Overview

SLD-Resolution is a proof procedure that:

- targets logical consequences of the form $P \models Q$ where $P$ is a definite logic program and $Q$ is a definite query;
- repeatedly applies a unique inference rule, the SLD-Resolution inference rule;
- makes use of a selection function that restricts the possible applications of the rule;
- relies on finding most general unifiers (mgu's) for sets of pairs of terms, which can be done thanks to the unification algorithm;
- can be represented by SLD-trees.

Reminder: definite clauses and queries

Definite logic programs consist of definite clauses, that are either facts or rules.

- Facts are atomic formulas $F$, also represented as $F \leftarrow$.
- Rules are formulas of the form $F_1 \land \ldots \land F_n \leftarrow F_0$ where $n > 0$ and $F_0, \ldots, F_n$ are atomic formulas, also represented as $F_0 \leftarrow F_1, \ldots, F_n$.

The variables that occur in facts or rules are implicitly universally quantified.

Definite queries are closed formulas of the form $\exists X_1 \ldots \exists X_k (F_0 \land \ldots \land F_n)$ where $k, n \in \mathbb{N}$ and $F_1, \ldots, F_n$ are atomic formulas.

Selection functions

Negations of definite queries are definite goals. Hence definite goals are logically equivalent to formulas of the form $\neg F_1 \lor \ldots \lor \neg F_n$ where $n > 0$ and $F_1, \ldots, F_n$ are atomic formulas, also represented as $\leftarrow F_1, \ldots, F_n$.

The empty goal is also considered in proofs, and corresponds to the case $n = 0$; it is denoted $\square$ and represents a contradiction.

A selection function is a function that takes a nonempty goal $\leftarrow F_1, \ldots, F_n$ as argument, and returns one of $F_1, \ldots, F_n$.

Intuitively, a selection function means the following.

To solve the goal $\leftarrow F_1, \ldots, F_n$, consider $F_i$ ($1 \leq i \leq n$) and solve the selected subgoal $\leftarrow F_i$.

We will obtain a substitution $\theta$ and we will then have to solve $\leftarrow (F_1, \ldots, F_{i-1}, F_{i+1}, \ldots, F_n) \theta$. 
The inference rule (1)

The SLD-Resolution inference rule takes two premises:
- a goal $G$ of the form $A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_m$,
- a definite clause $C$ of the form $B_0 \leftarrow B_1, \ldots, B_n$

such that:
- $A_i$ is the subgoal returned by the selection function applied to $G$;
- $A_i$ is of the form $p(t_0, \ldots, t_k)$ and $B_0$ is of the form $p(t'_0, \ldots, t'_k)$, where $p(t_0, \ldots, t_k)$ and $p(t'_0, \ldots, t'_k)$ share no common variable (always possible by renaming the variables in $C$).

The inference rule (2)

The consequent of the rule is the formula

$$
\leftarrow A_1\theta, \ldots, A_{i-1}\theta, B_1\theta, \ldots, B_n\theta, A_{i+1}\theta, \ldots, A_m\theta
$$

where $\theta$ is an mgu of $t_j$ and $t'_j$ for all nonnull $j \leq k$.

Prolog uses the selection function that always picks up the leftmost subgoal. In that case, the inference rule can be rewritten as:

$$
\leftarrow p(t_0, \ldots, t_k), A_2, \ldots, A_m \quad p(t'_0, \ldots, t'_k) \leftarrow B_1, \ldots, B_n
$$

where $\theta$ is an mgu of $t_j$ and $t'_j$ for all $j \leq k$.

SLD-tree (1)

Given a logic program $P$ and a query $Q$, we can represent all successful proofs of $Q$ from $P$, and all unsuccessful proof attempts of $Q$ from $P$, in the form of an SLD-tree.

Consider for instance the following definite logic program $P$.

\begin{align*}
grandfather(X, Z) & \leftarrow \text{father}(X, Y), \text{parent}(Y, Z). \\
p\text{arent}(X, Y) & \leftarrow \text{father}(X, Y). \\
p\text{arent}(X, Y) & \leftarrow \text{mother}(X, Y). \\
f\text{ather}(a, b). \\
m\text{other}(b, c).
\end{align*}

Consider the definite goal $\leftarrow \text{grandfather}(a, X)$.

SLD-tree (2)

The corresponding SLD-tree is:

$$
\leftarrow \text{grandfather}(a, X).
$$

$$
\leftarrow \text{father}(a, Y_0), \text{parent}(Y_0, X)
$$

$$
\leftarrow \text{mother}(b, X).
$$

The tree has two branches: one of them represents an unsuccessful attempt of proving $P \models \exists X \text{grandfather}(a, X)$; the other one represents a (complete) proof.
Prolog’s strategy (1)

Every time more than one clause can be used to match the leftmost subgoal, the SLD-tree branches.

Prolog’s strategy of selecting the first clause in the program that can match the current subgoal, and then applying backtracking, is equivalent to:

Prolog implements a depth-first search of the SLD-tree.

This strategy is not without drawbacks. Consider for instance the logic program $P$:

\[ p(X) \leftarrow p(X). \]

\[ p(a). \]

and the query $\exists X p(X)$.

SLD-derivation (1)

We want to show that $P \models \exists X_1 \ldots \exists X_k (F_1 \land \ldots \land F_n)$. Starting from the goal $G_0 = \leftarrow F_1, \ldots, F_n$, we apply the SLD-Resolution inference rule $p$ times, in accordance with the selection function $\mathcal{R}$ that has been chosen. A sequence of $p + 1$ goals is generated, that is called an SLD-derivation. Theoretically, we also consider the case where the rule is applied infinitely many times. Formally:

An SLD-derivation of $G_0$ (using $P$ and $\mathcal{R}$) is a finite or infinite sequence of goals

\[ G_0 \leadsto G_1 \ldots G_{p-1} \overset{\mathcal{R}}{\leadsto} G_p \ldots \]

where each $G_{i+1}$ is derived directly from $G_i$ and a renamed program clause $C_i$ via $\mathcal{R}$.

Prolog’s strategy (2)

The SLD-tree is the following:

\[ \leftarrow p(X). \]

\[ \leftarrow p(X). \]

\[ \leftarrow p(X). \]

A breadth-first search of the SLD-tree would result in a successful computation, whereas Prolog gets stuck in the leftmost branch.

SLD-derivation (2)

If the derivation is finite and the last goal is $\Box$, the derivation is a proof of $\exists X_1 \ldots \exists X_k (F_1 \land \ldots \land F_n)$ from $P$.

If the derivation is finite and the last goal is not $\Box$, the derivation corresponds to the beginning of an attempt of proving that $P \models \exists X_1 \ldots \exists X_k (F_1 \land \ldots \land F_n)$, or to a complete attempt of proving that $P \models \exists X_1 \ldots \exists X_k (F_1 \land \ldots \land F_n)$ that failed.

If the derivation is infinite, it corresponds to an unsuccessful attempt of proving that $P \models \exists X_1 \ldots \exists X_k (F_1 \land \ldots \land F_n)$ that runs forever.

The last case cannot be avoided in general. This is a corollary of a deep result (the undecidability of the notion of logical consequence).
SLD-derivation (3)

An SLD-derivation is an initial segment of a branch of an SLD-tree. It is a complete branch if either:
- the last goal is $\Box$ (the branch represents a complete proof), or
- the SLD-resolution inference rule cannot be applied to the last goal (the branch represents a complete proof attempt that failed), or
- the SLD-derivation is infinite.

To make sure that selected clauses and goals do not share common variables, and to avoid dealing with derivations that are the same up to a renaming of variables, it is convenient to assume that $i$ is added as a subscript to all variables occurring in the clause $C_i$ being used at step $i$.

Computed substitution (1)

Each finite SLD-derivation of the form
$$G_0 \stackrel{C_0}{\leadsto} G_1 \ldots G_{n-1} \stackrel{C_{n-1}}{\leadsto} G_n$$
yields a sequence $\theta_1, \ldots, \theta_n$ of mgu's. The composition
$$\theta = \begin{cases} 
\theta_1 \theta_2 \ldots \theta_n & \text{if } n > 0 \\
\varepsilon & \text{if } n = 0
\end{cases}$$
of mgu's is called the computed substitution of the derivation.

What matters eventually is the restriction of $\theta$ to the variables that occur in the query (in the initial goal).

Computed substitution (2)

First, we give a name to SLD-derivations that correspond to (complete) proofs from $P$ of the query $\exists X_1 \ldots \exists X_k (F_1 \land \ldots \land F_n)$:

A (finite) SLD-derivation of the form
$$G_0 \stackrel{C_0}{\leadsto} G_1 \ldots G_p \stackrel{C_p}{\leadsto} G_{p+1}$$
is called an SLD-refutation of $G_0$.

Then we define the part of the computed substitution that matters:

The restricted to the variables in $G_0$ of the computed substitution of an SLD-refutation of $G_0$ is called a computed answer substitution of $G_0$.

Example (1)

If $\theta$ is the computed answer substitution of $G_0$, what has been proved in fact is that $P \models (F_1 \land \ldots \land F_n)\theta$.

Consider the following logic program $P$:
$$C_0 = \text{admires}(X, Y) \leftarrow \text{famous}(Y), \text{nice}(Y).$$
$$C_1 = \text{famous}(X) \leftarrow \text{tennis_player}(X).$$
$$C_2 = \text{tennis_player}(\text{hewitt}).$$
$$C_3 = \text{impetuous}(\text{hewitt}).$$
$$C_4 = \text{tennis_player}(\text{rafter}).$$
$$C_5 = \text{nice}(\text{rafter}).$$
$$C_6 = \text{tennis_player}(\text{agassi}).$$
$$C_7 = \text{composed}(\text{agassi}).$$

Let us show that $P \models \exists X \exists Y \text{admires}(X, Y)$ using Prolog's selection function.
Example (2)

Let us make another attempt.

\[ G_0 \leftarrow \text{admires}(X, Y). \]

We use the rule \( \text{admires}(X_0, Y_0) \leftarrow \text{famous}(Y_0), \text{nice}(Y_0) \) (note the renaming of variables).

\[ G_1 \leftarrow \text{famous}(Y_0), \text{nice}(Y_0) \text{ and } \theta_1 = \{Y/Y_0\}. \]

We use the rule \( \text{famous}(X_1) \leftarrow \text{tennis_player}(X_1) \) (note the renaming of variables).

\[ G_2 \leftarrow \text{tennis_player}(X_1), \text{nice}(X_1) \text{ and } \theta_2 = \{Y_0/X_1\}. \]

We use the rule \( \text{famous}(Y) \leftarrow \text{tennis_player}(agassi) \).

\[ G_3 \leftarrow \text{nice}(agassi) \text{ and } \theta_3 = \{X_1/agassi\}. \]

\( G_0 \overset{C_0}{\rightarrow} G_1 \overset{C_1}{\rightarrow} G_2 \overset{C_2}{\rightarrow} G_3 \overset{C_3}{\rightarrow} G_4 \) is a maximal SLD-derivation that represents an attempt of proving \( P \models \exists X \exists Y \text{admires}(X, Y) \) that fails.

Example (3)

Hence trying to get a proof of \( \exists X \exists Y \text{admires}(X, Y) \), we get a proof of \( \forall X \text{admires}(X, \text{rafter}) \). Note that we get much more than we asked!

Now let us use another selection function: we select the rightmost subgoal instead of the leftmost one.

\[ G_0 \leftarrow \text{admires}(X, Y). \]

We use the rule \( \text{admires}(X_0, Y_0) \leftarrow \text{famous}(Y_0), \text{nice}(Y_0) \) (note the renaming of variables).

\[ G_1 \leftarrow \text{famous}(Y_0), \text{nice}(Y_0) \text{ and } \theta_1 = \{Y/Y_0\}. \]

We use the rule \( \text{famous}(X_1) \leftarrow \text{tennis_player}(X_1) \) (note the renaming of variables).

\[ G_2 \leftarrow \text{tennis_player}(X_1), \text{nice}(X_1) \text{ and } \theta_2 = \{Y_0/X_1\}. \]

We use the fact \( \text{nice}(\text{rafter}) \).

\[ G_3 \leftarrow \text{nice}(\text{rafter}) \text{ and } \theta_3 = \{X_1/\text{rafter}\}. \]

We use the fact \( \text{nice}(\text{rafter}) \).

\[ G_4 = \square \text{ and } \theta_4 = \varepsilon. \]

Example (4)

\( G_0 \overset{C_0}{\rightarrow} G_1 \overset{C_1}{\rightarrow} G_2 \overset{C_2}{\rightarrow} G_3 \overset{C_3}{\rightarrow} G_4 \) is an SLD-refutation of \( G_0 \). It represents a proof of \( P \models \exists X \exists Y \text{admires}(X, Y) \).

\[ \theta_1 \theta_2 \theta_3 \theta_4 = \{Y/Y_0\}\{Y_0/X_1\}\{X_1/\text{rafter}\}\varepsilon = \{Y/\text{rafter}, Y_0/\text{rafter}, X_1/\text{rafter}\}. \]

The computed answer substitution is \( \theta = \{Y/\text{rafter}\} \).

What has been proved actually is that \( P \models \text{admires}(X, \text{rafter}) \).

Note that when we apply the computed answer substitution to the matrix of the query (the query with the quantifiers removed), we get a formula that can contain variables. As usually, these variables are implicitly universally quantified.

Example (5)

Hence trying to get a proof of \( \exists X \exists Y \text{admires}(X, Y) \), we get a proof of \( \forall X \text{admires}(X, \text{rafter}) \). Note that we get much more than we asked!

Now let us use another selection function: we select the rightmost subgoal instead of the leftmost one.

\[ G_0 \leftarrow \text{admires}(X, Y). \]

We use the rule \( \text{admires}(X_0, Y_0) \leftarrow \text{famous}(Y_0), \text{nice}(Y_0) \) (note the renaming of variables).

\[ G_1 \leftarrow \text{famous}(Y_0), \text{nice}(Y_0) \text{ and } \theta_1 = \{Y/Y_0\}. \]

We use the fact \( \text{nice}(\text{rafter}) \).

\[ G_2 \leftarrow \text{famous}(\text{rafter}) \text{ and } \theta_2 = \{Y_0/\text{rafter}\}. \]

We use the rule \( \text{famous}(X_2) \leftarrow \text{tennis_player}(X_2) \).

\[ G_3 \leftarrow \text{tennis_player}(\text{rafter}) \text{ and } \theta_3 = \{X_2/\text{rafter}\}. \]
Example (4)

We use the fact $tennis\_player(\text{rafter})$.

$G_4 = \Box$ and $\theta_4 = \varepsilon$.

The computed answer substitution is the same.

Note that for this example, the second selection function is better than the first one: there is a unique maximal SLD-derivation for the second selection function, whereas there are three maximal SLD-derivations for the first one, two of each represent failed attempts of proving $\exists X \forall Y admires(X, Y)$.

Independence of computation rule

What has been observed is a general fact.

Let $P$ be a definite program and let $G$ be a definite goal. Let $\mathbb{R}_1$ and $\mathbb{R}_2$ denote two selection functions.

- The set of possible SLD-derivations of $G$ using $P$ and $\mathbb{R}_1$ is not isomorphic to the set of possible SLD-derivations of $G$ using $P$ and $\mathbb{R}_2$. The corresponding SLD-trees might differ in their number of branches and in the length of their branches. One can have an infinite branch in one tree and only finite branches in the other.

- But the possible SLD-refutations of $G$ using $P$ and $\mathbb{R}_1$ yield the same computed answer substitutions as the possible SLD-refutations of $G$ using $P$ and $\mathbb{R}_2$.

SLD-trees (1)

So theoretically—as far as computed answer substitutions are concerned—any selection function is as good as any other.

SLD-trees have been described and used informally. Formally:

Let $P$ be a definite program, $G_0$ a definite goal, and $\mathbb{R}$ a computation rule. The SLD-tree of $G_0$ (using $P$ and $\mathbb{R}$) is a (possibly infinite) labeled tree such that:

- the root of the tree is labeled with $G_0$;

- if the tree contains a node labeled with $G_i$, and there is a renamed clause $C_i \in P$ such that $G_{i+1}$ is derived from $G_i$ and $C_i$ via $\mathbb{R}$, then the node labeled with $G_i$ has a child labeled with $G_{i+1}$.

SLD-trees (2)

As mentioned, there is an isomorphism between the maximal branches of the SLD-tree of $G_0$ and the maximal derivations of $G_0$.

It would be good if we could avoid exploring the branches of the SLD-tree that are either infinite or correspond to failed attempts of deriving a contradiction from $G_0$. Unfortunately, this cannot be achieved.

Having selected a subgoal in the current subgoal, there is no perfect strategy for choosing the fact or rule in the program among all those that match the subgoal, and avoid exploring bad branches.

A breadth-first search of the SLD-tree would find a good branch whenever there is one.
Soundness and completeness (1)

Soundness and completeness are the two fundamental results that can characterize a proof system.

Soundness means that if \( F \) can be derived from \( P \) using the rules of inference of the proof system, then \( F \) is a logical consequence of \( P \).

Proving soundness is always easy: it suffices to check that the inference rules of the proof system derive logical consequences of the premises.

In the particular case of SLD-resolution, soundness means a bit more and is not so trivial, because it takes into account the notion of computed answer substitution, hence not only the existential statement we try to prove, but also the stronger statements that are actually derived.

Soundness and completeness (2)

**Proposition (soundness):** Let \( P \) be a definite program, \( \exists \) a selection function, and \( \theta \) a computed answer substitution for a definite goal \( G = \leftarrow A_1, \ldots, A_m \) w.r.t. an SLD-refutation of \( G \) using \( P \) and \( \exists \). Then \( P \models (A_1 \land \ldots \land A_m)\theta \).

Note that as a consequence, \( P \models \exists(A_1 \land \ldots \land A_m) \) (i.e., \( P \) logically implies the existential closure of \( A_1 \land \ldots \land A_m \)).

Remember that \( (A_1 \land \ldots \land A_m)\theta \) can contain variables, in which case \( (A_1 \land \ldots \land A_m)\theta \) is identified with its universal closure \( \forall((A_1 \land \ldots \land A_m)\theta) \).

It is suggested that you read the proof in the textbook, as well as section 3.6 (this section is inessential and will not be covered in lectures).

Soundness and completeness (3)

Completeness is usually much more difficult to prove, and much deeper. It says that the proof system contains enough rules to prove that \( P \models F \) whenever this is the case.

In the particular case of SLD-resolution, computed answer substitutions add an extra issue.

**Proposition (completeness):** Let \( P \) be a definite program, \( G = \leftarrow A_1, \ldots, A_m \) a definite goal, and \( \sigma \) a substitution such that \( P \models (A_1 \land \ldots \land A_n)\sigma \). For all selection functions \( \exists \), there exists a refutation of \( G \) using \( P \) and \( \exists \) with computed answer substitution \( \theta \) such that \( (A_1 \land \ldots \land A_n)\sigma \) is an instance of \( (A_1 \land \ldots \land A_n)\theta \).

Soundness and completeness (4)

In particular, if \( P \models \exists(A_1 \land \ldots \land A_n) \), then there exists a refutation of \( G \) using \( P \) and \( \exists \) with computed answer substitution \( \theta \) such that \( P \models (A_1 \land \ldots \land A_n)\theta \).