Prolog Tutorial

Dr A Sahu
Dept of Computer Science & Engineering
IIT Guwahati

Outline

- What is Prolog?
- An example program
- Syntax of terms
- Some simple programs
- Terms as data structures, unification
- The Cut

What is Prolog?

- Prolog is the most widely used language to have been inspired by logic programming research.
- Logic program: **consist of facts and rules**
- Computation **: is deduction**
- Some features:
  - Prolog uses **logical variables**. These are **not the same as variables in other languages**.
  - Programmers can use **logical variable** as ‘**holes**’ in data structures that are **gradually filled in as computation proceeds**.

What is Prolog?

- **Unification** is a built-in term-manipulation method
  - that passes parameters, returns results, selects and constructs data structures.
- Basic control flow model: **Backtracking**
- Clauses and data have : Same form
- Relation treat arguments and results uniformly
- The **relational form** of procedures makes it possible to define **‘reversible’ procedures**.

What is Prolog?

- Clauses provide a convenient way to express
  - Case analysis
  - Nondeterminism.
- *Sometimes it is necessary to use control features that are not part of 'logic'*.
- A Prolog program can also be seen as a relational database containing rules as well as facts.

Prolog: Hello World Program

```
| ?- write('hellow World').
hellow World
yes
| ?- write("hellow World").
[104,101,108,108,111,119,32,87,111,11
4,108,100]
yes
```

Compare Prolog: Relational Database and Queries

Relation

A concrete view of relation is a table with $n \geq 0$ columns and a possible infinite set of rows

A tuple $(a_1, a_2, \ldots, a_n)$ is an relation of a_i appears in column i, $1 \leq i \leq n$, of some row in the table

Compare Prolog: Relational Database and Queries

Logic programming deal with relation rather than functions

- Based on premise the programming with relation is more flexible then programming function
- Because relation treat arguments and result uniformly
- Informally
  - Relation have no sense of direction
  - No prejudice about who is computed from whom

Prolog Relational Database: Example

[a, b, c]= [a|[b,c]] = [Head is symbol | Tail is list]

Relation **append** is a set of tuples of the form (X,Y,Z) where Z consist if X followed by the element of Y.

X	Y	Z
[]	[]	[]
[a]	[]	[a]
…	…	….
[a,b]	[c,d]	[a,b,c,d]
…..	……	………

Relation are also called ***predicates.***

Query : Is a given tuple in relation append?

```
?-append([a],[b],[a,b]).
yes
```

```
?-append([a],[b],[]).
no
```

Writing append relation in prolog

Rules

```
append([],Y,Y).
append([H|X],Y,[H|Z]):- append(X,Y,Z).
```

Queries

```
?-append ([a,b],[c,d],[a,b,c,d]).
 yes
?-append([a,b],[c,d],Z).
Z=[a,b,c,d]
?-append([a,b],Y,[a,b,c,d]).
Y=[c,d]
?-append(X,[c,d],[a,b,c,d]).
X=[a,b]
?-append(X,[d,c],[a,b,c,d]).
no
```

Writing append relation in prolog

```
/* append.pl*/
appnd([],Y,Y).
appnd([H|X],Y,[H|Z]):- appnd(X,Y,Z).
```

Queries

```
?-consult('append.pl').
?-appnd ([a,b],[c,d],[a,b,c,d]).
 yes
?-appnd([a,b],[c,d],Z).
Z=[a,b,c,d]
?-appnd([a,b],Y,[a,b,c,d]).
Y=[c,d]
?-appnd(X,[c,d],[a,b,c,d]).
X=[a,b]
```

Prolog is a ‘Declarative’ language

- Clauses are statements about **what is true about a problem**, instead of instructions how to accomplish the solution.
- The Prolog system uses the clauses to work out how to accomplish the **solution by searching through the space of possible solutions**.
- *Not all problems have pure declarative specifications. Sometimes extralogical statements are needed.*

What a program looks like

```
/* At the Zoo */
elephant(gaj).
elephant(aswasthama).

panda(chi_chi).
panda(ming_ming).
```

Facts

Rules

```
dangerous(X)  :- big_teeth(X).
dangerous(X)  :- venomous(X).
guess(X, tiger):- stripey(X), big_teeth(X),
                     isaCat(X).
guess(X, zebra):- stripey(X), isaHorse(X).
```

Example: Concatenate lists a and b

Imperative language

```
node *concat(node*list1,node *list2){
      node *p;    p=list1;
      while(p->next->next!=NULL)
                  p=p->next;
      p->next=list2;
      return(list1);
}
```

functional language

```
cat(a,b) ≡
 if b = nil then a
else cons(head(a), cat(tail(a),b))
```

Declarative language

```
cat([], Z, Z).
cat([H|T], L, [H|Z]) :- cat(T, L, Z).
```

Factorial Program

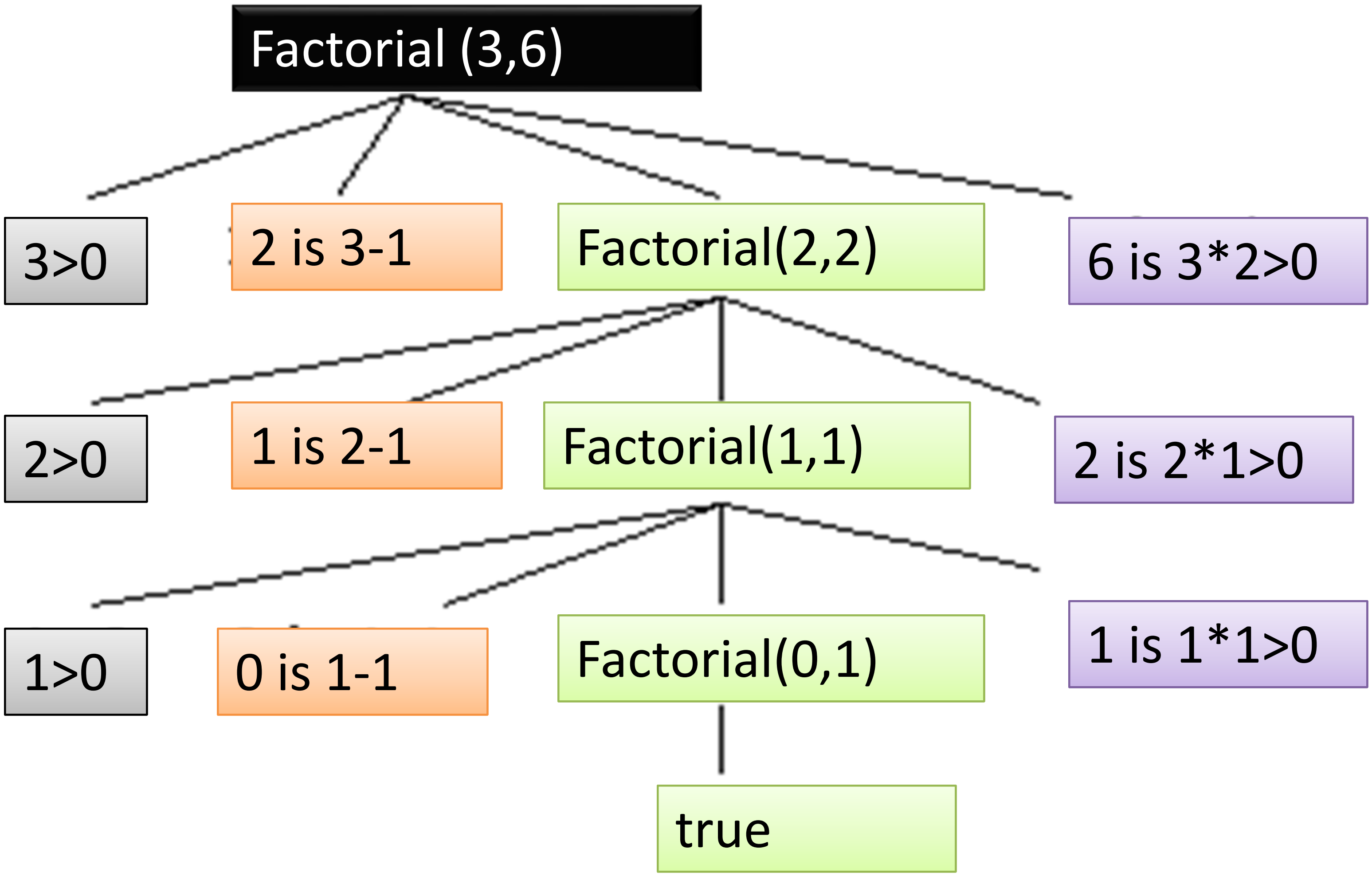
```
factorial(0,1).
factorial(N,F):- N>0, N1 is N-1,
    factorial(N1,F1),F is N * F1.
```

The Prolog goal to calculate the factorial of the number 3 responds with a value for W, the goal variable:

```
?- factorial(3,W).
   W=6
```

Factorial Program Evaluation

```
factorial(0,1).
factorial(N,F):- N>0, N1 is N-1, factorial(N1,F1),F is N*F1.
```



Complete Syntax of Terms

Term

Constant

Names an individual

Atom

alpha17
gross_pay
john_smith
dyspepsia
+
=/=
'12Q&A'

Number

0
1
57
1.618
2.04e-27
-13.6

Compound Term

Names an individual that has parts

likes(john, mary)
book(dickens, Z, cricket)
f(x)
[1, 3, g(a), 7, 9]
-(+(15, 17), t)
15 + 17 - t

Variable

Stands for an individual unable to be named when program is written

X
Gross_pay
Diagnosis
_257
_

General Rules

- variable start with
  - Capital letter or underscore
  - Mostly we use Capital X,Y,Z,L,M for variable
- atom start with
  - Mostly word written in small letters
  - likes, john, mary in likes (john, mary).
  - elephant gaj in elephant(gaj).

Compound Terms

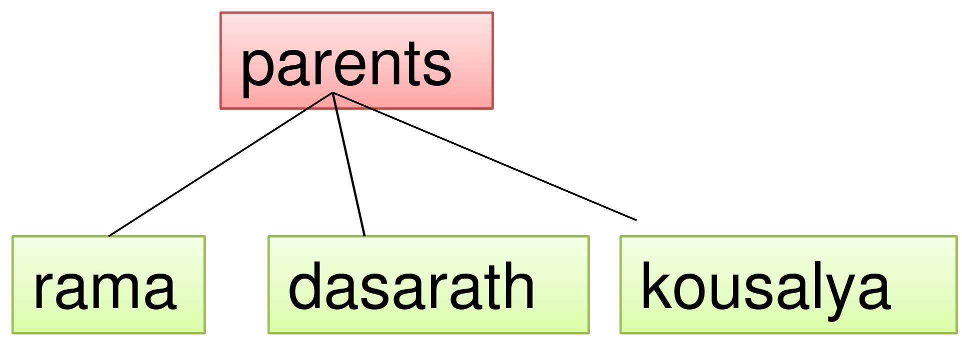
The parents of Rama are Dasarath and Kousalya.

```
parents(rama, dasarath, kousalya)
```

Functor (an atom) of arity 3.

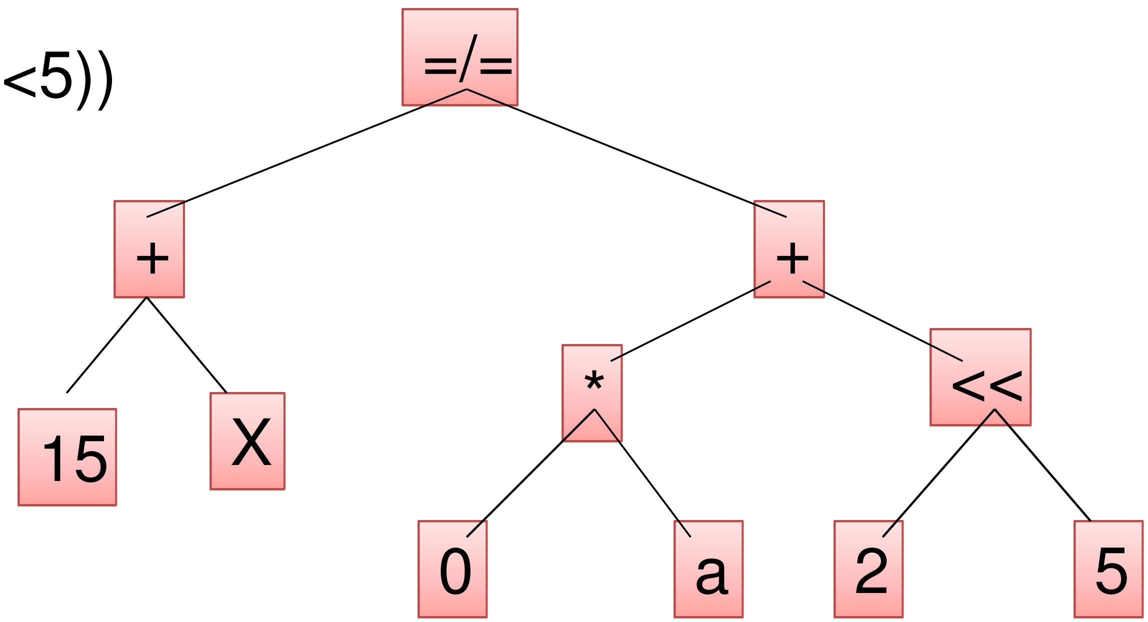
components (any terms)

It is possible to depict the term as a tree:



Compound Terms: Example

=/=(15+X, (0*a)+(2<<5))



X =\= Y **means X and Y stands for different numbers**

More about operators

- Any atom may be designated an operator. The only purpose is for convenience; the only effect is how the term containing the atom is parsed.
- Operators are ‘**syntactic sugar**’.
  - Easy to write in our own way
- Operators have three properties: position, precedence and associativity.

Examples of operator properties

Position	Operator Syntax	Normal Syntax
Prefix:	-2	-(2)
Infix:	5+17	+(17,5)
Postfix:	N!	!(N)

Associativity: left, right, none.
X+Y+Z is parsed as (X+Y)+Z
because addition is left-associative.

Precedence: an integer.
X+Y*Z is parsed as X+(Y*Z)
because multiplication has higher precedence.

These are all the same as the normal rules of arithmetic.

Logical Operation on Numbers

X =:= Y	X and Y stands for the same number
X =\= Y	X and Y stands for different numbers
X < Y	X is less then Y
X > Y	
X =< Y	**Not same as in C (>=, <=)**
X >= Y	

The last point about Compound Terms…

Constants are simply compound terms of arity 0.

badger   means the same as   badger()

Structure of Prolog Programs

- Programs consist of procedures.
- Procedures consist of clauses.
- Each clause is a fact or a rule.
- Programs are executed by posing queries.

Prolog Program: An Example

Predicate

Procedure for elephant

Facts

Clauses

elephant(**gaj**).
elephant(**aswasthama**).
elephant(X) :- grey(X), mammal(X), hasTrunk(X).

Rule

Example

Queries

Replies

```
?- elephant(gaj).
```

yes

```
?- elephant(arjun).
```

no

Clauses: Facts and Rules

‘if’
‘provided that’
‘turnstile’

Head **:-** *Body***.** This is a rule.

*Head***.** This is a fact.

Full stop at the end.

Body of a (rule) clause contains goals.

Head *Body*

likes(mary, X) :- human(X), honest(X).

Goals

Interpretation of Clauses

Clauses can be given a declarative reading or a procedural reading.

Form of clause:

HORN Clause

$H \;\; :- \;\; G_1, \;\; G_2, \;\; \ldots, \;\; G_n.$

Declarative reading:

"That H is provable follows from goals G_1, G_2, …, G_n being provable."

Procedural reading:

"To execute procedure H, the procedures called by goals G_1, G_2, …, G_n are executed first."

Another Example

Program

```
male(bertram).
male(percival).

female(lucinda).
female(camilla).

pair(X,Y):-male(X),
           female(Y).
```

Queries

```
?- pair(percival, X).
?- pair(apollo,daphne).
?- pair(camilla, X).
?- pair(X, lucinda).
?- pair(X, X).
?-pair(bertram,lucinda).
?- pair(X, daphne).
?- pair(X, Y).
```

Example 2

```
drinks(john,martini).
drinks(mary, gin).
drinks(susan, vodka).
drinks(john, gin).
drinks(fred, gin).

pair(X, Y, Z) :-
     drinks(X, Z),
     drinks(Y, Z).
```

```
?- pair(X,john,martini).
?- pair(mary,susan,gin).
?- pair(john,mary,gin).
?- pair(john,john,gin).
?- pair(X,Y, gin).
?- pair(bertram,
    lucinda, vodka).
?- pair(X, Y, Z).
```

This definition forces X and Y to be distinct:

```
pair(X,Y,Z):- drinks(X,Z), drinks(Y,Z), X \== Y.
```

Another Examples: Density Calculation

```
%popultaion.pl  Population in Million
pop(usa,280).     pop(india,1000).
pop(china,1200). pop(brazil,130).
area(usa,3). /* millions of sq miles */
area(india,1). area(china,4).
area(brazil,3).

density(X,Y) :- pop(X,P),
                area(X,A),
                Y is P/A.
```

The population density of country X is Y, if:

The population of X is P, **and**

The area of X is A, **and**

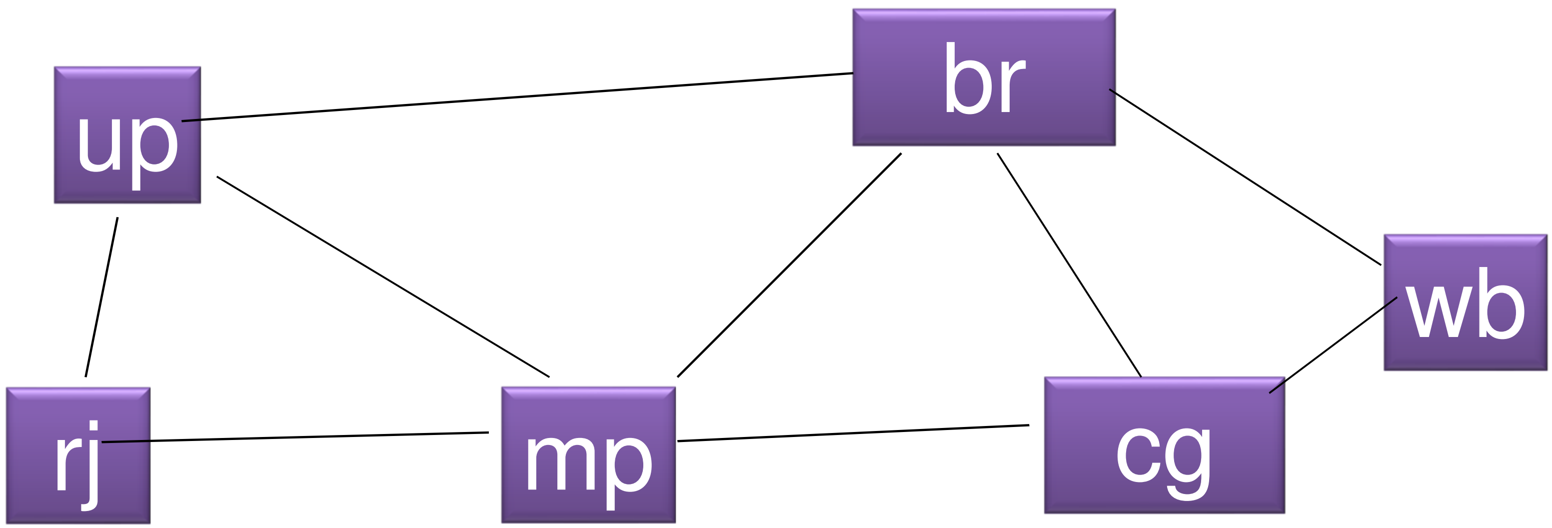
Y is calculated by dividing P by A.

Examples: Density Calculation

```
?- consult(population.pl).
% population compiled 0.00 sec,
  1,548 bytes
Yes
?- density(usa,D).
D = 93.3333
Yes
?- density(china,D).
D = 300
Yes
```

Example 3: Boarder of Indian States

(a) Representing a symmetric relation.
(b) Implementing a strange ticket condition.



How to represent this relation?
Note that borders are symmetric.

Example 3: Boarder of India States

This relation represents one 'direction' of border:

```
border(cg, wb).
border(cg, br).
border(br, wb).
border(mp, cg).
border(mp, br).
border(mp, up).
border(up, br).
border(rj, mp).
border(rj, up).
```

What about the other?

(a) Say `border(wb, cg).`

```
border(cg, wb).
⋮
```

(b) Say

```
adjacent(X,Y):-border(X,Y).
adjacent(X,Y):-border(Y,X).
```

(c) Say

```
border(X,Y):-border(Y,X).
```

Example 3: Boarder of India States

Now a somewhat strange type of discount ticket. For the ticket to be valid, one must pass through an intermediate state.

A valid ticket between a start and end state obeys the following rule:

```
valid(X,Y):-adjacent(X,Z),adjacent(Z,Y)
```

Example 3: Boarder of India States

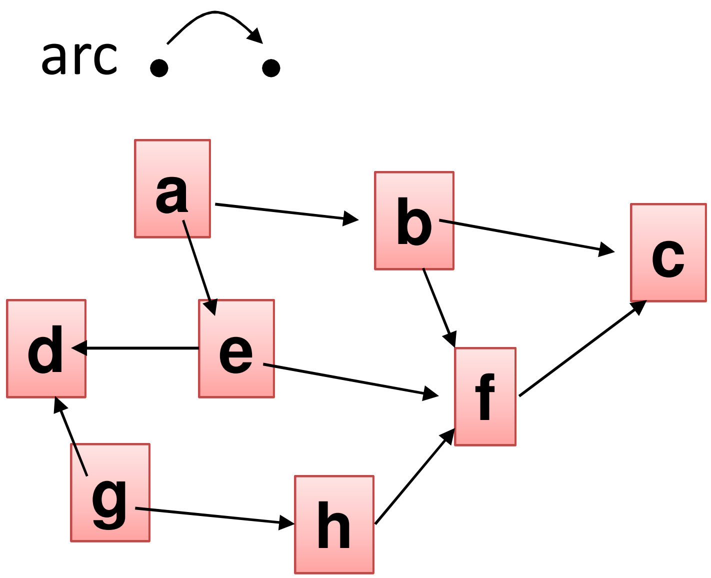
```
border(cg, wb).
border(cg, br).
border(br, wb).
border(mp, cg).
border(mp, br).
border(mp, up).
border(up, br).
border(rj, mp).
border(rj, up).

adjacent(X, Y) :- border(X, Y).
adjacent(X, Y) :- border(Y, X).
```

```
valid(X, Y) :-
        adjacent(X, Z),
        adjacent(Z, Y)
```

```
?- valid(rj, cg).
?- valid(rj, wb).
?- valid(mp, mp).
?- valid(X, wb).
?- valid(cg, X).
?- valid(X, Y).
```

Graph Example

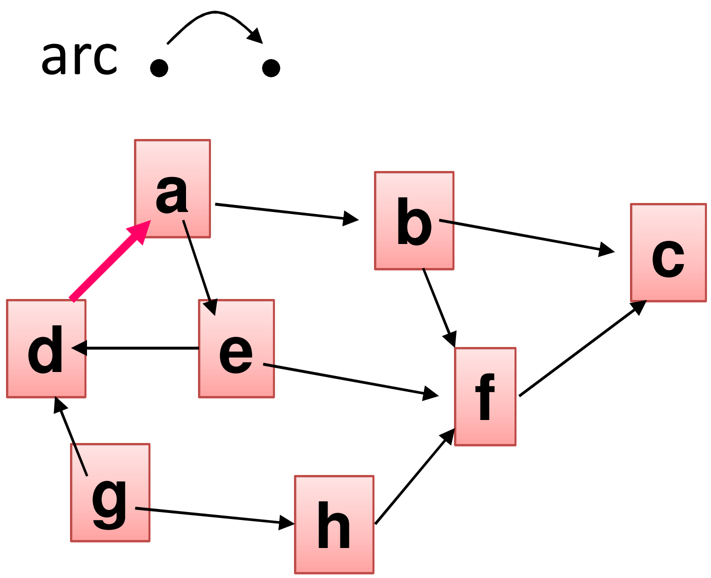



```
a(g, h). a(g, d). a(e, d).
a(h, f).  a(e, f).  a(a, e).
a(a, b). a(b, f).  a(b, c). a(f, c).
path(X, X).
path(X, Y) :- a(X, Z), path(Z, Y).
```

Prolog can distinguish between the 0-ary constant a *(the name of a node) and the 2-ary functor* a *(the name of a relation).*

```
?- path(f, f).
?- path(a, c).
?- path(g, e).
?- path(g, X).
?- path(X, h).
```

But what happens if...




```
a(g, h). a(g, d). a(e, d).
a(h, f).  a(e, f).  a(a, e).
a(a, b). a(b, f).  a(b, c). a(f, c).
a(d, a).
path(X, X).
path(X, Y) :- a(X, Z), path(Z, Y).
```

This program works only for acyclic graphs.
The program may infinitely loop given a cyclic graph.

We need to leave a 'trail' of visited nodes
== > (to be seen later).

Unification

- Two terms unify
  - if substitutions can be made for any variables in the terms so that the terms are made identical.
  - If no such substitution exists, the terms do not unify.
- The Unification Algorithm proceeds by recursive descent of the two terms.
  - Constants unify if they are identical
  - Variables unify with any term, including other variables
  - Compound terms unify if their functors and components unify.

Unification Examples

```
| ?- X=1+2.
     X = 1+2
     yes
| ?- f(g(Y))=f(X).
  X = g(Y)
  yes
| ?- X=f(Y).
  X = f(Y)
      yes
```

Unification Examples: 1

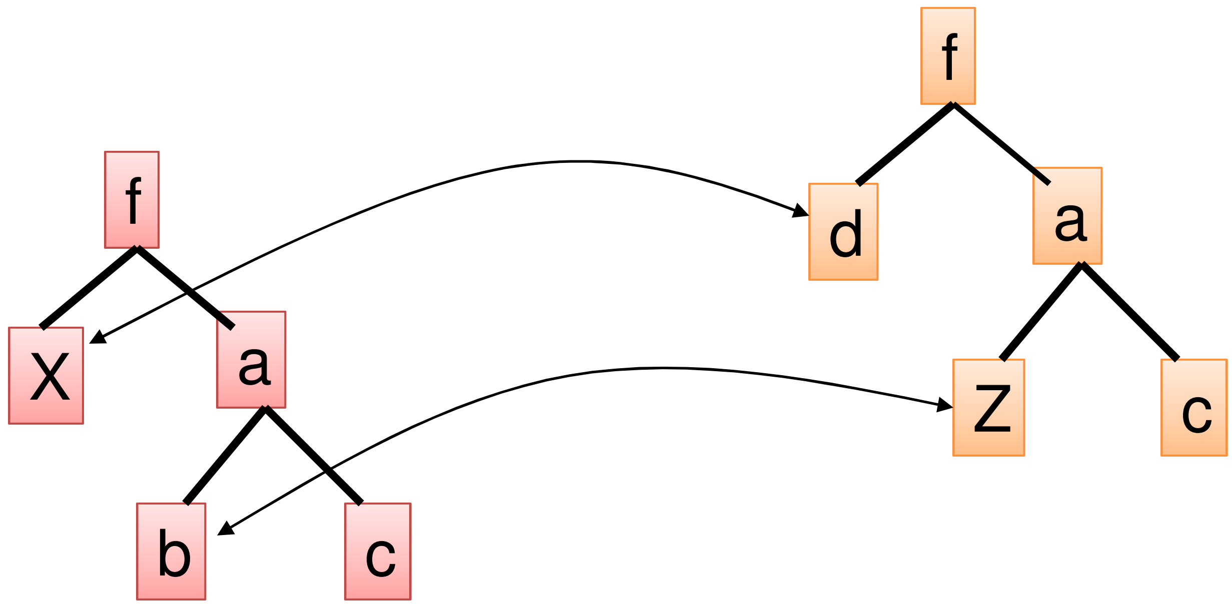
The terms f(X, a(b,c)) and f(d, a(Z, c)) unify.

| ?- f(X, a(b,c)) = f(d, a(Z, c)).

X = d

Z = b

yes



The terms are made equal if d is substituted for X, and b is substituted for Z.
We also say X is instantiated to d and Z is instantiated to b, or X/d, Z/b.

Unification : Examples 2

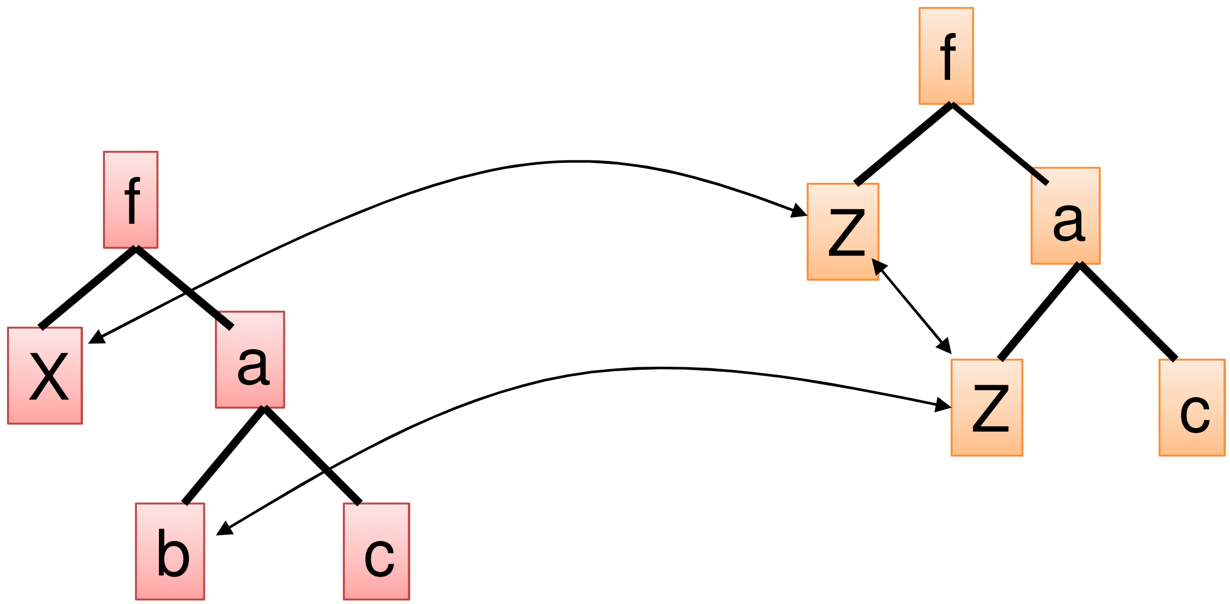
The terms f(X, a(b,c)) and f(Z, a(Z, c)) unify.

| ?- f(X, a(b,c)) =f(Z, a(Z, c)).

X = b

Z = b

yes

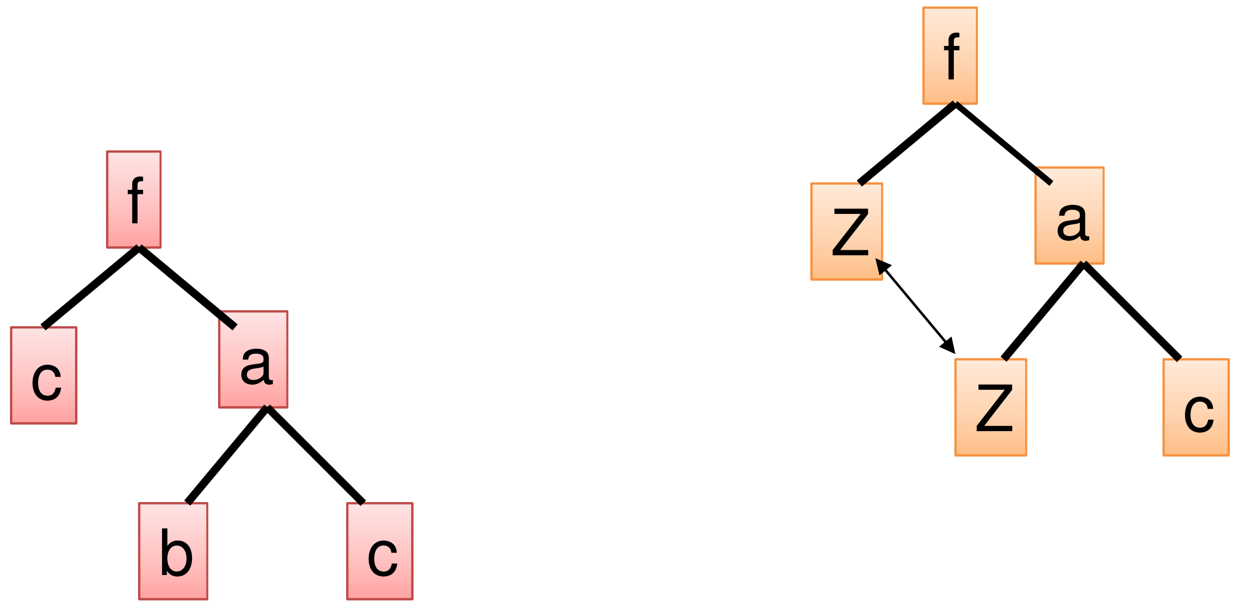


Note that Z co-refers within the term.
Here, X/b, Z/b.

Unification : Examples 3

The terms f(c, a(b,c)) and f(Z, a(Z, c)) do not unify.

```
| ?- f(c,a(b,c))=f(Z,a(Z,c)).
no
```

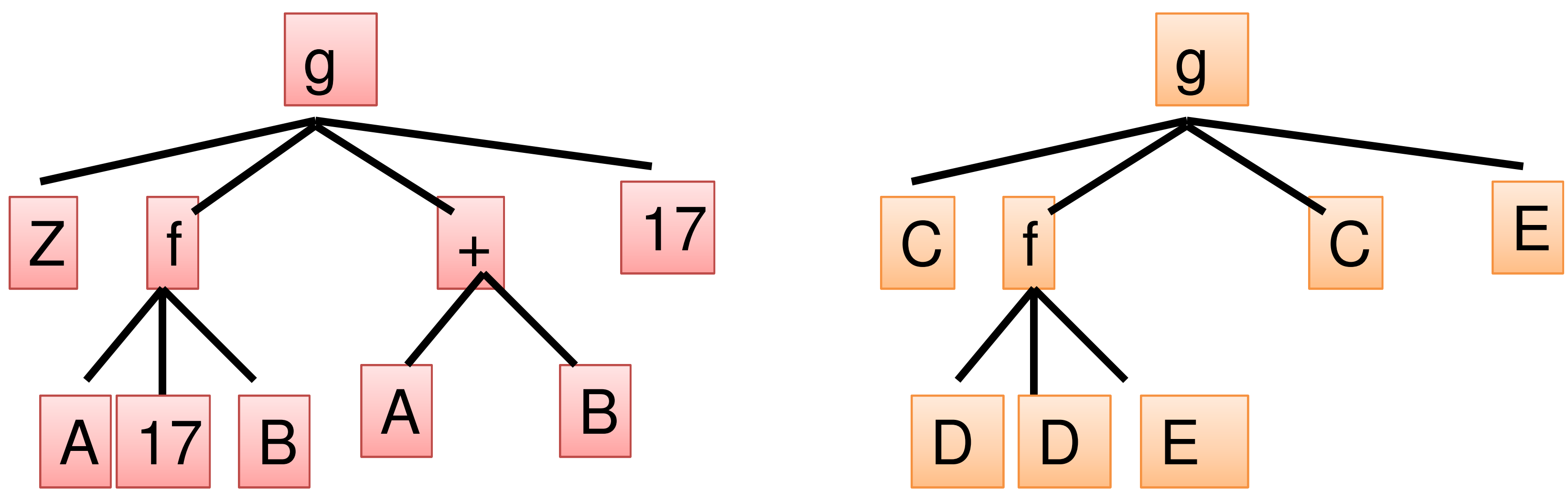


No matter how hard you try, these two terms cannot be made identical by substituting terms for variables.

Unification: Big Example

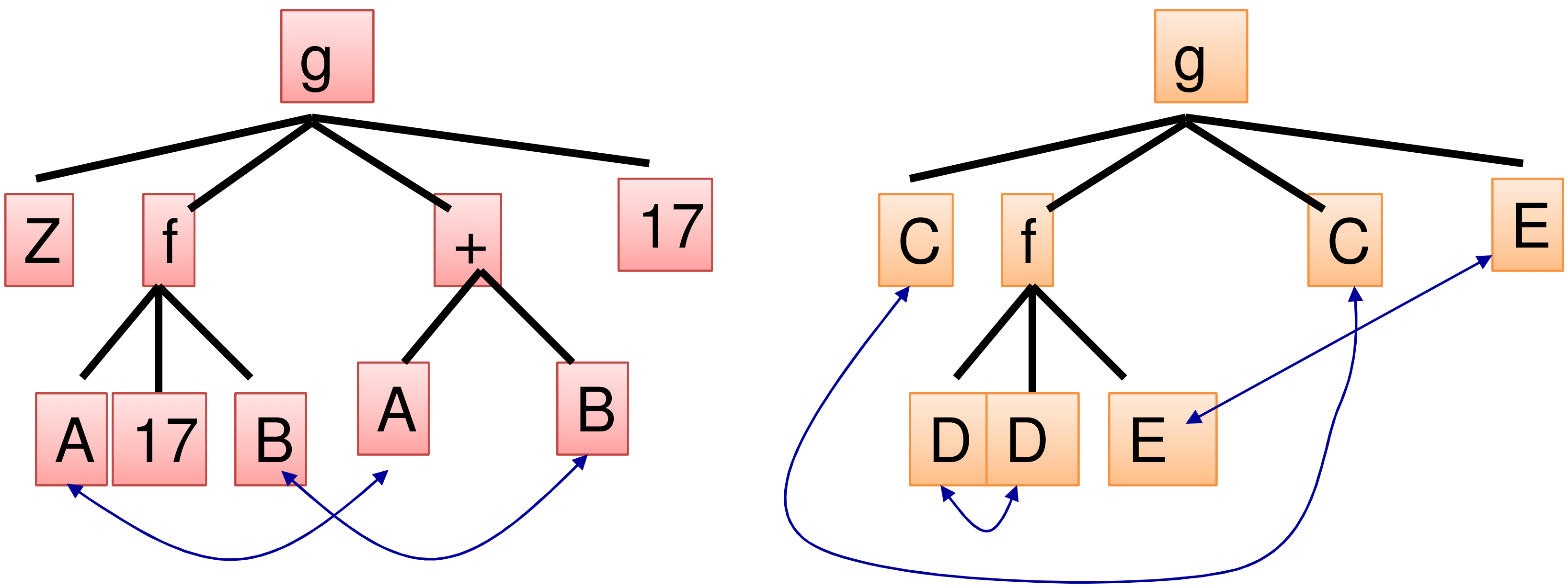
Do terms g(Z, f(A, 17, B), A+B, 17) and g(C, f(D, D, E), C, E) unify?

```
| ?- g(Z,f(A,17,B),A+B,17)=g(C,f(D,D,E),C,E).
A = 17  B = 17  C = 17+17 D = 17 E = 17 Z = 17+17
yes
```



Unification: Big Example

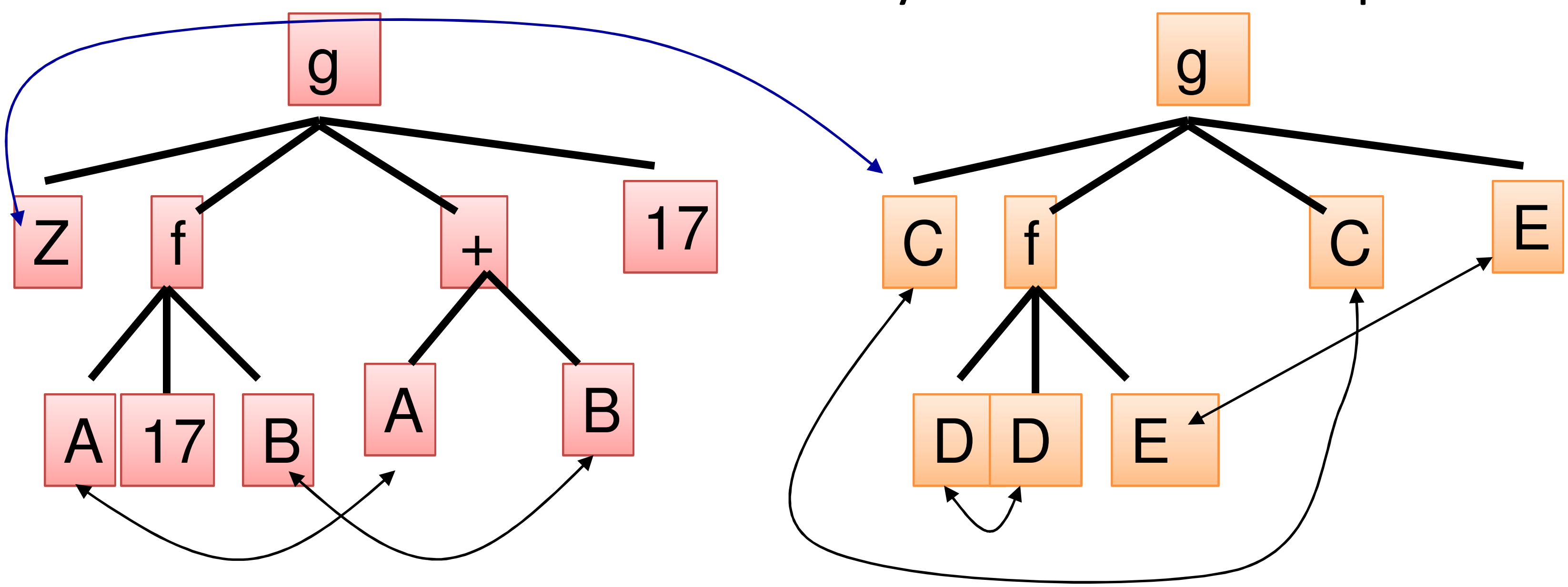
First write in the co-referring variables.



Unification: Big Example

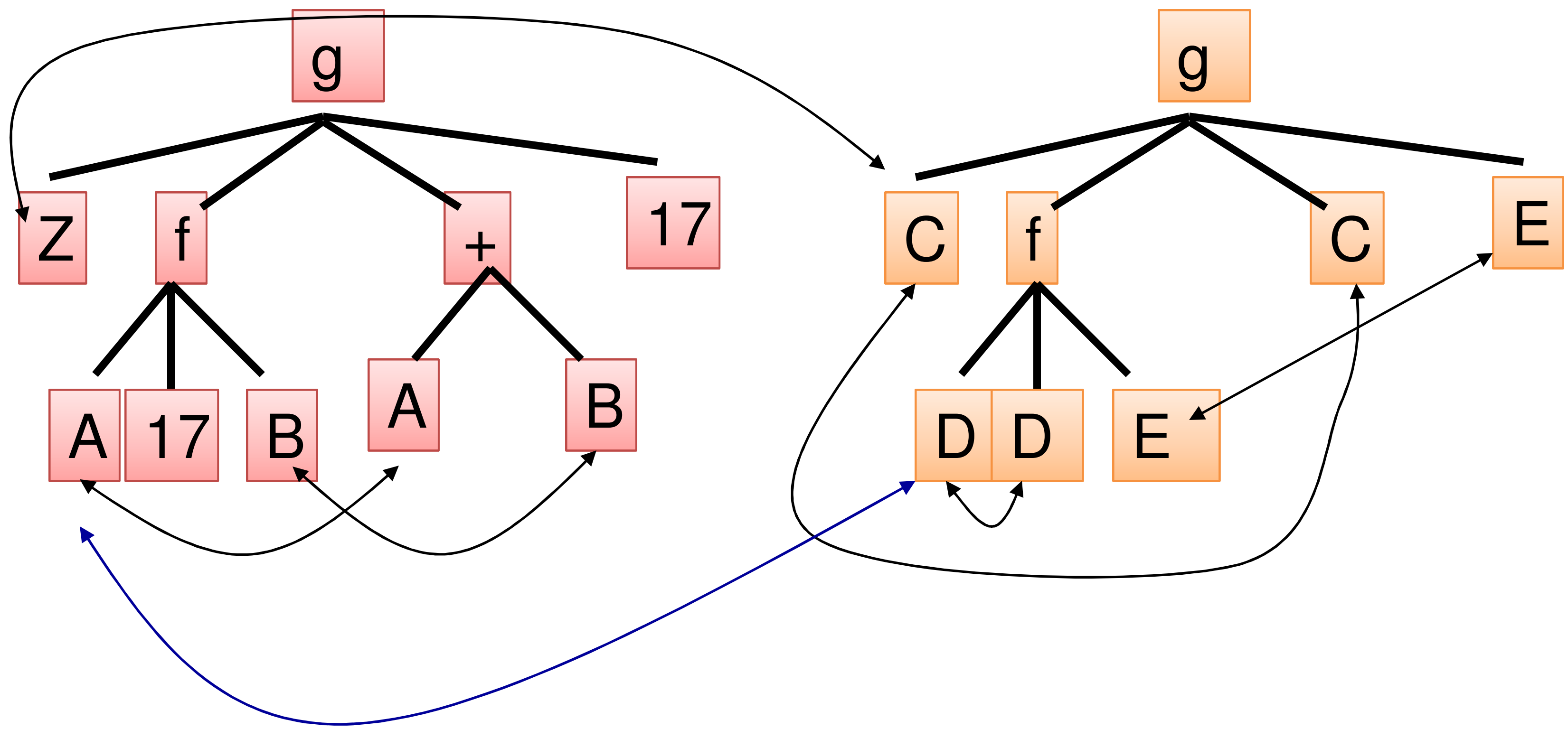
Z/C, C/Z

Now proceed by recursive descent
We go top-down, left-to-right, but the order does not matter as long as it is systematic and complete.



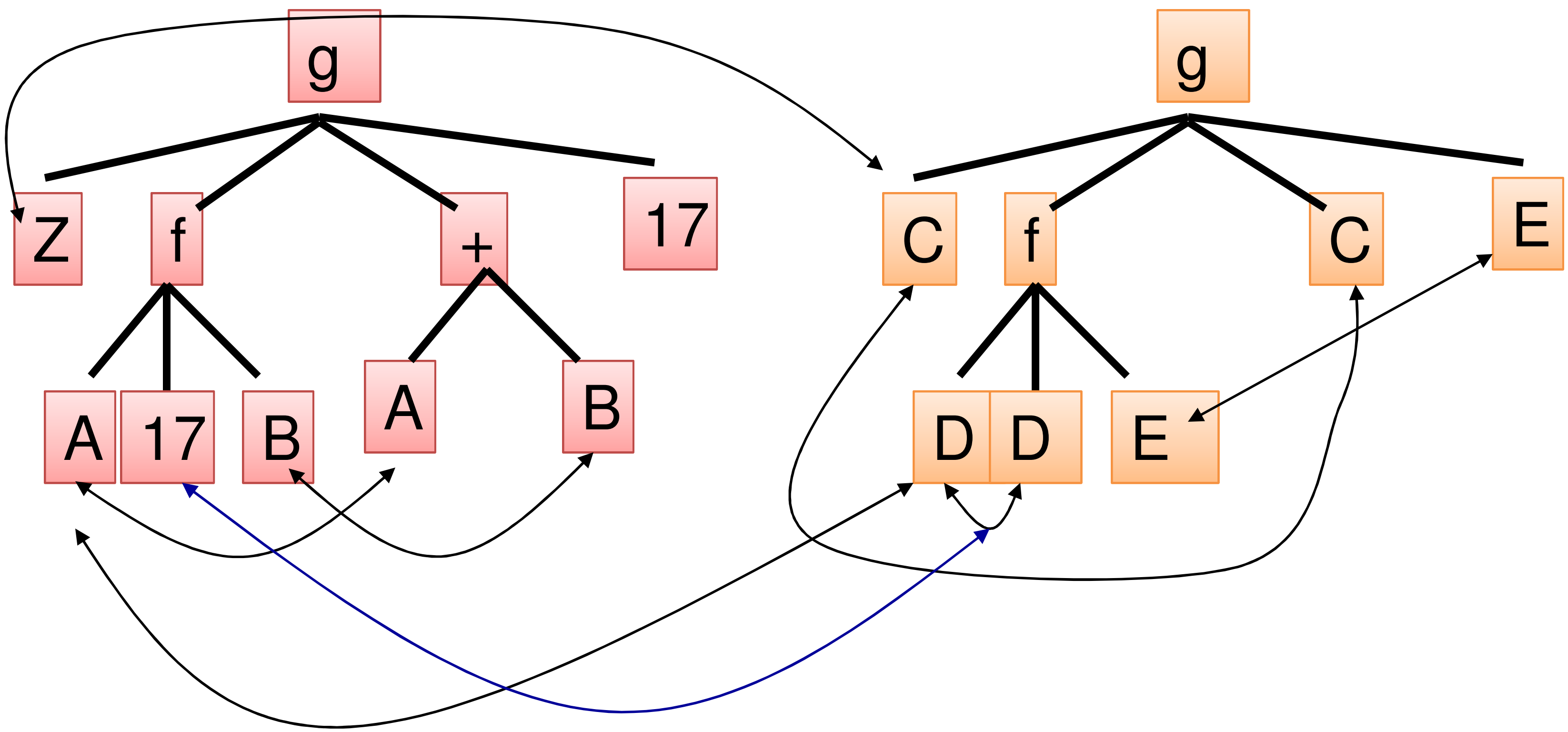
Unification: Big Example

Z/C, C/Z, A/D, D/A



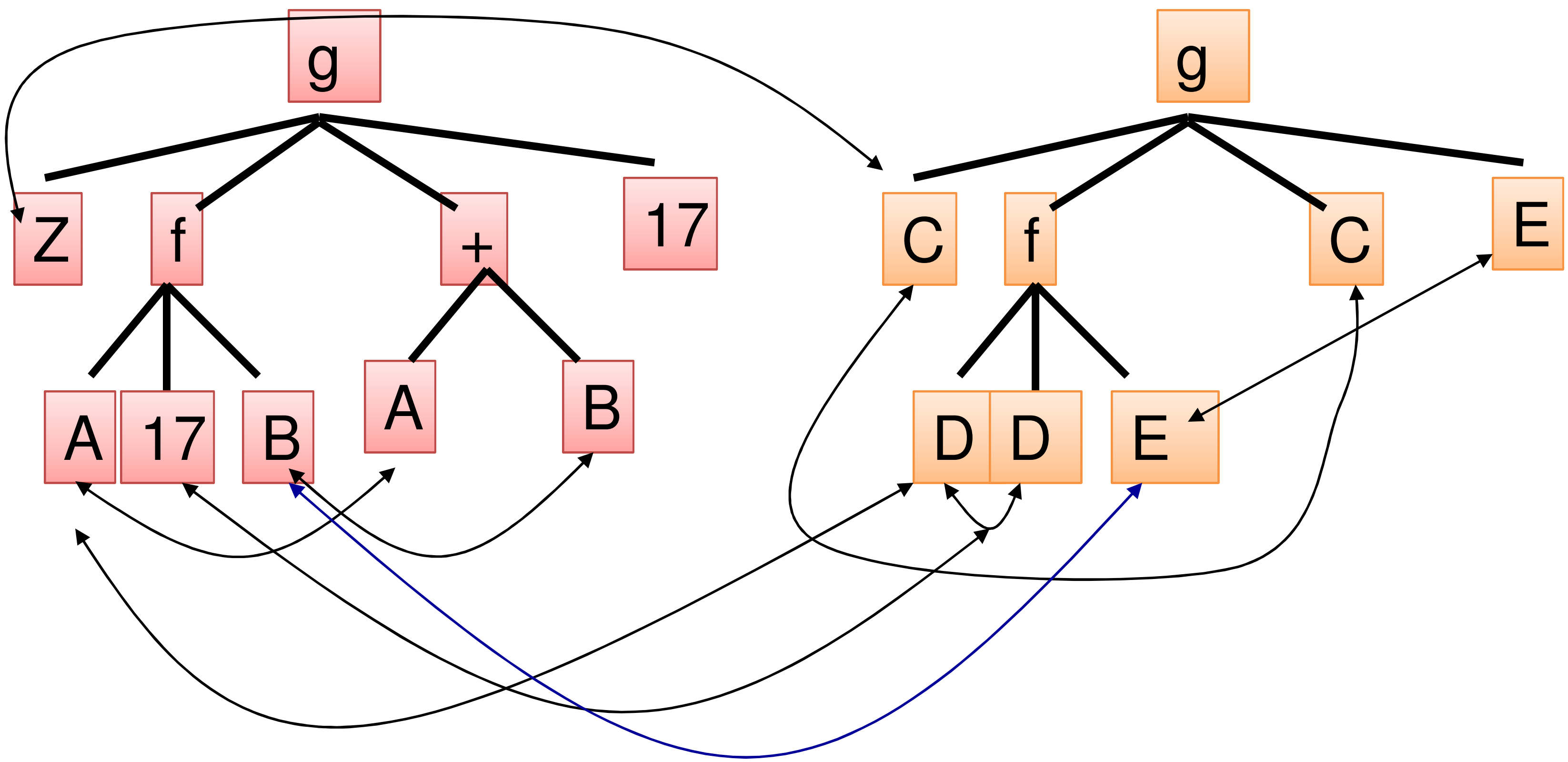
Unification: Big Example

Z/C, C/Z, A/17, D/17



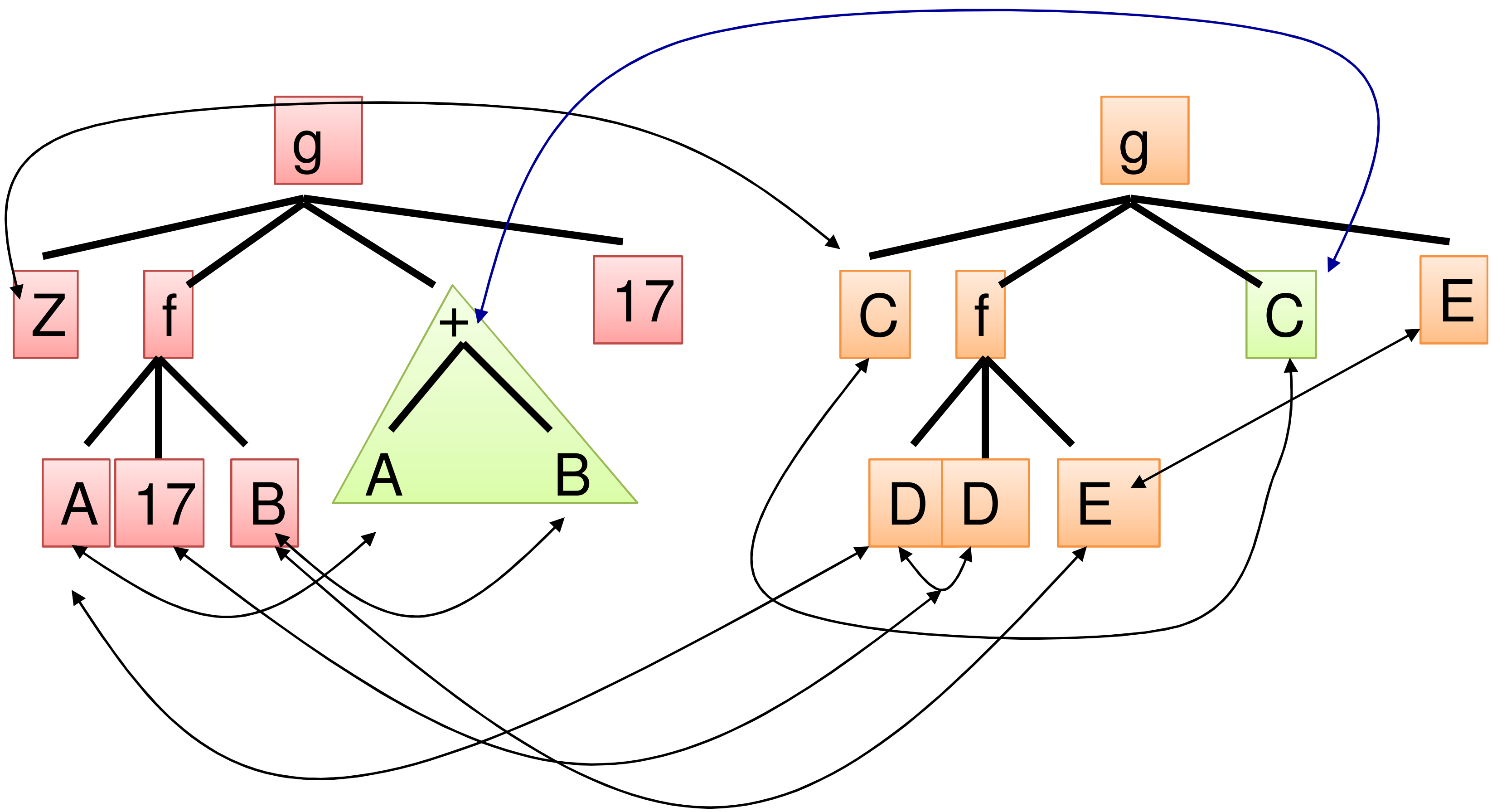
Unification: Big Example

Z/C, C/Z, A/17, D/17, B/E, E/B



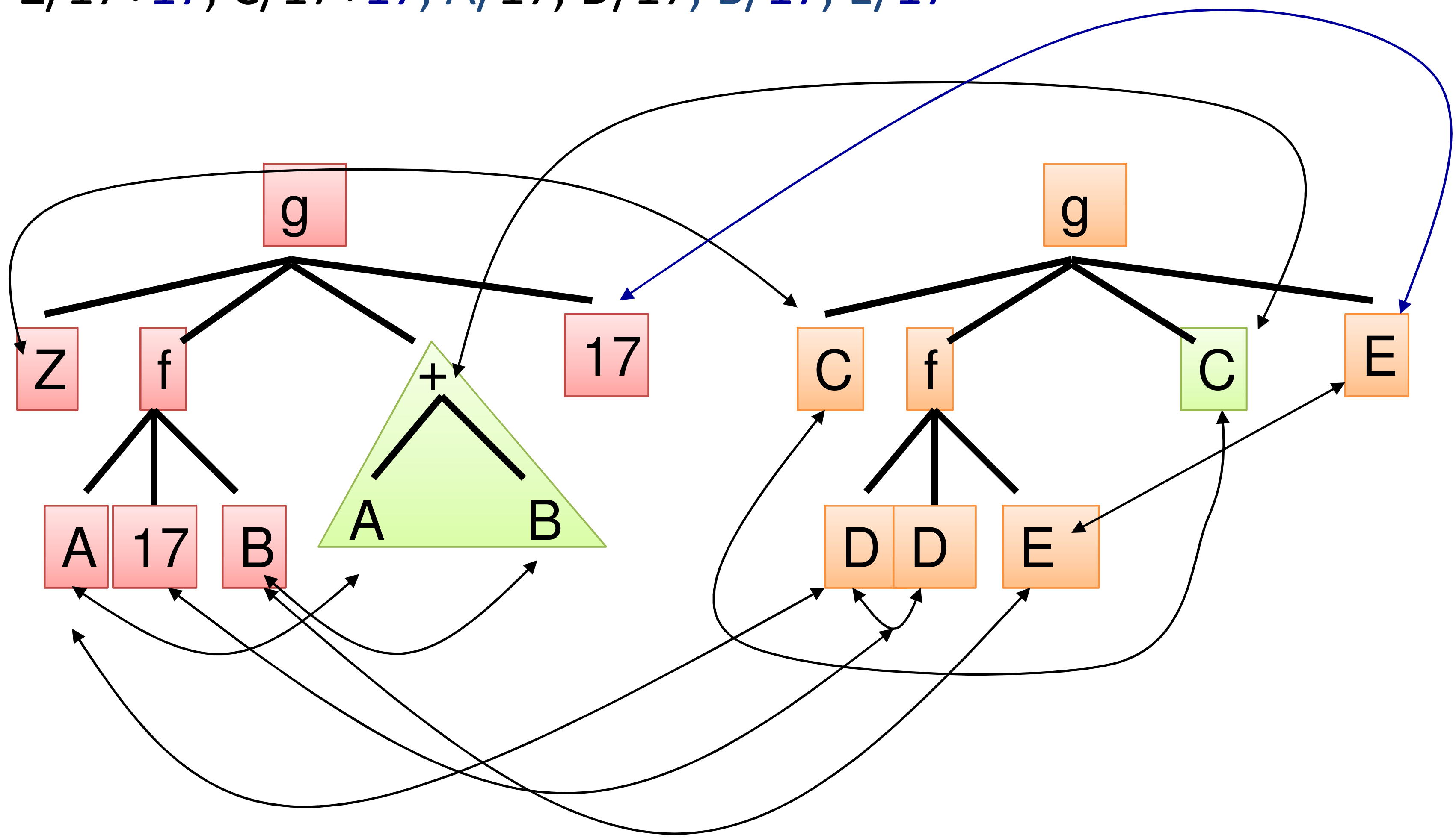
Unification: Big Example

Z/17+B, C/17+B, A/17, D/17, B/E, E/B



Unification: Big Example

Z/17+17, C/17+17, A/17, D/17, B/17, E/17



Thanks