

# PRuning Through Satisfaction

**Marijn J.H. Heule**, Benjamin Kiesl,  
Martina Seidl, and Armin Biere

UT Austin, Vienna University of Technology, and JKU Linz



Haifa Verification Conference

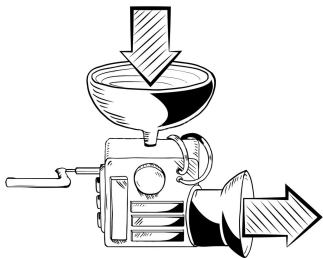
November 15, 2017

# 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)

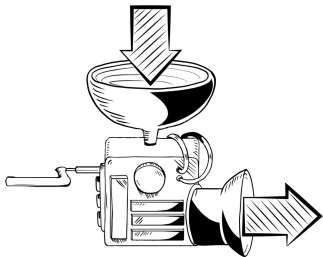
## SAT problem:

Given a propositional formula, is it satisfiable?

Input Formula in CNF



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

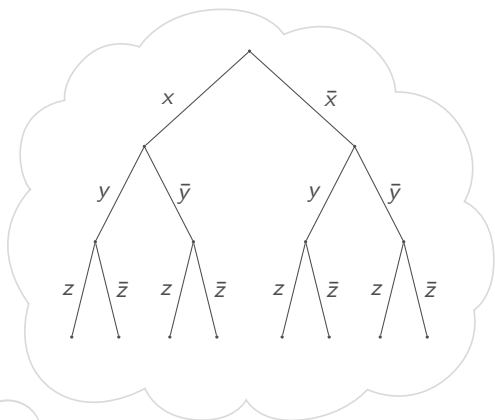
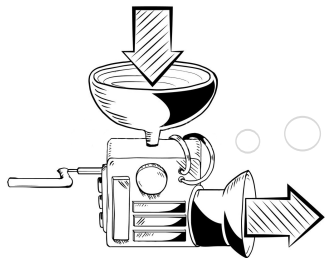


# Satisfiability Solving (Highly Simplified)

**SAT problem:**

Given a propositional formula, is it satisfiable?

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

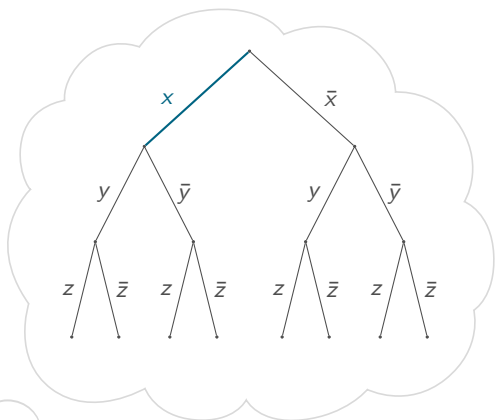
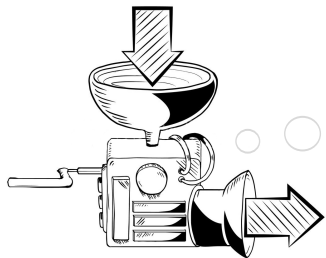


# Satisfiability Solving (Highly Simplified)

## SAT problem:

Given a propositional formula, is it satisfiable?

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

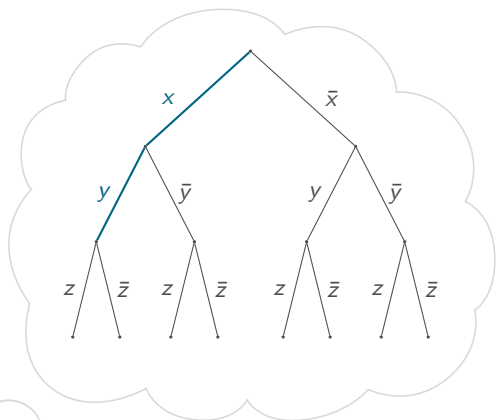
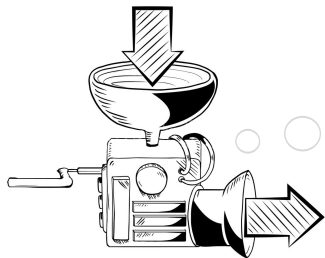


# Satisfiability Solving (Highly Simplified)

**SAT problem:**

Given a propositional formula, is it satisfiable?

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

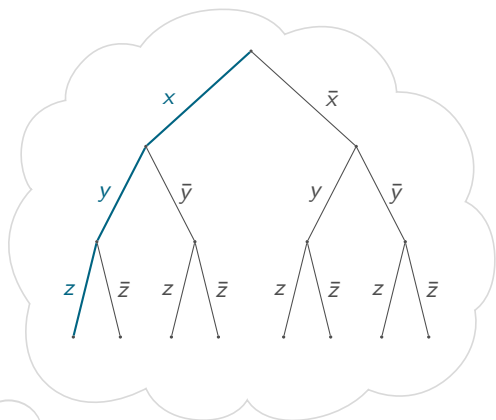
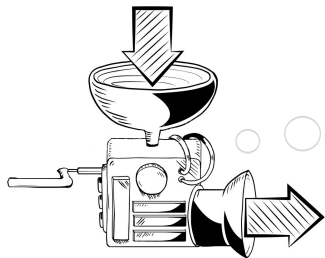


# Satisfiability Solving (Highly Simplified)

**SAT problem:**

Given a propositional formula, is it satisfiable?

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

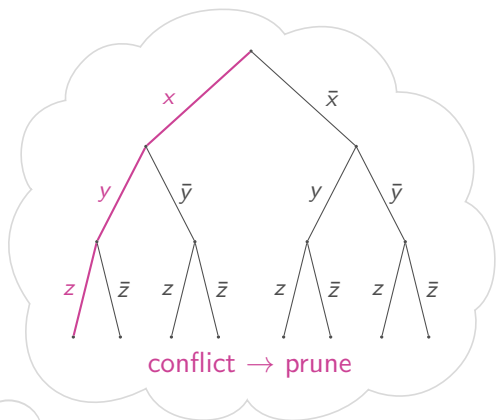
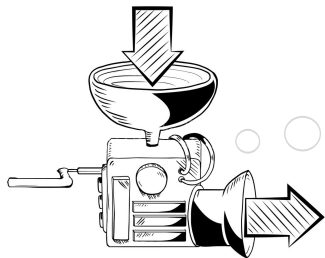


# Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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



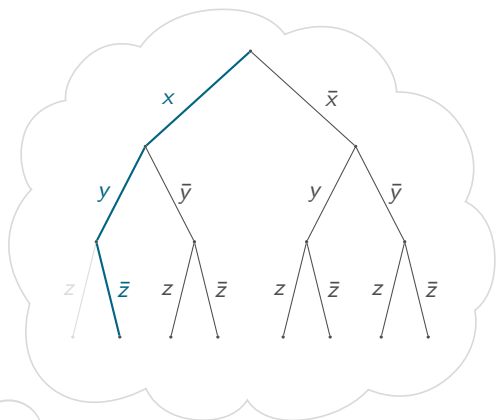
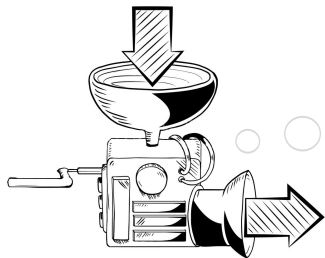


# Satisfiability Solving (Highly Simplified)

**SAT problem:**

Given a propositional formula, is it satisfiable?

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

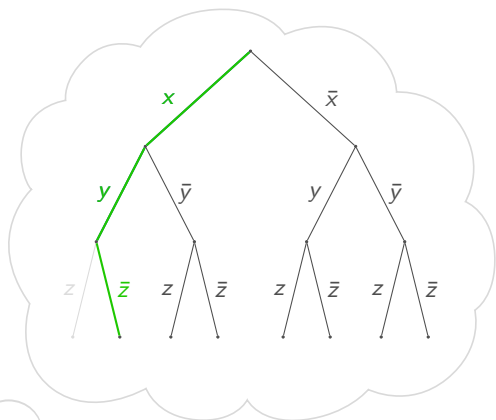
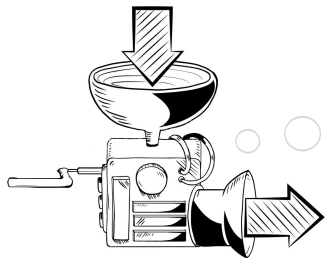


# Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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

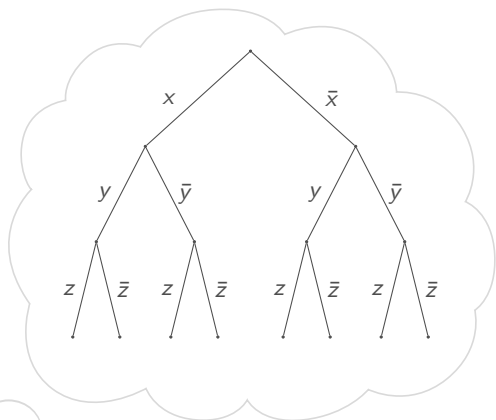
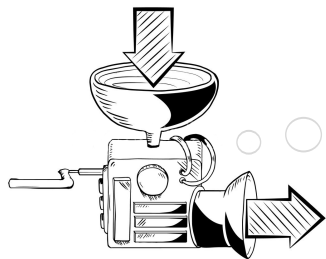


Satisfiable

# Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?  
Even satisfiable branches?

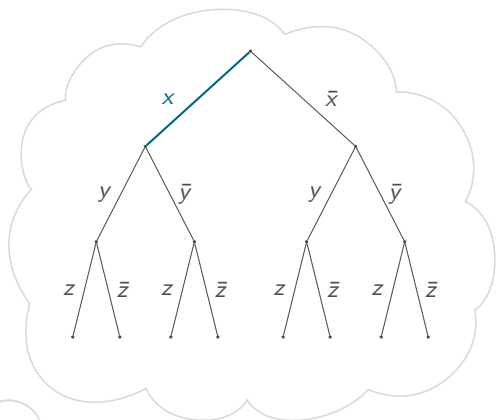
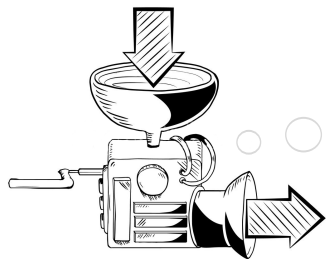
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



## Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?  
Even satisfiable branches?

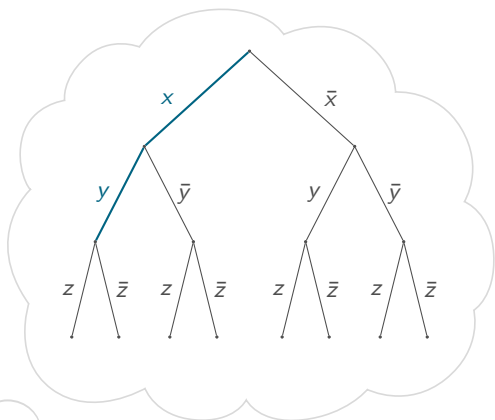
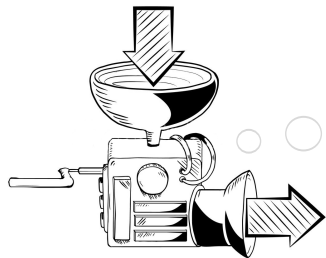
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



# Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?  
Even satisfiable branches?

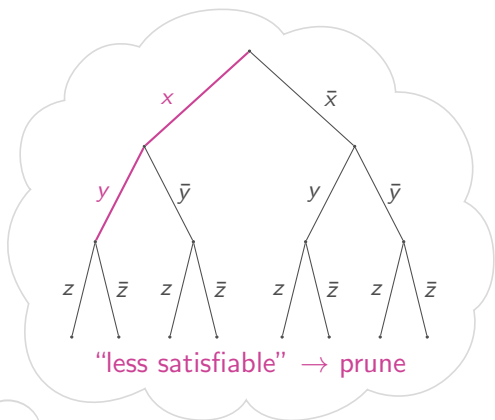
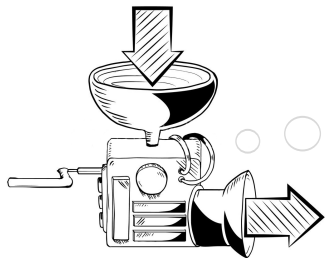
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



# Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?  
Even satisfiable branches?

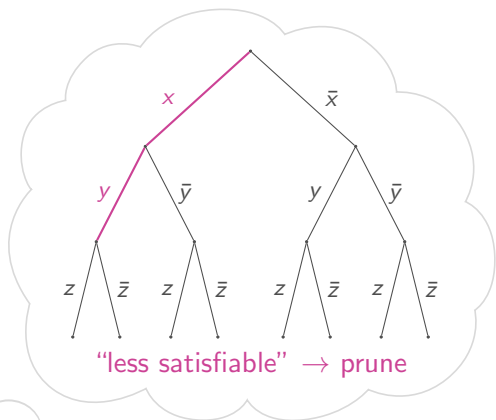
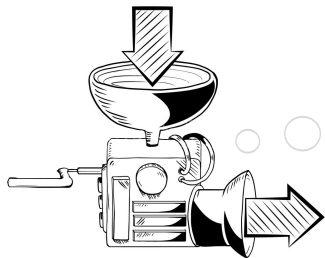
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



# Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?  
Even satisfiable branches?

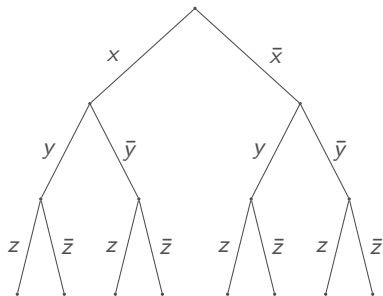
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



How to prune? Add **redundant** clauses!

## Pruning via Clause Addition

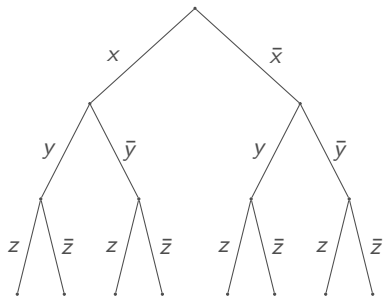
- A clause prunes all branches that falsify the clause.





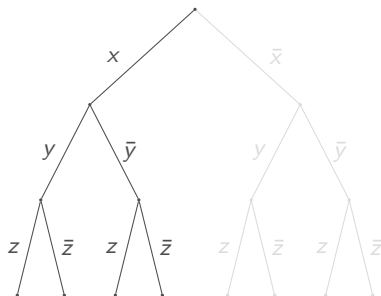
## Pruning via Clause Addition

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



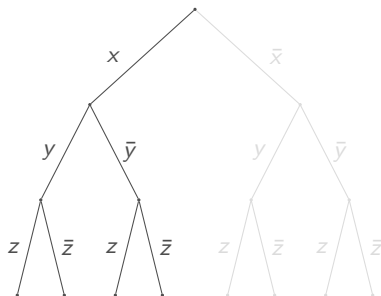
## Pruning via Clause Addition

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



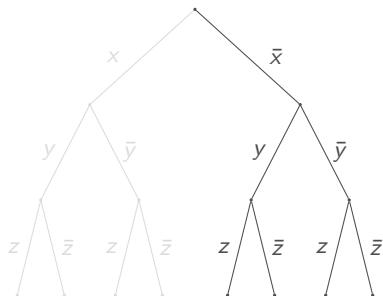
## Pruning via Clause Addition

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



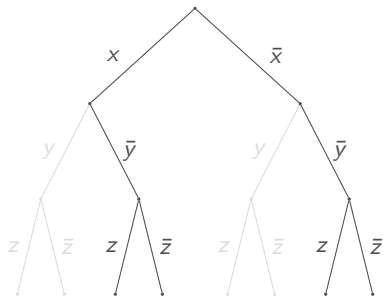
## Pruning via Clause Addition

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



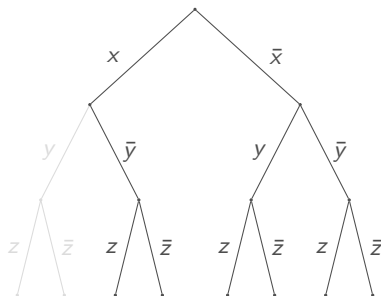
## Pruning via Clause Addition

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



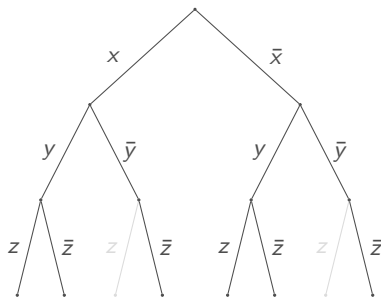
## Pruning via Clause Addition

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



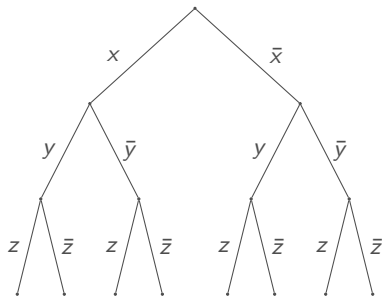
## Pruning via Clause Addition

- 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:**  $(\bar{x})$   $(\bar{y})$   $(\bar{x} \vee \bar{y})$   $(y \vee \bar{z})$   $(x \vee \bar{x})$





Introduction

The Positive Reduct

Conditional Autarkies

The Algorithm

Evaluation

Conclusions and Future Work

# The Positive Reduct

## Traditional Proofs vs. Interference-Based Proofs

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

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

# Traditional Proofs vs. Interference-Based Proofs

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

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

- ➔ Inference rules reason about the **presence** of facts.
  - If certain premises are present, infer the conclusion.

# Traditional Proofs vs. Interference-Based Proofs

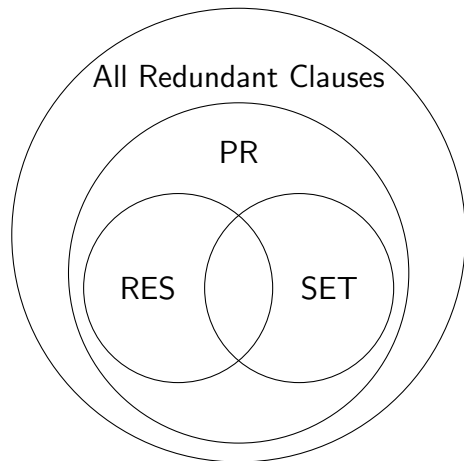
- In **traditional** proof systems, everything that is **inferred**, is **logically implied** by the premises.

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

- ➔ 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**.

## 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  $\alpha$  denote the smallest assignment that falsifies  $C$ . The **positive reduct** of  $F$  and  $\alpha$  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  $\alpha$  denote the smallest assignment that falsifies  $C$ . The **positive reduct** of  $F$  and  $\alpha$  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  $\alpha$  denote the smallest assignment that falsifies  $C$ . The **positive reduct** of  $F$  and  $\alpha$  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  $\alpha$  is **satisfiable**. In that case, **prune** the branch  $\alpha$ .

## The Positive Reduct: An Example

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

### 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**.

# Conditional Autarkies

## Autarkies

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

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  $\alpha$  is an **autarky** for formula  $F$  if every clause  $C \in F$  that is **touched** by  $\alpha$  is also **satisfied** by  $\alpha$ .

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  $\alpha$ ,  $F|_{\alpha}$  denotes a formula  $F$  without the clauses satisfied by  $\alpha$  and without the literals falsified by  $\alpha$ .

### Theorem ([Monien and Speckenmeyer 1985])

*Let  $\alpha$  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  $\alpha_{\text{aut}}$  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  $\alpha_{\text{aut}}$  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  $\alpha$ . Every satisfying assignment  $\omega$  of  $p(F, \alpha)$  is a conditional autarky of  $F$ .*

Recall: Given a formula  $F$  and a clause  $C$ . Let  $\alpha$  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  $\omega$  satisfy  $p(F, \alpha)$ . Removing all but one of the literals in  $C$  that are satisfied by  $\omega$  results in a **PR clause** w.r.t.  $F$ .

# The Algorithm

## Pseudo-Code of CDCL (formula $F$ )

```
1    $\alpha := \emptyset$ 
2   forever do
3      $\alpha := \text{Simplify} (F, \alpha)$ 
4     if  $F|_{\alpha}$  contains a falsified clause then
5        $C := \text{AnalyzeConflict} ()$ 
6       if  $C$  is the empty clause then return unsatisfiable
7        $F := F \cup \{C\}$ 
8        $\alpha := \text{BackJump} (C, \alpha)$ 
13  else
14     $I := \text{Decide} ()$ 
15    if  $I$  is undefined then return satisfiable
16     $\alpha := \alpha \cup \{I\}$ 
```

## Pseudo-Code of SDCL (formula $F$ )

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

# Evaluation

## Benchmark Suite: Pigeon Hole Formulas

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

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

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

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

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

OF = 32-bit overflow



## Results on Large Pigeon Hole Formulas

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

OF = 32-bit overflow

TO = timeout of 9000 seconds

# Conclusions and Future Work

## Conclusions

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.

Our SDCL solver finds short proofs of pigeon hole formulas:

- Integrated in the state-of-the-art solver **Lingeling**;
- Linear sized proofs in  $\mathcal{O}(n^3)$  can be found fully automatically;
- The implementation is efficient, **robust**, and open source.

## Future Work

- SDCL likely requires different heuristics compared to CDCL
- Can more branches be pruned using stronger SAT calls?
- How to minimize clauses from pruning through satisfaction?
- Can SLS techniques be used to find conditional autarkies?

# PRuning Through Satisfaction

**Marijn J.H. Heule**, Benjamin Kiesl,  
Martina Seidl, and Armin Biere

UT Austin, Vienna University of Technology, and JKU Linz



Haifa Verification Conference

November 15, 2017