TECHNISCHE UNIVERSITÄT DRESDEN

Computational Logic ∴ Group

Hannes Strass (based on slides by Michael Thielscher)

Faculty of Computer Science, Institute of Artificial Intelligence, Computational Logic Group

# SLD Resolution

Lecture 3, 23rd Oct 2023 // Foundations of Logic Programming, WS 2023/24

# Previously …

- A **substitution** replaces variables by terms, and is applied to terms.
- A **unifier** is a substitution that equates two terms when applied to them.
- The **Martelli-Montanari Algorithm** decides if a set of pairs of terms has a unifier and even outputs a (most general) unifier if one exists.
- The algorithm is **correct** (i.e., sound and complete) and **terminates**.

## Example

Consider $E_0 = \{ g(x, f(y)) \doteq g(a, z),\ f(x) \doteq f(a) \}$. The algorithm yields:

$$
\begin{aligned}
E_1 &= \{ x \doteq a,\ f(y) \doteq z,\ f(x) \doteq f(a) \} && \text{(decompose)}\\
E_2 &= \{ x \doteq a,\ z \doteq f(y),\ f(x) \doteq f(a) \} && \text{(orient)}\\
E_3 &= \{ x \doteq a,\ z \doteq f(y),\ f(a) \doteq f(a) \} && \text{(apply)}\\
E_4 &= \{ x \doteq a,\ z \doteq f(y),\ a \doteq a \} && \text{(decompose)}\\
E_5 &= \{ x \doteq a,\ z \doteq f(y) \} && \text{(decompose)}
\end{aligned}
$$

# Overview

The Logical Language of Programs

The Computation Mechanism: SLD Derivations

Choices and Their Impact

# The Logical Language of Programs

# Atoms, Term Bases, and Herbrand Bases

**Definition**

Let $TU_{F,V}$ be a term universe ($V$ Variables, $F$ function symbols) and $\Pi$ be a ranked alphabet of **predicate symbols**.

The **term base** $TB_{\Pi,F,V}$ (over $\Pi$, $F$, and $V$) is the smallest set of **atoms** with

1. if $p \in \Pi^{(0)}$ then $p \in TB_{\Pi,F,V}$;
2. if $p \in \Pi^{(n)}$ with $n \geq 1$ and $t_1, \ldots, t_n \in TU_{F,V}$, then $p(t_1, \ldots, t_n) \in TB_{\Pi,F,V}$.

$\rightsquigarrow$ Usual definition of atoms of first-order predicate logic.

**Definition**

Let $HU_F$ be a Herbrand universe, $\Pi$ ranked alphabet of predicate symbols.

The **Herbrand base** $HB_{\Pi,F}$ (over $\Pi$ and $F$) is given by $TB_{\Pi,F,\emptyset}$.

$\rightsquigarrow$ Herbrand base is the set of all variable-free (ground) atoms.

# Queries and Programs

**Definition**

- A **query** is a finite sequence $B_1, \ldots, B_n$ of atoms.
- The **empty query** (empty sequence of atoms) is denoted by $\Box$.
- A **(definite) clause** is an expression $H \leftarrow \vec{B}$ where

  $H$ is an atom (the **head** of the clause) and $\vec{B}$ is a query (the **body** of the clause).
- $H \leftarrow \vec{B}$ **unit clause** (also called: **fact**)

  $:\Longleftrightarrow \quad \vec{B}$ is empty (standard notation: $H \leftarrow$ )
- **Horn clause** $:\Longleftrightarrow$ clause or negated query
- **(definite) logic program** $:\Longleftrightarrow$ finite set of clauses

We will mostly use “program” and take it to mean “definite logic program”.

# Clauses and Queries: Examples

## Example

Let $x, y, z$ be variables. Then the following expressions are examples for …

- an atom:

$$direct(maui, honolulu)$$

- a query:

$$direct(frankfurt, x), direct(x, honolulu)$$

- a fact:

$$direct(maui, honolulu) \leftarrow$$

- a (definite) clause:

$$connection(x, y) \leftarrow direct(x, z), connection(z, y)$$

## Recall

In predicate logic, a **clause** is a disjunction of literals.

# Logical Reading of Clauses and Queries

## Clauses

A clause $H \leftarrow B_1, \ldots, B_n$ can be understood as the formula

$$\forall x_1, \ldots, x_k((B_1 \wedge \ldots \wedge B_n) \rightarrow H) \text{ (definite clause } \forall x_1, \ldots, x_k(\neg B_1 \vee \ldots \vee \neg B_n \vee H))$$

where $x_1, \ldots, x_k$ are the variables occurring in $H \leftarrow B_1, \ldots, B_n$.

(Thus a unit clause $H \leftarrow$ encodes $\forall x_1, \ldots, x_k H$.)

## Queries

A query $A_1, \ldots, A_n$ can be understood as the formula

$$\exists x_1, \ldots, x_k(A_1 \wedge \ldots \wedge A_n) \qquad \text{(or: } \neg\forall x_1, \ldots, x_k(\neg A_1 \vee \ldots \vee \neg A_n))$$

where $x_1, \ldots, x_k$ are the variables occurring in $A_1, \ldots, A_n$.

(Thus the empty query $\Box$ is equivalent to *true*.)

# What is Being Computed?

- ▷ A program $P$ can be interpreted as a set of axioms.
- ▷ A query $Q$ can be interpreted as the request for finding an instance $Q\theta$ which is a logical consequence of $P$.
- ▷ A successful derivation provides such a substitution $\theta$.
- ▷ In this way, the derivation is a proof of $Q\theta$ from the set of premises $P$.
- ▷ Thus SLD resolution provides a proof theory for programs.

$\rightsquigarrow$ To be continued in Lecture 4 (Correctness of SLD Resolution), where we introduce the corresponding *model theory*.

# How Do We Compute?

- A computation is a sequence of derivation steps.
- In each step an atom $A$ is selected in the current query and a program clause $H \leftarrow \vec{B}$ is chosen.
- If $A$ and $H$ are unifiable (in the sense of $A \doteq H$), then $A$ is replaced by $\vec{B}$ and an mgu of $A$ and $H$ is applied to the resulting query.
- The computation is successful if it ends with the empty query.
- The resulting answer substitution $\theta$ is obtained by combining the mgus of each step.

**Observation**

For atoms $A$ and $H$ to be unifiable, they must use the same predicate $p \in \Pi^{(n)}$ and furthermore, for $A = p(s_1, \ldots, s_n)$ and $H = p(t_1, \ldots, t_n)$ the resulting set $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ must have an mgu.

# The Computation Mechanism: SLD Derivations

# An SLD Derivation Step (No Variables)

**Note**

SLD = Selection rule driven Linear resolution for Definite clauses

**Definition**

Consider

- a program $P$
- a query $\vec{A}, B, \vec{C}$
- a clause $c = B \leftarrow \vec{B} \in P$

- $B$ is the **selected atom**
- The resulting query $\vec{A}, \vec{B}, \vec{C}$ is called the **SLD resolvent**
- Notation: $\vec{A}, B, \vec{C} \xrightarrow[c]{} \vec{A}, \vec{B}, \vec{C}$

# Example Ground Program and Query:

...

```
(1) happy :- sun, holidays.
(2) happy :- snow, holidays.
(3) snow :- cold, precipitation.
(4) cold :- winter.
(5) precipitation :- holidays.
(6) winter.
(7) holidays.

| ?- happy.
```

# An SLD Derivation Step (General Case)

**Definition**

Consider

- a program $P$
- a query $\vec{A}, B, \vec{C}$
- a clause $c \in P$
- a variant $H \leftarrow \vec{B}$ of $c$ that is variable disjoint with the query
- an mgu $\theta$ of $B$ and $H$

- **SLD resolvent** of $\vec{A}, B, \vec{C}$ and $c$ w.r.t. $B$ with mgu $\theta$ $:\Longleftrightarrow$ $(\vec{A}, \vec{B}, \vec{C})\theta$
- **SLD derivation step** $:\Longleftrightarrow$ $\vec{A}, B, \vec{C} \xrightarrow[c]{\theta} (\vec{A}, \vec{B}, \vec{C})\theta$
- **input clause** $:\Longleftrightarrow$ variant $H \leftarrow \vec{B}$ of $c$

We say: "Clause $c$ is **applicable** to atom $B$."

# Example Program and Query:

```
(1) add(X,0,X).
(2) add(X,s(Y),s(Z)) :- add(X,Y,Z).

(3) mul(X,0,0).
(4) mul(X,s(Y),Z) :- mul(X,Y,U), add(X,U,Z).
...
```

```
| ?- mul(s(s(0)),s(s(0)),V).

| ?- mul(V,W,s(s(0))).
```

# The 4 Steps of Resolving Query and Clause

| | | |
|---|---|---|
| 1. | **Selection** | Select an atom in the query. |
| 2. | **Renaming** | Rename the clause (if necessary). |
| 3. | **Instantiation** | Instantiate query and clause by an mgu of the selected atom and the head of the clause. |
| 4. | **Replacement** | Replace the instance of the selected atom by the instance of the body of the clause. |

# SLD Derivations

## Definition

A maximal sequence of SLD derivation steps

$$Q_0 \xrightarrow[c_1]{\theta_1} Q_1 \cdots Q_n \xrightarrow[c_{n+1}]{\theta_{n+1}} Q_{n+1} \cdots$$

is an **SLD derivation of** $P \cup \{Q_0\}$

$$:\Longleftrightarrow$$

- $Q_0, \ldots, Q_{n+1}, \ldots$ are queries, each empty or with one atom selected in it;
- $\theta_1, \ldots, \theta_{n+1}, \ldots$ are substitutions;
- $c_1, \ldots, c_{n+1}, \ldots$ are clauses of $P$;
- for every SLD derivation step, **standardization apart** holds.

# Standardization Apart

## Definition

For a sequence of SLD derivation steps as before, let $Q_{i-1} \xrightarrow[c_i]{\theta_i} Q_i$ be the $i$-th SLD derivation step for all $i \geq 1$ and $c_i'$ be the input clause used in that step. Then **standardization apart** holds (for the $i$-th step)

$$:\Longleftrightarrow \qquad Var(c_i') \cap \left( Var(Q_0) \cup \bigcup_{j=1}^{i-1} \left( Var(\theta_j) \cup Var(c_j') \right) \right) = \emptyset$$

Intuitively: The input clause is variable disjoint from the initial query and from the substitutions and input clauses used at earlier steps.

# Result of a Derivation

## Definition

Let $\xi = Q_0 \xrightarrow{\theta_1} Q_1 \cdots \xrightarrow{\theta_n} Q_n$ be a finite SLD derivation.

- $\xi$ **successful** $:\Longleftrightarrow \quad Q_n = \Box$
- $\xi$ **failed** $:\Longleftrightarrow \quad Q_n \neq \Box$ and no clause is applicable to selected atom of $Q_n$

## Definition

Let $\xi$ be successful.

- **computed answer substitution (cas)** of $Q_0$ (w.r.t. $\xi$) $:= \ (\theta_1 \cdots \theta_n)|_{Var(Q_0)}$
- **computed instance** of $Q_0 \ := \ Q_0\theta_1 \cdots \theta_n$

# Choices and Their Impact

# Choices

In each SLD derivation step the following four choices are made:

| | |
|---|---|
| 1 | Choice of the renaming |
| 2 | Choice of the most general unifier |
| 3 | Choice of the selected atom in the query |
| 4 | Choice of the program clause |

How do they influence the result?

# Choices

| | |
|---|---|
| 1 | **Choice of the renaming** |
| 2 | **Choice of the most general unifier** |
| 3 | Choice of the selected atom in the query |
| 4 | Choice of the program clause |

# Resultants: What is proved after a step?

**Definition**

The **resultant** associated with $Q \xrightarrow{\theta} Q_1$ is the implication $Q\theta \leftarrow Q_1$.

**Definition**

Consider

- a program $P$
- a resultant $R = Q \leftarrow \vec{A}, B, \vec{C}$
- a clause $c$
- a variant $H \leftarrow \vec{B}$ of $c$ that is variable disjoint with $R$
- an mgu $\theta$ of $B$ and $H$

**SLD resolvent** of resultant $R$ and $c$ w.r.t. $B$ with mgu $\theta$ $:= (Q \leftarrow \vec{A}, \vec{B}, \vec{C})\theta$

**SLD resultant step** $:= Q \leftarrow \vec{A}, B, \vec{C} \xrightarrow[c]{\theta} (Q \leftarrow \vec{A}, \vec{B}, \vec{C})\theta$

# Resultants and SLD derivations

## Definition

Consider an SLD derivation

$$\xi = Q_0 \xrightarrow[c_1]{\theta_1} Q_1 \cdots Q_n \xrightarrow[c_{n+1}]{\theta_{n+1}} Q_{n+1} \cdots$$

For $i \geq 0$,

$$R_i := Q_0\theta_1 \cdots \theta_i \leftarrow Q_i$$

is called the **resultant of level** $i$ of $\xi$.

The resultant $R_i$ describes what is proved after $i$ derivation steps.
In particular:

- $R_0 = Q_0 \leftarrow Q_0$
- $R_n = Q_0\theta_1 \cdots \theta_n$, if $Q_n = \square$ (because $\square \mathrel{\hat{=}}$ "true")

# Propagation (1)

**Definition**

The **selected atom** of a resultant $Q \leftarrow Q_i$ is the atom that is selected in $Q_i$.

**Lemma 3.12**

Suppose that $R \xrightarrow[c]{\theta} R_1$ and $R' \xrightarrow[c]{\theta'} R'_1$ are two SLD resultant steps where

- $R$ is an instance of $R'$,
- in $R$ and $R'$ atoms in the same positions are selected.

Then $R_1$ is an instance of $R'_1$.

Proof: [Apt97, page 55]

# Propagation (2)

**Corollary**

Suppose that $Q \xrightarrow[c]{\theta} Q_1$ and $Q' \xrightarrow[c]{\theta'} Q'_1$ are two SLD derivation steps where

- $Q$ is an instance of $Q'$,
- in $Q$ and $Q'$ atoms in the same positions are selected.

Then $Q_1$ is an instance of $Q'_1$.

# Similar SLD derivations

## Definition

Consider two (initial fragments of) SLD derivations

$$\xi = Q_0 \xrightarrow[c_1]{\theta_1} Q_1 \cdots Q_n \xrightarrow[c_{n+1}]{\theta_{n+1}} Q_{n+1}$$

$$\xi' = Q'_0 \xrightarrow[c_1]{\theta'_1} Q'_1 \cdots Q'_n \xrightarrow[c_{n+1}]{\theta'_{n+1}} Q'_{n+1}$$

We say that $\xi$ and $\xi'$ are **similar**

$$:\Longleftrightarrow$$

- $\text{length}(\xi) = \text{length}(\xi')$,
- $Q_0$ and $Q'_0$ are variants,
- in $Q_i$ and $Q'_i$ atoms in the same positions are selected ($i \in [0, n]$)

# A Theorem on Variants

**Theorem 3.18**

Consider two similar SLD derivations $\xi, \xi'$. For every $i \geq 0$, the resultants $R_i$ and $R'_i$ of level $i$ of $\xi$ and $\xi'$, respectively, are variants of each other.

**Proof.**

By induction on $i$.

Base Case ($i = 0$): $R_0 = Q_0 \leftarrow Q_0$ and $R'_0 = Q'_0 \leftarrow Q'_0$ are variants of each other.

Inductive Case ($i \rightsquigarrow i+1$): Consider $R_i \xrightarrow[c_{i+1}]{\theta_{i+1}} R_{i+1}$ and $R'_i \xrightarrow[c_{i+1}]{\theta'_{i+1}} R'_{i+1}$.

| | |
|---|---|
| | $R_i$ variant of $R'_i$ (induction hypothesis) |
| implies | $R_i$ instance of $R'_i$ and vice versa |
| implies | $R_{i+1}$ instance of $R'_{i+1}$ and vice versa (Lemma 3.12) |
| implies | $R_{i+1}$ variant of $R'_{i+1}$ |

$\square$

# Answer Substitutions of similar derivations

**Corollary**

Consider two similar successful SLD derivations of $Q_0$ with computed answer substitutions $\theta$ and $\eta$. Then $Q_0\theta$ and $Q_0\eta$ are variants of each other.

**Proof.**

By Theorem 3.18 applied to the final resultants $Q_0\theta \leftarrow \Box$ and $Q_0\eta \leftarrow \Box$ of these SLD derivations. $\Box$

This shows that choice 1 (choice of a renaming) and choice 2 (choice of an mgu) have no influence – modulo renaming – on the statement proved by a successful SLD derivation.

# Choices

| | |
|---|---|
| 1 | Choice of the renaming |
| 2 | Choice of the most general unifier |
| 3 | **Choice of the selected atom in the query** |
| 4 | Choice of the program clause |

# Selecting Atoms in Queries

## Definition

Let INIT be the set of *all* initial fragments of *all* possible SLD derivations in which the last query is non-empty.

- A **selection rule** is a function which for every $\xi^< \in \mathit{INIT}$ yields an occurrence of an atom in the last query of $\xi^<$.
- An SLD derivation $\xi$ is **via** a selection rule $\mathcal{R}$

$$:\Longleftrightarrow$$

  for every initial fragment $\xi^<$ of $\xi$ ending with a non-empty query $Q$, the selected atom of $Q$ is exactly $\mathcal{R}(\xi^<)$.

PROLOG employs the simple selection rule “select the leftmost atom.”

# Switching Lemma

## Lemma 3.32

Consider an SLD derivation $\xi = Q_0 \xrightarrow[c_1]{\theta_1} Q_1 \cdots Q_n \xrightarrow[c_{n+1}]{\theta_{n+1}} Q_{n+1} \xrightarrow[c_{n+2}]{\theta_{n+2}} Q_{n+2} \cdots$

where

- $Q_n$ includes two atoms $A_1$ and $A_2$,
- $A_1$ is the selected atom of $Q_n$,
- $A_2\theta_{n+1}$ is the selected atom of $Q_{n+1}$.

Then the SLD derivation $\xi' = Q_0 \xrightarrow[c_1]{\theta_1} Q_1 \cdots Q_n \xrightarrow[c_{n+2}]{\theta'_{n+1}} Q'_{n+1} \xrightarrow[c_{n+1}]{\theta'_{n+2}} Q_{n+2} \cdots$

for some $Q'_{n+1}$, $\theta'_{n+1}$, and $\theta'_{n+2}$ is such that:

- $A_2$ is the selected atom of $Q_n$
- $A_1\theta'_{n+1}$ is the selected atom of $Q'_{n+1}$
- $\theta'_{n+1}\theta'_{n+2} = \theta_{n+1}\theta_{n+2}$.

Proof: [Apt97, page 65]

# Independence of Selection Rule

**Theorem 3.33**

Let $\xi$ be a successful SLD derivation of $P \cup \{Q_0\}$.
Then for every selection rule $\mathcal{R}$ there exists a successful SLD derivation $\xi'$ of $P \cup \{Q_0\}$ via $\mathcal{R}$ such that

- cas of $Q_0$ (w.r.t. $\xi$) = cas of $Q_0$ (w.r.t. $\xi'$),
- $\xi$ and $\xi'$ are of the same length.

This shows that choice 3 (choice of a selected atom) has no influence in case of successful queries.

# Proof Sketch of Theorem 3.33.

Consider an SLD derivation $\xi = Q_0 \xrightarrow[c_1]{\theta_1} \cdots \xrightarrow[c_n]{\theta_n} Q_n = \Box$ that is not via $\mathcal{R}$.

Then there is a smallest $i \geq 1$ such that:

- $\xi$ is via $\mathcal{R}$ up to $Q_{i-1}$.
- $\mathcal{R}$ selects $A$ in $Q_i$.
- $A\theta_{i+1} \cdots \theta_{i+j}$ is the selected atom of $Q_{i+j}$ in $\xi$ for some $j \geq 1$ ($\xi$ is successful).

$$\xi = Q_0 \quad \cdots \quad Q_i \quad \cdots \quad Q_{i+j-1} \xrightarrow[c_{i+j}]{\theta_{i+j}} Q_{i+j} \xrightarrow[c_{i+j+1}]{\theta_{i+j+1}} Q_{i+j+1} \quad \cdots \quad Q_n$$

Apply Switching Lemma once:

$$\xi = Q_0 \quad \cdots \quad Q_i \quad \cdots \quad Q_{i+j-1} \xrightarrow[c_{i+j+1}]{\theta'_{i+j}} Q'_{i+j} \xrightarrow[c_{i+j}]{\theta'_{i+j}} Q_{i+j+1} \quad \cdots \quad Q_n$$

Apply Switching Lemma further $j - 1$ times.

# Choices

| | |
|---|---|
| 1 | Choice of the renaming |
| 2 | Choice of the most general unifier |
| 3 | Choice of the selected atom in the query |
| 4 | **Choice of the program clause** |

# SLD Trees visualize Search Space

**Definition**

**SLD Tree** for $P \cup \{Q_0\}$ via selection rule $\mathcal{R}$ $:\Longleftrightarrow$

- the branches are SLD derivations of $P \cup \{Q_0\}$ via $\mathcal{R}$;
- every node $Q$ with selected atom $A$ has exactly one descendant for every clause $c$ of $P$ which is applicable to $A$. This descendant is a resolvent of $Q$ and $c$ w.r.t. $A$.

**Definition**

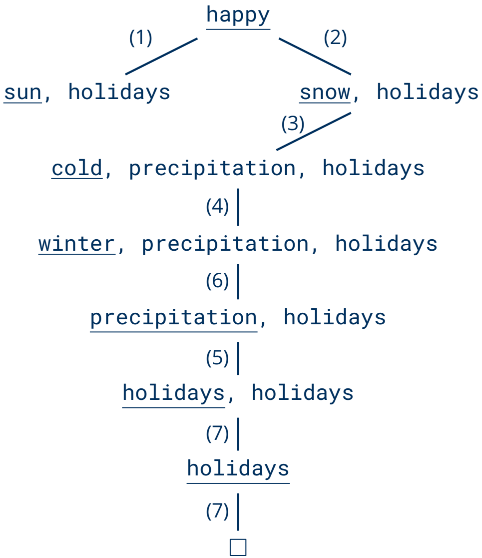
- SLD tree **successful** $:\Longleftrightarrow$ tree contains the node $\Box$.
- SLD tree **finitely failed** $:\Longleftrightarrow$ tree is finite and not successful.

SLD tree via "leftmost selection rule" corresponds to Prolog's search space.

# SLD Trees: Example

```
(1) happy :- sun, holidays.
(2) happy :- snow, holidays.
(3) snow :- cold, precipitation.
(4) cold :- winter.
(5) precipitation :- holidays.
(6) winter.
(7) holidays.

| ?- happy.
```



# Variant Independence

**Definition**

A selection rule $\mathcal{R}$ is **variant independent**

$$:\Longleftrightarrow$$

in all initial fragments of SLD derivations that are similar (cf. Slide 27), $\mathcal{R}$ chooses the atom in the same position in the last query.

**Example**

- The selection rule "select leftmost atom" is variant independent.
- The selection rule "select leftmost atom if query contains variable $x$, otherwise select rightmost atom" is variant dependent.

# The Branch Theorem

**Theorem 3.38**

Consider an SLD tree $\mathcal{T}$ for $P \cup \{Q_0\}$ via a variant independent selection rule $\mathcal{R}$. Then every SLD derivation of $P \cup \{Q_0\}$ via $\mathcal{R}$ is similar to a branch in $\mathcal{T}$.

This shows that choice 4 (choice of a program clause) has no influence on the search space as a whole.

# Proof Sketch of Theorem 3.38

Let $\xi = Q_0 \longrightarrow Q_1 \longrightarrow Q_2 \longrightarrow \ldots$ be an SLD derivation of $P \cup \{Q_0\}$ via $\mathcal{R}$.

By induction on $i \geq 0$ "find" branch $Q'_0, Q'_1, Q'_2, \ldots$ in $\mathcal{T}$ similar to $\xi$:

- $Q'_0 = Q_0$ (in particular they are variants).
- By definition of $\mathcal{T}$: The existence of $Q'_i$ implies the existence of $Q'_{i+1}$ (apply the same clause as to $Q_i$).
- Now $Q_0 \longrightarrow \ldots \longrightarrow Q_i$ and $Q'_0 \longrightarrow \ldots \longrightarrow Q'_i$ are similar.
- By variant independence of $\mathcal{R}$, in $Q_i$ and $Q'_i$ atoms in the same positions are selected.
- Thus $Q_0 \longrightarrow \ldots \longrightarrow Q_{i+1}$ and $Q'_0 \longrightarrow \ldots \longrightarrow Q'_{i+1}$ are also similar.

# Conclusion

## Summary

- A proof theory for (definite) logic programs is given by **SLD resolution**.
- A query is resolved with a (variant of a) program clause to another query.
- There are choices to be made (renaming of clause, mgu of query atom and clause, selected atom in query, program clause) with consequences.
- The search space can be visualized by (selection rule-induced) **SLD trees**.

## Suggested action points:

- Clarify the relationship of SLD resolution and “ordinary” FOL resolution.
- Obtain SLD resolutions (with mgus) for the examples on Slide 15.
- Use Prolog’s `trace` predicate to check your results.