ANL Loop

Jiří Vyskočil

2017
ANL Loop

- The first use in provers developed at ANL (Argonne National Laboratory) where the most well-known prover is Otter by W. McCune.

- We assume that the input conjecture have to be part of the resulting refutation proof.

- ANL Loop guarantees the exploration of all needed combinations of clauses for complete resolution.

- ANL Loop tries to avoid redundant inferences as much as possible.

- It is independent of the chosen clause selection strategy.
ANL Loop

SOS := input clause;  // Clauses in list SOS (set of support) are not available to make inferences;
// they are waiting to participate in the search.

usable := empty set;  // This list contains clauses that are available to make inferences.

while (SOS is not empty and no refutation has been found)
{
    1. Let given_clause be the “best” clause in SOS;
    2. Move given_clause from SOS to usable;
    3. Infer and process new clauses using the inference rules in effect where:
       □ each new clause must have:
          ▪ the given_clause as one of its parents and
          ▪ members of usable as its other parents;
    4. new clauses that pass the retention tests are appended to SOS;
}

logical reasoning and programming
Clause Selection Strategies

- **DFS (Depth-First Search):**
  - Choosing the last/newest resolvent in SOS.
  - This is not complete (endless looping possible).
  - Does not guarantee the shortest proof.

- **BFS (Breadth-First Search):**
  - Choosing the first/oldest resolvent in SOS.
  - It is complete.
  - It will find the shortest proof if exists.
  - “ply-by-ply search”

- **Best First/Clause Search:**
  - We will choose “the best” clause in SOS.
  - If this selection does not guarantee completeness then we can combine this strategy with BFS (e.g. every 10th clause is selected by BFS)
ANL Loop with Subsumption

*SOS* := input clause;

*usable* := empty set;

**while** (*SOS* is not empty and no refutation has been found)

{  
  1. Let *given_clause* be the “best” clause in *SOS*;
  2. *SOS* := *SOS* \ *given_clause*;
     
     If *usable* ⊑ \{*given_clause*\} or *SOS* ⊑ \{*given_clause*\} then continue;
     
     *usable* := \{ D ∈ *usable* | *given_clause* △ D \} \cup \{*given_clause*\};
  3. Infer and process new clauses using the inference rules in effect where:
     
     □ each new clause *Q* must have:
       
       ▪ the *given_clause* as one of its parents and
       ▪ members of *usable* as its other parents;
       ▪ *usable* △ \{*Q*\}
  4. new clauses that pass the retention tests are appended to *SOS*;
}

logical reasoning and programming