Compositional Propositional Proofs

Marijn J.H. Heule

THE UNIVERSITY OF TEXAS AT AUSTIN

Joint work with
Armin Biere

JOHANNES KEPLER UNIVERSITY LINZ

LPAR-20, November 25, 2015

Motivation

Satisfiability solvers are used in amazing ways...

- Hardware verification: Centaur x86 verification
- Combinatorial problems:
  - Ramsey numbers and van der Waerden numbers
    [Dransfield, Marek, and Truszczynski, 2004; Kouril and Paul, 2008]
  - Gardens of Eden in Conway's Game of Life
    [Hartman, Heule, Kwekkeboom, and Noels, 2013]
  - Erdős Discrepancy Problem [Konev and Lisitsa, 2014]

..., but satisfiability solvers have errors.

- Documented bugs in SAT, SMT, and QBF solvers;
  [Brummayer and Biere, 2009; Brummayer et al., 2010]
- Implementation errors often imply conceptual errors;
- Problem partitioning can introduce errors as well;
- Compute proofs to validate the results of solvers.

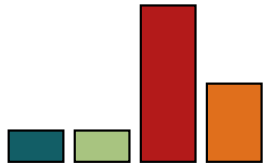
From Resolution to Clausal DRAT Proofs

Easy to Emit

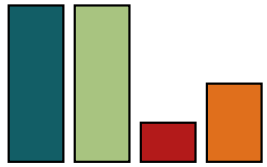
Compactness

Checked Efficiently

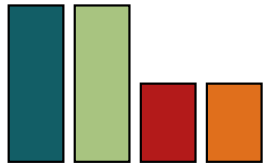
Expressiveness



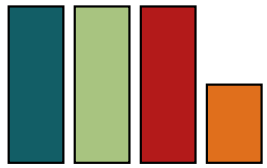
Resolution Proofs
Zhang and Malik, 2003
Van Gelder, 2008; Biere, 2008



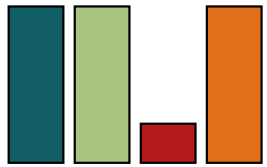
Clausal Proofs
Goldberg and Novikov, 2003
Van Gelder, 2008



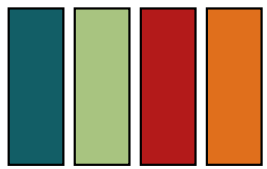
Clausal proofs + clause deletion
Heule, Hunt, Jr., and Wetzler [STVR 2014]



Optimized clausal proof checker
Heule, Hunt, Jr., and Wetzler [FMCAD 2013]



Clausal RAT proofs
Heule, Hunt, Jr., and Wetzler [CADE 2013]



DRAT proofs (RAT + deletion)
Wetzler, Heule, and Hunt, Jr. [SAT 2014]

All problems related to propositional proofs solved? No!

Many challenges still exist. For example,

Proofs are still too large for some applications [IWIL 2015]:

- Clausal proof logging has been mandatory during the SAT 2013 and 2014 competitions. Some proofs could not be validated because they were larger than the 100 GB limit;
- Proofs for some hard-combinatorial problems are huge.

No reasonably fast mechanically-verified clausal proof checker.

- First steps realized, lots of work required. [Wetzler 2015]

No parallel proof checker nor proof logging of parallel solvers:

- Merging proofs from multiple solvers is not trivial;
- Naive parallel proof checking can result in slow validation;
- Our LPAR 2015 paper shows how to deal with both.

Clausal Proofs

Resolution Asymmetric Tautology (RAT) [IJCAR 2012]

Given a clause $C = (l_1 \vee \cdots \vee l_k)$ and a CNF formula F:

- $\overline{C}$ denotes the conjunction of its negated literals $(\bar{l}_1) \wedge \cdots \wedge (\bar{l}_k)$
- $F \vdash_1 \epsilon$ denotes that unit propagation on F derives a conflict
- C is an asymmetric tautology w.r.t. F if and only if $F \wedge \overline{C} \vdash_1 \epsilon$
- C is a resolution asymmetric tautology on $l \in C$ w.r.t. F iff for all resolvents $C \diamond D$ with $D \in F$ and $\bar{l} \in D$ holds that $F \wedge \overline{C \diamond D} \vdash_1 \epsilon$

Example

Consider the formula $F = (a \vee c) \wedge (\bar{b} \vee \bar{c}) \wedge (b \vee d)$:

- The clause $(a \vee d)$ is an asymmetric tautology w.r.t. F
- The clause $(b \vee c)$ is a resolution asymmetric tautology w.r.t. F on b

Theorem: Given a formula F and a clause C having RAT with respect to F, then F and $F \cup \{C\}$ are equi-satisfiable.

Clausal Proof System using RAT addition and deletion

Learn: add a clause
* Clause C has RAT w.r.t. F

init

F

Unsatisfiable
* Learn empty clause

Satisfiable
* Forget last clause

Forget: remove a clause
* Clause C has RAT w.r.t. $F \setminus \{C\}$

Composition Rules

Base Rules

Given a formula $\bigcirc_i$ (multi-set), a clause C and a modification $m \in \{a, d\}$, a proof step is denoted as $\bigcirc_i \xrightarrow{m(C)} \bigcirc_{i+1}$.

ADD: $\dfrac{}{\bigcirc \xrightarrow{a(C)} \bigcirc\, C}$ where C has RAT on $l \in C$ w.r.t. $\bigcirc$

DEL: $\dfrac{}{\bigcirc\, C \xrightarrow{d(C)} \bigcirc}$ (no side condition)

DEL has no side condition for refutations (unsatisfiability).
For satisfiability proofs, DEL has the ADD side condition.

Consider the proof

$$\bigcirc_0 \xrightarrow{m_1(C_1)} \bigcirc_1 \xrightarrow{m_2(C_2)} \bigcirc_2 \ldots \bigcirc_{n-1} \xrightarrow{m_n(C_n)} \bigcirc_n$$

$\triangle = m_1(C_1)m_2(C_2)\cdots m_n(C_n)$ gives a derivation from $\bigcirc_0$ to $\bigcirc_n$.

Compositional Triples

We represent rules using compositional triples: $(\bigcirc_{\text{pre}}, \triangle, \bigcirc_{\text{post}})$.

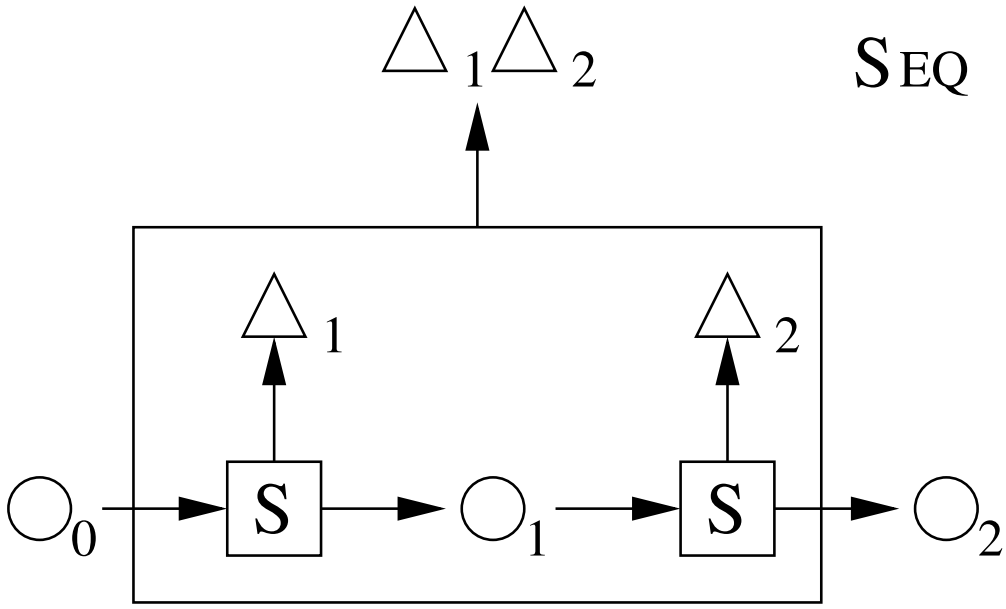
Triples consists of a pre-CNF $\bigcirc_{\text{pre}}$, a proof $\triangle$, and a post-CNF $\bigcirc_{\text{post}}$, denoting that proof $\triangle$ is a derivation from $\bigcirc_{\text{pre}}$ to $\bigcirc_{\text{post}}$.

Triple $(\bigcirc_{\text{pre}}, \triangle, \bigcirc_{\text{post}})$ is valid if and only if $\bigcirc_{\text{pre}} \xrightarrow{\triangle} \bigcirc_{\text{post}}$.

Proposition: Given a valid composition triple $(\bigcirc_{\text{pre}}, \triangle, \bigcirc_{\text{post}})$, if $\bigcirc_{\text{pre}}$ is satisfiable then $\bigcirc_{\text{post}}$ is satisfiable as well.

We focus on the contrapositive, e.g., if $\bigcirc_{\text{post}}$ contains the empty clause then $\bigcirc_{\text{pre}}$ is unsatisfiable.

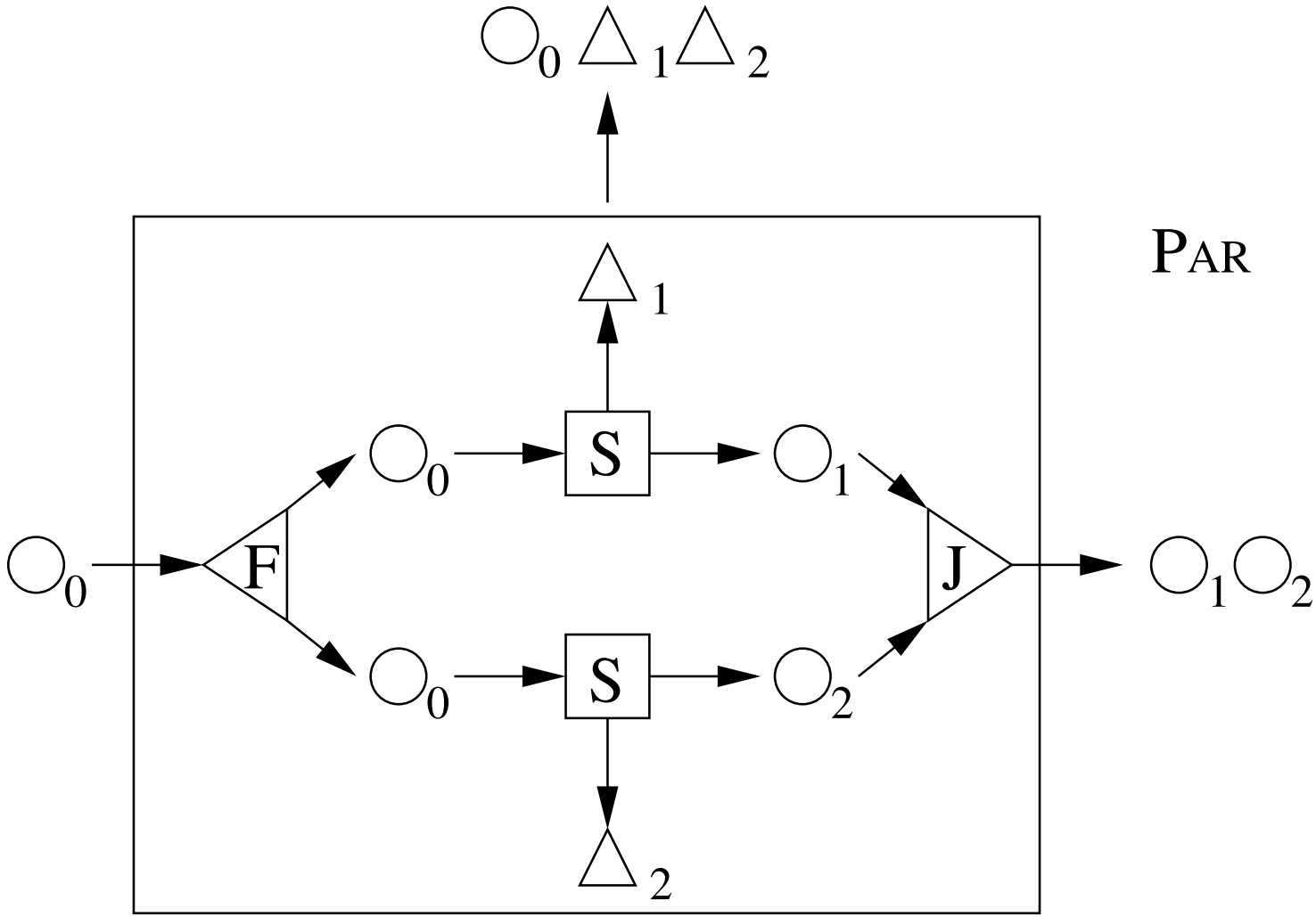
SEQ Rule



S: solve

No restrictions: both $\triangle_1$ and $\triangle_2$ may contain RAT clauses.

PAR Rule



F: fork S: solve J: join

Restricted: $\triangle_1$ and $\triangle_2$ can only have asymmetric tautologies.

Composition Rules

We propose two composition rules which combine two compositional triples into one.

The first rule SEQ, short for "sequential", combines two compositional triples for which the post-CNF of one triple equals the pre-CNF of the other triple.

$$\frac{\bigcirc_0 \xrightarrow{\triangle_1} \bigcirc_1 \quad \bigcirc_1 \xrightarrow{\triangle_2} \bigcirc_2}{\bigcirc_0 \xrightarrow{\triangle_1\triangle_2} \bigcirc_2}$$

The second rule PAR, short for "parallel", combines two compositional triples for which the two pre-CNFs are equal.

$$\frac{\bigcirc_0 \xrightarrow{\triangle_1} \bigcirc_1 \quad \bigcirc_0 \xrightarrow{\triangle_2} \bigcirc_2}{\bigcirc_0 \xrightarrow{\bigcirc_0\triangle_1\triangle_2} \bigcirc_1\bigcirc_2}$$

Parallel Proof Checking

Parallel Proof Checking using SEQ Rule

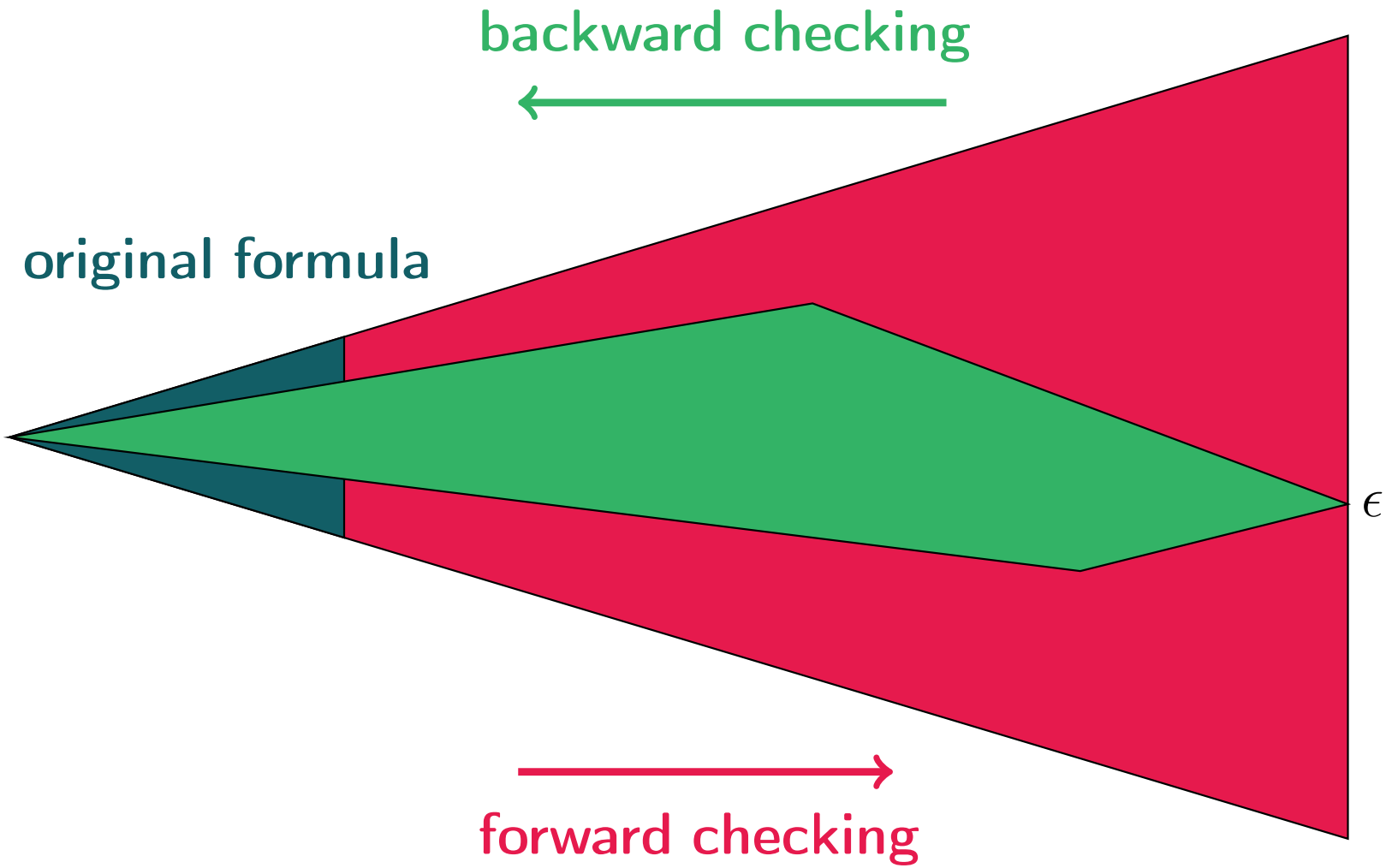
Given a refutation $\triangle$ for formula $\bigcirc_0$, apply the following steps:

- partition $\triangle$ into k subproofs $\triangle_1, \ldots, \triangle_k$ (sequential). Simply use Unix' `split`.
- compute the post-formulas $\bigcirc_i$ by applying subproof $\triangle_i$ to formula $\bigcirc_{i-1}$ (sequential).
- check that all $\bigcirc_i \xrightarrow{\triangle_{i+1}} \bigcirc_{i+1}$ are valid derivations for $i \in \{0..k-1\}$ with $\bigcirc_k = \epsilon$ (parallel).

Costs of first two (sequential) steps are relatively small.

$$\frac{\bigcirc_0 \xrightarrow{\triangle_1} \bigcirc_1 \quad \bigcirc_1 \xrightarrow{\triangle_2} \bigcirc_2 \quad \ldots \quad \bigcirc_{k-1} \xrightarrow{\triangle_k} \epsilon}{\bigcirc_0 \xrightarrow{\triangle_1 \triangle_2 \ldots \triangle_k} \epsilon}$$

Forward vs Backward Proof Checking (1)



Forward vs Backward Proof Checking (2)

Forward Checking checks each addition step in a derivation.

Backward Checking initializes by marking the empty clause. Afterwards the proof is checked is reverse order. Only marked clauses are checked, which will mark clauses using conflict analysis. Many addition steps may be skipped (up to 99%).

How to perform backward checking subproofs without ϵ?

- Initialize marking only clauses that added, but not deleted;
- Unmark clauses that are subsumed by a marked clause;
- Proceed as usual by checking the proof in reverse order.

Backward checking generalization: empty clause subsumes all.

For subproofs: many addition steps can be skipped, although not as many as with refutations.

Parallel Proof Generation for Cube-and-Conquer

Cube-and-Conquer is a powerful parallel SAT-solving paradigm which realizes linear speed-ups on many hard-combinatorial problems — even when using thousands of cores. [HVC 2012]

Cube-and-Conquer consists of two phases: First, the input is partitioned using a look-ahead solver. The resulting cubes are afterwards solved with CDCL solvers.

How to construct proofs for a Cube-and-Conquer solver?

$$\dfrac{\dfrac{\bigcirc_0 \xrightarrow{\triangle_1} \bigcirc_1 \quad \bigcirc_0 \xrightarrow{\triangle_2} \bigcirc_2 \quad \ldots \quad \bigcirc_0 \xrightarrow{\triangle_k} \bigcirc_k}{\bigcirc_0 \xrightarrow{\bigcirc_0 \ldots \bigcirc_0 \triangle_1 \triangle_2 \ldots \triangle_k} \bigcirc_1 \bigcirc_2 \ldots \bigcirc_k} \qquad \bigcirc_1 \bigcirc_2 \ldots \bigcirc_k \xrightarrow{\triangle_c} \epsilon}{\bigcirc_0 \xrightarrow{\bigcirc_0 \ldots \bigcirc_0 \triangle_1 \triangle_2 \ldots \triangle_k \triangle_c} \epsilon}$$

Tools and Evaluation

Tools

New proof checker `Drabt`: Armin Biere implemented a clausal proof + derivation checker to validate results using two tools.

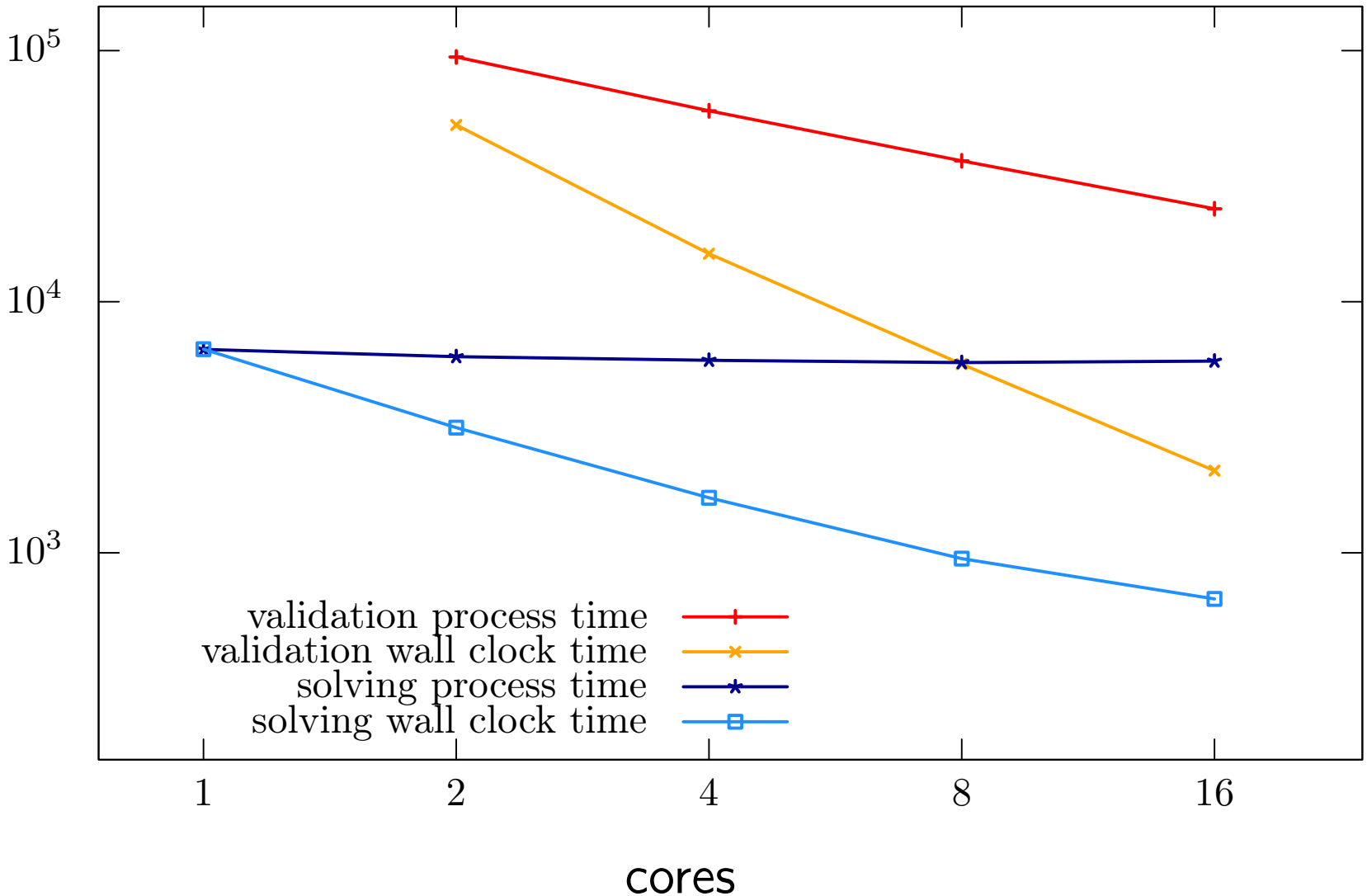
Proof checker `DRAT-trim`: Added support to validate derivations backwards and to unmark using subsumption.

Cube solver `march_cc`: The original version only emitted cubes that could not be solved using lookahead techniques. Now all cubes appear in the partition.

Conquer solver `iLingeling`: Added DRUP proof logging support. Each CDCL solver within `iLingeling` produces its own proof. The post-CNFs are the set of negated solved cubes.

Evaluation: Parallel Proof Generation and Checking

Using Cube-and-Conquer to generate the proof; check in parallel



Evaluation: Parallel Proof Checking

We evaluated our parallel proof checking method using several "short" DRAT proofs for hard-combinatorial problems that were solved by symmetry-breaking [CADE-25].

Experiments on TACC cluster having 16 cores per node.

instance	size	check	split	CNFs	seq-chk	par-chk	$\frac{\text{seq+init}}{\text{par+init}}$	$\frac{\text{seq}}{\text{par}}$
EDP1161	2,180	3,331	2.91	85.70	3,288	455.93	6.20	7.21
Ramsey4	20.01	2.55	0.04	1.91	4.19	0.43	2.58	9.74
tph6	2.78	0.61	0.01	1.25	2.03	0.22	2.22	9.23
tph7	5.09	1.30	0.02	1.39	2.70	0.29	2.41	9.31
tph8	10.68	2.98	0.03	1.61	4.29	0.46	2.82	9.32
tph9	34.18	6.17	0.04	1.98	7.33	0.83	3.28	8.83
tph10	19.86	11.78	0.06	2.51	12.67	1.32	3.92	9.60
tph11	56.49	22.96	0.09	3.39	22.64	2.85	4.13	7.94
tph12	92.29	39.42	0.15	4.73	39.07	3.89	5.01	10.04

size in MB, times in seconds

Conclusions

Conclusions

Conclusions:

- We presented composition rules and tools to produce and validate propositional proofs in parallel;
- We can produce proofs of Cube-and-Conquer solvers;
- Parallel proof validation based on the rules is efficient.

Future work:

- Apply and evaluate the method to a huge proof, 100+ TB;
- Develop a method to trim large proofs in parallel;
- Develop a method to produce proofs from `Treengeling`;
- Implement a mechanically-verified clausal proof checker.

Conclusions

Conclusions:

- We presented composition rules and tools to produce and validate propositional proofs in parallel;
- We can produce proofs of Cube-and-Conquer solvers;
- Parallel proof validation based on the rules is efficient.

Future work:

- Apply and evaluate the method to a huge proof, 100+ TB;
- Develop a method to trim large proofs in parallel;
- Develop a method to produce proofs from `Treengeling`;
- Implement a mechanically-verified clausal proof checker.

Thanks! Questions?