



**PSC 2023/24** (375AA, 9CFU)

Principles for Software Composition

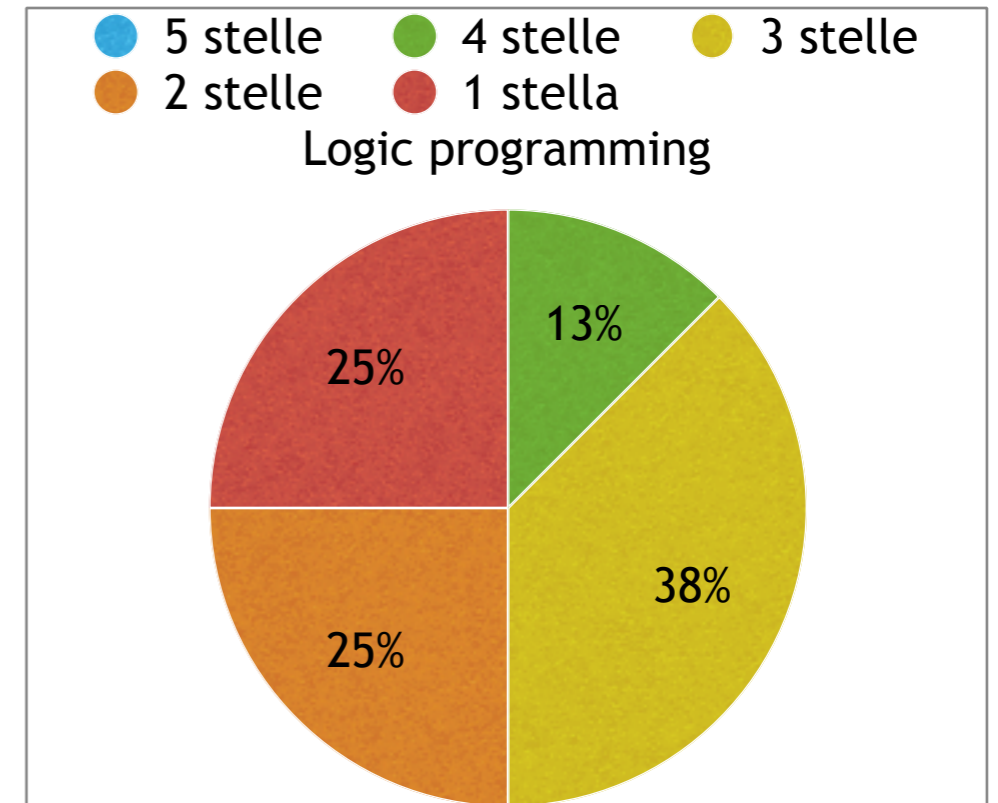
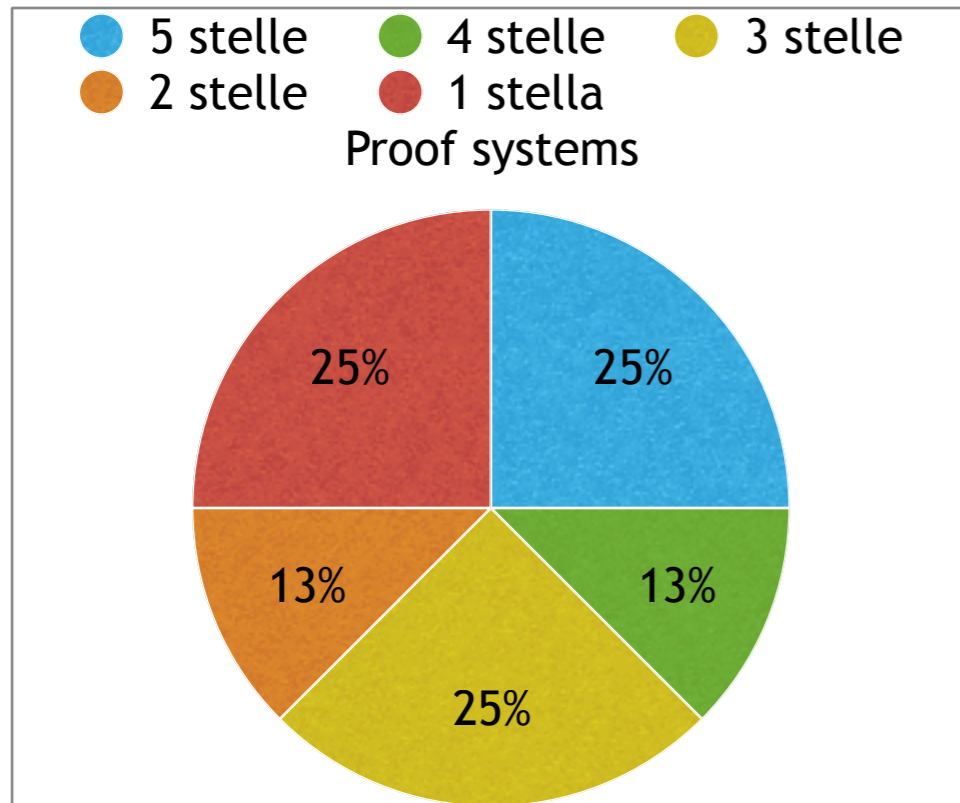
Roberto Bruni

<http://www.di.unipi.it/~bruni/>

[http://didawiki.di.unipi.it/doku.php/  
magistraleinformatica/psc/start](http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/start)

04 - Logical Systems

# From your forms

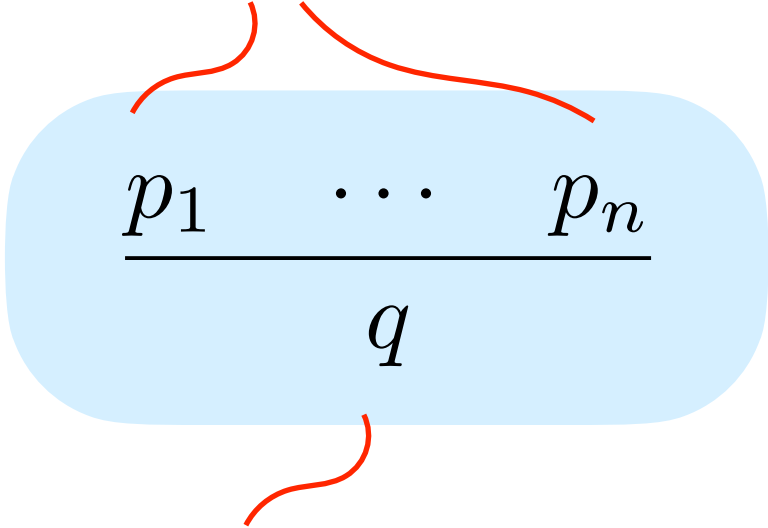


(over 8 answers)

# Inference rules

# Inference rules

premises (one, none, many)


$$\frac{p_1 \quad \cdots \quad p_n}{q}$$

if the premises are valid,  
then the conclusion is also valid

conclusion (one)

$p_1, \dots, p_n, q$  are formulas

any variable they contain is universally quantified (implicitly)

a rule instance is obtained by applying some  $\rho$  to  $p_1, \dots, p_n, q$



# Axioms

no premises

$$\frac{}{q}$$

the conclusion is valid

conclusion (always valid, it is a fact)

# Rule instances

$$(prod) \frac{E_0 \longrightarrow n_0 \quad E_1 \longrightarrow n_1}{E_0 \otimes E_1 \longrightarrow n} \quad n = n_0 \cdot n_1$$

$$\rho \triangleq [E_0 = 1, E_1 = 1 \oplus 2, n_0 = 1, n_1 = 3, n = 3]$$

an instance  
of *(prod)*

$$(prod) \frac{1 \longrightarrow 1 \quad 1 \oplus 2 \longrightarrow 3}{1 \otimes (1 \oplus 2) \longrightarrow 3} \quad 3 = 1 \cdot 3$$

this is also  
an instance  
of *(prod)*!

$$\rho \triangleq [E_0 = 1, E_1 = 1 \oplus 2, n_0 = 3, n_1 = 5, n = 15]$$

$$(prod) \frac{1 \longrightarrow 3 \quad 1 \oplus 2 \longrightarrow 5}{1 \otimes (1 \oplus 2) \longrightarrow 15} \quad 15 = 3 \cdot 5$$

but it is unlikely that such premises will be valid

# More instances

$$(prod) \frac{E_0 \longrightarrow n_0 \quad E_1 \longrightarrow n_1}{E_0 \otimes E_1 \longrightarrow n} \quad n = n_0 \cdot n_1$$

an instance  
of *(prod)*

$$(prod) \frac{E \otimes 2 \longrightarrow k \quad E \oplus 1 \longrightarrow 3}{(E \otimes 2) \otimes (E \oplus 1) \longrightarrow 3k}$$

variables can be shared

# Logical System

$$R = \left\{ \frac{}{r}, \frac{p_1 \cdots p_n}{q}, \dots \right\}$$

A **logical system** is just a set of axioms and inference rules

if an inference rule contains some variables,  
we assume all its instances are in  $R$

# Derivation

Given a logical system  $R$ , a **derivation in  $R$** , is written

$$d \Vdash_R q$$

where

- either  $d = \left( \frac{}{q} \right) \in R$  is an axiom of  $R$ ;
- or  $d = \left( \frac{d_1, \dots, d_n}{q} \right)$  for some derivations  $d_1 \Vdash_R p_1, \dots, d_n \Vdash_R p_n$  such that  $\left( \frac{p_1, \dots, p_n}{q} \right) \in R$  is an inference rule of  $R$ .

a derivation is a proof tree (whose leaves are axioms)

# Example

$$S ::= \epsilon \mid ( S ) \mid S S$$

$$R = \left\{ \frac{}{\epsilon \in \mathcal{L}}, \frac{S \in \mathcal{L}}{(S) \in \mathcal{L}}, \frac{S_0 \in \mathcal{L} \quad S_1 \in \mathcal{L}}{S_0 S_1 \in \mathcal{L}} \right\}$$

$$d = \frac{\frac{\frac{}{\epsilon \in \mathcal{L}}}{() \in \mathcal{L}} \quad \frac{\frac{\frac{}{\epsilon \in \mathcal{L}}}{() \in \mathcal{L}}}{(( )) \in \mathcal{L}}}{()(( )) \in \mathcal{L}}}{(( ))(( )) \in \mathcal{L}}$$

$$d \Vdash_R (( ))(( )) \in \mathcal{L}$$

# Example

$$R = \left\{ \frac{}{N \longrightarrow n}, \frac{E_0 \longrightarrow n_0 \quad E_1 \longrightarrow n_1}{E_0 \oplus E_1 \longrightarrow n_0 + n_1}, \frac{E_0 \longrightarrow n_0 \quad E_1 \longrightarrow n_1}{E_0 \otimes E_1 \longrightarrow n_0 \cdot n_1} \right\}$$

$$d = \frac{\frac{1 \longrightarrow 1 \quad 2 \longrightarrow 2}{(1 \oplus 2) \longrightarrow 3} \quad \frac{3 \longrightarrow 3 \quad 4 \longrightarrow 4}{(3 \oplus 4) \longrightarrow 7}}{(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21}$$

$$d \Vdash_R (1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21$$

# Theorems

Given a logical system  $R$ , a **theorem of  $R$**  is written

$$\Vdash_R q$$

$$\exists d. d \Vdash_R q$$

where  $q$  is a formula such that we can find a derivation for  $q$  in  $R$

The set of all theorems of  $R$  is denoted by  $I_R$

$$I_R \triangleq \{ q \mid \Vdash_R q \}$$



# Inline notation

$$d = \frac{\frac{\overline{1 \longrightarrow 1} \quad \overline{2 \longrightarrow 2}}{(1 \oplus 2) \longrightarrow 3} \quad \frac{\overline{3 \longrightarrow 3} \quad \overline{4 \longrightarrow 4}}{(3 \oplus 4) \longrightarrow 7}}{(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21}$$

$$(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21$$

$$\swarrow (1 \oplus 2) \longrightarrow 3, (3 \oplus 4) \longrightarrow 7$$

$$\swarrow 1 \longrightarrow 1, 2 \longrightarrow 2, (3 \oplus 4) \longrightarrow 7$$

$$\swarrow 2 \longrightarrow 2, (3 \oplus 4) \longrightarrow 7$$

$$\swarrow (3 \oplus 4) \longrightarrow 7$$

$$\swarrow 3 \longrightarrow 3, 4 \longrightarrow 4 \quad \swarrow 4 \longrightarrow 4 \quad \swarrow \square$$

goal oriented  
derivation

nothing left to prove

# Backtracking

$$(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21$$

$$\nearrow (1 \oplus 2) \longrightarrow 7, (3 \oplus 4) \longrightarrow 3$$

$$\nearrow 1 \longrightarrow 1, 2 \longrightarrow 6, (3 \oplus 4) \longrightarrow 3$$

$$\nearrow 2 \longrightarrow 6, (3 \oplus 4) \longrightarrow 3$$

fail! need to backtrack to the last choice and retry

$$\nearrow 1 \longrightarrow 2, 2 \longrightarrow 5, (3 \oplus 4) \longrightarrow 3$$

fail! need to backtrack to the last choice and retry

....

alternatively, all possibilities can be explored in parallel  
(breadth-first vs depth-first)

goal oriented  
derivation  
(depth-first)

# Logic programming

# PROLOG

Prolog is a simple, yet powerful declarative programming language, based on first-order predicate logic

PROgrammation en LOGique

[’70] (Univ. Marseilles) A. Colmerauer, P. Roussel, R. Kowalski  
(aimed at processing natural (French) language)

```
Every psychiatrist is a person.  
Every person he analyzes is sick.  
Jacques is a psychiatrist in Marseille.  
Is Jacques a person?  
Where is Jacques?  
Is Jacques sick?
```

```
Yes. In Marseille.  
I don't know.
```

```
TOUT PSYCHIATRE EST UNE PERSONNE.  
CHAQUE PERSONNE QU'IL ANALYSE, EST MALADE.  
JACQUES EST UN PSYCHIATRE A *MARSEILLE.  
EST-CE QUE *JACQUES EST UNE PERSONNE?  
OU EST *JACQUES?  
EST-CE QUE *JACQUES EST MALADE?  
OUI. A MARSEILLE. JE NE SAIS PAS.
```

# Algorithm

algorithm = logic + control

what  
(problem description)

how  
(steps to reach a solution)

Horn clauses

resolution

PROLOG  
database

PROLOG  
interpreter

# Formulas

$X = \{x, y, \dots\}$  a set of variables

$\Sigma = \{\Sigma_n\}_n$  a signature of function symbols  $c, f, g, \dots$

$\Pi = \{\Pi_n\}_n$  a signature of predicate symbols  $p, q, \dots$

atomic formula

$$a = p(t_1, \dots, t_n) \quad \begin{array}{l} p \in \Pi_n \\ t_1, \dots, t_n \in T_{\Sigma, X} \end{array}$$

formula

$a_1, \dots, a_n$  a possibly empty conjunction of atomic formulas

# Example

$X = \{S, S_0, S_1, \dots\}$  a set of variables

$\Sigma_0 = \{\epsilon, (, )\}$  a set of constants

$\Sigma_2 = \{- \_ -\}$  a binary (infix) operator

$\Pi_1 = \{- \in \mathcal{L}\}$  a unary predicate symbol

atomic formula

$S) \in \mathcal{L}$

formula

$S) \in \mathcal{L}, SS)) \in \mathcal{L}$

# Example

$X = \{N, E, E_0, E_1, \dots, n, n_0, n_1, \dots\}$  a set of variables

$\Sigma_0 = \{0, 1, 2, \dots, 0, 1, 2, \dots\}$  a set of constants

$\Sigma_2 = \{- \oplus -, - \otimes -\}$  a set of binary (infix) operators

$\Pi_2 = \{- \longrightarrow -\}$  a binary (infix) predicate symbol

atomic formula

$$E \oplus 2 \longrightarrow 5$$

formula

$$E \oplus 2 \longrightarrow 5, E \otimes 7 \longrightarrow n$$



# Logic programs

Horn clause

an atomic formula  
(the HEAD)  $h :- r$  a formula  
(the BODY)

$a :- a_1, \dots, a_n$  analogous to  $\frac{a_1 \cdots a_n}{a}$

a set (or list) of Horn clauses

logic program  $L$

$$L = \left\{ \begin{array}{c} \dots \\ h :- r. \\ \dots \end{array} \right\}$$

# Applications

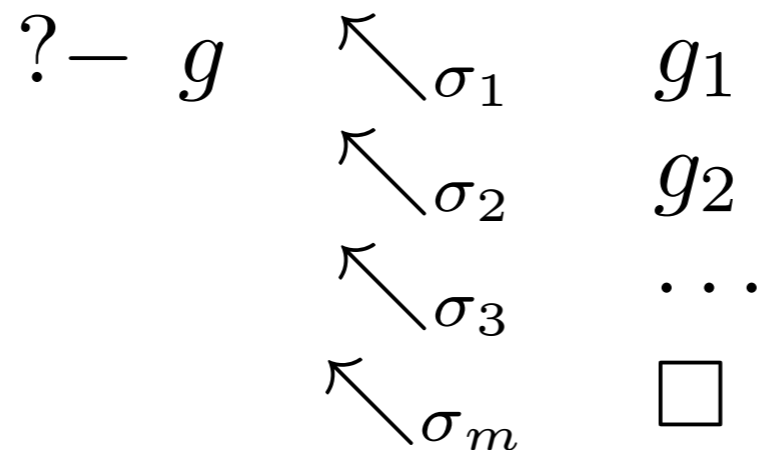
a logic program serves to answer the following question:  
given a formula  $g$  that we want to prove,  
what are the valid instances of  $g$ ?

$$? - (1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow n, \quad n \oplus E \longrightarrow 26$$

# SLD resolution

**Idea:** iteratively reduce the initial goal  $g$  by applying one of the Horn clauses in  $L$  to one of the atomic formulas in the goal;

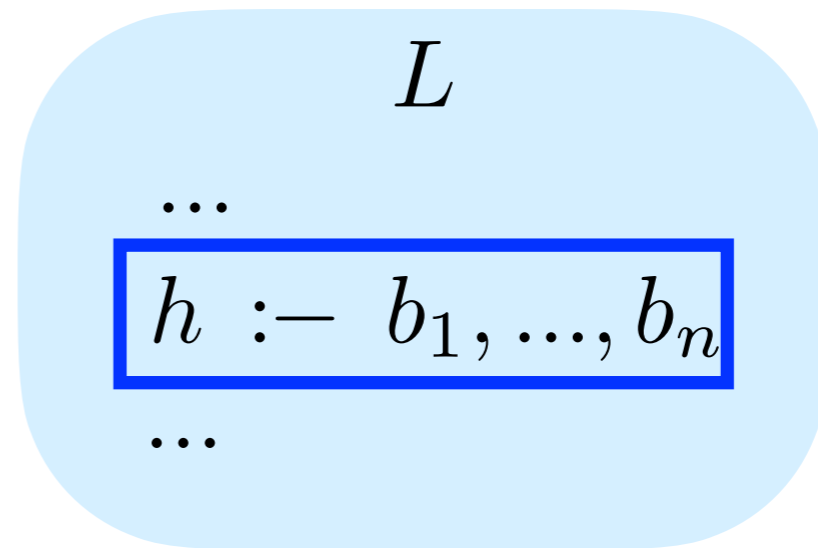
each application computes a most general unifier (mgu), replaces the selected formula with the body of the selected clause and applies the mgu to the new goal



then  $g\sigma_1\sigma_2\dots\sigma_m$  is a theorem

# SLD resolution

?-  $a_1, \dots, a_i, \dots, a_k$



repeat the following until no goal is left:

1. select a clause of the goal  $a_i$  (e.g., the first);
2. select a Horn clause  $h :- b_1, \dots, b_n$  in  $L$  whose head unifies with  $a_i$ ;
3. let  $\sigma$  be a most general unifier ( $a_i\sigma = h\sigma$ );
4. replace  $a_i$  with  $b_1, \dots, b_n$ ;
5. apply the computed substitution  $\sigma$  to the goal  $(a_1, \dots, b_1, \dots, b_n, \dots, a_k)\sigma$

# Pay attention

atomic goals can share variables: the substitution must be applied to all of them to propagate the information



$(a_1, \dots, b_1, \dots, b_n, \dots, a_k)\sigma$



$a_1, \dots, (b_1, \dots, b_n)\sigma, \dots, a_k$

# Pay attention

the same clause can be reused many times:  
each time its variables must be renamed (before unification)  
with *fresh* identifiers to avoid clashes

repeat the following until no goal is left:

1. select a clause of the goal  $a_i$  (e.g., the first);
2. select a Horn clause  $h :- b_1, \dots, b_n$  in  $L$ ;
3. let  $\rho : X \rightarrow X$  rename the variables in  $vars(h :- b_1, \dots, b_n)$  to fresh ones;
4.  $(h :- b_1, \dots, b_n)\rho$  is called a *variant* of the original clause;
5. let  $\sigma$  be a most general unifier ( $a_i\sigma = (h\rho)\sigma$ );
6. replace  $a_i$  with  $(b_1, \dots, b_n)\rho$ ;
7. apply the computed substitution  $\sigma$  to the goal  $(a_1, \dots, (b_1, \dots, b_n)\rho, \dots, a_k)\sigma$

# Pay attention

in the computed substitutions, only the variables that appears in the goal are of some interest to us

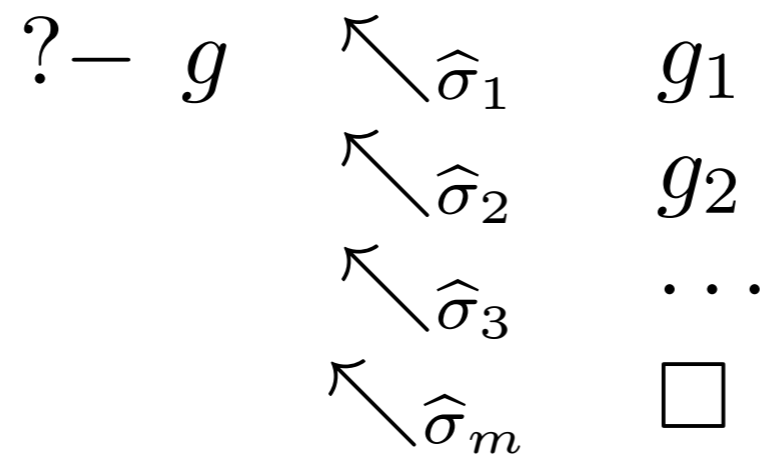
$$\begin{array}{l} \sigma : X \rightarrow T_{\Sigma, X} \\ Y \subseteq X \end{array} \quad \sigma|_Y(x) \triangleq \begin{cases} \sigma(x) & \text{if } x \in Y \\ x & \text{otherwise} \end{cases}$$

we only record this partial information

$$a_1, \dots, a_i, \dots, a_k \quad \hat{\sigma} \quad (a_1, \dots, (b_1, \dots, b_n)\rho, \dots, a_k)\sigma$$

$$\hat{\sigma} \triangleq \sigma|_{vars(a_1, \dots, a_k)}$$

# Computed answer substitution



$$\sigma = \hat{\sigma}_1 \cdot \hat{\sigma}_2 \cdot \hat{\sigma}_3 \cdots \hat{\sigma}_m$$

is called cas (computed answer substitution)

where

$$t(\sigma_1 \cdot \sigma_2) \stackrel{\Delta}{=} t\sigma_1\sigma_2 = \sigma_2(\sigma_1(t)) = (\sigma_2 \circ \sigma_1)(t)$$



# Example

$\Sigma_0 = \{0, \dots\}$        $\Pi_3 = \{\text{sum}, \dots\}$

$\Sigma_1 = \{s, \dots\}$

sum as a predicate       $\text{sum}(x, y, z)$       means       $x + y = z$

(a fact)

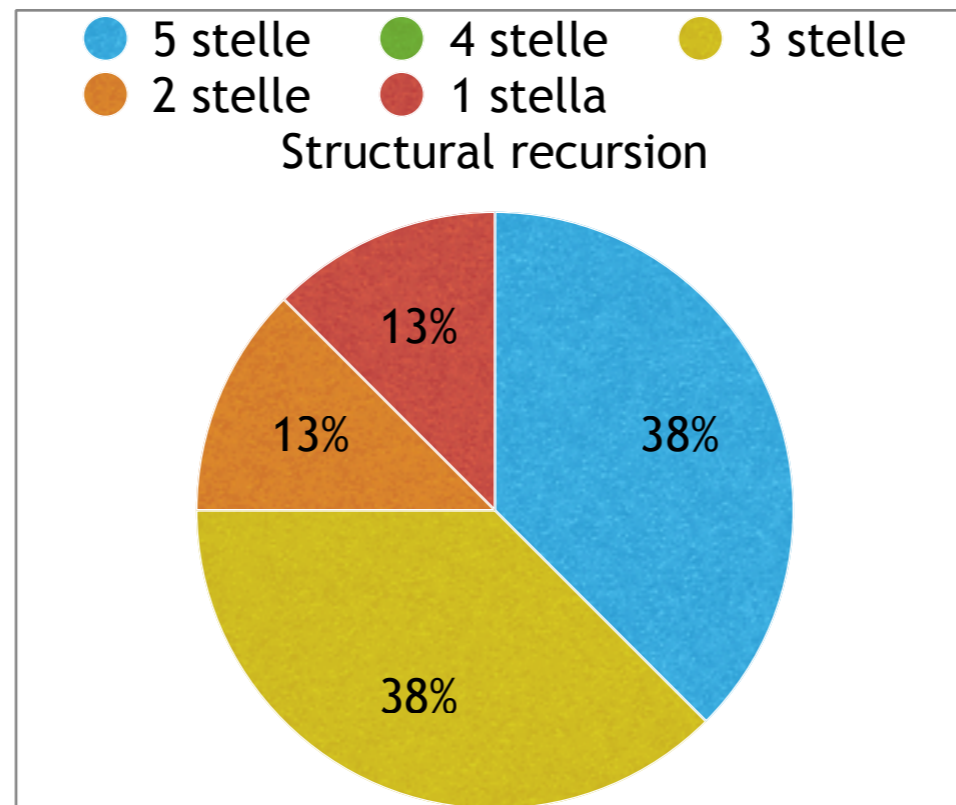
$L$

$\text{sum}(0, y, y).$   
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z).$

in PROLOG  
each statement  
ends with dot

# Structural recursion

the previous definition is an example of structural recursion



(over 8 answers)

# 2+2=?

our goal  $? - \text{sum}(s(s(0)), s(s(0)), n)$ .

$L$

$\text{sum}(0, y, y)$ .  
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z)$ .

$\{\text{sum}(s(s(0)), s(s(0)), n) \stackrel{?}{=} \text{sum}(0, y', y')\}$  **fails**

$\{\text{sum}(s(s(0)), s(s(0)), n) \stackrel{?}{=} \text{sum}(s(x_1), y_1, s(z_1))\}$  **succeeds**

$$\sigma_1 = [x_1 = s(0), y_1 = s(s(0)), n = s(z_1)]$$

$$\hat{\sigma}_1 = [n = s(z_1)]$$

$$\hat{\sigma}_1 = [n = s(z_1)]$$

$$\text{sum}(s(s(0)), s(s(0)), n) \xleftarrow{\hat{\sigma}_1} (\text{sum}(x_1, y_1, z_1))\sigma_1 = \text{sum}(s(0), s(s(0)), z_1)$$

# 1+2=?

our new goal

$$\text{sum}(s(0), s(s(0)), z_1).$$

$L$

$\text{sum}(0, y, y).$   
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z).$

$$\{\text{sum}(s(0), s(s(0)), z_1) \stackrel{?}{=} \text{sum}(0, y', y')\} \quad \text{fails}$$

$$\{\text{sum}(s(0), s(s(0)), z_1) \stackrel{?}{=} \text{sum}(s(x_2), y_2, s(z_2))\} \quad \text{succeeds}$$

$$\sigma_2 = [x_2 = 0, y_2 = s(s(0)), z_1 = s(z_2)]$$

$$\hat{\sigma}_2 = [z_1 = s(z_2)]$$

$$\text{sum}(s(s(0)), s(s(0)), n) \begin{array}{l} \nwarrow_{\hat{\sigma}_1} \text{sum}(s(0), s(s(0)), z_1) \\ \nwarrow_{\hat{\sigma}_2} (\text{sum}(x_2, y_2, z_2))\sigma_2 \\ = \text{sum}(0, s(s(0)), z_2) \end{array}$$

$$\hat{\sigma}_1 = [n = s(z_1)]$$

$$\hat{\sigma}_2 = [z_1 = s(z_2)]$$

# 0+2=?

our new goal

$$\text{sum}(0, s(s(0)), z_2).$$

*L*

sum(0, *y*, *y*).  
 sum(s(*x*), *y*, s(*z*)) :- sum(*x*, *y*, *z*).

$$\{\text{sum}(0, s(s(0)), z_2) \stackrel{?}{=} \text{sum}(0, y_3, y_3)\}$$

succeeds

$$\sigma_3 = [y_3 = s(s(0)), z_2 = s(s(0))]$$

$$\hat{\sigma}_3 = [z_2 = s(s(0))]$$

$$\text{sum}(s(s(0)), s(s(0)), n)$$

↖  $\hat{\sigma}_1$

$$\text{sum}(s(0), s(s(0)), z_1)$$

$$\hat{\sigma}_1 = [n = s(z_1)]$$

↖  $\hat{\sigma}_2$

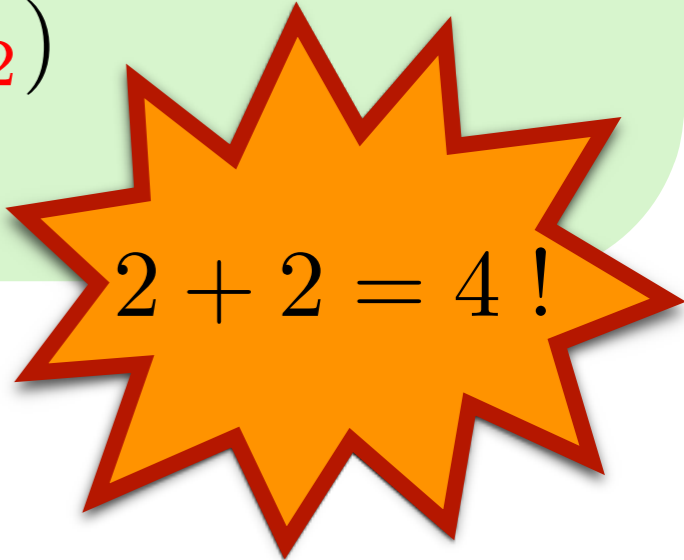
$$\text{sum}(0, s(s(0)), z_2)$$

$$\hat{\sigma}_2 = [z_1 = s(z_2)]$$

↖  $\hat{\sigma}_3$

□

$$\hat{\sigma}_3 = [z_2 = s(s(0))]$$



$$\hat{\sigma}_1 \cdot \hat{\sigma}_2 \cdot \hat{\sigma}_3 = [n = s(s(s(s(0))))]$$

# 1+n=3?

our goal  $? - \text{sum}(s(0), n, s(s(s(0))))$ .

$L$

$\text{sum}(0, y, y)$ .  
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z)$ .

$\{\text{sum}(s(0), n, s(s(s(0)))) \stackrel{?}{=} \text{sum}(0, y', y')\}$  **fails**

$\{\text{sum}(s(0), n, s(s(s(0)))) \stackrel{?}{=} \text{sum}(s(x_1), y_1, s(z_1))\}$  **succeeds**

$$\sigma_1 = [x_1 = 0, y_1 = n, z_1 = s(s(0))]$$

$$\hat{\sigma}_1 = []$$

$$\text{sum}(s(0), n, s(s(s(0)))) \xleftarrow{\hat{\sigma}_1} (\text{sum}(x_1, y_1, z_1))\sigma_1$$

$$= \text{sum}(0, n, s(s(0)))$$

$$\hat{\sigma}_1 = []$$

# 0+n=2?

our new goal

$$\text{sum}(0, n, s(s(0)))$$

*L*

sum(0, *y*, *y*).  
 sum(*s(x)*, *y*, *s(z)*) :- sum(*x*, *y*, *z*).

$$\{\text{sum}(0, n, s(s(0))) \stackrel{?}{=} \text{sum}(0, y_2, y_2)\}$$

succeeds

$$\sigma_2 = [y_2 = s(s(0)), n = s(s(0))]$$

$$\hat{\sigma}_2 = [n = s(s(0))]$$

$$\text{sum}(s(0), n, s(s(s(0)))) \quad \swarrow_{\hat{\sigma}_1} \quad \text{sum}(0, n, s(s(0)))$$

$$\swarrow_{\hat{\sigma}_2} \quad \square$$

$$\hat{\sigma}_1 = []$$

$$\hat{\sigma}_2 = [n = s(s(0))]$$

$$\hat{\sigma}_1 \cdot \hat{\sigma}_2 = [n = s(s(0))]$$

(1 + n = 3) ⇒ n = 2!

# Jumping creatures



Assuming that:

1. All jumping creatures are green
2. All small jumping creatures are Martians
3. All green Martians are intelligent
4. Ngtrks is small and green
5. Pgvdrk is a jumping Martian

Who is intelligent?

```
green(X) :- jumping(X) .
martian(X) :- small(X) , jumping(X) .
intelligent(X) :- green(X) , martian(X) .
small(ngtrks) .
green(ngtrks) .
jumping(pgvdrk) .
martian(pgvdrk) .
```

?— intelligent(*W*).

intelligent(*W*)

↙ green(*W*) , martian(*W*)

↙ jumping(*W*) , martian(*W*)

↙ martian(pgvdrk)

↙ □ [ *W* = pgvdrk ]

martian(ngtrks)

↙ small(ngtrks) , jumping(ngtrks)

↙ jumping(ngtrks)

*fail!*



# Badge exercise



A binary tree  $T$  is either empty (*nil*) or it is composed of a root value and two successors, which are binary trees themselves.

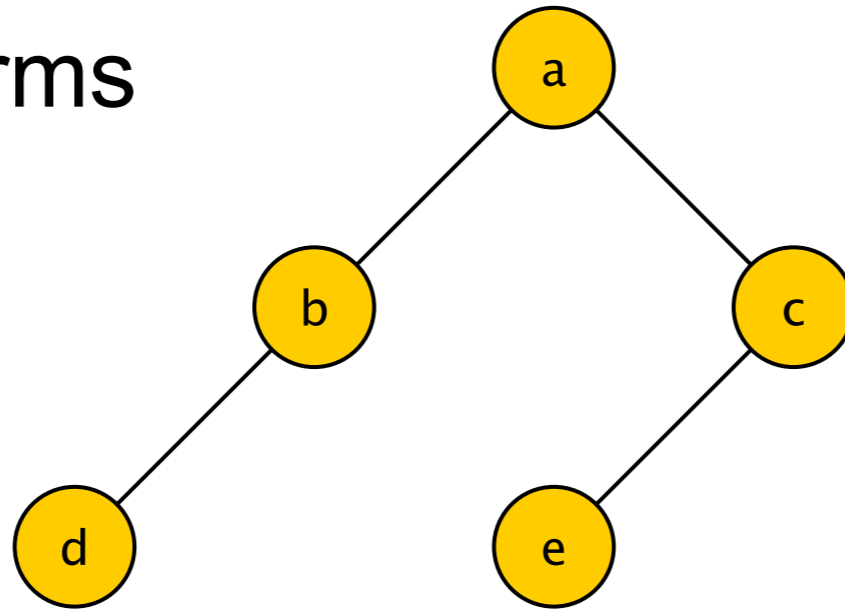
$T$  is symmetric if you can draw a vertical line through the root node and then the right subtree is the mirror image of the left subtree (we are only interested in the structure, values are not relevant).

1. Given the signature below, write the Horn clauses to check whether one tree is the mirror image of another.
2. Then extend the code to check if a tree is symmetric.

$$\Sigma_0 = \{\text{nil}, a, b, \dots\} \quad \Sigma_3 = \{\text{node}\} \quad \Pi_1 = \{\text{sym}\} \quad \Pi_2 = \{\text{mirror}\}$$

# Badge exercise

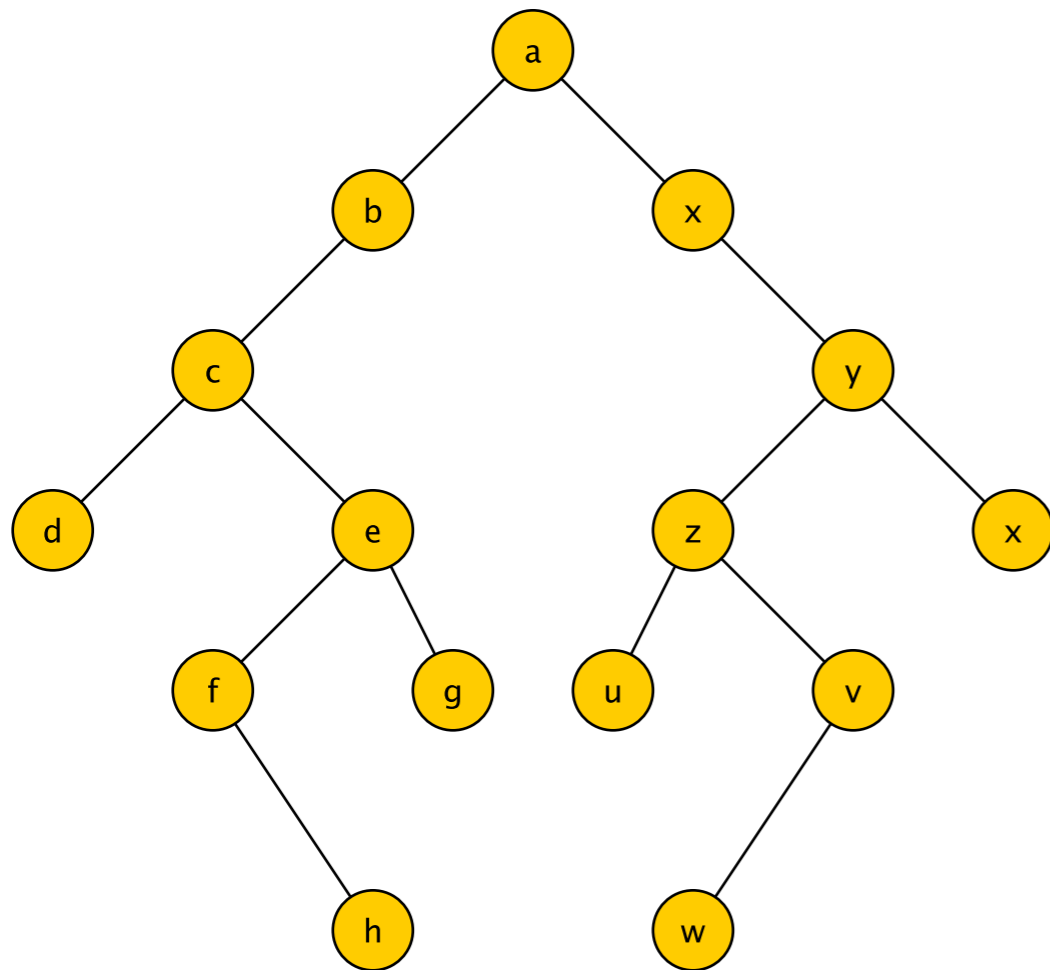
example:  
trees as terms



```
node( a , node( b , node( d , nil , nil ) ,  
nil ) ,  
node( c , node( e , nil , nil ) ,  
nil ) )
```

# Badge exercise

an example of  
symmetric tree



and one that is  
not symmetric

