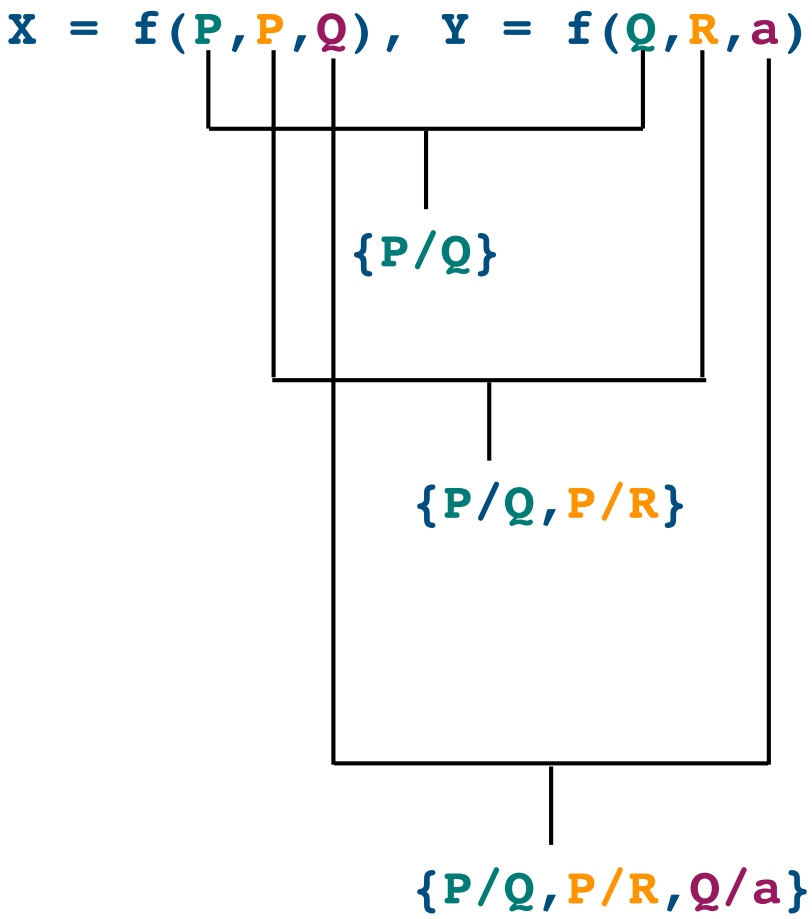
Unification

At the core of how Prolog computes is **Unification**, which is based on **Substitution**.

There are 3 rules for unification:

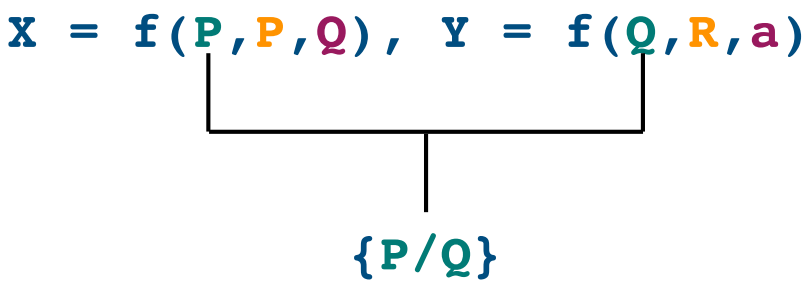
- Atoms unify if they are identical
  - e.g., monday & monday unifty but not monday & wednesday.
- Variables unify with anything.
  - e.g., X & monday unify, X & black (friday).
- Compound terms unfiy only if their top-function symbols and arities match and their arguments unify recursively.
  - e.g., black(X) & black(friday) unify, next(thursday, Y) & next(thursday, friday) unify, play(sunday) & study(X) do not unify.

Unification Example

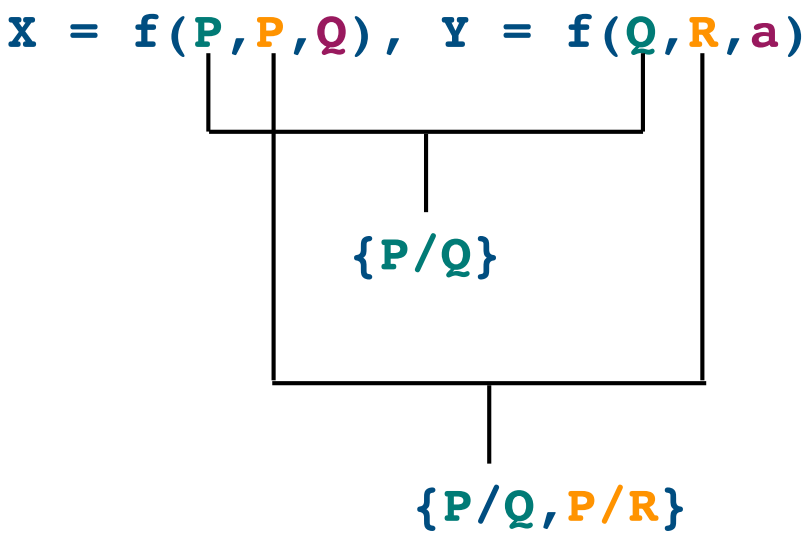


How to get the correct solution {P/a,R/a,Q/a}?

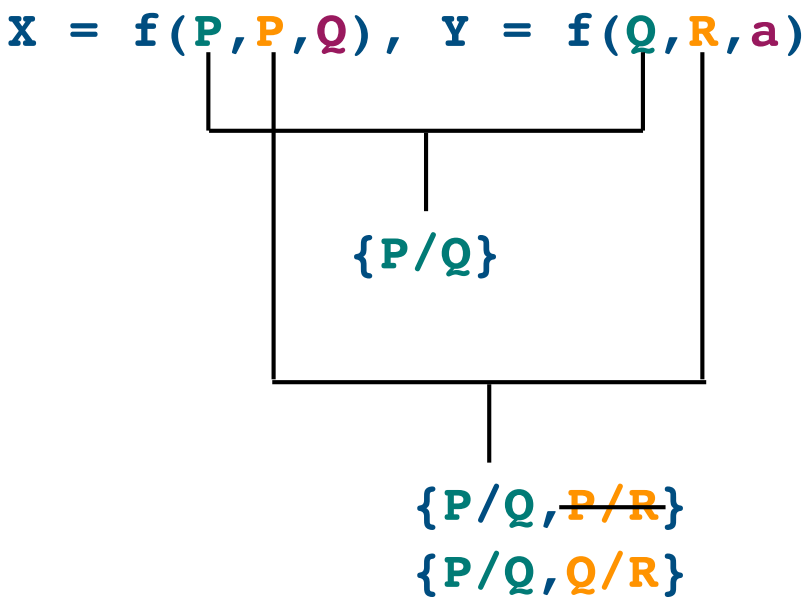
Unification Example



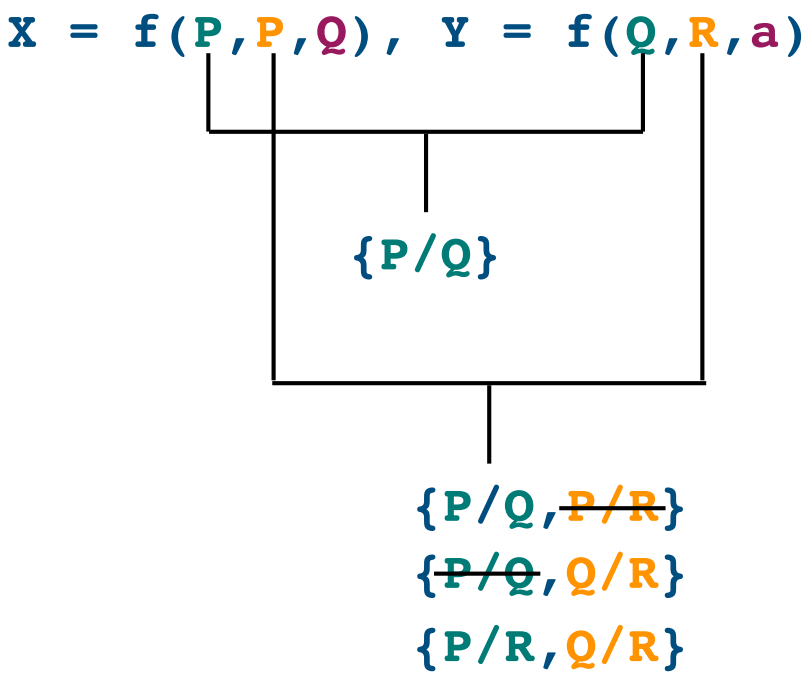
Unification Example



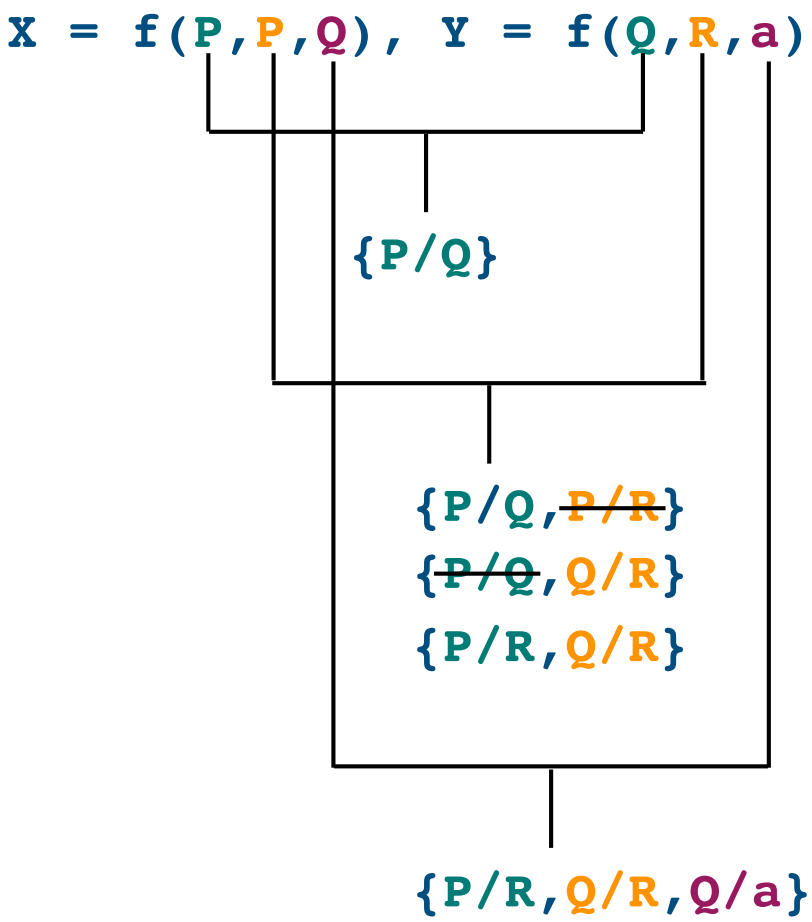
Unification Example



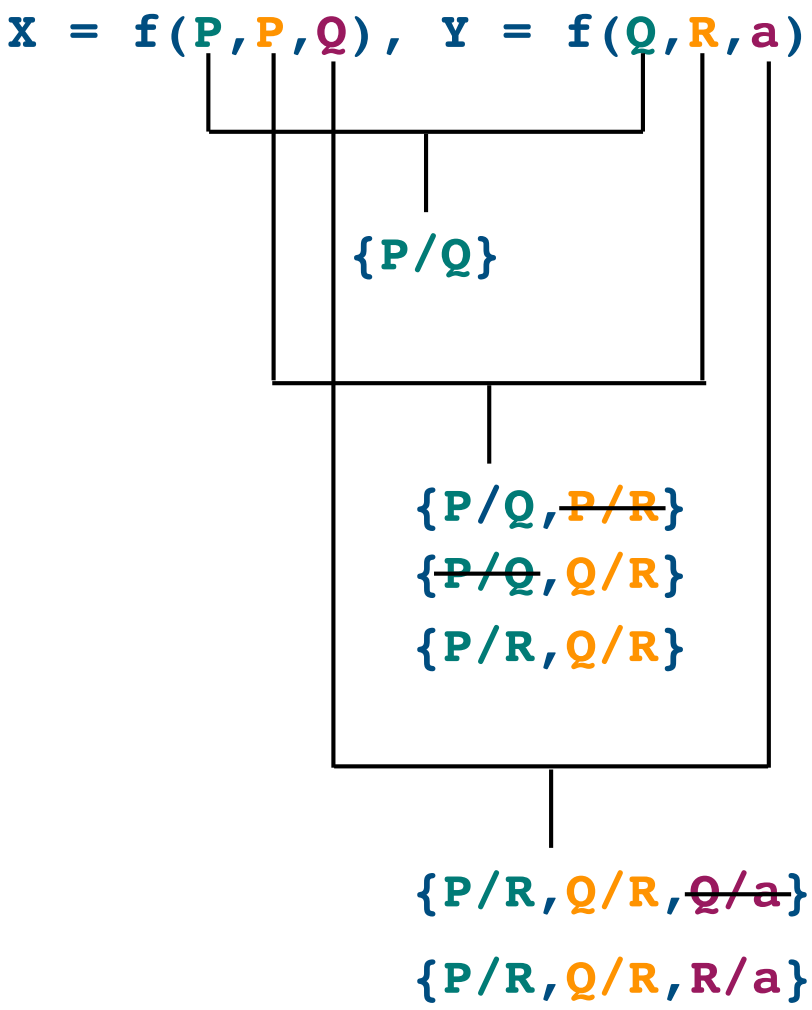
Unification Example



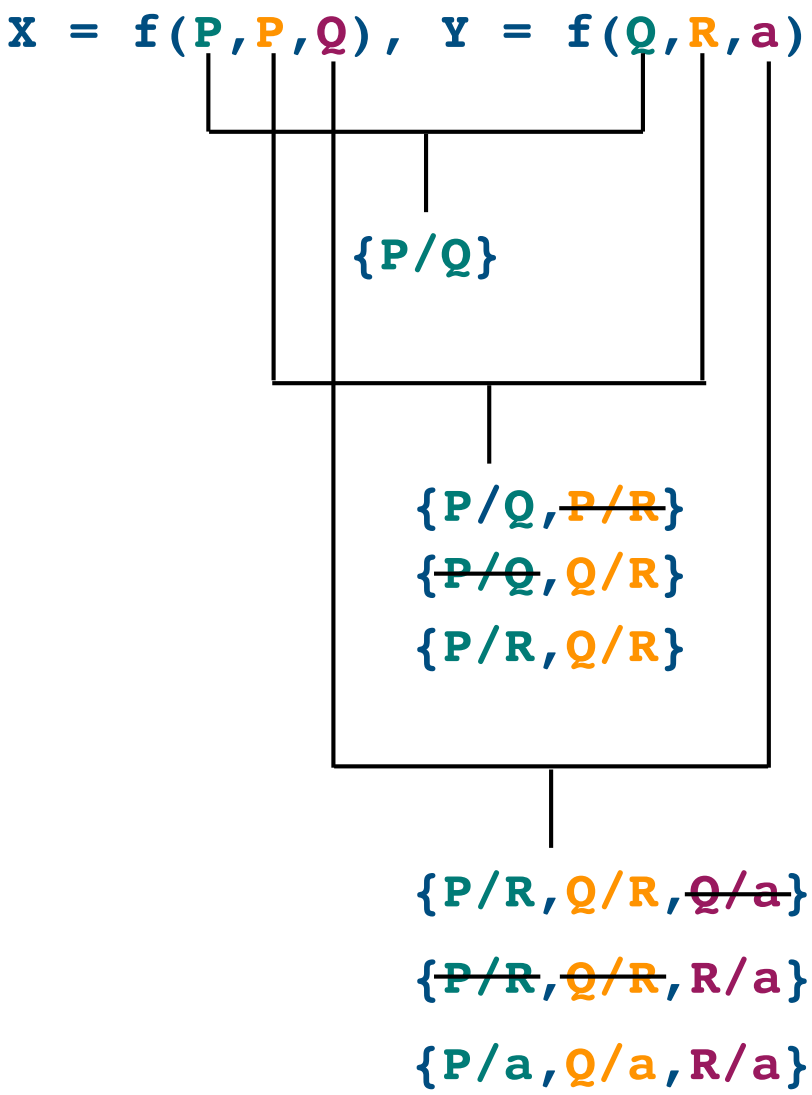
Unification Example



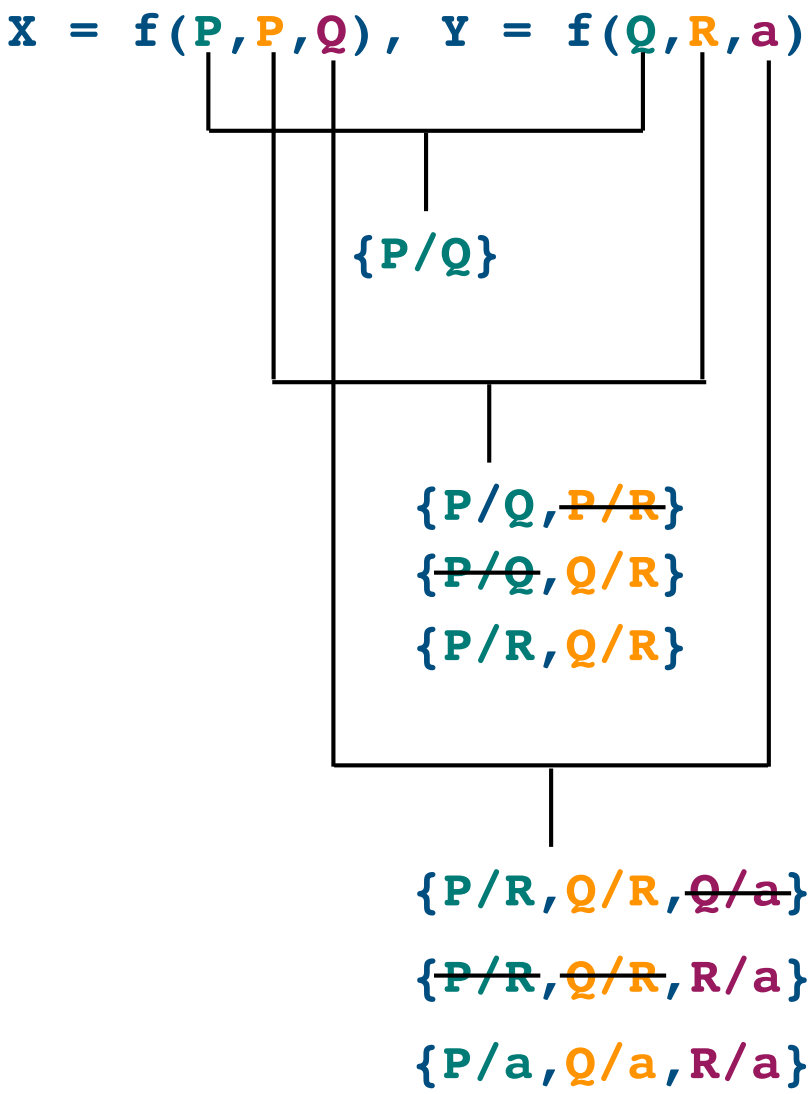
Unification Example



Unification Example



Unification Example



Propagate a current unifier to the previous and vice versa!

```
unify(X,Y,θ) =
 X = Xθ
 Y = Yθ
 case
  X is a variable that does not occur in Y:
   return (θ{X/Y} ∪ {X/Y}) /*replace X with Y in the substitution terms of θ add X/Y to θ*/
  Y is a variable that does not occur in X:
   return (θ{Y/X} ∪ {Y/X}) /*replace Y with X in the substitution terms of θ add Y/X to θ*/
  X and Y are indentical constants or variables:
   return θ

  X is f(X1,...,Xn) and Y is f(Y1,...,Yn):
   return (fold_left (fun θ (X,Y) -> unify(X,Y,θ))
               θ [(X1,Y1),...,(Xn,Yn)])
  otherwise:
   raise FAIL
}

let unify(X,Y) = unify(X,Y,ϵ)
```

Unify

```
unify(f(P,P,Q),f(Q,R,a),ϵ):

   X = f(P,P,Q), Y = f(Q,R,a)
   fold_left (fun θ (X,Y) -> unify(X,Y,θ)) ϵ [(P,Q),(P,R),(Q,a)]

        unify(P,Q,ϵ)
           X = P, Y = Q
           θ = [P/Q]

              unify(P,R,[P/Q])
                 X = P[P/Q] = Q, Y = R[P/Q] = R
                 θ = [P/Q]{Q/R} ∪ {Q/R} = [P/R, Q/R]

                    unify(Q,a,[P/R, Q/R])
                       X = Q[P/R, Q/R] = R, Y = a
                       θ = [P/R, Q/R]{R/a} ∪ {R/a} = [P/a, Q/a, R/a]
```

θ = [P/a, Q/a, R/a]

Example

Abstract Interpreter Example

Program P:

```
fatherOf(abe,homer).

parentOf(homer,bart).

grandfatherOf(X, Y) :–
        fatherOf(X, Z), parentOf(Z, Y).
```

Goal G:

```
?-grandfatherOf(abe, U)
```

Resolvent: grandfatherOf(abe, U)

Abstract Interpreter Example

Program P:

fatherOf(abe,homer).

parentOf(homer,bart).

grandfatherOf(X, Y) :–

fatherOf(X, Z), parentOf(Z, Y).

Goal G:

?-grandfatherOf(abe, U)

?-grandfatherOf(abe, U)

Resolvent: None

Abstract Interpreter Example

Program P:

fatherOf(abe,homer).

parentOf(homer,bart).

grandfatherOf(X, Y) :–

fatherOf(X, Z), parentOf(Z, Y).

Goal G:

?-grandfatherOf(abe, U)

?-grandfatherOf(abe, U)

Resolvent: fatherOf(abe, Z), parentOf(Z, U)

Abstract Interpreter Example

Program P:

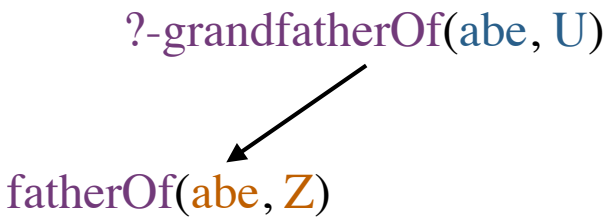
```
fatherOf(abe,homer).

parentOf(homer,bart).

grandfatherOf(X, Y) :–
        fatherOf(X, Z), parentOf(Z, Y).
```

Goal G:

```
?-grandfatherOf(abe, U)
```



Resolvent: parentOf(Z, U)

Program P:

fatherOf(abe,homer).

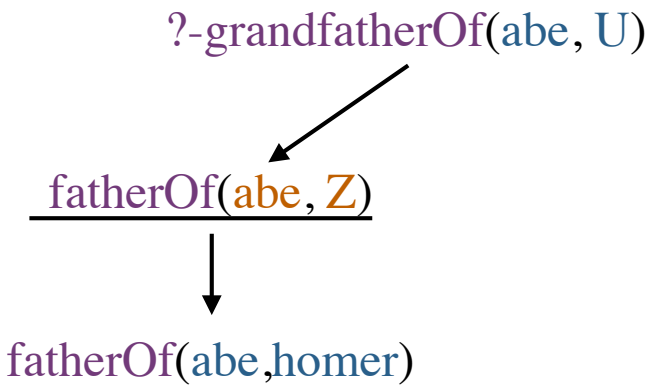
parentOf(homer,bart).

grandfatherOf(X, Y) :–

fatherOf(X, Z), parentOf(Z, Y).

Goal G:

?-grandfatherOf(abe, U)



Resolvent: parentOf(Z, U)

Abstract Interpreter Example

Program P:

fatherOf(abe,homer).

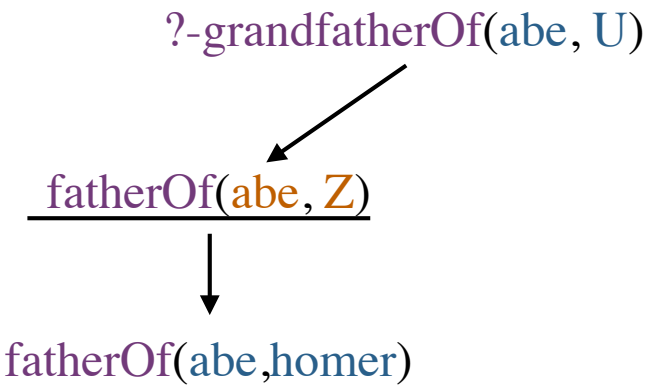
parentOf(homer,bart).

grandfatherOf(X, Y) :–

fatherOf(X, Z), parentOf(Z, Y).

Goal G:

?-grandfatherOf(abe, U)



Resolvent: parentOf(homer, U)

Abstract Interpreter Example

Program P:

fatherOf(abe,homer).

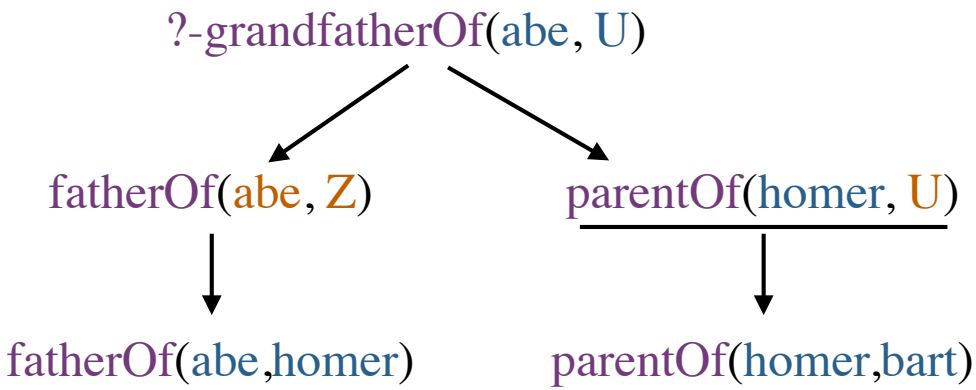
parentOf(homer,bart).

grandfatherOf(X, Y) :–

fatherOf(X, Z), parentOf(Z, Y).

Goal G:

?-grandfatherOf(abe, U)



Resolvent: None

Abstract Interpreter Example

Program P:

fatherOf(abe,homer).

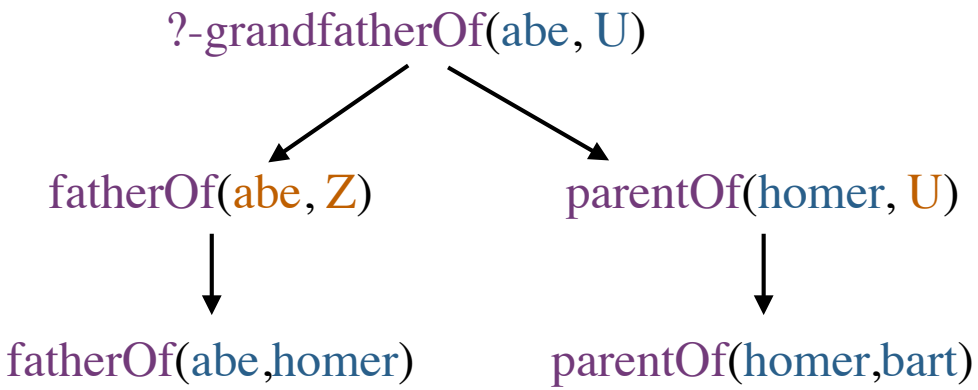
parentOf(homer,bart).

grandfatherOf(X, Y) :–

fatherOf(X, Z), parentOf(Z, Y).

Goal G:

?-grandfatherOf(abe, bart)



Resolvent: None

Program P:

fatherOf(abe,homer).

parentOf(homer,bart).

grandfatherOf(X, Y) :–

fatherOf(X, Z), parentOf(Z, Y).

Goal G:

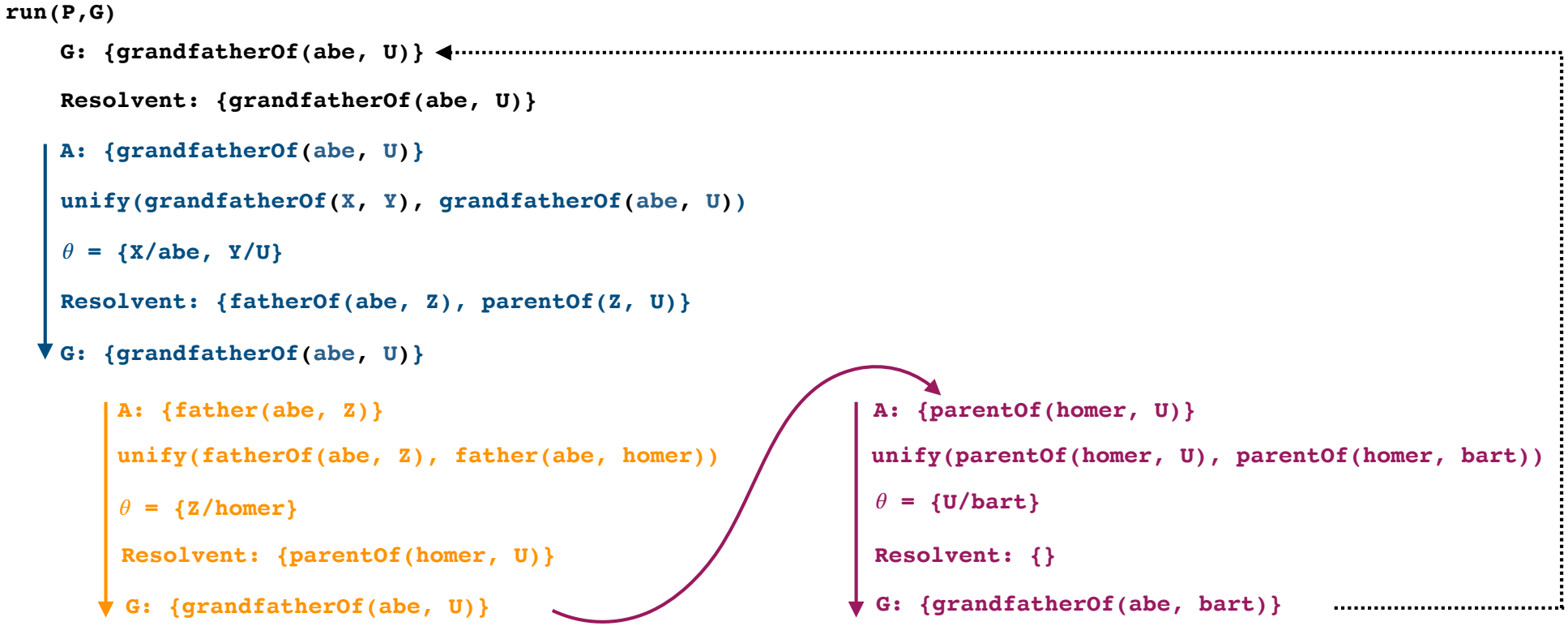
?-grandfatherOf(abe, U)

Input: A goal Goal and a program P

Output: An instance of Goal that is a logical consequence of P.

Algorithm: run(P,Goal)

```
L: G = Goal
   Initialise resolvent to G.
   while (the resolvent is not empty) {
      choose a goal A from the resolvent //random goal
      choose a (renamed) clause A' <- B1,...,Bn from P
        such that A and A' unify with a unifier θ // random rule
      (if no such goal and clause exist, exit the while loop).
      replace A by B1,...,Bn in the resolvent
      apply θ to the resolvent and G
   }
   If the resolvent is empty, then output G, else goto L.
```



- In the code, renaming freshens a clause (or a term) by returning a new clause (or a new term) where the clause (or term) variables have been renamed with fresh variables.
- We may need to apply a rule multiple times in the nested loop. Keep a rule refreshed before using avoids naming confliction.