To evaluate LPs, we use the procedural semantics.

\[ \text{LP} \Rightarrow \text{Query} \Rightarrow G \]

\[(G, \emptyset) \vdash (\Box, \mathcal{U}) \]

\[\text{identical subst.} \Rightarrow \text{performs binary SCD resolution} \Rightarrow \text{variables of } \mathcal{U} \text{ is the answer subst.}\]

The definition of \( \vdash \) still has 2 indeterminisms:

- **Indeterminism 1**: Which program clause \( K \) should be used for the next resolution step?

- **Indeterminism 2**: Which literal \( A_i \) of the query should be used for the next res. step?

For this reason, there could be several \((G_2, \mathcal{U}_2)\) with \((G_1, \mathcal{U}_1) \vdash (G_2, \mathcal{U}_2)\).

Ex. 4.3.1. Family example

Query: \(?- \text{ancestor}(Z, \text{aline}).\)

\[
\begin{align*}
(G, \emptyset) & \vdash (\neg \text{motherOf}(Z, \text{aline}), \mathcal{U}) \\
(G, \emptyset) & \vdash (\text{ancestor}(Z, \text{aline}), \mathcal{U})
\end{align*}
\]

\[
\begin{align*}
\{\text{ancestor}(Z, \text{aline})\} & \vdash (\neg \text{motherOf}(Z, \text{aline}), \mathcal{U}) \Rightarrow \{V/Z, X/\text{aline}\} \\
\{\text{ancestor}(Z, \text{aline})\} & \vdash (\text{ancestor}(Z, \text{aline}), \mathcal{U}) \Rightarrow \{V/Z, X/\text{aline}\}
\end{align*}
\]
\[
\{\text{ancestor}(z,\text{aline})\}, \emptyset \vdash (\text{find}(2,y), \text{ance}(y,\text{aline})) [V/2, X/\text{aline}]
\]

\text{Indet. 1 (choice of the prolog clause)} \quad (\star)

When continuing with (\star), then we have to solve

\text{Indet. 2 (choice of the next negative literal for the resolution step)}.

This could lead to the answer subst. \{2/senate\}
or to answer subst. \{2/susanne\}.

\Rightarrow \text{Indet. 1 influences the result}

The computation could also be non-terminating
(the rightmost path of the computation tree is infinite).

\Rightarrow \text{Indet. 2 influences the termination behavior.}

To improve efficiency, we do not want to build up the whole computation tree.

We first solve Indet. 2, i.e., we fix a strategy
which decides which literal of the query has
to be solved next (e.g., only perform resolution
with the leftmost literal of any query).

This would make the search tree much smaller.

Does this affect the completeness of
binary SCD resolution (i.e., would this reduced
search tree still contain all possible solutions)?

It turns out that this restriction is
still complete, i.e., the reduced search tree
still contains all solutions.

So one can take any selection strategy
which
selects the next literal of queries.

(Reason for the name “SCD resolution”)

Reason: Exchange Lemma

If one can first do a resolution step with \( \lnot A_i \)
and then with \( \lnot A_j \), then one could also
exchange these two steps and one obtains the
same result.

Lemma 4.3.2. (Exchange Lemma)

Let \( A_1, \ldots, A_n, B, C, \ldots, C_n, D, E_1, \ldots, E_m \in \Phi \) (\( \Sigma, \Delta, \Psi \))
where
\( \{ \lnot A_1, \ldots, \lnot A_n \}, \{ B, \lnot C, \ldots, \lnot C_n \}, \{ D, \lnot E_1, \ldots, \lnot E_m \} \)
are pairwise
variable-disjoint. Let \( \tau_1 \) be mgu of \( A_i \) and \( B \) and let
\( \tau_2 \) be the mgu of \( \sigma_1(A_j) \) and \( D \). So the following SCD-steps
are possible:

\[ \{ \lnot A_1, \ldots, \lnot A_i, \ldots, \lnot A_j, \ldots, \lnot A_n \} \quad \{ B, \lnot C, \ldots, \lnot C_n \} \]
\[ \mathcal{U}_1(\{\neg A_1, \ldots, \neg C_1, \ldots, \neg C_n, \ldots, \neg A_j, \ldots, \neg A_k\}) \quad \{D, \neg E_1, \ldots, \neg E_m\} \]

\[ \mathcal{U}_2(\mathcal{U}_1(\{\neg A_1, \ldots, \neg C_1, \ldots, \neg C_n, \ldots, \neg E_1, \ldots, \neg E_m, \ldots, \neg A_k\})) \]

Then there exists an mgu \( \mathcal{U}_1' \) of \( A_j \) and \( D \) and an mgu \( \mathcal{U}_2' \) of \( \mathcal{U}_1'(A_k) \) and \( B \). Thus, the following SLD-steps are also possible:

\[ \{\neg A_1, \ldots, \neg A_k, \ldots, \neg A_j, \ldots, \neg A_k\} \quad \{D, \neg E_1, \ldots, \neg E_m\} \]

\[ \mathcal{U}_1'(\{\neg A_1, \ldots, \neg C_1, \ldots, \neg C_n, \ldots, \neg E_1, \ldots, \neg E_m, \ldots, \neg A_k\}) \quad \{B, \neg C_1, \ldots, \neg C_n\} \]

\[ \mathcal{U}_2'(\mathcal{U}_1'(\{\neg A_1, \ldots, \neg C_1, \ldots, \neg C_n, \ldots, \neg E_1, \ldots, \neg E_m, \ldots, \neg A_k\})). \]

Moreover, the substitutions \( \mathcal{U}_2 \circ \mathcal{U}_1 \) and \( \mathcal{U}_2' \circ \mathcal{U}_1' \) are the same up to variable renaming (i.e., there is a variable renaming \( \mathcal{V} \) such that \( \mathcal{V}_2' \circ \mathcal{V}_1' = \mathcal{V} \circ \mathcal{U}_2 \circ \mathcal{U}_1 \)).

\[ \text{Ex. 4.3.3 Illustrates the exchange-lemma.} \]

\[ p(Z, Z) :- r(Z). \quad \text{Query: } Z \leftarrow p(X, Y), q(X). \]

\[ q(W). \]
\[ \{ \neg p(x, y), \neg q(x) \}, \emptyset \} \vdash \{ \neg r(z), \neg q(z) \}, \{ x/2, y/23 \} \]
\[ \vdash \{ \neg r(z) \}, \{ y/2, x/2, y/23 \} \]

Here, we first solved literal 1 and then literal 2.
By the exchange lemma, then one could also first solve literal 2 and then literal 1, and get the same result.
\[ \{ \neg p(x, y), \neg q(x) \}, \emptyset \} \vdash \{ \neg p(w, y) \}, \{ x/w \} \]
\[ \vdash \{ \neg r(y) \}, \{ y/w, z/y \} \circ \{ x/w \} \]
\[ \{ y/w, z/y, x/y \} \]

Indeed, the results are the same up to the variable renaming \( \mathcal{V} = \{ y/2, z/y \} \).

Proof of the exchange lemma (Lemma 4.3.2):
Since the clauses are variable-disjoint, the mgu \( \sigma_n \) of \( A_i \) and \( B \) does not modify \( D \), i.e., \( \sigma_n(D) = D \).
Then for the mgu \( \sigma_2 \) of \( \sigma_n(A_j) \) and \( D \), we have
\[ \sigma_2(\sigma_n(A_j)) = \sigma_2(D) = \sigma_2(\sigma_n(D)) \]

Thus, \( A_j \) and \( D \) are unifiable and \( \sigma_2 \circ \sigma_n \) is a unifier. Therefore, there also exists an mgu \( \sigma_n' \) of \( A_j \) and \( D \). Hence, there exists a substitution \( \sigma \) with
\[ \sigma_2 \circ \sigma_n = \sigma \circ \sigma_n' \] (every unifier can be obtained as an instance of the mgu).
So we can indeed start with a resolution step on $A_j$ and $D$. Now we have to show that afterwards, a resolution step on $\Gamma_n(A_i)$ and $B$ would be possible, i.e., that $\Gamma_n(A_i)$ and $B$ unify.

This holds, because $\Gamma$ is a unifier of $\Gamma_n(A_i)$ and $B$:

$$\Gamma(\Gamma_n(A_i)) = \Gamma_2(\Gamma_n(A_i)) \quad \text{by (8)}$$

$$= \Gamma_2(\Gamma_n(B)) \quad \text{as $\Gamma_n$ is unifier of $A_j$ and $B$}$$

$$= \Gamma(\Gamma_n(B)) \quad \text{by (8)}$$

$$= \Gamma(B) \quad \text{as $\Gamma_n$ is unifier of $A_j$ and $D$ and therefore, due to variable-disjointness, it does not modify $D$}$$

As $\Gamma$ is a unifier of $\Gamma_n(A_i)$ and $B$, they also have an unifier $\Gamma_2$. Thus, there exists a subst. $\delta$ with

$$\Gamma = \delta \circ \Gamma_2.$$  

It remains to show that $\Gamma_2 \circ \Gamma_n$ and $\Gamma_2 \circ \Gamma_n'$ are the same up to variable renaming. To show this, we prove that $\Gamma_2 \circ \Gamma_n$ is an instance of $\Gamma_2 \circ \Gamma_n'$ and that $\Gamma_2 \circ \Gamma_n'$ is an instance of $\Gamma_2 \circ \Gamma_n$.

To show that $\Gamma_2 \circ \Gamma_n$ is an instance of $\Gamma_2 \circ \Gamma_n'$:

$$\Gamma_2 \circ \Gamma_n = \Gamma \circ \Gamma_n' \quad \text{by (8)}$$
\[ = \delta \circ \sigma_2 \circ \sigma_1 \quad \text{by (4 \& 8)} \]

In a similar way, one can also show the other direction (course notes).

The exchange rule implies that we can use an arbitrary selection strategy for the next literal. Then every solution (path to \( \square \)) can still be found with the same answer set up to variable renaming.

In Prolog, one always selects the leftmost literal.

**Def 4.3.4 (Canonical Computation)**

A computation \((G_1, \sigma_1) \vdash^+ (G_2, \sigma_2) \vdash^+ \ldots\) is canonical iff in every resolution step one performs resolution with the leftmost literal of the query \(G_1\).

**Thm 4.3.5 (Solving Indet. 2)**

Let \( \mathcal{S} \) be a LP, let \( \mathcal{G} \) be a query.

For every computation \((\mathcal{G}, \sigma) \vdash^+ (\square, \sigma')\), there exists a canonical computation \((\mathcal{G}, \sigma) \vdash^+ (\square, \sigma')\) of the same length, where \( \sigma \) and \( \sigma' \) are the same up to variable
Yielding.

Proof Sketch: Consequence of the exchange lemma, because one can exchange the order of the resolution steps in the original computation until it becomes canonical.

(Technical proof in course notes).

This means that SLD resolution is still complete when restricting ourselves to Canonical Computations.

Ex. 4.3.6: In the computation tree of
Ex. 4.3.1 we can now remove all non-Canonical computations without losing any solution.
In this example, the infinite tree even becomes finite. This tree is called SLD tree.

Indet 2 influences the term behavior.
Since we know that Prolog only performs Canonical computations, this should be taken into account by the programmer to avoid undesired non-termination.

Ex. 4.3.7
Ex. 4.3.7
\[ \neg p. \]
\[ q(a). \]

?- q(b), p. \text{ terminates in Prolog, because there is no canonical comp. step starting in } \]
\[ \{ \neg q(b), \neg p \}, \emptyset \].

In contrast, the following query is non-terminating:

?- p, q(b).

Here, we have the Canonical computation:

\[ \{ \neg p, \neg q(b) \}, \emptyset \] \[ \rightarrow \] \[ \{ \neg p, \neg q(b) \}, \emptyset \] \[ \rightarrow \] \[ \ldots \]

Def 438. (SLD Tree)

Let \( S \) be a LP, let \( G \) be a query. The \textit{SLD Tree} of \( S \) wrt. \( G \) is a finite or infinite tree whose nodes are labeled with sequences of atomic formulas and whose edges are labeled with substitutions. The SLD tree is the smallest tree with:

1. If \( G = \{ \neg A_1, \ldots, \neg A_n \} \), then the root of the tree is labeled with \( A_1, \ldots, A_n \).
• If a node is labeled with $B_1, \ldots, B_n$, and $B_1$ is matched with the positive literals of $k$ variable-renamed program clauses $K_1, \ldots, K_k$ (where the clauses appear in this order in the program), then the node has $k$ children. The $i$-th child is labeled with the atoms that result from a canonical computation step using resolution with $K_i$. So if this computation has the form $(\{\neg B_1, \ldots, \neg B_n\}, \emptyset) \vdash (\{\neg C_1, \ldots, \neg C_m\}, \emptyset)$, then the $i$-th child is labeled with $C_1, \ldots, C_m$ and the edge to this child is labeled with $\sigma$ (restricted to the variables in $B_1, \ldots, B_n$).

• The answer substitutions can be obtained from the paths ending in $\Omega$. If the path is labeled by $\delta_1, \delta_2, \ldots, \delta_d$ then the answer subst. is $\delta_d \circ \ldots \circ \delta_2 \circ \delta_1$ (restricted to the variables of the original query $\mathcal{G}$).

• Besides successful paths, there can also be paths that end in a non-empty clause ("finite failure") and infinite paths.
To solve indeterminism, we have to determine the strategy that is used to build up the tree (and to search for $\square$).

**Possibility 1:** Breadth-First Search

- **Advantage:** Complete technique
  - (it finds every $\square$ in the SCD tree after finitely many steps $\Rightarrow$ semi-decision procedure)
  - **Disadvantage:** Inefficient

**Possibility 2:** Depth-First Search

- **Advantage:** Can be efficient if the solution is near the leftmost path
Disadvantage: not complete
→ programmer has to take the search strategy into account

In Prolog:
• depth-first search
• stops as soon as \( \square \) is found
• If the programmer enters ";", then one continues until the next \( \square \) is found.

**Ex 4.3.9** To demonstrate the effect of the order literals, we exchange the 2 literals in the body of the ancestor-rule. Now the SLD-tree is infinite (non-termination if one continues the search after the 2nd solution).

**Heuristics:** perform recursive calls only if arguments are sufficiently instantiated

**Ex. 4.3.10** To demonstrate the effect of the order of prog. clauses, we now exchange the
two ancestor rules. Now Prolog's search strategy immediately goes into the infinite path and does not terminate, i.e., it does not find any solution.

Heuristic: non-recursive clauses for a predicate \( p \) should come before recursive clauses for \( p \).

Prolog is not completely declarative, but one does have to consider its evaluation strategy.