mSAT: a SAT/SMT/McSAT library

Guillaume Bury

October 14, 2016

1 Introduction

The goal if the mSAT library is to provide a way to easily create atomated theorem provers based on a Sat solver. More precisely, the library, written in OCaml, provides functors which, once instantiated, provide a Sat, SMT or McSat solver.

Given the current state of the art of SMT solvers, where most Sat solvers are written in C and heavily optimised[1], the mSAT library does not aim to provide solvers competitive with the existing implementations, but rather an easy way to create reasonably eficient solvers.

mSAT currently uses the following techniques:

- 2-watch literals scheme
- Activity based decisions
- Restarts

Additionally, mSAT has the following features:

- Local assumptions
- Proof/Model output
- Adding clauses during proof search

[1]Some solvers now have instructions to manage a processor's cache

Contents

2 Sat Solvers: principles and formalization

2.1 Idea

2.2 Inference rules

The SAT algorithm can be formalized as follows. During the search, the solver keeps a set of clauses, containing the problem hypotheses and the learnt clauses, and a trail, which is the current ordered list of assumptions and/or decisions made by the solver.

Each element in the trail (decision or propagation) has a level, which is the number of decision appearing in the trail up to (and including) it. So for instance, propagations made before any decisions have level 0, and the first decision has level 1. Propagations are written $a \leadsto_C \top$, with C the clause that caused the propagation, and decisions $a \mapsto_n \top$, with n the level of the decision. Trails are read chronologically from left to right.

In the following, given a trail t and an atomic formula a, we will use the following notation: $a \in t$ if $a \mapsto_n \top$ or $a \leadsto_C \top$ is in t, i.e $a \in t$ is a is true in the trail t. In this context, the negation $\neg$ is supposed to be involutive (i.e $\neg\neg a = a$), so that, if $a \in t$ then $\neg\neg a = a \in t$.

The SAT algorithm has two states: first, it starts in the Solve state, where propagations and decisions are made, until a conflict is detected, at which point it enters in the Analyse state, where it analyzes the conflict, backtracks, and re-enter the Solve state. The Solve state take as argument the set of hypotheses and the trail, while the Analyze state also take as argument the current conflict clause.

We can now formalize the Sat algorithm using the inference rules in Figure 1. In order to completely recover the Sat algorithm, one must apply the rules with the following precedence and termination conditions, depending on the current state:

- If the empty clause is in $\mathbb{S}$, then the problem is unsat. If there is no more rule to apply, the problem is sat.
- If we are in Solve mode:
    1. First is the rule Conflict;
    2. Then the try and use Propagate;
    3. Finally, is there is nothing left to be propagated, the Decide rule is used.
- If we are in Analyze mode, we have a choice concerning the order of application. First we can observe that the rules Analyze-Propagate, Analyze-Decision and Analyze-Resolution can not apply simultaneously, and we will thus group them in a super-rule Analyze. We now have the choice of when to apply the Backjump rule compared to the Analyze rule: using Backjump eagerly will result in the first UIP strategy, while delaying it until later will yield the last UIP strategy, both of which are valid.

2.3 Invariants, correctness and completeness

The following invariants are maintained by the inference rules in Figure 1:

Trail Soundness In Solve$(\mathbb{S}, t)$, if $a \in t$ then $\neg a \notin t$

Sat

$$\textsc{Propagate}\ \frac{\mathrm{Solve}(\mathbb{S}, t)}{\mathrm{Sove}(\mathbb{S}, t :: a \leadsto_C \top)} \qquad \begin{array}{l} a \in C, C \in \mathbb{S}, \neg a \notin t \\ \forall b \in C.b \neq a \to \neg b \in t \end{array}$$

$$\textsc{Decide}\ \frac{\mathrm{Solve}(\mathbb{S}, t)}{\mathrm{Solve}(\mathbb{S}, t :: a \mapsto_n \top)} \qquad \begin{array}{l} a \notin t, \neg a \notin t, a \in \mathbb{S} \\ n = \mathrm{max\_level}(t) + 1 \end{array}$$

$$\textsc{Conflict}\ \frac{\mathrm{Solve}(\mathbb{S}, t)}{\mathrm{Analyze}(\mathbb{S}, t, C)} \qquad C \in \mathbb{S}, \forall a \in C.\neg a \in t$$

$$\textsc{Analyze-propagation}\ \frac{\mathrm{Analyze}(\mathbb{S}, t :: a \leadsto_C \top, D)}{\mathrm{Analyze}(\mathbb{S}, t, D)} \qquad \neg a \notin D$$

$$\textsc{Analyze-decision}\ \frac{\mathrm{Analyze}(\mathbb{S}, t :: a \mapsto_n \top, D)}{\mathrm{Analyze}(\mathbb{S}, t, D)} \qquad \neg a \notin D$$

$$\textsc{Analyze-Resolution}\ \frac{\mathrm{Analyze}(\mathbb{S}, t :: a \leadsto_C \top, D)}{\mathrm{Analyze}(\mathbb{S}, t, (C - \{a\}) \cup (D - \{\neg a\}))} \qquad \neg a \in D$$

$$\textsc{Backjump}\ \frac{\mathrm{Analyze}(\mathbb{S}, t :: a \mapsto_d \top :: t', C)}{\mathrm{Solve}(\mathbb{S} \cup \{C\}, t)} \qquad \begin{array}{l} \mathrm{is\_uip}(C) \\ d = \mathrm{uip\_level}(C) \end{array}$$

SMT

$$\textsc{Conflict-Theory}\ \frac{\mathrm{Solve}(\mathbb{S}, t)}{\mathrm{Analyze}(\mathbb{S}, t, C)} \qquad \begin{array}{l} \mathcal{T} \vdash C \\ \forall a \in C, \neg a \in t \end{array}$$

McSat

$$\frac{\mathrm{Solve}(\mathbb{S}, t)}{\mathrm{Solve}(\mathbb{S}, t :: a \mapsto_n v)} \qquad a \notin t, a \in \mathbb{S}, n = \mathrm{max\_level}(t) + 1$$

$$\frac{\mathrm{Solve}(\mathbb{S}, t)}{\mathrm{Solve}(\mathbb{S}, t :: a \leadsto_n \top)}$$

Figure 1: Inference rules

Conflict Analysis In Analyze($\mathbb{S}, t, C$), C is a clause implied by the clauses in $\mathbb{S}$, and $\forall a \in C, \neg a \in t$ (i.e C is entailed, yet false in the partial model formed by the trail t).

Equivalence In any rule $\frac{s_1}{s_2}$, the set of hypotheses (usually written $\mathbb{S}$) in s_1 is equivalent to that of s_2.

These invariants are relatively easy to prove, and provide an easy proof of correctness for the Sat algorithm. Termination can be proved by observing that the same trail appears at most twice during proof search (once during propagation, and eventually a second time right after backjumping[2]). Correctness and termination implies completeness of the Sat algorithm.

3 SMT solver architecture

3.1 Idea

We can represent a simplified version of the information flow (not taking into account backtracking) of usual SMT Solvers, using the graph in fig 2. In a SMT solver, after each propagation and decision, the solver sends the newly assigned literals to the theory. The theory then has the possibility to declare the current set of literals incoherent, and give the solver a tautology in which all literals are currently assigned to $\bot$[3], thus prompting the solver to backtrack.

3.2 Formalization and Theory requirements

An SMT solver is the combinaison of a Sat solver, and a theory $\mathcal{T}$. The role of the theory $\mathcal{T}$ is to stop the proof search as soon as the trail of the Sat solver is inconsistent. Thus, we can add the Conflict-Theory rule to the Sat ifnerence rules in order to get a SMT solver. We give a slightly lower precedence than the Conflict rule for performance reason (detecting boolean conflict is faster than theory specifi conflicts).

So, what is the minimum that a theory must implement to be used in a SMT solver ? Given its precedence, all a theory has to do in a SMT sovler, is to ensure that the current trail is consistent (when seens as a conjunction of literals). This reflects the fact that the Sat core takes care of the purely propositional content of the input problem, and leaves the theory specific reasoning to the theory, which then does not have to deal with propositional content such as disjunction. The theory, however, must have a global reasoning to ensure that the whole trail is consistent.

4 McSat: An extension of SMT Solvers

4.1 Introduction

Mcsat is an extension of usual SMT solvers, introduced in [2] and [1]. In usual SMT Solvers, interaction between the core SAT Solver and the Theory is pretty limited : the

[2]This could be avoided by making the Backjump rule directly propagate the relevant litteral of the conflict clause, but it needlessly complicates the rule.

[3]or rather for each literal, its negation is assigned to $\top$

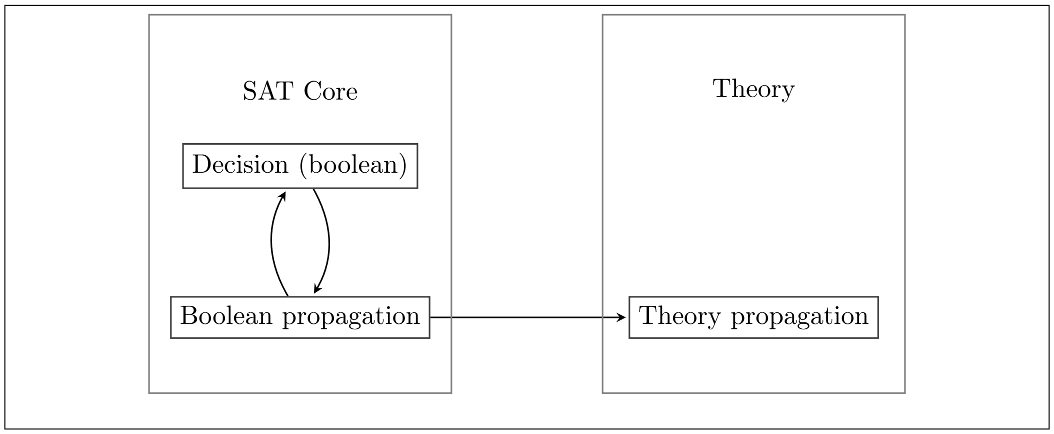

Figure 2: Simplified SMT Solver architecture

SAT Solver make boolean decision, and sends them to the theory, whose role is in return to stop the SAT Solver as soon as the current set of assumptions is incoherent. This means that the information that theories can give the SAT Solver is pretty limited, and furthermore it limits the ability of theories to guide the proof search.

While it appears to leave a reasonably simple job to the theory, since it completely hides the propositional structure of the problem, this simple interaction between the SAT Solver and the theory makes it harder to combine multiple theories into one. Usual techniques for combining theories in SMT solvers typically require to keep track of equality congruence classes and require of theories to propagate any equality they discover.

McSat extends the SAT paradigm by allowing more exchange of information between the theory and the SAT Solver. This is achieved by allowing the solver to not only decide on the truth value of atomic propositions, but also to decide assignments for terms that appear in the problem. For instance, if the SAT Solver assumes a formula $x = 0$, an arithmetic theory could propagate to the SAT Solver that the formula $x < 1$ must also hold, instead of waiting for the SAT Solver to guess the truth value of $x < 1$ and then inform the SAT Solver that the conjunction : $x = 0 \land \neg x < 1$ is incoherent.

This exchange of information between the SAT Solver and the theories results in the construction of a model throughout the proof search (which explains the name Model Constructing SAT).

The main addition of McSat is that when the solver makes a decision, instead of being restricted to making boolean assignment of formulas, the solver now can decide to assign a value to a term belonging to one of the literals. In order to do so, the solver first chooses a term that has not yet been assigned, and then asks the theory for a possible assignment. Like in usual SMT Solvers, a McSat solver only exchange information with one theory, but, as we will see, combination of theories into one becomes easier in this framework.

Using the assignments on terms, the theory can very easily do efficient propagation of formulas implied by the current assignments: is suffices to evaluate formulas using the current partial assignment. The information flow then looks like fig 3.

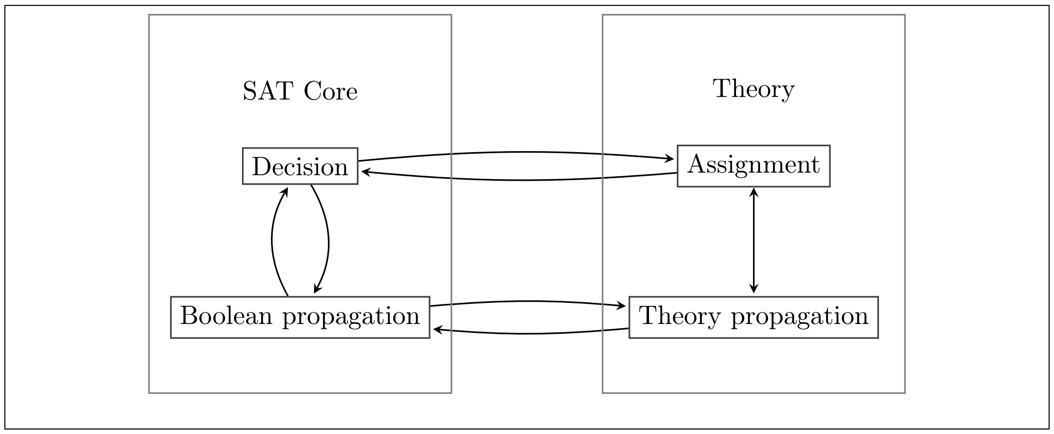

Figure 3: Simplified McSat Solver architecture

4.2 Decisions and propagations

In this document, semantic propagations are a bit different from the propagations used in traditional SMT Solvers. In the case of McSat (or at least the version presented here), semantic propagations strictly correspond to the evaluation of formulas in the current assignment. Moreover, in order to be able to correctly handle these semantic propagations during backtrack, they are assigned a level: each decision is given a level (using the same process as in a Sat solvers: a decision level is the number of decision previous to it, plus one), and a formula is propagated at the maximum level of the decisions used to evaluate it.

For instance, if the current trail is $\{x \mapsto_1 0, x + y + z = 0 \mapsto_2 \top, y \mapsto_3 0\}$, then $x + y = 0$ can be propagated at level 3, but $z = 0$ can not be propagated (at least not given to the sat solver, however nothing prevents the theory from propagating and using it internally).

4.3 First order terms

In the following, we use the following notations:

- $\mathbb{V}$ is an infinite set of variables
- $\mathbb{C}$ is a possibly infinite set of constants defined by a problem's logic[4]
- $\mathbb{S}$ is a finite set of constants defined by a problem's type definitions
- $\mathbb{T}(\mathbb{X})$ for the (infinite) set of first-order terms over $\mathbb{X}$ (for instance $a, f(0), x + y, \ldots$)
- $\mathbb{F}(\mathbb{T})$ for the (infinite) set of first order quantified formulas over the terms in $\mathbb{T}$

[4]For instance, the theory of arithmetic defines the usual operators $+, -, *, /$ as well as the constants $0, -5, \frac{1}{2}, -2.3, \ldots$

4.4 Models

A model traditionally is a triplet which comprises a domain, a signature and an interpretation function. Most problems define their signature using type definitions, and builtin theories such as arithmetic usually have canonic models, so in the following we consider the domain and siganture constant, and talk about the interpretation, more specifically, about the interpretation of symbols not defined by the theory[5], i.e non-interpreted functions. An intepretation $\mathcal{I}$ can easily be extended to a function from ground terms to model value by recursively applying it:

$$\mathcal{I}(f((e_i)_{1 \leq i \leq n})) = \mathcal{I}_f((\mathcal{I}(e_i))_{1 \leq i \leq n})$$

Before formalizing the SAT, SMT and McSat algorithms as inference rules, we need to formalize the notion of partial interpretation. Indeed, during the proof search, the McSat algorithm maintains a partial mapping from expressions to model values. The intention of this mapping is to represent a partial interpretation of the input problem, so that if the mapping is complete (i.e all variables are assigned), then it directly gives an interpretation of the input problem (quantified formula notwithstanding).

More than simply an incomplete interpretation, we also want to be able to give partial function (instead of complete functions) as interpretation of constants with positive arity. And even further, we'd like to specify these partial interpretations in a somewhat abstract way, using mappings from expressions to model values instead of a function from model values to model values. For instance, given a function symbol f of type int $\rightarrow$ int and an integer constant a, we'd like to specify that in our mapping, $f(a) \mapsto 0$, regardless of the values that f takes on other argument, and also regardless of the value mapped to a. To that end we introduce a notion of abstract partial interpretation.

An abstract partial interpretation σ is a mapping from ground expressions to model values. To each abstract partial interpretation correspond a set of complete models that realize it. More precisely, any mapping σ can be completed in various ways, leading to a set of potential interpretations:

$$\text{Complete}(\sigma) = \left\{ \mathcal{I} \mid \forall f((e_i)_{1 \leq i \leq n}) \mapsto v \in \sigma, \mathcal{I}_f((\mathcal{I}(e_i))_{1 \leq i \leq n}) = v \right\}$$

We can then consider that the natural interpretation corresponding to a given mapping is the partial interpretation on which all completion of the mapping agrees, i.e the intersection of all potential candidates:

$$\sigma_{\mathcal{I}} = \bigcap_{\mathcal{I} \in \text{Complete}(\sigma)} \mathcal{I}$$

Of course, it might happen that a mapping does not admit any potential interpretation, and thus has no natural interpretation, for instance there is no possible completion of the following mapping:

$$\sigma = \begin{cases} a \mapsto 0 \\ b \mapsto 0 \\ f(a) \mapsto 0 \\ f(b) \mapsto 1 \end{cases}$$

[5] Since theory-defined symbols, such as addition, already have an intepretation in the canonical domain.

4.5 Expected theory invariants

TODO: rewrite this section.

During proof search are maintained a set of assertions $\mathcal{S}$ and a partial assignment σ as a partial function from $\mathbb{T}(\mathbb{V}, \mathbb{C}, \mathbb{S})$ to $\mathbb{T}(\mathbb{C})$. σ is easily extended to a complete substitution function of $\mathbb{T}(\mathbb{V}, \mathbb{C}, \mathbb{S})$.

We say that σ is compatible with $\mathcal{S}$ iff for every $\varphi \in \mathcal{S}$, $\varphi\sigma$ is satisfiable (independently from the rest of the formulas in $\mathcal{S}$). Intuitively, this represents the fact that the substitution σ does not imply any ground contradictions.

A theory then has to ensure that for every expression $e \in \mathbb{T}(\mathbb{V}, \mathbb{C}, \mathbb{S})$ there exists $v \in \mathbb{T}(\mathbb{C})$ such that $\sigma' = \sigma \cup \{e \to v\}$ is coherent, and compatible with $\mathcal{S}$[6]. As soon as the current assignment is not coherent, compatible with the current assertions, or that there is a term e with no viable assignment, the theory should inform the SAT Solver to backtrack, since the current branch is clearly not satisfiable. If we reach the point where all expressions of the problem have been assigned, then we have a ground model for the current set of assertions, which is then also a model of the input problem.

Of interest if the fact that theories that respect this invariant can easily be combined to create a theory that also respect this invariant, as long as for any expression e, there is exactly one theory responsible for finding an assignment for e[7].

[6]Note that in particular, this implies that σ is coherent with $\mathcal{S}$

[7]this is similar to the usual criterion for combining SMT theories which is that they do not share symbols other than equality.

References

[1] Devan Jovanovic, Clark Barrett, and Leonardo de Moura. The design and implementation of the model constructing satisfiability calculus. 2013.

[2] Devan Jovanovic and Leonardo de Moura. A model-constructing satisfiability calculus. 2013.