# Programming in Prolog

## Unification and Search Strategy



Romain Barnoud

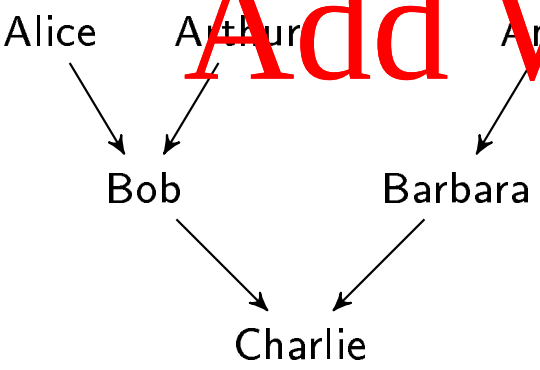


Thanks to:
Dr Fariba Sadri
Claudia Schulz

# How Prolog works? An informal example

## Program

```
parent(alice, bob).
parent(arthur, bob).
parent(anna, barbara).
parent(bob, charlie).
parent(barbara, charlie).

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




## Query

```
?- grandparent(X, charlie).
```

*Who are the grand-parents of Charlie?*

1. Find a rule that matches `granparent(X, charlie)`
2. Solve the query `?- parent(X,Y), parent(Y,charlie).`
3. Find a rule that matches `parent(X,Y)`
4. `parent(alice, bob)` works. Solve `?- parent(bob, charlie).`
5. `parent(bob, charlie)` is a fact: we have found a solution, `X = alice`.
6. Backtrack to point 4 to find other solutions (Arthur and Anna).

# Prolog terms

A ***prolog term*** is one of the following:

- *Atom/Constant* [starts with lower case letter or anything between quotes]

```
john    computing_MSc    x123    'logic and ai'
```

- *Number* [integer or float]

```
0    42    -1729    2.718    6.626E-34
```

- *Variable* [starts with an upper case letter or an underscore]

```
My_variable    X    _Anonymous    _123    _
```

- *Compound term* [$functor(t_1, \ldots, t_N)$:
  functor (*i.e.* a constant name) applied to $N$ terms
  $N$ is called the arity of the term (*NB*: constants are 0-arity terms).]

```
dob(alice, 1970)
world_record('100m', 9.58, date(16, august, 2009))
'long function name 3'(X, cst, _)
```

A ***ground*** term is a term that contains no variable.

# Substitutions

## Definition

A substitution $\theta = \{X_1 \mapsto t_1, X_2 \mapsto t_2, \ldots, X_n \mapsto t_n\}$ is a mapping from variables to terms.



## Applying a substitution to a term



A substitution $\theta = \{X_1 \mapsto t_1, X_2 \mapsto t_2, \ldots, X_n \mapsto t_n\}$ ***applied*** to a term $t$, written $t\theta$, is a new term $s$ identical to $t$, where every occurrence of $X_i$ has been replaced by $t_i$ (for all $i$, simultaneously).
$s$ is called an ***instance*** of $t$.



## Examples

- `f(A,B)` $\{$`A`$\mapsto$`z, B` $\mapsto$ `y`$\}$ gives `f(z,y)`
- `g(X,f(c,X),Y)` $\{$`X`$\mapsto$`a, Y` $\mapsto$ `Z`$\}$ gives `g(a,f(c,a),Z)`
- `h(X,Y,Z)` $\{$`X`$\mapsto$`W, Y` $\mapsto$ `f(W), Z` $\mapsto$ `f(b)`$\}$ gives `h(W,f(W),f(b))`

# Unification

**Definition**

Two terms $T_1$ and $T_2$ unify if there exists a substitution $\theta$ such that $T_1\theta \equiv T_2\theta$.



Do the following terms unify?
If they do, give the instantiation of the variables.



| | |
|---|---|
| `john & john` | `g(a, b, c) & h(a, b, c)` |
| `alice & Alice` | `p(X, 2) & p(f(g), Y)` |
| `3 & 3.0` | `f(a, X) & f(X, b)` |
| `518 & '518'` | `p(X, f(Y)) & p(a, X)` |
| `_000 & Variable` | `p(X, f(Y)) & p(f(f(b)), X)` |
| | `p(X, f(Y)) & p(Y, X)` |

# Unification

**Definition**

Two terms $T_1$ and $T_2$ unify if there exists a substitution $\theta$ such that $T_1\theta \equiv T_2\theta$.





Do the following terms unify?
If they do, give the instantiation of the variables.



| | |
|---|---|
| `john` & `john` ✓ | `g(a, b, c)` & `h(a, b, c)` ✗ |
| `alice` & `Alice` ✗ | `p(X, 2)` & `p(f(g), Y)` ✓ |
| `3` & `3.0` ✗ | `f(a, X)` & `f(X, b)` ✗ |
| `518` & `'518'` ✗ | `p(X, f(Y))` & `p(a, X)` ✗ |
| `_000` & `Variable` ✓ | `p(X, f(Y))` & `p(f(f(b)), X)` ✓ |
| | `p(X, f(Y))` & `p(Y, X)` ✗/✓ |

# = vs. == vs. is vs. =:=

In the following, `S`, `T` and `X` are terms,
In the following, `Expr`, `Expr1` and `Expr2` are arithmetic expressions.



- `S = T` will succeed iff `S` and `T` unify
- `S == T` will succeed iff `S` and `T` are identical



- `X is Expr` will succeed iff
  the evaluation of `Expr` can be unified with `X`
- `Expr1 =:= Expr2` will succeed iff
  `Expr1` and `Expr2` evaluate to the same number



***Opposite predicates:***

| = | == | is | =:= |
|---|---|---|---|
| \= | \== | undef. | =\= |

# = vs. == vs. is vs. =:=

Let's practice!



| Term 1 | Term 2 | = | == | is | =:= |
|---|---|---|---|---|---|
| james | james | | | | |
| X | emily | | | | |
| X | Y | | | | |
| 12 | 5+7 | | | | |
| X | 3+4 | | | | |
| X-Y | 15-2*5 | | | | |

# = vs. == vs. is vs. =:=

Let's practice!



| Term 1 | Term 2 | = | == | is | =:= |
|---|---|---|---|---|---|
| james | james | yes | yes | ✗ | ✗ |
| X | emily | X=emily | no | ✗ | ✗ |
| X | Y | X=_0, Y=_0 | no | ✗ | ✗ |
| 12 | 5+7 | no | no | yes | yes |
| X | 3+4 | X=3+4 | no | X=12 | ✗ |
| X-Y | 15-2*5 | X=15, Y=2*5 | no | no | ✗ |

NB: `?- X=Y, X==Y.` will succeed

`?- X==Y, X=Y.` will not

# Search Strategy: How Prolog answers queries



## Prolog queries

A query is a conjunction of goals: `?- G1, G2, ..., Gn.`

An answer to a query is a substitution $\theta$ such that `G1`$\theta$, `G2`$\theta$, ..., `Gn`$\theta$ are logical consequences of the program.



But how does Prolog find this substitution (or prove that it does not exists)?

# Search Strategy: How Prolog answers queries

## Prolog Search Startegy

1. To solve a query '`?- G1, G2, ..., Gn.`' start with solving the ***first*** goal `G1`.
2. To solve `G1`, find a fact/clause '`H :- B1, B2, ..., Bm`', whose head matches `G1` (i.e. $\exists\theta$ such that $\texttt{G1}\theta \equiv \texttt{H}\theta$).
   If more than one clause satisfy the above condition, we have reached a ***choice point***: in which case, select potential clauses ***from top to bottom***.
   - 3.a If `G1` is the only goal in the query ($n = 1$) and the selected clause is a fact ('`H.`', $m = 0$) — ***succeed***
   - 3.b If a such clause and substitution exist, solve query '`?- B1`$\theta$`, B2`$\theta$`, ..., Bm`$\theta$`, G2`$\theta$`, ..., Gn`$\theta$`.`' (case 1)
   - 3.c If no such clause and substitution, ***backtrack*** to the last choice point and pick the next satisfiable clause. (case 2 for a previous goal)
   - 3.d If there are no more choice points (i.e. all clauses for all choice points have been tried) — ***fail***

# Search Strategy: How Prolog answers queries



## Prolog Search Tree

- At each step, the applicable clauses represent alternative evaluations paths (i.e. different branches of the search tree)
- Prolog searches this tree, *left to right*, *depth-first*, to find a successful evaluation paths
- A path/branch of the search tree ***fails*** if the leaf query has no applicable clause
- A path/branch of the search tree ***succeeds*** if the leaf query is an empty conjunction

# Search Strategy: Example 1

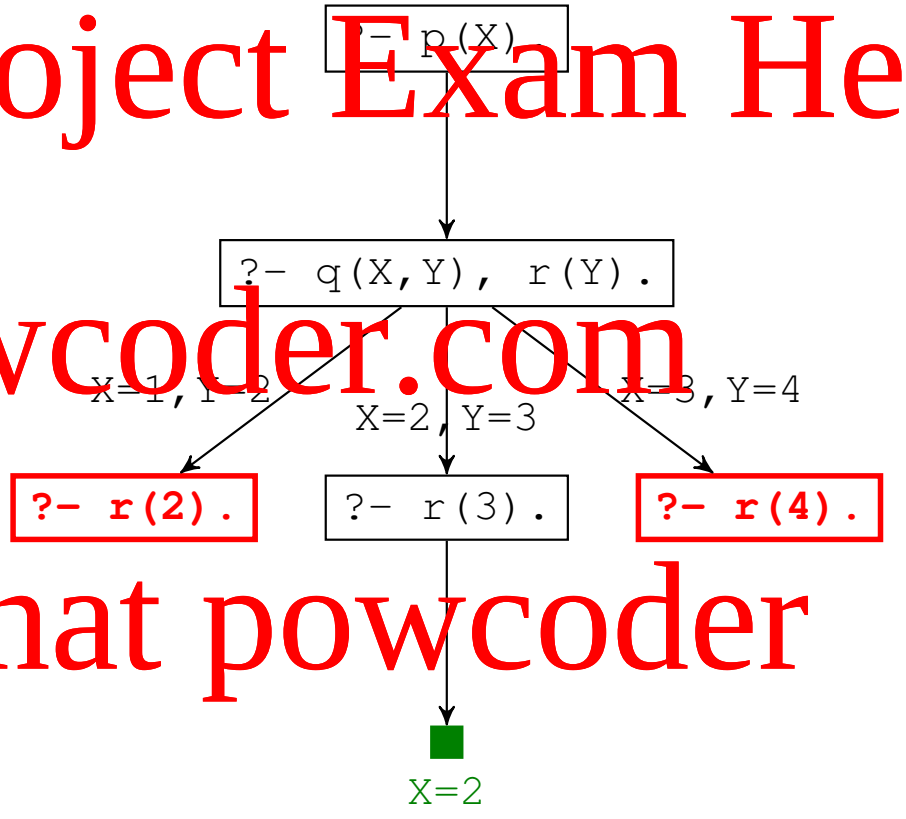


**Program**

```
p(X) :- q(X,Y), r(Y).
p(X) :- s(X).

q(1,2). q(2,3). q(3,4).
r(3).
s(1). s(2).
```

**Query**

```
?- p(X).
```



# Search Strategy: Example 1

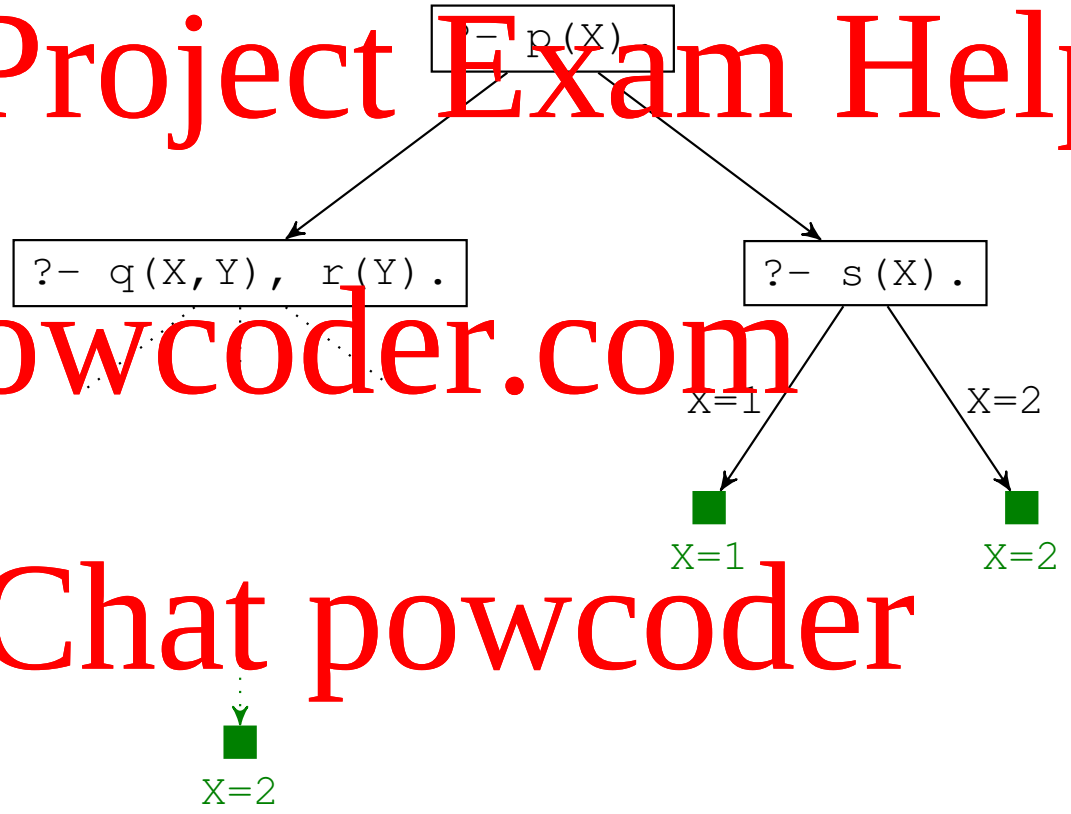


**Program**

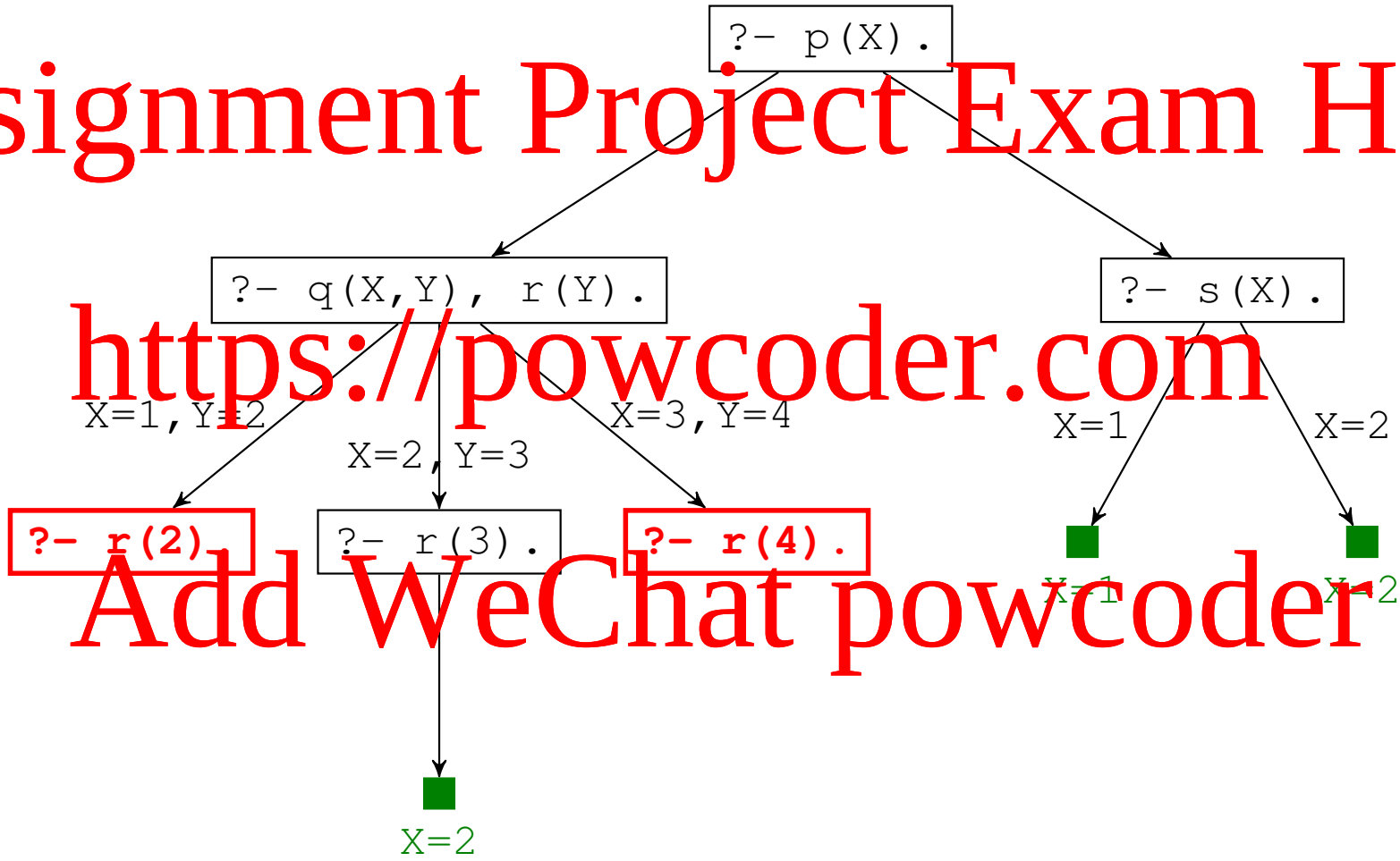
```
p(X) :- q(X,Y), r(Y).
p(X) :- s(X).

q(1,2). q(2,3). q(3,4).
r(3).
s(1). s(2).
```

**Query**

```
?- p(X).
```



# Search Strategy: Example 1 - Complete tree



# Search Strategy: Example 2



## Program

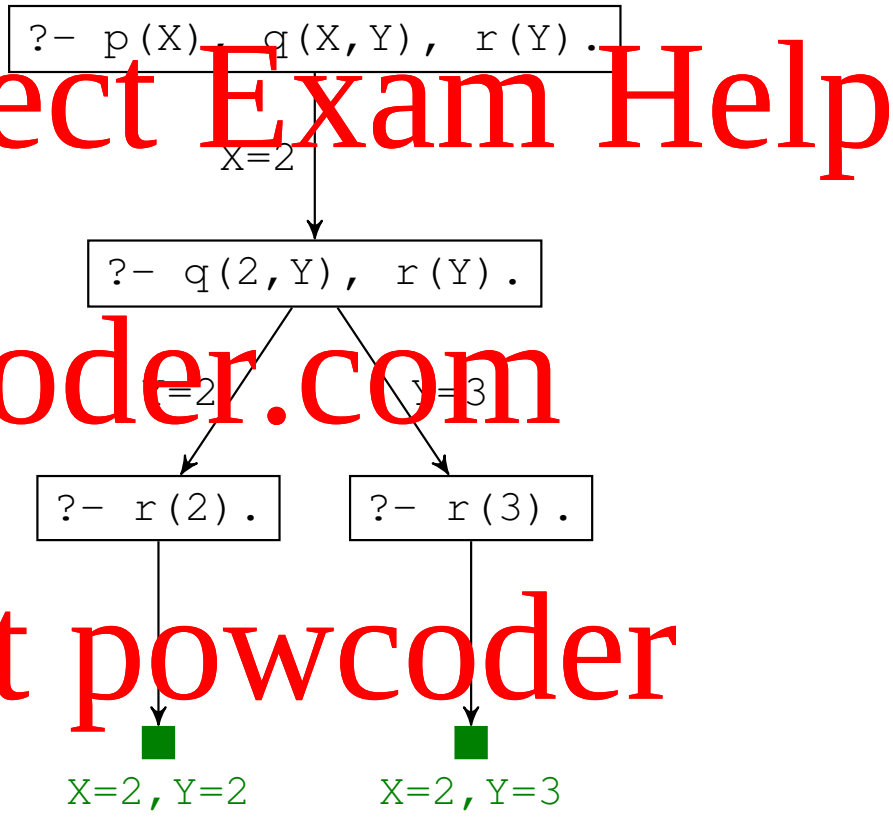
```
p(2).
p(X) :- s(X).
p(X) :- t.

q(X,X).
q(1,5). q(1,3). q(3,1).

r(2). r(3). r(5).

s(1). s(4).
```



## Query

```
?- p(X), q(X,Y), r(Y).
```





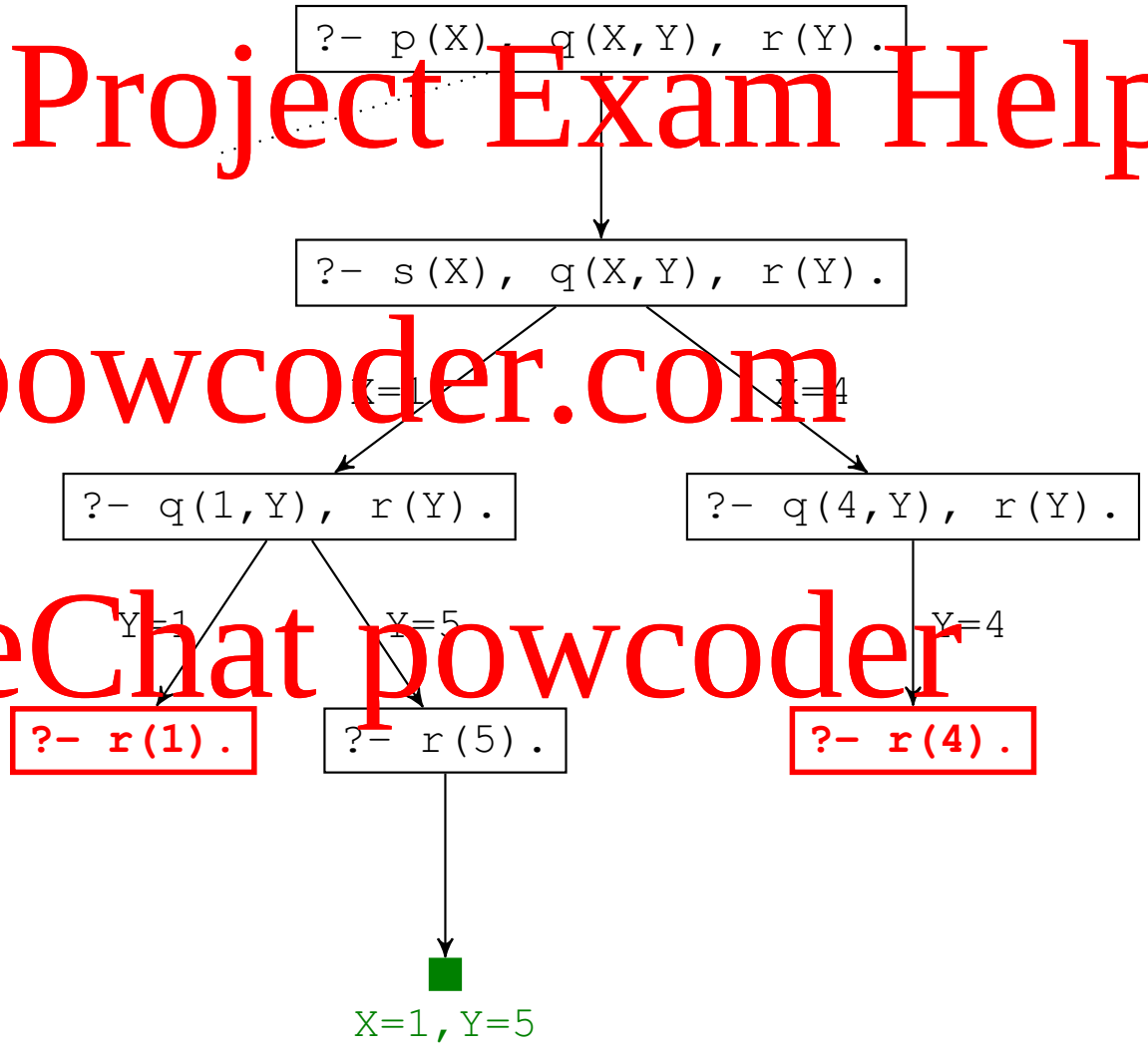
# Search Strategy: Example 2

## Program

```
p(2).
p(X) :- s(X).
p(X) :- t.

q(X,X).
q(1,5). q(1,3). q(3,1).

r(2). r(3). r(5).

s(1). s(4).
```

## Query

```
?- p(X), q(X,Y), r(Y).
```



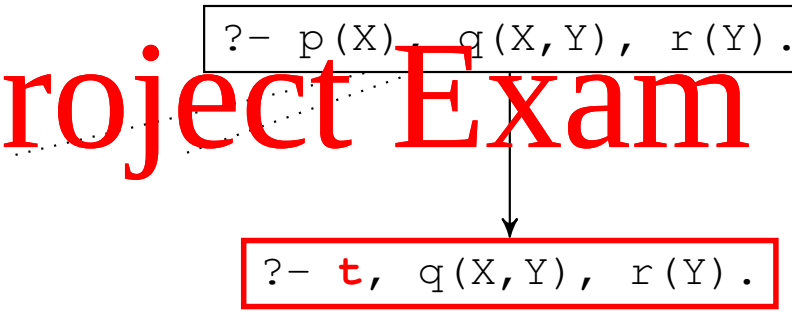
# Search Strategy: Example 2

## Program

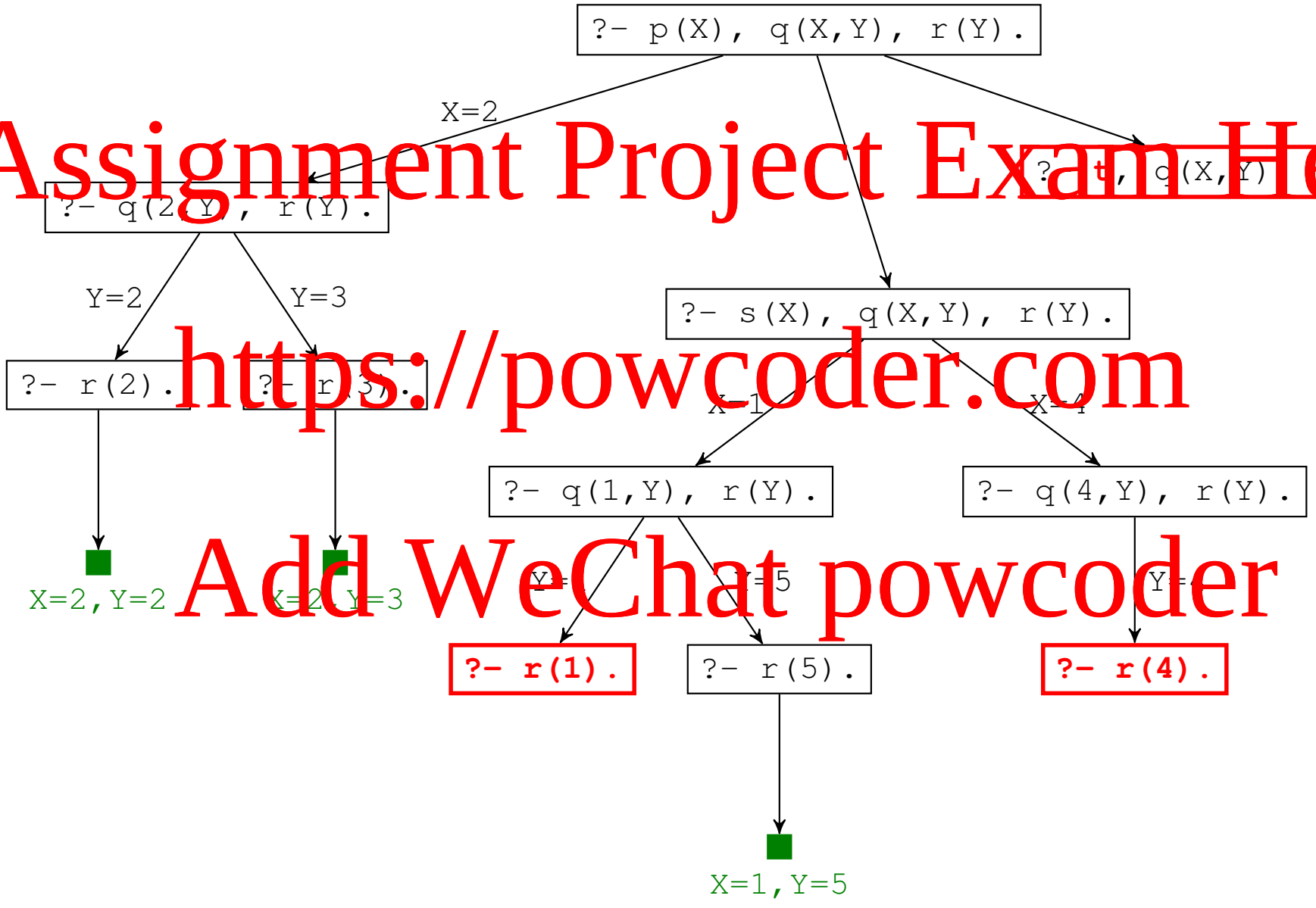
```
p(2).
p(X) :- s(X).
p(X) :- t.

q(X,X).
q(1,5). q(1,3). q(3,1).

r(2). r(3). r(5).

s(1). s(4).
```

## Query

```
?- p(X), q(X,Y), r(Y).
```



# Search Strategy: Example 2 - Complete tree



# What have we learned today?



- What is unification and how it works
- What algorithm is used by Prolog to generate answers
- Why the order of clauses and the order of goals in clauses/queries matter