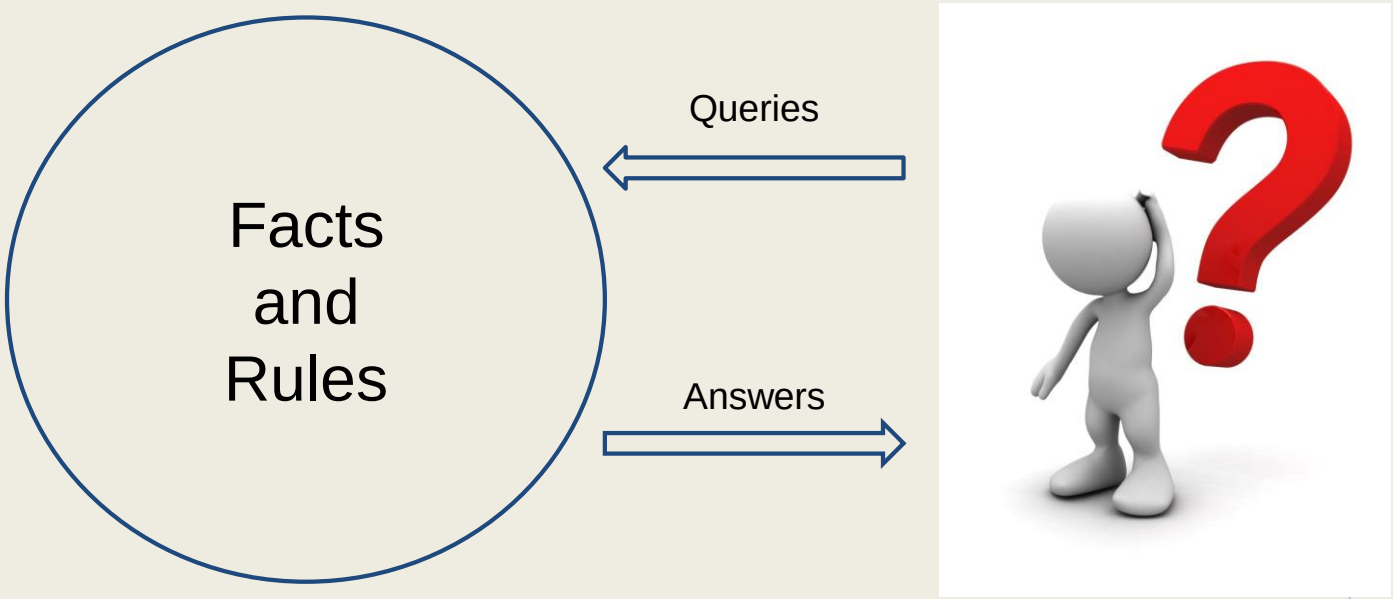
Prolog Recitation

CENG242 Spring 2017-2018

Introduction



Knowledge Base

Introduction to Prolog

- Prolog is a logic programming language. It has its roots in first-order logic.
- Prolog is declarative. The program logic is expressed in terms of relations between objects.
- It is used for symbolic and non-numerical computations.
- It has a built in inference and search mechanism.
- A logic program consists of clauses such as facts and rules. These clauses represents the knowledge base.
- A computation in Prolog is initiated by running a query on the knowledge base.

SWI Prolog

- Fast compiler : Even very large applications can be loaded in seconds on most machines.
- Flexibility : SWI-Prolog can easily be integrated with C.
- The homeworks and quizes will be tested with moodle which uses SWI Prolog compiler.
- SWI Prolog can be started by writing "swipl" command in the terminal.

Example

Knowledge base;

- Doll is a toy.
- Train is a toy.
- Ann plays with train.
- Ann likes the toy that she plays with.
- John likes anything that Ann likes.

Example in Prolog

```
toy(doll).
toy(train).
plays(ann,train).
likes(ann, X) :- toy(X), plays(ann, X).
likes(john, Y) :- likes(ann, Y).
```

Example in Prolog:Testing

- Run it
  - Write “swipl” to the terminal
  - Load the file by writing “[*FILENAME*].”

- Test the KB
  - toy(doll). : Is `doll` a `toy`?
  - likes(john, Z). :Is there a `Z` that `john` likes?
    i.e., List all the `Z`’s that `john` likes.

Syntax

- Program logic is expressed in terms of relations, and a computation is initiated by running a query over these relations.
- Relations and queries are constructed using Prolog's single data type, the term.
- Relations are defined by clauses.
- A predicate represents some relation or property in the system.

Data Types

- Atoms : Names which always begin with a lowercase letter.
- Ex : `doll, toy, plays`

- Number : Floats or integers.
- Ex : `1, 2, 0.5`

- Variable : Placeholders which always begin with an uppercase letter or an underline character.
- Ex : `X, Y`

- Structure : Compound terms.
- Ex : `plays(ann, train).`

Clauses

- Each statement in a Prolog program is called a clause.
- Every clause is terminated with a full-stop (".").
- Facts, rules and queries are clauses.

Facts

- A fact is just one predicate.
- A fact is an unconditionally true statement.
- It is a one-line statement that ends with a full-stop.

Ex :

```
apple.
        car(bmw).
        female(mary).
        eats(mary, icecream).
```

Rules

- Head :- Body.
- In the body of a rule:
  - ":-" stands for if
  - "," stands for and
  - ";" stands for or
- The body is the conditional part. The head is the conclusion.
- The body can contain conjunction or disjunction of predicates.
- Rules are predicates that are true depending on a given condition.
- It is possible to define recursive rules.
- Order of clauses and goals is important.

Ex :

```
likes(ann, X) :- toy(X), plays(ann, X).
likes(john, Y) :- likes(ann, Y).
```

Queries / Goals

- They are questions to the knowledge base.
- The Prolog engine tries to entail the goal using the facts and the rules.
- There are two kinds of answer:
  - - Yes/No.
  - - Unified Answer.

Ex :

```
?- toy(doll).
true.
?- parent(X, Y).
X = pam,
Y = bob .
```

Proof Search and Backtracking

- Prolog does goal driven search by maintaining a unification on the variables.
- Unification is an algorithmic process of solving equations between symbolic expressions. A solution of a unification problem is denoted as a substitution, that is, a mapping assigning a value to each variable of the problem's expressions.
- Using rules, Prolog substitutes the current goals (which matches a rule head) with new sub-goals (the rule body), until the new sub-goals happen to be simple facts.
- Prolog returns the first answer matching the query. When prolog discovers that a branch fails or if you type ‘;’ to get other answers, it backtracks to the previous node and tries to apply an alternative rule at that node.
- Built-in Prolog predicates are not traced, that is, the internals of calls to things like member are not further explained by tracing them.

Proof Search and Backtracking

Example :

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
predecessor(X, Z) :- parent(X, Z).
predecessor(X, Z) :- parent(X, Y), predecessor(Y, Z).
```

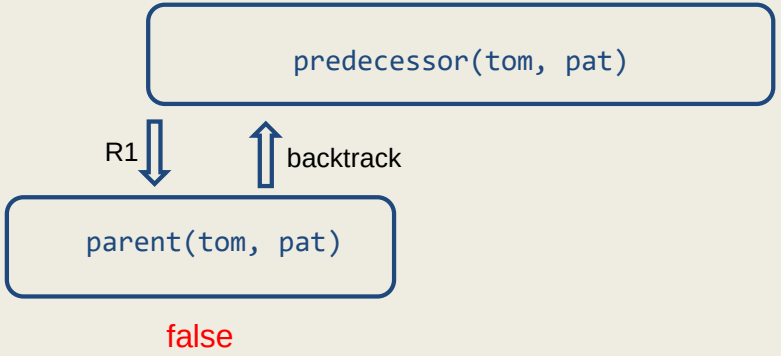
Proof Search and Backtracking

predecessor(tom, pat)

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
predecessor(X, Z) :- parent(X, Z).
predecessor(X, Z) :-
        parent(X, Y),
        predecessor(Y,Z).
```

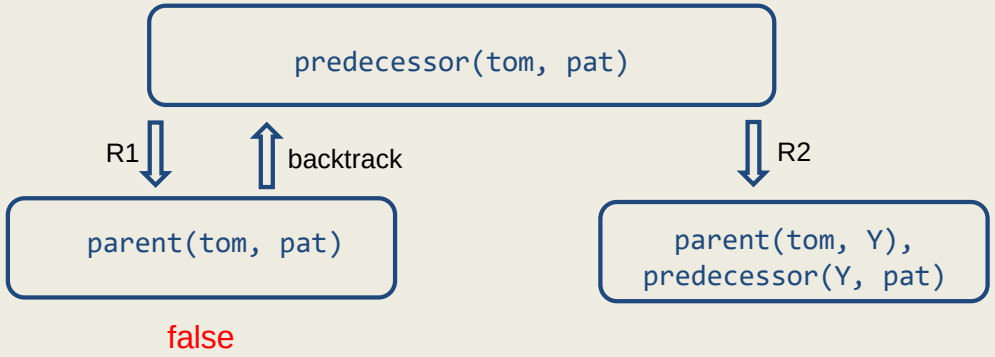
Proof Search and Backtracking

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
predecessor(X, Z) :- parent(X, Z).
predecessor(X, Z) :-
        parent(X, Y),
        predecessor(Y,Z).
```



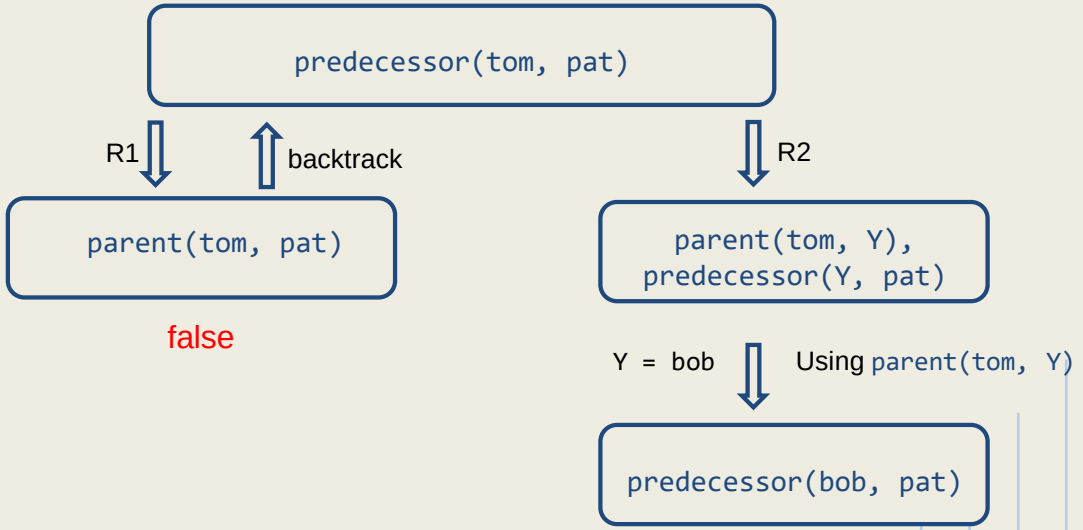
Proof Search and Backtracking

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
predecessor(X, Z) :- parent(X, Z).
predecessor(X, Z) :-
        parent(X, Y),
        predecessor(Y,Z).
```



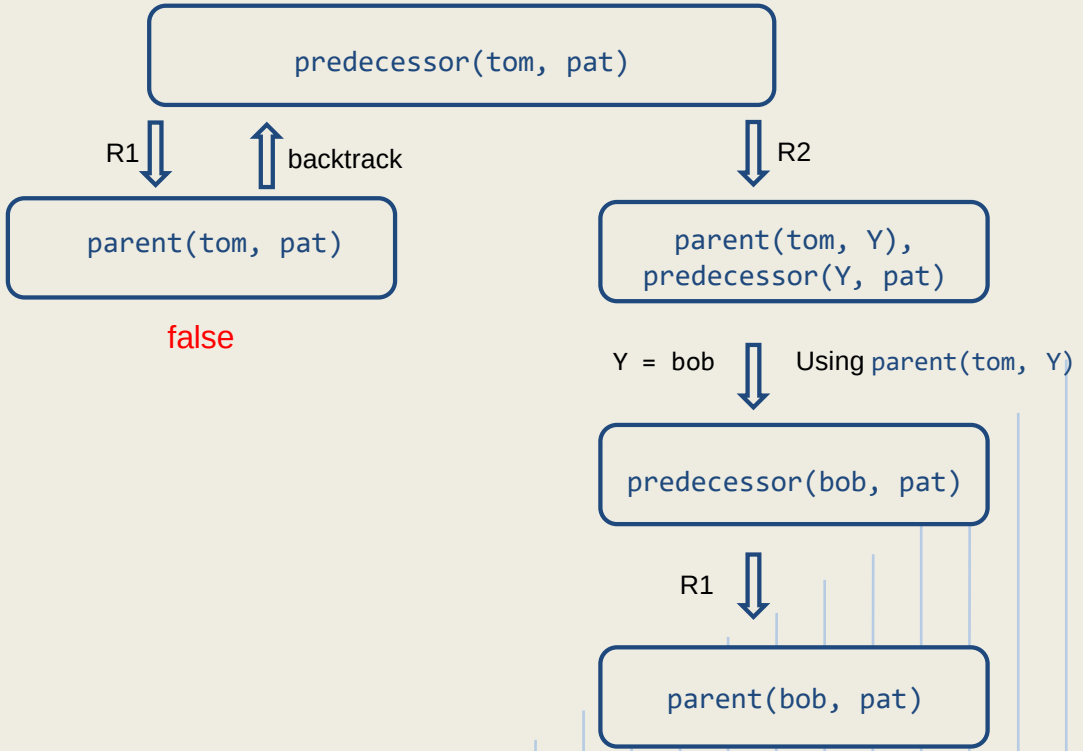
Proof Search and Backtracking

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
predecessor(X, Z) :- parent(X, Z).
predecessor(X, Z) :-
        parent(X, Y),
        predecessor(Y,Z).
```



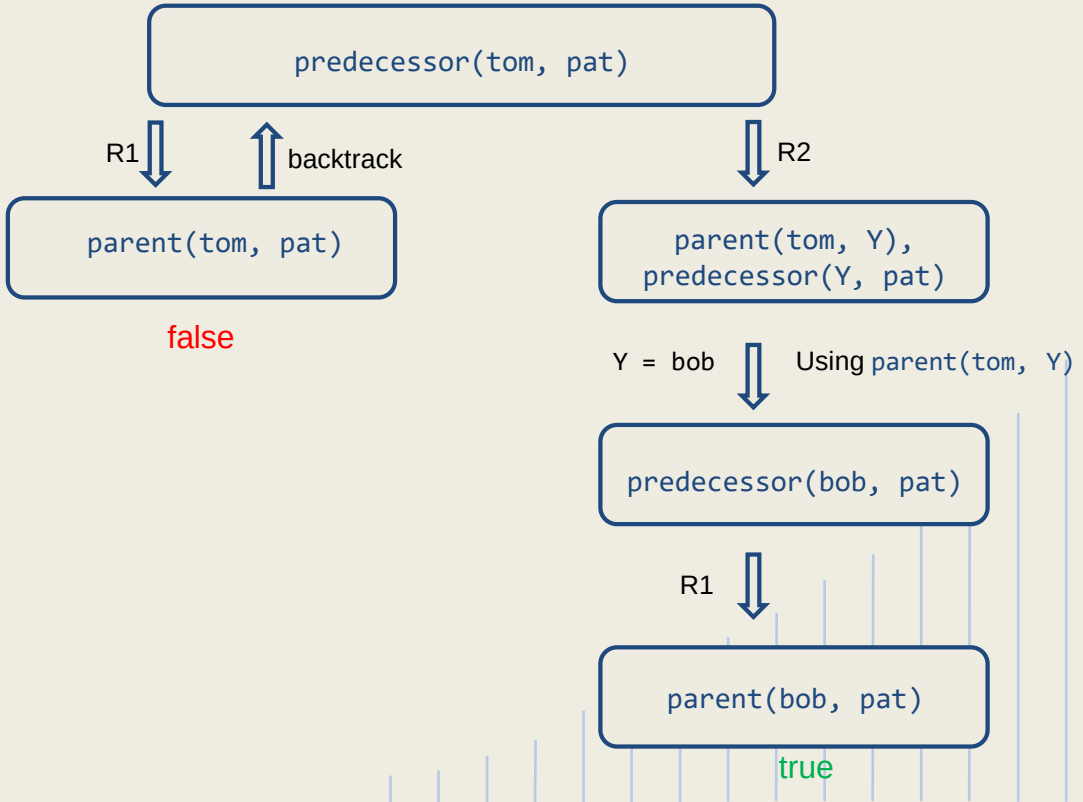
Proof Search and Backtracking

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
predecessor(X, Z) :- parent(X, Z).
predecessor(X, Z) :-
        parent(X, Y),
        predecessor(Y,Z).
```



Proof Search and Backtracking

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
predecessor(X, Z) :- parent(X, Z).
predecessor(X, Z) :-
        parent(X, Y),
        predecessor(Y,Z).
```



Proof Search and Backtracking

- Goal : `predecessor(tom, pat)`.
- The rule that appears first, is applied first. Unifying: `{tom/X} , {pat/Z}`.
- The goal is replaced by `parent(tom, pat)`.
- No fact is present for `parent(tom, pat)`.
- Next rule is applied. Unifying: `{tom/X} , {pat/Z}`.
- New goal : `parent(tom, Y), predecessor(Y, pat)`.
- The first one matches one of the facts `{bob/Y}`.
- Second sub-goal: `predecessor(bob, pat)`.
- Applying the first rule. Unifying: `{bob/X} , {pat/Z}`.
- The goal is replaced by `parent(bob, pat)`.
- The fact `parent(bob, pat)` is present.
- Prolog returns.

Tracing

trace : Activates the debugger
notrace : Switches the debugger off

Ex:

```
[trace]  ?- predecessor(tom, pat).
   Call: (6) predecessor(tom, pat) ? creep
   Call: (7) parent(tom, pat) ? creep
   Fail: (7) parent(tom, pat) ? creep
   Redo: (6) predecessor(tom, pat) ? creep
   Call: (7) parent(tom, _G3534) ? creep
   Exit: (7) parent(tom, bob) ? creep
   Call: (7) predecessor(bob, pat) ? creep
   Call: (8) parent(bob, pat) ? creep
   Exit: (8) parent(bob, pat) ? creep
   Exit: (7) predecessor(bob, pat) ? creep
   Exit: (6) predecessor(tom, pat) ? creep
true .
```

Lists

- Lists are sequences of any number of items
- They are consist of two parts : L = [Head | Tail].
- Lists are handled as trees in Prolog.

Ex:

```
?- [H|T] = [a,b,c,d,e].
H = a,
T = [b, c, d, e].

?- [a|[b|[c|[d|[]]]]] = [a,b,c,d].
true.

?- member(a, [a,b,c]).
true .
```

Operators

- = : Matching operator. X = Y does not evaluate X or Y.
- is : operator forces evaluation of expression on RHS forcing instantiation of values on LHS to the evaluated value.

Ex:

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

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

Arithmetic Operators

- + : Addition
- - : Subtraction
- * : Multiplication
- / : Division
- mod : Modulo
- X<Y :- X is less than Y
- X>Y :- X is greater 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

Logical Operators

- “,” : Logical Conjuction
- “;” : Logical Disjunction
- “:-” : Logical Implication
- “not” : Negation
- “->” : If-then-else

Ex:

```
student(marry).
person(X) :- student(X). % X is a person if X is a student

animal(monkey).
animal(hawk).
flies(hawk).
bird(X) :- animal(X), flies(X).       % X is a bird if X is an animal and X flies.

dead(michaeljackson).
alive(X) :- not(dead(X)).  % X is alive if X is not dead

min(A, B, Min) :- A < B -> Min = A ; Min = B. % If A < B then Min is A, else Min is B
```

Cut

In Prolog, test/fail control is specified with the cut symbol, "!".

Ex:

```
max(A,B,B) :- A < B.
max(A,B,A).
```

However, in the presence of backtracking, incorrect answers can result as is shown here.

```
?- max(3,4,M).
M = 4;
M = 3
```

Cut

To prevent backtracking to the second rule the cut symbol is inserted into the first rule.

```
max(A,B,B) :- A < B,!.
max(A,B,A).
```

Now the erroneous answer will not be generated.

Other examples:

```
% if p holds then r implies g, and if ¬p holds then t implies g.
g :- p,!,r.
g :- t.

% Don’t try other choices of red and color if X satisfies red
color(X,red) :- red(X), !.
```