mSAT & Archsat : Experimenting with McSat

Guillaume Bury

9 Feb, 2016

Deducteam, Inria; Université Paris Diderot

Introduction

- McSat: Model Constructing Sat
- Implementation as a functor : mSAT
- Instanciation with meaningful theories : Archsat

McSat

Simplified SMT control flow

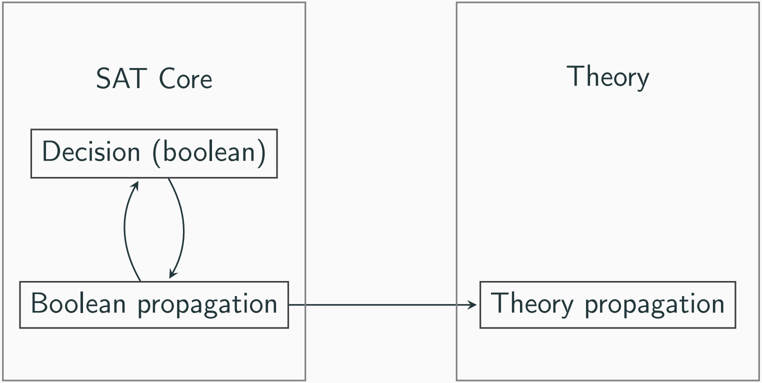

Figure 1: Simplified SMT Solver architecture

Motivation

Further integrate theory reasoning in the SAT solver

- Devan Jovanovic, Clark Barrett, and Leonardo de Moura. "The Design and Implementation of the Model Constructing Satisfiability Calculus". In: 2013
- Devan Jovanovic and Leonardo de Moura. "A Model-Constructing Satisfiability Calculus". In: 2013

McSat principle

- Decisions on propositions but also on assignment for terms
- Construction of a model that satisfies the clauses
- Exchange information between theories through assignments

Simplified McSAT control flow

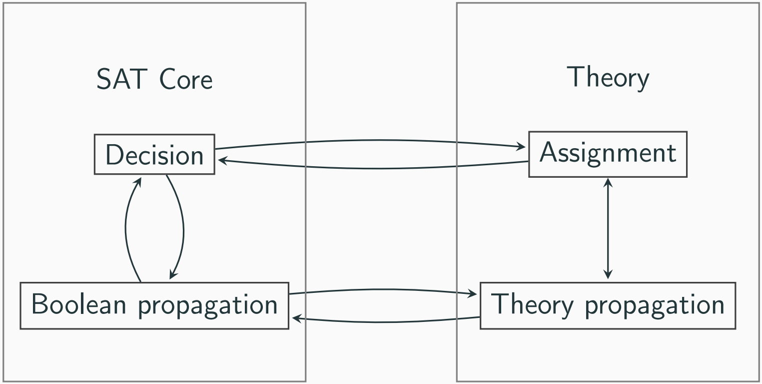

Figure 2: Simplified McSat Solver architecture

Theory invariant

Given a set of assertions $\mathcal{S}$, and a current assignment $\sigma \in \mathcal{T} \to \mathcal{T}$.

σ is coherent iff $\bigcup_{e \mapsto t \in \sigma} e = t$ is satisfiable in the theory (for instance, $\{x \mapsto 1; y \mapsto 2; x + y \mapsto 0\}$ is not coherent).

Assignments: the theory should ensure that for every sub-expression e, there should exist a term t, such that, $\sigma' = \sigma \cup \{e \mapsto t\}$ is coherent and every formula in $\mathcal{S}\sigma'$ is satisfiable (independently from the others).

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$

- $a \mapsto 0$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$

- $a \mapsto 0$
- $b \mapsto 0$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$

- $a \mapsto 0$
- $b \mapsto 0$
- $c \mapsto 0$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$

- $a \mapsto 0$
- $b \mapsto 0$
- $c \mapsto 0$
- $f(a) \mapsto 0$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$

- $a \mapsto 0$
- $b \mapsto 0$
- $c \mapsto 0$
- $f(a) \mapsto 0$
- $f(c) \mapsto 1$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$
- $\neg[a = c], [f(a) = f(c)]$

- $a \mapsto 0$
- $b \mapsto 0$
- $c \mapsto 0$
- $f(a) \mapsto 0$
- $f(c) \mapsto 1$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$
- $\neg[a = c], [f(a) = f(c)]$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$
- $\neg[a = c], [f(a) = f(c)]$
- $\neg[a = b], \neg[b = c], [a = c]$

McSat - Dealing with equality

- $[a = b]$
- $[b = c]$
- $[f(a) \neq f(c)]$
- $\neg[a = c], [f(a) = f(c)]$
- $\neg[a = b], \neg[b = c], [a = c]$

- Conflict at level 0

mSAT

mSAT: a modular SAT

- Derived from Atl-Ergo-Zero
- Very close to MiniSat
- Written in OCaml (~5k loc)
- Provides functors to make SAT/SMT/McSat solvers

Joint work with Simon Cruanès

Features

- 2-watched litterals, restarts, activity for decisions
- Push/pop operations
- Generic functors
- Proof/Model output

Interface for terms

```
module type Formula = sig
  type t (** The type of formulas *)

  val neg : t -> t (** Negation of a formula *)
  val norm : t -> t * bool
  (** Normalizes a formula, and returns if it was
      negated. *)

  val hash : t -> int
  val equal : t -> t -> bool
  val compare : t -> t -> int
  (** Usual functions *)

end
```

Interface for theories (1)

```
module type Theory = sig

  type assumption =
    | Lit of formula
    | Assign of term * term

  type slice = {
    start: int; length : int; get : int -> formula;
    push : formula list -> proof -> unit;
    propagate : formula -> int -> unit;
  }
```

Interface for theories (2)

```
  type res =
    | Sat of level
    | Unsat of formula list * proof

  val assume : slice -> res

  val assign : term -> term
end
```

Proof objects

```
type proof
and proof_node = {
  conclusion : clause;
  step : step;
}
and step =
  | Hypothesis
  | Lemma of lemma
  | Resolution of proof * proof * atom
(** Lazy type for proof trees. *)

val expand : proof -> proof_node
(** Expands a proof into a proof_node *)
```

Work to do

- Balance activity for literals and terms
- Work on conflict clauses
- Allow fine tuning of parameters
- Proof certificate output

Start using mSAT!

- Available on opam
- Source code on github (https://github.com/Gbury/mSAT)
- Used in Ziperposition, a superposition-based prover

Archsat

Archsat

- Written in OCaml (~12k loc)
- Uses the McSat functor from mSAT
- Prototype for experimenting

A plugin for each task

- Plugin examples:
  - Equality
  - Uninterpreted functions/predicates
  - Logical Connectives ($\wedge, \vee, \Rightarrow, \ldots$)
  - Quantified formulas ($\forall, \exists$)
- Each plugin is independant
- Each plugin can register options on the command line
- They can be turned on/off through the command line

Lazy CNF conversion

- Add clauses while solving
- Distinguish clausal calculus (SAT) from logic connectors $(\vee, \wedge, \Rightarrow \ldots)$

Clauses	Assumed atoms
• $\neg[(A \wedge B) \Rightarrow A]$	

Lazy CNF conversion

- Add clauses while solving
- Distinguish clausal calculus (SAT) from logic connectors $(\vee, \wedge, \Rightarrow \ldots)$

Clauses	Assumed atoms
• $\neg[(A \wedge B) \Rightarrow A]$	• $P \equiv (A \wedge B) \Rightarrow A$

Lazy CNF conversion

- Add clauses while solving
- Distinguish clausal calculus (SAT) from logic connectors $(\vee, \wedge, \Rightarrow \ldots)$

Clauses	Assumed atoms
• $\neg[(A \wedge B) \Rightarrow A]$	• $P \equiv (A \wedge B) \Rightarrow A$
• $\neg[P], [A \wedge B]$	
• $\neg[P], \neg[A]$	

Lazy CNF conversion

- Add clauses while solving
- Distinguish clausal calculus (SAT) from logic connectors $(\vee, \wedge, \Rightarrow \ldots)$

Clauses	Assumed atoms
• $\neg[(A \wedge B) \Rightarrow A]$	• $P \equiv (A \wedge B) \Rightarrow A$
• $\neg[P], [A \wedge B]$	• $Q \equiv A \wedge B$
• $\neg[P], \neg[A]$	• $\neg A$

Lazy CNF conversion

- Add clauses while solving
- Distinguish clausal calculus (SAT) from logic connectors ($\vee, \wedge, \Rightarrow \ldots$)

Clauses	Assumed atoms
• $\neg[(A \wedge B) \Rightarrow A]$	• $P \equiv (A \wedge B) \Rightarrow A$
• $\neg[P], [A \wedge B]$	• $Q \equiv A \wedge B$
• $\neg[P], \neg[A]$	• $\neg A$
• $\neg[Q], [A]$	
• $\neg[Q], [B]$	

Lazy CNF conversion

- Add clauses while solving
- Distinguish clausal calculus (SAT) from logic connectors $(\lor, \land, \Rightarrow \ldots)$

Clauses	Assumed atoms
• $\neg[(A \land B) \Rightarrow A]$	• $P \equiv (A \land B) \Rightarrow A$
• $\neg[P], [A \land B]$	• $Q \equiv A \land B$
• $\neg[P], \neg[A]$	• $\neg A$
• $\neg[Q], [A]$	• B
• $\neg[Q], [B]$	• $\rightarrow$ conflict !

Congruence closure

Equality plugin:

- Uses Union-find
- Maintains coherence of assignments with regards to equality

Uninterpreted function plugin:

- Maintains coherence of assignmentw with regards to semantics of functions, i.e that if $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the same assignments, then $f(x_1, \ldots, x_n)$ and $f(y_1, \ldots, y_n)$ also have the same assignment.

Instanciation

- Introduce meta-variables for universally quantified variables
- If a model is found:
    - Try and unify true predicates with false predicates
    - Start the search again
- If Unsat, then problem solved

Isntanciation - example

- $[\forall x, p(x)]$
- $\neg[p(a)]$

Isntanciation - example

- $[\forall x, p(x)]$
- $\neg[p(a)]$

- $p(a) \mapsto \bot$

Isntanciation - example

- $[\forall x, p(x)]$
- $\neg[p(a)]$
- $\neg[\forall x, p(x)], [p(X)]$

- $p(a) \mapsto \bot$

Isntanciation - example

- $[\forall x, p(x)]$
- $\neg[p(a)]$
- $\neg[\forall x, p(x)], [p(X)]$

- $p(a) \mapsto \bot$
- $p(X) \mapsto \top$

Isntanciation - example

- $[\forall x, p(x)]$
- $\neg[p(a)]$
- $\neg[\forall x, p(x)], [p(X)]$
- $\neg[\forall x, p(x)], [p(a)]$

- Conflict !

Finding instanciations

Different unification algorithms:

- Robinson unification
- Rigid E-unification
- Superposition with atomic clauses

Further work

- Other instanciation strategies
- New theories (linear arithmetic, algebraic datatypes, . . .)
- Outputs proof certificates (dedukti, coq)