Propositional Proof Theory

Marijn J.H. Heule

THE UNIVERSITY OF
TEXAS
AT AUSTIN

IPA Course: Formal Methods June 12, 2018

Proofs of Unsatisfiability

Interference-Based Proofs

Propagation Redundancy

Satisfaction-Driven Clause Learning

Conclusions

Proofs of Unsatisfiability

Certifying Satisfiability and Unsatisfiability

- Certifying satisfiability of a formula is easy:

$$(x \vee y) \wedge (\bar{x} \vee \bar{y}) \wedge (z \vee \bar{z})$$

Certifying Satisfiability and Unsatisfiability

- Certifying satisfiability of a formula is easy:
  - Just consider a satisfying assignment: $x\bar{y}z$

$$(x \vee y) \wedge (\bar{x} \vee \bar{y}) \wedge (z \vee \bar{z})$$

  - We can easily check that the assignment is satisfying:
    Just check for every clause if it has a satisfied literal!

Certifying Satisfiability and Unsatisfiability

- Certifying satisfiability of a formula is easy:
  - Just consider a satisfying assignment: $x\bar{y}z$

$$(x \vee y) \wedge (\bar{x} \vee \bar{y}) \wedge (z \vee \bar{z})$$

  - We can easily check that the assignment is satisfying:
    Just check for every clause if it has a satisfied literal!

- Certifying unsatisfiability is not so easy:
  - If a formula has n variables, there are 2^n possible assignments.

  ➥ Checking whether every assignment falsifies the formula is costly.

  - More compact certificates of unsatisfiability are desirable.

    ➥ Proofs

What Is a Proof in SAT?

- In general, a proof is a string that certifies the unsatisfiability of a formula.
  - Proofs are efficiently (usually polynomial-time) checkable...

What Is a Proof in SAT?

- In general, a proof is a string that certifies the unsatisfiability of a formula.
  - Proofs are efficiently (usually polynomial-time) checkable...
    ... but can be of exponential size with respect to a formula.

What Is a Proof in SAT?

- In general, a proof is a string that certifies the unsatisfiability of a formula.
  - Proofs are efficiently (usually polynomial-time) checkable...
    ... but can be of exponential size with respect to a formula.

- Example: Resolution proofs
  - A resolution proof is a sequence $C_1, \ldots, C_m$ of clauses.
  - Every clause is either contained in the formula or derived from two earlier clauses via the resolution rule:

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D}$$

  - C_m is the empty clause (containing no literals), denoted by $\bot$.
  - There exists a resolution proof for every unsatisfiable formula.

Resolution Proofs

- Example: $F = (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z}) \wedge (x \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (u)$
- Resolution proof:
  $(\bar{x} \vee \bar{y} \vee z), (\bar{z}), (\bar{x} \vee \bar{y}), (x \vee \bar{y}), (\bar{y}), (\bar{u} \vee y), (\bar{u}), (u), \bot$

Resolution Proofs

- Example: $F = (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z}) \wedge (x \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (u)$
- Resolution proof:
  $(\bar{x} \vee \bar{y} \vee z), (\bar{z}), (\bar{x} \vee \bar{y}), (x \vee \bar{y}), (\bar{y}), (\bar{u} \vee y), (\bar{u}), (u), \bot$

$$\dfrac{\dfrac{\dfrac{\dfrac{\bar{x} \vee \bar{y} \vee z \qquad \bar{z}}{\bar{x} \vee \bar{y}} \qquad x \vee \bar{y}}{\bar{y}} \qquad \bar{u} \vee y}{\bar{u}} \qquad u}{\bot}$$

Resolution Proofs

- Example: $F = (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z}) \wedge (x \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (u)$
- Resolution proof:
  $(\bar{x} \vee \bar{y} \vee z), (\bar{z}), (\bar{x} \vee \bar{y}), (x \vee \bar{y}), (\bar{y}), (\bar{u} \vee y), (\bar{u}), (u), \bot$

$$\dfrac{\dfrac{\dfrac{\dfrac{\bar{x} \vee \bar{y} \vee z \qquad \bar{z}}{\bar{x} \vee \bar{y}} \qquad x \vee \bar{y}}{\bar{y}} \qquad \bar{u} \vee y}{\bar{u}} \qquad u}{\bot}$$

- Drawbacks of resolution:
  - For many seemingly simple formulas, there are only resolution proofs of exponential size.
  - State-of-the-art solving techniques are not succinctly expressible.

Interference-Based Proofs

Traditional Proofs vs. Interference-Based Proofs

- In traditional proof systems, everything that is inferred, is logically implied by the premises.

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D}\ (\text{res}) \qquad \frac{A \qquad A \to B}{B}\ (\text{mp})$$

Traditional Proofs vs. Interference-Based Proofs

- In traditional proof systems, everything that is inferred, is logically implied by the premises.

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D}\ (\text{res}) \qquad \frac{A \qquad A \to B}{B}\ (\text{mp})$$

➥ Inference rules reason about the presence of facts.

- If certain premises are present, infer the conclusion.

Traditional Proofs vs. Interference-Based Proofs

- In traditional proof systems, everything that is inferred, is logically implied by the premises.

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D}\ (\text{res}) \qquad \frac{A \qquad A \to B}{B}\ (\text{mp})$$

↪ Inference rules reason about the presence of facts.

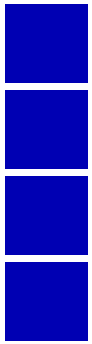
- If certain premises are present, infer the conclusion.

- Different approach: Allow not only implied conclusions.
  - Require only that the addition of facts preserves satisfiability.
  - Reason also about the absence of facts.

  ↪ This leads to interference-based proof systems.

Interference-Based Proofs

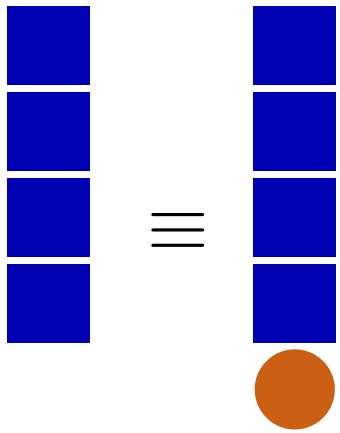
Formula



Proof

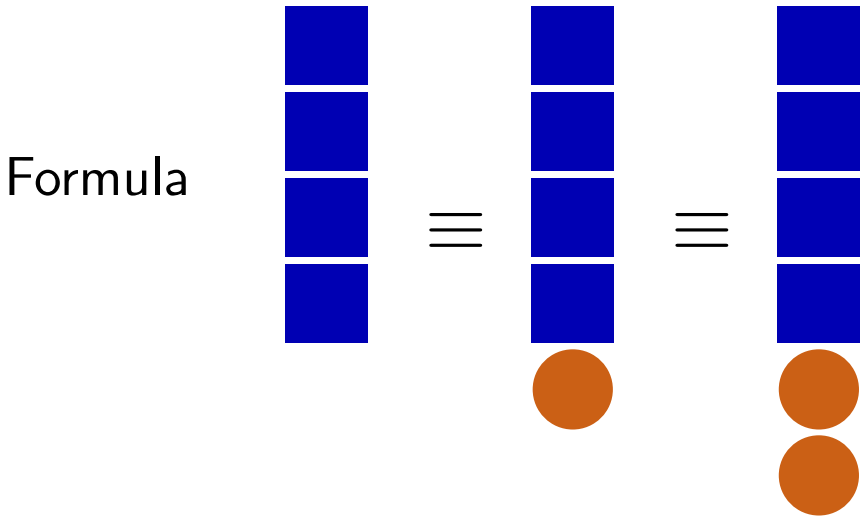
Interference-Based Proofs

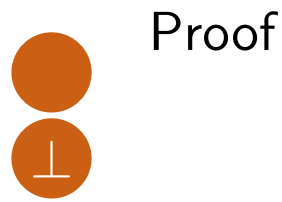
Formula



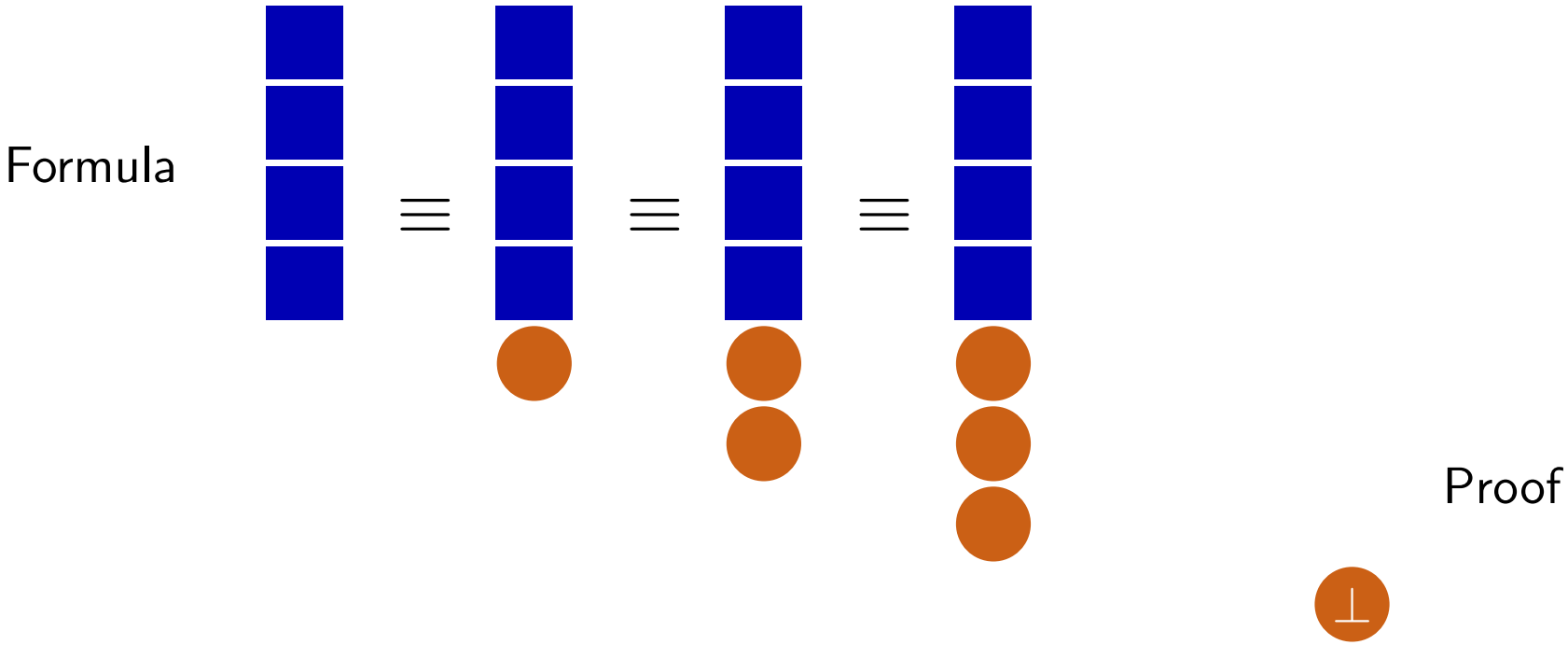
Proof

Interference-Based Proofs

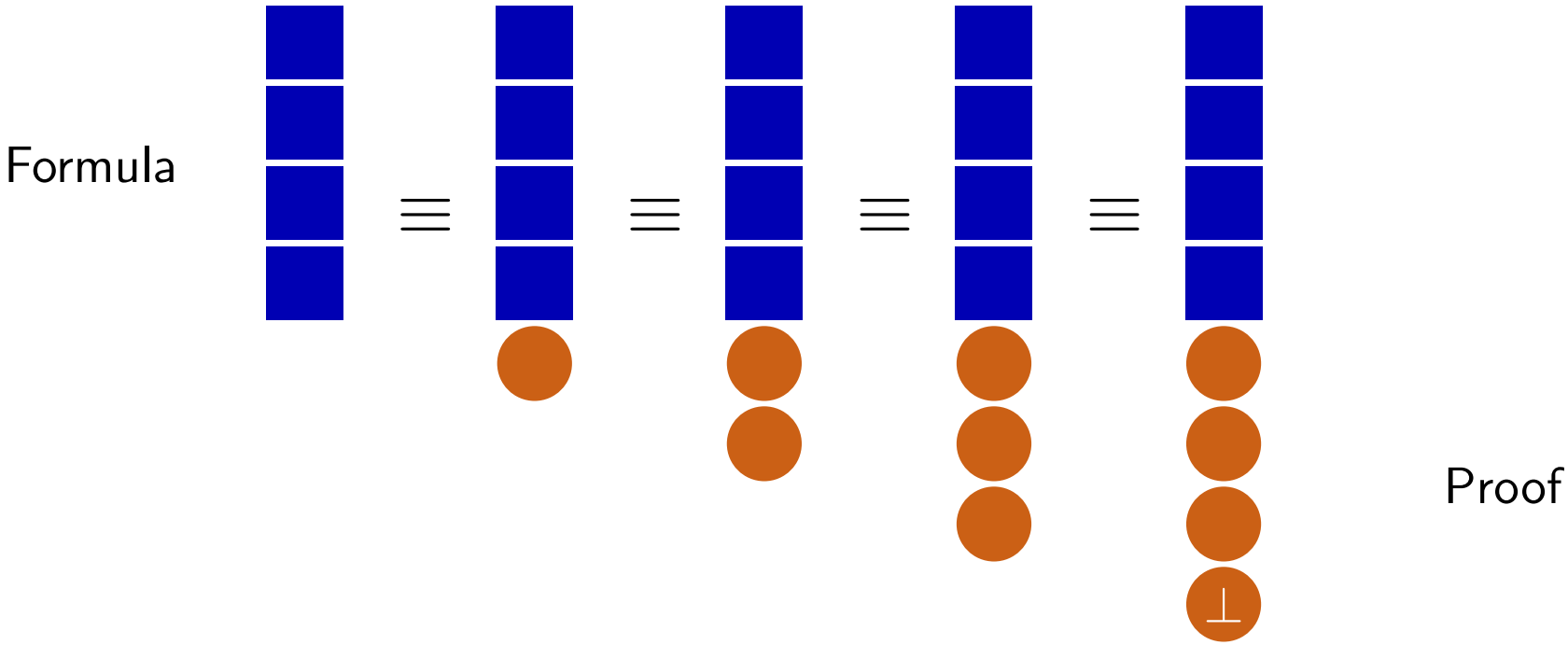




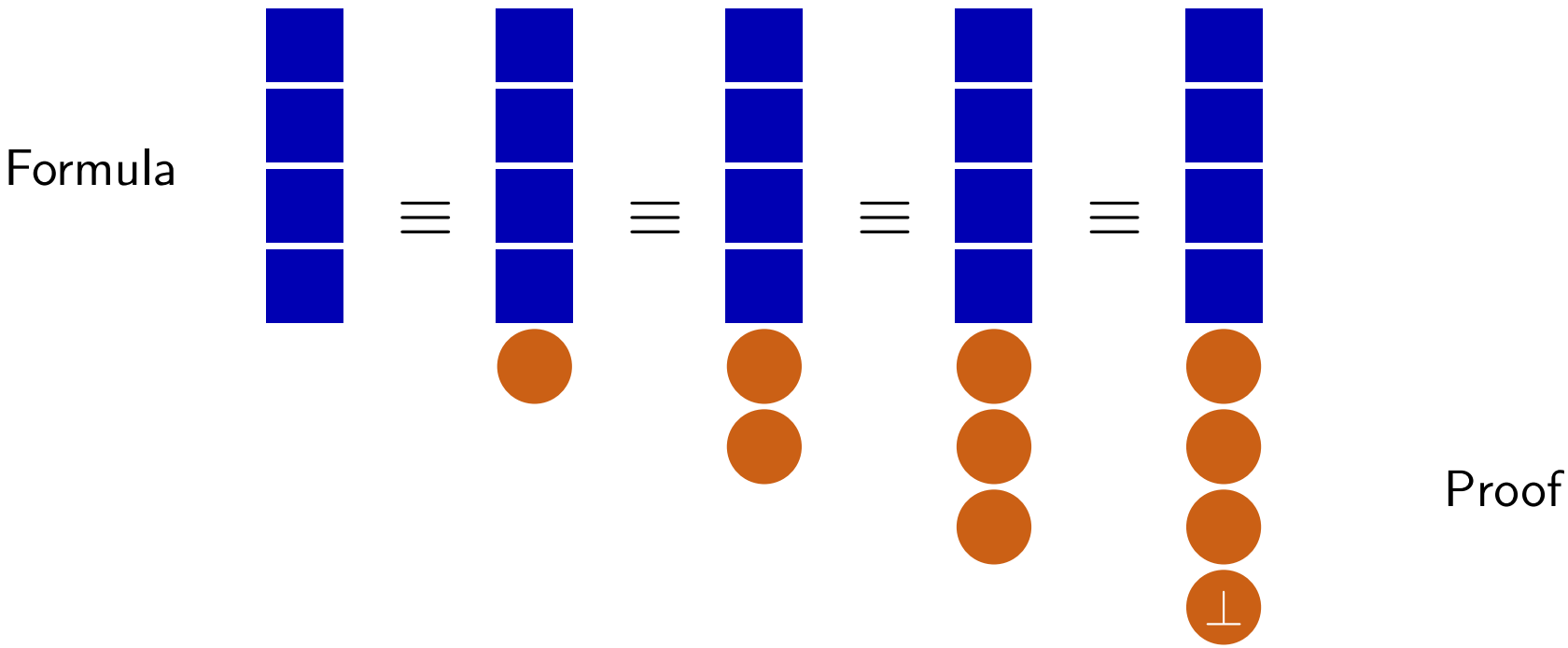
Interference-Based Proofs



Interference-Based Proofs

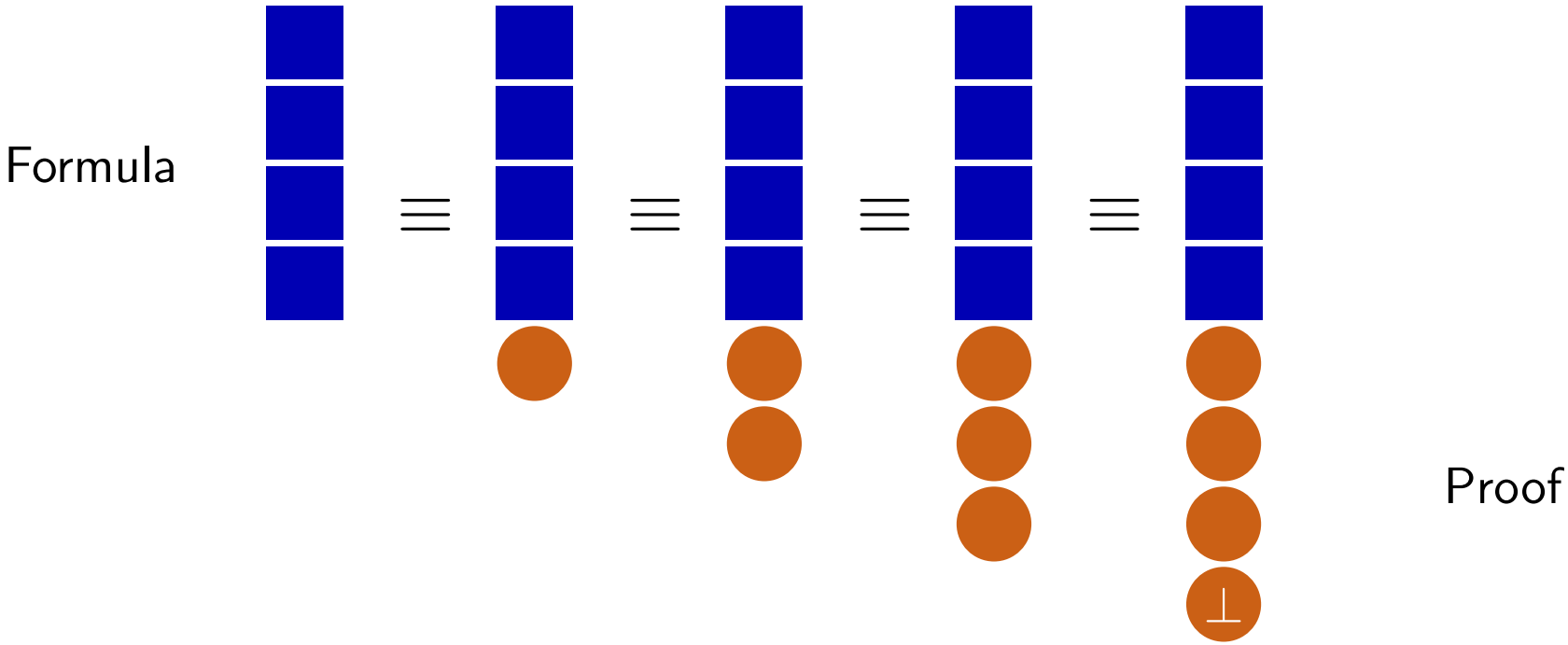


Interference-Based Proofs



- Checking whether additions preserve satisfiability should be efficient.
- Clauses whose addition preserves satisfiability are called redundant.

Interference-Based Proofs



- Checking whether additions preserve satisfiability should be efficient.
- Clauses whose addition preserves satisfiability are called redundant.

↪ Idea: Allow only the addition of clauses that fulfill an efficiently checkable redundancy criterion.

DRAT: An Interference-Based Proof System

- Popular example of an interference-based proof system: DRAT
- DRAT allows the addition of so-called resolution asymmetric tautologies (RATs) to a formula (whatever that means).
  - It can be efficiently checked if a clause is a RAT.
  - RATs are not necessarily implied by the formula.
  - But RATs are redundant: their addition preserves satisfiability.
  - A RAT check involves reasoning about the absence of facts.
    - A clause is a RAT w.r.t. a formula if the formula contains no clause such that ...

Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal x in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{x}}$, the resolvent $(C \setminus \{x\}) \cup (D \setminus \{\bar{x}\})$ obtained from resolving C and D on l is a tautology.

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal x in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{x}}$, the resolvent $(C \setminus \{x\}) \cup (D \setminus \{\bar{x}\})$ obtained from resolving C and D on l is a tautology.

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Example

Consider the formula $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.
First clause is not blocked.
Second clause is blocked by both a *and* $\bar{c}$.
Third clause is blocked by c

Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal x in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{x}}$, the resolvent $(C \setminus \{x\}) \cup (D \setminus \{\bar{x}\})$ obtained from resolving C and D on l is a tautology.

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Example

Consider the formula $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.
First clause is not blocked.
Second clause is blocked by both a *and* $\bar{c}$.
Third clause is blocked by c

Proposition

Adding or removing a blocked clause preserves satisfiability.

Blocked Clause Elimination (BCE)

Definition (BCE)

While there is a blocked clause C in a CNF F, remove C from F.

Example

Consider $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.
After removing either $(a \vee \bar{b} \vee \bar{c})$ *or* $(\bar{a} \vee c)$, *the clause* $(a \vee b)$ *becomes blocked (no clause with either* $\bar{b}$ *or* $\bar{a}$*).*

An extreme case in which BCE removes all clauses!

Blocked Clause Elimination (BCE)

Definition (BCE)

While there is a blocked clause C in a CNF F, remove C from F.

Example

Consider $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.
After removing either $(a \vee \bar{b} \vee \bar{c})$ *or* $(\bar{a} \vee c)$, *the clause* $(a \vee b)$ *becomes blocked (no clause with either* $\bar{b}$ *or* $\bar{a}$*).*

An extreme case in which BCE removes all clauses!

Proposition

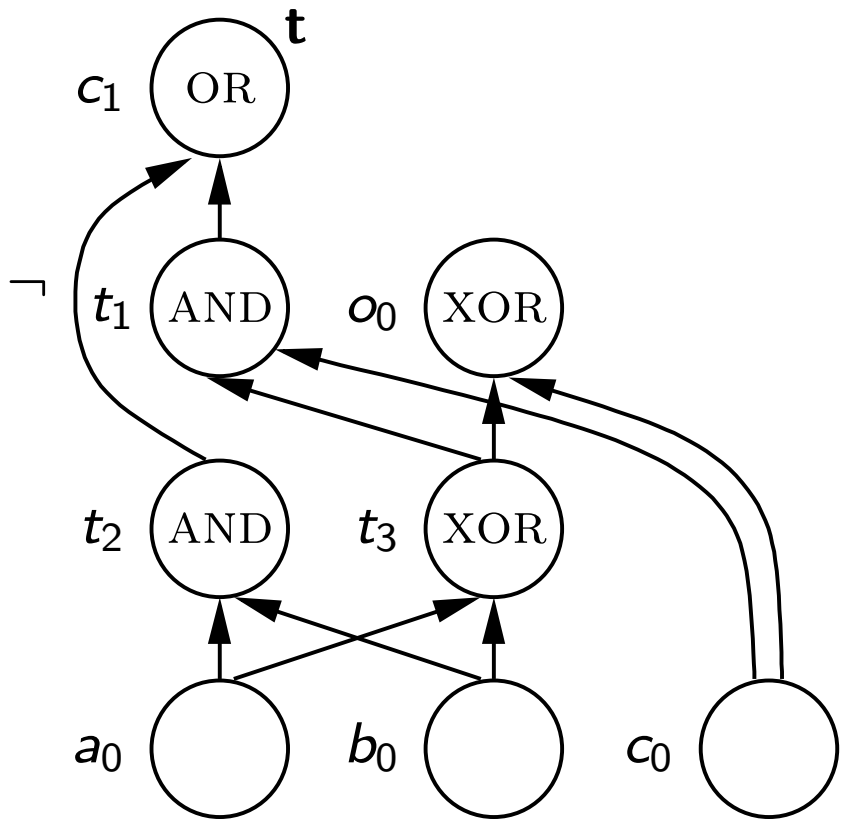
BCE is confluent, i.e., has a unique fixpoint

- Blocked clauses stay blocked w.r.t. removal

BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

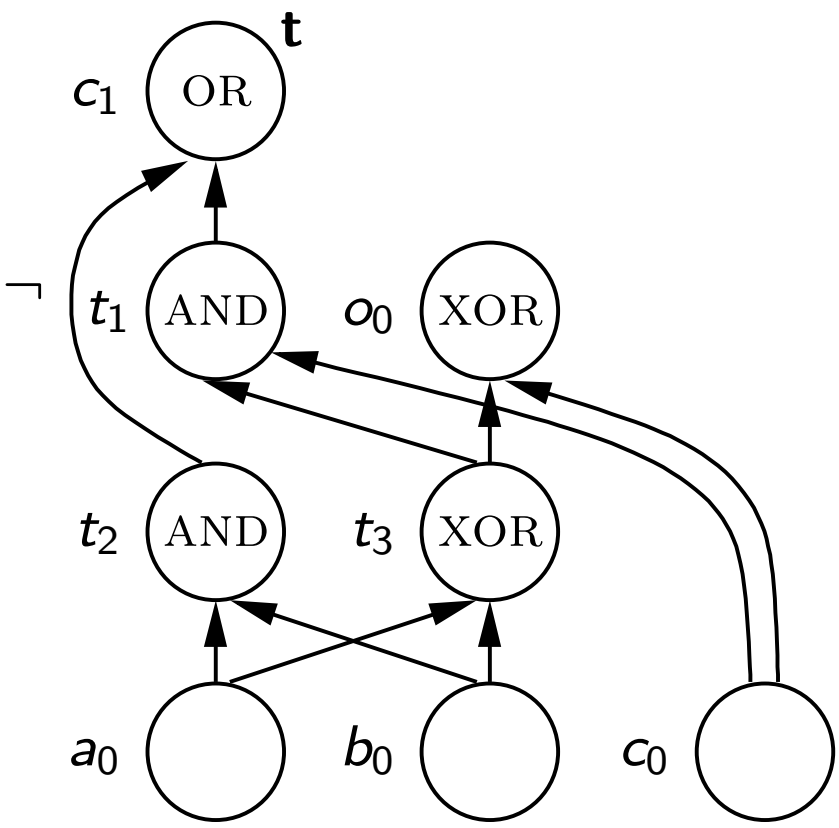
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
$(c_1 \vee \bar{t}_1)$
$(c_1 \vee t_2)$

$(\bar{o}_0 \vee t_3 \vee c_0)$
$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$
$(o_0 \vee t_3 \vee \bar{c}_0)$
$(o_0 \vee \bar{t}_3 \vee c_0)$

$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
$(\bar{t}_2 \vee a_0)$
$(\bar{t}_2 \vee b_0)$

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
$(t_3 \vee a_0 \vee \bar{b}_0)$
$(t_3 \vee \bar{a}_0 \vee b_0)$



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

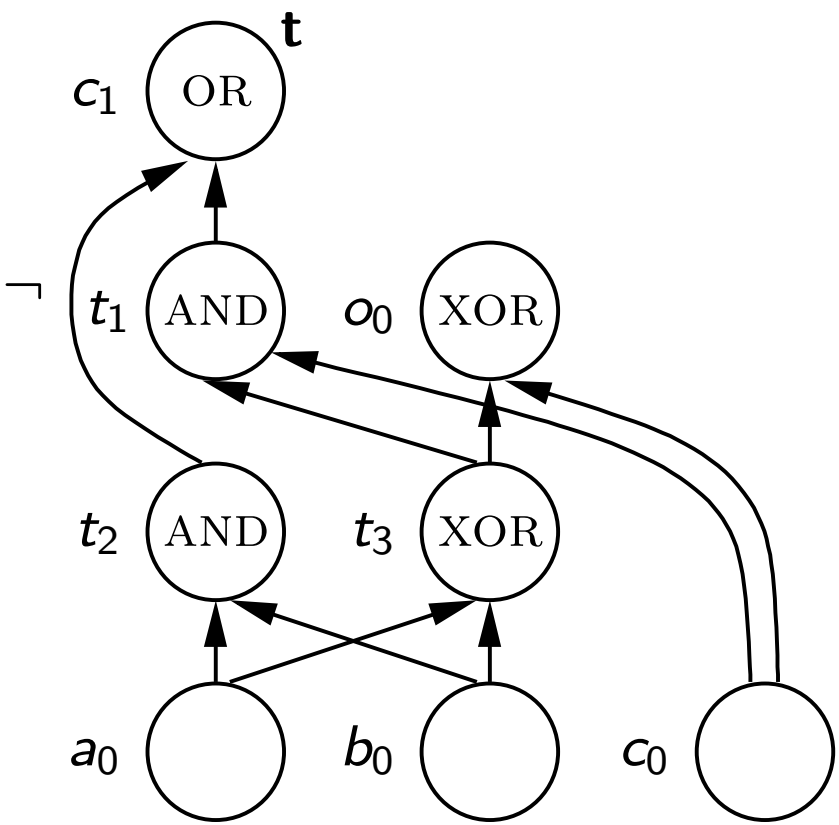
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
~~$(c_1 \vee \bar{t}_1)$~~
~~$(c_1 \vee t_2)$~~

$(\bar{o}_0 \vee t_3 \vee c_0)$
$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$
$(o_0 \vee t_3 \vee \bar{c}_0)$
$(o_0 \vee \bar{t}_3 \vee c_0)$

$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
$(\bar{t}_2 \vee a_0)$
$(\bar{t}_2 \vee b_0)$

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
$(t_3 \vee a_0 \vee \bar{b}_0)$
$(t_3 \vee \bar{a}_0 \vee b_0)$



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

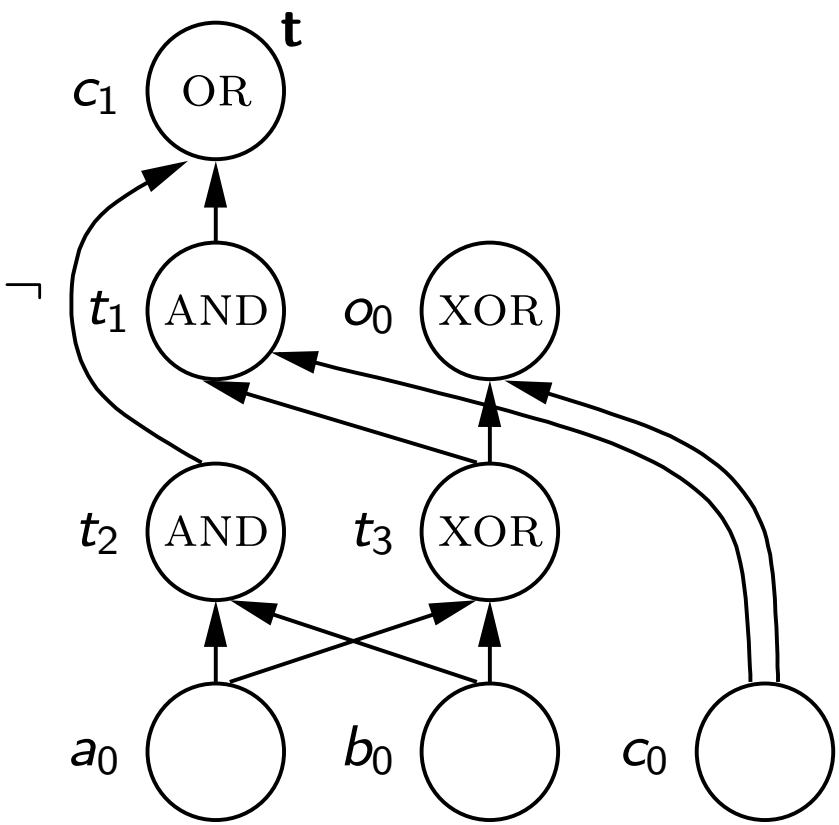
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
~~$(c_1 \vee \bar{t}_1)$~~
~~$(c_1 \vee t_2)$~~

~~$(\bar{o}_0 \vee t_3 \vee c_0)$~~
~~$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee t_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee \bar{t}_3 \vee c_0)$~~

$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
$(\bar{t}_2 \vee a_0)$
$(\bar{t}_2 \vee b_0)$

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
$(t_3 \vee a_0 \vee \bar{b}_0)$
$(t_3 \vee \bar{a}_0 \vee b_0)$



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

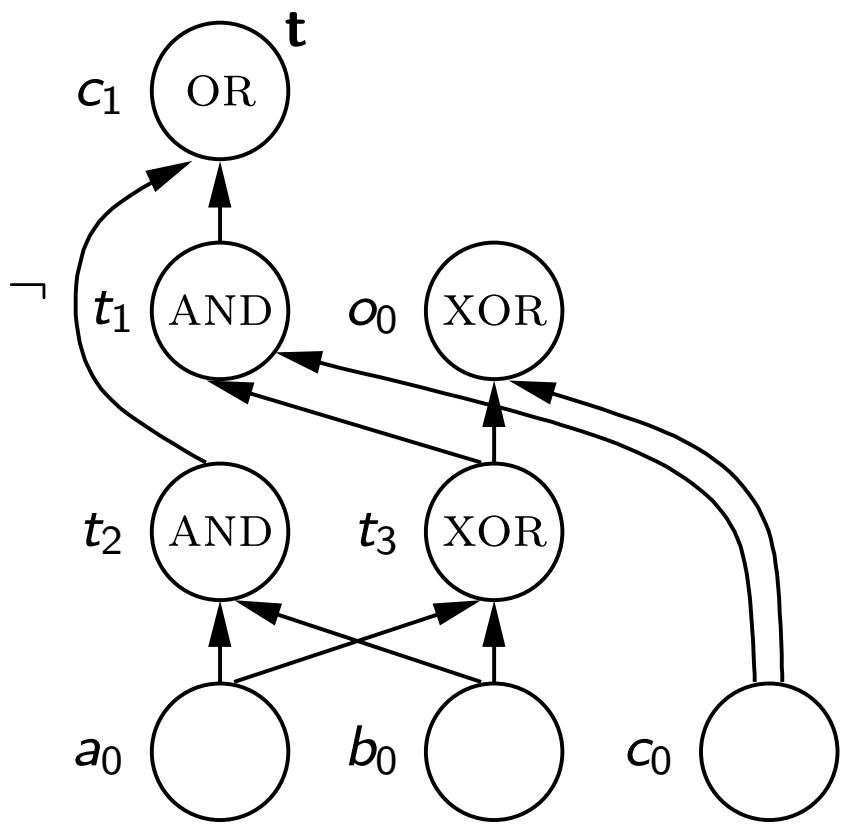
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
~~$(c_1 \vee \bar{t}_1)$~~
~~$(c_1 \vee t_2)$~~

~~$(\bar{o}_0 \vee t_3 \vee c_0)$~~
~~$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee t_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee \bar{t}_3 \vee c_0)$~~

~~$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$~~
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
$(\bar{t}_2 \vee a_0)$
$(\bar{t}_2 \vee b_0)$

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
$(t_3 \vee a_0 \vee \bar{b}_0)$
$(t_3 \vee \bar{a}_0 \vee b_0)$



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

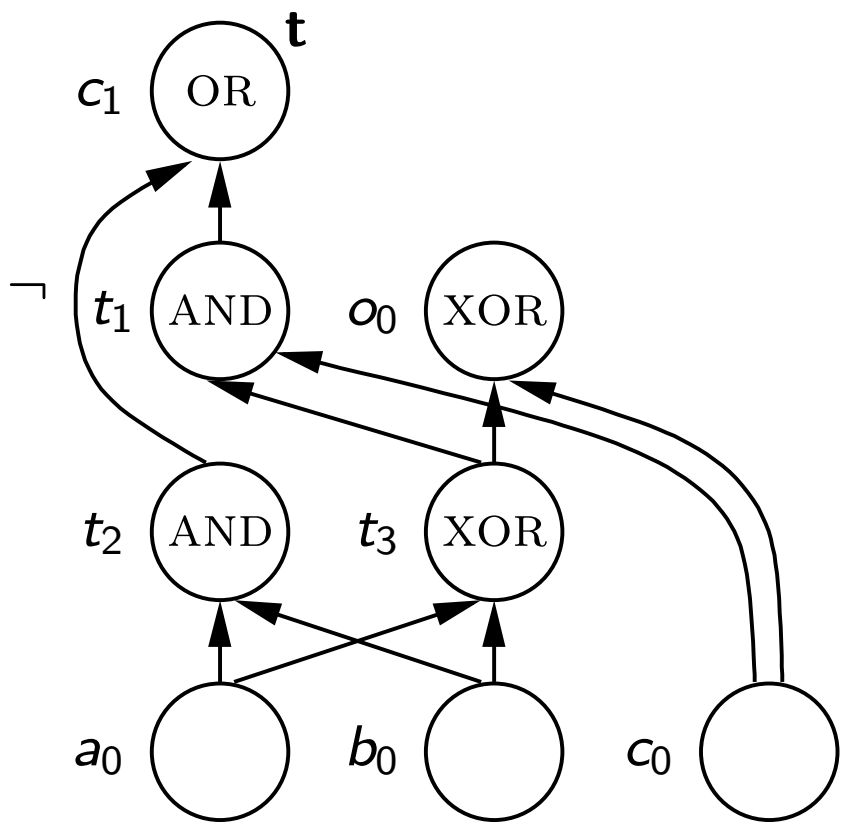
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
~~$(c_1 \vee \bar{t}_1)$~~
~~$(c_1 \vee t_2)$~~

~~$(\bar{o}_0 \vee t_3 \vee c_0)$~~
~~$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee t_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee \bar{t}_3 \vee c_0)$~~

~~$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$~~
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
~~$(\bar{t}_2 \vee a_0)$~~
~~$(\bar{t}_2 \vee b_0)$~~

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
$(t_3 \vee a_0 \vee \bar{b}_0)$
$(t_3 \vee \bar{a}_0 \vee b_0)$



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

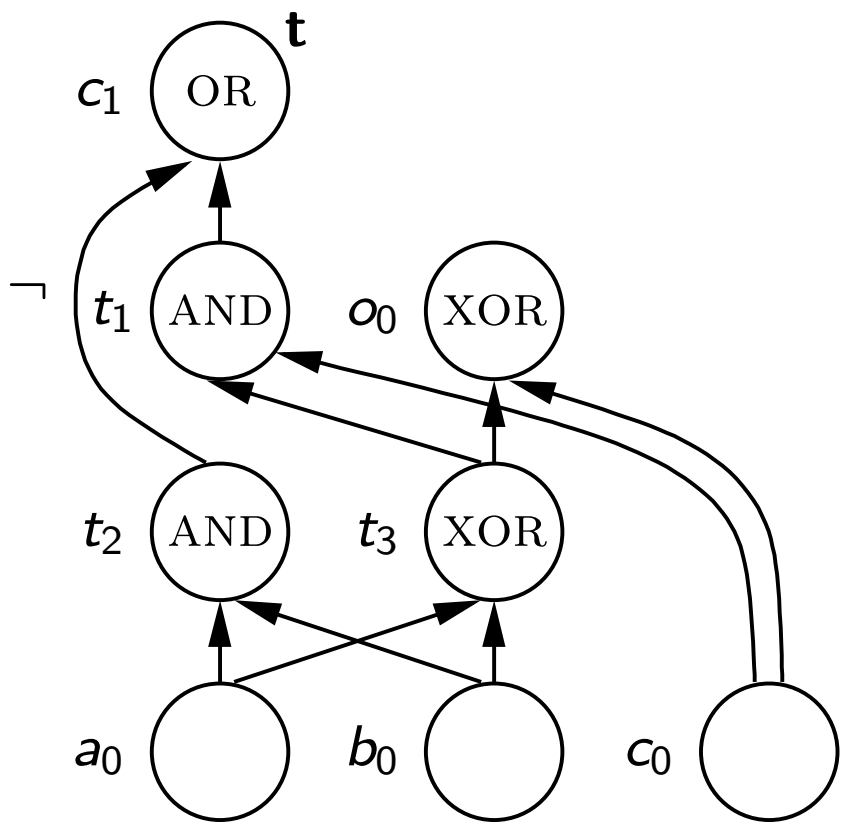
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
~~$(c_1 \vee \bar{t}_1)$~~
~~$(c_1 \vee t_2)$~~

~~$(\bar{o}_0 \vee t_3 \vee c_0)$~~
~~$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee t_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee \bar{t}_3 \vee c_0)$~~

~~$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$~~
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
~~$(\bar{t}_2 \vee a_0)$~~
~~$(\bar{t}_2 \vee b_0)$~~

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
~~$(t_3 \vee a_0 \vee \bar{b}_0)$~~
~~$(t_3 \vee \bar{a}_0 \vee b_0)$~~



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

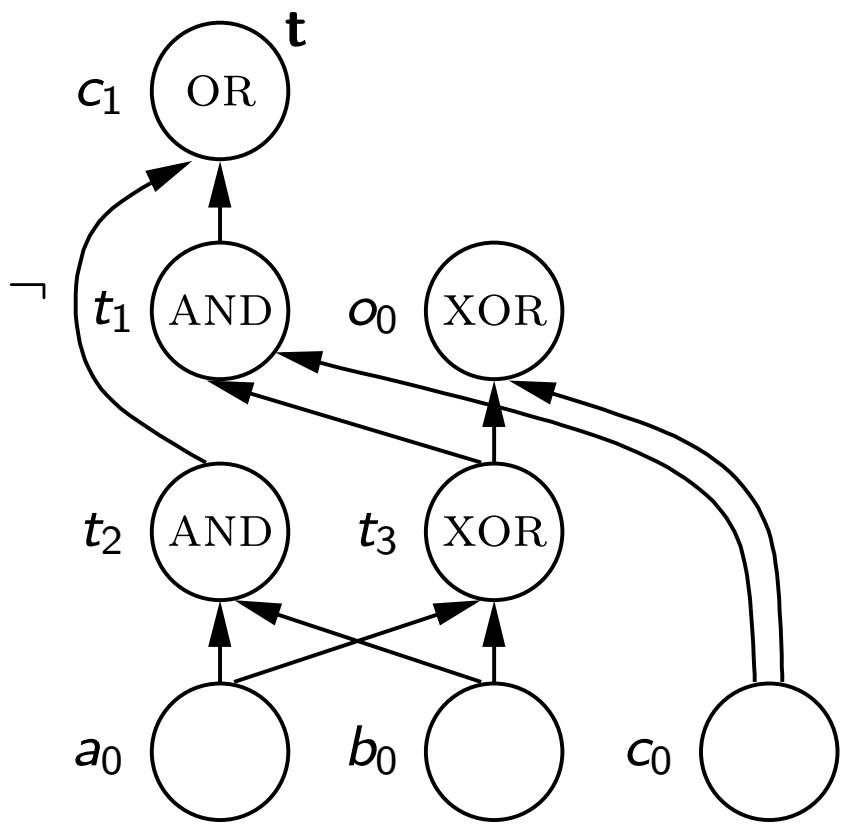
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
~~$(c_1 \vee \bar{t}_1)$~~
~~$(c_1 \vee t_2)$~~

~~$(\bar{o}_0 \vee t_3 \vee c_0)$~~
~~$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee t_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee \bar{t}_3 \vee c_0)$~~

~~$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$~~
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

~~$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$~~
~~$(\bar{t}_2 \vee a_0)$~~
~~$(\bar{t}_2 \vee b_0)$~~

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
~~$(t_3 \vee a_0 \vee \bar{b}_0)$~~
~~$(t_3 \vee \bar{a}_0 \vee b_0)$~~



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

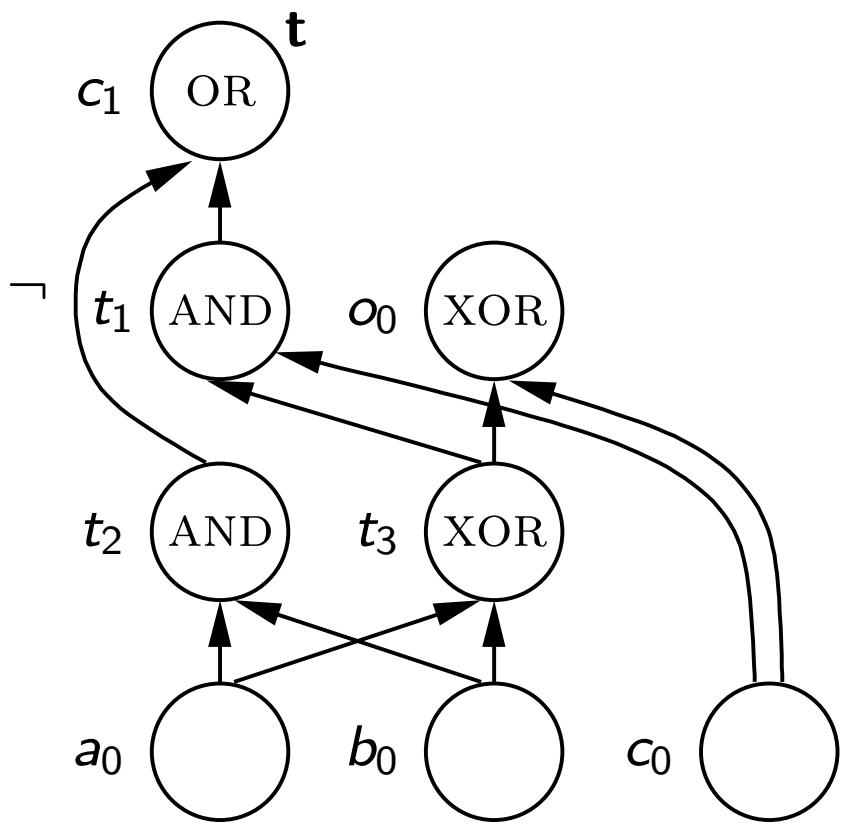
~~$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$~~
~~$(c_1 \vee \bar{t}_1)$~~
~~$(c_1 \vee t_2)$~~

~~$(\bar{o}_0 \vee t_3 \vee c_0)$~~
~~$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee t_3 \vee \bar{c}_0)$~~
~~$(o_0 \vee \bar{t}_3 \vee c_0)$~~

~~$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$~~
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

~~$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$~~
~~$(\bar{t}_2 \vee a_0)$~~
~~$(\bar{t}_2 \vee b_0)$~~

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
~~$(t_3 \vee a_0 \vee \bar{b}_0)$~~
~~$(t_3 \vee \bar{a}_0 \vee b_0)$~~



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

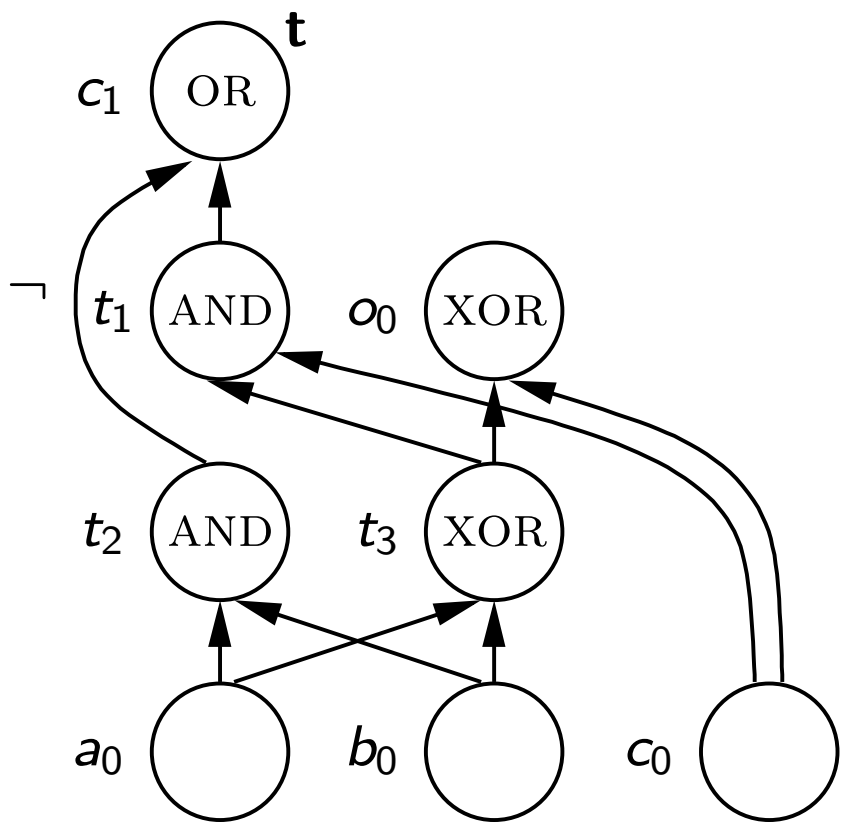
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
$(c_1 \vee \bar{t}_1)$
$(c_1 \vee t_2)$

$(\bar{o}_0 \vee t_3 \vee c_0)$
$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$
$(o_0 \vee t_3 \vee \bar{c}_0)$
$(o_0 \vee \bar{t}_3 \vee c_0)$

$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
$(\bar{t}_2 \vee a_0)$
$(\bar{t}_2 \vee b_0)$

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
$(t_3 \vee a_0 \vee \bar{b}_0)$
$(t_3 \vee \bar{a}_0 \vee b_0)$



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

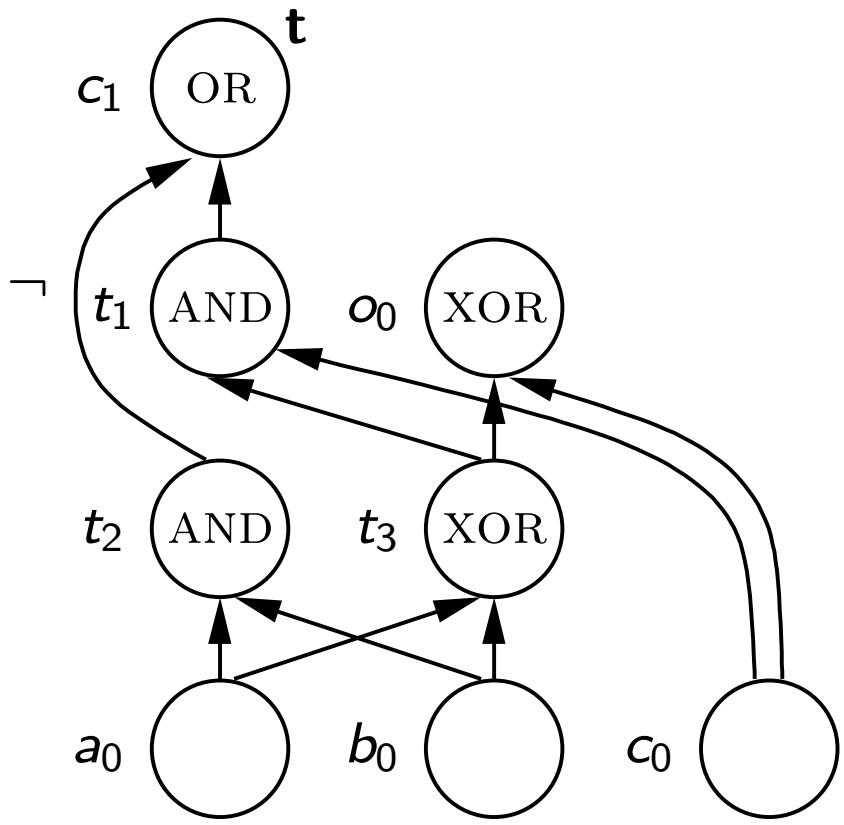
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
$(c_1 \vee \bar{t}_1)$
$(c_1 \vee t_2)$

$(\bar{o}_0 \vee t_3 \vee c_0)$
$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$
$(o_0 \vee t_3 \vee \bar{c}_0)$
$(o_0 \vee \bar{t}_3 \vee c_0)$

$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
$(\bar{t}_2 \vee a_0)$
$(\bar{t}_2 \vee b_0)$

$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
$(t_3 \vee a_0 \vee \bar{b}_0)$
$(t_3 \vee \bar{a}_0 \vee b_0)$



BCE very effective on circuits [JärvisaloBiereHeule'10]

BCE converts the Tseitin encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence, etc.

Example of circuit simplification by BCE on Tseitin encoding

(c_1)

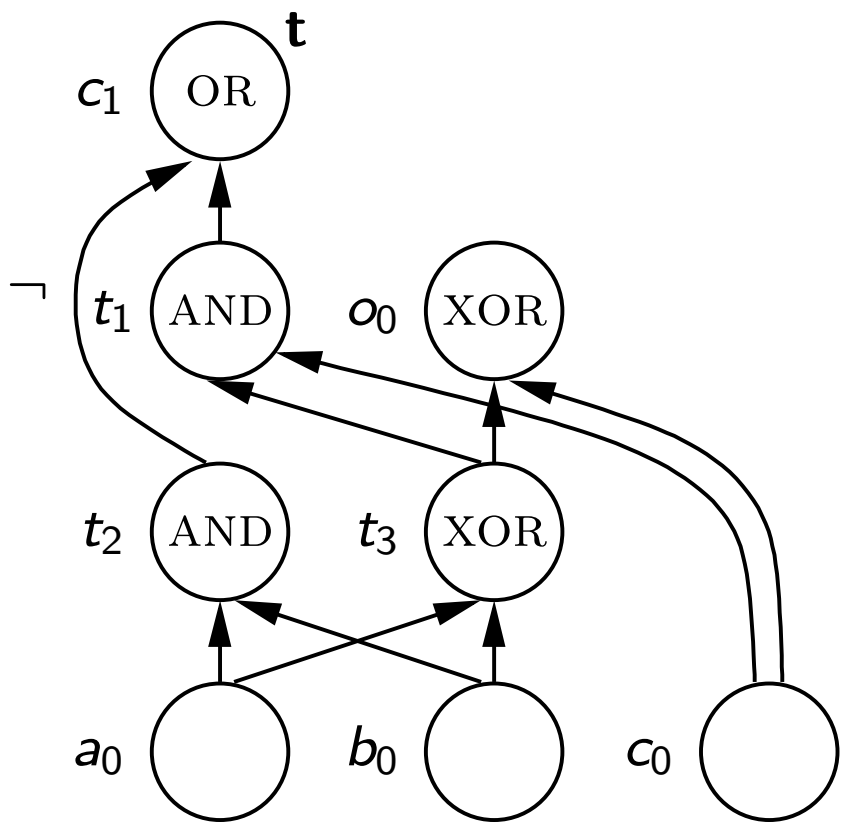
$(\bar{c}_1 \vee t_1 \vee \bar{t}_2)$
$(c_1 \vee \bar{t}_1)$
$(c_1 \vee t_2)$

$(\bar{o}_0 \vee t_3 \vee c_0)$
$(\bar{o}_0 \vee \bar{t}_3 \vee \bar{c}_0)$
$(o_0 \vee t_3 \vee \bar{c}_0)$
$(o_0 \vee \bar{t}_3 \vee c_0)$

$(t_1 \vee \bar{t}_3 \vee \bar{c}_0)$
$(\bar{t}_1 \vee t_3)$
$(\bar{t}_1 \vee c_0)$

$(t_2 \vee \bar{a}_0 \vee \bar{b}_0)$
$(\bar{t}_2 \vee a_0)$
$(\bar{t}_2 \vee b_0)$

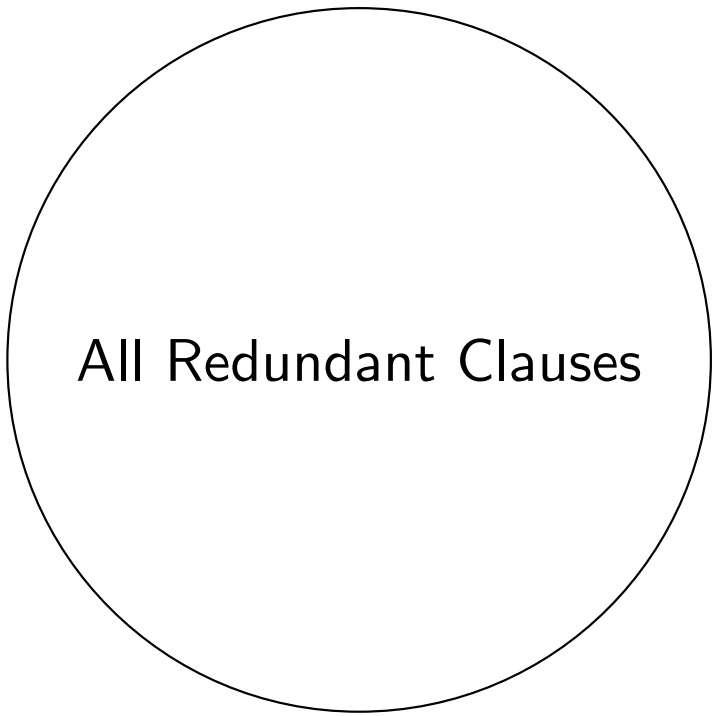
$(\bar{t}_3 \vee a_0 \vee b_0)$
$(\bar{t}_3 \vee \bar{a}_0 \vee \bar{b}_0)$
$(t_3 \vee a_0 \vee \bar{b}_0)$
$(t_3 \vee \bar{a}_0 \vee b_0)$



Propagation Redundancy

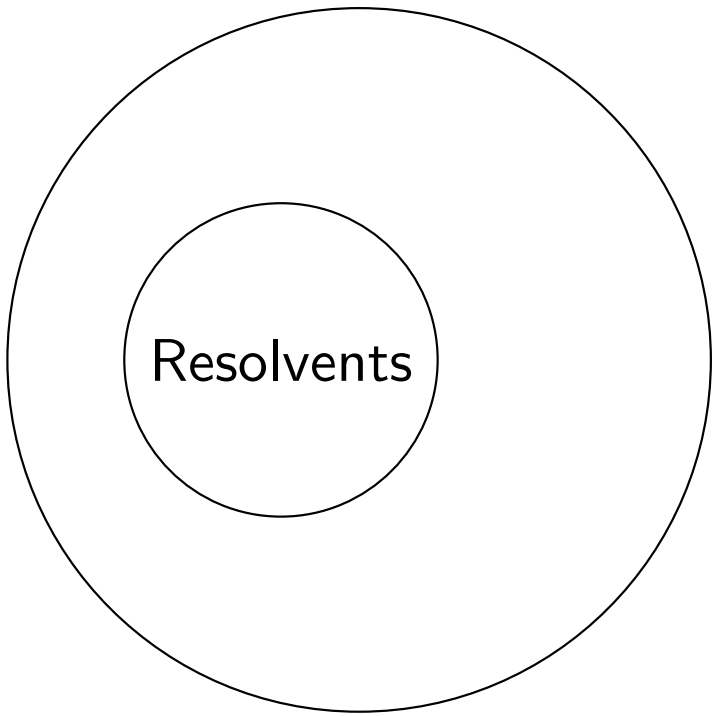
Redundant Clauses

- Strong proof systems allow addition of many redundant clauses.



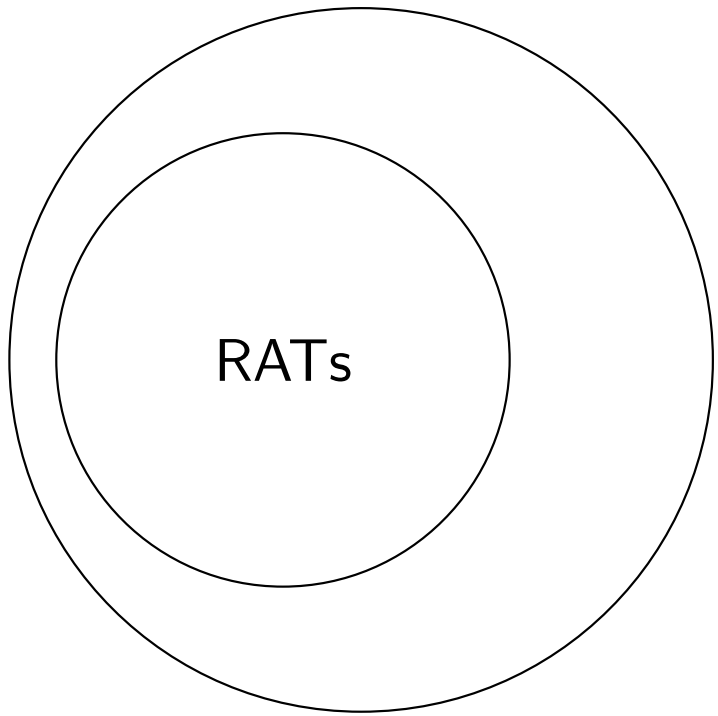
Redundant Clauses

- Strong proof systems allow addition of many redundant clauses.



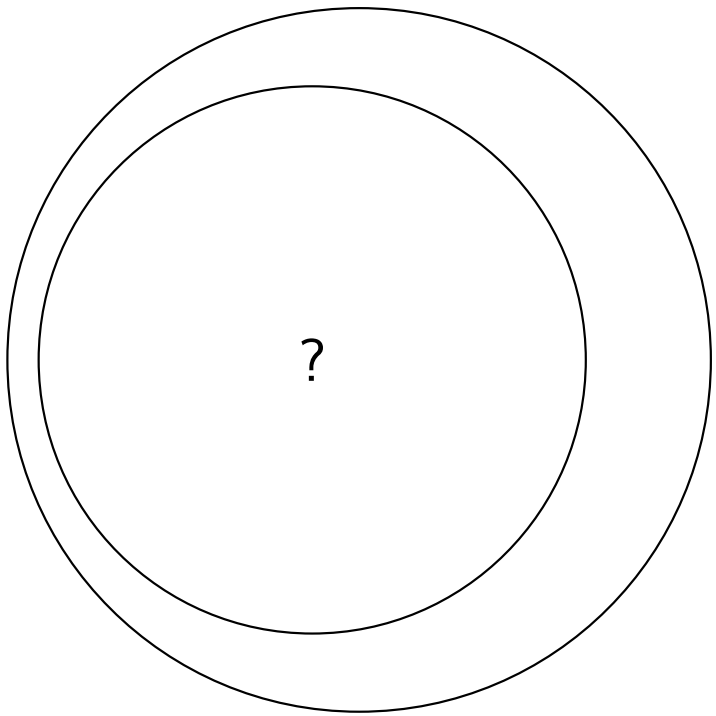
Redundant Clauses

- Strong proof systems allow addition of many redundant clauses.



Redundant Clauses

- Strong proof systems allow addition of many redundant clauses.

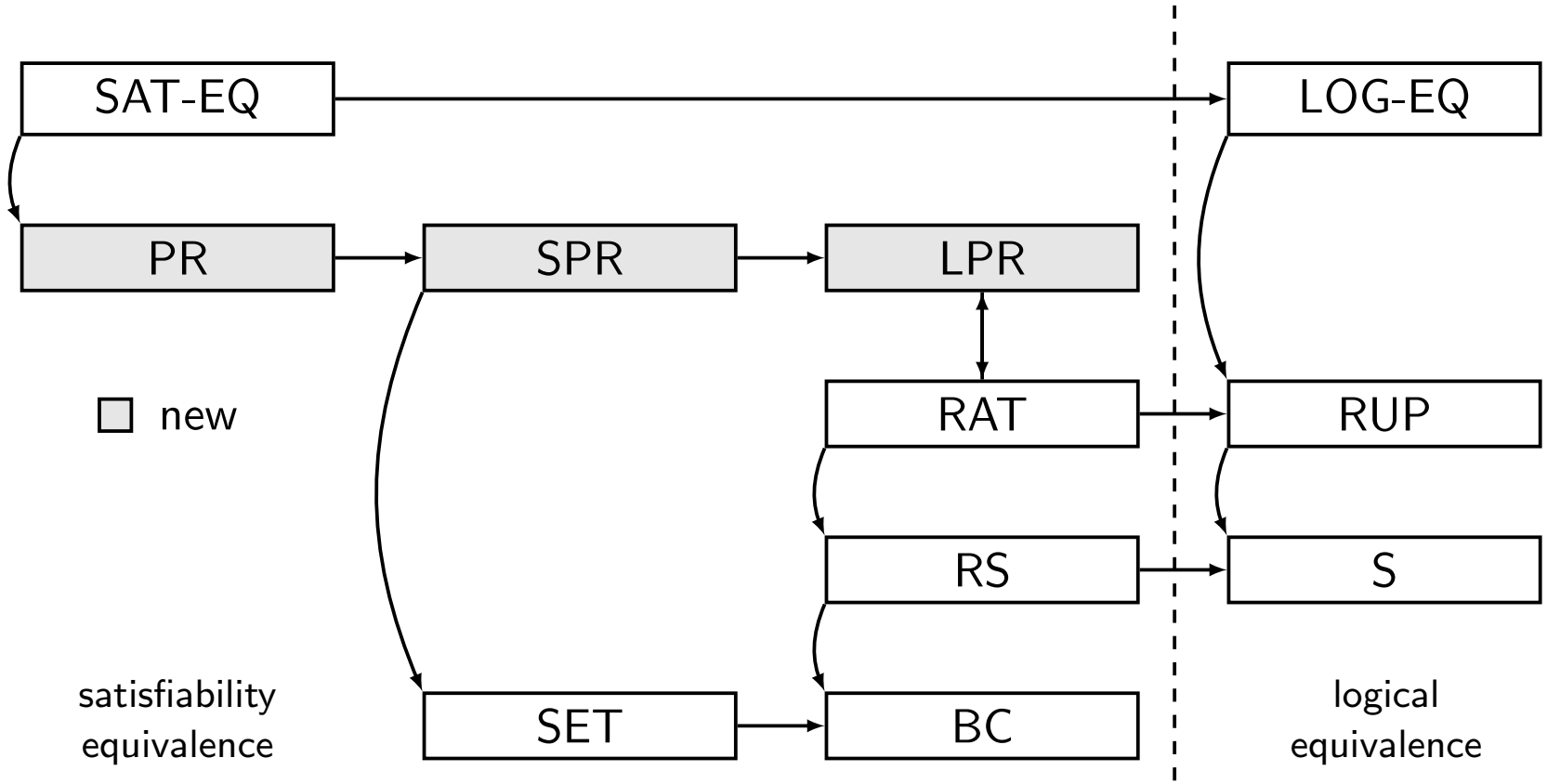


- Are stronger redundancy notions still efficiently checkable?

New Propositional Proof Systems

- We introduced new clause-redundancy notions:
  - Propagation-redundant (PR) clauses
  - Set-propagation-redundant (SPR) clauses
  - Literal-propagation-redundant (LPR) clauses
- LPR clauses coincide with RAT.
- SPR clauses strictly generalize RATs.
- PR clauses strictly generalize SPR clauses.
- The redundancy notions provide the basis for new proof systems.

New Landscape of Redundancy Notions



Stronger Proof Systems: What Are They Good For?

- The new proof systems can give short proofs of formulas that are considered hard.
- We have short SPR and PR proofs for the well-known pigeon hole formulas (linear in the size of the input).
  - Pigeon hole formulas have only exponential-size resolution proofs.
  - If the addition of new variables via definitions is allowed, there are polynomial-size proofs.
    - So-called extended resolution proofs.
- Our proofs do not require new variables.
  - ➥ Search space of possible clauses is finite.
  - ➥ Makes search for such clauses easier.

Redundancy as an Implication

A formula G is at least as satisfiable as a formula F if $F \models G$.

Given a formula F and assignment α, we denote with $F|_\alpha$ the reduced formula after removing from F all clauses satisfied by α and all literals falsified by α.

Theorem

Let F be a formula, C a clause, and α the smallest assignment that falsifies C. Then, C is redundant w.r.t. F iff there exists an assignment ω such that 1) ω satisfies C; and 2) $F|_\alpha \models F|_\omega$.

This is the strongest notion of redundancy. However, it cannot be checked in polynomial time (assuming P $\neq$ NP), unless bounded.

Checking Redundancy Using Unit Propagation

- Unit propagation (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula, C a clause, and α the smallest assignment that falsifies C. C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F|_\alpha$ results in a conflict.
- Implied by UP is used in SAT solvers to determine redundancy of learned clauses and therefore $\vdash_1$ is a natural restriction of $\models$.
- We bound $F|_\alpha \models F|_\omega$ by $F|_\alpha \vdash_1 F|_\omega$.
- Example: $F = (x \vee y \vee z) \wedge (\overline{x} \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{y} \vee z)$ and $G = (z)$. Observe that $F \models G$, but that $F \not\vdash_1 G$.

Hand-crafted PR Proofs of Pigeon Hole Formulas

We manually constructed PR proofs of the famous pigeon hole formulas and the two-pigeons-per-hole family.

- The proofs consist only of binary and unit clauses.
- Only original variables appear in the proof.
- All proofs are linear in the size of the formula.

↪ Our proofs are smaller than Cook's extended resolution proofs.

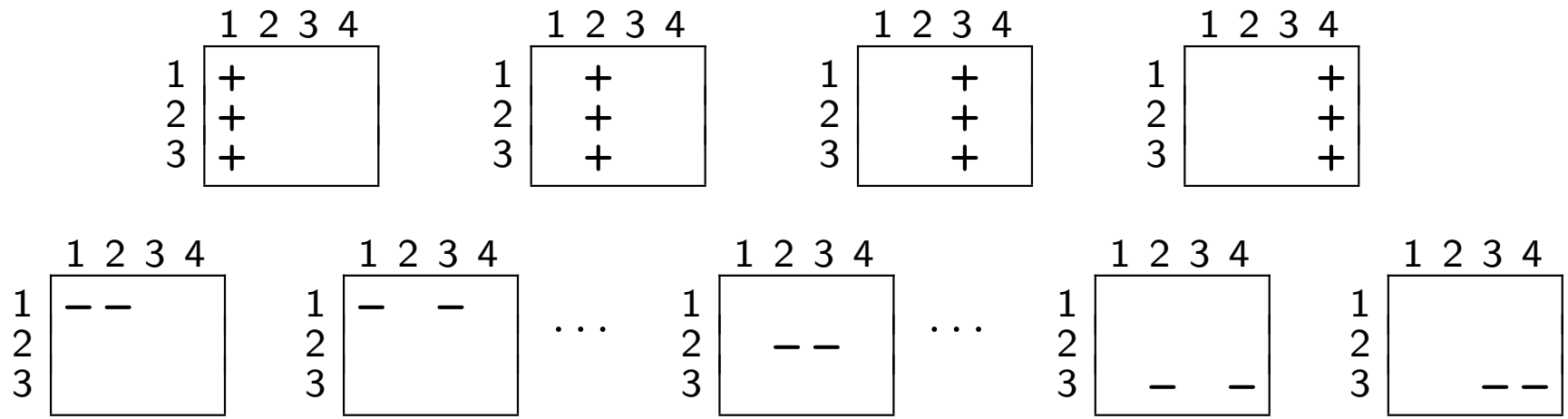
- All resolution proofs of these formulas are exponential in size.

Pigeon Hole Formulas

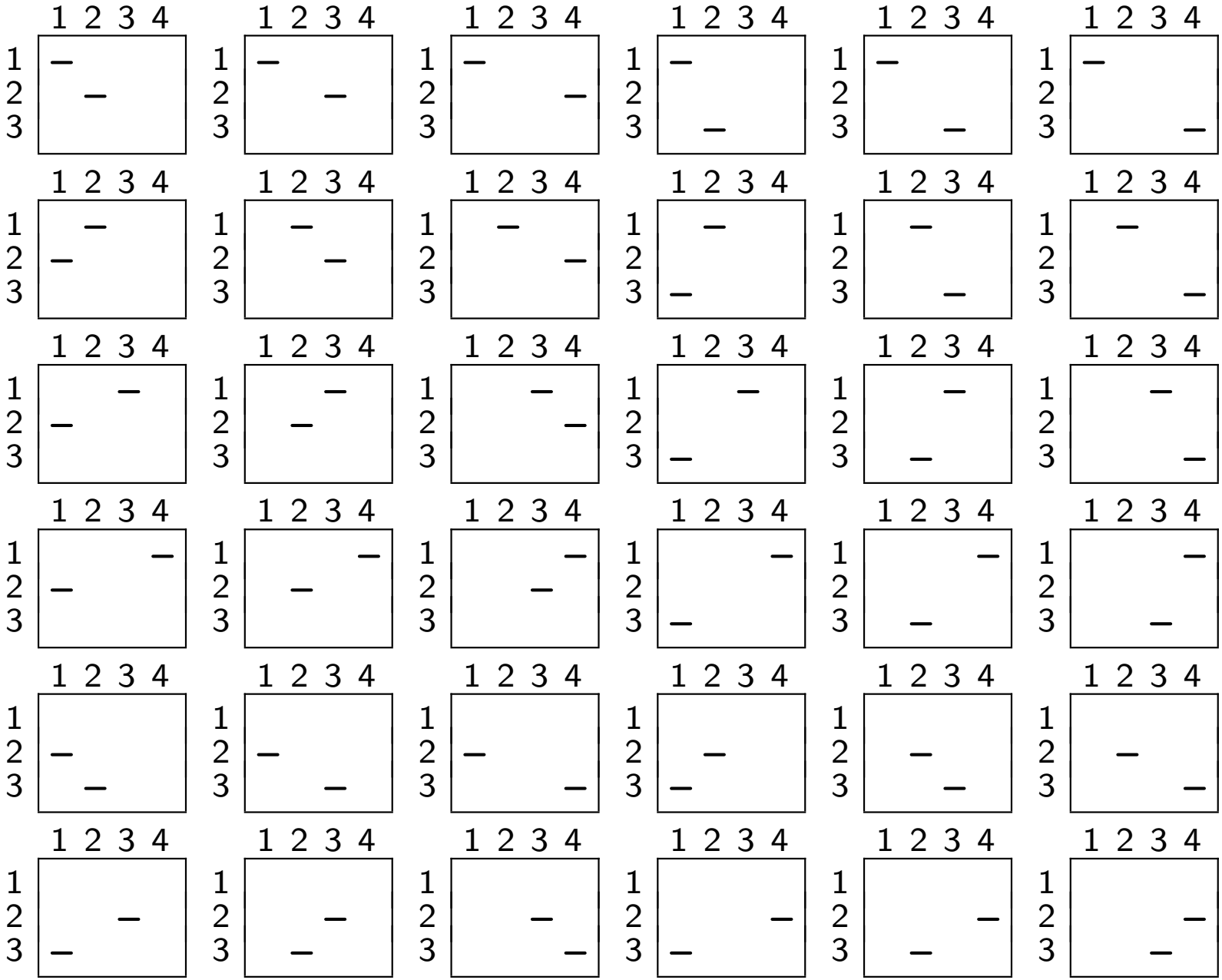
Can $n+1$ pigeons be placed in n holes (at most one pigeon per hole)?

$$PHP_n := \bigwedge_{1 \le p \le n+1} (x_{1,p} \vee \cdots \vee x_{n,p}) \wedge \bigwedge_{1 \le h \le n} \bigwedge_{1 \le p < q \le n+1} (\overline{x}_{h,p} \vee \overline{x}_{h,q})$$

Or in array notation for PHP_3 (inspired by Haken):

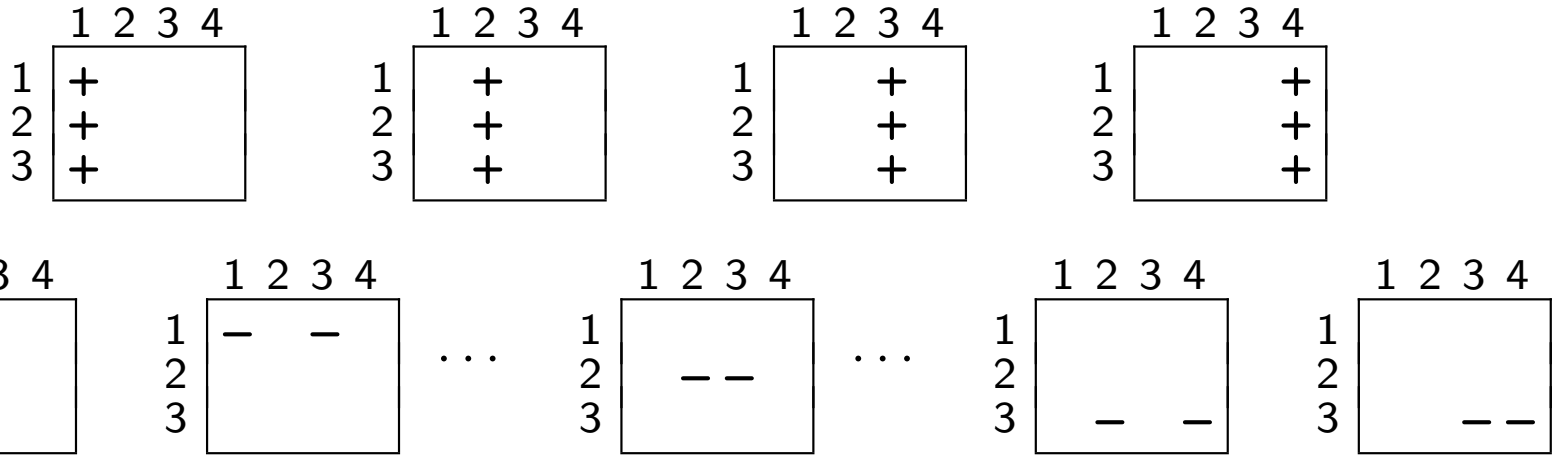


All Binary PR Clauses for PHP_3

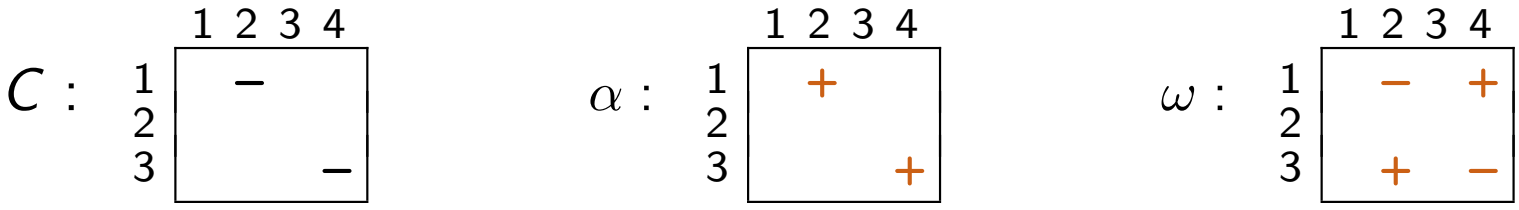


PR Clauses for Pigeon Hole Formulas

Array notation for PHP_3 (inspired by Haken):



Key observation: each $(\bar{x}_{h,p} \vee \bar{x}_{k,q})$ with $h \neq k$, $p \neq q$ is a PR clause.



One can learn a unit clause after learning n such binary clauses.

One can reduce PHP_n to PHP_{n-1} by learning n such unit clauses.

Efficient PR Proof Checker

We implemented an efficient PR proof checker on top of the `DRAT-trim` checker (used to validate SAT competition results).

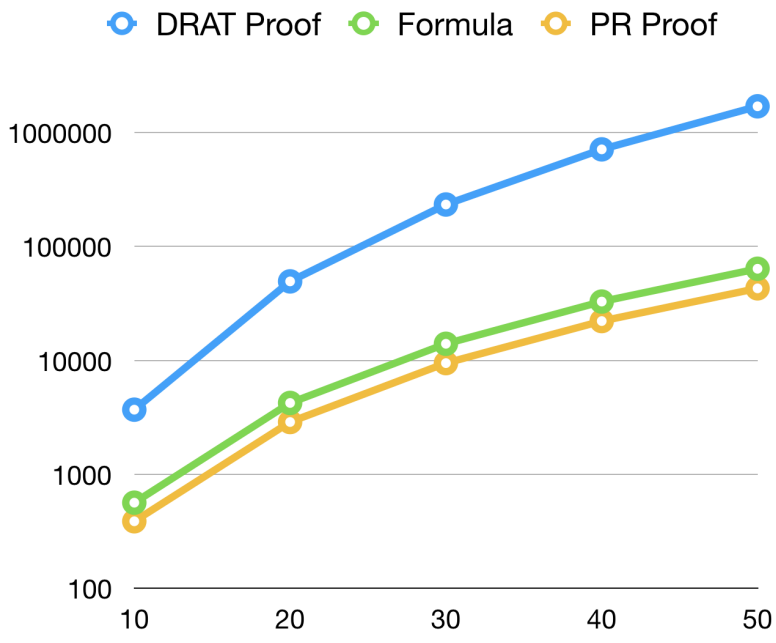
- Complexity is $\mathcal{O}(m^3)$ with m being the number of proof steps.
- However the worst-case is similar to DRAT proof checking...

➥ ..., and DRAT proof checking is in practice almost linear in the size of the formula and proof, by aggressively deleting clauses to limit the size of F.

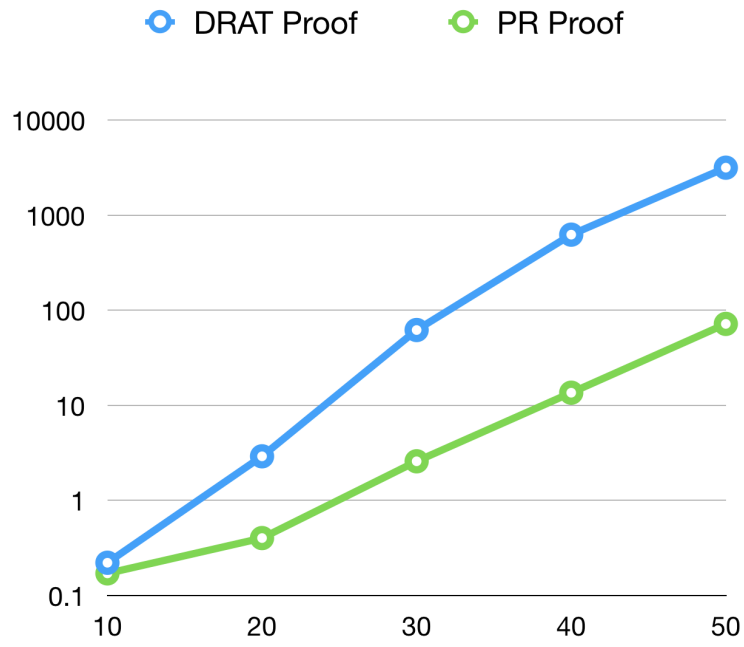
PRcheck (CNF formula F; PR proof $(C_1, \omega_1), \ldots, (C_m, \omega_m)$)

for $i \in \{i, \ldots, m\}$ **do**
  for $D \in F$ **do**
    if $D|\omega_i \neq \top$ **and** $(D|\alpha_i = \top$ **or** $D|\omega_i \subset D|\alpha_i)$ **then**
      if $F|\alpha_i \not\vdash_1 D|\omega_i$ **then return** *failure*
  $F := F \cup \{C_i\}$
return *success*

Comparison of Proof Size and Validation Times



size in the number of clauses



validation time in seconds

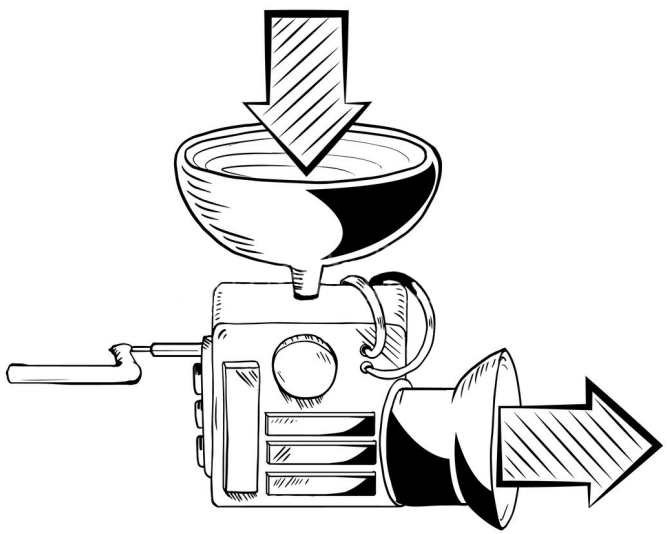
Satisfaction-Driven Clause Learning

Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



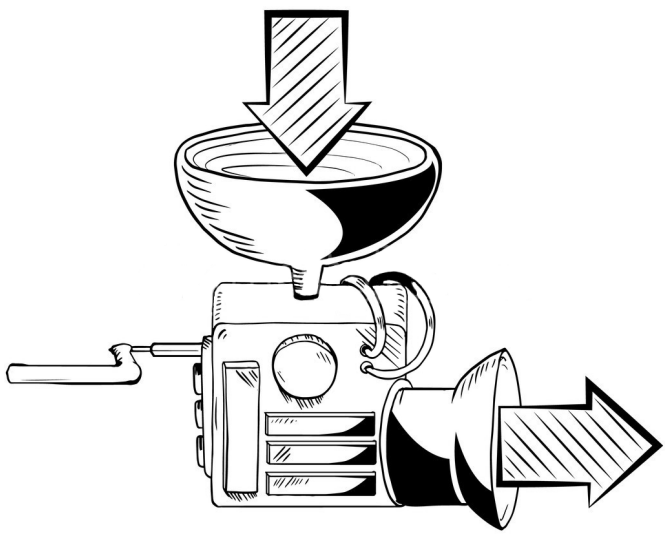
Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

Input Formula in CNF

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$

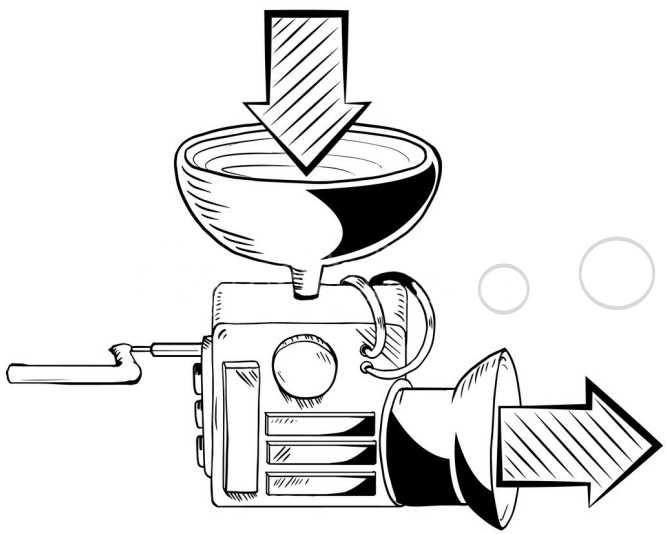


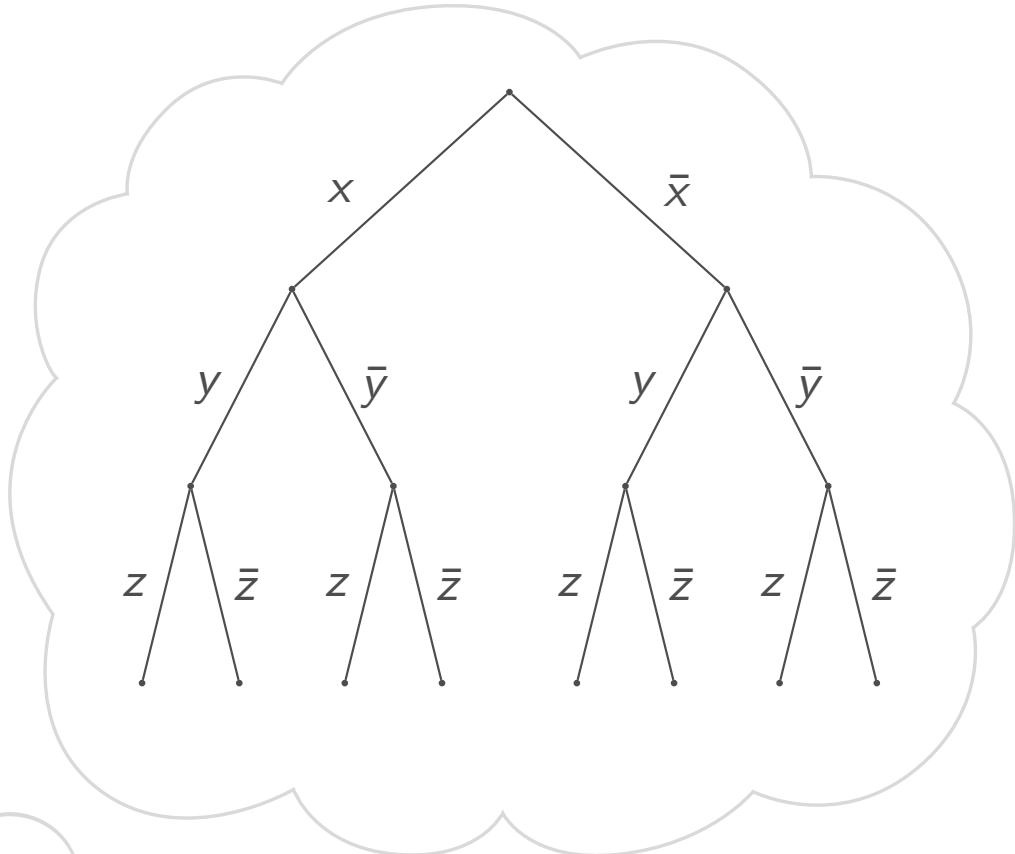
Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



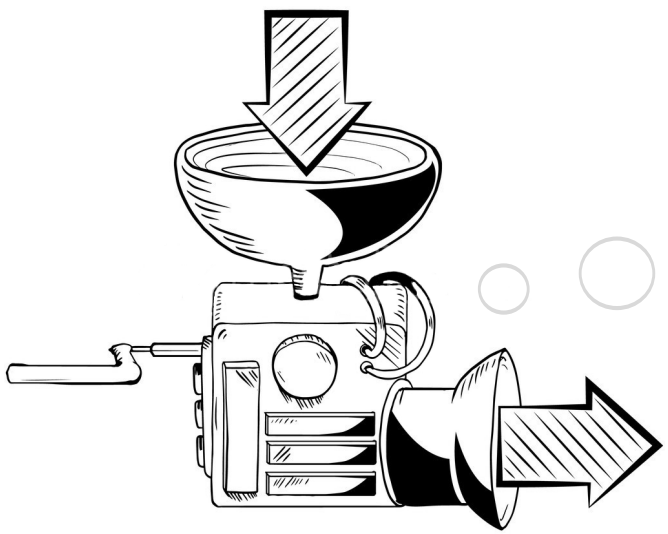


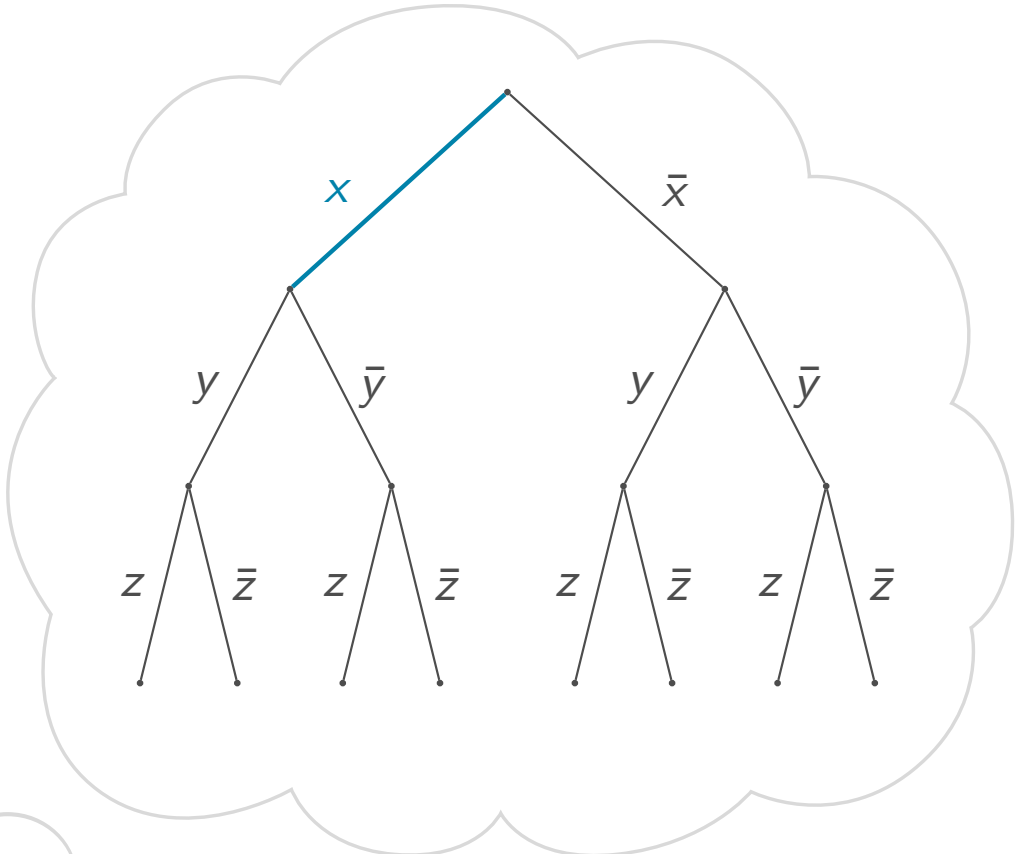
Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



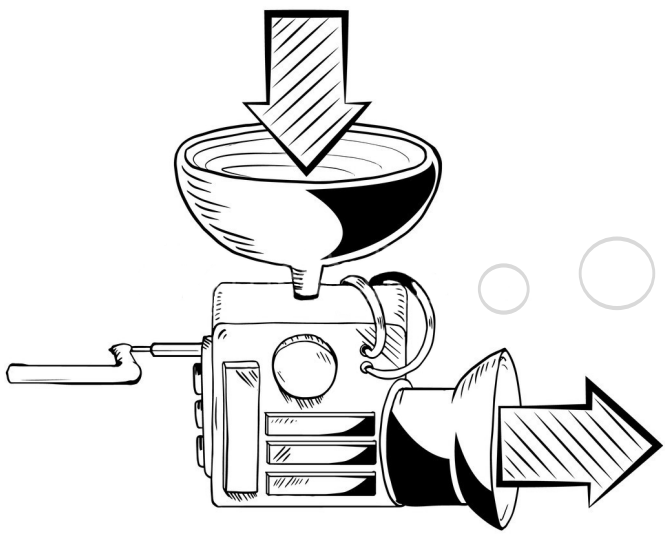


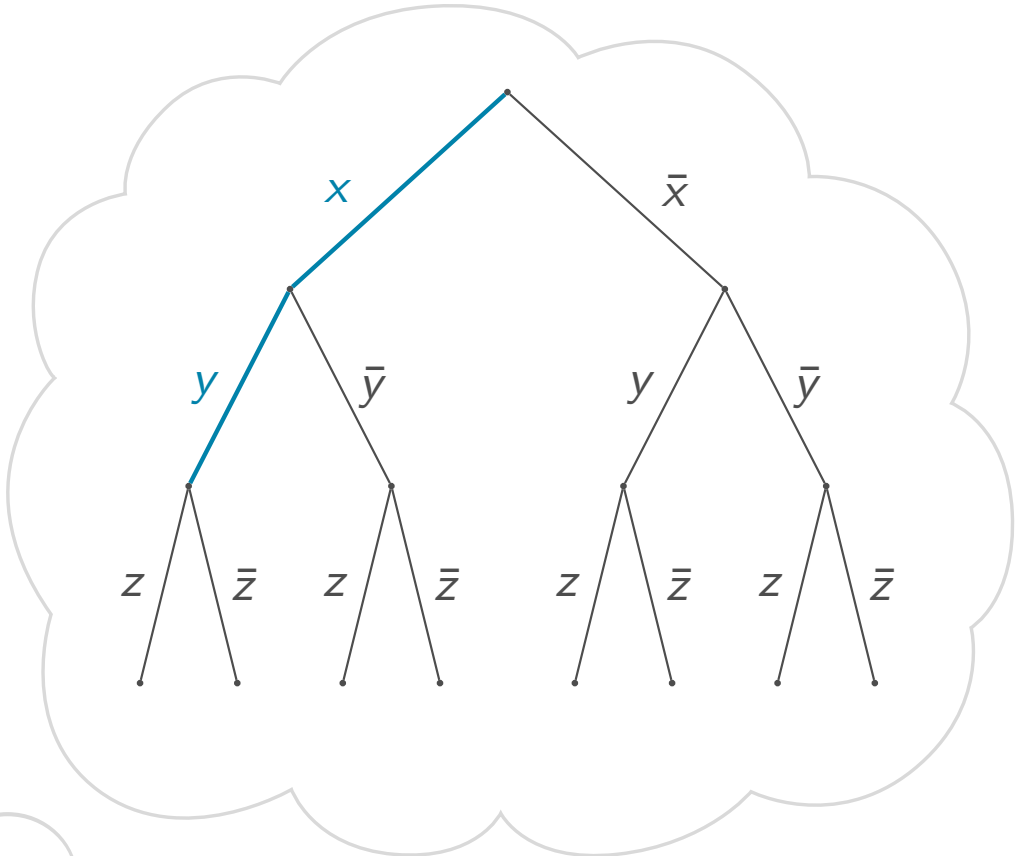
Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



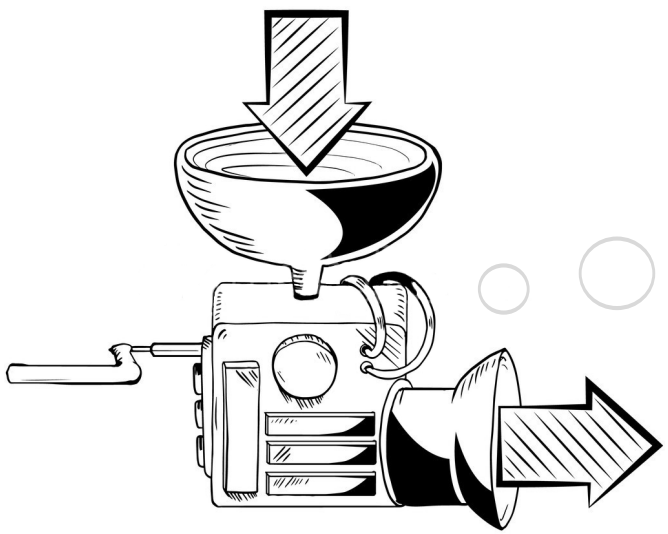


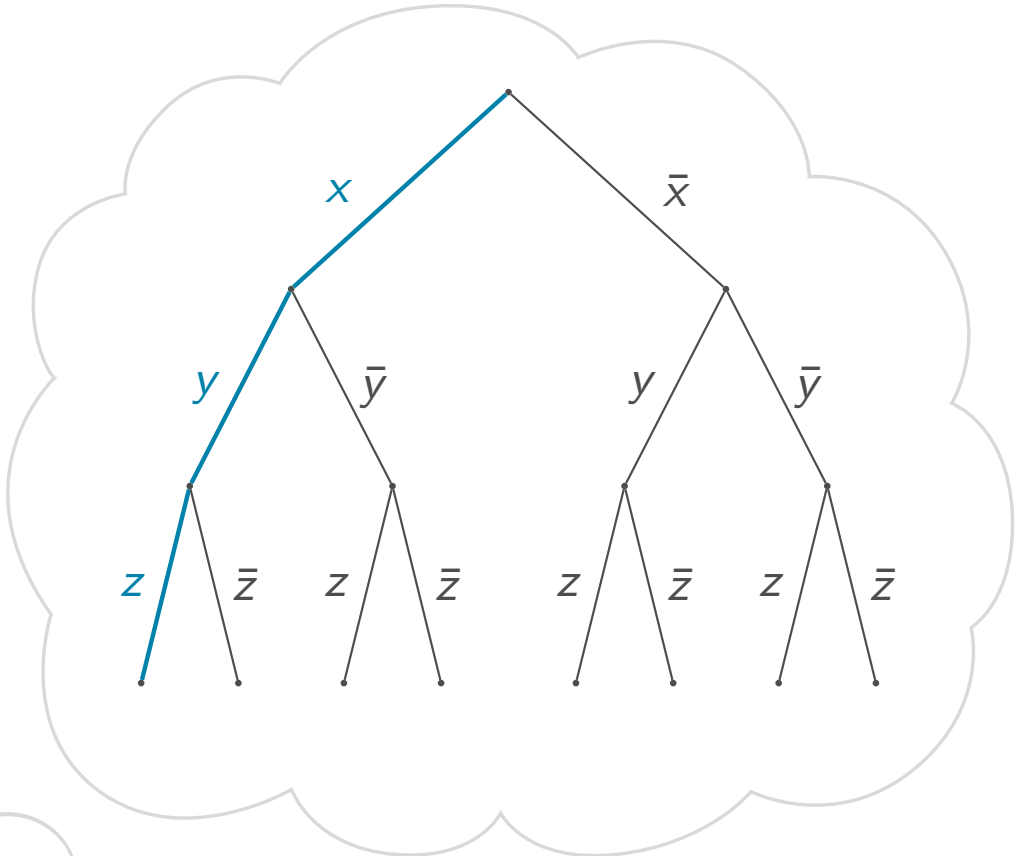
Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



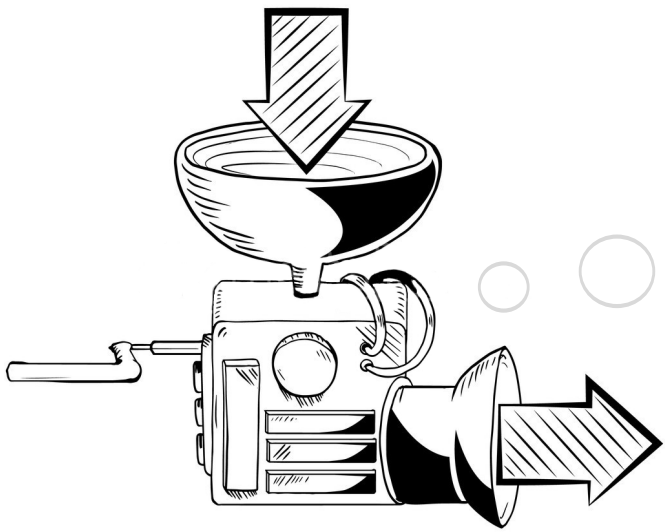


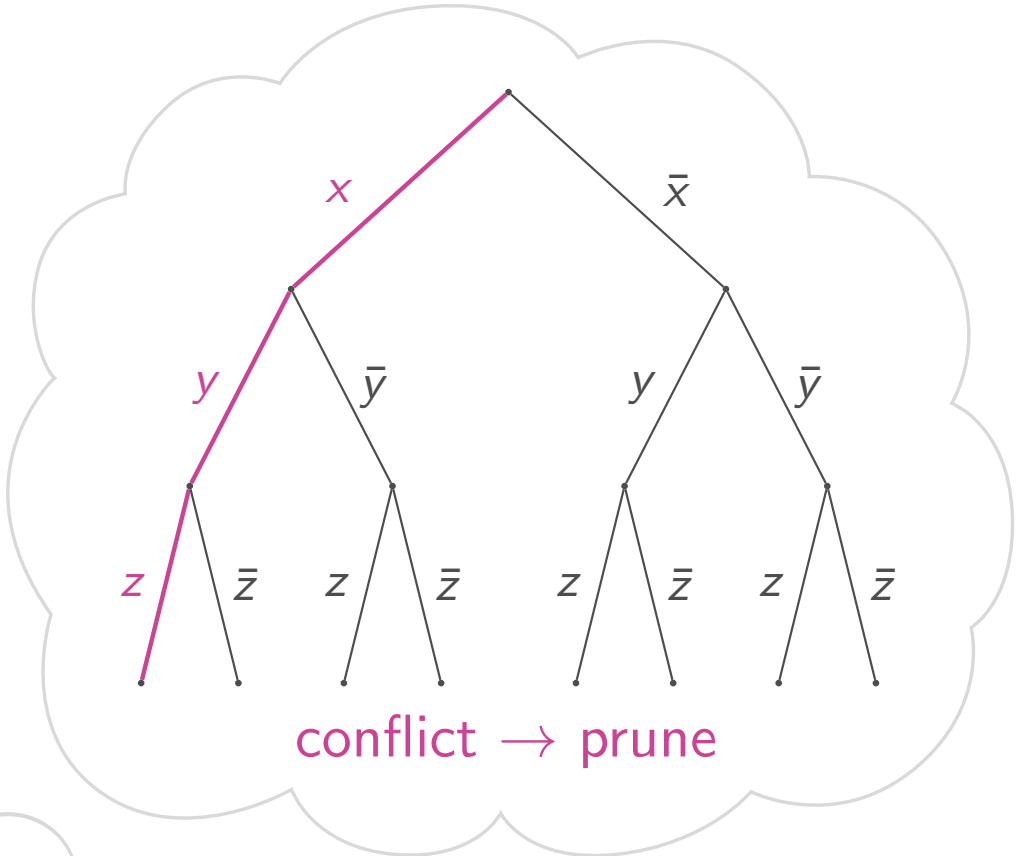
Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



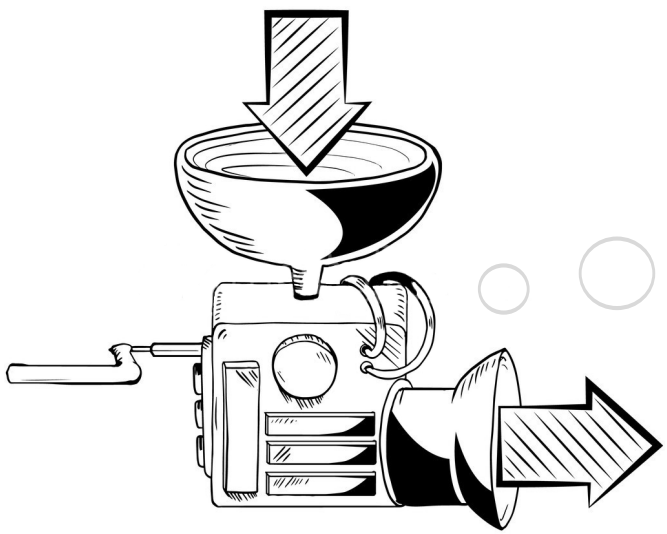


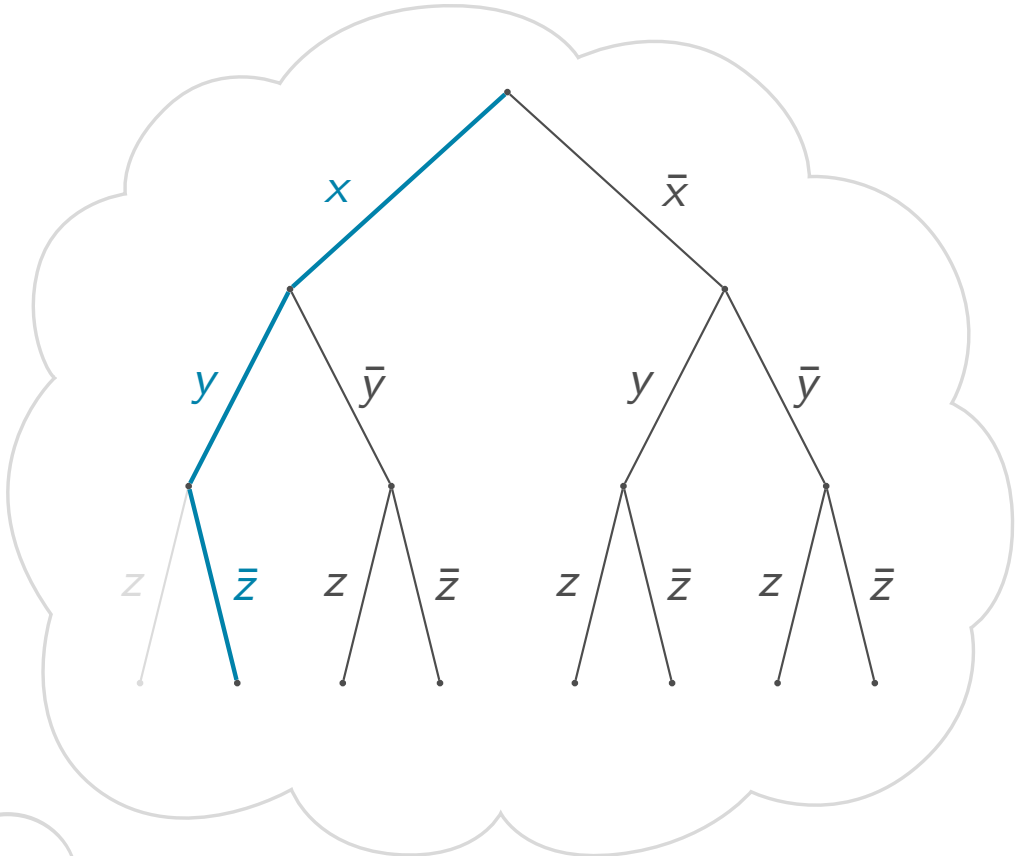
Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$

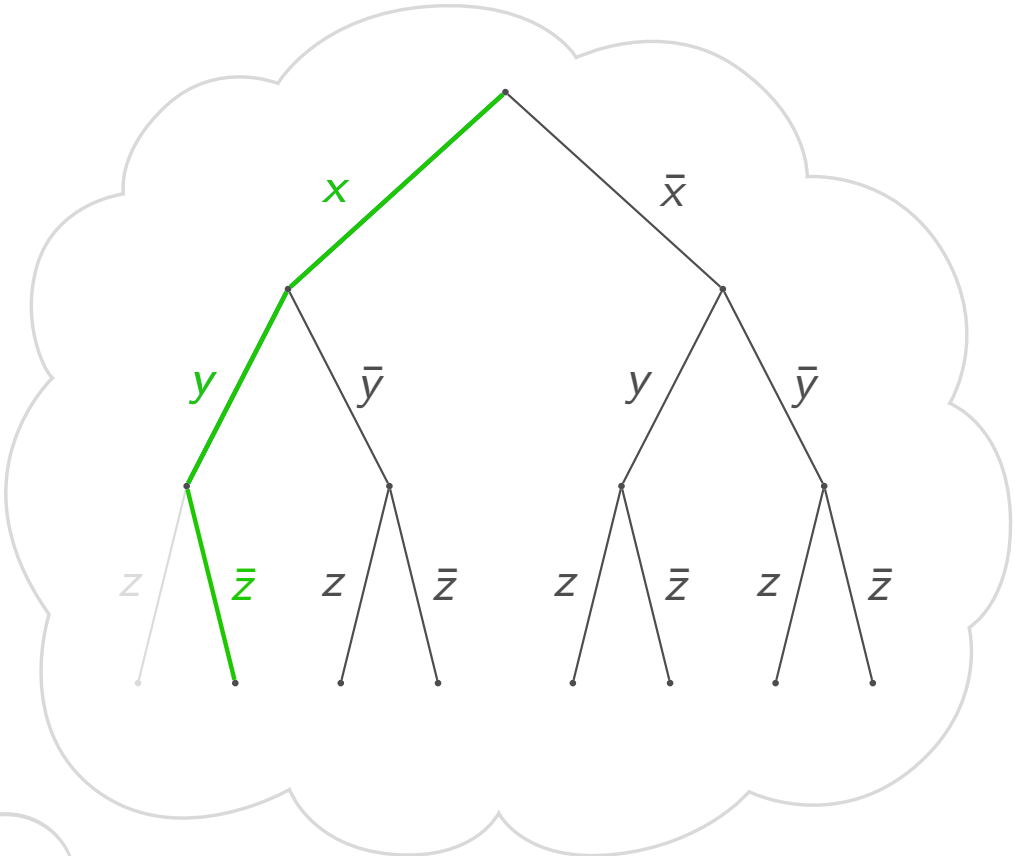




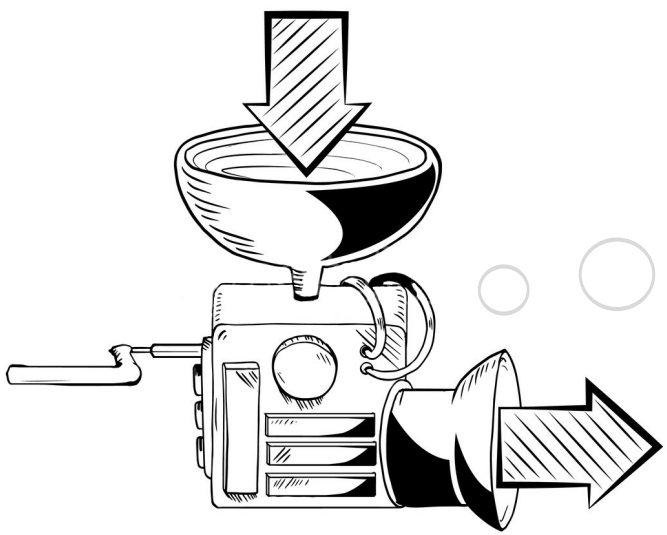
Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?



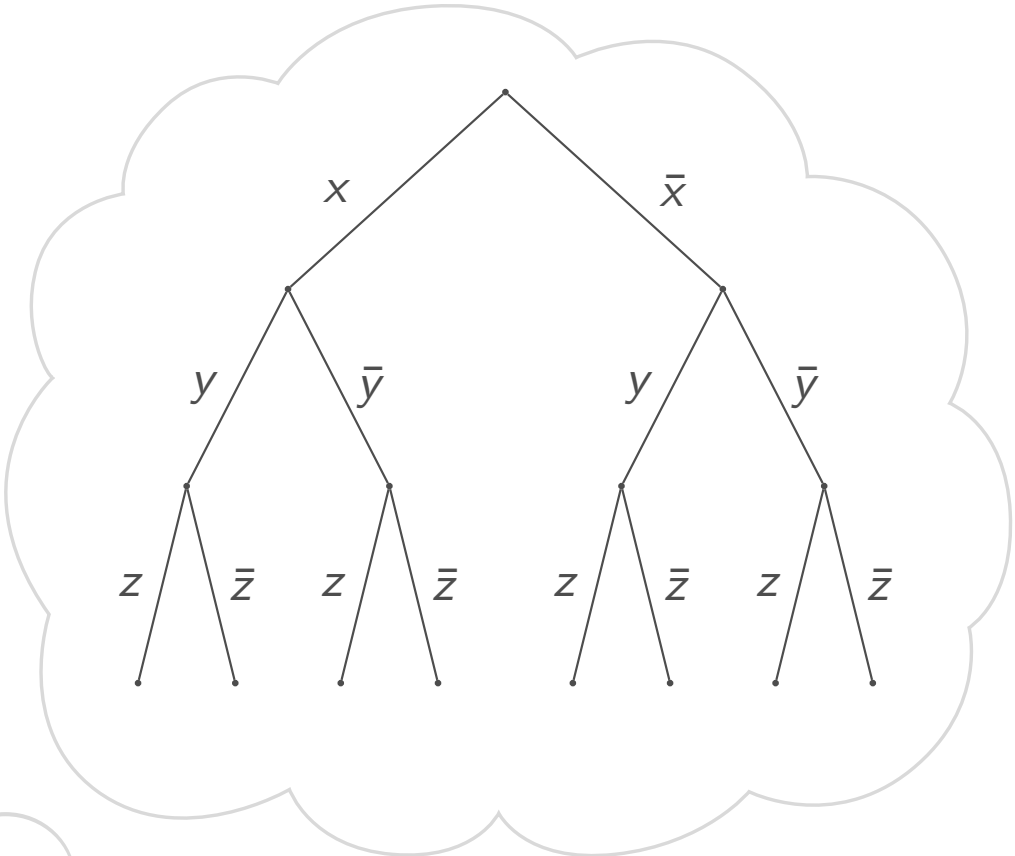
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



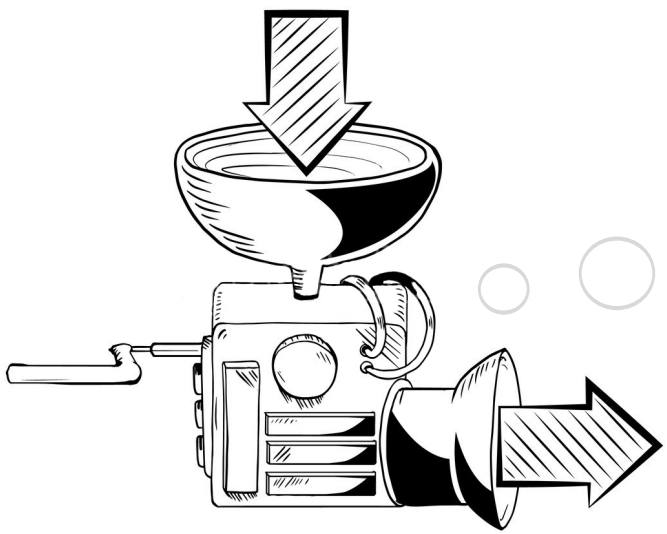
Satisfiable

Key Idea: Prune Less Satisfiable Branches

Can we prune earlier?
Even satisfiable branches?

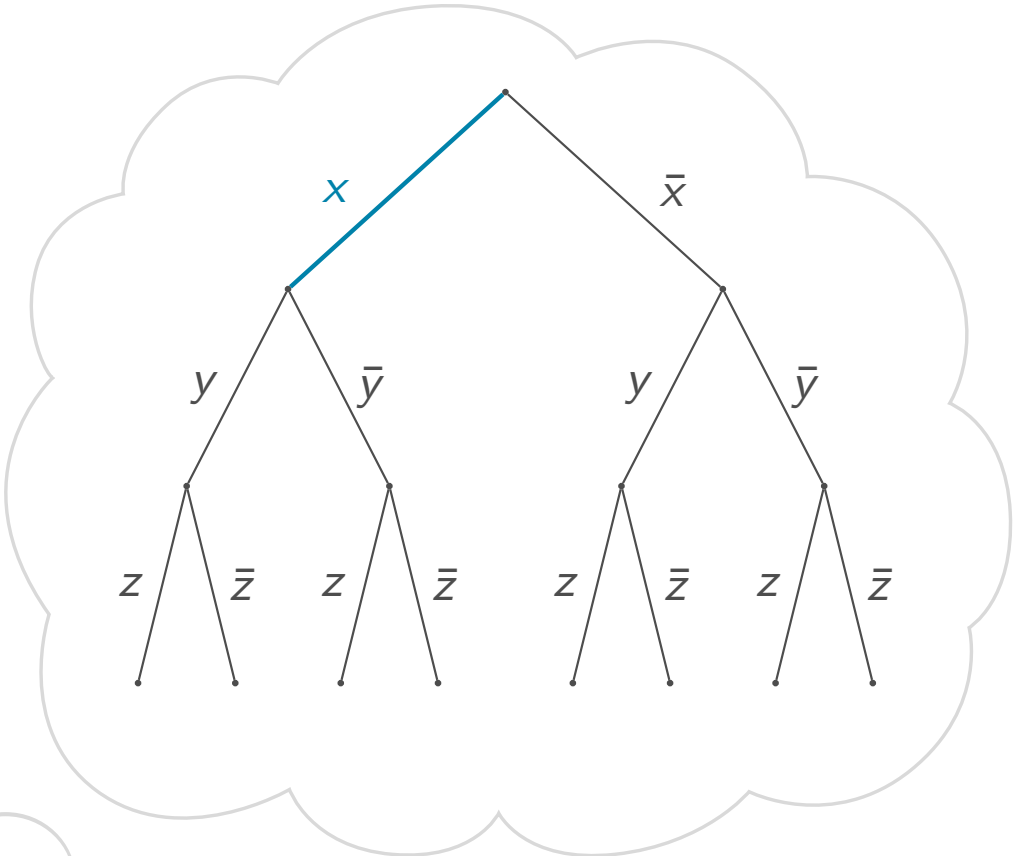


$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$

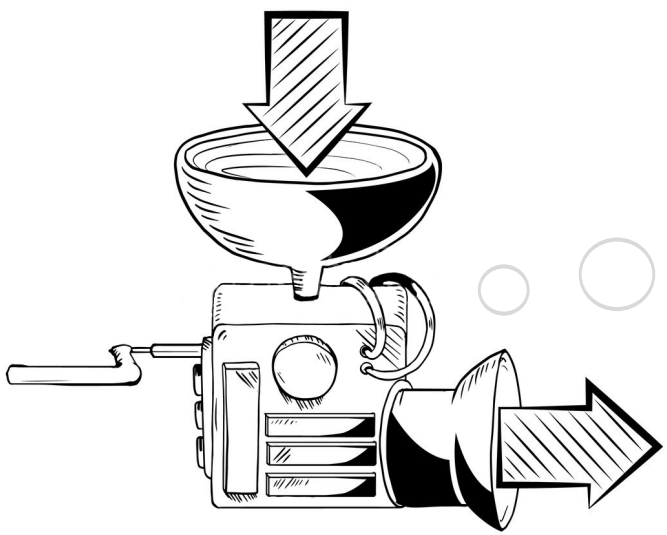


Key Idea: Prune Less Satisfiable Branches

Can we prune earlier?
Even satisfiable branches?

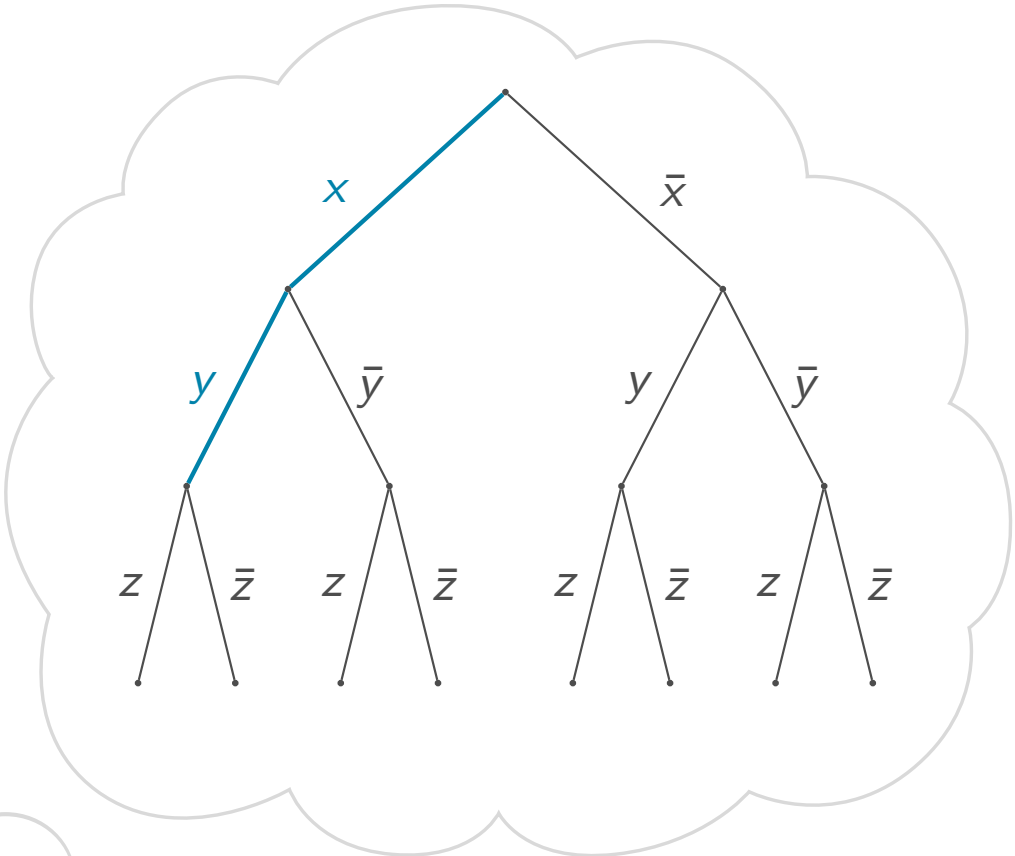


$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$

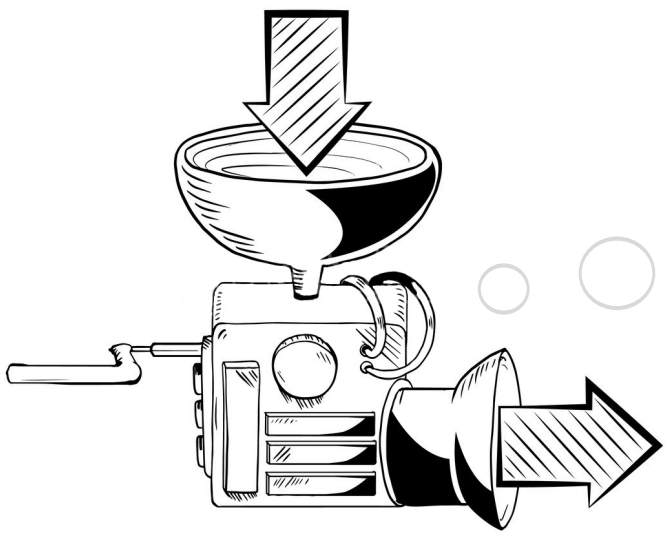


Key Idea: Prune Less Satisfiable Branches

Can we prune earlier?
Even satisfiable branches?



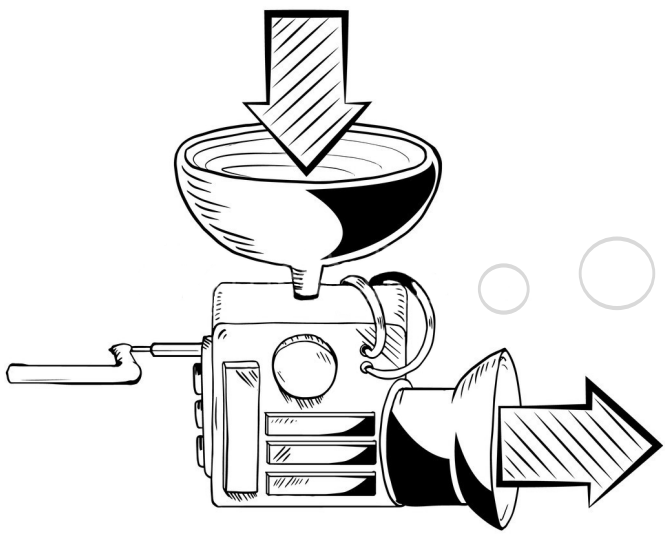
$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$

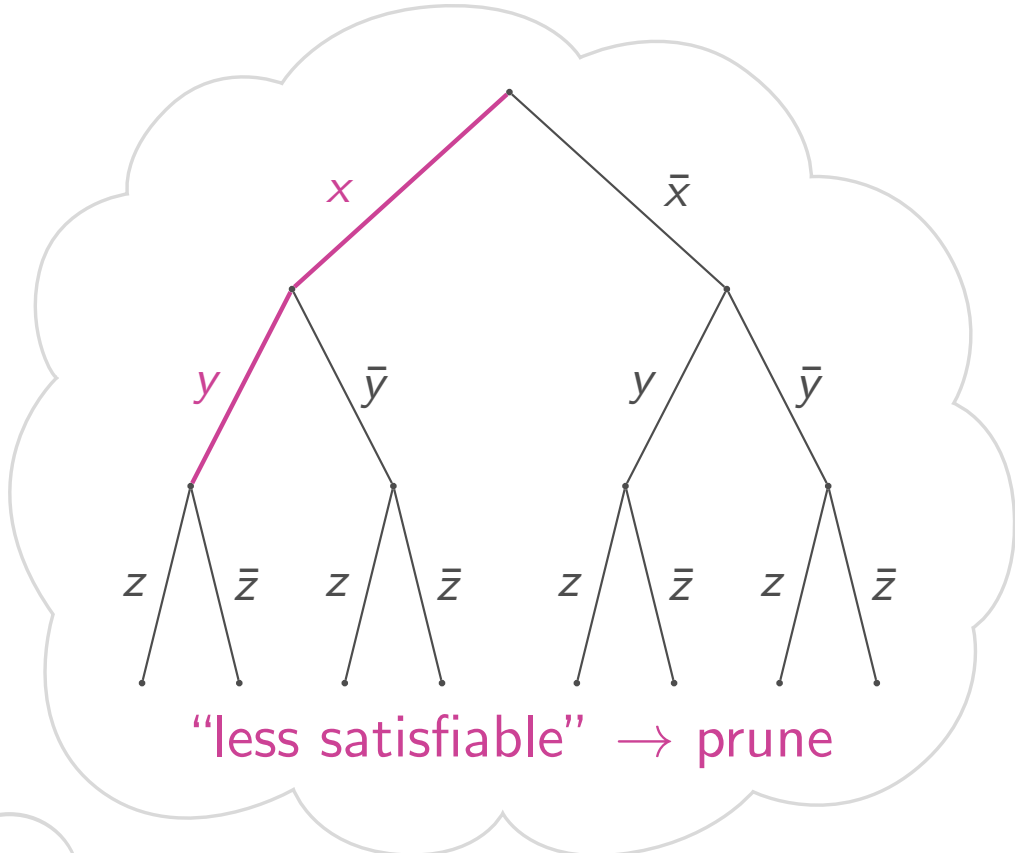


Key Idea: Prune Less Satisfiable Branches

Can we prune earlier?
Even satisfiable branches?

$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$

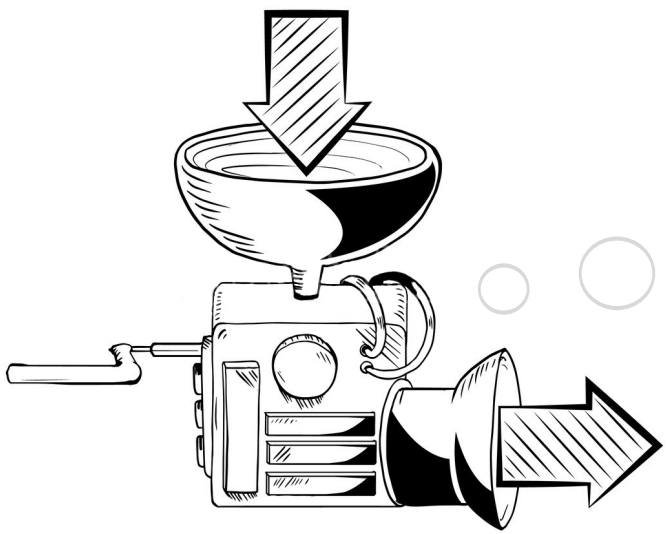


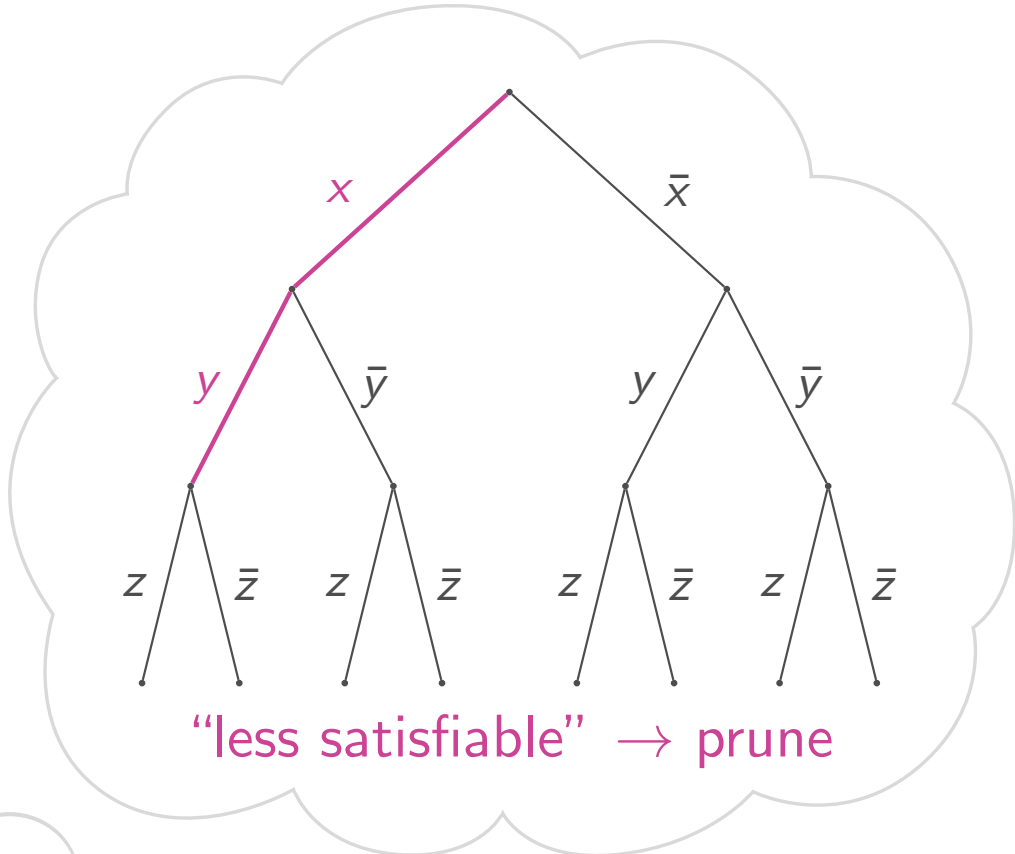


Key Idea: Prune Less Satisfiable Branches

Can we prune earlier?
Even satisfiable branches?

$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$

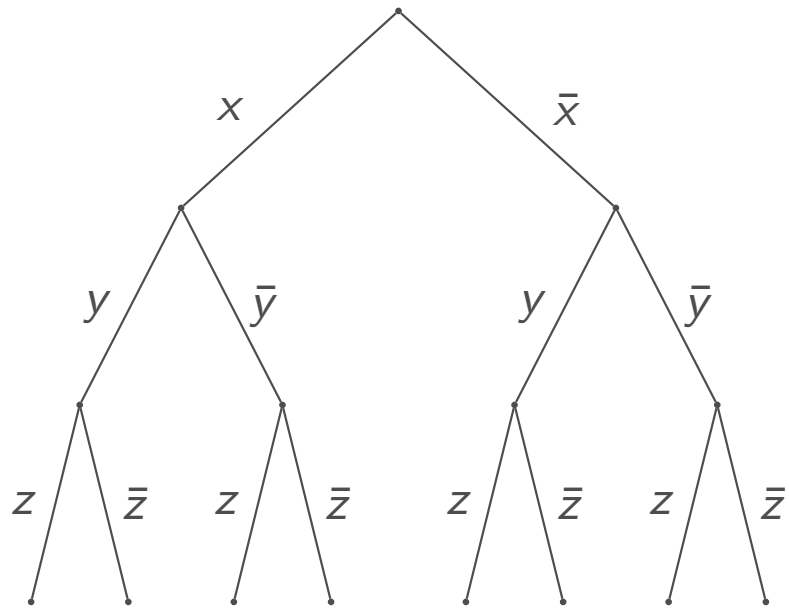




How to prune? Add redundant clauses!

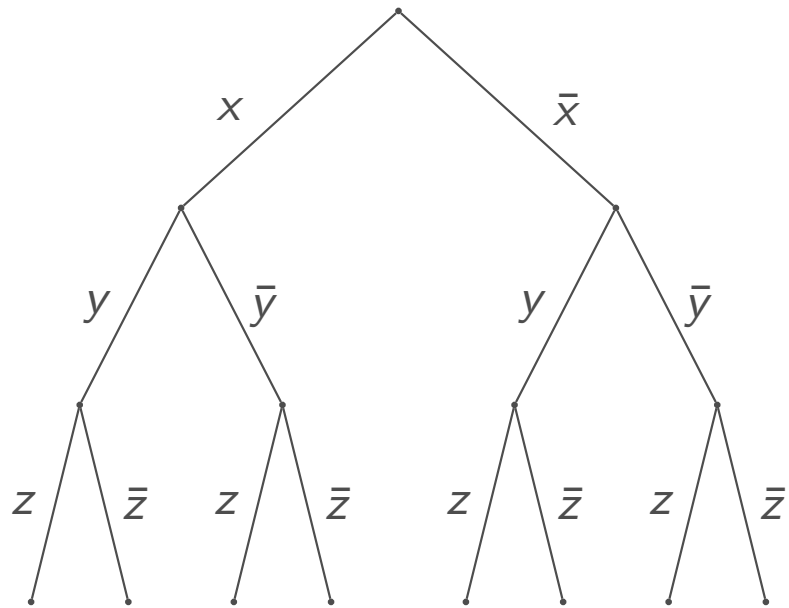
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.



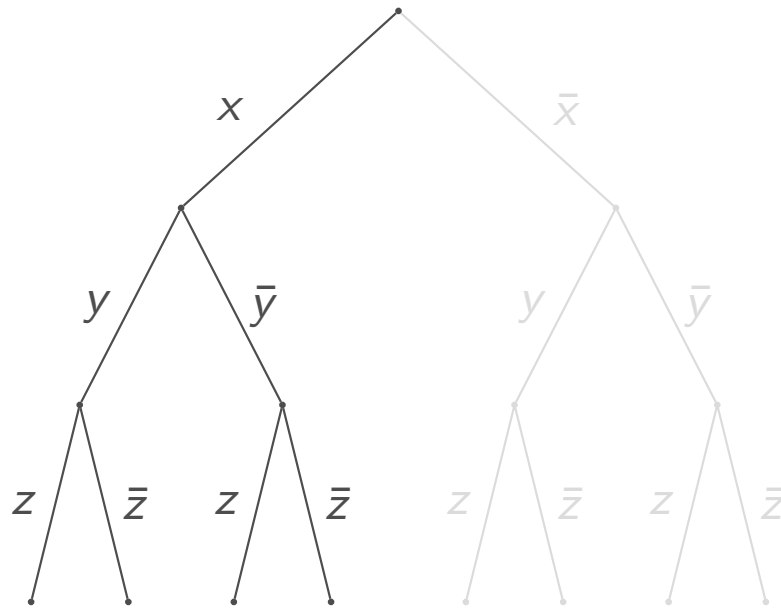
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- Example: The clause (x) prunes all branches where x is false.



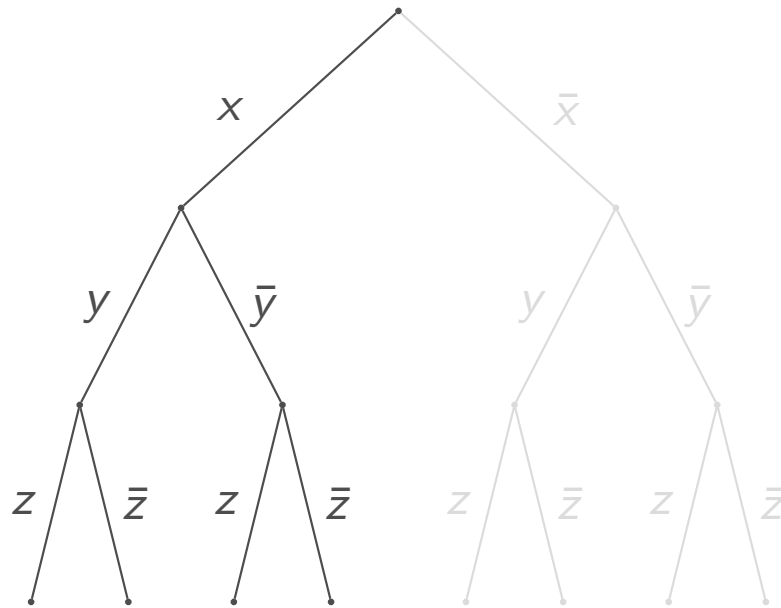
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- Example: The clause (x) prunes all branches where x is false.



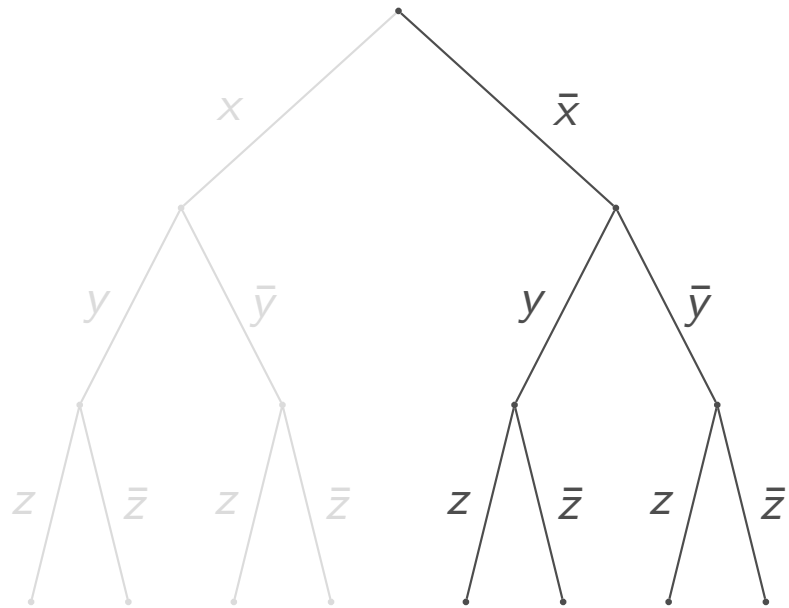
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- Example: The clause (x) prunes all branches where x is false.
- Other Examples:



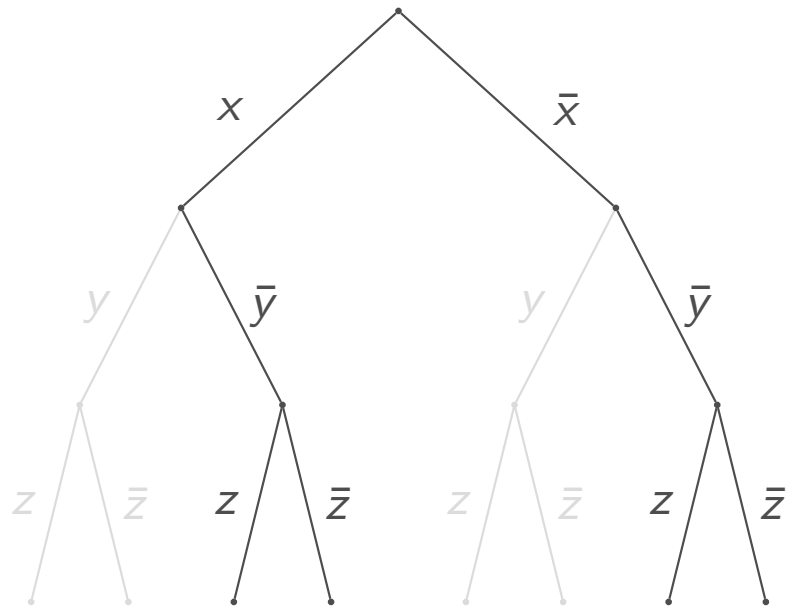
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- Example: The clause (x) prunes all branches where x is false.
- Other Examples: $(\bar{x})$



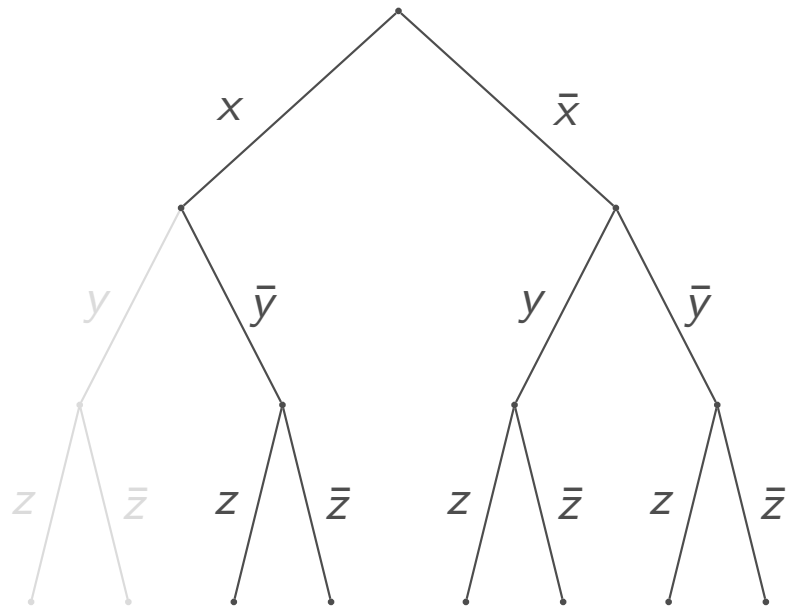
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- Example: The clause (x) prunes all branches where x is false.
- Other Examples: $(\bar{x})$ $(\bar{y})$



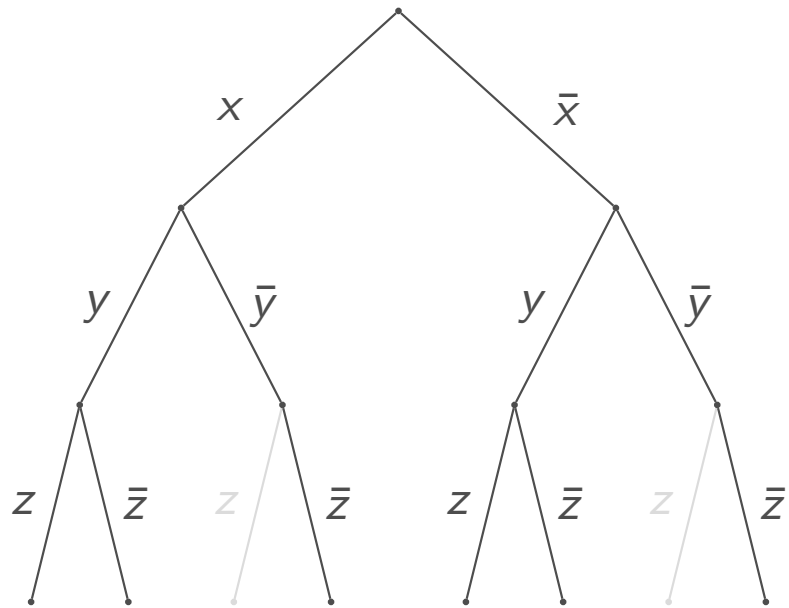
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- Example: The clause (x) prunes all branches where x is false.
- Other Examples: $(\bar{x})$ $(\bar{y})$ $(\bar{x} \vee \bar{y})$



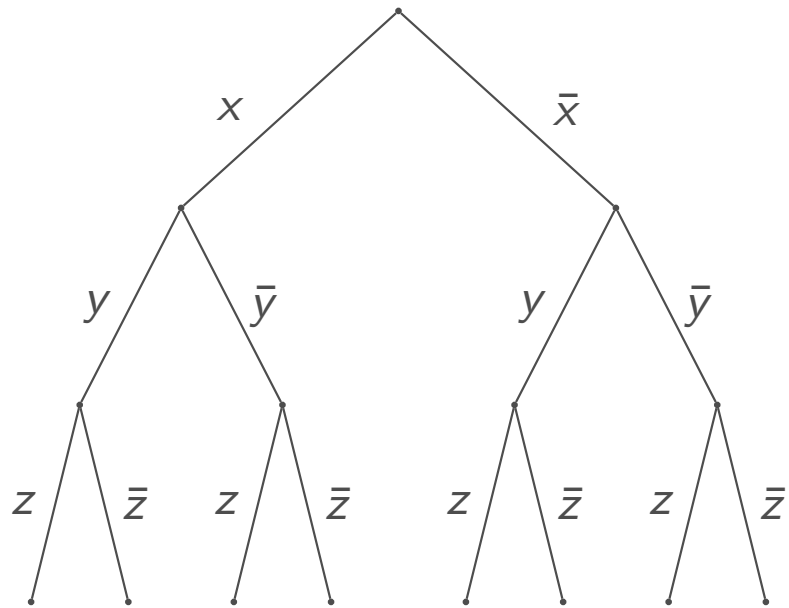
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- Example: The clause (x) prunes all branches where x is false.
- Other Examples: $(\bar{x})$ $(\bar{y})$ $(\bar{x} \vee \bar{y})$ $(y \vee \bar{z})$



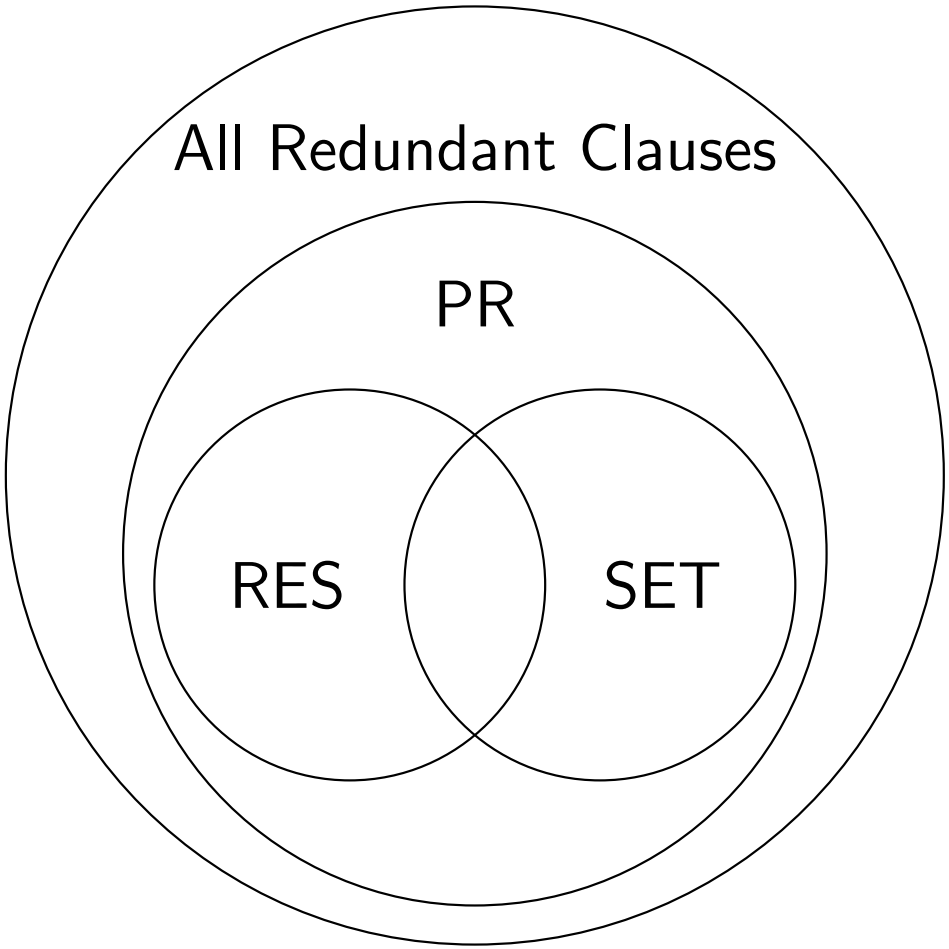
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- Example: The clause (x) prunes all branches where x is false.
- Other Examples: $(\bar{x})$ $(\bar{y})$ $(\bar{x} \vee \bar{y})$ $(y \vee \bar{z})$ $(x \vee \bar{x})$



Redundant Clauses

A clause C is redundant w.r.t. a formula F if and only if F and $F \wedge C$ are either both satisfiable or both unsatisfiable.



PR = Propagation Redundant Clauses [CADE'17]

RES = Resolvents

SET = Set-Blocked Clauses [IJCAR'16]

Finding Redundant Clauses: The Positive Reduct

Determining whether a clause C is SET or PR w.r.t. a formula F is an NP-complete problem.

How to find SET and PR clauses? Encode it in SAT!

Finding Redundant Clauses: The Positive Reduct

Determining whether a clause C is SET or PR w.r.t. a formula F is an NP-complete problem.

How to find SET and PR clauses? Encode it in SAT!

Given a formula F and a clause C. Let α denote the smallest assignment that falsifies C. The positive reduct of F and α is a formula which is satisfiable if and only if C is SET w.r.t. F.

Finding Redundant Clauses: The Positive Reduct

Determining whether a clause C is SET or PR w.r.t. a formula F is an NP-complete problem.

How to find SET and PR clauses? Encode it in SAT!

Given a formula F and a clause C. Let α denote the smallest assignment that falsifies C. The positive reduct of F and α is a formula which is satisfiable if and only if C is SET w.r.t. F.

Positive reducts are typically very easy to solve!

Finding Redundant Clauses: The Positive Reduct

Determining whether a clause C is SET or PR w.r.t. a formula F is an NP-complete problem.

How to find SET and PR clauses? Encode it in SAT!

Given a formula F and a clause C. Let α denote the smallest assignment that falsifies C. The positive reduct of F and α is a formula which is satisfiable if and only if C is SET w.r.t. F.

Positive reducts are typically very easy to solve!

Key Idea: While solving a formula F, check whether the positive reduct of F and the current assignment α is satisfiable. In that case, prune the branch α.

The Positive Reduct: An Example

Given a formula F and a clause C. Let α denote the smallest assignment that falsifies C. The positive reduct of F and α, denoted by $p(F, \alpha)$, is the formula that contains C and all *assigned*(D, α) with $D \in F$ and D is satisfied by α.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Let $C_1 = (\bar{x})$, so $\alpha_1 = x$.
The positive reduct $p(F, \alpha_1) = (\bar{x}) \wedge (x) \wedge (x)$ is unsatisfiable.

Let $C_2 = (\bar{x} \vee \bar{y})$, so $\alpha_2 = x\,y$.
The positive reduct $p(F, \alpha_2) = (\bar{x} \vee \bar{y}) \wedge (x \vee y) \wedge (x \vee \bar{y})$ is satisfiable.

Autarkies

A non-empty assignment α is an autarky for formula F if every clause $C \in F$ that is touched by α is also satisfied by α.

A pure literal and a satisfying assignment are autarkies.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.
Assignment $\alpha_1 = \bar{z}$ is an autarky: $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.
Assignment $\alpha_2 = x\,\bar{y}\,z$ is an autarky: $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Autarkies

A non-empty assignment α is an autarky for formula F if every clause $C \in F$ that is touched by α is also satisfied by α.

A pure literal and a satisfying assignment are autarkies.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.
Assignment $\alpha_1 = \bar{z}$ is an autarky: $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.
Assignment $\alpha_2 = x\,\bar{y}\,z$ is an autarky: $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Given an assignment α, $F|_\alpha$ denotes a formula F without the clauses satisfied by α and without the literals falsified by α.

Theorem ([Monien and Speckenmeyer 1985])

Let α be an autarky for formula F.
Then, F and $F|_\alpha$ are satisfiability equivalent.

Conditional Autarkies

An assignment $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}}$ is a conditional autarky for formula F if α_{aut} is an autarky for $F|_{\alpha_{\text{con}}}$.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.
Let $\alpha_{\text{con}} = x$ and $\alpha_{\text{aut}} = \bar{y}$, then $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}} = x\,\bar{y}$ is a conditional autarky for F:

$$\alpha_{\text{aut}} = \bar{y} \text{ is an autarky for } F|_{\alpha_{\text{con}}} = (\bar{y} \vee \bar{z}).$$

Conditional Autarkies

An assignment $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}}$ is a conditional autarky for formula F if α_{aut} is an autarky for $F|_{\alpha_{\text{con}}}$.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.
Let $\alpha_{\text{con}} = x$ and $\alpha_{\text{aut}} = \bar{y}$, then $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}} = x\,\bar{y}$ is a conditional autarky for F:

$$\alpha_{\text{aut}} = \bar{y} \text{ is an autarky for } F|_{\alpha_{\text{con}}} = (\bar{y} \vee \bar{z}).$$

Let $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}}$ be a conditional autarky for formula F.
Then F and $F \wedge (\alpha_{\text{con}} \rightarrow \alpha_{\text{aut}})$ are satisfiability-equivalent.

In the above example, we could therefore learn $(\bar{x} \vee \bar{y})$.

Learning PR clauses

Theorem
Given a formula F and an assignment α. Every satisfying assignment ω of $p(F, \alpha)$ is a conditional autarky of F.

Recall: Given a formula F and a clause C. Let α denote the smallest assignment that falsifies C. C is SET w.r.t. F if and only if $p(F, \alpha)$ is satisfiable.

Let assignment ω satisfy $p(F, \alpha)$. Removing all but one of the literals in C that are satisfied by ω results in a PR clause w.r.t. F.

Pseudo-Code of CDCL (formula F)

```
1    α := ∅
2    forever do
3       α := Simplify (F, α)
4       if F|α contains a falsified clause then
5          C := AnalyzeConflict ()
6          if C is the empty clause then return unsatisfiable
7          F := F ∪ {C}
8          α := BackJump (C, α)
13      else
14         l := Decide ()
15         if l is undefined then return satisfiable
16         α := α ∪ {l}
```

Pseudo-Code of SDCL (formula F)

```
1   α := ∅
2   forever do
3     α := Simplify (F, α)
4     if F|α contains a falsified clause then
5       C := AnalyzeConflict ()
6       if C is the empty clause then return unsatisfiable
7       F := F ∪ {C}
8       α := BackJump (C, α)
9     else if p(F, α) is satisfiable then
10      C := AnalyzeWitness ()
11      F := F ∪ {C}
12      α := BackJump (C, α)
13    else
14      l := Decide ()
15      if l is undefined then return satisfiable
16      α := α ∪ {l}
```

Benchmark Suite: Pigeon Hole Formulas

Can $n+1$ pigeons be placed in n holes (at-most-one pigeon per hole)?

$$PHP_n := \bigwedge_{1 \leq p \leq n+1} (x_{1,p} \vee \cdots \vee x_{n,p}) \wedge \bigwedge_{1 \leq h \leq n,\, 1 \leq p < q \leq n+1} (\overline{x}_{h,p} \vee \overline{x}_{h,q})$$

The binary clauses encode the constraint $\leq_1 (x_{h,1}; \ldots; x_{h,n+1})$.

There exists more compact encodings, such as the sequential counter and minimal encoding, for at-most-one constraints.

We include these encodings to evaluate the robustness of the solver.

Tool Comparison

We used three tools in our evaluation:

- EBDDRES: A tool based on binary decision diagrams that can convert a refutation into an extended resolution proof.
- GLUCOSER: A SAT solver with extended learning, i.e., a technique that introduces new variables and could potentially solve pigeon hole formulas in polynomial time.
- LINGELING (PR): Our SDCL solver.

Results on Small Pigeon Hole Formulas

formula	input #var	input #cls	EBDDRES time	EBDDRES #node	GLUCOSER time	GLUCOSER #lemma	LINGELING (PR) time	LINGELING (PR) #lemma
PHP_{10}-std	110	561	1.00	3M	22.71	329,470	0.07	329
PHP_{11}-std	132	738	3.47	9M	146.61	1,514,845	0.11	439
PHP_{12}-std	156	949	10.64	27M	307.29	2,660,358	0.16	571
PHP_{13}-std	182	1,197	30.81	76M	982.84	6,969,736	0.22	727
PHP_{10}-seq	220	311	OF	——	1.62	25,712	0.07	327
PHP_{11}-seq	264	375	OF	——	6.94	77,747	0.10	437
PHP_{12}-seq	312	445	OF	——	19.40	174,084	0.14	569
PHP_{13}-seq	364	521	OF	——	172.76	1,061,318	0.18	725
PHP_{10}-min	180	281	28.60	81M	0.64	15,777	0.06	329
PHP_{11}-min	220	342	143.92	399M	1.82	34,561	0.10	439
PHP_{12}-min	264	409	OF	——	9.87	121,321	0.13	571
PHP_{13}-min	312	482	OF	——	57.66	483,789	0.18	727

OF = 32-bit overflow

Results on Large Pigeon Hole Formulas

formula	input #var	input #cls	EBDDRES time	EBDDRES #node	GLUCOSER time	GLUCOSER #lemma	LINGELING (PR) time	LINGELING (PR) #lemma
PHP_{20}-std	420	4,221	OF	——	TO	——	1.61	2,659
PHP_{30}-std	930	13,981	OF	——	TO	——	13.45	8,989
PHP_{40}-std	1,640	32,841	OF	——	TO	——	67.41	21,319
PHP_{50}-std	2,550	63,801	OF	——	TO	——	241.14	41,649
PHP_{20}-seq	840	1,221	OF	——	TO	——	1.05	2,657
PHP_{30}-seq	1,860	2,731	OF	——	TO	——	6.55	8,987
PHP_{40}-seq	3,280	4,841	OF	——	TO	——	27.10	21,317
PHP_{50}-seq	5,100	7,551	OF	——	TO	——	86.30	41,647
PHP_{20}-min	760	1,161	OF	——	TO	——	1.03	2,659
PHP_{30}-min	1,740	2,641	OF	——	TO	——	6.30	8,989
PHP_{40}-min	3,120	4,721	OF	——	TO	——	26.65	21,319
PHP_{50}-min	4,900	7,401	OF	——	TO	——	85.00	41,649

OF = 32-bit overflow TO = timeout of 9000 seconds

Conclusions

Conclusions

- We introduced new redundancy notions for SAT.
- The redundancy notions strictly generalize RAT.
- Proof systems based on these redundancy notions are strong.
  - They allow for short proofs without new variables.

Conclusions

- We introduced new redundancy notions for SAT.
- The redundancy notions strictly generalize RAT.
- Proof systems based on these redundancy notions are strong.
  - They allow for short proofs without new variables.
- SDCL generalizes the well-known CDCL paradigm by allowing to prune branches that are potentially satisfiable:
  - Such branches can be found using the positive reduct;
  - Pruning can be expressed in the PR proof system;
  - Runtime and proofs can be exponentially smaller.