

Resolution Proof Systems and Clause Learning Satisfiability Algorithms

Jan Johannsen

Institut für Informatik
LMU München

LCC, 15. 07. 2007

Outline

Proof Complexity

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

Outline

Proof Complexity

DLL Algorithms

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

Outline

Proof Complexity

DLL Algorithms

Resolution Trees with Lemmas

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

Outline

Proof Complexity

DLL Algorithms

Resolution Trees with Lemmas

A Lower Bound

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

Outline

Proof Complexity

Motivation

Resolution

Refinements

DLL Algorithms

Resolution Trees with Lemmas

A Lower Bound

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

Motivation

Resolution

Refinements

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

Propositional Proof Complexity

Resolution and
Clause Learning

Jan Johannsen

Main Objective

*Study the length of proofs in calculi for
propositional logic*

Proof Complexity

Motivation

Resolution

Refinements

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

Main Objective

Study the length of proofs in calculi for propositional logic

Motivation:

► Complexity Theory

polynomially bounded proof system \rightarrow **NP** = **co-NP**
 \rightsquigarrow “Cook’s program”

Main Objective

Study the length of proofs in calculi for propositional logic

Motivation:

- ▶ Complexity Theory
- ▶ Independence results for “feasible theories”
 - cf. A. Beckmann & JJ, *Bounded Arithmetic and Resolution-Based Proof Systems*, 2004

Propositional Proof Complexity

Main Objective

Study the length of proofs in calculi for propositional logic

Motivation:

- ▶ Complexity Theory
- ▶ Independence results for “feasible theories”
- ▶ Efficiency of SAT algorithms

Proof Systems and SAT algorithms

Resolution and
Clause Learning

Jan Johannsen

Observation

Let A be a complete SAT algorithm.

Proof Complexity

Motivation

Resolution

Refinements

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

Proof Systems and SAT algorithms

Resolution and
Clause Learning

Jan Johannsen

Observation

Let A be a complete SAT algorithm.

Run of A on unsatisfiable F is a **proof** that $F \notin \text{SAT}$

Proof Complexity

Motivation

Resolution

Refinements

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

Proof Systems and SAT algorithms

Observation

Let A be a complete SAT algorithm.

Run of A on unsatisfiable F is a proof that $F \notin \text{SAT}$

\rightsquigarrow A induces **proof system** P_A

Observation

Let A be a complete SAT algorithm.

Run of A on unsatisfiable F is a proof that $F \notin \text{SAT}$

\rightsquigarrow A induces proof system P_A

P_A may seem artificial, but is natural for many A .

Observation

Let A be a complete SAT algorithm.

Run of A on unsatisfiable F is a proof that $F \notin \text{SAT}$

\rightsquigarrow A induces proof system P_A

P_A may seem artificial, but is natural for many A .

For common backtracking algorithms (DLL),
it is related to **Resolution**

The Resolution rule

Clause: disjunction $a_1 \vee \dots \vee a_k$ of **literals** $a_i = x$ or $a_i = \bar{x}$.

The Resolution rule

Clause: disjunction $a_1 \vee \dots \vee a_k$ of literals $a_i = x$ or $a_i = \bar{x}$.

The **width** of C is $w(C) := k$.

The Resolution rule

Clause: disjunction $a_1 \vee \dots \vee a_k$ of literals $a_i = x$ or $a_i = \bar{x}$.

The width of C is $w(C) := k$.

Formula (in CNF): conjunction $C_1 \wedge \dots \wedge C_m$ of clauses.

The Resolution rule

Clause: disjunction $a_1 \vee \dots \vee a_k$ of literals $a_i = x$ or $a_i = \bar{x}$.

The width of C is $w(C) := k$.

Formula (in CNF): conjunction $C_1 \wedge \dots \wedge C_m$ of clauses.

Resolution rule

If C, D are clauses with $x \in C$ and $\bar{x} \in D$, then

$$\text{Res}_x(C, D) := (C \setminus x) \cup (D \setminus \bar{x})$$

Resolution proofs

Definition

A **Resolution derivation** of clause C from formula F is a dag labelled with clauses s.t.

Resolution proofs

Definition

A **Resolution derivation** of clause C from formula F is a dag labelled with clauses s.t.

- ▶ every node has in-degree 0 or 2

Resolution proofs

Definition

A **Resolution derivation** of clause C from formula F is a dag labelled with clauses s.t.

- ▶ every node has in-degree 0 or 2
- ▶ there is exactly one sink labelled C

Definition

A **Resolution derivation** of clause C from formula F is a dag labelled with clauses s.t.

- ▶ every node has in-degree 0 or 2
- ▶ there is exactly one sink labelled C
- ▶ If v has 2 predecessors u and u' , then

$$C_v \supseteq \text{Res}_x(C_u, C_{u'})$$

for some variable x

Definition

A **Resolution derivation** of clause C from formula F is a dag labelled with clauses s.t.

- ▶ every node has in-degree 0 or 2
- ▶ there is exactly one sink labelled C
- ▶ If v has 2 predecessors u and u' , then

$$C_v \supseteq \text{Res}_x(C_u, C_{u'})$$

for some variable x

- ▶ if v is a source, then $C_v \in F$

Definition

A Resolution derivation of clause C from formula F is a dag labelled with clauses s.t.

- ▶ every node has in-degree 0 or 2
- ▶ there is exactly one sink labelled C
- ▶ If v has 2 predecessors u and u' , then

$$C_v \supseteq \text{Res}_x(C_u, C_{u'})$$

for some variable x

- ▶ if v is a source, then $C_v \in F$

A **Resolution refutation** of F is a derivation of the empty clause \square from F .

Regular and Tree Resolution

Definition

Resolution refutation is **tree-like**, if the underlying dag is a tree.

Tree-like Resolution is denoted Res^* .

Regular and Tree Resolution

Definition

Resolution refutation is **tree-like**, if the underlying dag is a tree.

Tree-like Resolution is denoted Res^* .

Definition

Resolution refutation is **regular**, if on no path a variable is eliminated twice.

Regular Resolution is denoted regRes .

Regular and Tree Resolution

Definition

Resolution refutation is **tree-like**, if the underlying dag is a tree.

Tree-like Resolution is denoted Res^* .

Definition

Resolution refutation is **regular**, if on no path a variable is eliminated twice.

Regular Resolution is denoted regRes .

Proposition

$$\text{Res}^* \leq \text{regRes} \leq \text{Res}$$

Theorem (Ben-Sasson et al.)

There are formulae F_n such that

- ▶ *there are regRes-proofs of F_n of size $n^{O(1)}$.*
- ▶ *Res*-proofs of F_n require size $2^{\Omega(n/\log n)}$,*

Theorem (Ben-Sasson et al.)

There are formulae F_n such that

- ▶ *there are regRes-proofs of F_n of size $n^{O(1)}$.*
- ▶ *Res*-proofs of F_n require size $2^{\Omega(n/\log n)}$,*

Theorem (Alekhnovich et al.)

There are formulae G_n such that

- ▶ *there are Res-proofs of G_n of size $n^{O(1)}$.*
- ▶ *regRes-proofs of G_n require size $2^{\Omega(n)}$,*

Theorem (Ben-Sasson et al.)

There are formulae F_n such that

- ▶ *there are regRes-proofs of F_n of size $n^{O(1)}$.*
- ▶ *Res*-proofs of F_n require size $2^{\Omega(n/\log n)}$,*

Theorem (Alekhnovich et al.)

There are formulae G_n such that

- ▶ *there are Res-proofs of G_n of size $n^{O(1)}$.*
- ▶ *regRes-proofs of G_n require size $2^{\Omega(n)}$,*

Thus: $\text{Res}^* \not\leq \text{regRes} \not\leq \text{Res}$

Outline

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

The Basic Algorithm

Tree Resolution

Clause Learning

Resolution Trees
with Lemmas

A Lower Bound

Proof Complexity

DLL Algorithms

The Basic Algorithm

Tree Resolution

Clause Learning

Resolution Trees with Lemmas

A Lower Bound

Algorithm DLL (Davis, Logemann, Loveland 1962)

```
procedure sat( $F, \alpha$ )
  simplify( $F, \alpha$ )
  if  $F\alpha = \emptyset$  then return TRUE
  if  $\square \in F\alpha$  then return FALSE
  choose literal  $a$  in  $F\alpha$ 
    if sat( $F, \alpha[a := 0]$ )  $\neq \perp$  then return  $\alpha[a := 0]$ 
    if sat( $F, \alpha[a := 1]$ )  $\neq \perp$  then return  $\alpha[a := 1]$ 
  return  $\perp$ 
```

Algorithm DLL (Davis, Logemann, Loveland 1962)

```
procedure sat( $F, \alpha$ )  
  simplify( $F, \alpha$ )  
  if  $F\alpha = \emptyset$  then return TRUE  
  if  $\square \in F\alpha$  then return FALSE  
  choose literal  $a$  in  $F\alpha$   
    if sat( $F, \alpha[a := 0]$ )  $\neq \perp$  then return  $\alpha[a := 0]$   
    if sat( $F, \alpha[a := 1]$ )  $\neq \perp$  then return  $\alpha[a := 1]$   
  return  $\perp$ 
```

E.g.: remove pure literals
unit propagation

Algorithm DLL (Davis, Logemann, Loveland 1962)

```
procedure sat( $F, \alpha$ )
  simplify( $F, \alpha$ )
  if  $F\alpha = \emptyset$  then return TRUE
  if  $\square \in F\alpha$  then return FALSE
  choose literal  $a$  in  $F\alpha$ 
    if sat( $F, \alpha[a := 0]$ )  $\neq \perp$  then return  $\alpha[a := 0]$ 
    if sat( $F, \alpha[a := 1]$ )  $\neq \perp$  then return  $\alpha[a := 1]$ 
  return  $\perp$ 
```

E.g.: choose a in a shortest clause
choose most frequent a

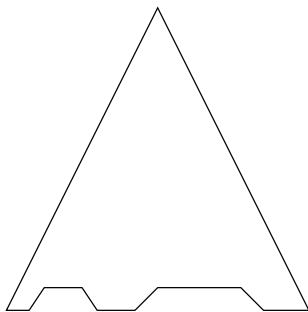
DLL and tree resolution

Let F be unsatisfiable.

DLL and tree resolution

Let F be unsatisfiable.

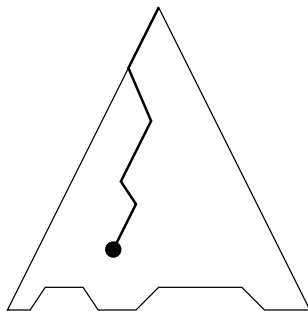
- ▶ Run of DLL on $F \rightsquigarrow$ search tree



DLL and tree resolution

Let F be unsatisfiable.

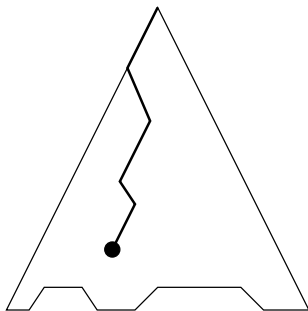
- ▶ Run of DLL on $F \rightsquigarrow$ search tree
- ▶ path to $v \hat{=} \text{assignment } \alpha_v$



DLL and tree resolution

Let F be unsatisfiable.

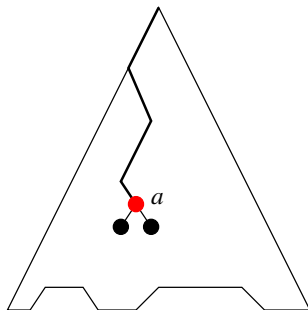
- ▶ Run of DLL on $F \rightsquigarrow$ search tree
- ▶ path to $v \hat{=}$ assignment α_v
- ▶ for each v : C_v with $\alpha_v(C_v) = 0$



DLL and tree resolution

Let F be unsatisfiable.

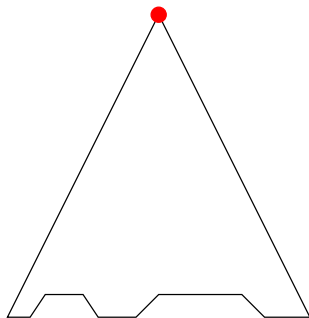
- ▶ Run of DLL on $F \rightsquigarrow$ search tree
- ▶ path to $v \hat{=}$ assignment α_v
- ▶ for each v : C_v with $\alpha_v(C_v) = 0$
 - ▶ Leaf v
 α_v sets a_1, \dots, a_k to 1
 \rightsquigarrow clause $C_v \subseteq \bar{a}_1 \vee \dots \vee \bar{a}_k$ in F
 - ▶ Interior node v
at its sons: $D \vee a$ and $D' \vee \bar{a}$
 \rightsquigarrow set $C_v := D \vee D'$



DLL and tree resolution

Let F be unsatisfiable.

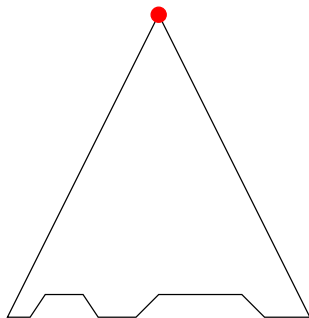
- ▶ Run of DLL on $F \rightsquigarrow$ search tree
- ▶ path to $v \hat{=}$ assignment α_v
- ▶ for each v : C_v with $\alpha_v(C_v) = 0$
 - ▶ Leaf v
 α_v sets a_1, \dots, a_k to 1
 \rightsquigarrow clause $C_v \subseteq \bar{a}_1 \vee \dots \vee \bar{a}_k$ in F
 - ▶ Interior node v
at its sons: $D \vee a$ and $D' \vee \bar{a}$
 \rightsquigarrow set $C_v := D \vee D'$
 - ▶ Root w : $C_w = \square$



DLL and tree resolution

Let F be unsatisfiable.

- ▶ Run of DLL on $F \rightsquigarrow$ search tree
- ▶ path to $v \hat{=}$ assignment α_v
- ▶ for each v : C_v with $\alpha_v(C_v) = 0$
 - ▶ Leaf v
 α_v sets a_1, \dots, a_k to 1
 \rightsquigarrow clause $C_v \subseteq \bar{a}_1 \vee \dots \vee \bar{a}_k$ in F
 - ▶ Interior node v
at its sons: $D \vee a$ and $D' \vee \bar{a}$
 \rightsquigarrow set $C_v := D \vee D'$
 - ▶ Root w : $C_w = \square$



\rightsquigarrow Res*-refutation of F

Unit propagation

```
procedure unit( $F, \alpha$ )
  while there is unit clause  $a$  in  $F\alpha$ 
     $\alpha := \alpha[a := 1]$ 
  return  $\alpha$ 
```

Unit propagation

```
procedure unit( $F, \alpha$ )  
  while there is unit clause  $a$  in  $F\alpha$   
     $\alpha := \alpha[a := 1]$   
  return  $\alpha$ 
```

Unit propagation is the “workhorse” of SAT-solvers.

Unit propagation

```
procedure unit( $F, \alpha$ )  
  while there is unit clause  $a$  in  $F\alpha$   
     $\alpha := \alpha[a := 1]$   
  return  $\alpha$ 
```

Unit propagation is the “workhorse” of SAT-solvers.

For clause learning: record **reason** with any variable set.

The conflict graph

$a \vee x \vee \bar{y}$

$\bar{b} \vee \bar{x} \vee z$

$\bar{c} \vee y \vee z$

$x \vee \bar{z} \vee \bar{w}$

$y \vee w$

The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$$a := 0$$

$$b := 1$$

$$c := 1$$

The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$$a := 0$$

$$b := 1$$

$$c := 1$$

$$a := 0$$

$$b := 1$$

$$c := 1$$

The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$$a := 0$$

$$b := 1$$

$$c := 1$$

$$x := 0$$

$$a := 0$$

$$b := 1$$

$$x := 0$$

$$c := 1$$

The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

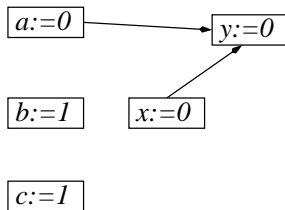
$a := 0$

$b := 1$

$c := 1$

$x := 0$

$y := 0$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

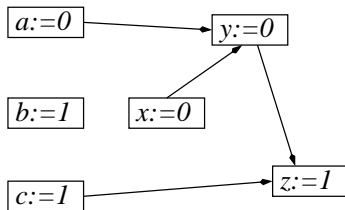
$b := 1$

$c := 1$

$x := 0$

$y := 0$

$z := 1$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$$a := 0$$

$$b := 1$$

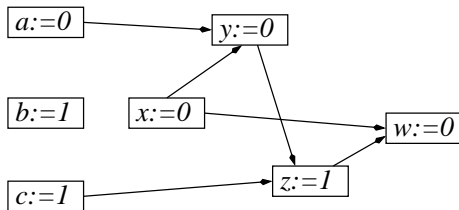
$$c := 1$$

$$x := 0$$

$$y := 0$$

$$z := 1$$

$$w := 0$$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

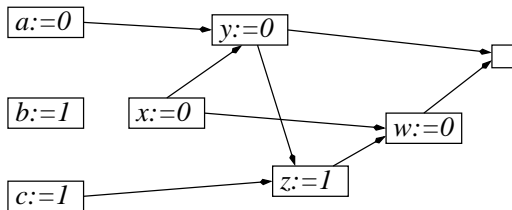
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

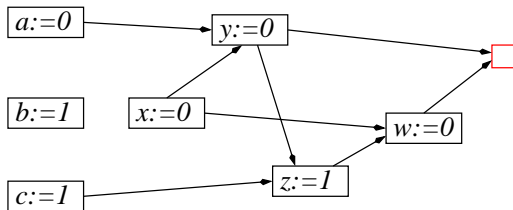
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

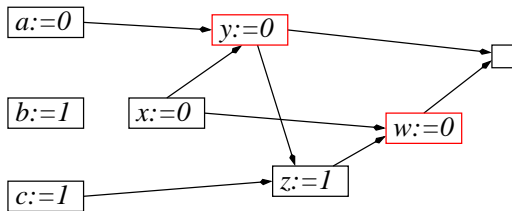
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

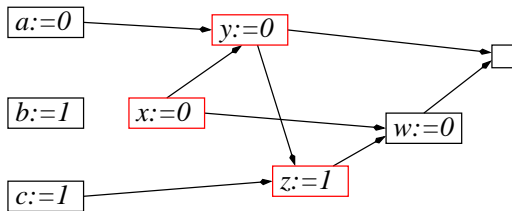
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

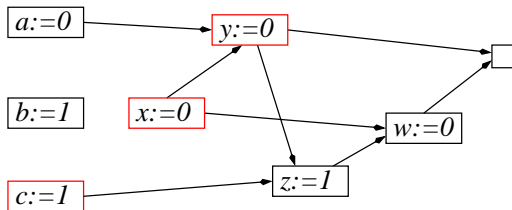
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

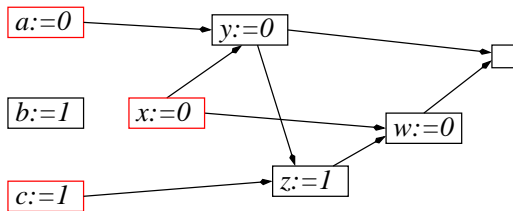
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

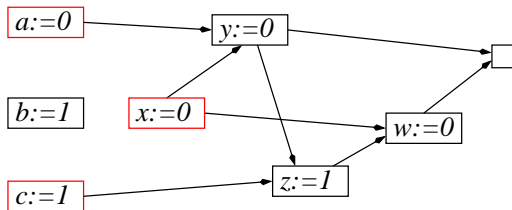
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



Clause learned: $a \vee \bar{c} \vee x$

The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

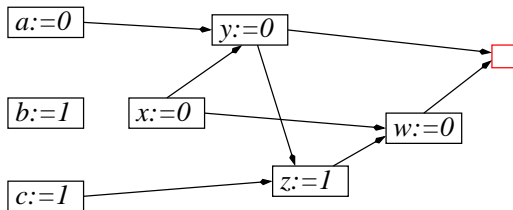
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



Clause learned: $a \vee \bar{c} \vee x$

The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

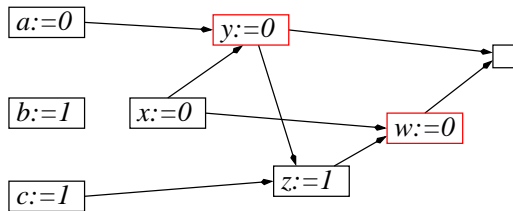
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



Clause learned: $a \vee \bar{c} \vee x$

The conflict graph

$$a \vee x \vee \bar{y}$$

$$\bar{b} \vee \bar{x} \vee z$$

$$\bar{c} \vee y \vee z$$

$$x \vee \bar{z} \vee \bar{w}$$

$$y \vee w$$

$a := 0$

$b := 1$

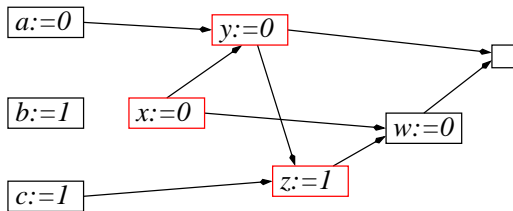
$c := 1$

$x := 0$

$y := 0$

$z := 1$

$w := 0$



Clause learned: $a \vee \bar{c} \vee x$

The learned clause

$$a \vee x \vee \bar{y} \quad \bar{b} \vee \bar{x} \vee z \quad \bar{c} \vee \bar{x} \vee z \quad x \vee \bar{z} \vee \bar{w} \quad y \vee w$$

Computation of learned clause $\hat{=}$ Resolution derivation:

$$\frac{\frac{\frac{y \vee w}{x \vee y \vee \bar{z}} \quad x \vee \bar{z} \vee \bar{w}}{\bar{c} \vee x \vee y} \quad \bar{c} \vee \bar{x} \vee z}{a \vee \bar{c} \vee x} \quad a \vee x \vee \bar{y}}$$

The learned clause

$$a \vee x \vee \bar{y} \quad \bar{b} \vee \bar{x} \vee z \quad \bar{c} \vee \bar{x} \vee z \quad x \vee \bar{z} \vee \bar{w} \quad y \vee w$$

Computation of learned clause $\hat{=}$ Resolution derivation:

$$\frac{\frac{\frac{y \vee w}{x \vee y \vee \bar{z}} \quad x \vee \bar{z} \vee \bar{w}}{\bar{c} \vee x \vee y} \quad \bar{c} \vee \bar{x} \vee z}{a \vee \bar{c} \vee x} \quad a \vee x \vee \bar{y}}$$

Therefore: adding does not affect satisfiability!

SAT algorithms and Resolution

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

The Basic Algorithm

Tree Resolution

Clause Learning

Resolution Trees
with Lemmas

A Lower Bound

DLL = tree-like Resolution

SAT algorithms and Resolution

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

The Basic Algorithm

Tree Resolution

Clause Learning

Resolution Trees
with Lemmas

A Lower Bound

DLL with learning

DLL = tree-like Resolution

SAT algorithms and Resolution

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

The Basic Algorithm

Tree Resolution

Clause Learning

Resolution Trees
with Lemmas

A Lower Bound

DLL with learning
and restarts

DLL with learning

DLL = tree-like Resolution

SAT algorithms and Resolution

DLL with learning and restarts = general Resolution

DLL with learning

DLL = tree-like Resolution

SAT algorithms and Resolution

DLL with learning and restarts = general Resolution

DLL with learning = regular Resolution

DLL = tree-like Resolution

SAT algorithms and Resolution

DLL with learning and restarts = general Resolution

DLL with learning > regular Resolution

DLL = tree-like Resolution

SAT algorithms and Resolution

DLL with learning and restarts = general Resolution

DLL with learning = Resolution Trees with Lemmas
regular Resolution

DLL = tree-like Resolution

Outline

Proof Complexity

DLL Algorithms

Resolution Trees with Lemmas

Definition

Regularity

Input Lemmas and Clause Learning

A Lower Bound

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

Resolution Trees
with Lemmas

Definition

Regularity

Input Lemmas and
Clause Learning

A Lower Bound

Resolution Trees with Lemmas

Definition

A **Resolution tree with lemmas** (RTL) for formula F is an ordered binary tree labelled with clauses s.t.

Resolution Trees with Lemmas

Definition

A **Resolution tree with lemmas** (RTL) for formula F is an ordered binary tree labelled with clauses s.t.

- ▶ $C_{\text{root}} = \square$

Resolution Trees with Lemmas

Definition

A **Resolution tree with lemmas** (RTL) for formula F is an ordered binary tree labelled with clauses s.t.

- ▶ $C_{\text{root}} = \square$
- ▶ If v has 2 predecessors u and u' , then

$$C_v \supseteq \text{Res}_x(C_u, C_{u'})$$

for some variable x

Resolution Trees with Lemmas

Definition

A **Resolution tree with lemmas** (RTL) for formula F is an ordered binary tree labelled with clauses s.t.

- ▶ $C_{\text{root}} = \square$
- ▶ If v has 2 predecessors u and u' , then

$$C_v \supseteq \text{Res}_x(C_u, C_{u'})$$

for some variable x

- ▶ if v is a leaf, then

$$C_v \in F \text{ or } C_v = C_u \text{ for some } u \prec v$$

\prec is the **post-order** on trees.

Regular RTL

Observation

RTL is equivalent to (dag-like) Resolution.

Regular RTL

Observation

RTL is equivalent to (dag-like) Resolution.

Definition

A regular Resolution tree with lemmas (regRTL)
is an RTL that is regular.

Observation

RTL is equivalent to (dag-like) Resolution.

Definition

A regular Resolution tree with lemmas (regRTL)
is an RTL that is regular.

Proposition

$$\text{regRes} \leq \text{regRTL} \leq \text{Res} = \text{RTL}$$

Theorem (Hoffmann, following Beame et al.)

There are formulae F_n such that

- ▶ *there are regRTL-proofs of F_n of size $n^{O(1)}$.*
- ▶ *regRes-proofs of F_n require size $2^{\Omega(n)}$,*

Thus: $\text{regRes} \not\leq \text{regRTL} \leq \text{Res}$

Theorem (Hoffmann, following Beame et al.)

There are formulae F_n such that

- ▶ there are regRTL -proofs of F_n of size $n^{O(1)}$.
- ▶ regRes -proofs of F_n require size $2^{\Omega(n)}$,

Thus: $\text{regRes} \not\leq \text{regRTL} \leq \text{Res}$

Open Problem

Can regRTL be separated from Res ?

Definition

Clause C in an RTL is derived by **input resolution**, if in the subtree rooted at C , every node has a child that is a leaf.

Definition

Clause C in an RTL is derived by **input resolution**, if in the subtree rooted at C , every node has a child that is a leaf.

A **Resolution tree with input lemmas** (RTI) is defined as an RTL, with the modified clause:

Definition

Clause C in an RTL is derived by **input resolution**, if in the subtree rooted at C , every node has a child that is a leaf.

A **Resolution tree with input lemmas** (RTI) is defined as an RTL, with the modified clause:

- ▶ if v is a leaf, then $C_v \in F$ or $C_v = C_u$ for some $u \prec v$ with C_u derived by input resolution.

The learned clause

$$a \vee x \vee \bar{y} \quad \bar{b} \vee \bar{x} \vee z \quad \bar{c} \vee \bar{x} \vee z \quad x \vee \bar{z} \vee \bar{w} \quad y \vee w$$

Computation of learned clause $\hat{=}$ Resolution derivation:

$$\frac{\frac{\frac{y \vee w}{x \vee y \vee \bar{z}} \quad x \vee \bar{z} \vee \bar{w}}{\bar{c} \vee x \vee y} \quad \bar{c} \vee \bar{x} \vee z}{a \vee \bar{c} \vee x} \quad a \vee x \vee \bar{y}}$$

Theorem (Hoffmann)

Let A be a DLL algorithm with clause learning.

*If A takes r recursive calls on unsatisfiable formula F ,
then F has a regRTI-refutation of size $O(nr)$.*

Theorem (Hoffmann)

Let A be a DLL algorithm with clause learning.

*If A takes r recursive calls on unsatisfiable formula F ,
then F has a regRTI-refutation of size $O(nr)$.*

Proposition

$$\text{regRes} \leq \text{regRTI} \leq \text{regRTL}$$

Outline

Proof Complexity

DLL Algorithms

Resolution Trees with Lemmas

A Lower Bound

The Pigeonhole Principle

The lower bound

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

The Pigeonhole
Principle

The lower bound

The Pigeonhole Principle

... says: There is no injective map $[n + 1] \rightarrow [n]$

The Pigeonhole Principle

... says: There is no injective map $[n + 1] \rightarrow [n]$

The formula PHP_n :

▶ variables $x_{i,j}$ for $i \leq n + 1$ and $j \leq n$

The Pigeonhole Principle

... says: There is no injective map $[n + 1] \rightarrow [n]$

The formula PHP_n :

- ▶ variables $x_{i,j}$ for $i \leq n + 1$ and $j \leq n$
- ▶ pigeon clauses $x_{i,1} \vee \dots \vee x_{i,n}$ for every i

The Pigeonhole Principle

... says: There is no injective map $[n + 1] \rightarrow [n]$

The formula PHP_n :

- ▶ variables $x_{i,j}$ for $i \leq n + 1$ and $j \leq n$
- ▶ pigeon clauses $x_{i,1} \vee \dots \vee x_{i,n}$ for every i
- ▶ hole clauses $\bar{x}_{i,j} \vee \bar{x}_{i',j}$ for $i < i'$

Complexity of the Pigeonhole Principle

Resolution and
Clause Learning

Jan Johannsen

Proof Complexity

DLL Algorithms

Resolution Trees
with Lemmas

A Lower Bound

The Pigeonhole
Principle

The lower bound

Theorem (Haken 1985)

Resolution Proofs of PHP_n require size $2^{\Omega(n)}$.

Complexity of the Pigeonhole Principle

Theorem (Haken 1985)

Resolution Proofs of PHP_n require size $2^{\Omega(n)}$.

Theorem (Buss, Pitassi 1997)

There are Resolution proofs of PHP_n of size $n^3 2^n$.

Complexity of the Pigeonhole Principle

Theorem (Haken 1985)

Resolution Proofs of PHP_n require size $2^{\Omega(n)}$.

Theorem (Buss, Pitassi 1997)

There are Resolution proofs of PHP_n of size $n^3 2^n$.

Theorem (Iwama, Miyazaki 1999)

Tree-like Resolution Proofs of PHP_n require size $2^{\Omega(n \log n)}$.

The lower bound

Goal: solving PHP_n takes long when learning only short clauses.

The lower bound

Goal: solving PHP_n takes long when learning only short clauses.

Show lower bound for $\text{RTL}(k)$:

A refutation R in RTL is in $\text{RTL}(k)$, if every lemma C used in R is of width $w(C) \leq k$.

The lower bound

Goal: solving PHP_n takes long when learning only short clauses.

Show lower bound for $\text{RTL}(k)$:

A refutation R in RTL is in $\text{RTL}(k)$, if every lemma C used in R is of width $w(C) \leq k$.

Theorem

For $k \leq n/2$, every $\text{RTL}(k)$ -refutation of PHP_n is of size $2^{\Omega(n \log n)}$.

The lower bound

Goal: solving PHP_n takes long when learning only short clauses.

Show lower bound for $\text{RTL}(k)$:

A refutation R in RTL is in $\text{RTL}(k)$, if every lemma C used in R is of width $w(C) \leq k$.

Theorem

For $k \leq n/2$, every $\text{RTL}(k)$ -refutation of PHP_n is of size $2^{\Omega(n \log n)}$.

Lower bound is shown for PHP_n with functional clauses:

$$\blacktriangleright \bar{x}_{i,j} \vee \bar{x}_{i,j'} \quad \text{for } j < j'$$

Matching restrictions

A **restriction** ρ is a partial truth assignment.

Notation: $F|_{\rho}$ for ρ applied to F .

Matching restrictions

A **restriction** ρ is a partial truth assignment.

Notation: $F \upharpoonright_{\rho}$ for ρ applied to F .

Property: Let R be a Res-derivation of C from F .

There is a Res-derivation R' of $C \upharpoonright_{\rho}$ from $F \upharpoonright_{\rho}$ of size $|R'| \leq |R|$. We denote R' by $R \upharpoonright_{\rho}$.

Matching restrictions

A **restriction** ρ is a partial truth assignment.

Notation: $F \upharpoonright_{\rho}$ for ρ applied to F .

Property: Let R be a Res-derivation of C from F .

There is a Res-derivation R' of $C \upharpoonright_{\rho}$ from $F \upharpoonright_{\rho}$ of size $|R'| \leq |R|$. We denote R' by $R \upharpoonright_{\rho}$.

Matching restriction: defined by $\{(i_1, j_1), \dots, (i_k, j_k)\}$:

$$\rho(x_{i,j}) = \begin{cases} 1 & \text{if } (i, j) \in \rho \\ 0 & \text{if } (i, j') \in \rho \text{ or } (i', j) \in \rho \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Matching restrictions

A **restriction** ρ is a partial truth assignment.

Notation: $F \upharpoonright_{\rho}$ for ρ applied to F .

Property: Let R be a Res-derivation of C from F .

There is a Res-derivation R' of $C \upharpoonright_{\rho}$ from $F \upharpoonright_{\rho}$ of size $|R'| \leq |R|$. We denote R' by $R \upharpoonright_{\rho}$.

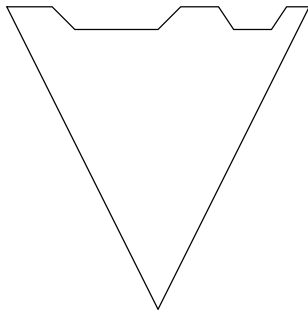
Matching restriction: defined by $\{(i_1, j_1), \dots, (i_k, j_k)\}$:

$$\rho(x_{i,j}) = \begin{cases} 1 & \text{if } (i, j) \in \rho \\ 0 & \text{if } (i, j') \in \rho \text{ or } (i', j) \in \rho \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Property: $\text{PHP}_n \upharpoonright_{\rho} \equiv \text{PHP}_{n-|\rho|}$.

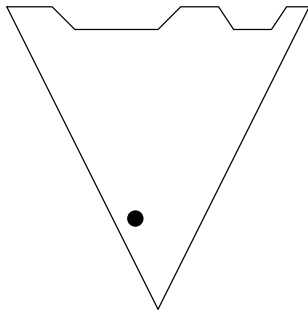
Proof of the lower bound

- ▶ Let R be a refutation of PHP_n



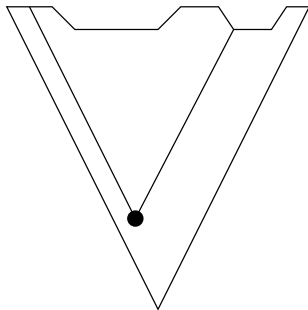
Proof of the lower bound

- ▶ Let R be a refutation of PHP_n
- ▶ Find first C with $w(C) \leq k$



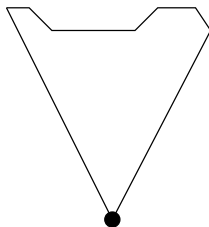
Proof of the lower bound

- ▶ Let R be a refutation of PHP_n
- ▶ Find first C with $w(C) \leq k$
- ▶ Subtree R_C is tree-like derivation of C
- ▶ Pick ρ with $C \upharpoonright_{\rho} = 0$



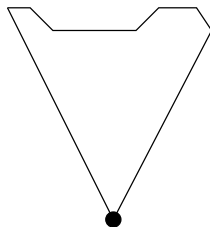
Proof of the lower bound

- ▶ Let R be a refutation of PHP_n
- ▶ Find first C with $w(C) \leq k$
- ▶ Subtree R_C is tree-like derivation of C
- ▶ Pick ρ with $C \upharpoonright_\rho = 0$
- ▶ $R_C \upharpoonright_\rho$ is refutation of $\text{PHP}_n \upharpoonright_\rho$



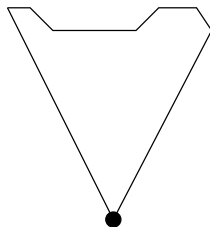
Proof of the lower bound

- ▶ Let R be a refutation of PHP_n
- ▶ Find first C with $w(C) \leq k$
- ▶ Subtree R_C is tree-like derivation of C
- ▶ Pick ρ with $C \upharpoonright_\rho = 0$
- ▶ $R_C \upharpoonright_\rho$ is refutation of $\text{PHP}_n \upharpoonright_\rho$
- ▶ ρ matching restriction \rightarrow
 $\text{PHP}_n \upharpoonright_\rho = \text{PHP}_{n-|\rho|}$



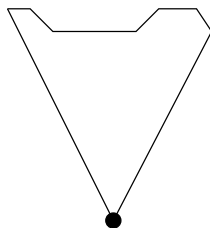
Proof of the lower bound

- ▶ Let R be a refutation of PHP_n
- ▶ Find first C with $w(C) \leq k$
- ▶ Subtree R_C is tree-like derivation of C
- ▶ Pick ρ with $C \upharpoonright_\rho = 0$
- ▶ $R_C \upharpoonright_\rho$ is refutation of $\text{PHP}_n \upharpoonright_\rho$
- ▶ ρ matching restriction \rightarrow
 $\text{PHP}_n \upharpoonright_\rho = \text{PHP}_{n-|\rho|}$
- ▶ lower bound by IWAMA/MIYAZAKI



Proof of the lower bound

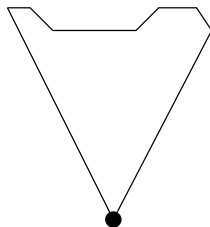
- ▶ Let R be a refutation of PHP_n
- ▶ Find first C with $w(C) \leq k$
- ▶ Subtree R_C is tree-like derivation of C
- ▶ Pick ρ with $C \upharpoonright_{\rho} = 0$
- ▶ $R_C \upharpoonright_{\rho}$ is refutation of $\text{PHP}_n \upharpoonright_{\rho}$
- ▶ ρ matching restriction \rightarrow
 $\text{PHP}_n \upharpoonright_{\rho} = \text{PHP}_{n-|\rho|}$
- ▶ lower bound by IWAMA/MIYAZAKI



Needed: For C in R with $w(C) \leq k$, there is a matching restriction ρ with $C \upharpoonright_{\rho} = 0$ and $|\rho| \leq k$

Proof of the lower bound

- ▶ Let R be a refutation of PHP_n
- ▶ Find first C with $w(C) \leq k$
- ▶ Subtree R_C is tree-like derivation of C
- ▶ Pick ρ with $C \upharpoonright_{\rho} = 0$
- ▶ $R_C \upharpoonright_{\rho}$ is refutation of $\text{PHP}_n \upharpoonright_{\rho}$
- ▶ ρ matching restriction \rightarrow
 $\text{PHP}_n \upharpoonright_{\rho} = \text{PHP}_{n-|\rho|}$
- ▶ lower bound by IWAMA/MIYAZAKI



Needed: For C in R with $w(C) \leq k$, there is a matching restriction ρ with $C \upharpoonright_{\rho} = 0$ and $|\rho| \leq k$

Lemma

For R an $\text{RTL}(k)$ -refutation of F , there is R' that contains no tautologies, and $|R'| \leq |R|$.

Properties of proofs

Lemma

For R an $\text{RTL}(k)$ -refutation of F , there is R' that contains no tautologies, and $|R'| \leq |R|$.

Lemma

For R an $\text{RTL}(k)$ -refutation of F , there is R' that contains no clause $D \supset C$ for $C \in F$, and $|R'| \leq |R|$.

Completing the proof

Main Lemma

Let C be a clause of width $w(C) \leq k \leq n/2$, such that

Completing the proof

Main Lemma

Let C be a clause of width $w(C) \leq k \leq n/2$, such that

- ▶ C is not a tautology

Completing the proof

Main Lemma

Let C be a clause of width $w(C) \leq k \leq n/2$, such that

- ▶ C is not a tautology
- ▶ C not subsumed by hole clause $\bar{x}_{i,j} \vee \bar{x}_{i',j}$
- ▶ C not subsumed by functional clause $\bar{x}_{i,j} \vee \bar{x}_{i,j'}$

Completing the proof

Main Lemma

Let C be a clause of width $w(C) \leq k \leq n/2$, such that

- ▶ C is not a tautology
- ▶ C not subsumed by hole clause $\bar{x}_{i,j} \vee \bar{x}_{i',j}$
- ▶ C not subsumed by functional clause $\bar{x}_{i,j} \vee \bar{x}_{i,j'}$

Then there is a matching restriction ρ with $C|_{\rho} = 0$ and $|\rho| \leq k$.