Execution of Prolog Programs

- **Unification**: (variable bindings)
  Specializes general rules to apply to a specific problem.

- **Backward Chaining/
  Top-Down Reasoning/
  Goal-Directed Reasoning**:
  Reduces a goal to one or more subgoals.

- **Backtracking**:
  Systematically searches for all possible solutions that can be obtained via unification and backchaining.

Prolog Search Trees

Encapsulate unification, backward chaining, and backtracking.

- Internal nodes are ordered list of subgoals.

- Leaves are success nodes or failures, where computation can proceed no further.

- Edges are labeled with variable bindings that occur by unification.

Describe *all possible computation* paths.

- There can be many success nodes.

- There can be infinite branches.

Prolog Execution Example

```
flight(ny,chicago).                  %Clause 1.
flight(ny,miami).                    %Clause 2.
flight(miami,austin).                %Clause 3.

trip(X,Z) :- flight(X,Z).            %Clause 1.
trip(X,Z) :- flight(X,Y),trip(Y,Z).  %Clause 2.
```


Prolog Search Tree for Example

```
?- trip(ny,austin).
```

Prolog Execution Trace

```
[trace]  ?- trip(ny,austin).
   Call: (7) trip(ny, austin) ? creep
   Call: (8) flight(ny, austin) ? creep
   Fail: (8) flight(ny, austin) ? creep
   Redo: (7) trip(ny, austin) ? creep
   Call: (8) flight(ny, _L205) ? creep
   Exit: (8) flight(ny, chicago) ? creep
   Call: (8) trip(chicago, austin) ? creep
   Call: (9) flight(chicago, austin) ? creep
   Fail: (9) flight(chicago, austin) ? creep
   Redo: (8) trip(chicago, austin) ? creep
   Call: (9) flight(chicago, _L233) ? creep
   Fail: (9) flight(chicago, _L233) ? creep
   Redo: (8) flight(ny, _L205) ? creep
   Exit: (8) flight(ny, miami) ? creep
   Call: (8) trip(miami, austin) ? creep
   Call: (9) flight(miami, austin) ? creep
   Exit: (9) flight(miami, austin) ? creep
   Exit: (8) trip(miami, austin) ? creep
   Exit: (7) trip(ny, austin) ? creep
Yes
```


Prolog Execution Example 2

Example 2 Database:

```
holiday(friday,april14).

 weather(friday,fair).
 weather(saturday,fair).
 weather(sunday,fair).

 weekend(saturday).
 weekend(sunday).

 picnic(Day) :- weather(Day,fair), weekend(Day).
 picnic(Day) :- holiday(Day,april14).
```

Pose the query:

```
picnic(When).
```

Three answers are generated:

```
?- picnic(When).

When = saturday ;
When = sunday ;
When = friday ;
No
```


Prolog Search Tree

Prolog uses **Depth-First Search** (DFS).

Problem with DFS

Can get stuck on infinitely recursive paths, even when a goal is provable.

E.g.,

```
married(X,Y) :- married(Y,X).
married(john,sue).
```

The query:

```
married(sue,john)?
```

The problem is with left recursion.

Solution:

```
spouse(X,Y) :- married(X,Y).
spouse(X,Y) :- married(Y,X).
```

Another example, as we saw last day:

```
above(X,Z) :- above(Y,Z), on(X,Y).
above(X,Y) :- on(X,Y).
```

Controlling Prolog's Reasoning with Cut

The goal "!", pronounced "cut" always succeeds immediately.

It has an important side effect: Once it is satisfied, it disallows either:

- backtracking back over the cut, or
- backtracking and applying a different clause of the same predicate to satisfy the present goal.

You can think of satisfying cut as making a commitment both

- to the variable bindings we've made during the application of this rule, and
- to this particular rule itself.

Describing Cut (!)

The cut goal succeeds whenever it is the current goal, and the **derivation tree is trimmed of all other choices** on the way back to and including the point in the derivation tree where the cut was introduced into the sequence of goals.

Cut tells us: "Do not pass back through this point when looking for alternative solutions. ! acts as a **marker back beyond which Prolog will not go**. **All the choices made prior to the cut are set**, and are treated as though they were the only possible choices.

You can think of Cut as telling the interpreter: "Trust me – if you get this far in the clause, there's no need to backtrack and try another choice for groving this goal,or to try another way of satisfying any of the subgoals that were already proved for this goal."

How to Trace with Cut

When a "!" goal is satisfied:

1. Find the rule that has that cut.
2. Put an oval around the tree branch
   - from the node where the first goal matches the head of that rule
   - down to the node where the first goal is that cut.

For every node circled, no further branches will be explored from that node.

1. Cut Can Reduce Your Search Space

Cut can be used to improve the efficiency of search by reducing Prolog's search space. E.g.,

When two predicates are mutually exclusive.

```
q(X) :- even(X), a(X).
q(X) :- odd(X), b(X).
```

With cut

```
q(X) :- even(X), !, a(X).
q(X) :- odd(X), b(X).
```

1. Reducing Search Space (cont.)

```
a(1) :- b.
a(2) :- e.
b    :- !, c.
b    :- d.
c    :- fail.
d.
e.
```


2. Cut Can Implement Exceptions to Rules

I.e., "To get the right answer".

Cut can be used to encode exceptions to rules. This is use in AI default reasoning.

```
bird(eagle).
bird(sparrow).
bird(penguin).
fly(penguin) :- !, fail.
fly(X) :- bird(X).
```


3. Cut Can Implement NAF

Cut can be used to implement negation as failure.

```
not(X) :- X, !, fail.
not(X).
```

Note that `not` is a meta-logical predicate. It takes a predicate as an argument. E.g.,

`not(fly(penguin)).`

4. Cut Can Remove Multiple Answers

Fixing our isaMother Example. Recall,

```
 1)  isaMother(X) :- female(X),parent(X,_).
 2)  isaFather(X) :- male(X),parent(X,_).

 3)  isa(X) :- female(X),parent(X,_), !.

 4)  top(X) :- isa(X).

 5)  top2(X):- female(X), isa2(X).
 6)  isa2(X):- parent(X,_), !.

 7)  top3(X):- female(X), isa3(X).
 8)  isa3(X):- !, parent(X,_).


 9)  parent(fred,sue).
10)  parent(janet,sue).
11)  parent(fred,tim).
12)  parent(janet,tim).
13)  parent(diane,william).
14)  parent(cathy,kit).

15)  male(fred).
16)  female(janet).
17)  female(diane).
18)  female(cathy).
```

```
?- isaMother(X).
X = janet;
X = janet;
X = diane;
X = cathy;
No

?- isa(X).
X = janet;
No

?- top(X).
X = janet;
No

?- top2(X).
X = janet;
X = diane;
X = cathy;
No

?- top3(X).
X = janet;
X = janet;
X = diane;
X = cathy;
No
```


... Picnic Example II

Returning to our Picnic Example:

```
holiday(friday,april14).
weather(friday,fair).
weather(saturday,fair).
weather(sunday,fair).
weekend(saturday).
weekend(sunday).
```

Rewrite picnic as follows:

```
picnic2(Day) :- weather(Day,fair), !, weekend(Day).
picnic2(Day) :- holiday(Day,april14).
```

Query `picnic2(When).` yields:

```
?- picnic2(When).
No
```


...Picnic Example III

Recall

```
holiday(friday,april14).
weather(friday,fair).
weather(saturday,fair).
weather(sunday,fair).
weekend(saturday).
weekend(sunday).
```

Rewrite picnic as follows:

```
picnic3(Day) :- weather(Day,fair), weekend(Day), !.
picnic3(Day) :- holiday(Day,april14).
```

Query `picnic3(When).` yields:

```
?- picnic3(When).

When = saturday ;
No
```


... Picnic Example IV

Recall

```
holiday(friday,april14).
weather(friday,fair).
weather(saturday,fair).
weather(sunday,fair).
weekend(saturday).
weekend(sunday).
```

Rewrite picnic as follows:

```
picnic4(Day) :- !, weather(Day,fair), weekend(Day).
picnic4(Day) :- holiday(Day,april14).
```

Query `picnic4(When).` yields:

```
?- picnic4(When).

When = saturday ;
When = sunday ;
No
```

Cut Summary

Cuts are:

+ very powerful.

+ can help:
  - improve efficiency (reduce search space)
  - get the right answer (treat exceptions to rules)
  - implement NAF
  - remove multiple answers

– difficult to use safely.

– make for difficult to understand programs.