Verified Efficient Clausal Proof Checking for SAT

Filip Marić, Faculty of Mathematics, Belgrade
(joint work with Florian Haftmann, TU Munich)

SVARM Workshop,
2. 4. 2011.

Overview

SAT solvers

- Decision procedures for satisfiability in propositional logic.
- Huge progress in last two decades.
- SAT solvers are efficient enough for many practical applications:
  - Hardware and software verification.
  - Solving combinatorial problems.
  - Solving optimization problems.
  - ...

Trust in SAT solvers results

- Critical areas of application (e.g. hardware and software verification).
- Solvers must be trusted.
- Two approaches:
    1. Verify SAT solvers (Lescuyer and Conchon, Marić, . . .);
    2. Generate and check certificates for each formula (Zhang, Goldberg and Novikov, Van Gelder, Biere, . . .).

Verification of SAT solvers

Formalization and verification of SAT solvers.

Advantages:

- No need for considering each specific instance.
- Helps better understanding SAT solving algorithms.

Drawbacks:

- Extremely complicated task.
- Many implementation details make the task even harder.
- Formalization and verification must be updated each time the SAT solver implementation changes.

Checking certificates

For each instance, a certificate is generated and checked by independent tools.

- Models for satisfiable formulae — trivially generated and checked.
- Proofs for unsatisfiable formulae — not so easy to generate and efficiently check.

Checking certificates

Advantages:

- Simpler to implement then verifying SAT solvers.
- No big changes are needed when SAT solvers are changed.

Drawbacks:

- SAT solvers must be modified.
- Time overhead for generating and checking proofs.
- Huge storage and memory requirements for storing and checking proofs (measured in GB for industrial instances).

Overview

1. Introduction
2. Unsatisfiability proof formats for SAT
3. Verified efficient checking of clausal proofs

Unsatisfiability proof formats

1. Resolution proofs (Zhang et al., Chaff)
    - Full resolution proofs
    - Resolution proof traces (compact)
    - RES, RPT (Van Gelder — SATComp)
2. Clausal proofs (Godberg i Novikov, Berkmin)
    - RUP (Van Gelder — SATComp)

Full resolution proofs

A series of resolution steps deriving the empty clause from the initial clauses.

Example

$$(c \vee e \vee a) \wedge (c \vee e \vee \overline{a}) \wedge (d \vee \overline{c} \vee e) \wedge (\overline{d} \vee \overline{c} \vee e) \wedge (\overline{b} \vee \overline{e}) \wedge (b \vee \overline{e})$$

Proof

$$\begin{array}{lll}
c \vee e \vee a & c \vee e \vee \overline{a} & c \vee e \\
d \vee \overline{c} \vee e & \overline{d} \vee \overline{c} \vee e & \overline{c} \vee e \\
c \vee e & \overline{c} \vee e & e \\
\overline{b} \vee \overline{e} & b \vee \overline{e} & \overline{e} \\
e & \overline{e} & \perp
\end{array}$$

Full resolution proofs

Advantages:

- Trivial to implement a checker.

Drawbacks

- Not trivial to modify SAT solvers to generate resolution proofs.
- Huge objects (several GB) — cannot always fit in main memory during checking!
- Checking time can be significant.

Resolution proof traces

A series of chains of input resolutions.

Example

$$\begin{array}{rcl} 1 & : & c \vee e \vee a \\ 2 & : & c \vee e \vee \overline{a} \\ 3 & : & d \vee \overline{c} \vee e \\ 4 & : & \overline{d} \vee \overline{c} \vee e \\ 5 & : & \overline{b} \vee \overline{e} \\ 6 & : & b \vee \overline{e} \end{array}$$

Proof

$$\begin{array}{rcll} 7 & : & e \vee a & 3, 4, 1 \\ 8 & : & \overline{e} & 5, 6 \\ 9 & : & & 4, 3, 2, 7, 8 \end{array}$$

Resolution proof traces

Advantages:

- Most widely adopted proof format for SAT.
- Proofs smaller then full resolution proofs (but still can be large).

Drawbacks

- More complicated checker then for full resolution proofs — in SAT competitions, proofs traces are first converted to full resolution proofs.
- Not so trivial to modify SAT solvers to generate resolution proofs.
- Checking time can be significant.

Clausal proofs

A sequence of clauses learned during SAT solving.

Example

$$(c \vee e \vee a) \wedge (c \vee e \vee \overline{a}) \wedge (d \vee \overline{c} \vee e) \wedge (\overline{d} \vee \overline{c} \vee e) \wedge (\overline{b} \vee \overline{e}) \wedge (b \vee \overline{e})$$

Proof

$$e \vee a$$
$$\overline{e}$$

How to check clausal proofs?

Let F be an unsatisfiable formula and $C_1, C_2, \ldots, C_k$ a series of clauses learnt derived during solving F. It suffices to show that

$$\begin{array}{rcl} F & \models & C_1, \\ F, C_1 & \models & C_2 \\ \ldots & & \\ F, C_1, \ldots, C_{k-1} & \models & C_k \\ F, C_1, \ldots, C_k & \models & \bot \end{array} \qquad \begin{array}{rcl} F, \overline{C_1} & \vdash & \bot, \\ F, C_1, \overline{C_2} & \vdash & \bot \\ \ldots & & \\ F, C_1, \ldots, C_{k-1}, \overline{C_k} & \vdash & \bot \\ F, C_1, \ldots, C_k & \vdash & \bot \end{array}$$

Trivial (input) resolution

- Checking $F, C_1, \ldots, C_{i-1}, \overline{C_i}$ for unsatisfiability is a new SAT instance and does not seem much easier then checking unsatisfiability of F!
- However, clause C_i is derived from $F, C_1, \ldots, C_{i-1}$ by trivial resolution, then the new SAT instance is easy (can be solved without search).
- Most SAT solvers derive clauses by using trivial resolution (during conflict analysis phase).

Trivial (input) resolution

Sequence $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}$ is a trivial resolution of a clause $\mathcal{C}$ from $\mathcal{F}$ iff each clause $\mathcal{C}_i$ is:

1. either an initial clause (i.e., $\mathcal{C}_i \in \mathcal{F}$) or
2. a resolvent of $\mathcal{C}_{i-1}$ and an initial clause c (i.e., $\mathcal{C}_i = \mathcal{C}_{i-1} \oplus_x c$ and $c \in \mathcal{F}$),

and each variable x is resolved only once.

Theorem

If $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}$, is trivial and $\mathcal{C} \notin \mathcal{F}$ then unsatisfiability of $\mathcal{C}_1, \mathcal{C}_2, \ldots, \overline{\mathcal{C}}$ can be shown by using only unit propagation.

Clausal proofs

Advantages:

- Can be significantly smaller than resolution proofs.
- It is trivial to modify SAT solvers to generate them.
- Proof generation overhead smaller compared to resolution proofs.

Drawbacks:

- Complicated to check — sophisticated algorithms and data structures must be used for efficient checking.
- If the solver that checks them is complex, how can it be trusted?
- For the given reasons, clausal proofs are not widely accepted in the SAT community.

Using clausal proofs

- RUP2RES — Van Gelder 2008.
- Clausal proofs are translated to resolution proofs and then checked.
- Translation need not be trusted because the RES proofs is independently checked.

Advantages:

- No need for complicated modifications of SAT solvers to generate proofs.

Drawbacks:

- Time needed to translated RUP to RES can be significant.
- After translation, resolution proofs are still huge.
- Checking time can be significant.

Overview

Present work

- Clausal proof checkers use data structures and algorithms used in modern SAT solvers (e.g. *two-watch literal scheme*).
- Formalization and verification of these has already been done within Isabelle/HOL (Marić, Ph.D. thesis).
- Reuse previous work for implementing formally verified proof checker for clausal proofs.

Problems

How to achieve the desired efficiency?

- Efficiency requires using imperative (mutable) data structures.
- Isabelle/HOL is purely functional.
- Imperative/HOL package enables using imperative data structure within Isabelle.
- From the Imperative/HOL specifications, it is possible to automatically extract executable code in SML or Haskell which uses imperative data structures and achieves high level of efficiency.

Preliminary experimental results

- Comparison to seminal work on clausal proofs (Goldber, Novikov, 2003.)
- Their benchmarks are still available, but proofs are not.
- To variants of our checker:
  - Automatically exported SML checker;
  - Checker manually implemented in C++, directly following verified specification.

Benchmark			Goldberg & Novikov (2003. 500MHz)			Marić & Haftmann (2010. 1.8GHz)			
name	vars	cls.	c. cls.	c. lits. ($\cdot 10^3$)	C++ (s)	c. cls.	c. lits. ($\cdot 10^3$)	SML (s)	C++ (s)
w10_45	16,931	51,803	4,285	89	20.5	3,017	100	10.7	4.6
w10_60	26,611	83,538	14,489	440	104.4	7,703	568	49.7	20.7
w10_70	32,745	103,556	32,847	1,303	354.6	15,451	1,637	142.2	61.4
c5315	5,399	15,024	16,132	416	7.0	18,006	609	14.9	4.8
c7552	7,652	20,423	22,307	726	17.3	32,560	2,153	64.6	21.3

Conclusions

- Clausal proofs are easy to produce, compactly represented unsatisfiability proofs for SAT.
- Checking clausal proofs consumes significantly less memory then other types of proofs.
- Clausal proof checking can be parallelized.
- Checking clausal proofs requires efficient BCP (nontrivial to implement and cannot be trusted by code inspection).
- We have built a formally verified proof checker for clausal proofs with encouraging experimental results.

Thank you

Thank you four your attention!

Trivial resolution

Proof: Suppose that in $C_1, C_2, \ldots, C$ all initial clauses precede resolvents. Let M be a valuation $\overline{C}$. The proof is by induction on the number of resolvents.

Let $C = C_k \oplus c$, for a $c \in F$. Let $C_k = A \vee \neg x$ and $c = B \vee x$. It holds that $C = A \vee B$. Since $M \vDash \neg\, C$, it holds that $M \vDash \neg\, A$ and $M \vDash \neg\, B$.

1. If C is the only resolvent, then $C_k \in F$. Therefore $M \vdash_{up_F} x$, and $M \vdash_{up_F} \neg x$, so $M \vdash_{up_F} \bot$.
2. If there are more reslovents, then $C_k \notin F$. Then the inductive hypothesis hold for C_k and $M, x \vdash_{up_f} \bot$. Since $c \in F$ it holds $M \vdash_{up_f} x$, so $M \vdash_{up_f} \bot$.