Lecture notes 23.0

SLDNF-Resolution (1)

Let $P$ be a definite logic program. We know the following.

(1) Let atomic formulas $F_1, \ldots, F_n$ be given. Then

\[ P \models \exists (F_1 \land \ldots \land F_n) \iff \text{we can prove } \exists (F_1 \land \ldots \land F_n) \]

from $P$ by SLD-Resolution.

(2) Let atomic formula $G$ be given. Then $\text{comp}(P) \models \forall \neg G$ iff we can prove $\forall \neg G$ from $P$ by negation as failure.

Proving $\forall \neg G$ from $P$ by negation as failure means: try to prove that $P \models G$ by SLD-Resolution and get the answer no.

For (1), an arbitrary selection function can be used; in particular, Prolog's selection function will do. For (2), not any selection function can be used; in particular, a fair selection function will do, but not Prolog's selection function.

Introduction (2)

Proving that $\text{comp}(P) \models \forall \neg G$ shows that $\forall \neg G$ holds in the minimal Herbrand model $M_P$ of $P$, which is quite interesting to know if $M_P$ is the only intended interpretation of $P$.

Two generalizations are now in view:

- Consider more general queries built from both atomic formulas and negations of atomic formulas.
- Consider more general logic programs containing rules whose bodies are built from atomic formulas and negations of atomic formulas.

Both are based on a generalization of SLD-resolution, called SLDNF-Resolution (i.e., SLD-Resolution + Negation as Failure).

General goals (1)

Let $P$ be the following definite logic program.

\begin{align*}
on(c, b). \\
on(b, a).
\end{align*}

The definite query $\exists X \exists Y \text{on}(X, Y)$ is a logical consequence of $P$. SLD-Resolution yields $\{X/c, Y/b\}$ and $\{X/b, Y/a\}$ as computed answer substitutions. The corresponding goal is $\forall X \forall Y \neg \text{on}(X, Y)$, also written $\neg \text{on}(X, Y)$.

$\neg \text{on}(X, Y), \neg \text{on}(Z, X)$ is a general goal. It should not be interpreted as $\forall X \forall Y \forall Z (\neg \text{on}(X, Y) \land \neg \text{on}(Z, X))$, and the associated query is not $\exists X \exists Y \exists Z (\text{on}(X, Y) \land \neg \text{on}(Z, X))$.

Behind this flaw is the fact that negation as failure can derive a statement such as $\neg \exists Z \text{on}(Z, c)$, not $\exists Z \neg \text{on}(Z, c)$. 
General goals (2)

This ‘explains’ why the general goal $\leftarrow \text{on}(X, Y), \neg \text{on}(Z, X)$ is interpreted as

$$\forall X \forall Y \exists Z (\neg \text{on}(X, Y) \land \neg \text{on}(Z, X))$$

and the associated query as

$$\exists X \forall Y (\text{on}(X, Y) \land \forall Z \neg \text{on}(Z, X))$$.

A general goal is defined as a goal of the form

$$\leftarrow L_1, \ldots L_n \quad n \geq 0$$

where each $L_i$ is an atomic or the negation of an atomic formula. But how are the variables implicitly quantified?

---

General goals (3)

As the previous example shows, the (free) variables that occur in a general goal are not all implicitly universally quantified. This is not in accordance with the convention that free variables in a formula are implicitly universally quantified, and it is definitely a messy situation.

For general goals, variables that have occurrences in positive goals are implicitly universally quantified, whereas other variables are implicitly existentially quantified. Moreover, universal quantifiers implicitly come before existential quantifiers.

For instance, $\leftarrow p(X, Y), \neg q(X, Z), \neg r(X, U, V), s(U)$ should be interpreted as

$$\forall X \forall Y \forall U \exists V (p(X, Y) \land \neg q(X, Z) \land \neg r(X, U, V) \land s(U))$$

---

General goals (4)

The corresponding explicit query is

$$\exists X \exists Y \forall Z \forall V (p(X, Y) \land \neg q(X, Z) \land \neg r(X, U, V) \land s(U))$$

The rules that govern how variables in general queries are implicitly quantified find their justification from the procedural interpretation of logic programs, not from the declarative interpretation of logic programs (based on the semantics of first-order logic).

With the logic program introduced previously, the quantifiers in the general query $\exists X \exists Y (\text{on}(X, Y) \land \forall Z \neg \text{on}(Z, X))$ are procedurally justified as follows.

First we try to solve the subgoal $\leftarrow \text{on}(Z, c)$. If fails.

Then we try to solve the subgoal $\leftarrow \text{on}(Z, c)$. If fails.

So the computed answer substitution is $\theta = \{X/c, Y/b\}$. This means that what has actually been proved is:

$$\forall ((\text{on}(X, Y) \land \neg \text{on}(Z, X)) \theta) \equiv \forall Z (\text{on}(c, b) \land \neg \text{on}(Z, c))$$

So in particular, we have shown that

$$\exists X \exists Y (\text{on}(X, Y) \land \forall Z \neg \text{on}(Z, X))$$.

If we want to discover two distinct sets, we might try to prove:

$$\exists X \exists Y \exists Z (\text{set}(X) \land \text{set}(Y) \land \text{member}(Z, X) \land \neg \text{member}(Z, Y))$$

Note that procedurally, the order of the last two atoms matters though declaratively, it does not.

---

General goals (5)

Then we try to solve the subgoal $\leftarrow \text{on}(Z, c)$. If fails.

So the computed answer substitution is $\theta = \{X/c, Y/b\}$. This means that what has actually been proved is:

$$\forall ((\text{on}(X, Y) \land \neg \text{on}(Z, X)) \theta) \equiv \forall Z (\text{on}(c, b) \land \neg \text{on}(Z, c))$$

So in particular, we have shown that

$$\exists X \exists Y (\text{on}(X, Y) \land \forall Z \neg \text{on}(Z, X))$$.

If we want to discover two distinct sets, we might try to prove:

$$\exists X \exists Y \exists Z (\text{set}(X) \land \text{set}(Y) \land \text{member}(Z, X) \land \neg \text{member}(Z, Y))$$

Note that procedurally, the order of the last two atoms matters though declaratively, it does not.
The first definition of SLDNF-Resolution is for general queries and definite logic programs.

Let $P$ be a definite logic program, $G_0$ a general goal, and $R$ a computation rule. An **SLDNF-derivation** of $G_0$ (using $P$ and $R$) is a finite or infinite sequence of general goals:

$$ G_0 \xrightarrow{C_0} G_1 \xrightarrow{C_1} \ldots \xrightarrow{C_{n-1}} G_n \ldots $$

such that for all $i$ smaller than the length of the derivation:

- the $R$-selected literal in $G_i$ is positive and $G_{i+1}$ is derived from $G_i$ and $C_i$ by one step of SLD-Resolution, or
- the $R$-selected literal in $G_i$ is of the form $\neg A$, the goal $A$ has a finitely failed SLD-tree and $G_{i+1}$ is obtained from $G_i$ by removing $\neg A$ (in which case $C_i = FF$).

**SLDNF-Resolution (2)**

*FF* is a special marker for ‘finitely failed.’

Basically, a derivation is represented by a main tree, plus auxiliary trees when the selected subgoal is negative.

For instance, with the program $P$ considered above, the SLDNF-derivation of $\leftarrow on(X, Y), \neg on(Z, X)$ can be represented by the following forest:

**SLDNF-Resolution (3)**

There can be three kinds of finite branches in the main tree of an SLDNF-Resolution:

- Finite branches that end with $\square$, corresponding to refutations associated with one computed answer substitution.
- Finite branches that end with *FF*:
  - either because the selected literal is positive and does not unify with the head of any clause,
  - or the selected literal is negative and finitely failed.
- Finite branches that end with a subgoal of the form $\leftarrow A$ when the derivation is stuck because $A$ is infinitely failed (the SLD-tree for $\leftarrow A$ is infinite and has no success branch).
SLDNF-Resolution (5)

SLDNF-Resolution is a sound proof procedure:

**Proposition:** Let \( P \) be a definite program and \( \leftarrow L_1, \ldots, L_n \) a general goal. If \( \leftarrow L_1, \ldots, L_n \) has an SLDNF-refutation with computed answer substitution \( \theta \), then \( \text{comp}(P) \models \forall (L_1 \theta \land \ldots \land L_n \theta) \).

Variables that only occur in negative literals are not instanciated by \( \theta \), which is in accordance with the fact that these variables are implicitly universally quantified in the corresponding query.

Is SLDNF-Resolution a complete proof procedure for definite programs and general queries? Yes if you quantify variables in a general query as we did. The textbook is misleading on this issue.

General Logic Programs (1)

SLDNF-Resolution becomes more powerful when negation is used not only in queries, but also in programs.

A **general clause** is a formula of the form

\[
A_0 \leftarrow L_1, \ldots, L_n \quad n \geq 0
\]

where \( A_0 \) is an atomic formula and each \( L_i \) is atomic or the negation of an atomic formula.

A **general logic program** is a set of general clauses.

Note that negation can occur in the body of a general clause, but not in the head. In particular, there is no negative fact in a general logic program.

General Logic Programs (2)

The Herbrand base is still a model of a general logic program. But general logic programs can have more than one minimal Herbrand model. For instance, \( p(a) \leftarrow \neg p(b) \) has both \( \{p(a)\} \) and \( \{p(b)\} \) as minimal Herbrand models.

Completion of definite logic programs characterizes negation as failure. Completion of general logic programs does not characterize negation as failure.

For instance, consider \( P = \{p(X) \leftarrow \neg p(X)\} \). The completion of \( P \) contains the formula

\[
\forall X_1[p(X_1) \leftrightarrow \exists X (X_1 \equiv X \land \neg p(X))],
\]

which is unsatisfiable. Hence \( \text{comp}(P) \) cannot be used to characterize which formulas can be derived from \( P \).

Stratified logic programs (1)

Still note that \( p(X) \leftarrow \neg p(X) \) is not unsatisfiable, but logically equivalent to \( p(X) \).

SLDNF-Resolution will not enable to derive from \( P = \{p(X) \leftarrow \neg p(X)\} \) that \( \forall X p(X) \) holds, not even that \( \exists X p(X) \) holds.

The notion of general logic program is in some sense too general. The notion of **stratified** logic programs represent an attempt to isolate a class of general logic programs that behave properly.
Stratified logic programs (2)

Consider the following example of general logic program.

\[ \text{founding}(X) \leftarrow \text{on}(Y, X), \text{on\_ground}(X). \]
\[ \text{on\_ground}(X) \leftarrow \neg \text{off\_ground}(X). \]
\[ \text{off\_ground}(X) \leftarrow \text{on}(X, Y). \]
\[ \text{on}(c, b). \]
\[ \text{on}(b, a). \]

This program is stratified. The lower stratum consists of the last three clauses, where \( \text{off\_ground} \) is fully defined. There is one stratum above, where \( \neg \text{off\_ground} \) is used.

A general program is stratified if it can be partitioned into strata or layers such that the negation of a relation is used in some stratum only if that relation is fully defined in the strata below.

Stratified logic programs (3)

The idea is the following.

- Write clauses involving \( p_1 \) \( \ldots \) \( p_n \).
- Write clauses involving
  \[ p_2, p_3, p_4, \ldots, p_n, \neg p_1, \ldots, \neg p_n. \]
- Write clauses involving
  \[ p_2, p_3, p_4, \ldots, p_n, \neg p_1, \ldots, \neg p_n, p_1, \ldots, \neg p_1, \ldots, \neg p_1. \]
- etc.

So a general logic program such as

\[ \leftarrow p(X) \leftarrow \neg q(X, Y) \]
\[ \leftarrow q(X, Y) \leftarrow \neg p(Y) \]

is not stratified: the two clauses have to belong to the same stratum, where both \( p \) is defined and \( \neg p \) is used.

Stratified logic programs (4)

More formally, given a predicate symbol \( q \) and a general logic program \( P \), denote by \( P^q \) the subset of \( P \) consisting of all clauses whose head is built from \( q \).

We say that \( P \) is stratified if there exists a partition \( \{P_1, \ldots, P_n\} \) of \( P \) such that:

- if \( p(\ldots) \leftarrow \ldots, q(\ldots), \ldots \in P_i \) then \( P^q \subseteq P_1 \cup \ldots \cup P_i \);
- if \( p(\ldots) \leftarrow \ldots, \neg q(\ldots), \ldots \in P_i \) then \( P^q \subseteq P_1 \cup \ldots \cup P_{i-1} \).

The completion of a stratified logic program is always satisfiable.

On the other hand, some logic programs have a satisfiable completion, despite the fact that they are not stratified. So the restriction to stratified programs eliminates good general logic programs.

Standard models

A stratified program \( P \) does not necessarily have a unique minimal Herbrand model. But among all the minimal Herbrand models of \( P \), one of them is more natural than the others. It is called the standard model of \( P \).

The standard model of \( P \) can be obtained via a generalization of the notion of immediate consequence operator and fixed point. For instance, the standard model of the general program \( P \) considered previously is:

\[ \text{on}(b, a), \text{on}(c, b), \text{off\_ground}(b), \text{off\_ground}(c), \]
\[ \text{on\_ground}(a), \text{founding}(a). \]

On the other hand, \( P \) has another minimal Herbrand model that represents an unintended interpretation, namely:

\[ \text{on}(b, a), \text{on}(c, b), \text{off\_ground}(b), \text{off\_ground}(c), \text{off\_ground}(a). \]