Archsat: SMT, Tableaux and beyond

Guillaume Bury

June 26, 2016

Université Paris Diderot; Inria; ENS Cachan

Introduction

- Archsat:
  - Prototype prover
  - SMT Core using Mcsat
  - Proof output
- Mcsat
  - For first-order logic (McSat), not propositional logic
  - Implemented in a separate library: mSAT

McSat

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$
- Propagation in $\neg p(a) \vee p(b)$: $p(b) \rightsquigarrow \top$

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$
- Propagation in $\neg p(a) \vee p(b)$: $p(b) \rightsquigarrow \top$
- Conflict: $\neg p(a) \vee \neg p(b)$ not satisfied

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$
- Propagation in $\neg p(a) \vee p(b)$: $p(b) \rightsquigarrow \top$
- Conflict: $\neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $\neg p(a) \vee \neg p(b)$ and $\neg p(a) \vee p(b)$

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$
- Propagation in $\neg p(a) \vee p(b)$: $p(b) \rightsquigarrow \top$
- Conflict: $\neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $\neg p(a) \vee \neg p(b)$ and $\neg p(a) \vee p(b)$
- New clause : $\neg p(a)$, backtrack to before decision.

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$
- Propagation in $\neg p(a) \vee p(b)$: $p(b) \rightsquigarrow \top$
- Conflict: $\neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $\neg p(a) \vee \neg p(b)$ and $\neg p(a) \vee p(b)$
- New clause : $\neg p(a)$, backtrack to before decision.
- Propagation: $p(a) \rightsquigarrow \bot$

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$
- Propagation in $\neg p(a) \vee p(b)$: $p(b) \rightsquigarrow \top$
- Conflict: $\neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $\neg p(a) \vee \neg p(b)$ and $\neg p(a) \vee p(b)$
- New clause : $\neg p(a)$, backtrack to before decision.
- Propagation: $p(a) \rightsquigarrow \bot$
- Decision: $p(b) \mapsto \top$

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$
- Propagation in $\neg p(a) \vee p(b)$: $p(b) \rightsquigarrow \top$
- Conflict: $\neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $\neg p(a) \vee \neg p(b)$ and $\neg p(a) \vee p(b)$
- New clause : $\neg p(a)$, backtrack to before decision.
- Propagation: $p(a) \rightsquigarrow \bot$
- Decision: $p(b) \mapsto \top$
- Propagation (nothing to do)

SAT Solving

- $\neg p(a) \vee p(b), \neg p(a) \vee \neg p(b)$
- Problem: find a model
- Decision: $p(a) \mapsto \top$
- Propagation in $\neg p(a) \vee p(b)$: $p(b) \rightsquigarrow \top$
- Conflict: $\neg p(a) \vee \neg p(b)$ not satisfied
- Resolution between $\neg p(a) \vee \neg p(b)$ and $\neg p(a) \vee p(b)$
- New clause : $\neg p(a)$, backtrack to before decision.
- Propagation: $p(a) \rightsquigarrow \bot$
- Decision: $p(b) \mapsto \top$
- Propagation (nothing to do)
- Model Found !

Simplified control flow

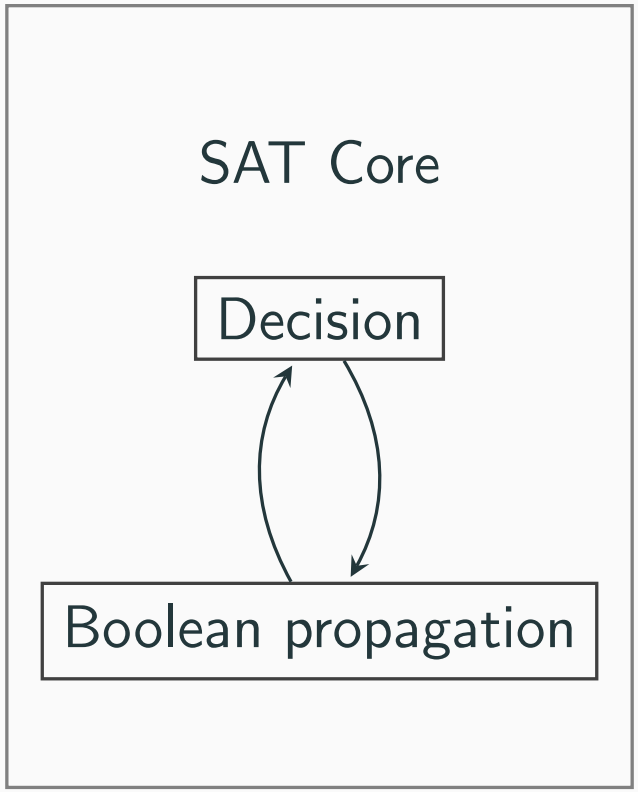

Figure 1: Simplified SMT Solver architecture

Simplified control flow

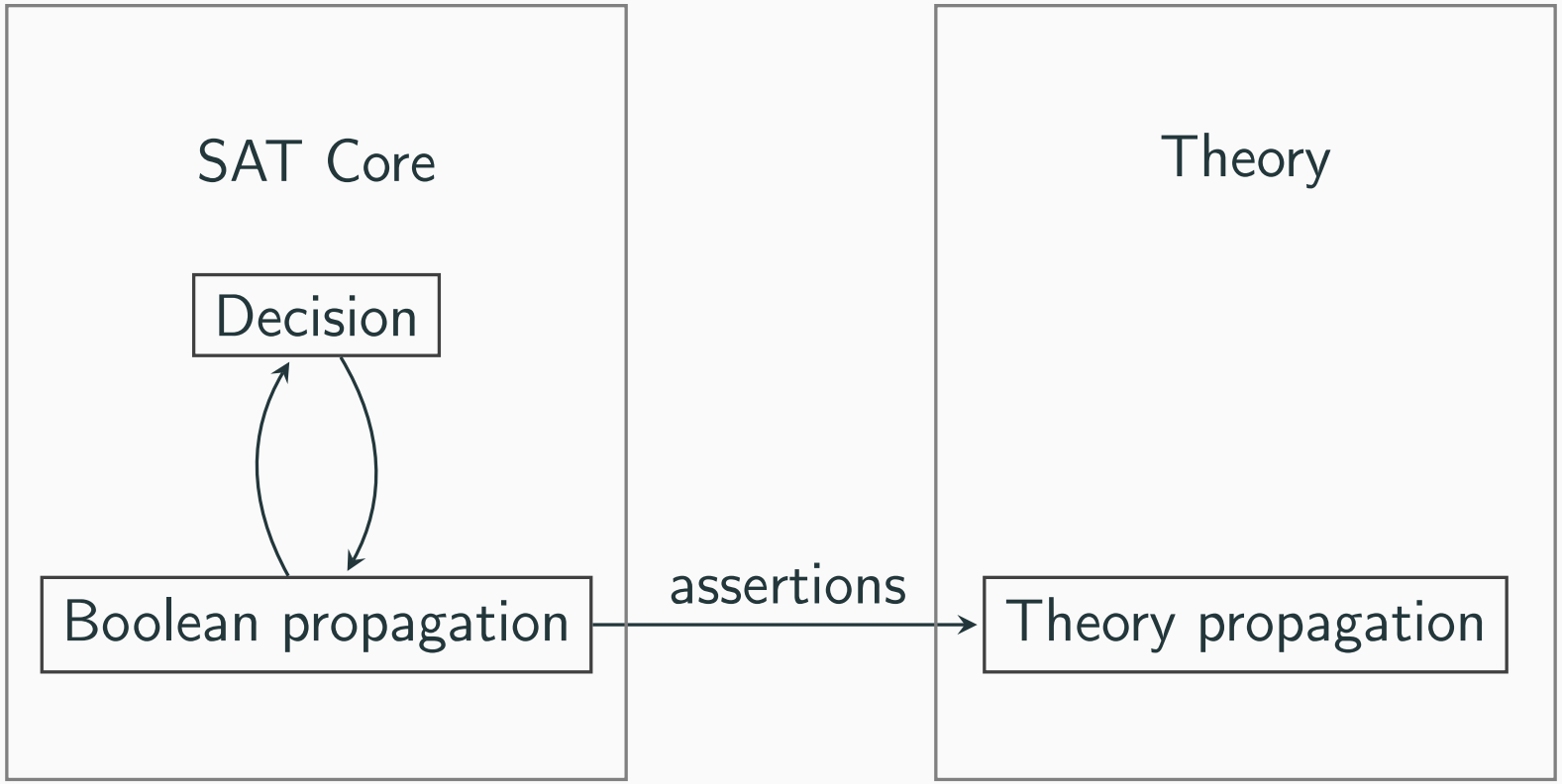

Figure 1: Simplified SMT Solver architecture

Motivation

Further integrate theory reasoning in the SAT solver

- **FMCAD13**
- **VMCAI13**

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

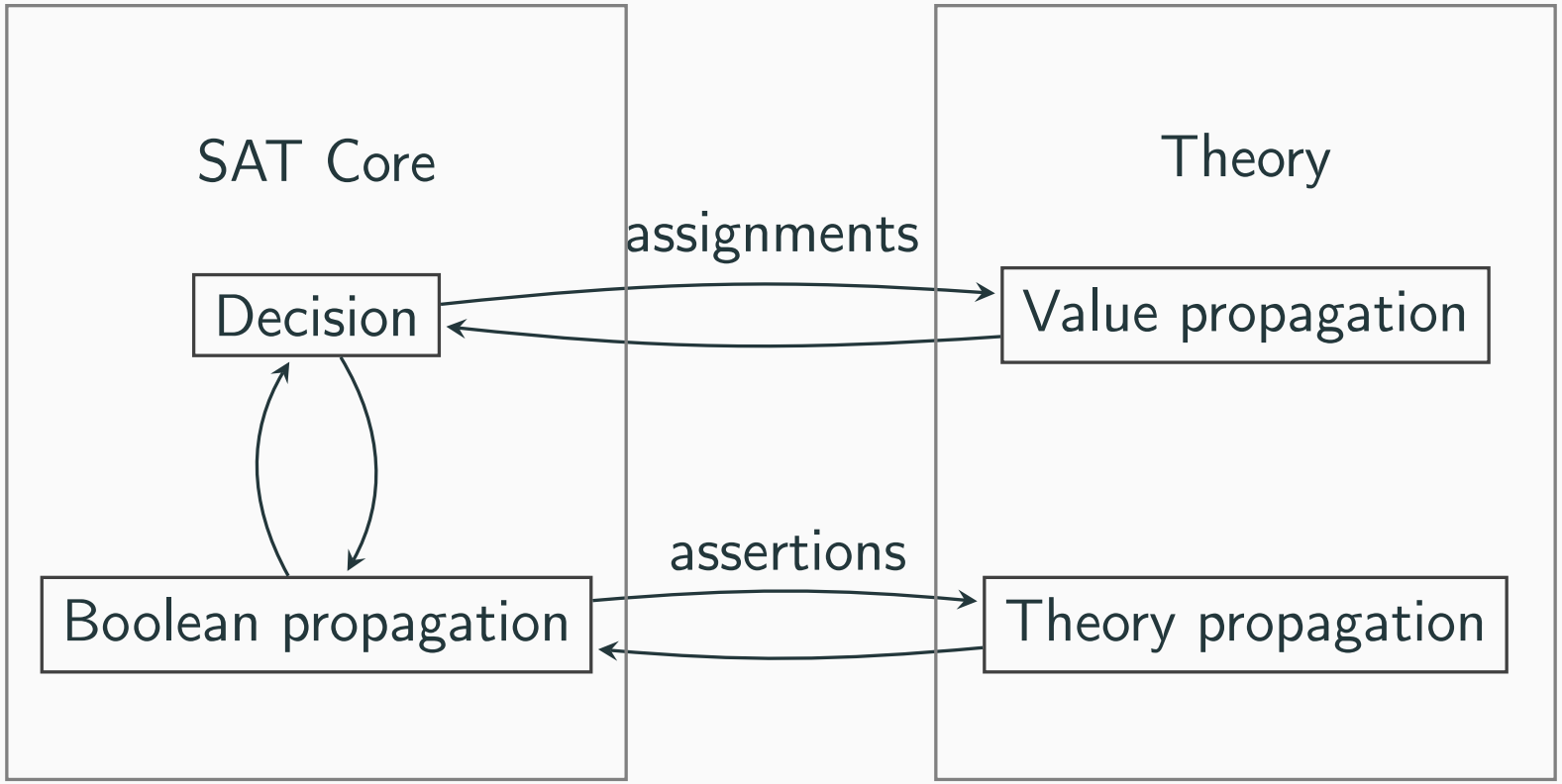

Figure 2: Simplified McSat Solver architecture

Theory invariant

- $\mathcal{S}$: set of assertions
- σ : assignment

σ consistent iff $\bigcup_{e \mapsto v \in \sigma} e = v$ is satisfiable in the theory.

For instance, $\{x \mapsto 1; y \mapsto 2; x + y \mapsto 0\}$ not consistent.

$\mathcal{S}$ compatible with σ: for every expression e, a value v s.t. $\sigma' = \sigma \cup \{e \mapsto v\}$ consistent and every formula in $\mathcal{S}\sigma'$ individually satisfiable

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $a \mapsto_1 0$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $a \mapsto_1 0$
- $b \mapsto_2 0$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto_3 1$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto_3 1$
- $[b = d] \rightsquigarrow_3 \bot$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto_3 1$
- $[b = d] \rightsquigarrow_3 \bot$
- $[c = d] \rightsquigarrow_{([b=d]\vee[c=d])} \top$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto_3 1$
- $[b = d] \rightsquigarrow_3 \bot$
- $[c = d] \rightsquigarrow_{([b=d]\vee[c=d])} \top$
- $c \mapsto ?$

McSat - Example

Conflict : $\neg[a = c] \vee \neg[c = d] \vee [a = d]$

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto_3 1$
- $[b = d] \rightsquigarrow_3 \bot$
- $[c = d] \rightsquigarrow_{([b=d] \vee [c=d])} \top$
- $c \mapsto ?$

McSat - Example

Backtrack : $\neg[a = c] \vee [b = d] \vee [a = d]$

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto_3 1$
- $[b = d] \rightsquigarrow_3 \bot$
- $[c = d] \rightsquigarrow_{([b=d] \vee [c=d])} \top$
- $c \mapsto ?$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$
- $\neg[a = c] \vee [b = d] \vee [a = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $[b = d] \rightsquigarrow_0 \top$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$
- $\neg[a = c] \vee [b = d] \vee [a = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $[b = d] \rightsquigarrow_0 \top$
- $a \mapsto_1 0$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$
- $\neg[a = c] \vee [b = d] \vee [a = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $[b = d] \rightsquigarrow_0 \top$
- $a \mapsto_1 0$
- $b \mapsto_2 0$

McSat - Example

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$
- $\neg[a = c] \vee [b = d] \vee [a = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $[b = d] \rightsquigarrow_0 \top$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto ?$

McSat - Example

Conflict : $\neg[a = b] \vee \neg[b = d] \vee [a = d]$

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$
- $\neg[a = c] \vee [b = d] \vee [a = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $[b = d] \rightsquigarrow_0 \top$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto ?$

McSat - Example

Backtrack : $\bot$

- $[a = b]$
- $[a = c]$
- $\neg[a = d]$
- $[b = d] \vee [c = d]$
- $\neg[a = c] \vee [b = d] \vee [a = d]$

- $[a = b] \rightsquigarrow_0 \top$
- $[a = c] \rightsquigarrow_0 \top$
- $[a = d] \rightsquigarrow_0 \bot$
- $[b = d] \rightsquigarrow_0 \top$
- $a \mapsto_1 0$
- $b \mapsto_2 0$
- $d \mapsto ?$

mSAT

mSAT: a modular SAT

- Core of Archsat
- Implements McSat
- Derived from Alt-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 literals, restarts, activity for decisions
- Push/pop operations
- Generic functors
- Proof/Model output

Functor solver

```
module Make
  (F  : Formula_intf.S)
  (Th : Theory_intf.S
        with type formula = F.t
         and type proof = F.proof) : S
      with type St.formula = F.t
       and type St.proof = F.proof
  (** Functor to create a SMT Solver parametrised by the atomic
      formulas and a theory. *)
```

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 on mSAT

- Balance activity for literals and terms
- Work on conflict clauses
- Allow fine tuning of parameters
- Proof certificate output (Dedukti, Coq)

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 independent

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

Lazy CNF conversion

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

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

Lazy CNF conversion

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

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

Lazy CNF conversion

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

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

Lazy CNF conversion

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

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

Instantiation

- 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

Instantiation - example

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

Instantiation - example

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

- $p(a) \mapsto \bot$

Instantiation - example

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

- $p(a) \mapsto \bot$

Instantiation - example

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

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

Instantiation - example

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

- Conflict !

Finding instantiations

Different unification algorithms:

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

Further work

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