Automated reasoning 2016

Lecture 5: Proof generation

Instructor: Ashutosh Gupta

TIFR, India

Compile date: 2016-04-26

Where are we and where are we going?

I assume, you have seen

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

We will see

- proof generation in CDCL($\mathcal{T}$) with $\mathcal{T}_{LRA}/\mathcal{T}_{EUF}$

Topic 5.1

Resolution proofs for propositional logic

Resolution Proofs

A resolution proof rule is

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

We say p is a pivot that produced $C \vee D$.

Example 5.1

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

$$\dfrac{\dfrac{\dfrac{p \vee q \qquad \neg p \vee q}{q} \qquad \neg q \vee r}{r} \qquad \neg r}{\bot}$$

Exercise 5.1

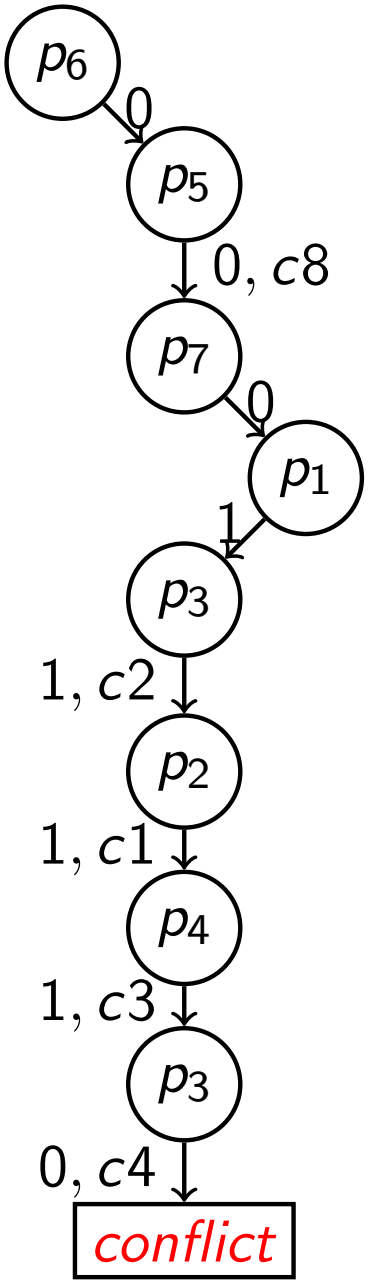
Prove the following generalized proof rule

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

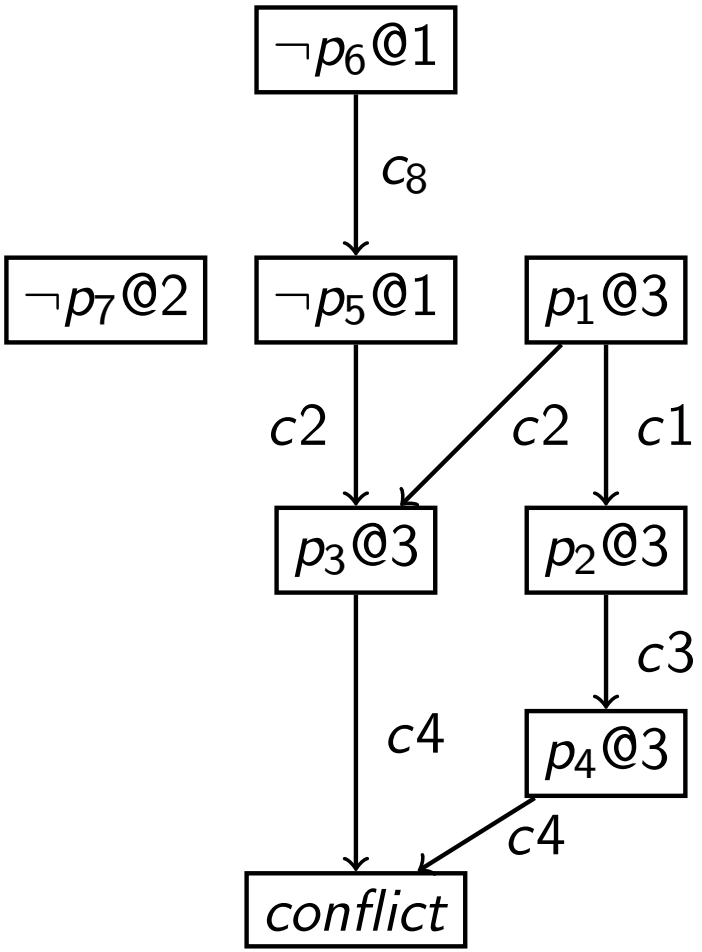
Implication graph for conflict graph

Example 5.2

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



Implication graph

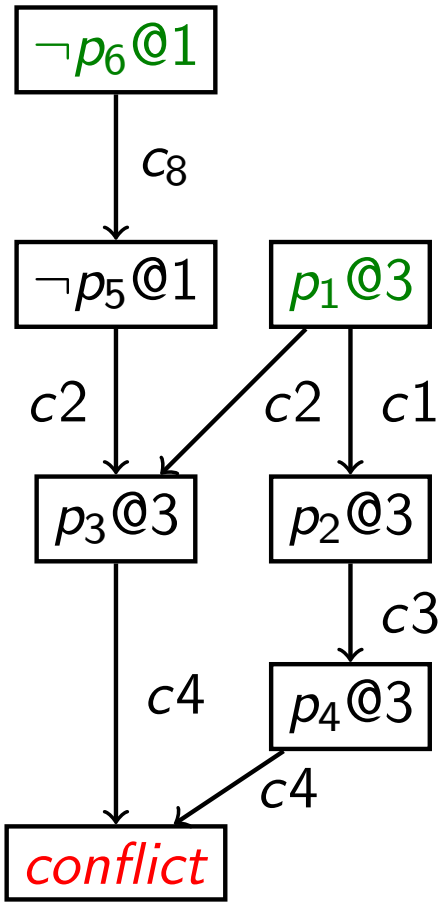


Reading proofs from implication graphs

- 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.

We demonstrate the process using an example.

Example 5.3



Input clauses:

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

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

Conflict clause : $p_6 \vee \neg p_1$

Conflict as a resolution proof:

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

$$\dfrac{p_3 \qquad \neg p_3}{\bot}$$

Resolution proofs for conflict clauses

Example 5.4 (contd.)

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

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($\mathcal{T}$) with proof generation

Algorithm 5.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

AddClauses($e(F)$); m := UnitPropagation();$dl := 0$;$dstack := \lambda x.0$;$proofs = \lambda C.C$;

do

  // backtracking

  while $\exists x\ \{x \mapsto 0, x \mapsto 1\} \subseteq m$ **do**

    $(C, dl, proof)$:= AnalyzeConflict(m,$proofs$); $proofs(C) := proof$;

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

    $m.resize(dstack(dl))$; AddClauses($\{C\}$); m := UnitPropagation();

  // Boolean decision

  if m *is partial* **then**

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

    $dl := dl + 1$; m := Decide(); m := UnitPropagation() ;

  // Theory propagation

  if $\forall x\ \{x \mapsto 0, x \mapsto 1\} \not\subseteq m$ **then**

    $(Cs, dl', proofs')$:= TheoryDeduction($\bigwedge e^{-1}(m)$);

    if $dl' < dl$ **then** $\{dl = dl'$; $m.resize(dstack(dl))$; $\}$;

    $proofs := proofs \cup proofs'$; AddClauses($e(Cs)$); m := UnitPropagation();

We also need proofs of Cs from the theory solvers

while m *is partial or* $\exists x\ \{x \mapsto 0, x \mapsto 1\} \subseteq m$;

return *sat*

Topic 5.2

Proofs in SAT solvers

Issues in generation 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 in SAT solvers

SAT solvers typically return two kinds of proofs

- List of learned clauses (low overhead)
- Resolution proofs (detailed)

Example 5.5

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
```

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

Proof checking

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

To check a learned clauses proof, we need to check the following things

1. Unit propagation on the input+learned clauses leads to conflict
2. Each learned clause is implied by the preceding+input clauses

Exercise 5.2

a. Give procedure for the 2nd step in the above proof checking
b. Give procedure for checking resolution proofs as presented in the the previous slide

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 one such case.

Redundent resolutions

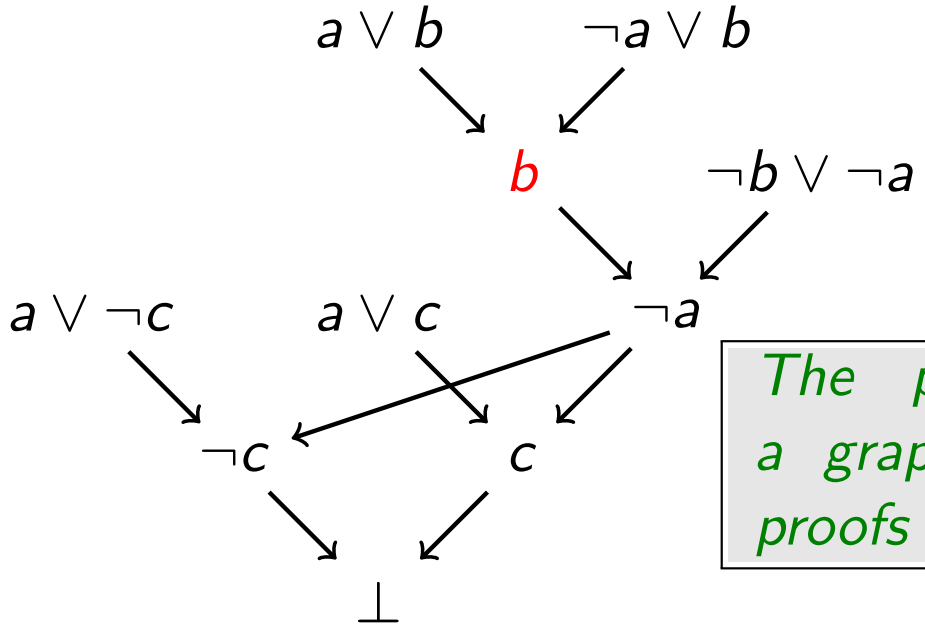
The process of resolution removes a literal in each step until none is left.
In a step, the pivot literal is removed and others may be introduced.

Definition 5.1

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

Example 5.6

Consider the following resolution proof:



The proof is shown as a graph to illustrate that proofs are DAGs not trees.

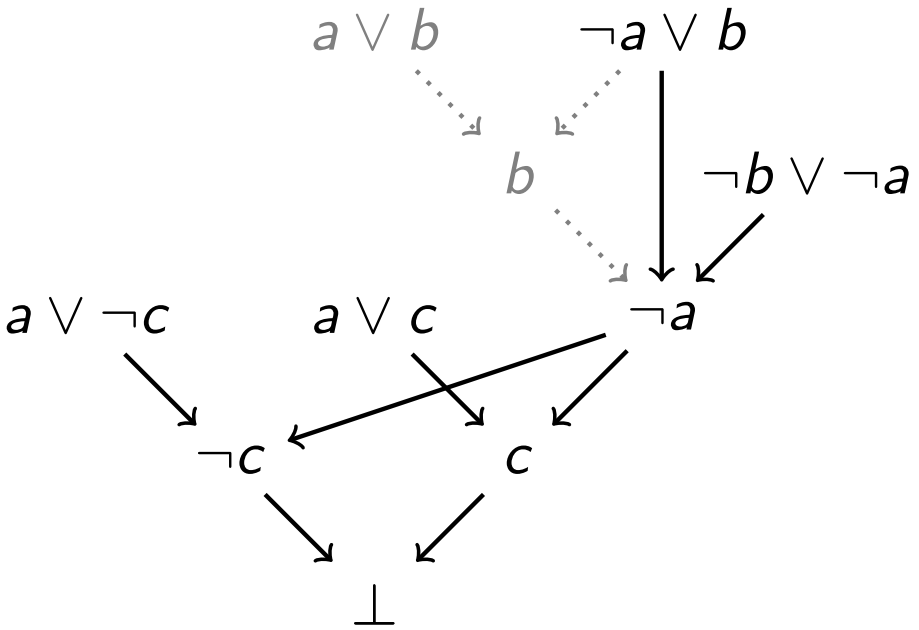
The resolution at b is redundant in both the paths to $\bot$.

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 5.7



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

Exercise 5.3

Which parent to choose?

Detecting redundant resolution - expansion set

Definition 5.2

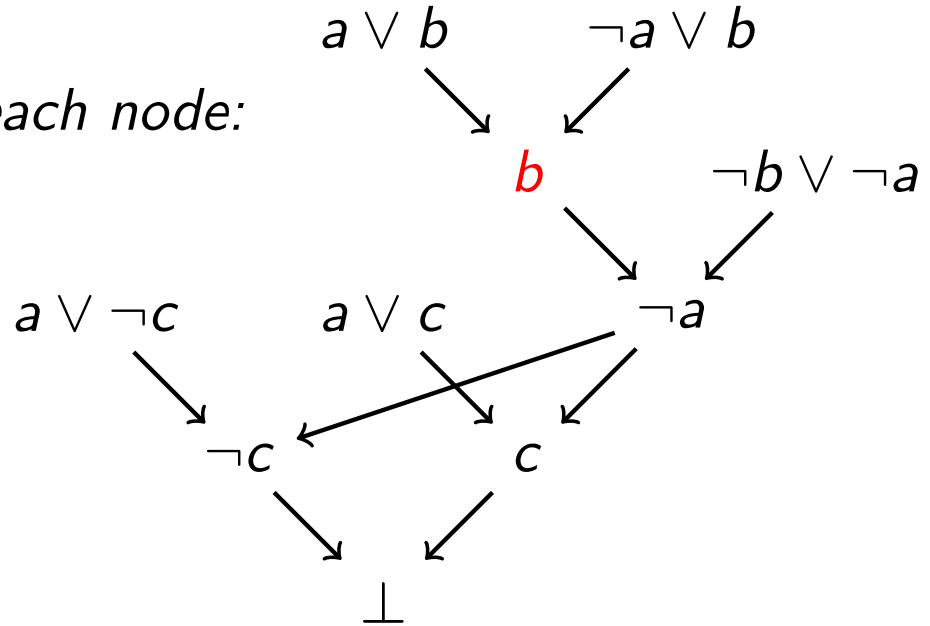
For a proof node v, expansion set $\rho(v)$ is the set of literals such that $\ell \in \rho(v)$ iff ℓ will be removed in all paths to $\bot$. ρ is defined as follows.

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

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

Exercise 5.4

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



Detecting redundant resolution (contd.)

Theorem 5.1

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

Exercise 5.5

a. *What is the complexity of computing ρ?*
b. *Prove $\rho(v) \supseteq$ literals in v*
c. *Given the above observation suggest an heuristic optimization.*

Topic 5.3

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 5.8

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

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

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 5.9

In the previous example,

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

End of Lecture 5