# Analysis of clause-learning algorithms with many restarts

**Albert Atserias**
Universitat Politècnica de Catalunya
Barcelona

Based on joint work with
Johannes Fichte and Marc Thurley

# Overview of the talk

**Overview**:

1. Satisfiability, resolution, algorithms
2. Propositional proof complexity
3. Analysis

# Part I

# SATISFIABILITY, RESOLUTION, ALGORITHMS

# Satisfiability

**Example**:

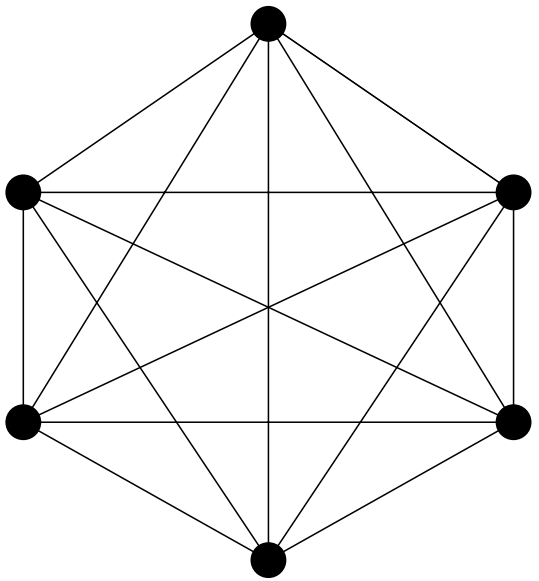
15 variables and 40 clauses

$$
\begin{array}{cccc}
x_1 \vee x_2 \vee x_6 & x_1 \vee x_3 \vee x_7 & x_1 \vee x_4 \vee x_8 & x_1 \vee x_5 \vee x_9 \\
x_2 \vee x_3 \vee x_{10} & x_2 \vee x_4 \vee x_{11} & x_2 \vee x_5 \vee x_{12} & x_3 \vee x_4 \vee x_{13} \\
x_3 \vee x_5 \vee x_{14} & x_4 \vee x_5 \vee x_{15} & x_6 \vee x_7 \vee x_{10} & x_6 \vee x_8 \vee x_{11} \\
x_6 \vee x_9 \vee x_{12} & x_7 \vee x_8 \vee x_{13} & x_7 \vee x_9 \vee x_{14} & x_8 \vee x_9 \vee x_{15} \\
x_{10} \vee x_{11} \vee x_{13} & x_{10} \vee x_{12} \vee x_{14} & x_{11} \vee x_{12} \vee x_{15} & x_{13} \vee x_{14} \vee x_{15} \\
\overline{x_1} \vee \overline{x_2} \vee \overline{x_6} & \overline{x_1} \vee \overline{x_3} \vee \overline{x_7} & \overline{x_1} \vee \overline{x_4} \vee \overline{x_8} & \overline{x_1} \vee \overline{x_5} \vee \overline{x_9} \\
\overline{x_2} \vee \overline{x_3} \vee \overline{x_{10}} & \overline{x_2} \vee \overline{x_4} \vee \overline{x_{11}} & \overline{x_2} \vee \overline{x_5} \vee \overline{x_{12}} & \overline{x_3} \vee \overline{x_4} \vee \overline{x_{13}} \\
\overline{x_3} \vee \overline{x_5} \vee \overline{x_{14}} & \overline{x_4} \vee \overline{x_5} \vee \overline{x_{15}} & \overline{x_6} \vee \overline{x_7} \vee \overline{x_{10}} & \overline{x_6} \vee \overline{x_8} \vee \overline{x_{11}} \\
\overline{x_6} \vee \overline{x_9} \vee \overline{x_{12}} & \overline{x_7} \vee \overline{x_8} \vee \overline{x_{13}} & \overline{x_7} \vee \overline{x_9} \vee \overline{x_{14}} & \overline{x_8} \vee \overline{x_9} \vee \overline{x_{15}} \\
\overline{x_{10}} \vee \overline{x_{11}} \vee \overline{x_{13}} & \overline{x_{10}} \vee \overline{x_{12}} \vee \overline{x_{14}} & \overline{x_{11}} \vee \overline{x_{12}} \vee \overline{x_{15}} & \overline{x_{13}} \vee \overline{x_{14}} \vee \overline{x_{15}}
\end{array}
$$

# Satisfiability

**Example**:

15 variables and 40 clauses



In every party of six,
either three of them are mutual friends,
or three of them are mutual strangers.

# CNF-formulas

**Variables ($x$)**:

$$p_1, \ldots, p_n$$

**Literals ($\ell$)**:

$$x, \overline{x}, x^1, x^0$$

**Clauses ($C$)**:

$$\ell_1 \vee \cdots \vee \ell_w$$

**CNF-Formulas ($F$)**:

$$C_1 \wedge \cdots \wedge C_m$$

**Partial assignments ($\alpha$)**:

$$\alpha : \{p_1, \ldots, p_n\} \to \{0, 1, \star\}$$

**Restriction $F|_\alpha$**:

Assign the variables and simplify.

# Satisfiability

**Satisfies/falsifies** $\ell = x^a$:

$$\text{if } \alpha(x) = a$$
$$\text{if } \alpha(x) = 1 - a$$

**Satisfies/falsifies** $C = \ell_1 \vee \cdots \vee \ell_w$:

$$\text{if it satisfies some } \ell_i \text{ in } C$$
$$\text{if it falsifies every } \ell_i \text{ in } C$$

**Satisfies/falsifies** $F = C_1 \wedge \cdots \wedge C_m$:

$$\text{if it satisfies every } C_i \text{ in } F$$
$$\text{if it falsifies some } C_i \text{ in } F.$$

# Resolution

**Resolution rule**:

$$\frac{A \vee x \qquad B \vee \bar{x}}{A \vee B}$$

**Resolution refutation of** $F = C_1 \wedge \cdots \wedge C_m$:

$$C_1, \ldots, C_m, \ldots, C_i, \ldots, C_j, \ldots, C_k, \ldots, \emptyset$$

**Soundness**:

If $F$ has a refutation, then $F$ is unsatisfiable.

**Completeness**:

If $F$ is unsatisfiable, then $F$ has a refutation (of length $O(2^n)$).

# Naif backtracking algorithm: recursive form

**Satisfiability problem**:

Is there a truth-assignment that satisfies $F$?

**Strengthened satisfiability problem**:

Given a partial $\alpha$, can $\alpha$ be extended to satisfy $F$?

# Naif backtracking algorithm: recursive form

**Satisfiability problem**:

Is there a truth-assignment that satisfies $F$?

**Strengthened satisfiability problem**:

Given a partial $\alpha$, can $\alpha$ be extended to satisfy $F$?

**Algorithm** $A(\alpha)$:

# Naif backtracking algorithm: recursive form

**Satisfiability problem**:

Is there a truth-assignment that satisfies $F$?

**Strengthened satisfiability problem**:

Given a partial $\alpha$, can $\alpha$ be extended to satisfy $F$?

**Algorithm** $A(\alpha)$:

if $\alpha$ satisfies $F$: return YES

# Naif backtracking algorithm: recursive form

**Satisfiability problem**:

Is there a truth-assignment that satisfies $F$?

**Strengthened satisfiability problem**:

Given a partial $\alpha$, can $\alpha$ be extended to satisfy $F$?

**Algorithm** $A(\alpha)$:

if $\alpha$ satisfies $F$: return YES
if $\alpha$ falsifies $F$: return NO

# Naif backtracking algorithm: recursive form

**Satisfiability problem**:

Is there a truth-assignment that satisfies $F$?

**Strengthened satisfiability problem**:

Given a partial $\alpha$, can $\alpha$ be extended to satisfy $F$?

**Algorithm** $A(\alpha)$:

if $\alpha$ satisfies $F$: return YES
if $\alpha$ falsifies $F$: return NO

choose $x$ in $V \setminus \mathrm{Dom}(\alpha)$ and $a$ in $\{0, 1\}$

# Naif backtracking algorithm: recursive form

**Satisfiability problem**:

Is there a truth-assignment that satisfies $F$?

**Strengthened satisfiability problem**:

Given a partial $\alpha$, can $\alpha$ be extended to satisfy $F$?

**Algorithm** $A(\alpha)$:

if $\alpha$ satisfies $F$: return YES
if $\alpha$ falsifies $F$: return NO

choose $x$ in $V \setminus \mathrm{Dom}(\alpha)$ and $a$ in $\{0, 1\}$
if $A(\alpha \cup \{x = a\})$ gives YES: return YES

# Naif backtracking algorithm: recursive form

**Satisfiability problem**:

Is there a truth-assignment that satisfies $F$?

**Strengthened satisfiability problem**:

Given a partial $\alpha$, can $\alpha$ be extended to satisfy $F$?

**Algorithm** $A(\alpha)$:

if $\alpha$ satisfies $F$: return YES
if $\alpha$ falsifies $F$: return NO

choose $x$ in $V \setminus \mathrm{Dom}(\alpha)$ and $a$ in $\{0, 1\}$
if $A(\alpha \cup \{x = a\})$ gives YES: return YES
if $A(\alpha \cup \{x = 1 - a\})$ gives YES: return YES

# Naif backtracking algorithm: recursive form

**Satisfiability problem**:

Is there a truth-assignment that satisfies $F$?

**Strengthened satisfiability problem**:

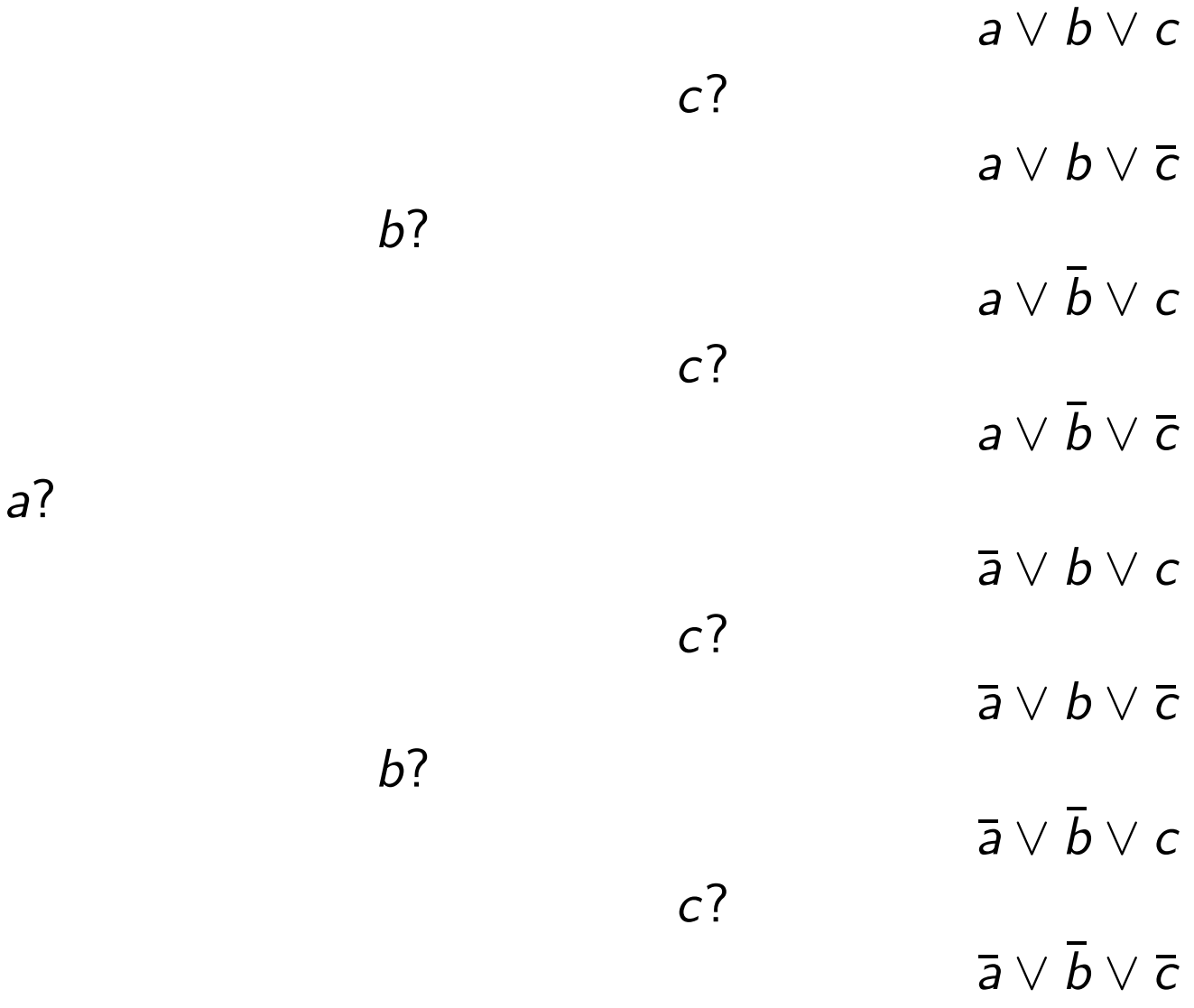
Given a partial $\alpha$, can $\alpha$ be extended to satisfy $F$?

**Algorithm** $A(\alpha)$:

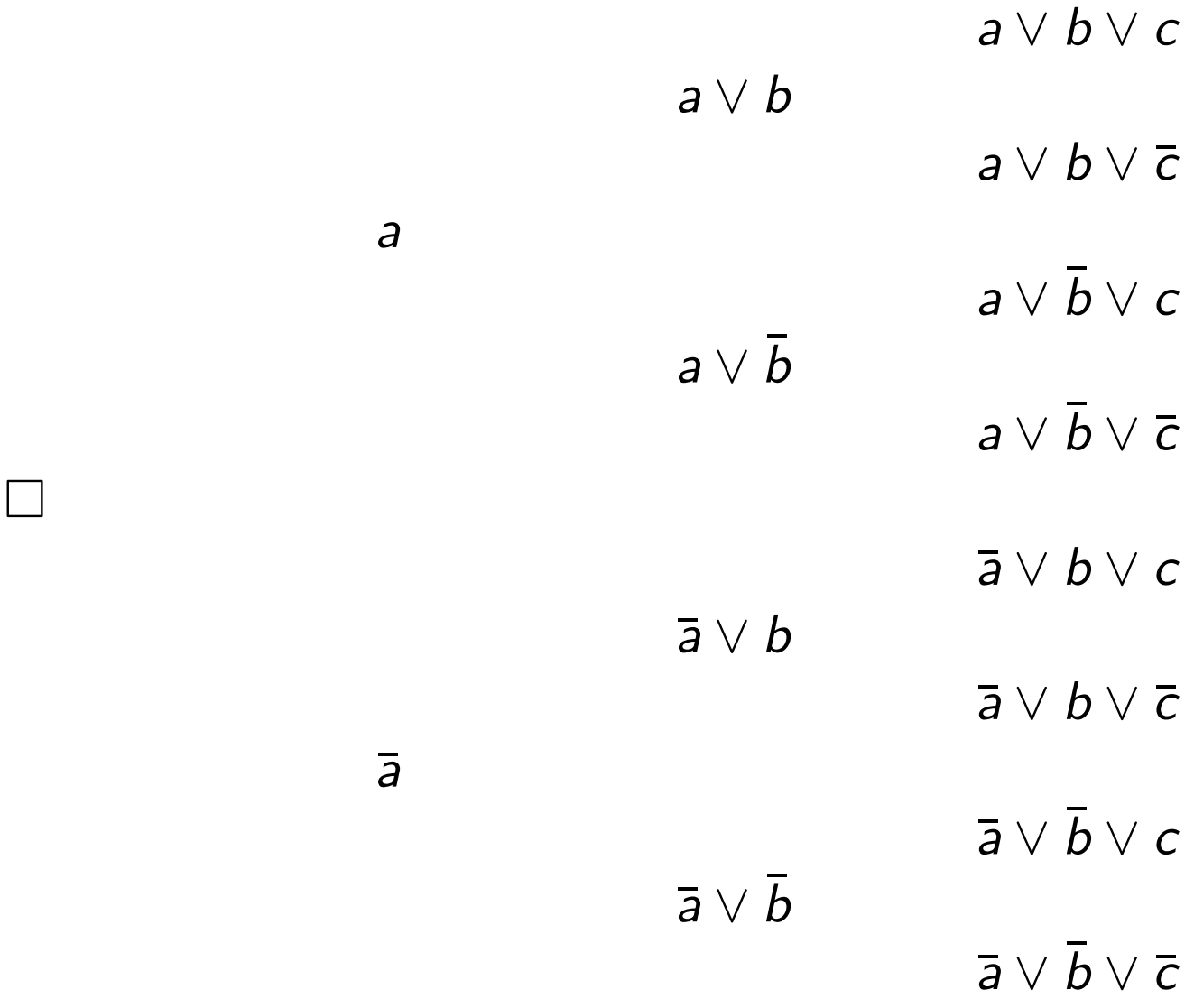
if $\alpha$ satisfies $F$: return YES
if $\alpha$ falsifies $F$: return NO

choose $x$ in $V \setminus \mathrm{Dom}(\alpha)$ and $a$ in $\{0, 1\}$
if $A(\alpha \cup \{x = a\})$ gives YES: return YES
if $A(\alpha \cup \{x = 1 - a\})$ gives YES: return YES
return NO

# Recursion trees vs. resolution proofs



# Recursion trees vs. resolution proofs



# Naif backtracking algorithm: iterative form

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:

if $\alpha$ satisfies $F$: return YES

if $\alpha$ falsifies $F$: go to CONFLICT

go to DECIDE

# Naif backtracking algorithm: iterative form

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:

if $\alpha$ satisfies $F$: return YES

if $\alpha$ falsifies $F$: go to CONFLICT

go to DECIDE

DECIDE:

choose $x$ in $V \setminus \mathrm{Dom}(\alpha)$ and $a$ in $\{0, 1\}$

append $x \stackrel{\mathrm{d}}{=} a$ to $\alpha$, go to DEFAULT

# Naif backtracking algorithm: iterative form

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:

if $\alpha$ satisfies $F$: return YES

if $\alpha$ falsifies $F$: go to CONFLICT

go to DECIDE

DECIDE:

choose $x$ in $V \setminus \mathrm{Dom}(\alpha)$ and $a$ in $\{0, 1\}$

append $x \stackrel{\mathrm{d}}{=} a$ to $\alpha$, go to DEFAULT

CONFLICT:

if $\alpha$ has no $\stackrel{\mathrm{d}}{=}$: return NO

if last $\stackrel{\mathrm{d}}{=}$ in $\alpha$ is $x \stackrel{\mathrm{d}}{=} a$: replace it by $x = 1 - a$

remove all later assignments from $\alpha$, go to DEFAULT

# Add unit-clause propagation

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:
- if $\alpha$ satisfies $F$: return YES
- if $\alpha$ falsifies $F$: go to CONFLICT
- if $F|_\alpha$ contains a unit-clause: go to UNIT
- go to DECIDE

# Add unit-clause propagation

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:
- if $\alpha$ satisfies $F$: return YES
- if $\alpha$ falsifies $F$: go to CONFLICT
- if $F|_\alpha$ contains a unit-clause: go to UNIT
- go to DECIDE

UNIT:
- choose unit-clause $x^a$ in $F|_\alpha$
- append $x = a$ to $\alpha$, go to DEFAULT

# Add clause-learning

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:
- if $\alpha$ satisfies $F$: return YES
- if $\alpha$ falsifies $F$: go to CONFLICT
- if $F|_\alpha$ contains a unit-clause: go to UNIT
- go to DECIDE

# Add clause-learning

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:
- if $\alpha$ satisfies $F$: return YES
- if $\alpha$ falsifies $F$: go to CONFLICT
- if $F|_\alpha$ contains a unit-clause: go to UNIT
- go to DECIDE

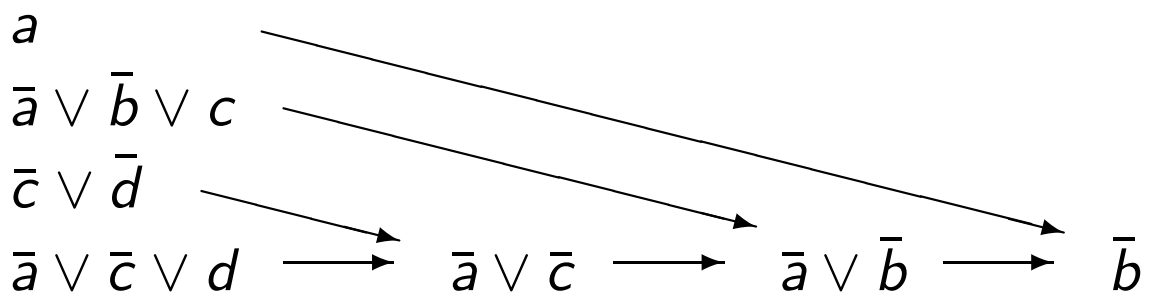
CONFLICT:
- add new $C$ to $F$ with $F \models C$ and $C|_\alpha = \emptyset$
- if $C$ is the empty clause: return NO
- remove assignments from the tail of $\alpha$ while $C|_\alpha = \emptyset$
- go to DEFAULT

# How is the new clause found?

Example:

$$F = a \wedge (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{c} \vee \bar{d}) \wedge (\bar{a} \vee \bar{c} \vee d)$$

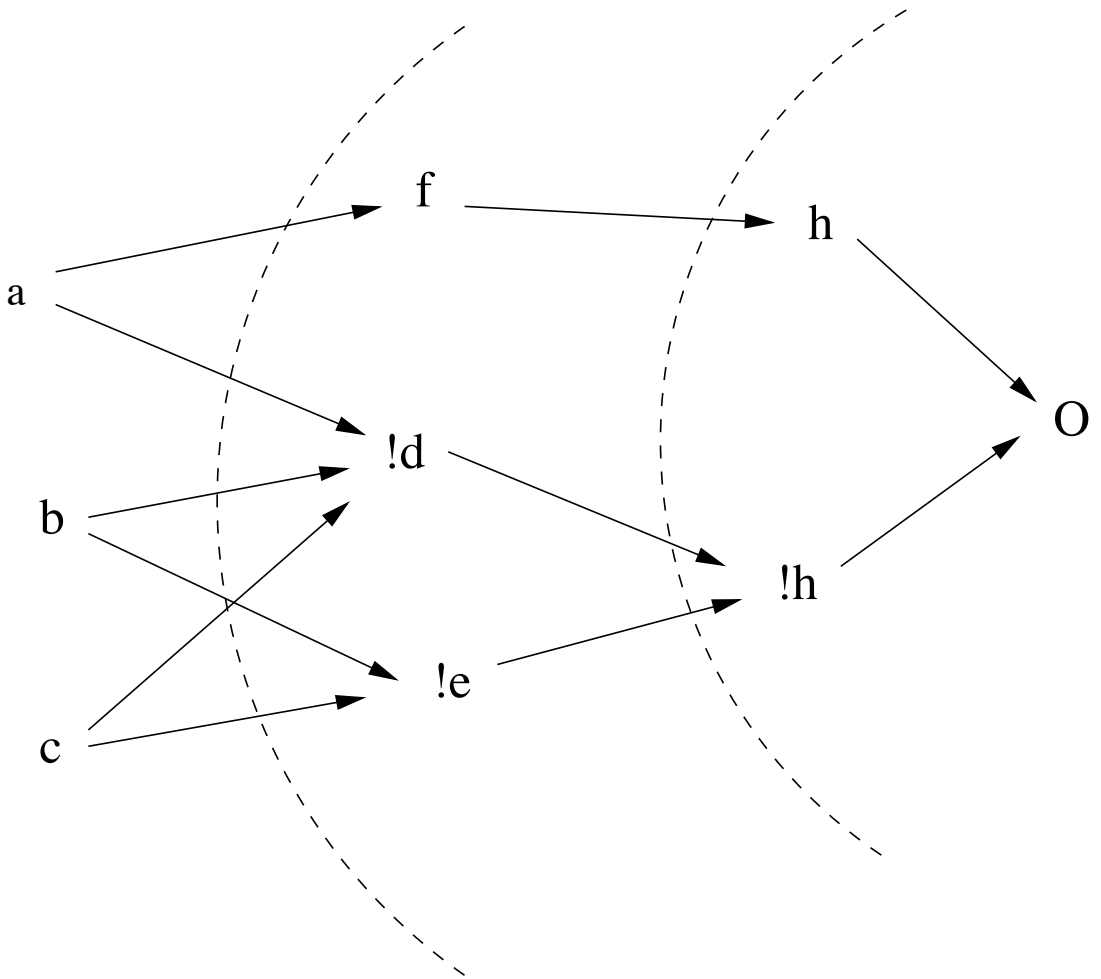
| | | |
|---|---|---|
| UNIT: | $a = 1$ | due to $a$ |
| DECIDE: | $b \overset{\text{d}}{=} 1$ | choice |
| UNIT: | $c = 1$ | due to $\bar{a} \vee \bar{b} \vee c$ |
| UNIT: | $d = 0$ | due to $\bar{c} \vee \bar{d}$ |
| CONFLICT: | | due to $\bar{a} \vee \bar{c} \vee d$. |




Add (or learn) $\bar{b}$.

# How is the new clause found?

**Cuts in a conflict graph**:



# How is the new clause found?

**Tracing the conflict backwards**:

$$S = (x_1 = a_1,\ x_2 \overset{\text{d}}{=} a_2,\ \cdots\ x_r = 1)$$

Initialization:

choose a clause $A \in F$ such that $A|_S = \emptyset$;
set $C_r := A$;
set $S_r := S^{\leq r-1}$.

For $x_i = a_i$ added by UNIT:

choose a clause $A \in F$ such that $A|_{S_{i+1}} = \{x_i^{a_i}\}$;
set $C_i := \mathrm{Res}(C_{i+1}, A, x_i)$;
set $S_i := S^{\leq r-i}$.

For $x_i \overset{\text{d}}{=} a_i$ added by DECISION:

set $C_i := C_{i+1}$;
set $S_i := S^{\leq r-i}$.

# How is the new clause found?

**Learning strategies**:

- DECISION: Add $C_1$.
- 1UIP: Add first $C_i$ with 1 literal of last decision level.
- ASSERTING: Add any $C_i$ with 1 literal of last decision level.

# Add "occasional" restarts

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:
- if $\alpha$ satisfies $F$: return YES
- if $\alpha$ falsifies $F$: go to CONFLICT
- if $F|_\alpha$ contains a unit-clause: go to UNIT
- go to DECIDE

CONFLICT:
- add new $C$ to $F$ with $F \models C$ and $C|_\alpha = \emptyset$
- if $C$ is the empty clause: return NO
- choose whether to restart (with current $F$) or continue
- remove assignments from the tail of $\alpha$ while $C|_\alpha = \emptyset$
- go to DEFAULT

# Add systematic restarts

**Algorithm** $A$:

Let $\alpha$ be the empty list

DEFAULT:
- if $\alpha$ satisfies $F$: return YES
- if $\alpha$ falsifies $F$: go to CONFLICT
- if $F|_\alpha$ contains a unit-clause: go to UNIT
- go to DECIDE

CONFLICT:
- add new $C$ to $F$ with $F \models C$ and $C|_\alpha = \emptyset$
- if $C$ is the empty clause: return NO
- restart (with current $F$)

# Clause-learning algorithm with restarts vs. resolution

Run algorithm $A$ on a CNF-formula $F$ with $n$ variables.

**Fact (M-SS96, BKS04)**

*If there exist choices that make $A$ return NO after $m$ conflicts, then $F$ has a resolution refutation of length $O(mn)$.*

**Theorem (PD09 see also BKS04, HBPvG08, BHJ08, AFT09)**

*If $F$ has a resolution refutation of length $m$, then there exist choices that make $A$ return NO after $O(m)$ conflicts.*

# Part II

# PROPOSITIONAL PROOF COMPLEXITY

# Goals of propositional proof complexity

**Fundamental problem addressed**:

For a given propositional proof system (a non-deterministic algorithm for UNSAT), find explicit hard instances and prove it.

**Original goal**:

Approach to proving NP $\neq$ co-NP (and hence P $\neq$ NP).

**More realistic and current goal**:

Approach to proving that (and understanding why) large families of algorithms do not solve SAT (or UNSAT) in polynomial time.

# Sample result

Let $\neg\mathrm{PHP}_n^{n+1}$ be the CNF-formula on $x_{i,j}$-variables:

$$x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,n} \qquad \text{for } i \in [n+1]$$

$$\overline{x_{i,j}} \vee \overline{x_{i',j}} \qquad \text{for } i, i' \in [n+1], i \neq i' \text{ and } j \in [n].$$

# Sample result

Let $\neg\mathrm{PHP}_n^{n+1}$ be the CNF-formula on $x_{i,j}$-variables:

$$x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,n} \qquad \text{for } i \in [n+1]$$
$$\overline{x_{i,j}} \vee \overline{x_{i',j}} \qquad \text{for } i, i' \in [n+1], i \neq i' \text{ and } j \in [n].$$

**Theorem (Haken 1986)**

*Every resolution refutation of $\neg\mathrm{PHP}_n^{n+1}$ requires $2^{\Omega(n)}$ clauses.*

# Sample result

Let $\neg\mathrm{PHP}_n^{n+1}$ be the CNF-formula on $x_{i,j}$-variables:

$$x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,n} \qquad \text{for } i \in [n+1]$$
$$\overline{x_{i,j}} \vee \overline{x_{i',j}} \qquad \text{for } i, i' \in [n+1], i \neq i' \text{ and } j \in [n].$$

**Theorem (Haken 1986)**

*Every resolution refutation of $\neg\mathrm{PHP}_n^{n+1}$ requires $2^{\Omega(n)}$ clauses.*

**Interpretation**:

Every algorithm for UNSAT that (implicitely or explicitely) generates a resolution proof is exponential on $\neg\mathrm{PHP}_n^{n+1}$.

# Another sample result

**Theorem (BEGJ98, BIW00)**

*There are CNF-formulas with $n$ variables and $n^{O(1)}$ clauses that:*

- *have resolution proofs with $n^{O(1)}$ clauses,*
- *do not have tree-resolution proofs with $2^{\Omega(n/\log(n))}$ clauses.*

# Another sample result

Theorem (BEGJ98, BIW00)

*There are CNF-formulas with $n$ variables and $n^{O(1)}$ clauses that:*

- *have resolution proofs with $n^{O(1)}$ clauses,*
- *do not have tree-resolution proofs with $2^{\Omega(n/\log(n))}$ clauses.*

**Interpretation**:

Learning new clauses for later re-use has the potential of being exponentially faster than naive DPLL. And we undestand why.

# Another sample result

**Theorem (BEGJ98, BIW00)**

*There are CNF-formulas with $n$ variables and $n^{O(1)}$ clauses that:*

- *have resolution proofs with $n^{O(1)}$ clauses,*
- *do not have tree-resolution proofs with $2^{\Omega(n/\log(n))}$ clauses.*

**Interpretation**:

Learning new clauses for later re-use has the potential of being exponentially faster than naive DPLL. And we undestand why.

**Question**:

How do we find which clauses to learn?

# Resolution width

**Width**:

- of a clause: number of literals in it.
- of a CNF-formula: maximum width of the clauses.
- of a resolution proof: maximum width of the clauses.

# Resolution width

**Width**:

- of a clause: number of literals in it.
- of a CNF-formula: maximum width of the clauses.
- of a resolution proof: maximum width of the clauses.

Let $F$ be a CNF-formula with $n$ variables and width $k$.

Theorem (B-SW99)

*If* $F$ *has a resolution refutation of length* $m$,
*then* $F$ *has a resolution refutation of width* $O(\sqrt{n \log m} + k)$.

# Resolution width

**Width**:

- of a clause: number of literals in it.
- of a CNF-formula: maximum width of the clauses.
- of a resolution proof: maximum width of the clauses.

Let $F$ be a CNF-formula with $n$ variables and width $k$.

Theorem (B-SW99)

*If* $F$ *has a resolution refutation of length* $m$,
*then* $F$ *has a resolution refutation of width* $O(\sqrt{n \log m} + k)$.

**Interpretation**:

For $k$-CNF formulas that have polynomial refutations, one can be found in subexponential-time $O(2^{n^{\frac{1}{2}+\epsilon}})$.

# Finding the proof

**Question 1**:

Could it be that for CNF-formulas that have refutations of length $m$, one could always be found in time polynomial in $m$?

# Finding the proof

**Question 1**:

Could it be that for CNF-formulas that have refutations of length $m$, one could always be found in time polynomial in $m$?

Theorem (AR08)

*Answer to Question 1 is no, even for probabilistic algorithms, unless the k-clique problem on n-vertex graphs can be solved by a probabilistic algorithm in time* $n^{o(k)}$.

# Finding the proof

**Question 1**:

Could it be that for CNF-formulas that have refutations of length $m$, one could always be found in time polynomial in $m$?

Theorem (AR08)

*Answer to Question 1 is no, even for probabilistic algorithms, unless the k-clique problem on n-vertex graphs can be solved by a probabilistic algorithm in time* $n^{o(k)}$.

**Interpretation**:

Non-determinism in the SAT-solver result looks unavoidable.

# Finding the proof: continued

Recall: for CNF-formulas that have refutations of width $w$, one can always be found in time $O(n^w)$.

**Question 2**: Could one define a choice strategy for the SAT-solver that, for CNF-formulas that have refutations of width $w$, finds one in time polynomial in $n^w$?

# Finding the proof: continued

Recall: for CNF-formulas that have refutations of width $w$, one can always be found in time $O(n^w)$.

**Question 2**: Could one define a choice strategy for the SAT-solver that, for CNF-formulas that have refutations of width $w$, finds one in time polynomial in $n^w$?

Theorem (AFT09)

*Answer to Question 2 is yes, for a very general probabilistic choice strategy.*

# Part III

# ANALYSIS

# Choice strategy under analysis

**Learning scheme**:

- Any scheme that always adds an asserting clause.
- Natural particular case: DECISION scheme, 1UIP scheme, etc.

**Restart policy**:

- Any policy that allows any controlled number of conflics between restarts.
- Natural particular case: restart at every conflict.

**Decision strategy**:

- Any strategy that allows a controlled number of rounds of arbritrary decisions between rounds of totally random ones.
- Natural particular case: totally random decisions all the time.

# Rounds of the algorithm

A round is a sequence

$$\text{UNIT}^*(,\text{DECIDE},\text{UNIT}^*)^*$$

where each $\text{UNIT}^*$ is maximal.

# Rounds of the algorithm

A round is a sequence

$$\text{UNIT}^*(,\text{DECIDE},\text{UNIT}^*)^*$$

where each $\text{UNIT}^*$ is maximal.

A conclusive round is one where CONFLICT would be next.

# Rounds of the algorithm

A round is a sequence

$$\text{UNIT}^*(,\text{DECIDE},\text{UNIT}^*)^*$$

where each $\text{UNIT}^*$ is maximal.

A conclusive round is one where CONFLICT would be next.
An inconclusive round is one where CONFLICT would not be next.

# Rounds of the algorithm

A round is a sequence

$$\text{UNIT}^*(,\text{DECIDE},\text{UNIT}^*)^*$$

where each $\text{UNIT}^*$ is maximal.

A conclusive round is one where CONFLICT would be next.
An inconclusive round is one where CONFLICT would not be next.
A complete round is a maximal round.

# Clause absorption

Let $F$ be a set of clauses.
Let $C$ be a clause.
Let $R$ be an inconclusive round started with $F$.

# Clause absorption

Let $F$ be a set of clauses.
Let $C$ be a clause.
Let $R$ be an inconclusive round started with $F$.

**Fact**

*If $C$ belongs to $F$ and $R$ falsifies all literals of $C$ but one, then $R$ satisfies the remaining one.*

# Clause absorption

Let $F$ be a set of clauses.
Let $C$ be a clause.
Let $R$ be an inconclusive round started with $F$.

**Fact**

*If $C$ belongs to $F$ and $R$ falsifies all literals of $C$ but one, then $R$ satisfies the remaining one.*

**Definition**

$F$ absorbs $C$ if every inconclusive round that falsifies all literals of $C$ but one, satisfies the remaining one.

# Example and non-example

Let

$$F = (a \vee \bar{b}) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee d \vee e).$$

# Example and non-example

Let

$$F = (a \vee \bar{b}) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee d \vee e).$$

**Note**: $F$ absorbs $a \vee c$. Why?

# Example and non-example

Let

$$F = (a \vee \bar{b}) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee d \vee e).$$

**Note**: $F$ absorbs $a \vee c$. Why?
$a = 0$ implies $b = 0$ and $b = 0$ implies $c = 1$, by UNIT;

# Example and non-example

Let

$$F = (a \vee \bar{b}) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee d \vee e).$$

**Note**: $F$ absorbs $a \vee c$. Why?
$a = 0$ implies $b = 0$ and $b = 0$ implies $c = 1$, by UNIT;
$c = 0$ implies $b = 1$ and $b = 1$ implies $a = 1$, by UNIT.

# Example and non-example

Let

$$F = (a \vee \bar{b}) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee d \vee e).$$

**Note**: $F$ absorbs $a \vee c$. Why?
$a = 0$ implies $b = 0$ and $b = 0$ implies $c = 1$, by UNIT;
$c = 0$ implies $b = 1$ and $b = 1$ implies $a = 1$, by UNIT.

**Note**: $F$ does not absorb $\bar{b} \vee d \vee e$. Why?

# Example and non-example

Let

$$F = (a \vee \bar{b}) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee d \vee e).$$

**Note**: $F$ absorbs $a \vee c$. Why?
$a = 0$ implies $b = 0$ and $b = 0$ implies $c = 1$, by UNIT;
$c = 0$ implies $b = 1$ and $b = 1$ implies $a = 1$, by UNIT.

**Note**: $F$ does not absorb $\bar{b} \vee d \vee e$. Why?
Look at the inconclusive round $e \overset{\mathrm{d}}{=} 0, d \overset{\mathrm{d}}{=} 0$.

# Example and non-example

Let

$$F = (a \vee \bar{b}) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee d \vee e).$$

**Note**: $F$ absorbs $a \vee c$. Why?
$a = 0$ implies $b = 0$ and $b = 0$ implies $c = 1$, by UNIT;
$c = 0$ implies $b = 1$ and $b = 1$ implies $a = 1$, by UNIT.

**Note**: $F$ does not absorb $\bar{b} \vee d \vee e$. Why?
Look at the inconclusive round $e \overset{\mathrm{d}}{=} 0, d \overset{\mathrm{d}}{=} 0$.

**Note**: Both $F \models a \vee c$ and $F \models \bar{b} \vee d \vee e$. Why?

# Example and non-example

Let

$$F = (a \vee \bar{b}) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee d \vee e).$$

**Note**: $F$ absorbs $a \vee c$. Why?
$a = 0$ implies $b = 0$ and $b = 0$ implies $c = 1$, by UNIT;
$c = 0$ implies $b = 1$ and $b = 1$ implies $a = 1$, by UNIT.

**Note**: $F$ does not absorb $\bar{b} \vee d \vee e$. Why?
Look at the inconclusive round $e \overset{\text{d}}{=} 0, d \overset{\text{d}}{=} 0$.

**Note**: Both $F \models a \vee c$ and $F \models \bar{b} \vee d \vee e$. Why?
Resolve 1st and 2nd, and 1st and 3rd, respectively.

# Key properties of absorption

**Logical consequence**:
If $F$ absorbs $C$, then $F \models C$.

# Key properties of absorption

**Logical consequence**:
If $F$ absorbs $C$, then $F \models C$.

**Contradiction**:
If $F$ absorbs $x$ and $\neg x$,
then any round started with $F$ yields a conflict without decisions.

# Key properties of absorption

**Logical consequence**:
If $F$ absorbs $C$, then $F \models C$.

**Contradiction**:
If $F$ absorbs $x$ and $\neg x$,
then any round started with $F$ yields a conflict without decisions.

**Monotonicity**:

- if $C \in F$, then $F$ absorbs $C$,
- if $F \subseteq G$ and $F$ absorbs $C$, then $G$ absorbs $C$,
- if $C \subseteq D$ and $F$ absorbs $C$, then $F$ absorbs $D$.

# Non-absorbed resolvents

Let $F$ be a CNF-formula with $n$ variables.
Let $\frac{A \quad B}{C}$ be a valid resolution inference; $C$ non-empty.

# Non-absorbed resolvents

Let $F$ be a CNF-formula with $n$ variables.
Let $\frac{A \quad B}{C}$ be a valid resolution inference; $C$ non-empty.

## Lemma (for DECISION learning scheme)

*If* $F$ *absorbs* $A$ *and* $B$, *but not* $C$,
*then there exists a round* $R$ *started with* $F$ *such that:*

1. $R$ *is conclusive and learns a clause* $C'$ *with* $C' \subseteq C$,
2. $R$ *makes at most* $|C|$ *decisions.*

# Non-absorbed resolvents

Let $F$ be a CNF-formula with $n$ variables.
Let $\frac{A \quad B}{C}$ be a valid resolution inference; $C$ non-empty.

Lemma (for DECISION learning scheme)

*If $F$ absorbs $A$ and $B$, but not $C$,*
*then there exists a round $R$ started with $F$ such that:*

1. *$R$ is conclusive and learns a clause $C'$ with $C' \subseteq C$,*
2. *$R$ makes at most $|C|$ decisions.*

**Interpretation of 1**:

When $R$ happens, $C$ becomes absorbed.

**Intrepretation of 2**:

$R$ has probability $\Omega\left(\frac{1}{(2n)^{|C|}}\right)$ of happening.

# Bottom-line

**Theorem**

*If* $F$ *has a resolution refutation of width* $w$*,*
*then the algorithm learns the empty clause after* $O(n^{2k})$ *restarts, with probability at least* $0.99$*.*

**Theorem**

*If* $F$ *has a resolution refutation of length* $m$*,*
*then there exist choices to learn the empty clause after* $O(m)$ *restarts.*

# Part IV

# CONCLUDING REMARKS

# Concluding remarks

**Results**:

- Cannot avoid non-determinism to simulate full-resolution.
- Can replace non-determinism by bounded-error randomness to simulate bounded-width resolution.
- Under very mild conditions of learning scheme, restart policy, and decision strategy.

**Questions**:

- Are there similar results for other fragments of resolution?
- Should we stop being shy about restarting?
- Experimental validation of aggressive restart?
- Space (memory) issues?

# Philosophical issue

**Quoting Pipadsrisawat and Darwiche 2010**:

> *Our main theorem shows that the components discussed here are already sufficient to achieve the full power of [resolution]. According to this perspective, any [additional components] can now be viewed as simply heuristics for guiding resolution. This suggests that we should probably construct these components from the resolution point-of-view rather than the search point-of-view, which is usually taken in practice.*