Proofs for Satisfiability Problems

Marijn J.H. Heule

THE UNIVERSITY OF TEXAS AT AUSTIN

Joint work with

Armin Biere

JOHANNES KEPLER UNIVERSITY LINZ | JKU

$\forall$X.Xπ, July 18, 2014

Outline

Introduction

Introduction: "Small Example"

$$
\begin{array}{l}
(x_5 \vee x_8 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_3 \vee \bar{x}_7) \wedge (\bar{x}_5 \vee x_3 \vee x_8) \wedge \\
(\bar{x}_6 \vee \bar{x}_1 \vee \bar{x}_5) \wedge (x_8 \vee \bar{x}_9 \vee x_3) \wedge (x_2 \vee x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_8 \vee x_4) \wedge \\
(\bar{x}_9 \vee \bar{x}_6 \vee x_8) \wedge (x_8 \vee x_3 \vee \bar{x}_9) \wedge (x_9 \vee \bar{x}_3 \vee x_8) \wedge (x_6 \vee \bar{x}_9 \vee x_5) \wedge \\
(x_2 \vee \bar{x}_3 \vee \bar{x}_8) \wedge (x_8 \vee \bar{x}_6 \vee \bar{x}_3) \wedge (x_8 \vee \bar{x}_3 \vee \bar{x}_1) \wedge (\bar{x}_8 \vee x_6 \vee \bar{x}_2) \wedge \\
(x_7 \vee x_9 \vee \bar{x}_2) \wedge (x_8 \vee \bar{x}_9 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_9 \vee x_4) \wedge (x_8 \vee x_1 \vee \bar{x}_2) \wedge \\
(x_3 \vee \bar{x}_4 \vee \bar{x}_6) \wedge (\bar{x}_1 \vee \bar{x}_7 \vee x_5) \wedge (\bar{x}_7 \vee x_1 \vee x_6) \wedge (\bar{x}_5 \vee x_4 \vee \bar{x}_6) \wedge \\
(\bar{x}_4 \vee x_9 \vee \bar{x}_8) \wedge (x_2 \vee x_9 \vee x_1) \wedge (x_5 \vee \bar{x}_7 \vee x_1) \wedge (\bar{x}_7 \vee \bar{x}_9 \vee \bar{x}_6) \wedge \\
(x_2 \vee x_5 \vee x_4) \wedge (x_8 \vee \bar{x}_4 \vee x_5) \wedge (x_5 \vee x_9 \vee x_3) \wedge (\bar{x}_5 \vee \bar{x}_7 \vee x_9) \wedge \\
(x_2 \vee \bar{x}_8 \vee x_1) \wedge (\bar{x}_7 \vee x_1 \vee x_5) \wedge (x_1 \vee x_4 \vee x_3) \wedge (x_1 \vee \bar{x}_9 \vee \bar{x}_4) \wedge \\
(x_3 \vee x_5 \vee x_6) \wedge (\bar{x}_6 \vee x_3 \vee \bar{x}_9) \wedge (\bar{x}_7 \vee x_5 \vee x_9) \wedge (x_7 \vee \bar{x}_5 \vee \bar{x}_2) \wedge \\
(x_4 \vee x_7 \vee x_3) \wedge (x_4 \vee \bar{x}_9 \vee \bar{x}_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge \\
(x_6 \vee x_7 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_6 \vee x_2 \vee x_3) \wedge (\bar{x}_8 \vee x_2 \vee x_5)
\end{array}
$$

- Does there exist an assignment satisfying all clauses?

Introduction: "Small Example"

$$
\begin{aligned}
&(x_5 \vee x_8 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_3 \vee \bar{x}_7) \wedge (\bar{x}_5 \vee x_3 \vee x_8) \wedge \\
&(\bar{x}_6 \vee \bar{x}_1 \vee \bar{x}_5) \wedge (x_8 \vee \bar{x}_9 \vee x_3) \wedge (x_2 \vee x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_8 \vee x_4) \wedge \\
&(\bar{x}_9 \vee \bar{x}_6 \vee x_8) \wedge (x_8 \vee x_3 \vee \bar{x}_9) \wedge (x_9 \vee \bar{x}_3 \vee x_8) \wedge (x_6 \vee \bar{x}_9 \vee x_5) \wedge \\
&(x_2 \vee \bar{x}_3 \vee \bar{x}_8) \wedge (x_8 \vee \bar{x}_6 \vee \bar{x}_3) \wedge (x_8 \vee \bar{x}_3 \vee \bar{x}_1) \wedge (\bar{x}_8 \vee x_6 \vee \bar{x}_2) \wedge \\
&(x_7 \vee x_9 \vee \bar{x}_2) \wedge (x_8 \vee \bar{x}_9 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_9 \vee x_4) \wedge (x_8 \vee x_1 \vee \bar{x}_2) \wedge \\
&(x_3 \vee \bar{x}_4 \vee \bar{x}_6) \wedge (\bar{x}_1 \vee \bar{x}_7 \vee x_5) \wedge (\bar{x}_7 \vee x_1 \vee x_6) \wedge (\bar{x}_5 \vee x_4 \vee \bar{x}_6) \wedge \\
&(\bar{x}_4 \vee x_9 \vee \bar{x}_8) \wedge (x_2 \vee x_9 \vee x_1) \wedge (x_5 \vee \bar{x}_7 \vee x_1) \wedge (\bar{x}_7 \vee \bar{x}_9 \vee \bar{x}_6) \wedge \\
&(x_2 \vee x_5 \vee x_4) \wedge (x_8 \vee \bar{x}_4 \vee x_5) \wedge (x_5 \vee x_9 \vee x_3) \wedge (\bar{x}_5 \vee \bar{x}_7 \vee x_9) \wedge \\
&(x_2 \vee \bar{x}_8 \vee x_1) \wedge (\bar{x}_7 \vee x_1 \vee x_5) \wedge (x_1 \vee x_4 \vee x_3) \wedge (x_1 \vee \bar{x}_9 \vee \bar{x}_4) \wedge \\
&(x_3 \vee x_5 \vee x_6) \wedge (\bar{x}_6 \vee x_3 \vee \bar{x}_9) \wedge (\bar{x}_7 \vee x_5 \vee x_9) \wedge (x_7 \vee \bar{x}_5 \vee \bar{x}_2) \wedge \\
&(x_4 \vee x_7 \vee x_3) \wedge (x_4 \vee \bar{x}_9 \vee \bar{x}_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge \\
&(x_6 \vee x_7 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_6 \vee x_2 \vee x_3) \wedge (\bar{x}_8 \vee x_2 \vee x_5)
\end{aligned}
$$

- How to make (compact) proofs for unsatisfiable problems?

Proof Systems

Proof Systems: Resolution Rule and Resolution Chains

Resolution Rule

$$\frac{(x \vee a_1 \vee \ldots \vee a_i) \qquad (\bar{x} \vee b_1 \vee \ldots \vee b_j)}{(a_1 \vee \ldots \vee a_i \vee b_1 \vee \ldots \vee b_j)}$$

- Many SAT techniques can be simulated by resolution.

Proof Systems: Resolution Rule and Resolution Chains

Resolution Rule

$$\frac{(x \vee a_1 \vee \ldots \vee a_i) \qquad (\bar{x} \vee b_1 \vee \ldots \vee b_j)}{(a_1 \vee \ldots \vee a_i \vee b_1 \vee \ldots \vee b_j)}$$

- Many SAT techniques can be simulated by resolution.

A resolution chain is a sequence of resolution steps.
The resolution steps are performed from left to right.

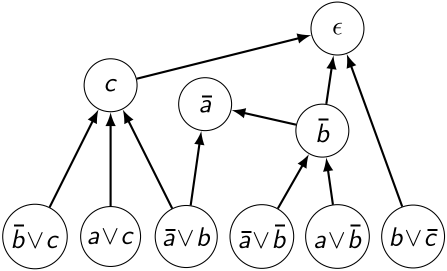
Example

- $(c) := (\bar{a} \vee \bar{b} \vee c) \diamond (\bar{a} \vee b) \diamond (a \vee c)$
- $(\bar{a} \vee c) := (\bar{a} \vee b) \diamond (a \vee c) \diamond (\bar{a} \vee \bar{b} \vee c)$
- The order of the clauses in the chain matter

Proof Systems: Resolution Proofs versus Clausal Proofs

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

A resolution graph of F is:



A resolution proof consists of all nodes and edges of the resolution graph

- Graphs from CDCL solvers have ~ 400 incoming edges per node
- Resolution proof logging can heavily increase memory usage ($\times 100$)

A clausal proof is a list of all nodes sorted by topological order

- Clausal proofs are easy to emit and relatively small
- Clausal proof checking requires to reconstruct the edges (costly)

Proof Systems: Extended Resolution and Generalizations

Extended Resolution Rule

Given a Boolean formula F without the Boolean variable x, the clauses $(x \vee \bar{a} \vee \bar{b}) \wedge (\bar{x} \vee a) \wedge (\bar{x} \vee b)$ are redundant with respect to F.

- All existing techniques can be simulated by extended resolution
- For several techniques it is not known how to do the simulation

Blocked Clauses [Kullmann'99]

A clause C is blocked on literal $l \in C$ w.r.t. a formula F is all resolvents of C and D with $\bar{l} \in D$ are tautologies.

Example

Consider the formula $F = (\bar{x} \vee a) \wedge (\bar{x} \vee b)$. Clause $(x \vee \bar{a} \vee \bar{b})$ is blocked on x with respect to F, because $(x \vee \bar{a} \vee \bar{b}) \diamond_x (\bar{x} \vee a) = (\bar{a} \vee \bar{b} \vee a)$ and $(x \vee \bar{a} \vee \bar{b}) \diamond_x (\bar{x} \vee b) = (\bar{a} \vee \bar{b} \vee b)$ are both tautologies.

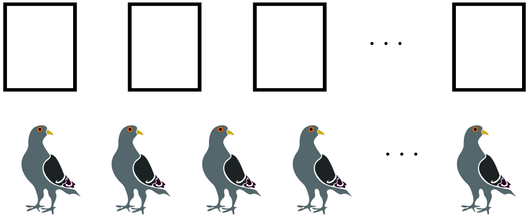
Theorem: Addition of an arbitrary blocked clause preserves satisfiability.

Proof Systems: Pigeon Hole Principe Proofs

Classic problem: Can n pigeons be in $n-1$ pigeon holes?

$n-1$ holes:

n pigeons:



Hard for resolution: proofs are exponential in size!

ER proofs can be exponentially smaller [Cook'76]

- reduce a problem with n pigeons and $n-1$ holes
  into a problem with $n-1$ pigeons and $n-2$ holes

Proof Search

Proof Search: Conflict-Driven Clause Learning (CDCL)

The leading search paradigm is conflict-driven clause learning:

- During each step the current assignment is extended;
- If the assignment is falsified a conflict clause is computed;
- Each conflict clause can be expressed as a resolution chain;
- Decisions are based on variables in recent conflict clauses.

CDCL solvers use lots of pre- or in-processing techniques:

- Most techniques can be expressed using resolution chains;
- Weakening techniques can be ignored for UNSAT proofs;
- Some techniques are even difficult to express using extended resolution and its generalizations: e.g. Gaussian elimination, cardinality resolution, and symmetry breaking.

Proof Formats

Proof Formats: The Input Format DIMACS

$$E := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$$

The input format of SAT solvers is known as DIMACS

- header starts with `p cnf` followed by the number of variables (n) and the number of clauses (m)
- the next m lines represent the clauses
- positive literals are positive numbers
- negative literals are negative numbers
- clauses are terminated with a `0`

```
p cnf 3 6
-2  3 0
 1  3 0
-1  2 0
-1 -2 0
 1 -2 0
 2 -3 0
```

Most proof formats use a similar syntax.

Proof Formats: TraceCheck Overview

TraceCheck is the most popular resolution-style format.

$$E := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$$

TraceCheck is readable and resolution chains make it relatively compact

$$\begin{array}{rcl}
\langle\text{trace}\rangle & = & \{\langle\text{clause}\rangle\} \\
\langle\text{clause}\rangle & = & \langle\text{pos}\rangle\langle\text{literals}\rangle\langle\text{antecedents}\rangle \\
\langle\text{literals}\rangle & = & \text{“}*\text{”} \mid \{\langle\text{lit}\rangle\}\,\text{“0”} \\
\langle\text{antecedents}\rangle & = & \{\langle\text{pos}\rangle\}\,\text{“0”} \\
\langle\text{lit}\rangle & = & \langle\text{pos}\rangle \mid \langle\text{neg}\rangle \\
\langle\text{pos}\rangle & = & \text{“1”} \mid \text{“2”} \mid \cdots \mid \langle\text{max}-\text{idx}\rangle \\
\langle\text{neg}\rangle & = & \text{“}-\text{”}\,\langle\text{pos}\rangle
\end{array}$$

1	−2	3	0	**0**		
2	1	3	0	**0**		
3	−1	2	0	**0**		
4	−1	−2	0	**0**		
5	1	−2	0	**0**		
6	2	−3	0	**0**		
7	−2	0	**4**	**5**	**0**	
8	3	0	**1**	**2**	**3**	**0**
9	0	**6**	**7**	**8**	**0**	

Proof Formats: TraceCheck Examples

TraceCheck is the most popular resolution-style format.

$$E := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$$

TraceCheck is readable and resolution chains make it relatively compact

The clauses **1** to **6** are input clauses

Clause **7** is the resolvent **4** and **5**:

- $(\bar{b}) := (\bar{a} \vee \bar{b}) \diamond (a \vee \bar{b})$

Clause **8** is the resolvent **1**, **2** and **3**:

- $(c) := (\bar{b} \vee c) \diamond (\bar{a} \vee b) \diamond (a \vee c)$
- NB: the antecedents are swapped!

Clause **9** is the resolvent **6**, **7** and **8**:

- $\epsilon := (b \vee \bar{c}) \diamond (\bar{b}) \diamond (c)$

```
1 -2  3 0 0
2  1  3 0 0
3 -1  2 0 0
4 -1 -2 0 0
5  1 -2 0 0
6  2 -3 0 0
7 -2  0 4 5 0
8  3  0 1 2 3 0
9  0  6 7 8 0
```

Proof Formats: TraceCheck Don't Cares

Support for unsorted clauses, unsorted antecedents and omitted literals.

- Clauses are not required to be sorted based on the clause index

8	3	0	**1**	**2**	**3**	**0**	$\equiv$	**7**	-2	0	**4**	**5**	**0**
7	-2	0	**4**	**5**	**0**			**8**	3	0	**1**	**2**	**3**	**0**

- The antecedents of a clause can be in arbitrary order

7	-2	0	**5**	**4**	**0**		$\equiv$	**7**	-2	0	**4**	**5**	**0**
8	3	0	**3**	**1**	**2**	**0**		**8**	3	0	**1**	**2**	**3**	**0**

- For learned clauses, the literals can be omitted using *

7	$\star$	**5**	**4**	**0**			$\equiv$	**7**	-2	0	**4**	**5**	**0**
8	$\star$	**3**	**1**	**2**	**0**			**8**	3	0	**1**	**2**	**3**	**0**

Proof Formats: Reverse Unit Propagation (RUP)

Unit Propagation

Given an assignment φ, extend it by making unit clauses true — until fixpoint or a clause becomes false

Reverse Unit Propagation (RUP)

A clause $C = (l_1 \vee l_2 \vee \cdots \vee l_k)$ has reverse unit propagation w.r.t. formula F if unit propagation of the assignment $\varphi = \bar{C} = (\bar{l}_1 \wedge \bar{l}_2 \wedge \ldots \wedge \bar{l}_k)$ on F results in a conflict.
We write: $F \wedge \bar{C} \vdash_1 \epsilon$

A clause sequence $C_1, \ldots, C_m$ is a RUP proof for formula F

- $F \wedge C_1 \wedge \cdots \wedge C_{i-1} \wedge \bar{C}_i \vdash_1 \epsilon$
- $C_m = \epsilon$

Proof Formats: RUP, DRUP, RAT, and DRAT

RUP and extensions is the most popular clausal-style format.

$$E := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$$

RUP is much more compact than TraceCheck because it does not includes the resolution steps.

$$\begin{array}{rcl}
\langle\text{proof}\rangle & = & \{\langle\text{lemma}\rangle\} \\
\langle\text{lemma}\rangle & = & \langle\text{delete}\rangle\{\langle\text{lit}\rangle\}\text{“0”} \\
\langle\text{delete}\rangle & = & \text{“”} \mid \text{“d”} \\
\langle\text{lit}\rangle & = & \langle\text{pos}\rangle \mid \langle\text{neg}\rangle \\
\langle\text{pos}\rangle & = & \text{“1”} \mid \text{“2”} \mid \cdots \mid \langle\text{max}-\text{idx}\rangle \\
\langle\text{neg}\rangle & = & \text{“}-\text{”}\langle\text{pos}\rangle
\end{array}$$

```
-2  0
 3  0
 0
```

$$E \wedge (b) \vdash_1 \epsilon$$
$$E \wedge (\bar{b}) \wedge (\bar{c}) \vdash_1 \epsilon$$
$$E \wedge (\bar{b}) \wedge (c) \vdash_1 \epsilon$$

Proof Formats: Open Issues and Challenges

How get useful information from a proof?

- Clausal or variable core
- Resolution proof from a clausal proof
- Interpolant
- Proof minimization
- Inside the SAT solver or using an external tool?
- What would be a good API to manipulate proofs?

How to store proofs compactly?

- Question is important for resolution and clausal proofs
- Current formats are "readable" and hence large
- Time for a binary format? How much can be saved?

Proof Production

Producing Resolution Proofs

Producing a resolution proof from a SAT solver can hard

- Expressing some powerful techniques in CDCL solvers as resolution chains is non-trivial (e.g. clause minimization), both figuring out the antecedents and the resolution order;
- Storing the resolution graph requires a lot of memory and requires techniques to reduces the memory consumption;
- It is not clear how to deal with techniques that go beyond resolution (e.g. bounded variable addition).

Producing Clausal Proofs

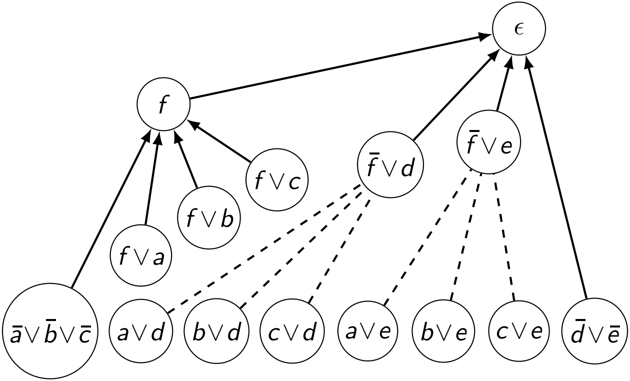
In most cases, emitting a clausal proof is easy and cheap

- Learning: Add a clause to the proof;
- Strengthening: Add the shortened clause, delete original;
- Weakening: Delete the clause;
- Works for several techniques based on extended resolution;
- Dump all actions directly to disk, no memory overhead.

For some techniques it is not known how to do it elegantly

- in particular: Gaussian elimination, cardinality resolution, and symmetry breaking.

Producing Proofs with Generalized Extended Resolution



Proof Consumption

Proof Consumption

Resolution Proofs

Validating resolution proofs consists of checking whether the added clauses can be constructed from the list of antecedents.

- Validation can be challenging due to the enormous size of proofs, i.e., file I/O costs are much higher than CPU time.

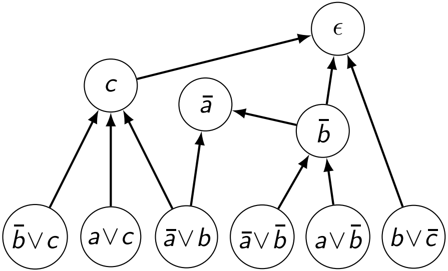
Clausal Proofs

Validating resolution proofs consists of finding the antecedents.

Reconstructing a Resolution Graph from a Clausal Proof

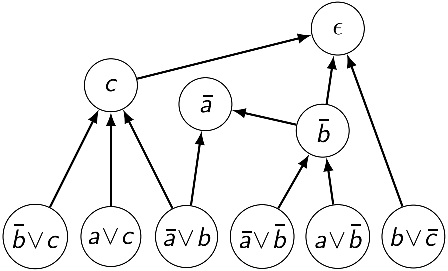
Consider the resolution graph on the left. The clausal proof is $\{(\bar{b}), (\bar{a}), (c), \epsilon\}$.

One can obtain smaller cores using reconstruction heuristics [FMCAD13].

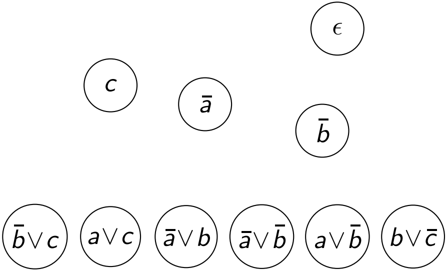


Reconstructing a Resolution Graph from a Clausal Proof

Consider the resolution graph on the left. The clausal proof is $\{(\bar{b}), (\bar{a}), (c), \epsilon\}$.



One can obtain smaller cores using reconstruction heuristics [FMCAD13].

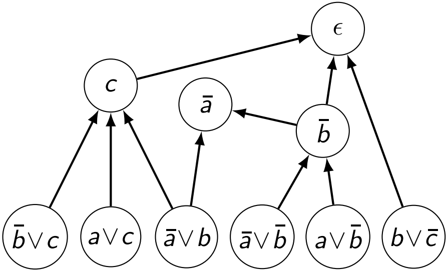


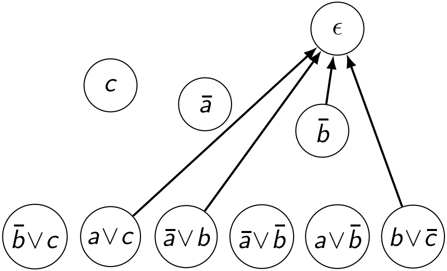
Reconstruction starts w/o incoming edges and traverses the proof in reverse order and marks using conflict analysis.

Reconstructing a Resolution Graph from a Clausal Proof

Consider the resolution graph on the left. The clausal proof is $\{(\bar{b}), (\bar{a}), (c), \epsilon\}$.

One can obtain smaller cores using reconstruction heuristics [FMCAD13].



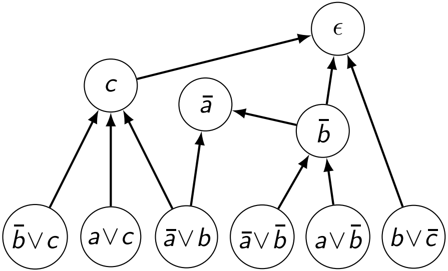


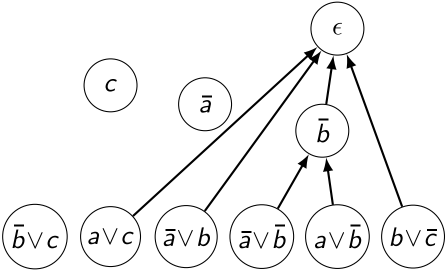
Reconstruction starts w/o incoming edges and traverses the proof in reverse order and marks using conflict analysis.

Reconstructing a Resolution Graph from a Clausal Proof

Consider the resolution graph on the left. The clausal proof is $\{(\bar{b}), (\bar{a}), (c), \epsilon\}$.

One can obtain smaller cores using reconstruction heuristics [FMCAD13].





Reconstruction starts w/o incoming edges and traverses the proof in reverse order and marks using conflict analysis.

Applications

Applications

Validating the output of SAT solvers:

- Voluntary during SAT Competition (SC) 2007, 2009, 2011;
- Mandatory during SC 2013 (DRUP) and 2014 (DRAT);
- Validating output is about as expensive as SAT solving;
- Debug SAT solvers especially in combination with fuzzing.

Produce unsatisfiable cores:

- Useful for many applications: minimal unsatisfiable core extraction, MaxSAT, diagnosis, model checking, and SMT.

Resolution proofs are useful for extracting interpolants:

- However, resolution proofs are huge and hard to obtain;
- This was the state-of-the-art until the invention of IC3.

Conclusions

Conclusions

Proofs of unsatisfiability useful for several applications:

- Validate results of SAT solvers;
- Extracting minimal unsatisfiable cores;
- Computing Interpolants;
- Tools that use SAT solvers, such as theorem provers.

Challenges:

- Reduce size of proofs on disk and in memory;
- Reduce the cost to validate clausal proofs;
- How to deal with Gaussian elimination, cardinality resolution, and symmetry breaking?

Thanks!