POPL2 class (2020-04-23)

SLD Resolution

Linear resolution for Definite clauses with Selection function

(using logic programming notation):

$$\frac{\leftarrow A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_m \qquad B_0 \leftarrow B_1, \ldots, B_n}{\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\theta}$$

$$\frac{\forall \neg (A_1 \wedge \cdots \wedge A_{i-1} \wedge A_i \wedge A_{i+1} \wedge \cdots \wedge A_m) \qquad \forall (B_0 \leftarrow B_1 \wedge \cdots \wedge B_n)}{\forall \neg (A_1 \wedge \cdots \wedge A_{i-1} \wedge B_1 \wedge \cdots \wedge B_n \wedge A_{i+1} \wedge \cdots \wedge A_m)\theta}$$

The goal clause may include several atomic formulas which unify with the head of some clause in the program

There exists a function which for a given goal selects the subgoal for unication. The function is called the *selection function* or the *computation rule*.

SLD-Resolution principle

goal clause G_0 $\qquad \leftarrow A_1, \ldots, A_m \qquad (m \geq 0)$

subgoal A_i is selected (if possible) by the computation rule.

$B_0 \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ whose head unifies with A_i

$$G_1 \qquad \leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\theta_1$$

SLD-Resolution

There are two cases when it is not possible to obtain G_{i+1} from G_i:

- the first is when the selected subgoal cannot be resolved (i.e. is not unifiable) with the head of any program clause;
- the other case appears when $G_i = \square$ (i.e. the empty goal).

A goal *Gi*+1 is said to be *derived* (*directly*) from *Gi* and *Ci* via *R* (or alternatively, *Gi* and *Ci* *resolve* into *Gi*+1).

Definition 3.12 (SLD-derivation) Let G_0 be a definite goal, P a definite program and $\Re$ a computation rule. An *SLD-derivation* of G_0 (using P and $\Re$) is a finite or infinite sequence of goals:

$$G_0 \overset{C_0}{\rightsquigarrow} G_1 \cdots G_{n-1} \overset{C_{n-1}}{\rightsquigarrow} G_n \ldots$$

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

SLD-Resolution (Example)

$$proud(X) \leftarrow parent(X, Y), newborn(Y).$$
$$parent(X, Y) \leftarrow father(X, Y).$$
$$parent(X, Y) \leftarrow mother(X, Y).$$
$$father(adam, mary).$$
$$newborn(mary).$$

$$G_0 \quad : \quad \leftarrow proud(Z).$$
$$C_0 \quad : \quad proud(X_0) \leftarrow parent(X_0, Y_0), newborn(Y_0).$$

$$G_1 \quad : \quad \leftarrow parent(Z, Y_0), newborn(Y_0).$$
$$C_1 \quad : \quad parent(X_1, Y_1) \leftarrow father(X_1, Y_1).$$

$$G_2 \quad : \quad \leftarrow father(Z, Y_0), newborn(Y_0).$$
$$C_2 \quad : \quad father(adam, mary).$$

$$G_3 \quad : \quad \leftarrow newborn(mary).$$
$$C_3 \quad : \quad newborn(mary).$$

$$G_4 \quad : \quad \square$$

SLD Resolution

$$
\begin{array}{ll}
1: & grandfather(X,Z) \leftarrow father(X,Y), parent(Y,Z). \\
2: & parent(X,Y) \leftarrow father(X,Y). \\
3: & parent(X,Y) \leftarrow mother(X,Y). \\
4: & father(a,b). \\
5: & mother(b,c).
\end{array}
$$

$\leftarrow grandfather(a, X).$

$grandfather(X_0, Z_0) \leftarrow father(X_0, Y_0), parent(Y_0, Z_0).$

$\leftarrow father(a, Y_0), parent(Y_0, X).$

$father(a, b).$

$\leftarrow parent(b, X).$

$parent(X_2, Y_2) \leftarrow mother(X_2, Y_2).$

$\leftarrow mother(b, X).$

$mother(b, c).$

$\square$

SLD Refutation

Definition 3.15 (SLD-refutation) A (finite) SLD-derivation:

$$G_0 \overset{C_0}{\leadsto} G_1 \cdots G_n \overset{C_n}{\leadsto} G_{n+1}$$

where $G_{n+1} = \Box$ is called an *SLD-refutation* of G_0.

SLD-derivations that end in the empty goal are of special importance since they correspond to refutations of (and provide answers to) the initial goal:

$$G_0 \overset{C_0}{\leadsto} G_1 \cdots G_n \overset{C_n}{\leadsto} \Box$$

Failed Derivation

Definition 3.17 (Failed derivation) A derivation of a goal clause G_0 whose last element is not empty and cannot be resolved with any clause of the program is called a *failed* derivation. $\blacksquare$

$$\leftarrow grandfather(a, X).$$

$$grandfather(X_0, Z_0) \leftarrow father(X_0, Y_0), parent(Y_0, Z_0).$$

$$\leftarrow father(a, Y_0), parent(Y_0, X).$$

$$father(a, b).$$

$$\leftarrow parent(b, X).$$

$$parent(X_2, Y_2) \leftarrow father(X_2, Y_2).$$

$$\leftarrow father(b, X).$$

complete derivation mean a *refutation*, a *failed derivation* or an *infinite derivation*.

SLD-tree

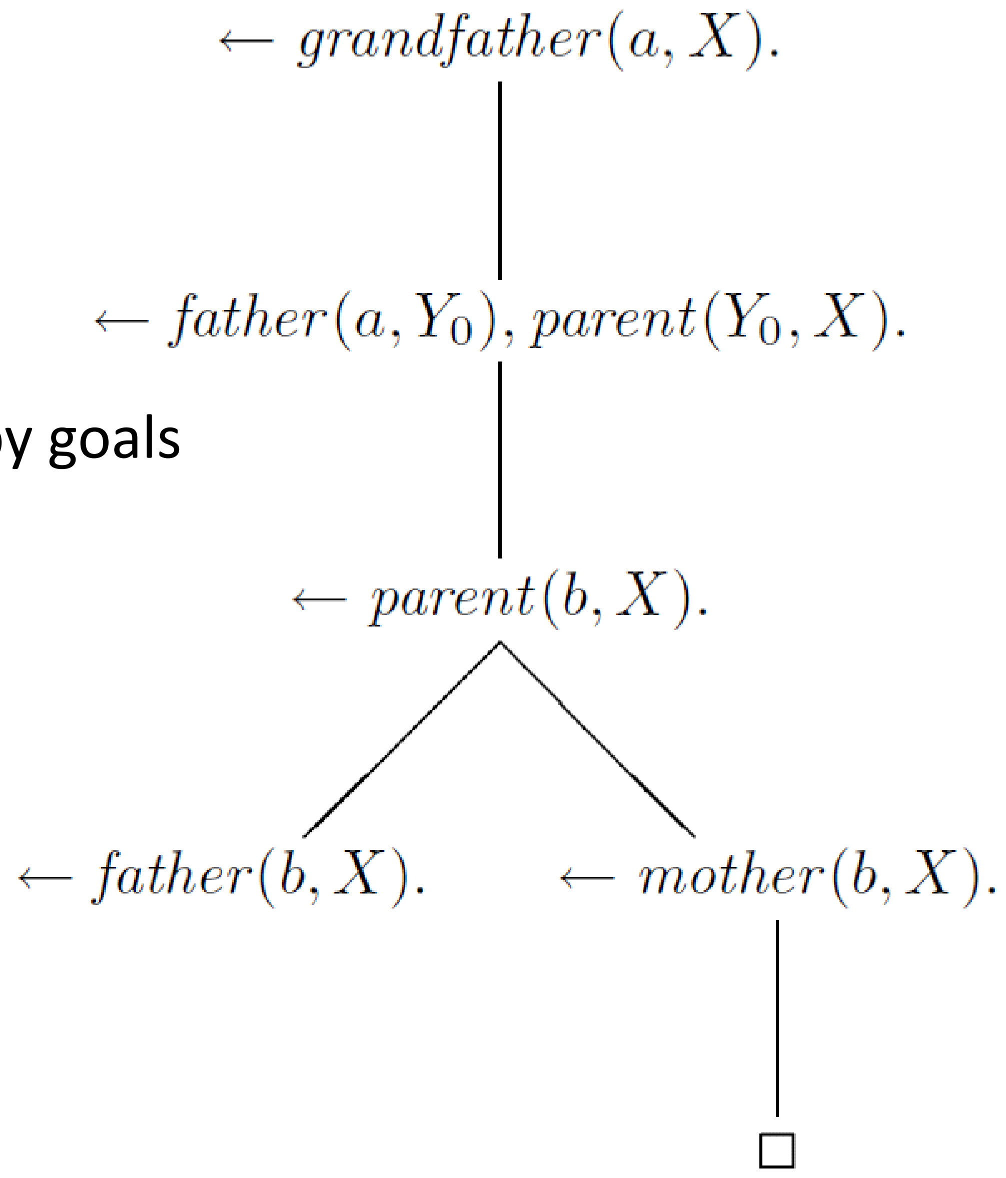
Definition 3.18 (SLD-tree) Let P be a definite program, G_0 a definite goal and $\Re$ a computation rule. The SLD-tree of G_0 (using P and $\Re$) is a (possibly infinite) labelled tree satisfying the following conditions:

- the root of the tree is labelled by G_0;
- if the tree contains a node labelled by 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 $\Re$ then the node labelled by G_i has a child labelled by G_{i+1}. The edge connecting them is labelled by C_i.

SLD-tree

The nodes of an SLD-tree are thus labelled by goals of a derivation. The edges are
labelled by the clauses of the program.

one-to-one correspondence
between the paths of the SLD-tree and the complete derivations of *G*0 under a fixed computation rule *R*



Negation in Logic Programming

- Denite programs express positive knowledge
- in real life the negative information is seldom stated explicitly.
- 'c on top of b'
- No Negative information
- 'b not on top of c'

$$
\begin{aligned}
&above(X, Y) \leftarrow on(X, Y).\\
&above(X, Y) \leftarrow on(X, Z), above(Z, Y).\\
&on(c, b).\\
&on(b, a).
\end{aligned}
$$

- lack of information is taken as evidence to the contrary
- -*A* provided that *A* is a ground atomic formula which cannot be derived by the inference rules

closed world assumption (*cwa*)

$$\frac{P \nvdash A}{\neg A} \quad (cwa)$$

Negation

- non-provability for definite programs is undecidable
- *-A* is derivable from *P* if the goal *A* has a f*initely failed SLD-tree w.r.t. P*

$$\frac{\leftarrow A \text{ has a finitely failed SLD-tree}}{\neg A} \quad (naf)$$

negation as (finite) *failure* rule (*naf*).

$$above(X,Y) \leftarrow on(X,Y).$$
$$above(X,Y) \leftarrow on(X,Z), above(Z,Y).$$
$$on(c,b).$$
$$on(b,a).$$

$$above(X,Y) \leftarrow above(X,Y).$$

The SLD-tree of the goal *above*(*b; c*) still contains no refutations but the tree is now infinite. Thus, it cannot be concluded that -*above*(*b; c*) using *naf* . However, it still follows from the *cwa*

SLDNF-resolution

- By combining SLDresolution with negation as finite failure it is possible to generalize the notion of goal to include both positive and negative literals.

Definition 4.6 (General goal) A *general goal* is a goal of the form:

$$\leftarrow L_1, \ldots, L_n. \qquad (n \geq 0)$$

where each L_i is a positive or negative literal.

The combination of SLD-resolution, to resolve positive literals, and negation as (finite) failure, to resolve negative literals, is called *SLDNF-resolution*:

SLDNF Resolution

Definition 4.7 (SLDNF-resolution for definite programs) Let P be a definite program, G_0 a general goal and $\Re$ a computation rule. An *SLDNF-derivation* of G_0 (using P and $\Re$) is a finite or infinite sequence of general goals:

$$G_0 \overset{C_0}{\rightsquigarrow} G_1 \cdots G_{n-1} \overset{C_{n-1}}{\rightsquigarrow} G_n \cdots$$

where $G_i \overset{C_i}{\rightsquigarrow} G_{i+1}$ if either:

(i) the $\Re$-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;

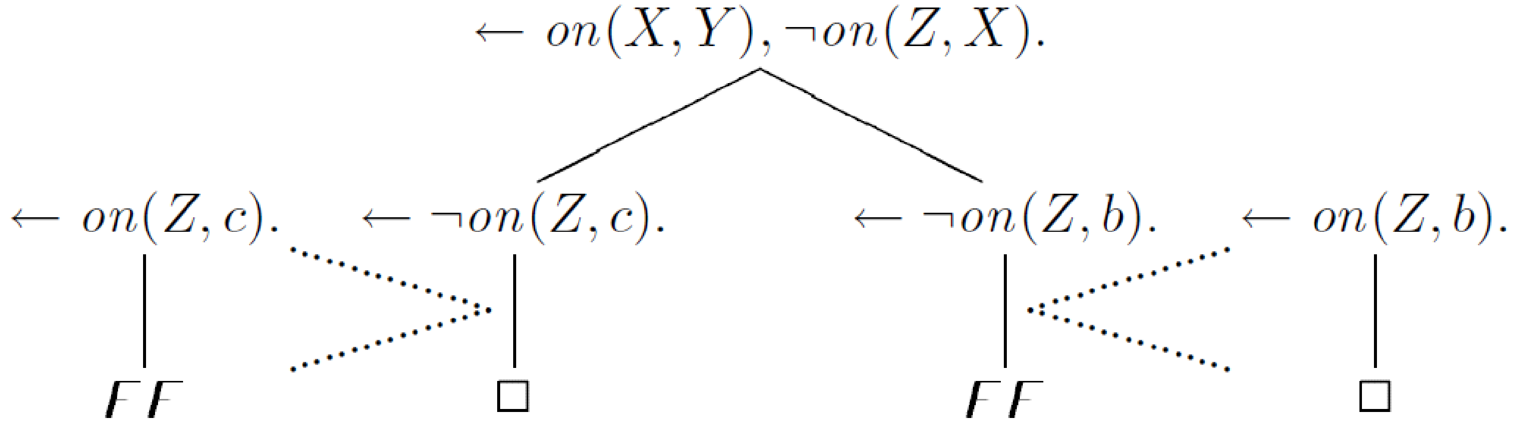
(ii) the $\Re$-selected literal in G_i is of the form $\neg A$, the goal $\leftarrow A$ has a finitely failed SLD-tree and G_{i+1} is obtained from G_i by removing $\neg A$ (in which case C_i is a special marker FF).

SLDNF Resolution

- SLDNF-derivations can lead
  - Refutation
  - Finite failure : A derivation is said to be (fi*nitely*) *failed* if (1) the selected literal is positive and does not unify with the head of any clause or (2) the selected literal is negative and finitely failed;
  - Infinitely failure : A derivation is said to be *stuck* if the selected subgoal is of the form *:A* and *A* is infinitely failed;

$$on(c, b).$$
$$on(b, a).$$

$$\text{goal} \leftarrow \ on(X, Y), \neg on(Z, X)$$



SLDNF Resolution

- Are there any blocks, *X* and *Y* , such that *X* is not on top of *Y* ?

$$\text{goal} \leftarrow \neg on(X, Y)$$

- -*on*(*X; Y*) fails since the goal *on*(*X; Y*) succeeds.
- success of *on*(*X; Y*) does not necessarily mean that there is *no* block which is not on top of another block | only that there *exists* at least one block which is on top of another block.
- -*on*(*X; Y*) should not be read as an existential query but rather as
- a universal test: “For all blocks, *X* and *Y* , is *X* not on top of *Y* ?"