Prolog

But first: A cautionary tale about sources

http://zapatopi.net/treeoctopus/

Prolog

- SWI-Prolog available at: http://www.swi-prolog.org
- Useful to try examples (available on web site)
- Next slides [grey background not mine] from http://computing.unn.ac.uk/staff/cgdh1/teaching/CM033APM/Lectures/L2.PPT

A PROLOG Program

- A PROLOG program is a set of ***facts*** and ***rules***.
- A simple program with just facts :

```
parent(alice, jim).
parent(jim,   tim).
parent(jim,   dave).
parent(jim,   sharon).
parent(tim,   james).
parent(tim,   thomas).
```

- Each line is a ***fact*** (a.k.a. a tuple or a row).
- Each line states that some person `X` is a parent of some (other) person `Y`. [KB: What assumptions are being made here?]
- In GNU PROLOG [KB: and SWI Prolog] the program is kept in an ASCII file.

Running a PROLOG Program

- First, run GNU [KB: or SWI] PROLOG :

```
$ gprolog
GNU Prolog 1.2.1
By Daniel Diaz
Copyright (C) 1999,2000 Daniel Diaz
| ?-
```

- The last line is the PROLOG prompt. Next, load the file containing the program (***consulting*** the program) :

```
| ?- [lect1].     % File is called lect1.pl
compiling ...
yes
| ?-
```

- PROLOG always say `yes` if something works.

A PROLOG Query

- Now we can ask PROLOG questions :

```
| ?- parent(alice, jim).
yes
| ?- parent(jim, herbert).
no
| ?-
```

- Not very exciting. But what about this :

```
| ?- parent(alice, Who).
Who = jim
yes
| ?-
```

- `Who` is called a ***logical variable***.
  - PROLOG will set a logical variable to any value which makes the query succeed.

A PROLOG Query II

- Logical variables start with an upper case letter. Anything that starts with a lower case letter is a literal constant.
- Sometimes there is more than one correct answer to a query.
- PROLOG gives the answers one at a time. To get the next answer type `;`.

```
| ?- parent(jim, Who).
Who = tim ? ;
Who = dave ? ;
Who = sharon ? ;
yes
| ?-
```

NB : The `;` do not actually appear on the screen.

- After finding that `jim` was a parent of `sharon` GNU PROLOG detects that there are no more alternatives for `parent` and ends the search.

A PROLOG Search Space

- The easiest way to understand how a PROLOG program works is to draw a ***search space***.
  - Sometimes called a ***search tree***.
  - A pictorial representation of ***Robinson's Unification Algorithm*** in action.
- PROLOG is not very bright - it simply searches the program from top to bottom [KB: Does this remind anyone of DFS?] checking each fact in turn.
- For the query to succeed PROLOG must be able to make the query the same as the fact it is checking against.
  - The query and the fact must ***MATCH*** ***[KB: UNIFY]***.
- C marks a choice point : a place where PROLOG has more than one way of obtaining a match.
  - Choice points are used in ***backtracking***.

A PROLOG Search Space II

- Match 1 : `alice \= jim, jim = Who.`
  - Since `Who` is a logical variable PROLOG can set it to any value required to obtain a match, in this case `Who` is set to `jim`.
  - This is no help as `alice` and `jim` are different literal constants. There is no way they can be made equal.
  - The match ***FAILS***.
- PROLOG automatically ***backtracks*** to the closest choice point (`C`) and checks the next fact for a match.
- Match 2 : `jim = jim, tim = Who.`
  - `jim` occurs in both the query and the fact. A constant always matches with itself.
  - Since `Who` is a logical variable PROLOG can set it to any value required to obtain a match, in this case `Who` is set to `tim`.
  - The match ***SUCCEEDS***.

A PROLOG Search Space III

- At this point PROLOG prints

```
Who = tim ?
```

- PROLOG is asking whether we want any more answers. If we press `RETURN` the search stops and we get the normal PROLOG prompt. If we press `;` the search continues (we ***force backtracking*** to the closest choice point).
- Match 3 : `jim = jim`, `dave = Who`.
  - `jim` occurs in both the query and the fact. A constant always matches with itself.
  - Since `Who` is a logical variable PROLOG can set it to any value required to obtain a match, in this case `Who` is set to `dave`.
  - The match SUCCEEDS. PROLOG prints

```
Who = dave ?
```

A PROLOG Search Space IV

- Match 4 : `jim = jim, sharon = Who.`
  - `jim` occurs in both the query and the fact. A constant always matches with itself.
  - Since `Who` is a logical variable PROLOG can set it to any value required to obtain a match, in this case `Who` is set to `sharon`.
  - The match SUCCEEDS. PROLOG prints

    ```
    Who = sharon ?
    ```
- Match 5 : `tim \= jim, james = Who.`
  - `tim and jim` are two different constants. They do not match.
  - Since `Who` is a logical variable PROLOG can set it to any value required to obtain a match, in this case `Who` is set to `james`.
  - The match FAILS.

A PROLOG Search Space V

- PROLOG automatically backtracks to `C` to find another alternative.
- Match 6 : `tim \= jim`, `thomas = Who`.
  - `tim and jim` are two different constants. They do not match.
  - Since `Who` is a logical variable PROLOG can set it to any value required to obtain a match, in this case `Who` is set to `thomas`.
  - The match FAILS.
- PROLOG automatically backtracks to `C` to find another alternative.
- There are no more alternatives at `C`. They have all been tried.
- The overall query FAILS.
  - Remember, it has already had three successes.
  - GNU PROLOG prints `yes`. Some implementations would print `no` in this case.

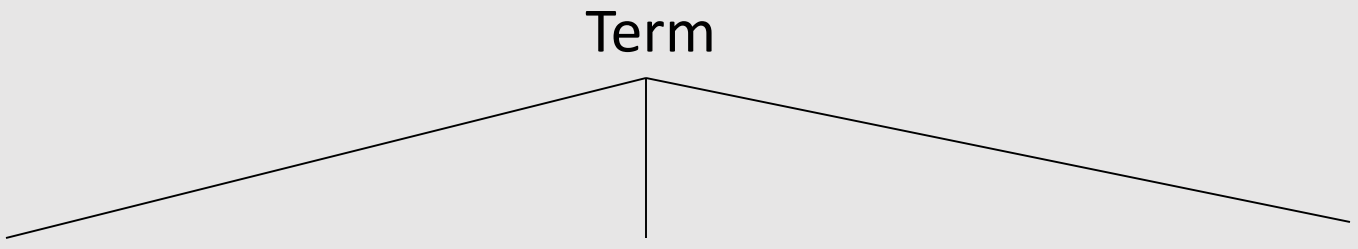
Clauses and Unification

From https://www.cl.cam.ac.uk/teaching/2001/PrologAI/PLVol1.ppt

Prolog is a ‘declarative’ language

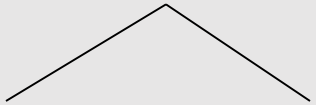
- Clauses are statements about what is true about a problem, instead of instructions how to accomplish the solution.
- The Prolog system uses the clauses to work out how to accomplish the solution by searching through the space of possible solutions.
- Not all problems have pure declarative specifications.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

Complete Syntax of Terms



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]

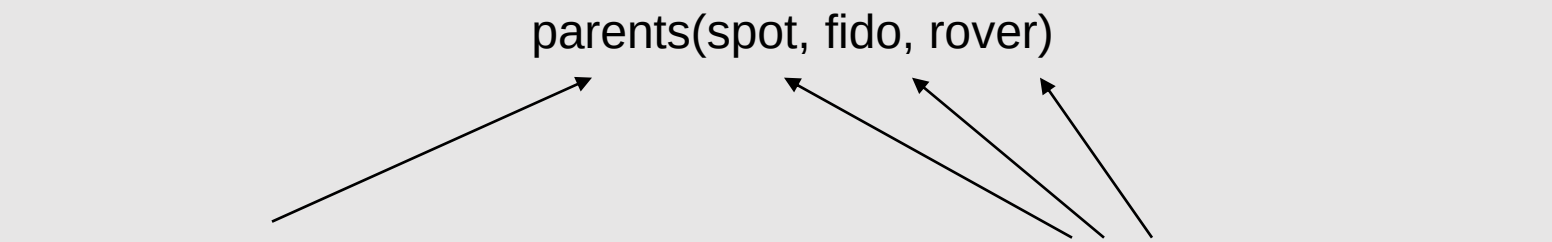
Variable

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

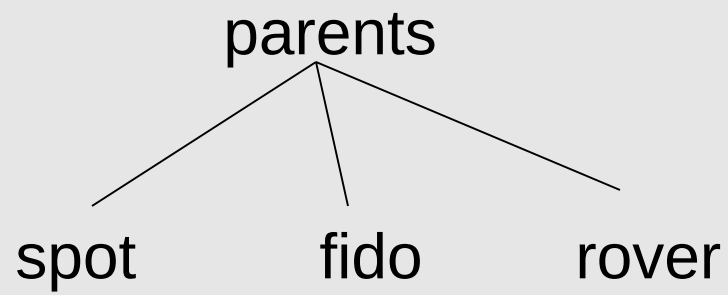
X
Gross_pay
Diagnosis
_257
_

Compound Terms

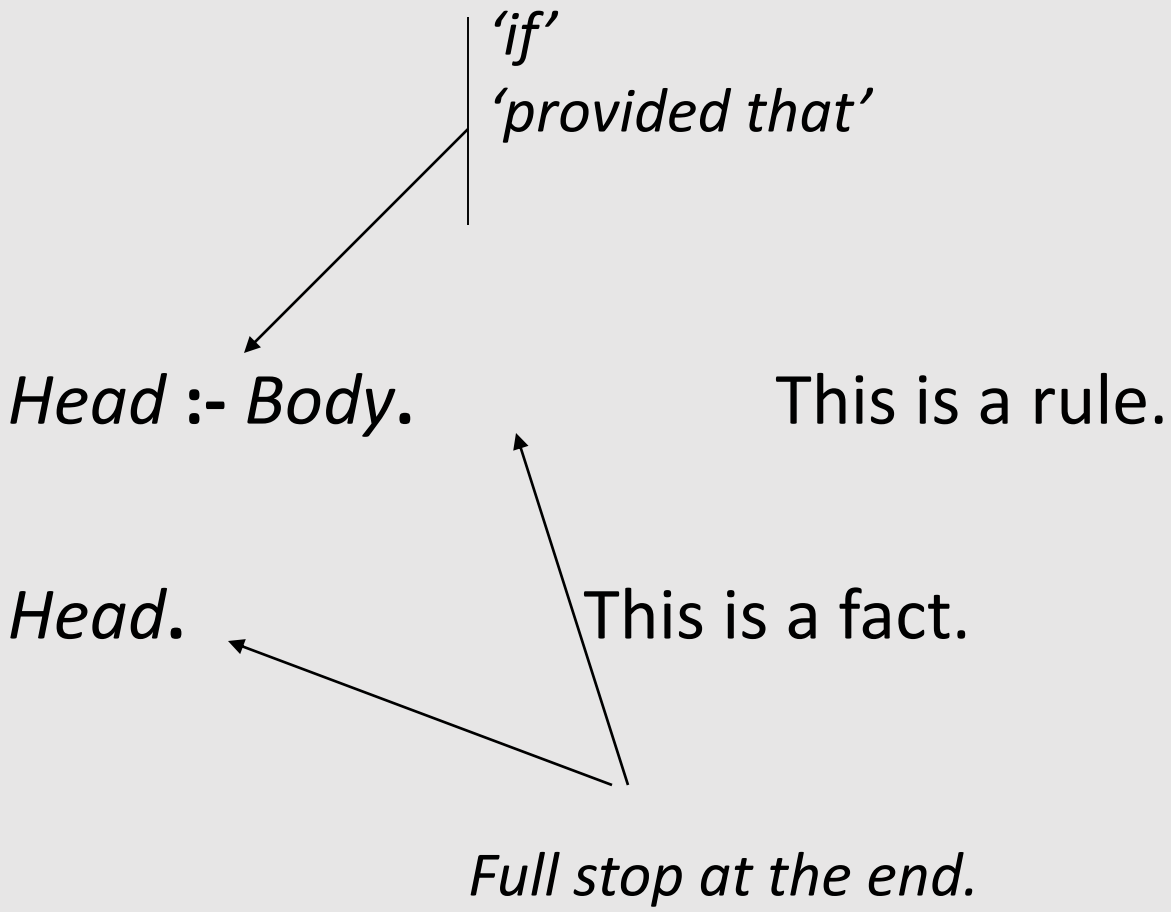
The parents of Spot are Fido and Rover.



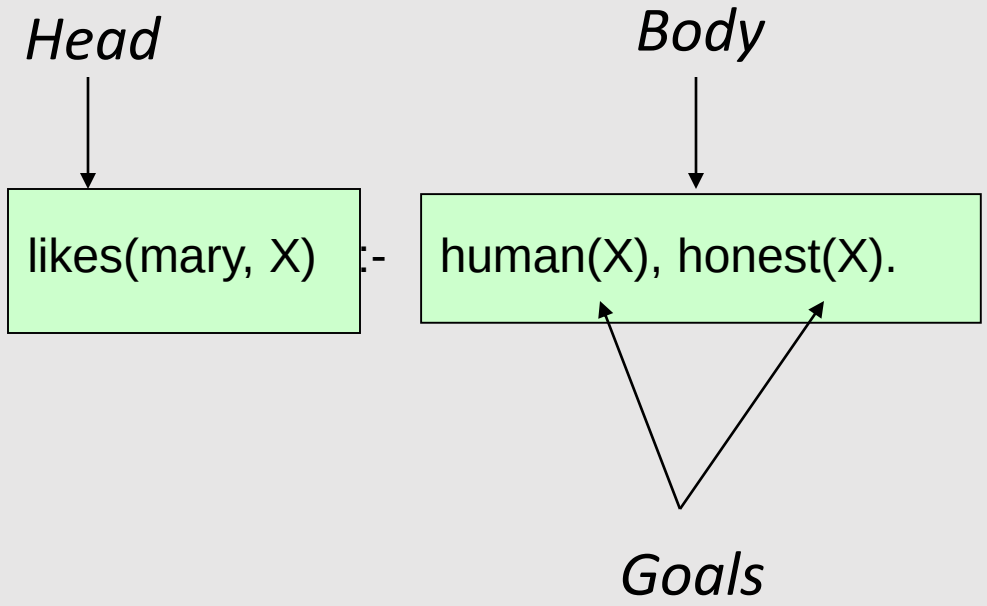
It is possible to depict the term as a tree:



Clauses: Facts and Rules



Body of a (rule) clause contains goals.



Interpretation of Clauses

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

Form of clause: H :- G_1, G_2, …, G_n.

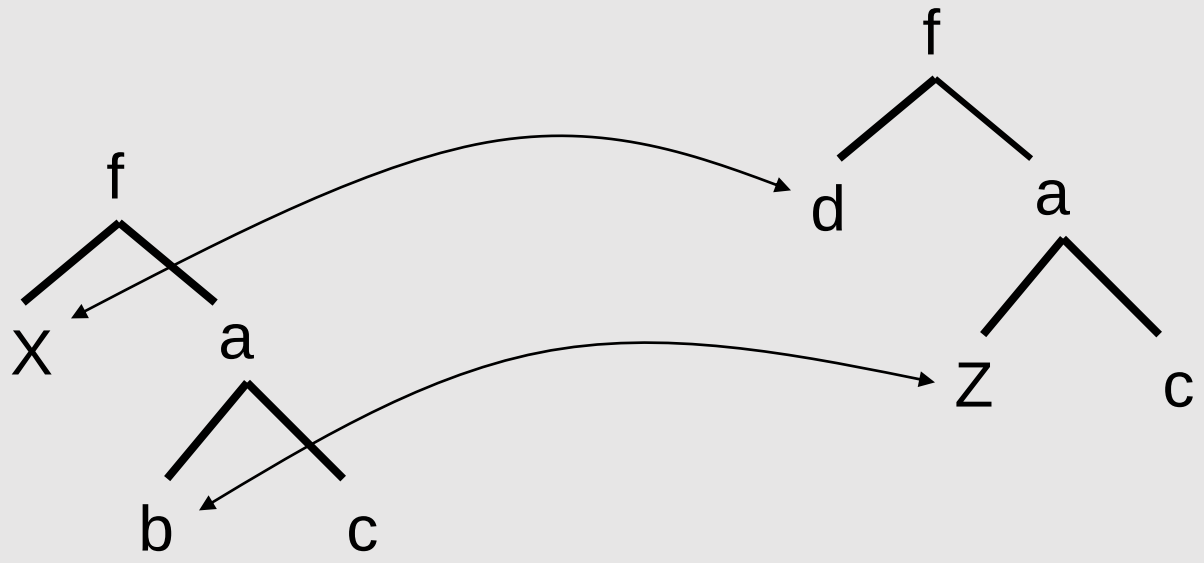
Declarative reading: "That H is provable follows from goals G_1, G_2, …, G_n being provable."

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

Unification

Examples

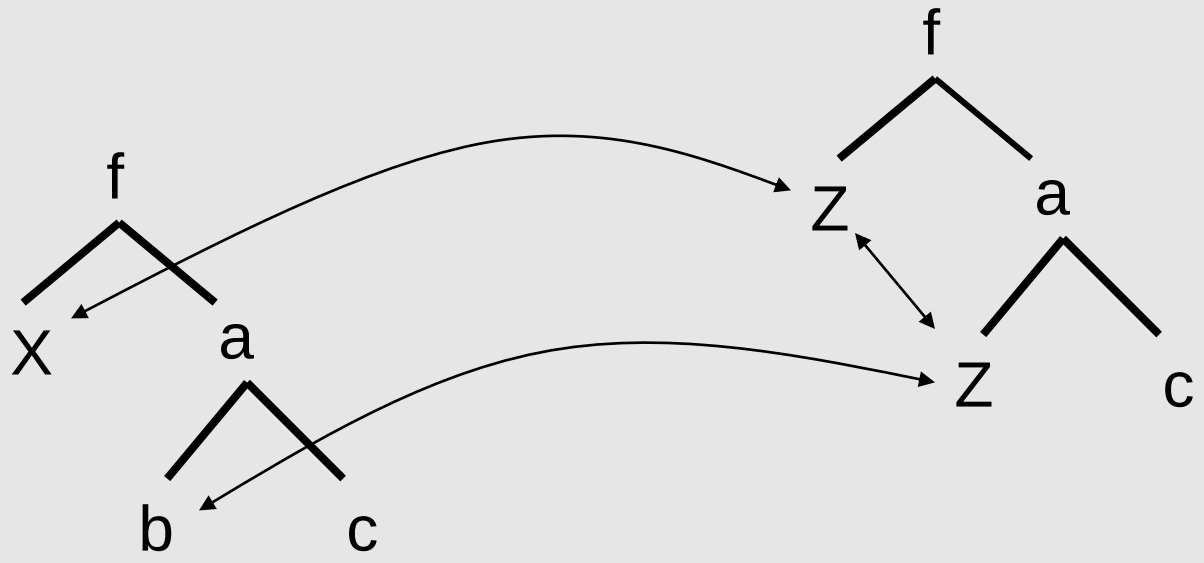
The terms f(X, a(b,c)) and f(d, a(Z, c)) unify.



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.

Examples

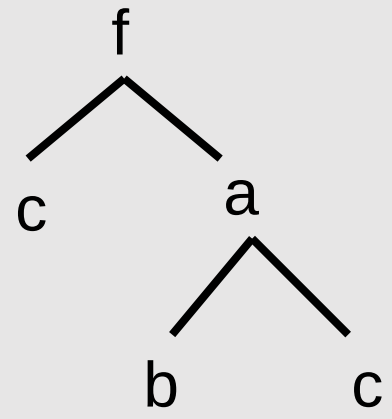
The terms f(X, a(b,c)) and f(Z, a(Z, c)) unify.

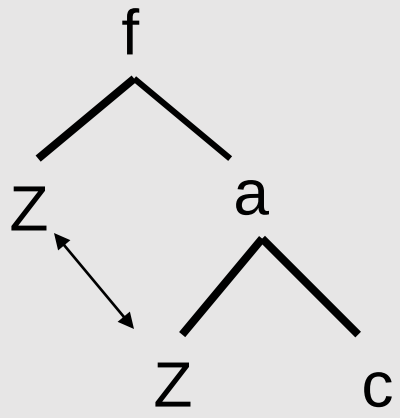


Note that Z co-refers within the term.
Here, X/b, Z/b.

Examples

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



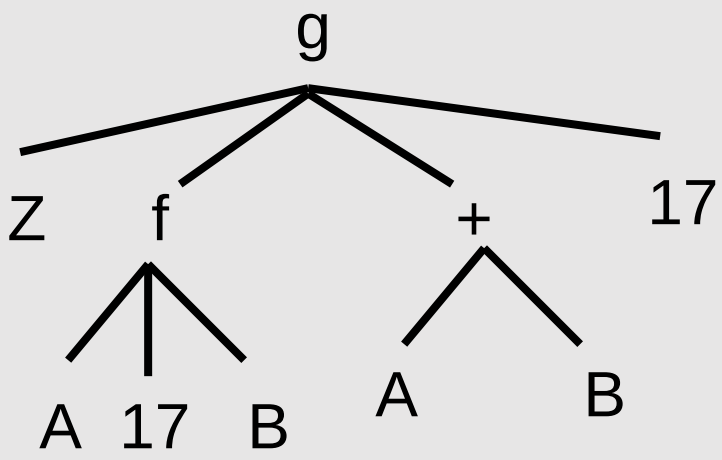


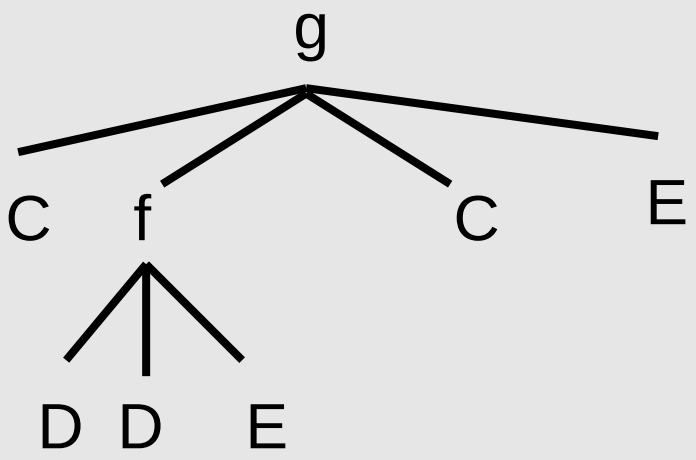
No matter how hard you try, these two terms cannot be made identical by substituting terms for variables.

Example

Do terms g(Z, f(A, 17, B), A+B, 17) and

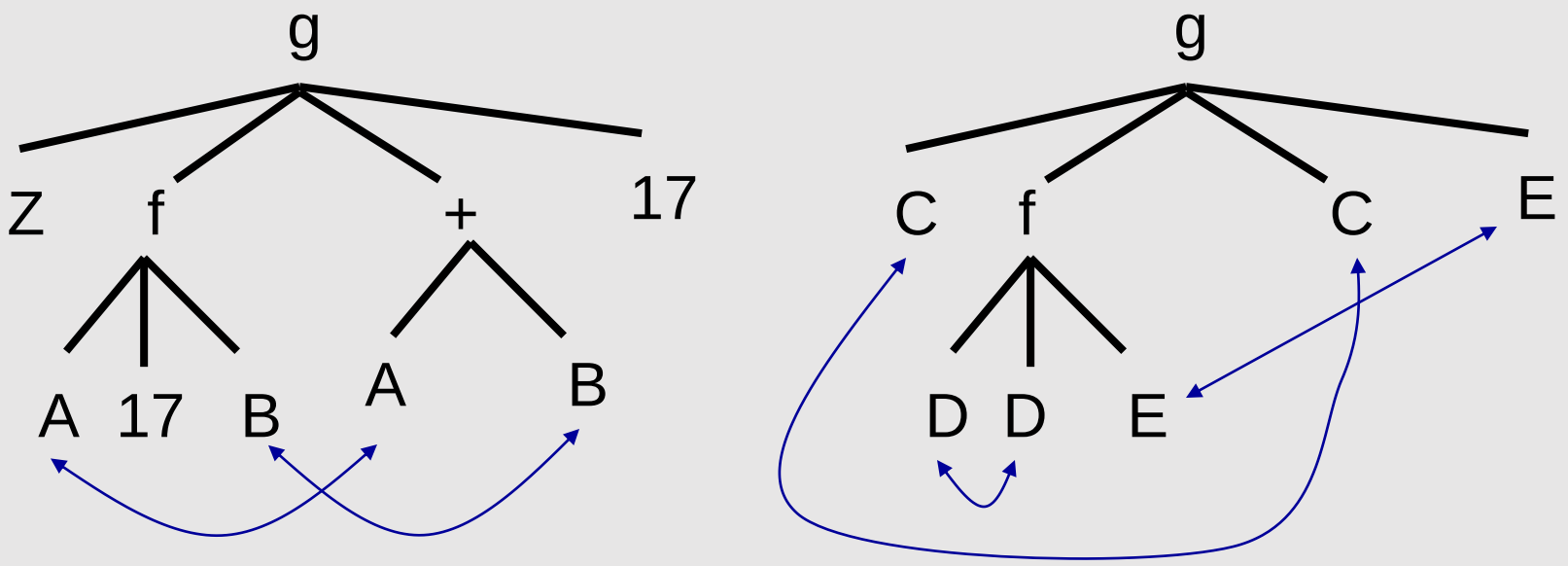
g(C, f(D, D, E), C, E) unify?





Example

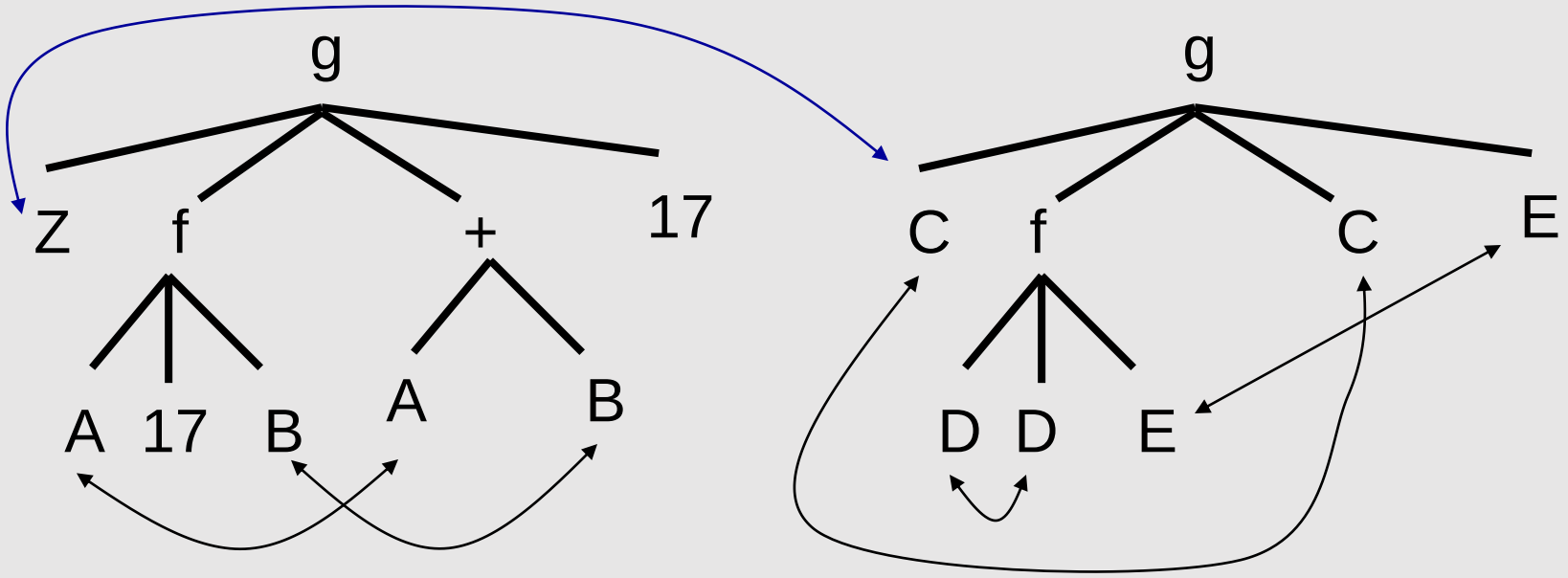
First write in the co-referring variables.



Example

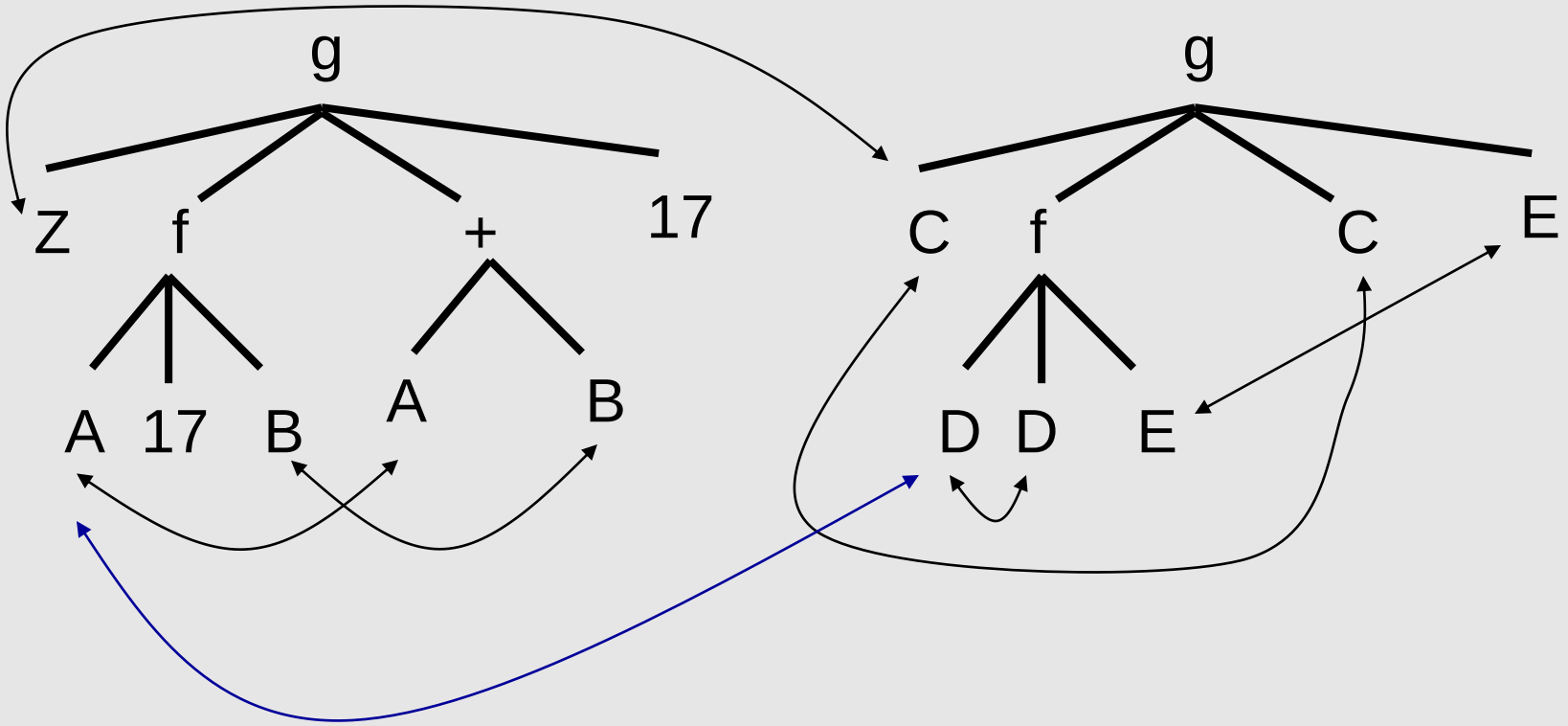
Z/C, C/Z

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



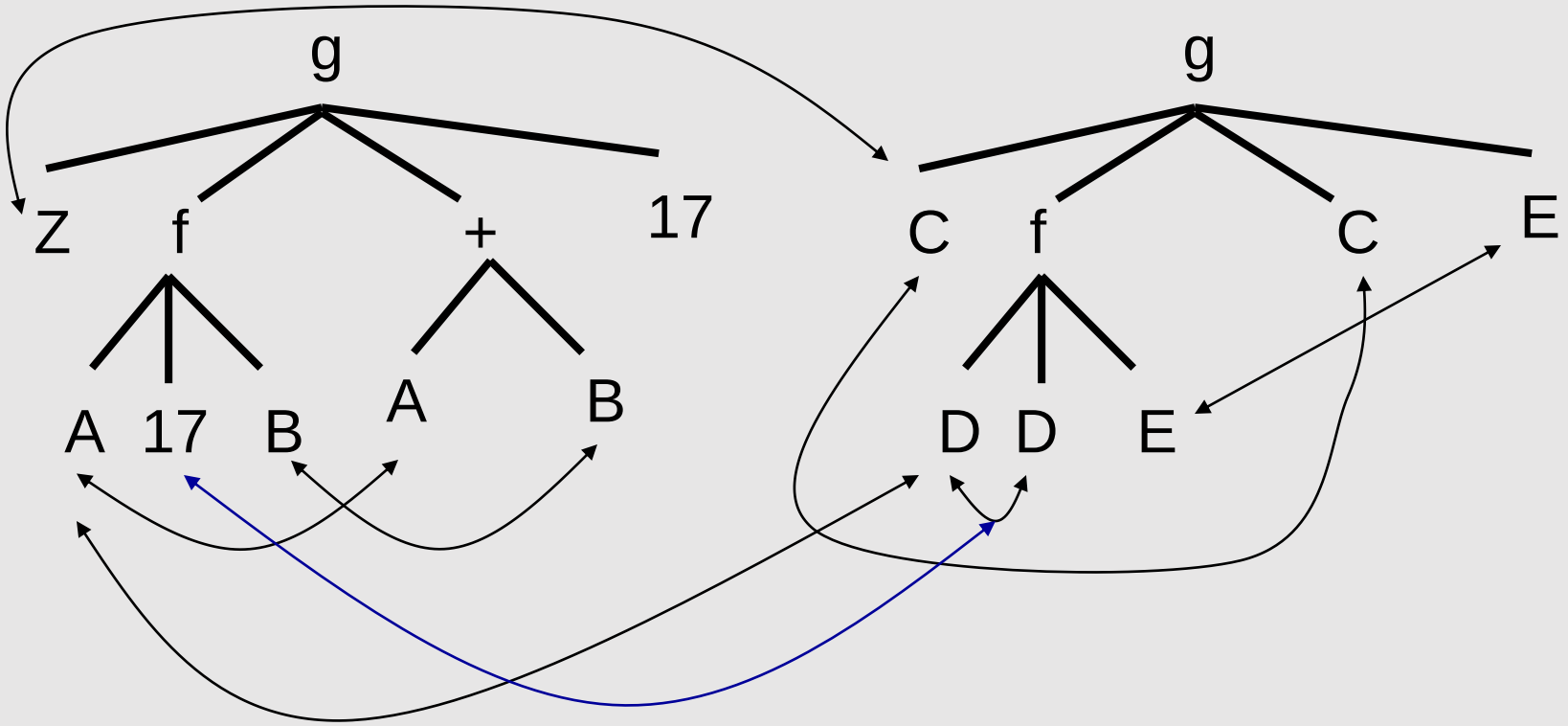
Example

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



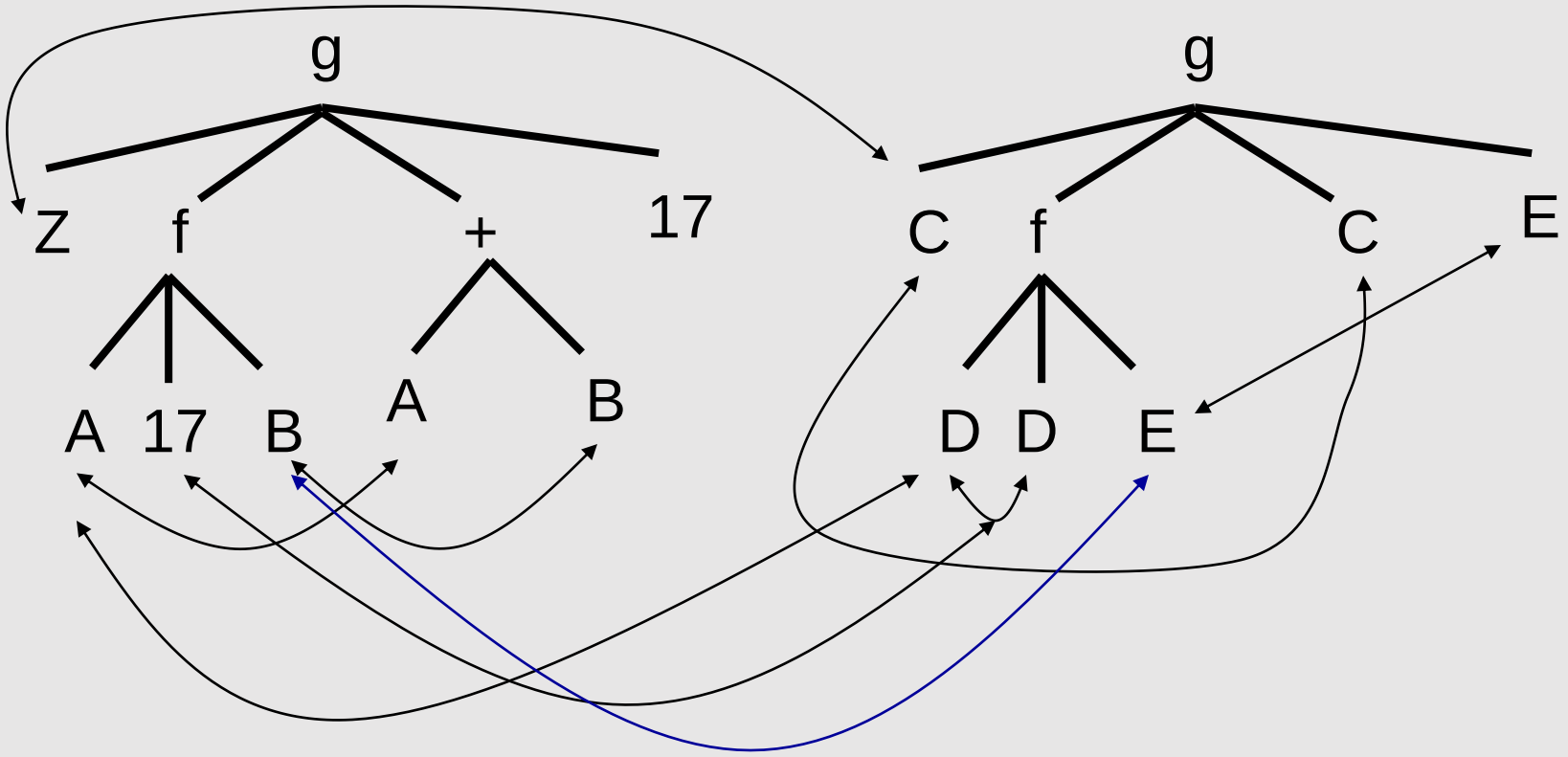
Example

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



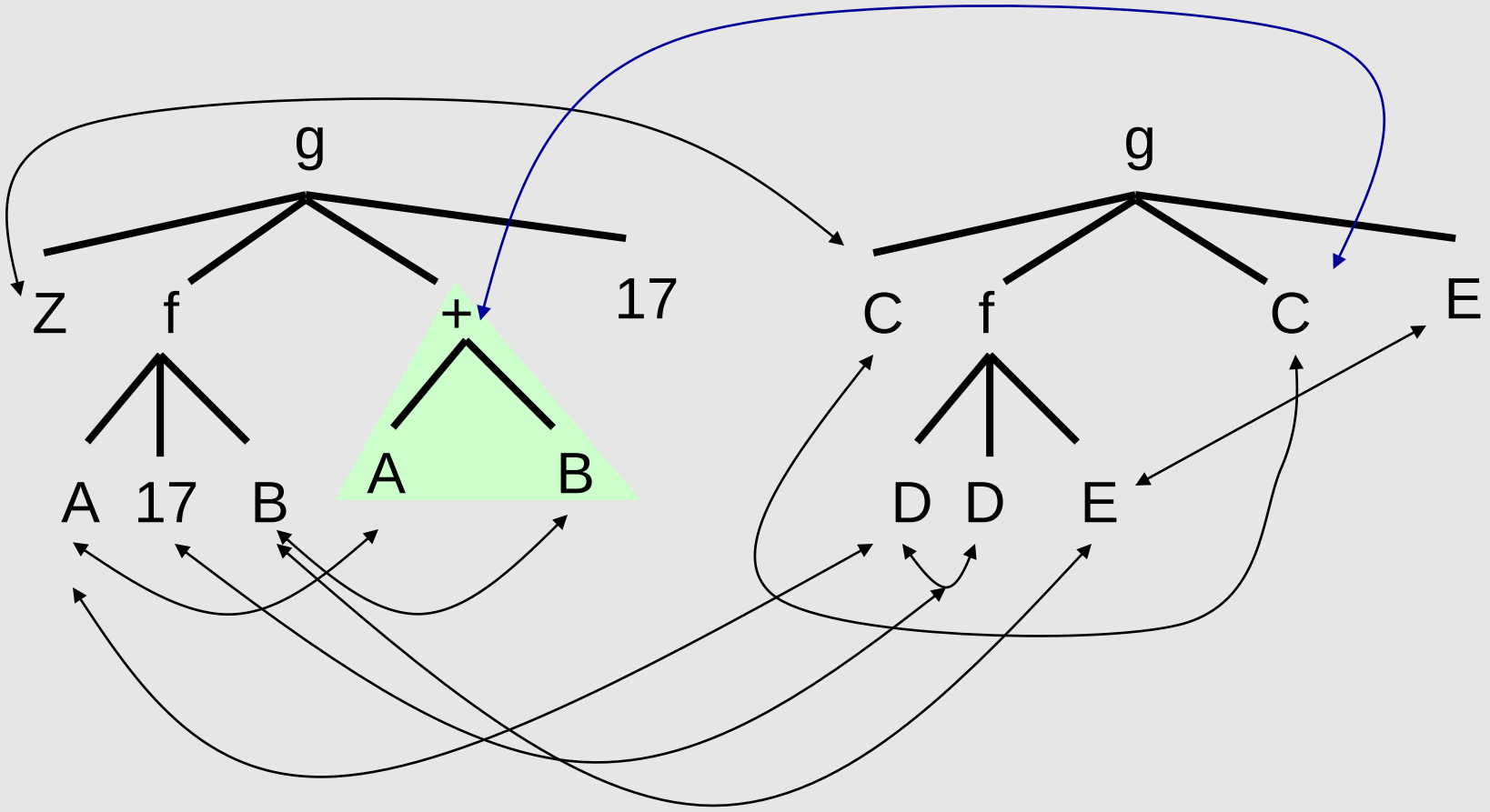
Example

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



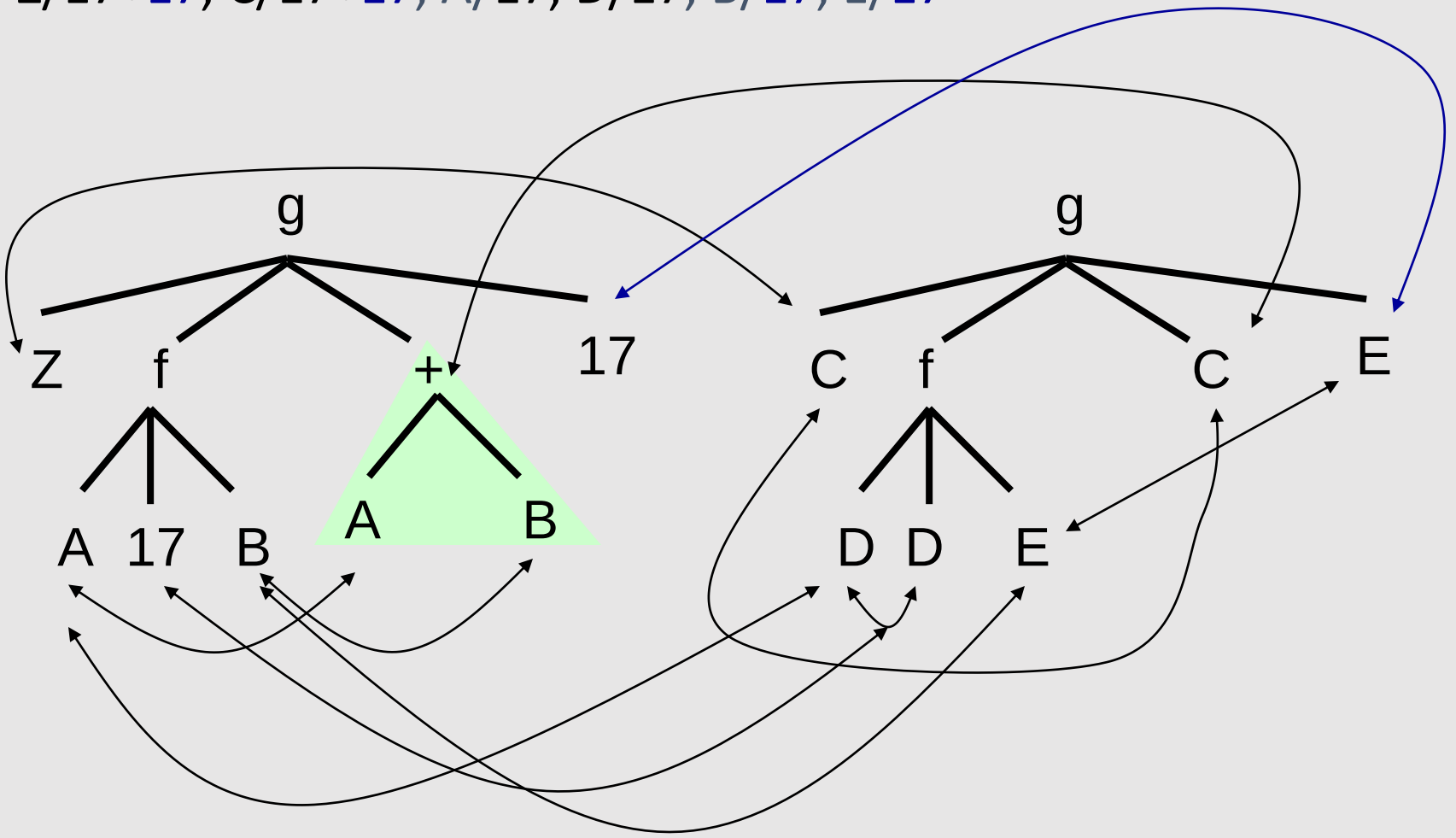
Example

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



Example

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



A proper example

See prolog files

In-class exercise

Here's what we're trying to capture:

- Some people know each other, some don't
- People like things and/or people
- People like themselves
- If two people know each other and like the same person/thing, they like each other

There are two files on Laulima. The first (LikeDemo.pl) is a naïve approach, treating Prolog as if it really were fully declarative. The second (LikeDemo2.pl) is better, but still fails on queries like "likes(sassy, jock)." Why? (use "trace." to step through what Prolog is doing). Can you fix it? Post your improved version on Laulima.

Equality in Prolog

See http://www.swi-prolog.org/pldoc/man?section=arith for exact definitions, but roughly:

- A = B. means: Can they be unified, symbolically?
- A == B. means: Are they exactly the same, before evaluation?
- A =:= B. means: Do they evaluate to the same thing? (i.e. arithmetic equal)
- A is B. means: When B is evaluated, can it unify with A?

Factorial

How would you write a factorial in Prolog? See my files on Laulima 2 versions.

Note: Semicolons are legal but will make you sad.