mSAT : An OCaml SAT Solver

Guillaume Bury

DEDUCTEAM (INRIA) - LSV / CNRS

guillaume.bury@inria.fr

Introduction

mSAT is a SAT solving library written in OCaml. It allows to solve the satisfiability of propositional problems in clausal normal form, and produce either a propositional model, or a resolution proof of the problem's unsatisfiability.

Conflict Driven Clause learning

Propagation If there exists a clause $C = C' \vee a$, where C' is false in the partial model, then add $a \mapsto \top$ to the partial model, and record C as the reason for a.

Decision Take an atom a which is not yet in the partial model, and add $a \mapsto \top$ to the model.

Conflict A conflict is a clause C that is false in the current partial model.

Analyze Perform resolution between the analyzed clause and the reason behind the propagation of its most recently assigned litteral, until the analyzed clause is suitable for backumping

Backjump A clause is suitable for backjumping if its most recently assigned litteral a is a decision. We can then backtrack to before the decision, and add the analyzed clause to the solver, which will then enable to propagate $a \mapsto \bot$.

SMT Formulas using first-order theories can be handled using a theory. Each formula propagated or decided is sent to the theory, which then has the duty to check whether the conjunction of all formulas seen so far is satisfiable, if not, it should return a theory tautology (as a clause), that is not satisfied in the current partial model.

Theory interface

```
type ('formula, 'proof) res =
  | Sat | Unsat of 'formula list * 'proof

type ('form, 'proof) slice = {
  start : int;
  length : int;
  get : int -> 'form;
  push : 'form list -> 'proof -> unit;
}

module type S = sig
  val dummy : level
  val backtrack : level -> unit
  val current_level : unit -> level
  val assume : (formula, proof) slice
    -> (formula, proof) res
  val if_sat : (formula, proof) slice
    -> (formula, proof) res
end
```

Proof management

```
type clause_premise =
    | Hyp | Local | Lemma of lemma
    | History of clause list

type proof = clause
and proof_node = {
  conclusion : clause;
  step : step;
}
and step =
  | Hypothesis
  | Assumption
  | Lemma of lemma
  | Duplicate of proof * atom list
  | Resolution of proof * proof * atom

val expand : proof -> proof_node
```

mSAT is available on opam and on github : `https://github.com/Gbury/mSAT`