4. Logic Programs

4.1. Syntax + Semantics of Logic Programs

4.2. Universality of Logic Programming

4.3. Indeterminisms for Evaluation of Logic Programs

4.1. Syntax + Semantics of Logic Programs

In LP, the order of the literals in a clause and the order of the clauses in a program is important.

⇒ From now on:

\[\text{clause} = \text{sequence of literals } \quad \text{(instead of 'set')}\]
\[\text{clause set} = \text{sequence of clauses}\]

In particular, a clause can contain the same literal several times and a clause set can contain the same clause several times.

Def 4.1.1 (Syntax of Logic Programs)

A non-empty finite set \( S \) of definite Horn clauses over a signature \( (\Sigma, \Delta) \) is called a logic program over \( (\Sigma, \Delta) \).

We distinguish the following forms of program clauses:
We distinguish the following forms of program clauses:

- **facts** are clauses of the form \{B\} where B is an atomic formula
- **rules** are clauses of the form \{B, \neg C_1, \ldots, \neg C_n\} with \(n \geq 1\) and \(B, C_1, \ldots, C_n\) are atomic formulas.

To execute a logic program, we use a query \(G\) of the form \{\neg A_1, \ldots, \neg A_k\} with \(k \geq 1\).

As usual, a clause stands for the universally quantified disjunction of its literals.

If \(P\) is called with the query \(G = \{\neg A_1, \ldots, \neg A_k\}\), then one has to prove

\[ P \models \exists x_1, \ldots, x_p \quad A_1 \land \ldots \land A_k \]

all variables occurring in \(A_1, \ldots, A_k\)

or, equivalently, one has to prove unsatisfiability of \(P \cup \{G\}\).

Since \(P \cup \{G\}\) is a set of Horn clauses and \(G\) is the only negative clause, by the completeness of SLD-resolution we know that only one single ground instance of \(G\) is needed.
only one single ground instance of $G$ is needed to prove unsatisfiability:
\[ \{ G \} \text{ unsat.} \]

- the set of all ground instances of $G \cup \{ \top \}$ is unsat.
  (i.e., the Herbrand expansion of $G \cup \{ \top \}$ is unsat.)

- there is an SLD resolution proof of $\top$ that starts with a ground instance of $G$,
  i.e., with $(\neg A_1 \lor \ldots \lor \neg A_k)[X_1/t_1, \ldots, X_p/t_p]$ for some ground terms $t_1, \ldots, t_p$.

So if $\exists \top \exists X_1, \ldots, X_p \ A_1 \land \ldots \land A_k$, then there exist ground terms $t_1, \ldots, t_p$ where
\[ \exists \top (A_1 \land \ldots \land A_k)[X_1/t_1, \ldots, X_p/t_p] \]

The logic program should not only check whether $\exists \top \exists X_1, \ldots, X_p \ A_1 \land \ldots \land A_k$ but it should compute answer substitutions like \[ \{ X_1/t_1, \ldots, X_p/t_p \} \].

These answer subst. can be computed during the SLD resolution proof.

Ex 4.12 Consider the following LP:
Ex 4.12 Consider the following LP:

\[
\begin{align*}
\text{motherOf} & \ (\text{ren, sus}), \\
\text{married} & \ (\text{gerd, ren}).
\end{align*}
\]

\[
\text{fatherOf} \ (F, C) : - \text{married} (F, W), \text{motherOf}(W, C).
\]

This is an alternative notation for the following clause set:

\[
B = \{ \{ \text{motherOf} \ (\text{ren, sus}) \}, \\
\{ \text{married} \ (\text{gerd, ren}) \}, \\
\{ \text{fatherOf} \ (F, C), \neg \text{married} \ (F, W), \neg \text{motherOf} \ (W, C) \} \}
\]

Consider the query \( ? - \text{fatherOf} \ (\text{gerd, y}). \)

Thus, we have to extend \( B \) by \( G = \{ \neg \text{fatherOf} \ (\text{gerd, y}) \} \).

We now use binary SLD resolution:

\[
\begin{align*}
\{ \neg \text{fatherOf} \ (\text{gerd, y}) \} & \quad \{ \text{fatherOf} \ (F, C), \neg \text{married} \ (F, W), \neg \text{motherOf} \ (W, C) \} \\
\{ \text{F/gerd, } \text{Y/C} \} & \quad \{ \text{married} \ (\text{gerd, ren}) \} \\
\{ \text{W/ren} \} & \quad \{ \text{motherOf} \ (\text{ren, sus}) \} \\
\{ \text{C/sus} \} & \quad \{ \text{Y/C} \}
\end{align*}
\]

The answer subst. can be obtained from the SLD proof by composing the ungu’s that were
SLD proof by composing the ungu’s that were used in the proof:

\[
\begin{align*}
\{ C/sus \} &= \{ W/ven \} \circ \{ Y/C, F/gerd \} \\
&= \{ Y/sus, F/gerd, W/ven, C/sus \}
\end{align*}
\]

The answer substitution is the part of this substitution that concerns the variables in the original query (i.e., \( Y \) in our example). Thus: Answer subst is \( \{ Y/susanne \} \).

We now define the semantics of logic programs in three ways: declarative, procedural, fixpoint.

**Semantics**

4.1.1. **Declarative Semantics of LP**

Idea: Re-use the semantics of predicate logic.

The semantics of a program \( P \) and a query \( G \) should consist of all ground instances of \( G \) that are entailed by \( P \).

**Def 4.13 (Declarative Semantics of a LP)**

Let \( P \) be a logic program and \( G = \{ -A_1, \ldots, -A_n \} \) be a query. Then the declarative semantics of \( P \) is \( G \).
Let \( P \) be a logic program and \( Q \) a query. Then the declarative semantics of \( P \) with \( G \) is:

\[
D_\text{ES}, G \models \{ \varphi(A_1, \ldots, A_n) \mid G \models \varphi(A_1, \ldots, A_n), \varphi \text{ is a ground substitution} \}
\]

**Ex 4.1.4.** We again regard \( P \) and \( G \) from Ex. 4.1.2.

\[
D_\text{ES}, G \models \{ \text{fatherOf(gerd, susanne)} \}
\]

Because \( P \models \text{fatherOf(gerd, susanne)} \).

If \( P \) also contained the fact motherOf(rene, peter),
then we would have

\[
D_\text{ES}, G \models \{ \text{fatherOf(gerd, susanne)}, \text{fatherOf(gerd, peter)} \}
\]

### 4.1.2 Procedural Semantics of LP

**Idea of procedural/operational semantics:**

Define an interpreter for the prog. language
which determines how the program should behave.

For LP: define an interpreter which checks
entailment in prop. logic.

I use SLD resolution and keep track of
the applied unifs to obtain answers subst.

Our interpreter works on configurations \((G, \varnothing)\):

1. negative
2. substitute
• Start with \((G, \emptyset)\)
  
  \[
  \begin{align*}
  \text{original} & \quad \text{empty (identical)} \\
  \text{query} & \quad \text{Substitution}
  \end{align*}
  \]

• Perform resolution repeatedly until one reads
  
  \((\emptyset, \sigma)\).
  
Then \(\sigma\) (restricted to the variables in the original query) is the answer substitution that is computed by the program.

The form of SLD resolution differs from ordinary SLD resolution in 3 aspects:

1. **Standardized SLD resolution**
   
   Only rename variables in the (definite) program clauses, not in the negative parent clause
   
   \[
   \begin{align*}
   \{\neg q(x)\} & \quad \{p, \neg r(x)\} \\
   \downarrow \quad \sigma = \{x / x'\} & \quad \{\neg q(x), \neg r(x')\}
   \end{align*}
   \]

2. **Binary SLD resolution**
3. Clauses are regarded as sequences of literals

\[
\{ \neg p, \neg p \} \quad \{ p \} \\
\{ \neg p \} \quad \{ \neg p \} \\
\{ \} \\
\]

**Def 4.15 (Procedural Semantics of LP)**

Let \( P \) be a logic program.

- A **configuration** is a pair \( (G, \sigma) \) where \( G \) is a query or \( \square \) and \( \sigma \) is a substitution.

- We have a computation step \( (G_1, \sigma_1) \vdash_\sigma (G_2, \sigma_2) \) if
  - \( G_1 = \{ \neg A_1, \ldots, \neg A_k \} \), \( k \geq 1 \)
  - there exists an \( \kappa \in \mathbb{P} \) and a variable renaming \( \gamma \) such that \( \gamma(\kappa) = \{ B, \gamma(C_1), \ldots, \gamma(C_n) \} \) with \( n \geq 0 \) and
    - \( \gamma(\kappa) \) is variable-disjoint from \( G_1 \) and \( \text{RANGE}(\sigma_1) \)
    - there is an \( 1 \leq i \leq k \) such that \( A_i \) and \( B \) are unifiable with \( \gamma \) under \( \sigma_1 \)
  - \( G_2 = \sigma(\gamma(\kappa) \cup \{ \neg A_1, \ldots, \neg A_k, C_1, \ldots, C_n, A_{i+n}, \ldots, A_k \}) \)
  - \( \sigma_2 = \sigma \circ \sigma_1 \)

- A **computation** of \( P \) for the query \( G \) is a finite or infinite sequence
\((G, \emptyset) \vdash_\emptyset (G_1, \sigma_1) \vdash_\emptyset (G_2, \sigma_2) \vdash_\emptyset \ldots\).

- A computation that starts with \((G, \emptyset)\)
  (for \(G = \{\neg A_1, \ldots, \neg A_k\}\)) and ends with \((G, G)\)
is **successful** with the solution \(\sigma (A_1 \land \ldots \land A_n)\).
The answer substitution is \(\sigma\) restricted to the
variables of \(G\).

- Then the **procedural semantics** of \(\mathcal{P}\) w.r.t. \(G\)
is

\[
\mathcal{P}^\mathcal{P}_G, G \models \{ \sigma' (A_1 \land \ldots \land A_n) \mid (G, \emptyset) \vdash_\emptyset (G, G), \\
\sigma' (A_1 \land \ldots \land A_n) \text{ is a ground instance} \\
of \sigma (A_1 \land \ldots \land A_n) \}.
\]

**Ex. 4.1.6** Query \(G = \{\neg \text{fatherOf}(\text{gerd}, Y)\}\)

\((\{\neg \text{fatherOf}(\text{gerd}, Y)\}, \emptyset)\)

\(\vdash_\emptyset (\{\neg \text{married}(\text{gerd}, W), \neg \text{motherOf}(W, C)\}, \{Y/C, F/\text{gerd}\})\)

\(\vdash_\emptyset (\{\neg \text{motherOf}(\text{rene}, C)\}, \{W/\text{rene}, Y/C, F/\text{gerd}\})\)

\(\vdash_\emptyset (\emptyset, \{C/\text{susanne}\}) \circ \{W/\text{rene}, Y/C, F/\text{gerd}\})\)

\(\{C/\text{sus}, W/\text{ren}, Y/\text{sus}, F/\text{gerd}\}\)

**Answer Subst.**: \(\{Y/\text{susanne}\}\)
There are 2 indeterminisms in the computation:

1. One has to choose the next prog. clause \( \mathcal{K} \) for the next resolution step.
   
   Influences success and result of the computation.

2. One has to choose the literal \( A_n \) for the next resolution step.
   
   Influences termination and efficiency of the computation.

\[ B = \{ p(X, Z), \neg p(X, Y), \neg p(Y, Z) \} \quad \text{and} \quad \neg \neg \neg p(V, b) \]

\[ \{ p(U, V) \}, \quad \{ \neg p(a, b) \} \]
Finite failure (does not end in $\square$).

However, there are also successful comp. seques: we could do the same first steps as before, but then use the second instead of the first prog. clause:

$$(\underbrace{\{
eg p(V, B)\}}, \emptyset)$$

$$\vdash_{S} (\underbrace{\{\neg p(V, B)\}}, \{ V/a, Y/B \} \circ (X/V, Z/B))$$

$$\vdash (\emptyset, \{U/B\} \circ \ldots)$$

$$\{U/B, V/a, Y/B, X/a, Z/B\}$$

Answer Subst: $\{V/a\}$.

Thus: $p(a, b) \in P \models S, G \|$.  

Alternatively, we could also use the 2. prog. clause in the first step:

$$(\underbrace{\{p(V, B)\}}, \emptyset)$$

$$\vdash_{S} (\emptyset, \{U/B, V/B\})$$

Answer Subst: $\{V/B\}$

Thus: $p(b, b) \in P \models S, G \|$

This 4.1.8 (Equivalence of Declarative and Procedural Semantics)

Let $P$ be a LP and $G$ be a query. Then we have $D \models S, G \| = P \models S, G \|$.  


Proof: due to soundness and completeness of SLD resolution on Horn clauses (see course notes).