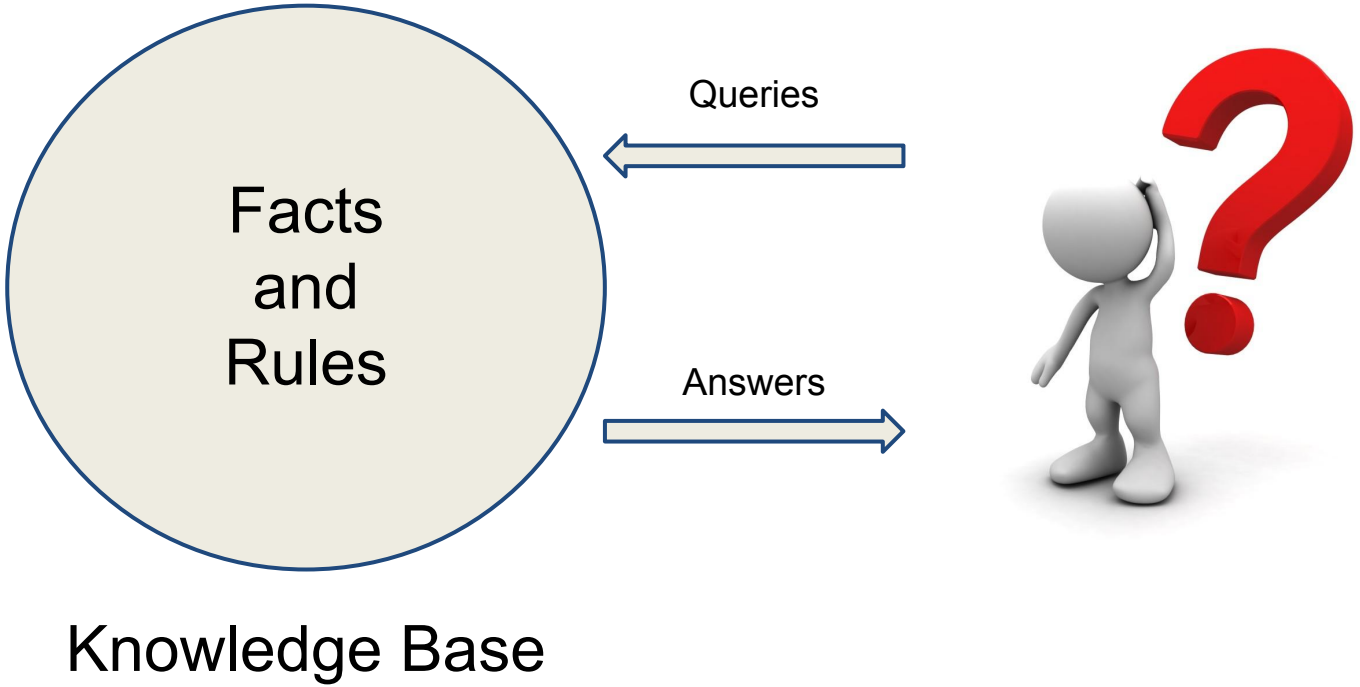
Prolog Recitation

CENG242 Spring 2018-2019

Introduction



Introduction to Prolog

- Prolog (**Pro**gramming in **Log**ic) 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 exams will be tested with CengClass which uses SWI Prolog compiler.
- SWI Prolog can be started by writing "**swipl**" command in the terminal.

Example

An 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.

Same Example in Prolog

The same knowledge base as **simple_kb.pl**;

```
toy(doll).

toy(train).

plays(ann,train).

likes(ann, X) :- toy(X), plays(ann, X).

likes(john, Y) :- likes(ann, Y).
```

Same Example in Prolog: Testing

- Run it by
  - Writing "**`swipl`**" to the terminal and loading the file by writing "**`[simple_kb].`**",
  - Writing "**`swipl -s simple_kb.pl`**".

- 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 are defined by **clauses/predicates**.
- A **predicate** represents some relation or property in the knowledge base.
- Relations and queries are constructed using Prolog's single data type, **the term**.

Prolog Terms

- **Atoms:** Strings beginning with a lowercase letter or enclosed in single or back quotes
  Ex: `doll, toy, plays, ‘a_b_c’`
- **Numbers:** Floats or integers.
  Ex: `1, 2, 0.5, 000123`
- **Variables:** Placeholders which always begin with an uppercase letter or an underline character (‘_’).
  Ex: `X, Y, _abc`
- **Structures:** Compound terms.
  Ex: `plays(ann, train), “abc”, ++(a, b)`

Clauses / Predicates

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

Facts

- A fact is just one predicate which 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

- Rules are predicates that are **true depending on a given condition**.
- Head :- Body.
- The body is the conditional part. The head is the conclusion. In order to prove the head clause, the body should be proven.
- The body can contain conjunction or disjunction of predicates.
- In the body of a rule: "**:-**" stands for if, "**,**" stands for and, "**;**" stands for or
- 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:
  - True / False.
  - Unified Answer.

Ex :

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

Arithmetic 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.
- **+** : 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 Conjunction
- **;** : Logical Disjunction
- **:-** : Logical Implication
- **not()** : Negation
- **->** : If-then-else

Lists

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

Ex:

```
?- [Head | Tail] = [a, 1, b, 3, 2.6, "hello"].
Head = a,
Tail = [1, b, 3, 2.6, "hello"].
?- [a|[b|[c|[d|[]]]]] = [a,b,c,d].
true.
?- member(a, [a,b,c]).
true .
```

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.

Backtracking Example

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).

predecessor(X, Y) :- parent(X, Y).
predecessor(X, Z) :- parent(X, Y), predecessor(Y, Z).
```

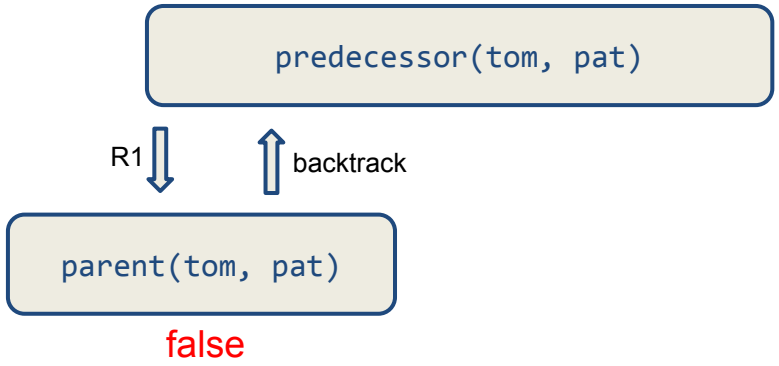
Backtracking Example

```
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).
```

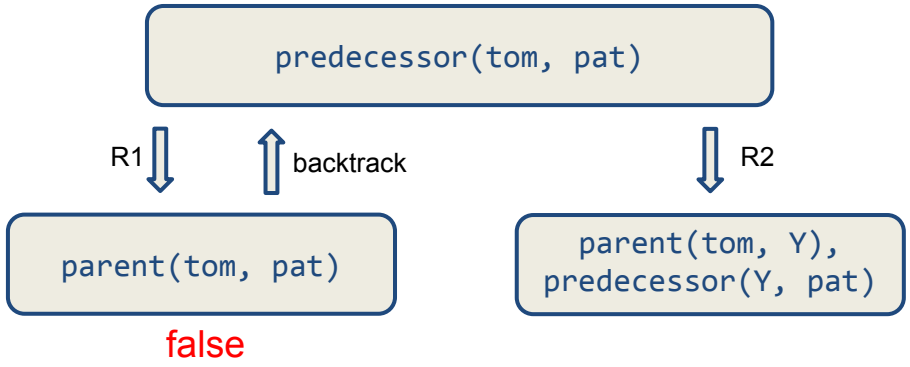
Backtracking Example

```
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).
```



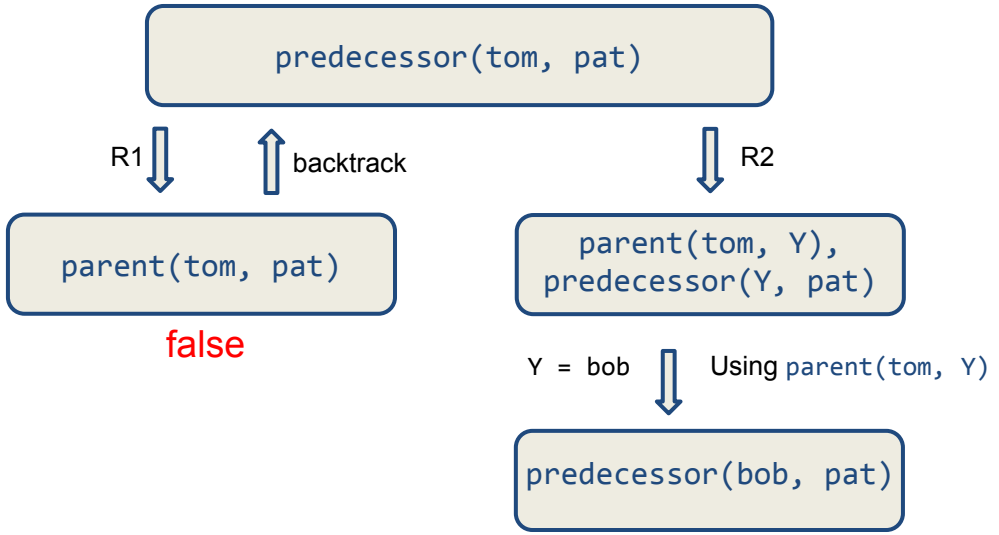
Backtracking Example

```
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).
```



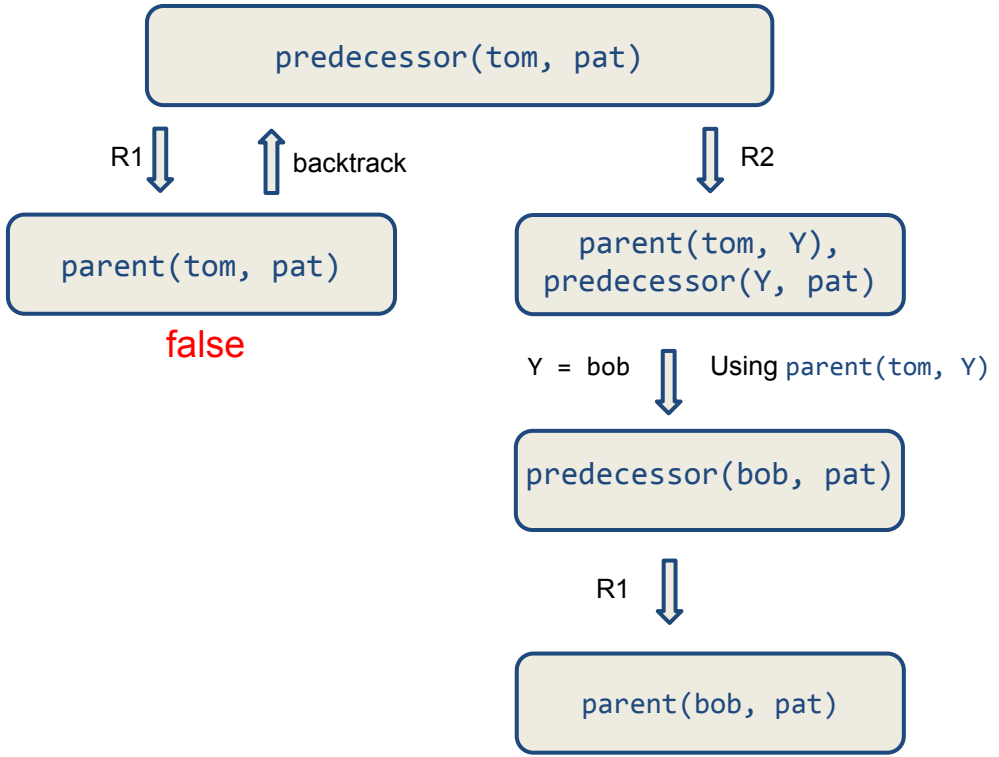
Backtracking Example

```
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).
```



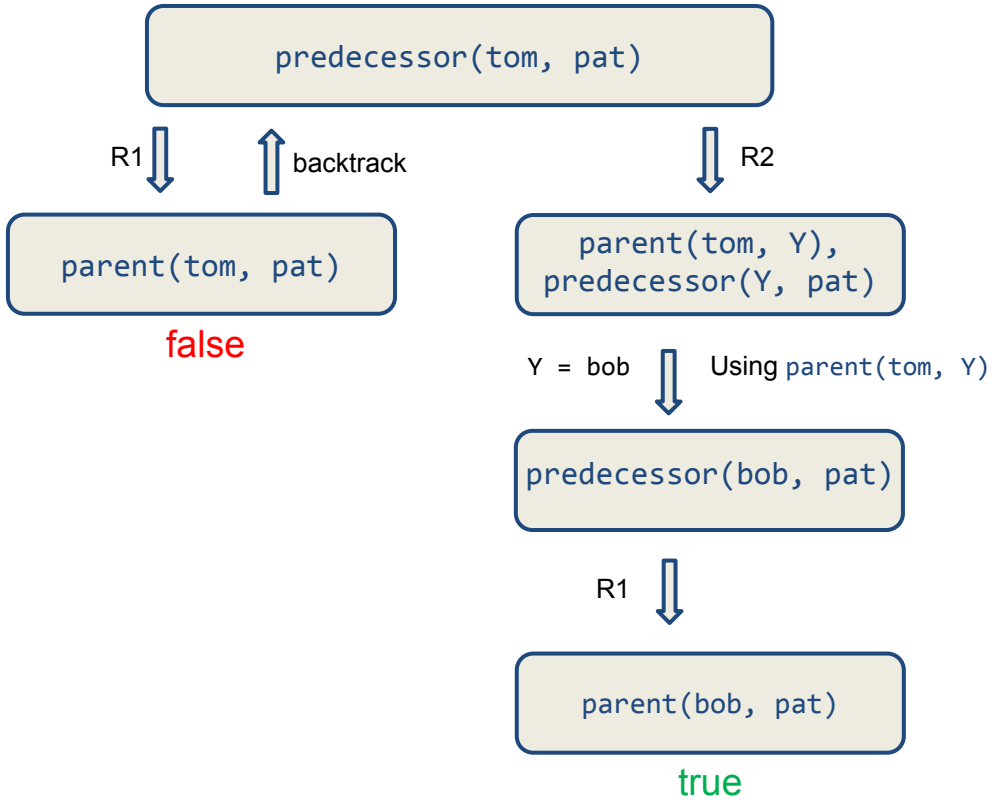
Backtracking Example

```
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).
```



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).
```



Tracing

- **trace** : Activates the debugger.
- **nodebug** : Switches the debugger off.

Ex:

```
?- trace.
true.
[trace]  ?- nodebug.
true.
?-
```

Cut

- The search tree can be pruned by a **cut** symbol, ‘!’.

- Cut prevents backtracking and deletes all the other backtracking points so far, leaving only the solution.

Useful Links

- http://www.learnprolognow.org/

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