mSAT: A Modular SAT Solver

Guillaume Bury

October 2, 2017

Université Paris Diderot; Inria; LSV, ENS Cachan

Introduction

Introduction: mSAT

- SAT/SMT Solving library in OCaml
- Modular: provide your own theory
- Proof producing: check your proofs in Coq

Some design decisions

- Forked from Alt-Ergo-Zero
- Imperative design
- Functorized for modularity
- Generative functors

SAT Solving

Goal of the algorithm

Input A set of clauses of propositional formulas, for instance:

$$P \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg Q)$$

Output Either:

- A model of the input clauses
- A proof the the clauses are unsatisfiable

Simplified control flow

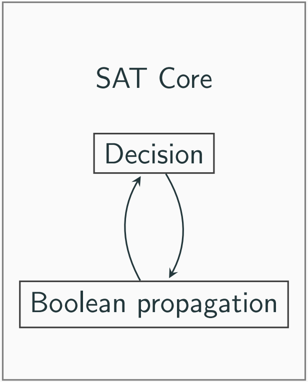


Figure 1: Simplified SAT Solver architecture

SAT Solving Algorithm

- Maintain a partial propositional model
- Propagation
  - If there exists a clause $C = a \vee c_1 \vee \ldots \vee c_n$, where every $c_i \leadsto \bot$ in the current partial model, then add $a \leadsto_C \top$ to the model
  - Record the clause C as the **reason** for the propagation of a
- Decision
  - When no propagation is possible
  - Choose an unassigned litteral a
  - Add $a \mapsto \top$ to the model

SAT Solving Algorithm

- When there is a clause $C = c_1 \vee \ldots \vee c_n$, where every $c_i \mapsto \bot$, begin analyzing with current clause C
- Walk back the propagations/decisions from most recent
- If the currently looked at atom is:
  - Not part of the current clause, continue
  - part of the current clause, and propagated by a clause D, perform a resolution between the current clause and D:

$$\frac{C \vee p \qquad \neg p \vee D}{C \vee D}$$

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b), C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b), C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$

SAT Solving - Example sat

- $C_1 = \neg p(a) \lor p(b), C_2 = \neg p(a) \lor \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$
- Propagation in $C_1 = \neg p(a) \lor p(b)$: $p(b) \rightsquigarrow_{C_1} \top$

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b), C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \leadsto_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b), C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \leadsto_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b)$, $C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \leadsto_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$
- New clause : $C_3 = \neg p(a)$, backtrack to before decision.

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b), C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \leadsto_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$
- New clause : $C_3 = \neg p(a)$, backtrack to before decision.
- Propagation: $p(a) \leadsto_{C_3} \bot$

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b), C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \leadsto_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$
- New clause : $C_3 = \neg p(a)$, backtrack to before decision.
- Propagation: $p(a) \leadsto_{C_3} \bot$
- Decision: $p(b) \mapsto \top$

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b), C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \rightsquigarrow_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$
- New clause : $C_3 = \neg p(a)$, backtrack to before decision.
- Propagation: $p(a) \rightsquigarrow_{C_3} \bot$
- Decision: $p(b) \mapsto \top$
- Propagation (nothing to do)

SAT Solving - Example sat

- $C_1 = \neg p(a) \vee p(b), C_2 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Decision: $p(a) \mapsto \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \rightsquigarrow_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$
- New clause : $C_3 = \neg p(a)$, backtrack to before decision.
- Propagation: $p(a) \rightsquigarrow_{C_3} \bot$
- Decision: $p(b) \mapsto \top$
- Propagation (nothing to do)
- Model Found !

SAT Solving - Example unsat

- $C_0 = p(a), C_1 = \neg p(a) \vee p(b), C_3 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false

SAT Solving - Example unsat

- $C_0 = p(a), C_1 = \neg p(a) \vee p(b), C_3 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Propagation: $p(a) \mapsto_{C_0} \top$

SAT Solving - Example unsat

- $C_0 = p(a), C_1 = \neg p(a) \vee p(b), C_3 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Propagation: $p(a) \mapsto_{C_0} \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \rightsquigarrow_{C_1} \top$

SAT Solving - Example unsat

- $C_0 = p(a), C_1 = \neg p(a) \vee p(b), C_3 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Propagation: $p(a) \mapsto_{C_0} \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \leadsto_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied

SAT Solving - Example unsat

- $C_0 = p(a), C_1 = \neg p(a) \vee p(b), C_3 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Propagation: $p(a) \mapsto_{C_0} \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \leadsto_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$

SAT Solving - Example unsat

- $C_0 = p(a), C_1 = \neg p(a) \vee p(b), C_3 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Propagation: $p(a) \mapsto_{C_0} \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \rightsquigarrow_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$
- Resolution between $T_1 = \neg p(a)$ and $C_0 = p(a)$

SAT Solving - Example unsat

- $C_0 = p(a), C_1 = \neg p(a) \vee p(b), C_3 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Propagation: $p(a) \mapsto_{C_0} \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \rightsquigarrow_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$
- Resolution between $T_1 = \neg p(a)$ and $C_0 = p(a)$
- Empty clause $C_4 = \bot$ reached

SAT Solving - Example unsat

- $C_0 = p(a), C_1 = \neg p(a) \vee p(b), C_3 = \neg p(a) \vee \neg p(b)$
- Problem: find a model or a proof of false
- Propagation: $p(a) \mapsto_{C_0} \top$
- Propagation in $C_1 = \neg p(a) \vee p(b)$: $p(b) \rightsquigarrow_{C_1} \top$
- Conflict: $C_2 = \neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $C_2 = \neg p(a) \vee \neg p(b)$ and $C_1 = \neg p(a) \vee p(b)$
- Resolution between $T_1 = \neg p(a)$ and $C_0 = p(a)$
- Empty clause $C_4 = \bot$ reached
- Input problem is unsatisfiable

Builtin SAT (1)

```
(* Module initialization *)
module Sat = Msat.Sat.Make()
module E = Msat.Sat.Expr (* expressions *)
module F = Msat.Tseitin.Make(E)
(* We create here two distinct atoms *)
let a = E.fresh ()
let b = E.make 1
```

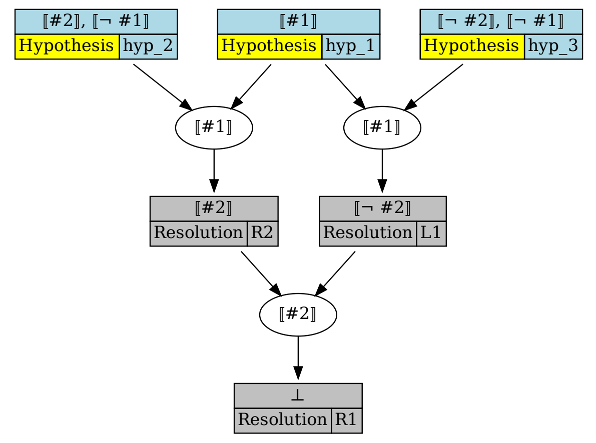
Builtin SAT (2)

```
(* Let's create some formulas *)
let p = F.make_atom a
let q = F.make_atom b
let r = F.make_and [p; q]
let s = F.make_or [F.make_not p; F.make_not q]

let () = Sat.assume (F.make_cnf r)
let _ = Sat.solve () (* Should return (Sat.Sat _) *)

let () = Sat.assume (F.make_cnf s)
let _ = Sat.solve () (* Should return (Sat.Unsat _) *)
```

SAT Solving - proofs



SMT Solving

Goal of the algorithm

Input A set of clauses of first-order formulas, for instance:

$$(a = b) \wedge (a <> c) \wedge (a <> d) \wedge (a = c \vee a = d)$$

Output Either:

- A model of the input clauses
- A proof the the clauses are unsatisfiable

Simplified control flow

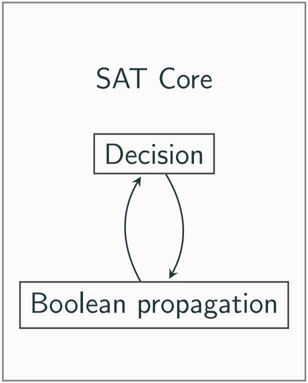


Figure 2: Simplified SAT/SMT Solver architecture

Simplified control flow

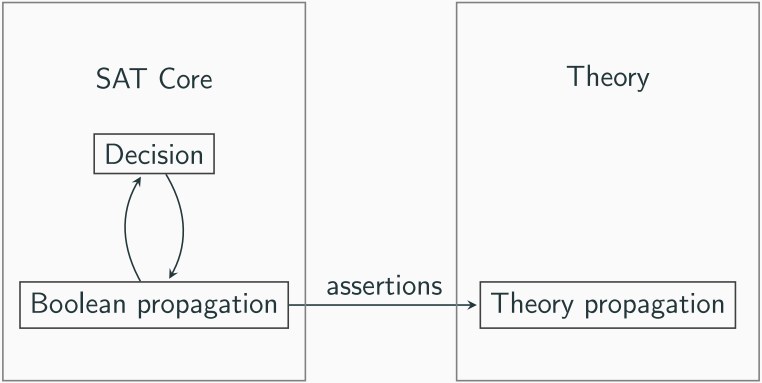
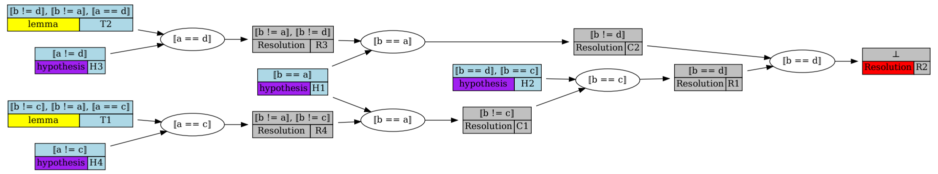


Figure 2: Simplified SAT/SMT Solver architecture

SMT proofs vs SAT proofs

- Leafs can be either:
  - A Hypothesis
  - A Theory lemma
- A theory lemma is a tautology in the theory, for instance:
  - Equality reflexivity: Lemma $= (a = a)$
  - Equality transitivty: Lemma $= \neg(a = b) \vee \neg(b = c) \vee (a = c)$
  - Equality substitution: Lemma $= \neg(a = b) \vee (f(a) = f(b))$

SMT proofs



⟦b != d⟧, ⟦b != a⟧, ⟦a == d⟧
lemma
T2
⟦a != d⟧
hypothesis
H3
⟦a == d⟧
⟦b != a⟧, ⟦b != d⟧
Resolution
R3
⟦b == a⟧
⟦b == a⟧
hypothesis
H1
⟦b != d⟧
Resolution
C2
⟦b == d⟧, ⟦b == c⟧
hypothesis
H2
⟦b == c⟧
⟦b == d⟧
Resolution
R1
⟦b == d⟧
⊥
Resolution
R2
⟦b != c⟧, ⟦b != a⟧, ⟦a == c⟧
lemma
T1
⟦a == c⟧
⟦b != a⟧, ⟦b != c⟧
Resolution
R4
⟦b == a⟧
⟦b != c⟧
Resolution
C1
⟦a != c⟧
hypothesis
H4

The Solver Functor

```
module Make
    (F : Formula_intf.S)
    (Th : Theory_intf.S with type formula = F.t
                         and type proof = F.proof)
    (Dummy: sig end) :
  S with type St.formula = F.t
     and type St.proof = F.proof
```

The Formula interface

```
type negated = Negated | Same_sign

module type S = sig
  type t
  type proof

  val hash : t -> int
  val equal : t -> t -> bool
  val print : Format.formatter -> t -> unit

  val dummy : t
  val neg : t -> t
  val norm : t -> t * negated
end
```

The Theory interface

```
type ('f, 'p) res = Sat | Unsat of 'f list * 'p
type 'f slice = { start:int; length:int; get:int -> 'f }
module type S = sig
  type f (** formulas *)
  type proof

  type level
  val dummy : level
  val current_level : unit -> level
  val backtrack : level -> unit
  val assume : (f, proof) slice -> (f, proof) res
  val if_sat : (f, proof) slice -> (f, proof) res
end
```

The Solver interface

```
type 'f sat_state =
  { eval : 'f -> bool; ... }

type ('c,'p) unsat_state =
  { conflict: unit -> 'c; proof : unit -> 'p }

type res = Sat of formula sat_state
         | Unsat of (clause, proof) unsat_state

val assume : ?tag:int -> atom list list -> unit

val solve : ?assumptions:atom list -> unit -> res
```

Proof output

- Dot output
- Forma Coq output

Conclusion

Related Works

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

Performances

solver (package)	Alt-ergo-zero (aez)	mSAT (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
uuf150 (100 pbs)	67.563	0.087	0.017	0.045
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

Conclusion

- Pure OCaml SAT Solver
- Decent performances
- Modular
- Proof producing (Coq, and soon Dedukti)
- Available on opam, and on github:
  https://github.com/Gbury/mSAT

Proof objects

```
type proof
and proof_node = {
  conclusion : clause;
  step : step;
}
and step =
  | Hypothesis
  | Assumption
  | Lemma of lemma
  | Duplicate of proof * atom list
  | Resolution of proof * proof * atom
(** The type of reasoning steps allowed in a proof. *)
```