4. 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.

$\Rightarrow$ From now on:

clause = <u>sequence</u> of literals (instead of "set")

clause set = <u>sequence</u> of clauses

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

<u>Def 4.1.1</u> (Syntax of Logic Programs)

A non-empty finite set $\mathcal{S}$ of definite Horn clauses over a signature (Σ, Δ) is called a <u>logic program</u> over (Σ, Δ).

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 $\{B, \neg C_1, \dots, \neg C_n\}$ with $n \geq 1$ and $B, C_1, \dots, C_n$ are atomic formulas.

To execute a logic program, we use a

query G of the form $\{\neg A_1, \dots, \neg A_k\}$ with $k \geq 1$.

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

If $\mathcal{P}$ is called with the query $G = \{\neg A_1, \dots, \neg A_k\}$, then one has to prove

$$\mathcal{P} \models \exists \underbrace{X_1, \dots, X_p}_{\text{all variables occurring in } A_1, \dots, A_k} \; A_1 \wedge \dots \wedge A_k$$

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

Since $\mathcal{P} \cup \{G\}$ 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:

$$\mathcal{P} \cup \{G\} \quad \text{unsat.}$$

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

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

So if $\mathcal{P} \models \exists X_1, \ldots, X_p \; A_1 \wedge \ldots \wedge A_k$,
then there exist ground terms $t_1, \ldots, t_p$ where

$$\mathcal{P} \models (A_1 \wedge \ldots \wedge A_k)[X_1/t_1, \ldots, X_p/t_p]$$

The logic program should not only check whether $\mathcal{P} \models \exists X_1, \ldots, X_p \; A_1 \wedge \ldots \wedge A_k$,
but it should compute <u>answer substitutions</u>
like $\{X_1/t_1, \ldots, X_p/t_p\}$.
These answer subst. can be computed during the SLD resolution proof.

<u>Ex 4.12</u> Consider the following LP: Slide 20

Ex 4.12 Consider the following LP: Slide 20

motherOf(ren, sus).

married(gerd, ren).

fatherOf(F, C) :- married(F, W), motherOf(W, C).

This is an alternative notation for the following clause set:

$\mathcal{P} = \{\ \{\text{motherOf(ren, sus)}\},$

$\{\text{married(gerd, ren)}\},$

$\{\text{fatherOf}(F,C), \neg\text{married}(F,W), \neg\text{motherOf}(W,C)\}\ \}$

Consider the query ?- fatherOf(gerd, Y).

Thus, we have to extend $\mathcal{P}$ by $G = \{\neg\text{fatherOf(gerd}, Y)\}$.

We now use binary SLD resolution:

$\{\neg\underline{\text{fatherOf(gerd}, Y)}\}$ $\qquad$ $\{\underline{\text{fatherOf}(F,C)}, \neg\text{married}(F,W), \neg\text{mO}(W,C)\}$

$\{F/\text{gerd}, Y/C\}$

$\{\neg\underline{\text{married(gerd}, W)}, \neg\text{mO}(W,C)\}$ $\qquad$ $\{\underline{\text{married(gerd, ren)}}\}$

$\{W/\text{ren}\}$

$\{\neg\text{mO(ren}, C)\}$ $\qquad$ $\{\text{mO(ren, sus)}\}$

$\{C/\text{sus}\}$

$\Box$

The answer subst. can be obtained from the SLD proof by composing the mgu's that were

SLD proof by composing the mgu's that were used in the proof:

$$\underset{\text{third}}{\{C/sus\}} \circ \underset{\text{second}}{\{W/ren\}} \circ \underset{\text{first}}{\{Y/C, F/gerd\}}$$

$$= \{Y/sus, F/gerd, W/ren, C/sus\}$$

The <u>answer substitution</u> 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 (operational), fixpoint (denotational) semantics

4.1.1. Declarative Semantics of LP

Idea: Re-use the semantics of predicate logic
The semantics of a program $\mathcal{P}$ and a query G should consist of all ground instances of G that are entailed by $\mathcal{P}$.

<u>Def 4.13</u> (Declarative Semantics of a LP) Slide 21

Let $\mathcal{P}$ be a logic program and $G = \{\neg A_1, \dots, \neg A_n\}$ be a query. Then the declarative semantics of $\mathcal{P}$ and G:

Let $\mathcal{P}$ be a logic program and $G = \{\neg A_1, \dots, \neg A_n\}$ be a query. Then the <u>declarative semantics</u> of $\mathcal{P}$ w.r.t. G is

$$\mathcal{D}[\![\mathcal{P}, G]\!] = \{\sigma(A_1 \wedge \dots \wedge A_n) \mid \mathcal{P} \models \sigma(A_1 \wedge \dots \wedge A_n), \sigma \text{ is a ground substitution}\}$$

Ex 4.1.4. We again regard $\mathcal{P}$ and G from Ex. 4.1.2.

$\mathcal{D}[\![\mathcal{P}, G]\!] = \{$ fatherOf(gerd, susanne) $\}$.

because $\mathcal{P} \models$ fatherOf(gerd, susanne)

If $\mathcal{P}$ also contained the fact motherOf(renate, peter), then we would have

$\mathcal{D}[\![\mathcal{P}, G]\!] = \{$ fatherOf(gerd, susanne), 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 pred logic

$\leadsto$ use SLD resolution and keep track of the applied mgu's to obtain answer subst.

Our interpreter works on <u>configurations</u> (G, σ)

negative ... substinegative Horn clause

Substi-tution

- start with $(G, \emptyset)$

original query

empty (identical) substitution

- perform resolution repeatedly until one reaches

$(\square, \sigma)$.

empty clause

Then σ (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. <u>standardized SLD resolution</u>

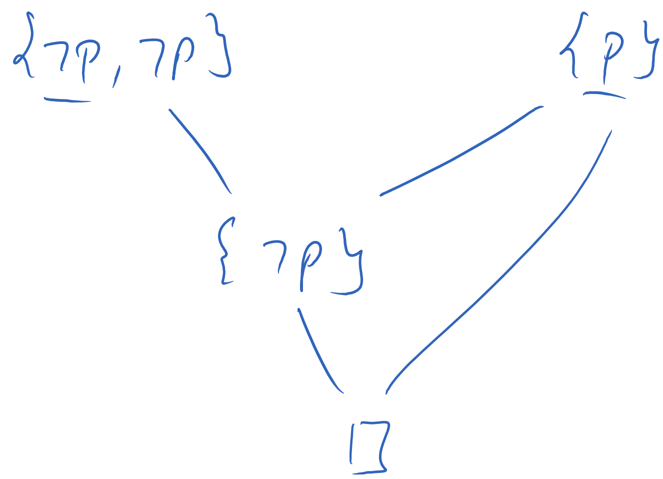
   only rename variables in the (definite) program clauses, not in the negative parent clause

$\{\neg p, \neg q(X)\}$ $\qquad$ $\{p, \neg r(X)\}$ $\quad \nu = \{X / X'\}$

$\{\neg q(X), \neg r(X')\}$

2. <u>binary SLD resolution</u>

3. clauses are regarded as sequences of literals



Def 4.15 (Procedural Semantics of LP)

Slide 21

Let $\mathcal{P}$ be a logic program.

- A configuration is a pair (G, σ) where G is a query or $\Box$ and σ is a substitution.
- We have a computation step $(G_1, \sigma_1) \vdash_{\mathcal{P}} (G_2, \sigma_2)$ iff
  - $G_1 = \{\neg A_1, \dots, \neg A_k\}$, $k \geq 1$
  - there exists a $K \in \mathcal{P}$ and a variable renaming ν such that $\nu(K) = \{B, \neg C_1, \dots, \neg C_n\}$ with $n \geq 0$ and
    - * $\nu(K)$ is variable-disjoint from G_1 and $RANGE(\sigma_1)$
    - * there is an $1 \leq i \leq k$ such that A_i and B are unifiable with mgu σ
  - $G_2 = \sigma(\{\neg A_1, \dots, \neg A_{i-1}, \neg C_1, \dots, \neg C_n, \neg A_{i+1}, \dots, \neg A_k\})$
  - $\sigma_2 = \sigma \circ \sigma_1$
- A computation of $\mathcal{P}$ for the query G is a finite or infinite sequence

$(G, \emptyset) \vdash_{\mathcal{P}} (G_1, \sigma_1) \vdash_{\mathcal{P}} (G_2, \sigma_2) \vdash_{\mathcal{P}} \dots$

- A computation that starts with $(G, \emptyset)$ (for $G = \{\neg A_1, \dots, \neg A_k\}$) and ends with $(\Box, \sigma)$ is successful with the solution $\sigma(A_1 \wedge \dots \wedge A_k)$.
  The answer substitution is σ restricted to the variables of G.
- Then the procedural semantics of $\mathcal{P}$ w.r.t. G is

transitive closure of $\vdash_{\mathcal{P}}$

$$P[\![\mathcal{P}, G]\!] = \{\sigma'(A_1 \wedge \dots \wedge A_k) \mid (G, \emptyset) \vdash_{\mathcal{P}}^{+} (\Box, \sigma),\ \sigma'(A_1 \wedge \dots \wedge A_k) \text{ is a ground instance of } \sigma(A_1 \wedge \dots \wedge A_k)\}.$$

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

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

$\vdash_{\mathcal{P}} (\{\neg married(gerd, W), \neg motherOf(W, C)\}, \{Y/C, F/gerd\})$

$\vdash_{\mathcal{P}} (\{\neg motherOf(renate, C)\}, \{W/renate, Y/C, F/gerd\})$

$\vdash_{\mathcal{P}} (\Box, \underbrace{\{C/susanne\} \circ \{W/renate, Y/C, F/gerd\}}_{\{C/sus,\ W/ren,\ Y/sus,\ F/gerd\}})$

Answer Subst: $\{Y/susanne\}$

$P⟦\mathcal{P}, G⟧ = \{\neg fatherOf(gerd, susanne)\}$

There are 2 indeterminisms in the computation:

1. One has to choose the next prog. clause K for the next resolution step.
   Influences success and result of the computation.

2. One has to choose the literal A_i for the next resolution step.
   Influences termination and efficiency of the computation.

Ex 4.17: There are different comp. sequences for the same query.

$\mathcal{P} = \{ \{p(X,Z), \neg q(X,Y), \neg p(Y,Z)\},$
$\{p(U,U)\},$
$\{q(a,b)\} \}$

$G = \{\neg p(V,b)\}$

$(\{\underline{\neg p(V,b)}\}, \varnothing)$

$\vdash_{\mathcal{P}} (\{\underline{\neg q(V,Y)}, \neg p(Y,b)\}, \{X/V, Z/b\})$

$\vdash_{\mathcal{P}} (\{\underline{\neg p(b,b)}\}, \{V/a, Y/b\} \circ \{X/V, Z/b\})$

$\vdash_{\mathcal{P}} (\{\neg q(b,Y'), \underline{\neg p(Y',b)}\}, \{X'/b, Z'/b\} \circ \dots)$

$\vdash_{\mathcal{P}} (\{\neg q(b,b)\}, \{U/b, Y'/b\} \circ \dots)$

This computation sequence cannot be continued:

finite failure (does not end in $\Box$).

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

$$(\{\neg \underline{p(V,b)}\}, \emptyset)$$

$$\vdash_{\mathcal{P}}^{+} (\{\neg \underline{p(b,b)}\}, \{V/a, Y/b\} \circ \{X/V, Z/b\})$$

$$\vdash (\Box, \underbrace{\{U/b\} \circ \text{——} \;\text{"}\; \text{————}}_{\{U/b,\, V/a,\, Y/b,\, X/a,\, Z/b\}})$$

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

Thus: $p(a,b) \in P[\![\mathcal{P}, G]\!]$.

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

$$(\{\neg p(V,b)\}, \emptyset)$$

$$\vdash_{\mathcal{P}} (\Box, \{U/b, V/b\}).$$

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

Thus: $p(b,b) \in P[\![\mathcal{P}, G]\!]$

Thm 4.1.8 (Equivalence of Declarative and Procedural Semantics)

Let $\mathcal{P}$ be a LP and G be a query.

Then we have $D[\![\mathcal{P}, G]\!] = P[\![\mathcal{P}, G]\!]$.

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