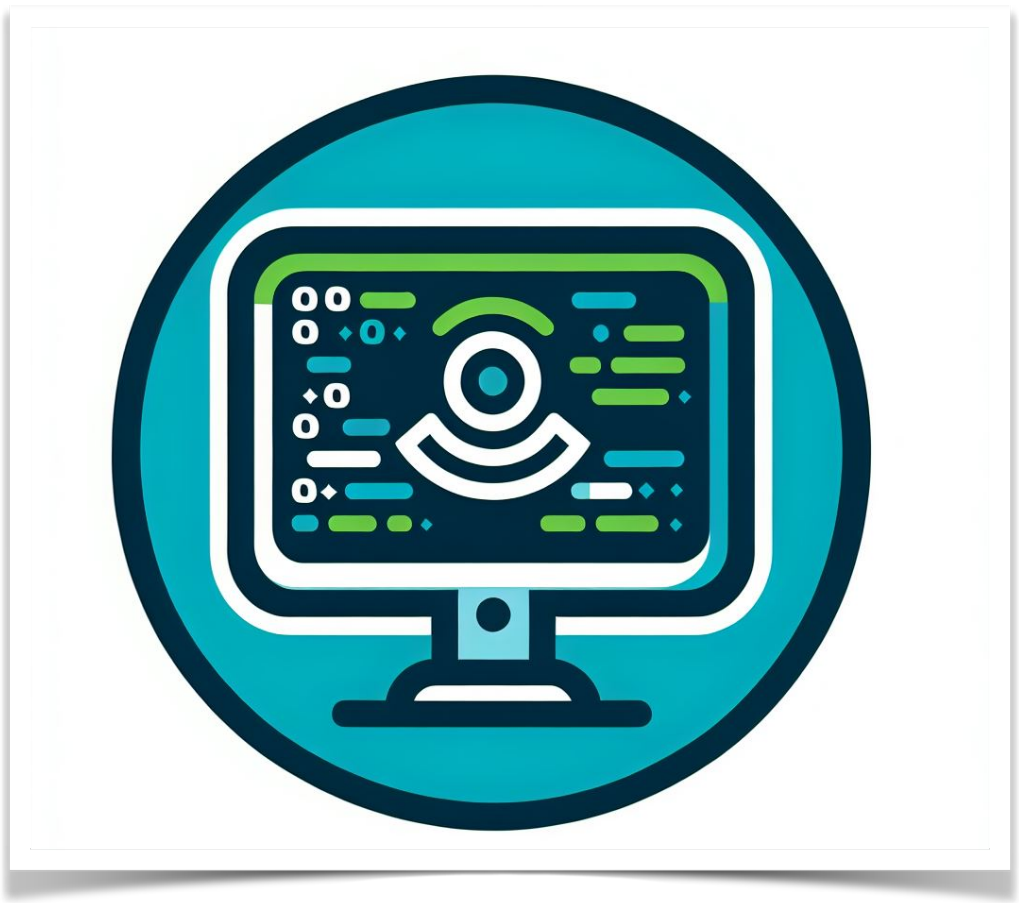


http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/start

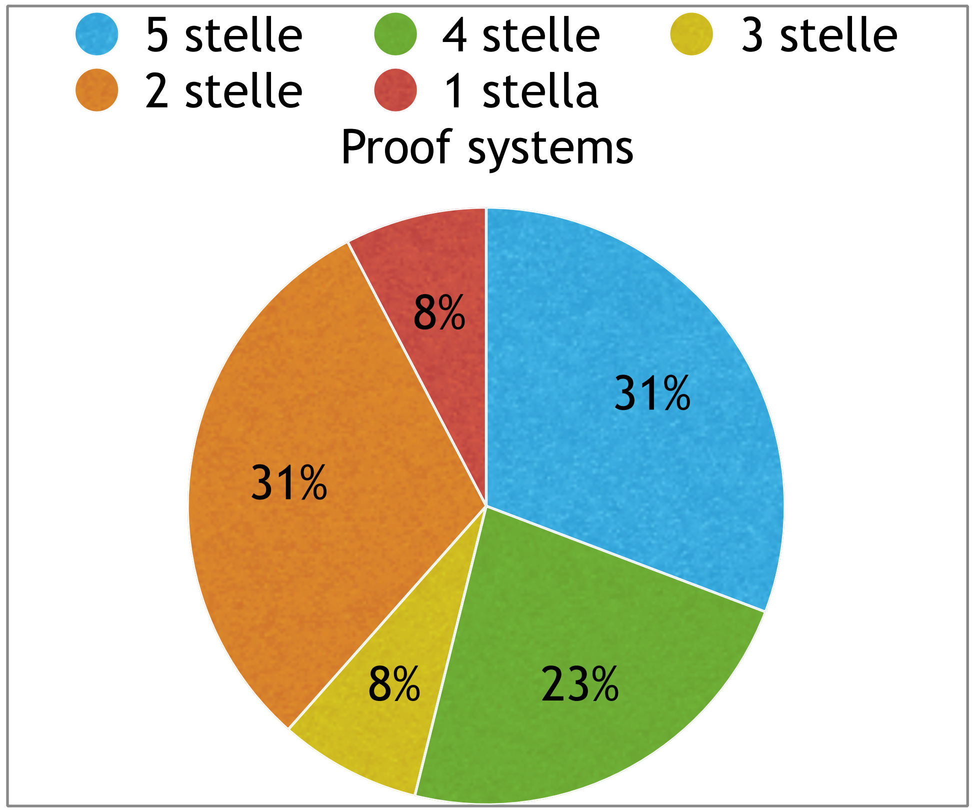
PSC 2024/25 (375AA, 9CFU)

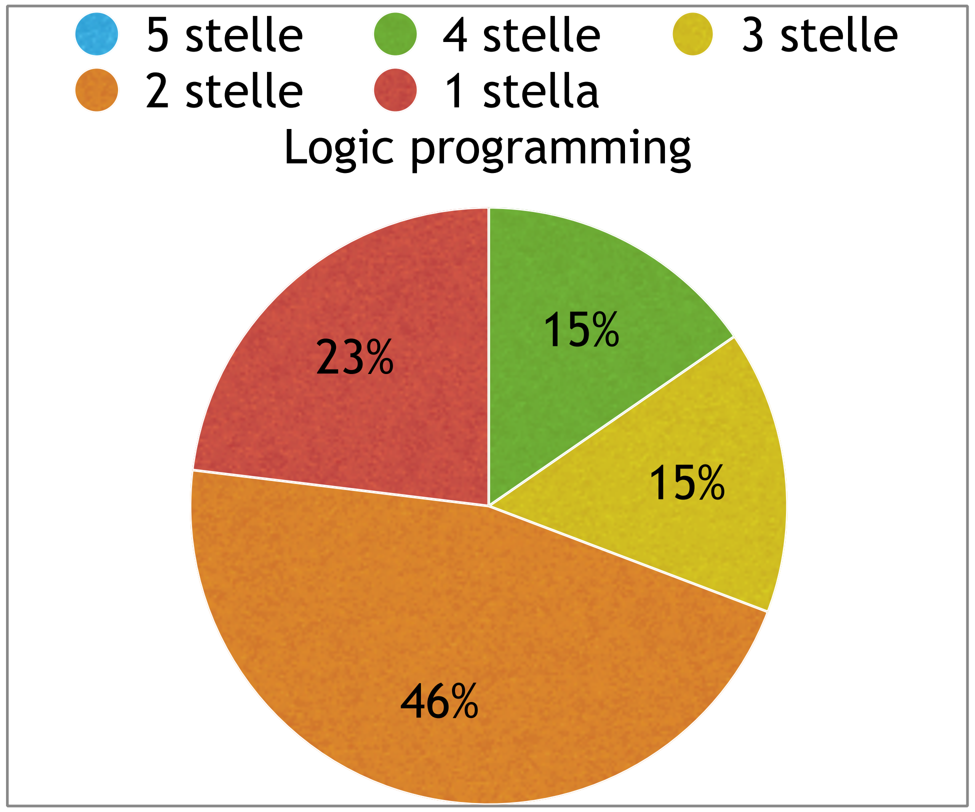
Principles for Software Composition

Roberto Bruni
http://www.di.unipi.it/~bruni/

04 - Logical Systems

From your forms





(over 13 answers)

Inference rules

Inference rules

$$\dfrac{\overbrace{p_1 \qquad \cdots \qquad p_n}^{\text{premises (one, none, many)}}}{\underbrace{q}_{\text{conclusion (one)}}}$$

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

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

any variable they contain is universally quantified (implicitly)

a rule instance is obtained by applying some ρ to $p_1, \ldots, p_n, q$

Axioms

no premises

$$\frac{\quad}{q}$$

the conclusion is valid

conclusion (always valid, it is a fact)

Rule instances

$$(prod)\frac{\mathsf{E}_0 \longrightarrow n_0 \quad \mathsf{E}_1 \longrightarrow n_1}{\mathsf{E}_0 \otimes \mathsf{E}_1 \longrightarrow n} \; n = n_0 \cdot n_1$$

$$\rho \triangleq [\mathsf{E}_0 = 1, \mathsf{E}_1 = 1 \oplus 2, n_0 = 1, n_1 = 3, n = 3]$$

an instance of (*prod*)

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

$$\rho \triangleq [\mathsf{E}_0 = 1, \mathsf{E}_1 = 1 \oplus 2, n_0 = 3, n_1 = 5, n = 15]$$

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

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

but it is unlikely that such premises will be valid

More instances

$$(prod)\,\frac{\mathsf{E}_0 \longrightarrow n_0 \qquad \mathsf{E}_1 \longrightarrow n_1}{\mathsf{E}_0 \otimes \mathsf{E}_1 \longrightarrow n} \quad n = n_0 \cdot n_1$$

an instance of (*prod*)

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

variables can be shared

Logical System

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

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

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

Example

$$\mathsf{S} ::= \epsilon \mid (\ \mathsf{S}\ ) \mid \mathsf{S}\ \mathsf{S}$$

$$R = \left\{ \frac{}{\epsilon \in \mathcal{L}}, \frac{\mathsf{S} \in \mathcal{L}}{(\ \mathsf{S}\ ) \in \mathcal{L}}, \frac{\mathsf{S}_0 \in \mathcal{L} \quad \mathsf{S}_1 \in \mathcal{L}}{\mathsf{S}_0\ \mathsf{S}_1 \in \mathcal{L}} \right\}$$

$$d = \cfrac{\cfrac{\cfrac{}{\epsilon \in \mathcal{L}}}{() \in \mathcal{L}} \qquad \cfrac{\cfrac{\cfrac{}{\epsilon \in \mathcal{L}}}{() \in \mathcal{L}}}{(()) \in \mathcal{L}}}{\cfrac{()(()) \in \mathcal{L}}{(()(())) \in \mathcal{L}}}$$

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

Example

$$R = \left\{ \frac{}{\mathsf{N} \longrightarrow n}, \frac{\mathsf{E}_0 \longrightarrow n_0 \quad \mathsf{E}_1 \longrightarrow n_1}{\mathsf{E}_0 \oplus \mathsf{E}_1 \longrightarrow n_0 + n_1}, \frac{\mathsf{E}_0 \longrightarrow n_0 \quad \mathsf{E}_1 \longrightarrow n_1}{\mathsf{E}_0 \otimes \mathsf{E}_1 \longrightarrow n_0 \cdot n_1} \right\}$$

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

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

Theorems

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

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

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

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

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

goal oriented derivation

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

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

$\nwarrow \quad 2 \longrightarrow 2 \,,\ (3 \oplus 4) \longrightarrow 7$

$\nwarrow \quad (3 \oplus 4) \longrightarrow 7$

nothing left to prove

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

Backtracking

goal oriented derivation (depth-first)

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

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

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

$\nwarrow \quad 2 \longrightarrow 6\,,\ (3 \oplus 4) \longrightarrow 3$

fail! need to backtrack to the last choice and retry

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

fail! need to backtrack to the last choice and retry

...

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

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

$$\boxed{\text{algorithm} = \text{logic} + \text{control}}$$

what (problem description)	how (steps to reach a solution)
Horn clauses	resolution
PROLOG database	PROLOG interpreter

Formulas

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

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

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

atomic formula

$a = p(t_1, ..., t_n)$ $\quad p \in \Pi_n$

$t_1, ..., t_n \in T_{\Sigma,X}$

formula

$a_1, ..., a_n$

a possibly empty conjunction of atomic formulas

Example

$X = \{\mathsf{S}, \mathsf{S}_0, \mathsf{S}_1, ...\}$ 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 $\mathsf{S}) \in \mathcal{L}$

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

Example

$X = \{\mathsf{N}, \mathsf{E}, \mathsf{E}_0, \mathsf{E}_1, \ldots, n, n_0, n_1, \ldots\}$ a set of variables

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

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

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

atomic formula $\mathsf{E} \oplus \mathsf{2} \longrightarrow 5$

formula $\mathsf{E} \oplus \mathsf{2} \longrightarrow 5, \mathsf{E} \otimes \mathsf{7} \longrightarrow n$

Logic programs

Horn clause

an atomic formula (the HEAD) $h \;{:-}\; r$ a formula (the BODY)

$a \;{:-}\; a_1, \ldots, a_n$ analogous to $\dfrac{a_1 \quad \cdots \quad a_n}{a}$

a set (or list) of Horn clauses

logic program L

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

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*?

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

SLD resolution

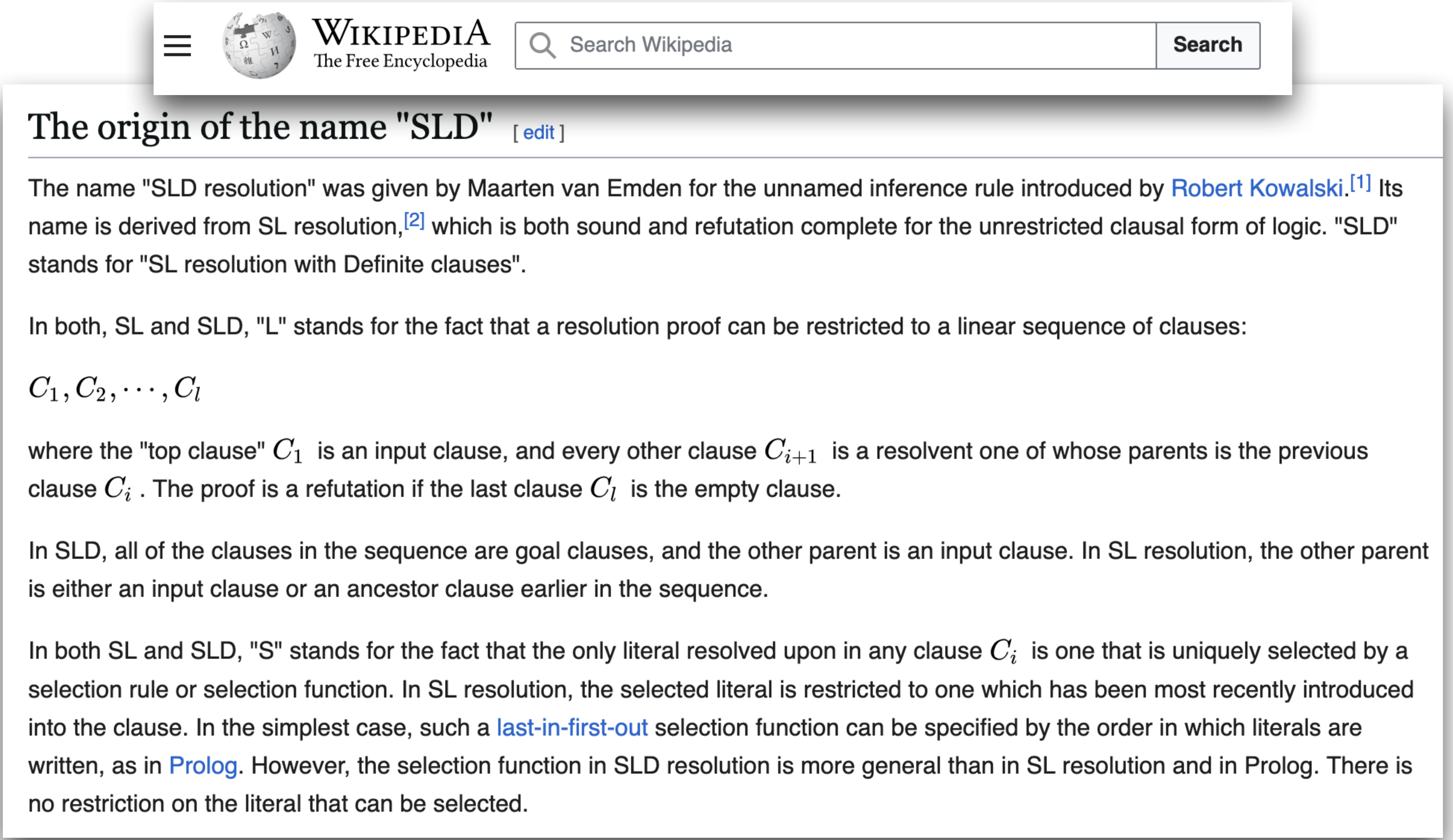
Idea: iteratively reduce the initial goal *g*
by applying one of the Horn clauses in L
to one of the atomic formulas in the goal;

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

$$\begin{array}{lll} ?-\ g & \nwarrow_{\sigma_1} & g_1 \\ & \nwarrow_{\sigma_2} & g_2 \\ & \nwarrow_{\sigma_3} & \ldots \\ & \nwarrow_{\sigma_m} & \Box \end{array}$$

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

SLD resolution



The origin of the name "SLD" [edit]

The name "SLD resolution" was given by Maarten van Emden for the unnamed inference rule introduced by Robert Kowalski.[1] Its name is derived from SL resolution,[2] which is both sound and refutation complete for the unrestricted clausal form of logic. "SLD" stands for "SL resolution with Definite clauses".

In both, SL and SLD, "L" stands for the fact that a resolution proof can be restricted to a linear sequence of clauses:

$$C_1, C_2, \cdots, C_l$$

where the "top clause" C_1 is an input clause, and every other clause C_{i+1} is a resolvent one of whose parents is the previous clause C_i. The proof is a refutation if the last clause C_l is the empty clause.

In SLD, all of the clauses in the sequence are goal clauses, and the other parent is an input clause. In SL resolution, the other parent is either an input clause or an ancestor clause earlier in the sequence.

In both SL and SLD, "S" stands for the fact that the only literal resolved upon in any clause C_i is one that is uniquely selected by a selection rule or selection function. In SL resolution, the selected literal is restricted to one which has been most recently introduced into the clause. In the simplest case, such a last-in-first-out selection function can be specified by the order in which literals are written, as in Prolog. However, the selection function in SLD resolution is more general than in SL resolution and in Prolog. There is no restriction on the literal that can be selected.

SLD resolution

$?\!-\, a_1, \ldots \boxed{a_i} \ldots, a_k$

$$\begin{array}{c} L \\ \ldots \\ \boxed{h \;:\!-\; b_1, \ldots, b_n} \\ \ldots \end{array}$$

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 $h :\!- b_1, \ldots, b_n$ in L whose head unifies with a_i;
3. let σ be a most general unifier $(a_i\sigma = h\sigma)$;
4. replace a_i with $b_1, \ldots, b_n$;
5. apply the computed substitution σ to the goal $(a_1, \ldots, b_1, \ldots, b_n, \ldots, a_k)\sigma$

Pay attention

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

✓ $(a_1, \ldots, b_1, \ldots, b_n, \ldots, a_k)\sigma$

✗ $a_1, \ldots, (b_1, \ldots, b_n)\sigma, \ldots, a_k$

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

Pay attention

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

$$\sigma : X \to T_{\Sigma,X}$$

$$Y \subseteq X$$

$$\sigma_{|Y}(x) \triangleq \begin{cases} \sigma(x) & \text{if } x \in Y \\ x & \text{otherwise} \end{cases}$$

we only record this partial information

$$a_1, \ldots, a_i, \ldots, a_k \;\; \nwarrow_{\widehat{\sigma}} \;\; (a_1, \ldots, (b_1, \ldots, b_n)\rho, \ldots, a_k)\sigma$$

$$\widehat{\sigma} \triangleq \sigma_{|vars(a_1,\ldots,a_k)}$$

Computed answer substitution

$$\begin{array}{lll} ?-\ g & \nwarrow_{\widehat{\sigma}_1} & g_1 \\ & \nwarrow_{\widehat{\sigma}_2} & g_2 \\ & \nwarrow_{\widehat{\sigma}_3} & \cdots \\ & \nwarrow_{\widehat{\sigma}_m} & \Box \end{array}$$

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

is called cas (computed answer substitution)

where

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

Example

$\Sigma_0 = \{0, \ldots\}$ $\Pi_3 = \{\mathsf{sum}, \ldots\}$

$\Sigma_1 = \{\mathsf{s}, \ldots\}$

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

L

(a fact)

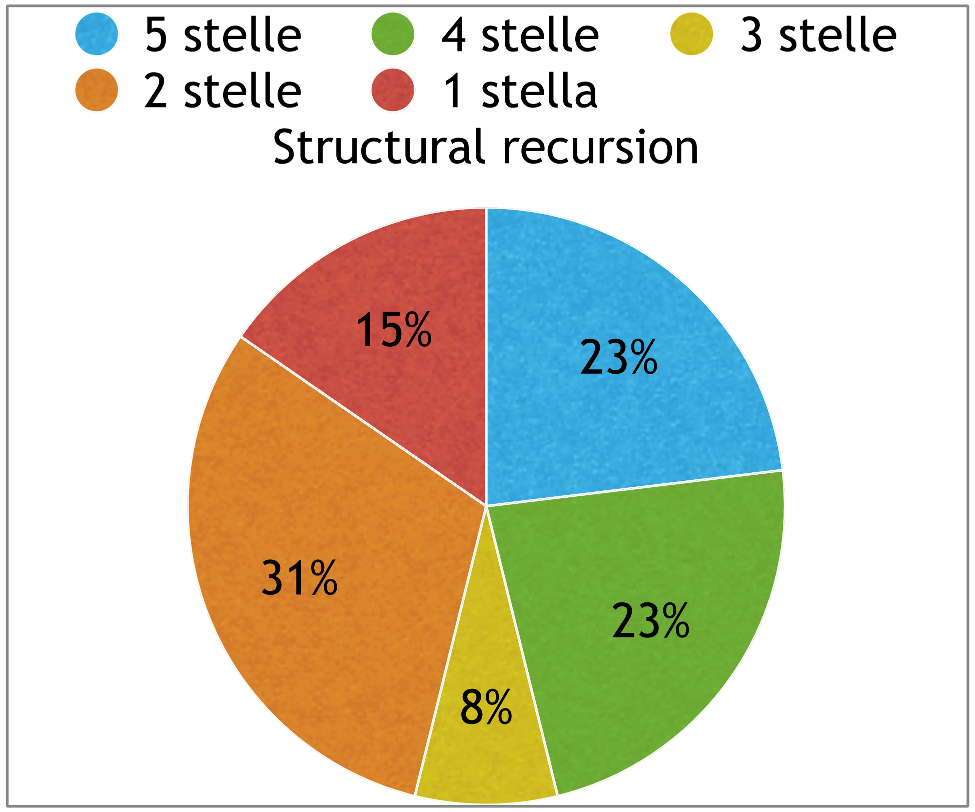
$\mathsf{sum}(0, y, y).$

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

in PROLOG each statement ends with dot

Structural recursion

the previous definition is an example of structural recursion



(over 13 answers)

2+2=?

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

L

$\mathsf{sum}(0, y, y).$

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

$\{\mathsf{sum}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0)), n) \overset{?}{=} \mathsf{sum}(0, y', y')\}$ fails

$\{\mathsf{sum}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0)), n) \overset{?}{=} \mathsf{sum}(\mathsf{s}(x_1), y_1, \mathsf{s}(z_1))\}$ succeeds

$$\sigma_1 = [x_1 = \mathsf{s}(0)\,,\; y_1 = \mathsf{s}(\mathsf{s}(0))\,,\; n = \mathsf{s}(z_1)]$$

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

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

$$\mathsf{sum}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0)), n) \;\nwarrow_{\widehat{\sigma}_1}\; (\,\mathsf{sum}(x_1, y_1, z_1)\,)\sigma_1$$

$$= \mathsf{sum}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), z_1)$$

1+2=?

our new goal

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

L

$\mathsf{sum}(0, y, y).$

$\mathsf{sum}(\mathsf{s}(x), y, \mathsf{s}(z)) :\!-\ \mathsf{sum}(x, y, z).$

$$\{\mathsf{sum}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), z_1) \overset{?}{=} \mathsf{sum}(0, y', y')\}$$ fails

$$\{\mathsf{sum}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), z_1) \overset{?}{=} \mathsf{sum}(\mathsf{s}(x_2), y_2, \mathsf{s}(z_2))\}$$ succeeds

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

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

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

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

$$\begin{aligned} \mathsf{sum}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0)), n) \ \nwarrow_{\widehat{\sigma}_1}\ & \mathsf{sum}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), z_1) \\ \nwarrow_{\widehat{\sigma}_2}\ & (\ \mathsf{sum}(x_2, y_2, z_2)\ )\sigma_2 \\ & = \mathsf{sum}(0, \mathsf{s}(\mathsf{s}(0)), z_2) \end{aligned}$$

0+2=?

our new goal

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

L

$\mathsf{sum}(0, y, y).$

$\mathsf{sum}(\mathsf{s}(x), y, \mathsf{s}(z)) :\!- \mathsf{sum}(x, y, z).$

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

succeeds

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

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

$$\mathsf{sum}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0)), n) \ \nwarrow_{\widehat{\sigma}_1}\ \mathsf{sum}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), z_1)$$

$$\nwarrow_{\widehat{\sigma}_2}\ \mathsf{sum}(0, \mathsf{s}(\mathsf{s}(0)), z_2)$$

$$\nwarrow_{\widehat{\sigma}_3}\ \square$$

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

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

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

$$2 + 2 = 4\ !$$

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

1+n=3?

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

L

$\mathsf{sum}(0, y, y).$

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

$\{\mathsf{sum}(\mathsf{s}(0), n, \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) \overset{?}{=} \mathsf{sum}(0, y', y')\}$ fails

$\{\mathsf{sum}(\mathsf{s}(0), n, \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) \overset{?}{=} \mathsf{sum}(\mathsf{s}(x_1), y_1, \mathsf{s}(z_1))\}$ succeeds

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

$$\widehat{\sigma}_1 = [\,]$$

$\widehat{\sigma}_1 = [\,]$

$$\mathsf{sum}(\mathsf{s}(0), n, \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) \ \nwarrow_{\widehat{\sigma}_1}\ (\,\mathsf{sum}(x_1, y_1, z_1)\,)\sigma_1$$
$$= \mathsf{sum}(0, n, \mathsf{s}(\mathsf{s}(0)))$$

0+n=2?

our new goal

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

L

$\mathsf{sum}(0, y, y).$

$\mathsf{sum}(\mathsf{s}(x), y, \mathsf{s}(z)) :\!-\, \mathsf{sum}(x, y, z).$

$$\{\mathsf{sum}(0, n, \mathsf{s}(\mathsf{s}(0))) \overset{?}{=} \mathsf{sum}(0, y_2, y_2)\}$$

succeeds

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

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

$\widehat{\sigma}_1 = [\,]$

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

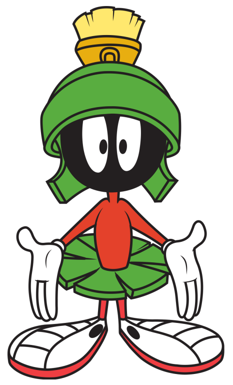
$$\mathsf{sum}(\mathsf{s}(0), n, \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) \ \nwarrow_{\widehat{\sigma}_1}\ \mathsf{sum}(0, n, \mathsf{s}(\mathsf{s}(0)))$$

$$\nwarrow_{\widehat{\sigma}_2}\ \square$$

$$(1 + n = 3) \Rightarrow n = 2\ !$$

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

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)

$\nwarrow$ green(W), martian(W)

$\nwarrow$ jumping(W), martian(W)

$\nwarrow$ martian(pgvdrk)

$\nwarrow \square$ $[W =$ pgvdrk$]$

martian(ngtrks)

$\nwarrow$ small(ngtrks), jumping(ngtrks)

$\nwarrow$ 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.

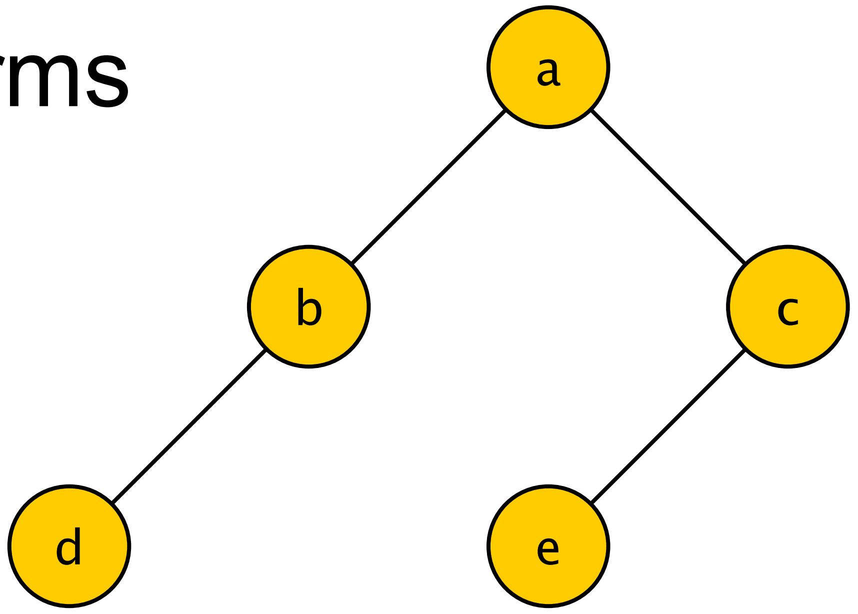
T is symmetric if you can draw a vertical line through the root node and then the right subtree is the mirror image of the left subtree (we are only interested in the structure, values are not relevant).

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.

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

Badge exercise

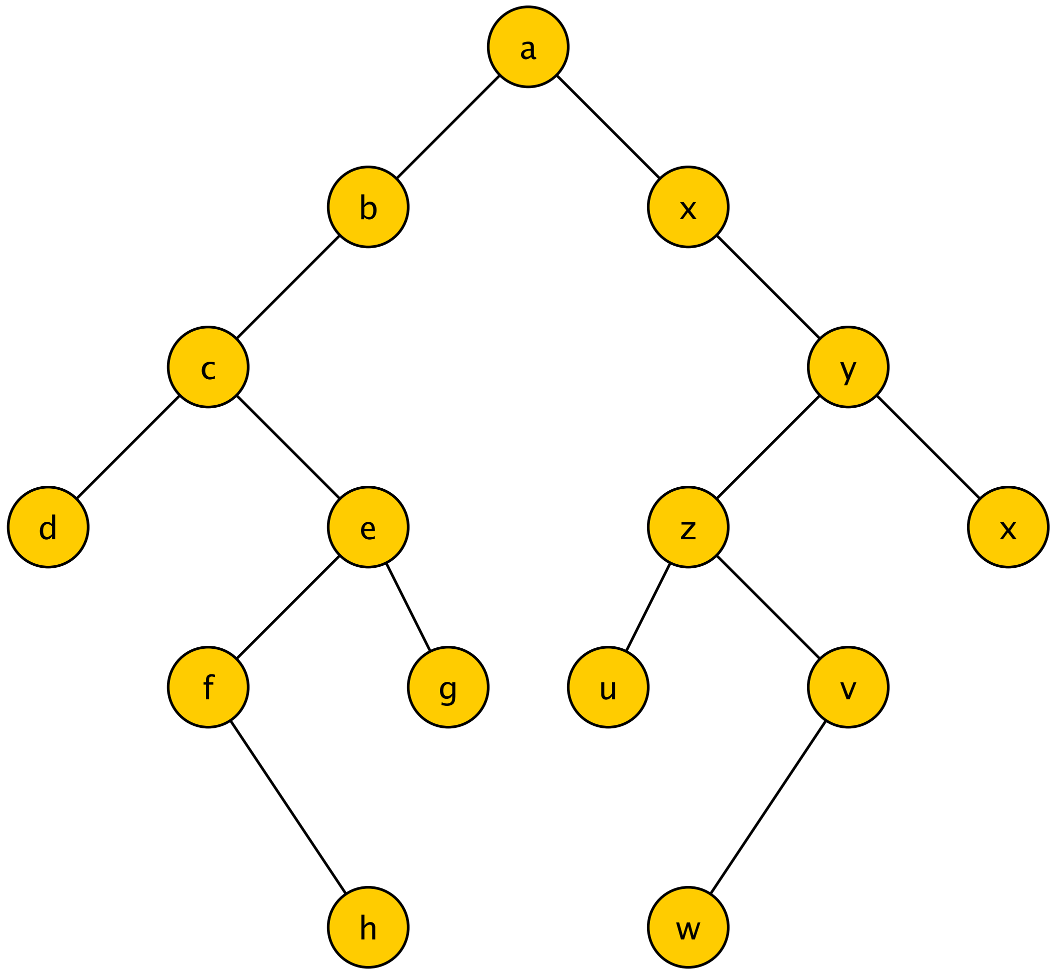
example:
trees as terms



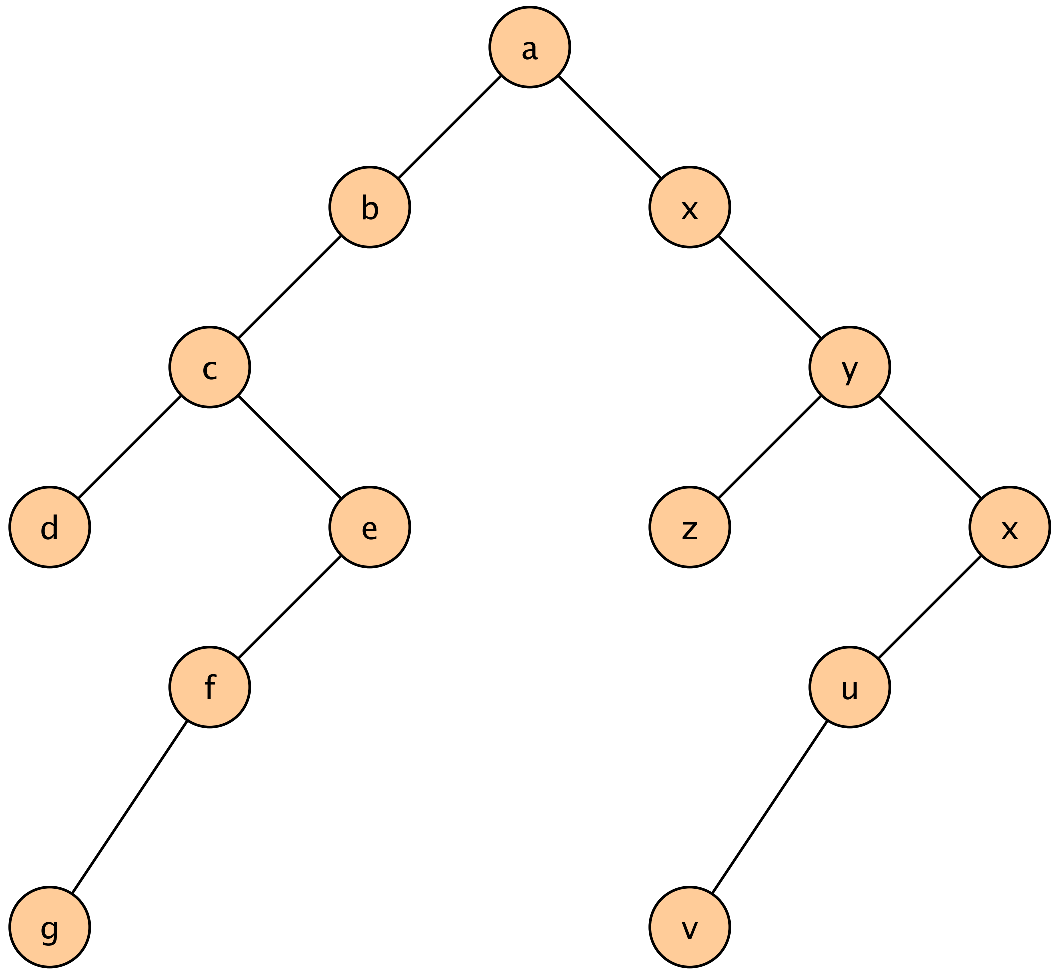
```
node( a , node( b , node( d , nil , nil ) ,
                    nil ) ,
          node( c , node( e , nil , nil ) ,
                    nil ) )
```

Badge exercise

an example of symmetric tree



and one that is not symmetric


a
b
x
c
y
d
e
z
x
f
u
g
v