

Prolog

- Rule
 - Prolog starts from the first rule/fact. If it does not succeed, Prolog tries the second. The query fails if we run out of rules/facts.
 - **The variables with the same name in a rule have the same instantiation** (binding to the same value) for each solution to a particular query.
 - **Identical variable names in separate rules are independent.**
- Exercise:
 - Given the facts that:

```
likes(john, mary).
likes(john, trains).
likes(peter, fast_cars).

hobby(john, trainspotting).
hobby(tim, sailing).
hobby(helen, trainspotting).
hobby(simon, sailing).
```
 - Write a rule that if person1 and person2 have the same hobby, person1 likes person2.
 - Solution:

```
likes(Person1, Person2) :- hobby(Person1, Hobby), hobby(Person2, Hobby).
```

Backtracking

- How does it work?
 - It starts by **trying to solve each goal in a query, left to right**. Recall that goals are connected by “,” which is the and operator.
 - **For each goal, it tries to match a fact or the head of a corresponding rule.**
 - **If a fact or a head matches, it goes on to match any remaining goals.**
 - But what shall we do **if we reach a point where a goal cannot be matched?**
 - Prolog **uses backtracking**.
 - When we reach a point where a goal cannot be matched, we **backtrack to the most recent spot where a choice of matching a particular fact or rule was made.**
 - **We try to match a different fact or rule. If this fails, we go back to the next previous point** where a choice was made, and try a different match there.
 - We **try alternatives until we are able to solve all the goals in the query or until all possible choices have been tried and found to fail.**

Recursion

- A way to loop in Prolog.
- A recursion should have **a first fact that acts as the base case**.
- Then it should have some rule(s) that performs some recursive operation.
- Example
 - To determine if there is a route to Grand Canyon.

```
on_route(grand_canyon).
on_route(Place) :- move(Place, _Method, NewPlace), on_route(NewPlace).
```

```
move(home, bus, boston).
move(boston, plane, las_vegas).
move(las_vegas, vehicle, grand_canyon)
```

- Exercise

- Given

```
parent(john,paul).      /* paul is john's parent */
parent(paul,tom).       /* tom is paul's parent */
parent(tom,mary).       /* mary is tom's parent */
```

- Write a rule to determine if X is Y's ancestor.

- Solution:

```
ancestor(X, Y) :- parent(X, Y). /* someone is your ancestor if they are your
parents */
ancestor(X, Y) :- parent(X, Z), /* or somebody is your ancestor if they are the
parent */
ancestor(Z, Y). /* of someone who is your ancestor */
```

- Exercise

- Given

```
prerequisite(cs231, cs151).
prerequisite(cs232, cs231).
prerequisite(cs333, cs231).
prerequisite(cs421, cs333).
```

- Write a rule that can find all prerequisites of a course.

- Solution:

```
all_prerequisite(X, Y) :- prerequisite(X, Y).
all_prerequisite(X, Y) :- prerequisite(X, Z), all_prerequisite(Z, Y).
```