mSAT: An OCaml SAT Solver
Guillaume Bury

To cite this version:
Guillaume Bury. mSAT: An OCaml SAT Solver. OCaml Users and Developers Workshop, Sep 2017, Oxford, United Kingdom. hal-01670765

HAL Id: hal-01670765
https://inria.hal.science/hal-01670765
Submitted on 21 Dec 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientifi c research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
mSAT: An OCaml SAT Solver

Guillaume Bury
DEDUC-EAM (INRIA) - LSV / CNRS
guillaume.bury@inria.fr

Introduction

mSAT: a SAT solving library in OCaml. It solves the satisfiability of propositional clauses. It is Modular: the user provides the theory. And it produces formal proofs.

Conflict Driven Clause Learning

Propagation: If there exists a clause \( C = C' \lor a \), where \( C' \) is false in the partial model, then add \( a \rightarrow \top \) to the partial model, and record \( C \) as the reason for \( a \).

Decision: Take an atom \( a \) that is not yet in the partial model, and add \( a \rightarrow \top \) 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 literal, until the analyzed clause is suitable for backjumping.

Backjump: A clause is suitable for backjumping if its most recently assigned literal \( a \) 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 \( a \rightarrow \bot \).

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 literal
- √ Backtrackable theories (less demanding than immutable theories)
- Features
  √ Functorized design, using generative functors
  √ Local assumptions
  √ Model output and proof output (Coq, dot)

Solver Interface

```ocaml
type ('f, 'p) res = Sat | Unsat of 'f list * 'p

module Make(Th: Theory_intf.S)() : sig
  type ('t, 'a) res = ('t, 'a) res
  type atom list
  type formula slice

  val assume : formula slice -> (formula, proof) res
  val backtrack : level -> unit
  val current_level : unit -> level
  val backtrack : level -> unit
  val solve : ?assumptions:atom list -> unit -> res

  val assume : formula slice -> (formula, proof) res
  val backtrack : level -> unit
  val current_level : unit -> level
  val backtrack : level -> unit
  val solve : ?assumptions:atom list -> unit -> res

  end
```

Problem Example

Are the following hypotheses satisfiable?

- \( H_1: a = b \)
- \( H_2: b = c \lor b = d \)
- \( H_3: a \not< \not< d \)
- \( H_4: a \not< \not< c \)

Theory Interface

```ocaml
type ('f, 'p) res = Sat | Unsat of 'f list * 'p

module Make(Th: Theory_intf.S)() : sig
  type ('t, 'a) res = ('t, 'a) res
  type atom list
  type formula slice

  val assume : formula slice -> (formula, proof) res
  val backtrack : level -> unit
  val current_level : unit -> level
  val backtrack : level -> unit
  val solve : ?assumptions:atom list -> unit -> res

  end
```

Proof Generation

- ✓ Each clause records its "history" which 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 expanded on demand
- ✓ No memory issue
- ✓ Enables various proof outputs:
  - • Dot/Graphviz (see example above)
  - • Coq (and soon Deduki) formal proofs

Other Solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>SAT</th>
<th>binary only</th>
<th>only pure SAT</th>
<th>C bindings</th>
<th>only pure SAT</th>
<th>Fixed theory</th>
<th>Fixed theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>regstab</td>
<td>SAT</td>
<td>binary only</td>
<td>only pure SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>minisat</td>
<td>SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>satsatools</td>
<td>SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>ocaml-sat-solvers</td>
<td>SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>Alt-ergo</td>
<td>SMT</td>
<td>binary only</td>
<td>Fixed theory</td>
<td>SMT</td>
<td>OCaml lib</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>Alt-ergo-zero</td>
<td>SMT</td>
<td>OCaml lib</td>
<td>Fixed theory</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>yices2</td>
<td>SMT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>Alt-ergo-zero</td>
<td>SMT</td>
<td>binary only</td>
<td>Fixed theory</td>
<td>SMT</td>
<td>OCaml lib</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>yices2</td>
<td>SMT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>Alt-ergo-zero</td>
<td>SMT</td>
<td>binary only</td>
<td>Fixed theory</td>
<td>SMT</td>
<td>OCaml lib</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
<tr>
<td>yices2</td>
<td>SMT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>C bindings</td>
<td>only pure SAT</td>
<td>Fixed theory</td>
<td>Fixed theory</td>
</tr>
</tbody>
</table>

Performances

<table>
<thead>
<tr>
<th>Solver</th>
<th>Alt-ergo-zero</th>
<th>mSAT</th>
<th>minisat</th>
<th>cryptominisat</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuf100 (1000 pbs)</td>
<td>0.125</td>
<td>0.012</td>
<td>0.004</td>
<td>0.006</td>
</tr>
<tr>
<td>uuf125 (1000 pbs)</td>
<td>2.217</td>
<td>0.030</td>
<td>0.006</td>
<td>0.013</td>
</tr>
<tr>
<td>uuf150 (1000 pbs)</td>
<td>67.563</td>
<td>0.087</td>
<td>0.017</td>
<td>0.045</td>
</tr>
<tr>
<td>pigeon/hole6</td>
<td>0.120</td>
<td>0.018</td>
<td>0.006</td>
<td>0.006</td>
</tr>
<tr>
<td>pigeon/hole7</td>
<td>4.257</td>
<td>0.213</td>
<td>0.015</td>
<td>0.073</td>
</tr>
<tr>
<td>pigeon/hole8</td>
<td>31.450</td>
<td>0.941</td>
<td>0.096</td>
<td>2.488</td>
</tr>
<tr>
<td>pigeon/hole9</td>
<td>timeout (600)</td>
<td>8.886</td>
<td>0.634</td>
<td>4.075</td>
</tr>
<tr>
<td>pigeon/hole10</td>
<td>timeout (600)</td>
<td>161.478</td>
<td>9.579</td>
<td>160.376</td>
</tr>
</tbody>
</table>

GUILLAUME BURY

Inria Discrete Mathematics and Algorithms Group
Inria Saclay - Ile de France
Campus de l’Université Paris-Saclay
Route de Saclay
91767 Orsay Cedex
Guillaume.Bury@inria.fr

Comparative Study of SAT Solvers

The work presented in this paper is supported by the French National Research Agency (ANR) through the CEMAD project (ANR-11-INFR-0003-01). The author thanks the reviewers for their helpful comments and suggestions.

References


```latex
\end{document}
```