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.

Implementation

- ▶ Imperative design
  - ✓ 2-watch litteral
  - ✓ Generative functors
  - ✓ Backtrackable theories (less demanding than immutable theories)
- ▶ Features
  - ✓ Functorized design
  - ✓ Local assumptions
  - ✓ Model output
  - ✓ Proof output (Coq, dot)

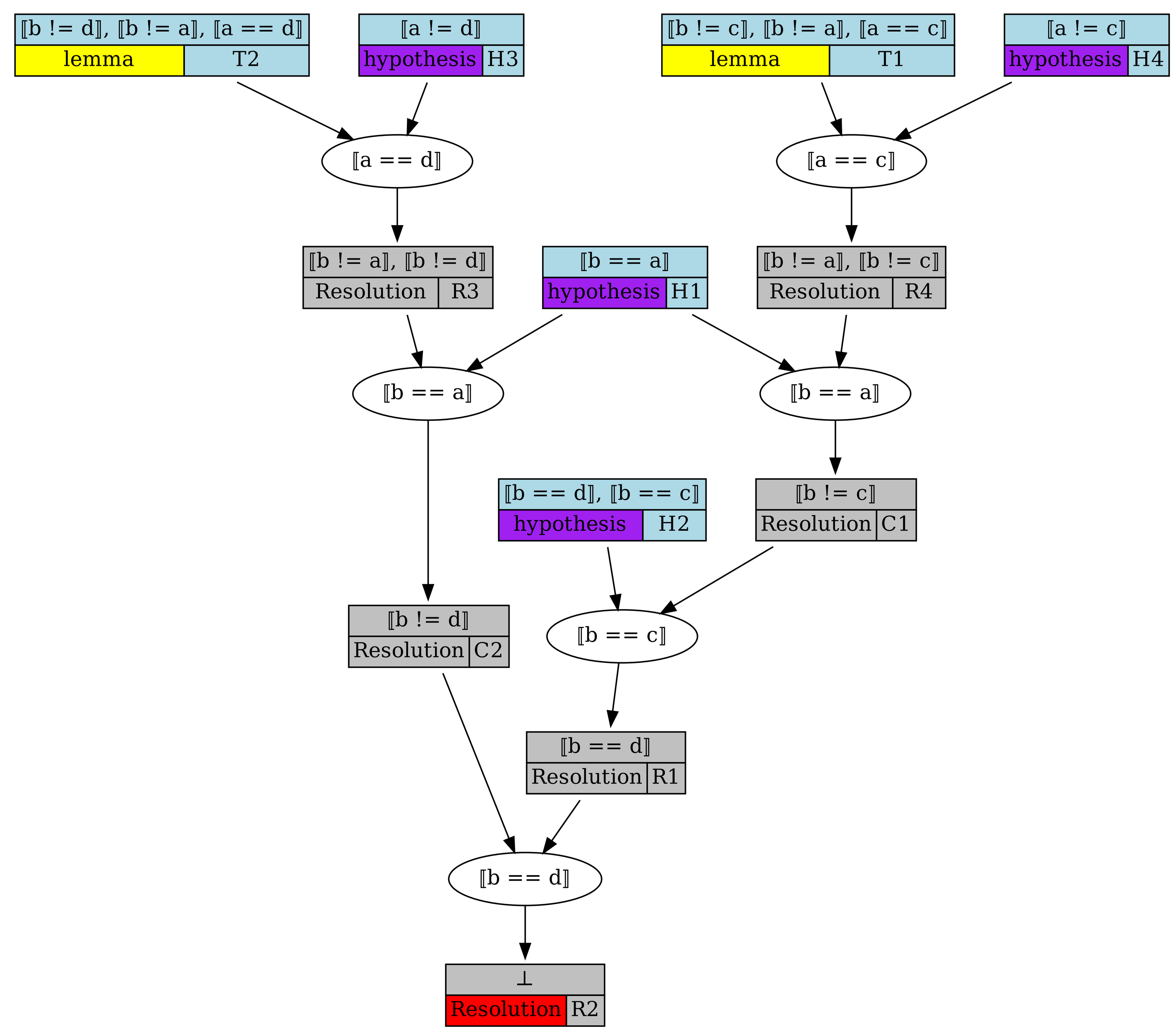
Other solvers

regstab	SAT	binary only	only pure SAT
minisat sattools ocaml-sat-solvers	SAT	C bindings	only pure SAT
Alt-ergo	SMT	binary only	Fixed theory
Alt-ergo-zero	SMT	OCaml lib	Fixed theory
ocamlyices yices2	SMT	C bindings	Fixed theory

Problem example

$H1: \ a = b \qquad H2: b = c \vee b = d$

$H3: a <> d \qquad H4: \ a <> c$



Proof generation

✓ Each clause records it "history", that is the clauses used during analyzing

✓ Minimal impact on proof search (already done to compute unsat-core)

✓ Sufficient to rebuild the whole resolution tree

✓ A proof is a clause and proof nodes are lazily expanded
$\rightarrow$ no memory issue

✓ Enables various proof output :

- Dot/Graphviz (see example above)
- Coq formal proof

Performances

solvers	aez	mSAT	minisat (minisat/sattools)	cryptominisat (sattools)
uuf100 (1000 pbs)	0.125	0.012	0.004	0.006
uuf125 (100 pbs)	2.217	0.030	0.006	0.013
pigeon/hole6	0.120	0.018	0.006	0.006
pigeon/hole7	4.257	0.213	0.015	0.073
pigeon/hole8	31.450	0.941	0.096	2.488
pigeon/hole9	timeout (600)	8.886	0.634	4.075
pigeon/hole10	timeout (600)	161.478	9.579 (minisat) 160.376 (sattools)	72.050

Done using https ://github.com/Gbury/sat-bench.

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