WIKIPEDIA

SLD resolution

SLD resolution (*Selective Linear Definite* clause resolution) is the basic inference rule used in logic programming. It is a refinement of resolution, which is both sound and refutation complete for Horn clauses.

Contents

The SLD inference rule

Given a goal clause, represented as the negation of a problem to be solved :

$$\neg L_1 \lor \cdots \lor \neg L_i \lor \cdots \lor \neg L_n$$

with selected literal $\neg L_i$, and an input definite clause:

$$L \lor \neg K_1 \lor \cdots \lor \neg K_m$$

whose positive literal (atom) L unifies with the atom L_i of the selected literal $\neg L_i$, SLD resolution derives another goal clause, in which the selected literal is replaced by the negative literals of the input clause and the unifying substitution θ is applied:

$$(\neg L_1 \lor \cdots \lor \neg K_1 \lor \cdots \lor \neg K_m \ \lor \cdots \lor \neg L_n)\theta$$

In the simplest case, in propositional logic, the atoms L_i and L are identical, and the unifying substitution θ is vacuous. However, in the more general case, the unifying substitution is necessary to make the two literals identical.

The origin of the name "SLD"

The name "SLD resolution" was given by Maarten van Emden for the unnamed inference rule introduced by Robert Kowalski.[1] Its name is derived from SL resolution,[2] which is both sound and refutation complete for the unrestricted clausal form of logic. "SLD" stands for "SL resolution with Definite clauses".

In both, SL and SLD, "L" stands for the fact that a resolution proof can be restricted to a linear sequence of clauses:

$C_1, C_2, \cdots, C_l$

where the "top clause" C_1 is an input clause, and every other clause C_{i+1} is a resolvent one of whose parents is the previous clause C_i. The proof is a refutation if the last clause C_l is the empty clause.

In SLD, all of the clauses in the sequence are goal clauses, and the other parent is an input clause. In SL resolution, the other parent is either an input clause or an ancestor clause earlier in the sequence.

In both SL and SLD, "S" stands for the fact that the only literal resolved upon in any clause C_i is one that is uniquely selected by a selection rule or selection function. In SL resolution, the selected literal is restricted to one which has been most recently introduced into the clause. In the simplest case, such a last-in-first-out selection function can be specified by the order in which literals are written, as in Prolog. However, the selection function in SLD resolution is more general than in SL resolution and in Prolog. There is no restriction on the literal that can be selected.

The computational interpretation of SLD resolution

In clausal logic, an SLD refutation demonstrates that the input set of clauses is unsatisfiable. In logic programming, however, an SLD refutation also has a computational interpretation. The top clause $\neg L_1 \vee \cdots \vee \neg L_i \vee \cdots \vee \neg L_n$ can be interpreted as the denial of a conjunction of subgoals $L_1 \wedge \cdots \wedge L_i \wedge \cdots \wedge L_n$. The derivation of clause C_{i+1} from C_i is the derivation, by means of backward reasoning, of a new set of sub-goals using an input clause as a goal-reduction procedure. The unifying substitution θ both passes input from the selected subgoal to the body of the procedure and simultaneously passes output from the head of the procedure to the remaining unselected subgoals. The empty clause is simply an empty set of subgoals, which signals that the initial conjunction of subgoals in the top clause has been solved.

SLD resolution strategies

SLD resolution implicitly defines a search tree of alternative computations, in which the initial goal clause is associated with the root of the tree. For every node in the tree and for every definite clause in the program whose positive literal unifies with the selected literal in the goal clause associated with the node, there is a child node associated with the goal clause obtained by SLD resolution.

A leaf node, which has no children, is a success node if its associated goal clause is the empty clause. It is a failure node if its associated goal clause is non-empty but its selected literal unifies with no positive literal of definite clauses in the program.

SLD resolution is non-deterministic in the sense that it does not determine the search strategy for exploring the search tree. Prolog searches the tree depth-first, one branch at a time, using backtracking when it encounters a failure node. Depth-first search is very efficient in its use of computing resources, but is incomplete if the search space contains infinite branches and the search strategy searches these in preference to finite branches: the computation does not terminate. Other search strategies, including breadth-first, best-first, and branch-and-bound search are also possible. Moreover, the search can be carried out sequentially, one node at a time, or in parallel, many nodes simultaneously.

SLD resolution is also non-deterministic in the sense, mentioned earlier, that the selection rule is not determined by the inference rule, but is determined by a separate decision procedure, which can be sensitive to the dynamics of the program execution process.

The SLD resolution search space is an or-tree, in which different branches represent alternative computations. In the case of propositional logic programs, SLD can be generalised so that the search space is an and-or tree, whose nodes are labelled by single literals, representing subgoals, and nodes are joined either by conjunction or by disjunction. In the general case, where conjoint subgoals share variables, the and-or tree representation is more complicated.

Example

Given the logic program:

```
q :- p.
p.
```

and the top-level goal:

```
q.
```

the search space consists of a single branch, in which `q` is reduced to `p` which is reduced to the empty set of subgoals, signalling a successful computation. In this case, the program is so simple that there is no role for the selection function and no need for any search.

In clausal logic, the program is represented by the set of clauses:

$q \vee \neg p$

p

and top-level goal is represented by the goal clause with a single negative literal:

$\neg q$

The search space consists of the single refutation:

$\neg q, \neg p, \mathit{false}$

where false represents the empty clause.

If the following clause were added to the program:

```
q :- r.
```

then there would be an additional branch in the search space, whose leaf node `r` is a failure node. In Prolog, if this clause were added to the front of the original program, then Prolog would use the order in which the clauses are written to determine the order in which the branches of the search space are investigated. Prolog would try this new branch first, fail, and then backtrack to investigate the single branch of the original program and succeed.

If the clause

```
p :- p.
```

were now added to the program, then the search tree would contain an infinite branch. If this clause were tried first, then Prolog would go into an infinite loop and not find the successful branch.

SLDNF

SLDNF[3] is an extension of SLD resolution to deal with negation as failure. In SLDNF, goal clauses can contain negation as failure literals, say of the form $not(p)$, which can be selected only if they contain no variables. When such a variable-free literal is selected, a subproof (or subcomputation) is attempted to determine whether there is an SLDNF refutation starting from the corresponding unnegated literal p as top clause. The selected subgoal $not(p)$ succeeds if the subproof fails, and it fails if the subproof succeeds.

See also

- John Alan Robinson

References

1. Robert Kowalski Predicate Logic as a Programming Language (http://www.doc.ic.ac.uk/~rak/papers/IFIP%2074.pdf) Memo 70, Department of Artificial Intelligence, University of Edinburgh. 1973. Also in Proceedings IFIP Congress, Stockholm, North Holland Publishing Co., 1974, pp. 569-574.
2. Robert Kowalski and Donald Kuehner, Linear Resolution with Selection Function (http://www.doc.ic.ac.uk/~rak/papers/sl.pdf) Artificial Intelligence, Vol. 2, 1971, pp. 227-60.
3. Krzysztof Apt and Maarten van Emden, Contributions to the Theory of Logic Programming (http://dl.acm.org/citation.cfm?doid=322326.322339), Journal of the Association for Computing Machinery. Vol, 1982 - portal.acm.org

- Jean Gallier, *SLD-Resolution and Logic Programming (https://cs.uwaterloo.ca/~david/cl/sld-gallier.pdf)* chapter 9 of *Logic for Computer Science: Foundations of Automatic Theorem Proving (http://www.cis.upenn.edu/~jean/gbooks/logic.html)*, 2003 online revision (free to download), originally published by Wiley, 1986
- John C. Shepherdson, *SLDNF-Resolution with Equality*, Journal of Automated Reasoning 8: 297-306, 1992; defines semantics with respect to which SLDNF-resolution with equality is sound and complete

External links

- [1] (http://foldoc.org/?SLD+resolution) Definition from the Free On-Line Dictionary of Computing