CS 433 Automated Reasoning 2024

Lecture 23: Proof generation

Instructor: Ashutosh Gupta

IITB India

Compile date: 2024-04-17

Evidence of unsat

If a formula is sat then the solver produces a model as an evidence of satisfiability.

Otherwise, it produces only UNSAT.

Solvers should also produce a proof for unsatisfiability.

Learned clauses will help us constructing the proofs.

Issues in generating proofs in SAT solvers or any solver

Proof format vs. checking

- Detailed proofs require non-trivial work from solvers, causing overhead.
- Missing details in proofs imply expensive proof checkers.

Proof minimization

- Problems of moderate size may have very large proofs
- Proofs often have redundancies
- It is wise to minimize proofs before dumping it out

Proof formats for SAT solvers

SAT solvers typically return two kinds of proofs

- Clausal proofs, i.e., list of learned clauses (low overhead)
- Resolution proofs (detailed)

Marijn J.H. Heule and Armin Biere. Proofs for Satisfiability Problems https://www.cs.utexas.edu/~marijn/publications/APPA.pdf

Topic 23.1

Clausal proof generation from SAT solver

Learned clause proofs

The list of learned clause can be considered proofs.

Example 23.1

Input CNF

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

Learned clauses

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

Learned clause proofs with deletions

A learned clause may be deleted over the run. A new entry is added with prefix `d`. The format is called DRAT.

Example 23.2

Input CNF

```
p cnf 5 8
-1 -2 -3 0
1  4 0
1  5 0
2  4 0
2  5 0
3  4 0
3  5 0
-4 -5 0
```

DRAT clausal proof

```
6 1 0
6 2 0
6 3 0
-6 4 0
-6 5 0
d  1 4 0
d  2 4 0
d  3 4 0
d  1 5 0
d  2 5 0
d  3 5 0
6 0
0
```

Proof checking

A proof is a proof only if an independent checker can check it efficiently.

Let $L_1, ..., L_m$ be learned clauses for CNF formula F such that $L_m = \emptyset$.

To check a learned clauses proof, we need to check the following for each L_i

$$F \wedge L_1 \wedge \cdots \wedge L_{i-1} \wedge \underbrace{\neg L_i}_{\text{conjunction of literals}}$$

results in contradiction after unit propagation.(Why?)

Exercise 23.1
Explain why?

Clausal Proof checking algorithm

Algorithm 23.1: ProofChecking

Input: CNF $F, L_1, \ldots, L_n$

marked $:= \lambda x.\bot$;

marked$(\emptyset) := \top$;

while *i is partial or n...1* **do**

  if *marked*(L_i) **then**

    $m :=$ UNITPROPAGATION$(\emptyset, F \wedge L_1 \wedge \cdots \wedge L_{i-1} \wedge \neg L_i)$;

    if $m \not\models F$ **then**

      for each clause L that participate in the conflict *marked*$(L) := \top$

    else

      throw "invalid proof"

return *"valid proof"*

Commentary: UNITPROPAGATION takes initial partial model as input, which is in the above case is empty. It returns a model that is enforced by unit propagation. If the model does not satisfy input formula, it is unsatisfiable.

Clausal proof checking is expensive

Sometimes more expensive than solving

- Gets exacerbated due to clause deletions in SAT solvers
  - deleted clauses are saved in the proof
  - too many deleted clauses
- No reuse of propagations
- No effcient representation of many simplifications,
  - e.g., Gaussian elimination, etc.
  - cannot be resolved without introducing complex proof format

Topic 23.2

Resolution proof generation from SAT solver

Resolution Proofs

A proof is written in a given proof system. Here, we choose resolution.

A resolution proof rule is

$$\frac{p \vee C \qquad \neg p \vee D}{C \vee D}.$$

Variable p is called the pivot of the inference.

Example 23.3

Suppose $F = (p \vee q) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$

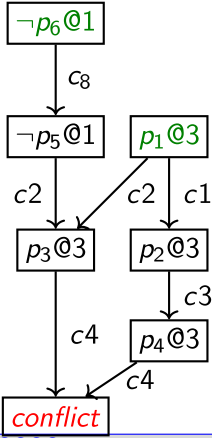
$$\cfrac{\cfrac{\cfrac{p \vee q \qquad \neg p \vee q}{q} \qquad \neg q \vee r}{r} \qquad \neg r}{\perp}$$

Reading proofs from implication graphs

- For each learned clause we assign a resolution proof that proves that the learned clause is implied by the clauses in the solver so far.

Le us demonstrate the process using an example.

Example 23.4



Input clauses:

$c_8 = (p_6 \vee \neg p_5) \qquad c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_1 = (\neg p_1 \vee p_2) \qquad c_3 = (\neg p_2 \vee p_4) \quad c_4 = (\neg p_3 \vee \neg p_4)$

Conflict clause : $p_6 \vee \neg p_1$

Conflict as a resolution proof:

$$\dfrac{\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \qquad \dfrac{p_6 \vee \neg p_5 \qquad \neg p_6}{\neg p_5}}{\neg p_1 \vee p_3} \qquad p_1}{p_3} \qquad \dfrac{\neg p_3 \vee \neg p_4 \qquad \dfrac{\neg p_2 \vee p_4 \qquad \dfrac{\neg p_1 \vee p_2 \qquad p_1}{p_2}}{p_4}}{\neg p_3}}{\bot}$$

Resolution proofs for conflict clauses

Example 23.5 (contd.)

$$\dfrac{\dfrac{\dfrac{\neg p_1 \lor p_3 \lor p_5 \qquad \dfrac{p_6 \lor \neg p_5 \qquad \neg \cancel{p_6}}{p_6 \lor \neg p_5}}{p_6 \lor \neg p_1 \lor p_3} \qquad \cancel{p_1}}{p_6 \lor \neg p_1 \lor p_3} \qquad \dfrac{\neg p_3 \lor \neg p_4 \qquad \dfrac{\neg p_2 \lor p_4 \qquad \dfrac{\neg p_1 \lor p_2 \qquad \cancel{p_1}}{\neg p_1 \lor p_2}}{\neg p_1 \lor p_4}}{\neg p_1 \lor \neg p_3}}{p_6 \lor \neg p_1 \lor \bot}$$

The above is a resolution proof of the conflict clause.

One more issue:

There may be a leaf of the above proof that is a conflict clause in itself.

- In the case, there must be a resolution proof for the conflict clause.
- We "stitch" that proof on top of the above proof .

CDCL with proof generation

Algorithm 23.2: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$; $proofs = \lambda C.C$;

$\textsc{UnitPropagation}(m, F)$;

do

  // backtracking

  while $m \not\models F$ **do**

    $(C, dl, proof) := \textsc{AnalyzeConflict}(m, F, proofs)$;

    $proofs(C) := proof$;

    if $C = \emptyset$ **then return** *unsat(proof)*;

    $m.resize(dstack(dl))$; $F := F \cup \{C\}$; $\mathrm{m} := \textsc{UnitPropagation}(m, F)$;

  // Boolean decision

  if *m is partial* **then**

    $dstack(dl) := m.size()$;

    $dl := dl + 1$; $\textsc{Decide}(m, F)$; $\textsc{UnitPropagation}(m, F)$;

while *m is partial or* $m \not\models F$;

return *sat*

Resolution proof format in SAT solvers

SAT solvers can dump resolution proofs in a standard format.

Example 23.6

Input CNF

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

Learned clauses

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

Resolution proof

```
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
```

$$\frac{\ell_1 \vee C_1 \quad \dots \quad \ell_k \vee C_k \quad \neg\ell_1 \vee \dots \vee \neg\ell_k \vee D}{C_1 \vee \dots \vee C_k \vee D}$$

Topic 23.3

Proof minimization

Recall: Resolution Proofs

A proof is written in a given proof system. Here, we may choose resolution for propositional logic.

A resolution proof rule is

$$\frac{p \lor C \qquad \neg p \lor D}{C \lor D}.$$

Variable p is called the pivot of the inference.

Example 23.7

Suppose $F = (p \lor q) \land (\neg p \lor q) \land (\neg q \lor r) \land \neg r$

$$\cfrac{\cfrac{\cfrac{p \lor q \qquad \neg p \lor q}{q} \qquad \neg q \lor r}{r} \qquad \neg r}{\bot}$$

Proof minimization

- There are several kinds of redundancies that may occur in proofs.
- We may apply several passes to minimize for each kind
- A minimization pass should preferably be a linear-time algorithm

Here we present two such cases.

Proofs as directed acyclic graphs

A proof is a directed acyclic graph, not a tree.

Example 23.8

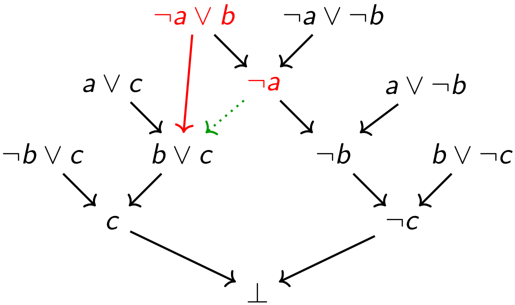
$$
\begin{tikzcd}
 & & & \neg a \lor b \arrow[dd] \arrow[dr] & & \neg a \lor \neg b \arrow[dl] & & \\
 & & a \lor c \arrow[dr] & & \neg a \arrow[dr] & & a \lor \neg b \arrow[dl] & \\
 & \neg b \lor c \arrow[dr] & & b \lor c \arrow[dl] & & \neg b \arrow[dr] & & b \lor \neg c \arrow[dl] \\
 & & c \arrow[drr] & & & & \neg c \arrow[dll] & \\
 & & & & \bot & & &
\end{tikzcd}
$$

Leaves are input clauses.

Minimization: stronger clauses

If a node in a proof is weaker than another node, we may replace the node.

Example 23.9



The red edge can be replaced by the dotted edge.

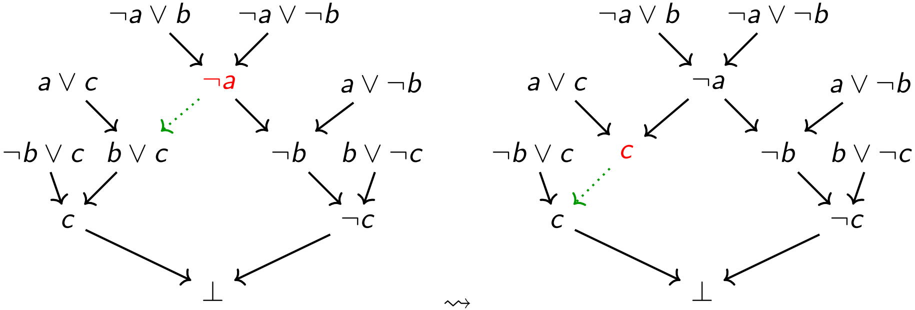
Exercise 23.2

When can we not apply the transformation?

Effect of strengthening : decedents become stronger

Due to stronger antecedents, the decedents can also become stronger.

Example 23.10

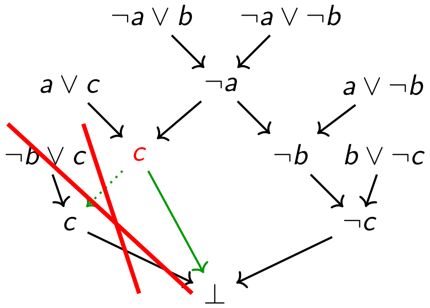


Effect of strengthening : resolutions eliminated

As nodes get stronger many resolutions become useless.

Proofs can be short circuited.

Example 23.11



Second minimization : redundant resolutions

The process of resolution removes a literal in each step until none is left.

In a step, the pivot literal is removed and others may be introduced.

Definition 23.1

if a pivot is repeated in a derivation path to $\bot$, then the earlier resolution is redundant in the path.

Example 23.12

Consider the following resolution proof:

$$\begin{tikzcd}
 & & a \lor b \arrow[dr] & & \neg a \lor b \arrow[dl] & \\
 & & & {\color{red} b} \arrow[dr] & & \neg b \lor \neg a \arrow[dl] \\
a \lor \neg c \arrow[dr] & & a \lor c \arrow[dr] & & \neg a \arrow[dlll] \arrow[dl] & \\
 & \neg c \arrow[dr] & & c \arrow[dl] & & \\
 & & \bot & & &
\end{tikzcd}$$

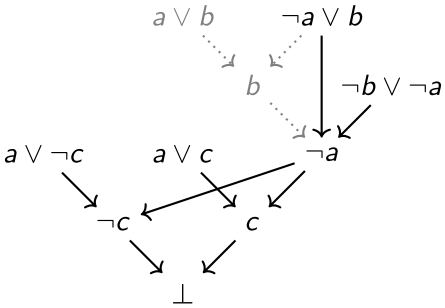
The resolution at ${\color{red} b}$ is redundant in both the paths to $\bot$.

Removing redundant resolution

By rewiring the proof, we may remove the redundant node v.

One of the parent of v will be wired to the children of v.

Example 23.13



After rewiring we may need to update clauses in some proof nodes.

Exercise 23.3

Which parent to choose?

Detecting redundant resolution - expansion set

Definition 23.2

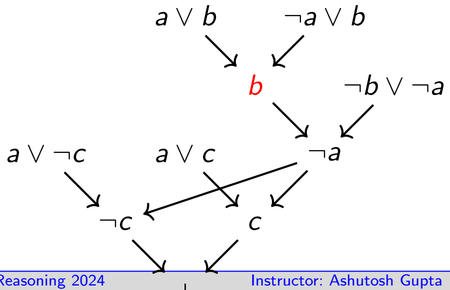
For a proof node v, expansion set $\rho(v)$ is the set of literals such that $\ell \in \rho(v)$ iff ℓ will be removed in all paths to $\bot$. ρ is defined as follows.

$$\rho(v) = \begin{cases} \emptyset & v = \bot \\ \bigcap\limits_{v' \in children(v)} \rho(v') \cup \{rlit(v, v')\} - \{\neg rlit(v, v')\} & otherwise \end{cases}$$

where $rlit(v, v')$ is the literal involved on the edge (v, v').

Exercise 23.4

Calculate $\rho(v)$ for each node:



Detecting redundant resolution (contd.)

Theorem 23.1

If pivot(v) or $\neg$pivot$(v) \in \rho(v)$ then v is redundant.

Exercise 23.5

a. What is the complexity of computing ρ?

b. Prove $\rho(v) \supseteq$ literals in v

c. Given the above observations suggest an heuristic optimization.

Topic 23.4

Proofs from theory solvers

Theory solvers

Each theory needs to have its own proof rules and instrumentation of the employed decision procedure to obtain proofs.

Here, we will look at two examples

- Theory of linear rational arithmetic ($\mathcal{T}_{LRA}$)
- Theory of equality with uninterpreted functions($\mathcal{T}_{EUF}$)

Proof generation in $\mathcal{T}_{LRA}$

In the theory of LRA, atoms are linear constraints over rational variables.

The following is the only proof rule for the theory.

$$\frac{a_1 x \leq b_1 \quad a_1 x \leq b_1}{(\lambda_1 a_1 + \lambda_2 a_2)x \leq (\lambda_1 b_1 + \lambda_2 b_2)} \lambda_1, \lambda_2 \geq 0$$

Example 23.14

Consider: $3x_1 \leq -6 \wedge x_1 - 3x_2 \leq 1 \wedge x_1 + x_2 \leq 2$

$$\cfrac{3x_1 \leq -6 \qquad \cfrac{x_1 - 3x_2 \leq 1 \quad x_1 + x_2 \leq 2}{4x_1 \leq 7} \lambda_1 = 1, \lambda_2 = 3}{0 \leq -1} \lambda_1 = 4/3, \lambda_2 = 1$$

LRA solver

There are many decision procedures for solving LRA.

We will present proof generation via Fourier-Motzkin algorithm for solving LRA.

Proof generation from Fourier-Motzkin

Observation:

- Fourier-Motzkin proceeds by replacing inequalities by other inequalities
- incoming inequalities are positive linear combination of old inequalities
- We may instrument Fourier-Motzkin to keep the record and produce proof if input is found to be unsat

Example 23.15

In the previous example,

$$\dfrac{-x_1 + x_2 + 2x_3 \leq 0 \quad x_1 - x_3 \leq 0}{x_2 + x_3 \leq 0} \qquad \dfrac{\dfrac{-x_1 + x_2 + 2x_3 \leq 0 \quad x_1 - x_2 \leq 0}{x_3 \leq 0} \qquad -x_3 \leq -1}{0 \leq -1}$$

End of Lecture 23