Definitions

Foundations of Logic Programming

(compiled by Ilina Stoilkovska and Isabel Juarez Castro)

1 Substitutions

1. A *substitution* is a finite mapping from variables to terms[1] which assigns to each variable x in its domain a term t different from x. We write it as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where

   - $x_1, \ldots, x_n$ are different variables,
   - $t_1, \ldots, t_n$ are terms,
   - for $i \in [1, n], x_i \neq t_i$.

   A pair $x_i \mapsto t_i$ is called a *binding*. When $n = 0$, the resulting substitution is called *empty substitution* and is denoted by ϵ.

2. Consider a substitution $\theta = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. If all $t_1, \ldots, t_n$ are ground, then θ is called *ground*, and if all $t_1, \ldots, t_n$ are variables, then θ is called a *pure variable substitution*. If θ is a 1-1 and onto mapping from its domain to itself, then θ is called a *renaming*.

3. We denote by $Dom(\theta)$ the set of variables $\{x_1, \ldots, x_n\}$, by $Ran(\theta)$ the set of terms $\{t_1, \ldots, t_n\}$, and by $VRan(\theta)$ the set of variables appearing in $t_1, \ldots, t_n$. Then we define $Var(\theta) = Dom(\theta) \cup VRan(\theta)$

4. The term $s\theta$ is called an *instance* of s. An instance is called *ground* if it contains no variables. If θ is a renaming, then $s\theta$ is called a *variant* of s.

5. The *composition* of substitutions θ and η, written as $\theta\eta$, is defined as follows. We put for a variable x

$$x(\theta\eta) := (x\theta)\eta$$

   .

[1] A substitution is then identified with the extension of such mapping to the mapping from terms to terms.

2 Unifiers

1. Let θ and τ be substitutions. We say that θ is *more general* than τ if for some substitution η we have $\tau = \theta\eta$.

2. - θ is called a *unifier* of s and t if $s\theta = t\theta$. If a unifier of s and t exists, we say that s and t are *unifiable*.
   - θ is called a *most general unifier* (*mgu* in short) of s and t if it is a unifier of s and t that is more general than all unifiers of s and t.
   - An mgu θ of s and t is called *strong* if for all unifiers η of s and t we have $\eta = \theta\eta$.

3. A substitution θ is called idempotent if $\theta\theta = \theta$.

4. θ is a strong mgu iff for every unifier η we have $\eta = \theta\eta$

5. A unifier θ of s and t is called relevant if $Var(\theta) \subseteq Var(s) \cup Var(t)$.

3 Queries and Programs

1. We define atoms, queries, clauses, programs and resultants as follows:

   - if p is an n-ary relation symbol and $t_1, \ldots, t_n$ are terms then $p(t_1, \ldots, t_n)$ is an *atom*,
   - a *query* is a finite sequence of atoms,
   - a *clause* is a construct of the form $H \leftarrow \mathbf{B}$, where H is an atom and $\mathbf{B}$ is a query; H is called its head and $\mathbf{B}$ its body,
   - a *program* is a finite set of clauses.
   - a *resultant* is a construct of the form $\mathbf{A} \leftarrow \mathbf{B}$, where $\mathbf{A}$ and $\mathbf{B}$ are queries.

   The empty query is denoted by $\Box$. When $\mathbf{B}$ is empty, $H \leftarrow \mathbf{B}$ is written $H \leftarrow$ and is called a *unit clause*.

4 SLD derivations

1. Consider a non-empty query $\mathbf{A}, B, \mathbf{C}$ and a clause c. Let $H \leftarrow \mathbf{B}$ be a variant of c variable disjoint with $\mathbf{A}, B, \mathbf{C}$. Suppose that B and H unify. Let θ be an mgu of B and H.
   Then $(\mathbf{A}, \mathbf{B}, \mathbf{C})\theta$ is called an *SLD-resolvent* of $\mathbf{A}, B, \mathbf{C}$ and c *w.r.t.* B, *with an mgu* θ. B is called the *selected atom* of $\mathbf{A}, B, \mathbf{C}$. We write then

   $$\mathbf{A}, B, \mathbf{C} \underset{c}{\overset{\theta}{\Longrightarrow}} (\mathbf{A}, \mathbf{B}, \mathbf{C})\theta$$

   and call it an *SLD-derivation step*. $H \leftarrow \mathbf{B}$ is called its *input clause*. If the clause c is irrelevant we drop a reference to it.

2. A maximal sequence $Q_0 \underset{c_1}{\overset{\theta_1}{\Longrightarrow}} Q_1 \ldots Q_n \underset{c_{n+1}}{\overset{\theta_{n+1}}{\Longrightarrow}} Q_{n+1} \ldots$ of SLD-derivation steps is called an *SLD-derivation* of $P \cup \{Q_0\}$ if

   - $Q_0, \ldots, Q_{n+1}, \ldots$ are queries, each empty or with one atom selected in it,
   - $\theta_1, \ldots, \theta_{n+1}, \ldots$ are substitutions,
   - $c_1, \ldots, c_{n+1}, \ldots$ are clauses of P,

   and for every step the following condition holds:

   - **Standardization apart**: the input clause employed is variable disjoint from the initial query Q_0 and from the substitutions and the input clauses used at earlier steps. More formally:

$$Var(c'_i) \cap (Var(Q_0) \cup \bigcup_{j=1}^{i-1} (Var(\theta_j) \cup Var(c'_j))) = \emptyset$$

     for $i \geq 1$, where c'_i is the input clause used in the step $Q_{i-1} \underset{c_i}{\overset{\theta_i}{\Longrightarrow}} Q_i$.

   If the program is clear from the context, we speak of an SLD-derivation of Q_0 and if the clauses $c_1, \ldots, c_{n+1}, \ldots$ are irrelevant we drop reference to them.

3. - A clause is called *applicable* to an atom if a variant of its head unifies with the atom.
   - The *length* of an SLD-derivation is the number of SLD-derivation steps used in it. So an SLD-derivation of length 0 consists of a single query Q such that either Q is empty or no clause of the program is applicable to its selected atom.

4. Consider a finite SLD-derivation $\xi := Q_0 \overset{\theta_1}{\Longrightarrow} Q_1 \ldots \overset{\theta_n}{\Longrightarrow} Q_n$ of a query $Q := Q_0$

   - ξ is called *successful* if $Q_n = \square$. The restriction $(\theta_1 \ldots \theta_n) \,|\, Var(Q)$ of the composition $\theta_1 \ldots \theta_n$ to the variables of Q is called a *computed answer substitution* (*c.a.s.* in short) of Q and $Q\theta_1 \ldots \theta_n$ is called a *computed instance* of Q.
   - ξ is called *failed* if Q_n is non-empty and no clause of P is applicable to the selected atom of Q_n.

5 Resultants

1. - Given an SLD-derivation step $Q \overset{\theta}{\Longrightarrow} Q_1$ we call $Q\theta \leftarrow Q_1$ the *resultant associated with* it.

- Consider a resultant $Q \leftarrow \mathbf{A}, B, \mathbf{C}$ and a clause c. Let $H \leftarrow \mathbf{B}$ be a variant of c variable disjoint with $Q \leftarrow \mathbf{A}, B, \mathbf{C}$ and θ an mgu of B and H. Then $(Q \leftarrow \mathbf{A, B, C})\theta$ is called an *SLD-resolvent* of $Q \leftarrow \mathbf{A}, B, \mathbf{C}$ and c *w.r.t.* B*, with an mgu* θ. B is called the *selected atom* of $Q \leftarrow \mathbf{A}, B, \mathbf{C}$.

  We write then $(Q \leftarrow \mathbf{A}, B, \mathbf{C}) \underset{c}{\overset{\theta}{\Longrightarrow}} (Q \leftarrow \mathbf{A, B, C})\theta$ and call it an *SLD-resultant step.* $H \leftarrow \mathbf{B}$ is called its *input clause.* If the clause c is irrelevant we drop a reference to it.

2. Consider an SLD-derivation

$$Q_0 \underset{c_1}{\overset{\theta_1}{\Longrightarrow}} Q_1 \dots Q_n \underset{c_{n+1}}{\overset{\theta_{n+1}}{\Longrightarrow}} Q_{n+1} \dots$$

   Let for $i \geq 0$

$$R_i := Q_0\theta_1 \dots \theta_i \leftarrow Q_i$$

   We call R_i the *resultant of level* i of the SLD-derivation $Q_0 \underset{c_1}{\overset{\theta_1}{\Longrightarrow}} Q_1 \dots Q_n \underset{c_{n+1}}{\overset{\theta_{n+1}}{\Longrightarrow}} Q_{n+1} \dots$

6 Properties of SLD-derivations

1. Consider an SLD-derivation

$$\xi := Q_0 \underset{c_1}{\overset{\theta_1}{\Longrightarrow}} Q_1 \dots Q_n \underset{c_{n+1}}{\overset{\theta_{n+1}}{\Longrightarrow}} Q_{n+1} \dots$$

   We say that the SLD-derivation

$$\xi' := Q'_0 \underset{c_1}{\overset{\theta'_1}{\Longrightarrow}} Q'_1 \dots Q'_n \underset{c_{n+1}}{\overset{\theta'_{n+1}}{\Longrightarrow}} Q'_{n+1} \dots$$

   is a *lift* of ξ if

   - ξ is of the same or smaller length than ξ',
   - Q_0 is an instance of Q'_0,
   - for $i \geq 0$, in Q_i and Q'_i, atoms in the same positions are selected.

2. Consider two SLD-derivations:

$$\xi := Q_0 \underset{c_1}{\overset{\theta_1}{\Longrightarrow}} Q_1 \dots Q_n \underset{c_{n+1}}{\overset{\theta_{n+1}}{\Longrightarrow}} Q_{n+1} \dots$$

   and

$$\xi' := Q'_0 \underset{c_1}{\overset{\theta'_1}{\Longrightarrow}} Q'_1 \dots Q'_n \underset{c_{n+1}}{\overset{\theta'_{n+1}}{\Longrightarrow}} Q'_{n+1} \dots$$

   We say that ξ and ξ' are *similar* if

   - ξ and ξ' are of the same length,
   - Q_0 and Q'_0 are variants of each other,
   - for $i \geq 0$, in Q_i and Q'_i, atoms in the same positions are selected.

7 Selection Rules

1. - Let *INIT* stand for the set of initial fragments of SLD-derivations in which the last query is non-empty. By a *selection rule* $\mathcal{R}$ we mean a function which, when applied to an element of *INIT* yields an occurrence of an atom in its last query.
   - Given a selection rule $\mathcal{R}$, we say that an SLD-derivation ξ is via $\mathcal{R}$ if all choices of the selected atoms in ξ are performed according to $\mathcal{R}$. That is, for each initial fragment $\xi^{<}$ of ξ ending with a non-empty query Q, $\mathcal{R}(\xi^{<})$ is the selected atom of Q.

8 SLD-trees

1. An *SLD-tree for* $P \cup \{Q\}$ *via* a selection rule $\mathcal{R}$ is a tree such that
   - its branches are SLD-derivations of $P \cup \{Q\}$ via $\mathcal{R}$,
   - every node Q with selected atom A has exactly one descendant for every clause c from P which is applicable to A. This descendant is a resolvent of Q and c w.r.t. A.
2. - We call an SLD-tree *successful* if it contains the empty query.
   - We call an SLD-tree *finitely failed* if it is finite and not successful.
3. We call a selection rule $\mathcal{R}$ *variant independent* if in all initial fragments of SLD-derivations which are similar, $\mathcal{R}$ chooses the atom in the same position in the last query.

9 Algebras

1. An *algebra* (sometimes called a *preinterpretation*) J for a language of terms $\mathcal{L}$ consists of:
   - a non-empty set D, called the *domain* of J,
   - an assignment to each n-ary function symbol f in $\mathcal{L}$ of a mapping f_J from D^n to D.
2. A *valuation* or *state over the domain* D is a mapping assigning each variable an element from D. Given now a state σ over D, we extend its domain to all terms, that is we assign a term t an element $\sigma(t)$ from D, proceeding by induction as follows:
   - $\sigma(f(t_1, \ldots, t_n)) = f_J(\sigma(t_1), \ldots, \sigma(t_n))$.

   So $\sigma(f(t_1, \ldots, t_n))$ is the result of applying the mapping f_J to the sequence of values (already) associated by σ with the terms $t_1, \ldots, t_n$. Observe that for a constant c, we have $\sigma(c) = c_J$, so $\sigma(c)$ does not depend on σ.

10 Interpretations

1. An *interpretation* I for a language $\mathcal{L}$ of programs consists of:
   - an algebra J with domain D
   - an assignment, to each n-ary relation symbol p in $\mathcal{L}$, of a subset p_I, of D^n.
2. We now define a relation $I \models_\sigma E$ between an interpretation I for $\mathcal{L}$, a state σ over the domain of I and an expression E. Intuitively, $I \models_\sigma E$ means that E is true when its variables are interpreted according to σ.
   - If $p(t_1, \ldots, t_n)$ is an atom, then
     $I \models_\sigma p(t_1, \ldots, t_n)$ iff $(\sigma(t_1), \ldots, \sigma(t_n)) \in p_I$,
   - if $A_1, \ldots, A_n$ is a query, then
     $I \models_\sigma A_1, \ldots, A_n$ iff $I \models_\sigma A_i$ for $i \in [1, n]$,
   - if $\mathbf{A} \leftarrow \mathbf{B}$ is a resultant, then
     $I \models_\sigma \mathbf{A} \leftarrow \mathbf{B}$ iff $I \models_\sigma \mathbf{A}$ under the assumption of $I \models_\sigma \mathbf{B}$.

   In particular, if $H \leftarrow \mathbf{B}$ is a clause, then
   $I \models_\sigma H \leftarrow \mathbf{B}$ iff $I \models_\sigma H$ under the assumption of $I \models_\sigma \mathbf{B}$,
   and for a unit clause $H \leftarrow$
   $I \models_\sigma H \leftarrow$ iff $I \models_\sigma H$.
   Finally, we say that an expression E *is true in the interpretation* I and write $I \models E$, when for all states σ we have $I \models_\sigma E$.

11 Term Interpretations

1. *Term Universe* $TU_\mathcal{L}$ for the language of programs $\mathcal{L}$ is the set of all terms of $\mathcal{L}$.
2. *Term base* $TB_\mathcal{L}$ is the set of atoms of L.
3. *Term algebra* for $\mathcal{L}$ is defined as:
   - $TU_\mathcal{L}$ is the domain
   - If f is n-ary function symbol in $\mathcal{L}$ then we assign to f a mapping $(TU_\mathcal{L})^n \rightarrow TU_\mathcal{L}$ which maps sequences $(t_1, \ldots, t_n)$ to $f(t_1, \ldots, t_n)$
4. A *term interpretation* I for $\mathcal{L}$ is an interpretation based on the term algebra for $\mathcal{L}$
5. A term interpretation I is a *term model* of a set of expressions S if I is a model of S.
6. A term interpretation I is *closed under substitution* if $A \in I$ implies $inst(A) \subseteq I$. For such I we have:

- $I = \{A \mid A \text{ is an atom and } I \models A\}$

7. A finite tree whose nodes are atoms, is called an *implication tree w.r.t.* P if for each of its nodes A with the direct descendants $B_1, \ldots, B_n$, the clause $A \leftarrow B_1, \ldots, B_n$ is in $inst(P)$. In particular, for each leaf A the clause $A \leftarrow$ is in $inst(P)$. We say that an atom *has an implication tree w.r.t.* P if it is the root of an implication tree w.r.t. P. An implication tree is called *ground* iff all its nodes are ground.

12 Completeness of the SLD-resolution

1. Given a program P and a query Q, we say that Q is *n-deep* if every atom in Q has an implication tree w.r.t. P and the total number of nodes in these implication trees is n.

13 Least Term Models

1. A term model of a set of expressions S is called the *least term model* of S if it is included in every term model of S.

14 Herbrand Interpretations

1. *Herbrand Interpretation*
   - $HU_{\mathcal{L}}$ set of all ground terms.
   - $HB_{\mathcal{L}}$ set of all ground atoms.
2. The *Herbrand algebra* J for $\mathcal{L}$ is defined as follows:
   - its domain is the $HU_{\mathcal{L}}$,
   - if f is an n-ary function symbol, then $f_J : (HU_{\mathcal{L}})^n \to HU_{\mathcal{L}}$
3. A *Herbrand interpretation* I: every p (relation symbol) is assigned a set of ground terms.

   $I := \{p(t_1, \ldots, t_n) \mid p \text{ is a } n\text{-ary relation symbol and } (t_1, \ldots, t_n) \in p_I\}$