First-Order Theorem Proving

Vampire Cookies

Laura Kovács and Andrei Voronkov
TU Wien and U. Manchester and EasyChair

From theory to practice

- Preprocessing and CNF transformation;
- Superposition system;
- Orderings;
- Selection functions;
- Fairness (saturation algorithms);
- Redundancy.

Vampire's preprocessing (incomplete list)

1. (Optional) Select a relevant subset of formulas.
2. (Optional) Add theory axioms;
3. Rectify the formula.
4. If the formula contains any occurrence of $\top$ or $\bot$, simplify the formula.
5. Remove if-then-else and let-in connectives.
6. Apply pure predicate elimination.
7. (Optional) Remove unused predicate definitions.
8. Convert the formula into equivalence negation normal form (ENNF).
9. Use a naming technique to replace some subformulas by their names.
10. Convert the formula into negation normal form (NNF).
11. Skolemize the formula.
12. (Optional) Replace equality axioms.
13. Determine a literal ordering to be used.
14. Transform the formula into its clausal normal form.
15. Remove tautologies.
16. Pure literal elimination.

How to Design a Good Saturation Algorithm?

A saturation algorithm must be fair: every possible generating inference must eventually be selected.

Two main implementation principles:

apply simplifying inferences eagerly; apply generating inferences lazily.	checking for simplifying inferences should pay off; so it must be cheap.

Given Clause Algorithm (no Simplification)

```
  input: init: set of clauses;
  var active, passive, queue: sets of clauses;
  var current: clauses ;
  active := ∅;
  passive := init;
  while passive ≠ ∅ do
*   current := select(passive);                      (* clause selection *)
    move current from passive to active;
*   queue:=infer(current, active);                   (* generating inferences *)
    if □ ∈ queue then return unsatisfiable;
    passive := passive ∪ queue
  od;
  return satisfiable
```

Given Clause Algorithm (with Simplification)

In fact, there is more than one . . .

Otter Saturation Algorithm

```
    input: init: set of clauses;
    var active, passive, unprocessed: set of clauses;
    var given, new: clause;
    active := ∅;
    unprocessed := init;
    loop
      while unprocessed ≠ ∅
        new:=pop(unprocessed);
        if new = □ then return unsatisfiable;
*       if retained(new) then                              (* retention test *)
*         simplify new by clauses in active ∪ passive ;(* forward simplification *)
          if new = □ then return unsatisfiable;
*         if retained(new) then                    (* another retention test *)
*           delete and simplify clauses in active and (* backward simplification *)
                                passive using new;
            move the simplified clauses to unprocessed;
            add new to passive
      if passive = ∅ then return satisfiable or unknown
*     given := select(passive);                          (* clause selection *)
      move given from passive to active;
*     unprocessed:=infer(given, active);          (* generating inferences *)
```

Age-Weight Ratio

How to select nice clauses?

- Small clauses are nice.
- Selecting only small clauses can postpone the selection of an old clause (e.g., input clause) for too long, in practice resulting in incompleteness.

Age-Weight Ratio

How to select nice clauses?

- Small clauses are nice.
- Selecting only small clauses can postpone the selection of an old clause (e.g., input clause) for too long, in practice resulting in incompleteness.

Solution:

- A fixed percentage of clauses is selected by weight, the rest are selected by age.
- So we use an age-weight ratio $a : w$: of each $a + w$ clauses select a oldest and w smallest clauses.

Limited Resource Strategy

Limited Resource Strategy: try to approximate which clauses are unreachable by the end of the time limit and remove them from the search space.

Limited Resource Strategy

Limited Resource Strategy: try to approximate which clauses are unreachable by the end of the time limit and remove them from the search space.

Try:

```
vampire --age_weight_ratio 4:1
  --forward_subsumption_resolution off
  --time_limit 20
   GRP140-1.p
```

CASC Mode

```
vampire --mode casc SET014-3.p
```