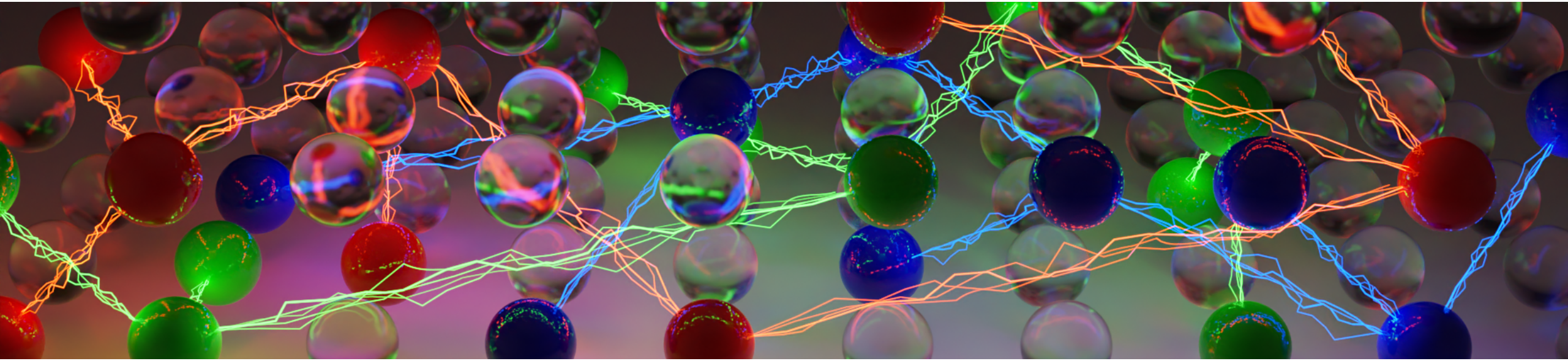
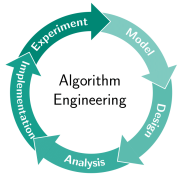


Practical SAT Solving

Lecture 9 – Redundancy Notions and Proofs of Unsatisfiability

Ashlin Iser, Dominik Schreiber | June 22, 2026



SAtRes
Scalable Automated Reasoning

Conflict-driven Clause Learning (CDCL)

Recap: Preprocessing

- Subsumption
- Self-subsuming Resolution
- Bounded Variable Elimination
- Blocked Clause Elimination
- Relationship with Tseitin Encoding
- Scheduling
  - Heuristic Bounds
  - Interleaving of Techniques (f.ex. Round Robin)
  - Inprocessing: Interleave with solving

Algorithm 1: CDCL(CNF Formula F, &Assignment A $\leftarrow \emptyset$)

```
1  if not PREPROCESSING then return UNSAT
2  while A is not complete do
3      UNIT PROPAGATION
4      if A falsifies a clause in F then
5          if decision level is 0 then return UNSAT
6          else
7              (clause, level) ← CONFLICT-ANALYSIS
8              add clause to F and backtrack to level
9              continue
10     if RESTART then backtrack to level 0
11     if CLEANUP then forget some learned clauses
12     if not INPROCESSING then return UNSAT
13     BRANCHING
14 return SAT
```

Propagation-based Redundancy Notions

Let a formula F, and a literal x be given.

Failed Literal Probing

If $F \wedge x \vdash_{UP} \bot$, then $F \models \neg x$

$\implies$ add $\{\neg x\}$ to F

Recap: Meaning of $\vdash$ and $\models$?

Propagation-based Redundancy Notions

Let a formula F, and a literal x be given.

Failed Literal Probing

If $F \wedge x \vdash_{UP} \bot$, then $F \models \neg x$

$\implies$ add $\{\neg x\}$ to F

Example (Failed Literal Probing)

Let $F := \big\{\{a, b\}, \{a, \neg b\}\big\}$.

Probing with $\neg a$ results in a conflict, i.e., $F \wedge \neg a \vdash_{UP} \bot$.

Ergo, we can deduce $F \equiv F \wedge a$.

Propagation-based Redundancy Notions

Let a formula F, a clause $C \in F$, and a literal $x \in C$ be given.

Asymmetric Literal Elimination (ALE)

If $F \setminus C \wedge \overline{C \setminus \{x\}} \vdash_{UP} \overline{x}$, then $F \models C \setminus \{x\}$.

$\Longrightarrow$ strengthen C to $C \setminus \{x\}$

Proof?

Propagation-based Redundancy Notions

Let a formula F, a clause $C \in F$, and a literal $x \in C$ be given.

Asymmetric Literal Elimination (ALE)

If $F \setminus C \wedge \overline{C \setminus \{x\}} \vdash_{UP} \overline{x}$, then $F \models C \setminus \{x\}$.

$\implies$ strengthen C to $C \setminus \{x\}$

Example (Asymmetric Literal Elimination (ALE))

Let $F := \{\{a, b\}, \{\neg b, \neg c\}, \{a, c, d\}\}$, $C := \{a, c, d\}$, and $x := c$.

ALE results in the following propagation: $\{\{a, b\}, \{\neg b, \neg c\}, \{\neg a\}, \{\neg d\}\} \vdash_{UP} \neg c$.

Ergo, we can deduce $F \models \{a, d\}$.

F can not have a model which satisfies $\{a, c, d\}$, but not $\{a, d\}$.

Propagation-based Redundancy Notions

Let a formula F, a clause $C \in F$, and a literal $x \in C$ be given.

Asymmetric Tautology Elimination (ATE)

If $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, then $F \setminus C \models C$.

$\implies$ remove C from F

Does that remind you of something?

Propagation-based Redundancy Notions

Let a formula F, a clause $C \in F$, and a literal $x \in C$ be given.

Asymmetric Tautology Elimination (ATE)

If $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, then $F \setminus C \models C$.

$\implies$ remove C from F

Example (Asymmetric Tautology Elimination (ATE))

Let $F := \{\{a, b, c\}, \{\neg b, d\}, \{a, c, d\}\}$, and $C := \{a, c, d\}$.

ATE results in the following propagation: $\{\{a, b, c\}, \{\neg b, d\}, \{\neg a\}, \{\neg c\}, \{\neg d\}\} \vdash_{UP} \bot$,

Ergo, we can deduce $F \equiv F \setminus \{a, c, d\}$.

$\{a, c, d\}$ follows from the other clauses in F.

Propagation-based Redundancy Notions

Variants: Efficient Algorithms and Implementations

- **Hidden Tautology Elimination (HTE) / Hidden Literal Elimination (HLE)**
  Restricted forms of ATE/ALE which only propagate over binary clauses.

- **Unhiding**: Efficient **HLE** algorithm based on randomized DFS and the parenthesis theorem[1]
  DFS assigns each node *entry/exit timestamps*; transitivity in the BIG is read off in $O(1)$:
  - Hidden literal: if timestamps show $\neg a_1 \rightarrow a_2$ (i.e. $a_1 \vee a_2$ is already implied), remove a_2 from the clause
  - Failed literal: if $a \rightarrow \neg a$ (a is an ancestor of its own negation), then a is globally false, no separate probing needed

- **Distillation**: Sort literals and clauses to simulate a trie, reusing propagations that share a common prefix.
  $\Rightarrow$ Detect ATs / ALs for *multiple clauses* at once.

[1] 2011, Heule et al., Efficient CNF Simplification Based on Binary Implication Graphs

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.
  What can we do in this case?
- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.
- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.

  $\Rightarrow$ **Clause elimination:** remove C.

  Why is that sound?

- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.

- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.
  $\Rightarrow$ **Clause elimination:** remove C.
  $\overline{P} \subseteq \overline{C}$ gives $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, the ATE condition $\Rightarrow F \setminus C \models C$.
- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.
- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.
  $\Rightarrow$ **Clause elimination:** remove C.
  $\overline{P} \subseteq \overline{C}$ gives $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, the ATE condition $\Rightarrow F \setminus C \models C$.
- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.
  What can we do in this case?
- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.
  $\Rightarrow$ **Clause elimination:** remove C.
  $\overline{P} \subseteq \overline{C}$ gives $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, the ATE condition $\Rightarrow F \setminus C \models C$.
- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.
  $\Rightarrow$ **Clause elimination:** remove C.
  Why is that sound?
- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.

  $\Rightarrow$ **Clause elimination:** remove C.

  $\overline{P} \subseteq \overline{C}$ gives $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, the ATE condition $\Rightarrow F \setminus C \models C$.

- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.

  $\Rightarrow$ **Clause elimination:** remove C.

  The extended prefix $\overline{P} \cup \{\overline{a}\}$ leads to a conflict, so C is redundant (the ATE condition, just like above).

- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.

  $\Rightarrow$ **Clause elimination:** remove C.

  $\overline{P} \subseteq \overline{C}$ gives $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, the ATE condition $\Rightarrow F \setminus C \models C$.

- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.

  $\Rightarrow$ **Clause elimination:** remove C.

  The extended prefix $\overline{P} \cup \{\overline{a}\}$ leads to a conflict, so C is redundant (the ATE condition, just like above).

- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.

  What can we do in this case?

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.
  $\Rightarrow$ **Clause elimination:** remove C.
  $\overline{P} \subseteq \overline{C}$ gives $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, the ATE condition $\Rightarrow F \setminus C \models C$.
- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.
  $\Rightarrow$ **Clause elimination:** remove C.
  The extended prefix $\overline{P} \cup \{\overline{a}\}$ leads to a conflict, so C is redundant (the ATE condition, just like above).
- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.
  $\Rightarrow$ **Literal elimination:** remove a from C and continue.
  Why is that sound?

Vivification

Given clause $C = \{a_1, a_2, \ldots, a_k\}$. Sequentially assign $\overline{a}_1, \overline{a}_2, \ldots$ and propagate over $F \setminus \{C\}$.

What can happen after assigning and propagating $\overline{P}$ with $P := \{a_1, \ldots, a_{i \leq k}\}$?

- Conflict ($\bot$): the negated prefix $\overline{P}$ is contradictory w.r.t. $F \setminus \{C\}$.
  $\Rightarrow$ **Clause elimination:** remove C.
  $\overline{P} \subseteq \overline{C}$ gives $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, the ATE condition $\Rightarrow F \setminus C \models C$.
- for some $a \in C \setminus P$, a is propagated true: $F \setminus C \wedge \overline{P} \vdash_{UP} a$.
  $\Rightarrow$ **Clause elimination:** remove C.
  The extended prefix $\overline{P} \cup \{\overline{a}\}$ leads to a conflict, so C is redundant (the ATE condition, just like above).
- for some $a \in C \setminus P$, a is propagated false: a can never help satisfy C in this context.
  $\Rightarrow$ **Literal elimination:** remove a from C and continue.
  $\overline{P} \subseteq \overline{C \setminus \{a\}}$ gives $F \setminus C \wedge \overline{C \setminus \{a\}} \vdash_{UP} \overline{a}$, the ALE condition $\Rightarrow F \setminus C \models C \setminus \{a\}$.

Recap.

Propagation-based Preprocessing

- Propagation-based Redundancy Notions:

  Failed Literal Probing, Asymmetric Literal Elimination, Asymmetric Tautology Elimination
- Efficient Implementation of Propagation-based Redundancy Removal with Binary Implication Graphs (BIGs)
- Vivification and Distillation: Interleaving Propagation with Clause Strengthening and Elimination

Next Up

Proofs of Unsatisfiability

Relationship with Proof Checking

Generalizations of Blocked Clauses

- **Reverse Unit Propagation (RUP)**
  A clause has the property RUP if and only if it is an Asymmetric Tautology (AT).

  CDCL's learned clauses are RUP at the moment of their learning.

- **Resolution Asymetric Tautologies (RATs)**
  A clause C is a RAT in a formula F if it contains a literal x such that
  each resolvent in $C \otimes_x F_{\overline{x}}$ is an asymmetric tautology.

  Blocked Sets in particular are RATs.

Proof Checking

SAT Solvers are complex software systems, and bugs are not uncommon.

Trustworthiness of SAT Solvers

- For satisfiable instances, SAT solvers can output the found assignment
- For unsatisfiable instances, SAT solvers can output a proof of unsatisfiability
- Both can be checked independently from the solver by much simpler, and **formally verified program**.

Example (Applications of Proof Checking)

Unsatisfiability of a formula might proof important properties of the problem at hand, such as:

- Absence of certain bugs in a hardware design or software verification,
- Optimality of a certain makespan in planning
- etc. pp.

Feasibility of Proof Checking

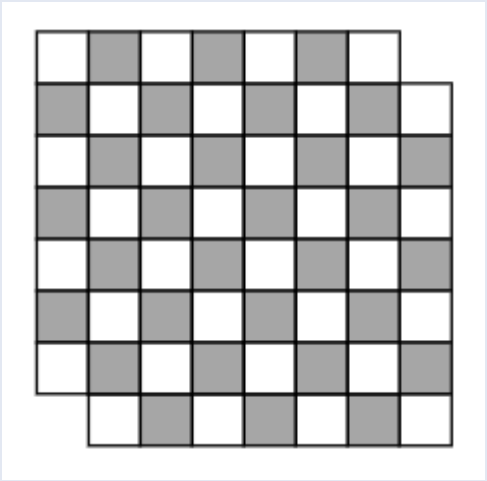
RAT proof checking is polynomial in the proof-size, but proof-size is worst-case exponential in the formula-size.

Pragmatics: if we could generate it, we can also check it.

Proof Systems: Motivating Example

Example (Mutilated Chessboards)

Let a chessboard be given with two diagonally opposite corners removed.
Is it possible to cover the remaining board with dominoes?

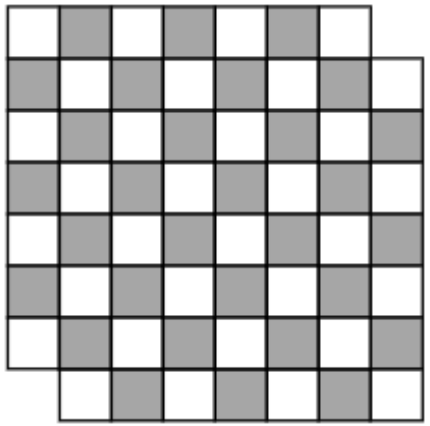


Proof Systems: Motivating Example

Example (Mutilated Chessboards)

Let a chessboard be given with two diagonally opposite corners removed.
Is it possible to cover the remaining board with dominoes?

Human: No, because each dominoe covers exactly one black and one white field, and there are two more black fields than white fields.

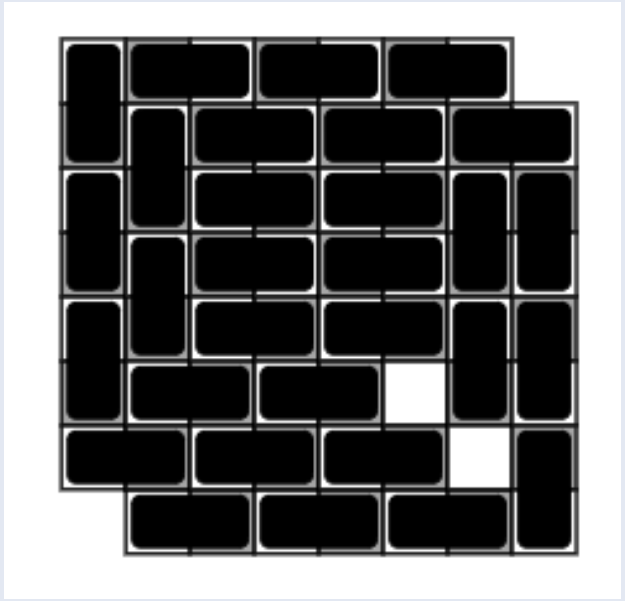


Proof Systems: Motivating Example

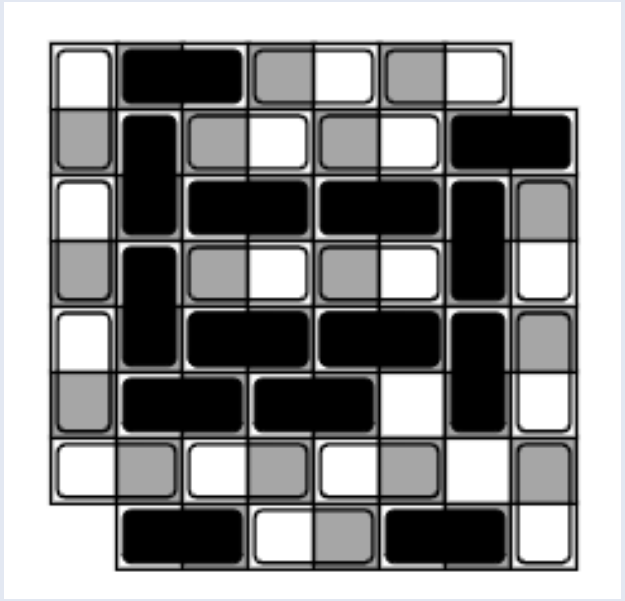
Example (Mutilated Chessboards)

Let a chessboard be given with two diagonally opposite corners removed.
Is it possible to cover the remaining board with dominoes?

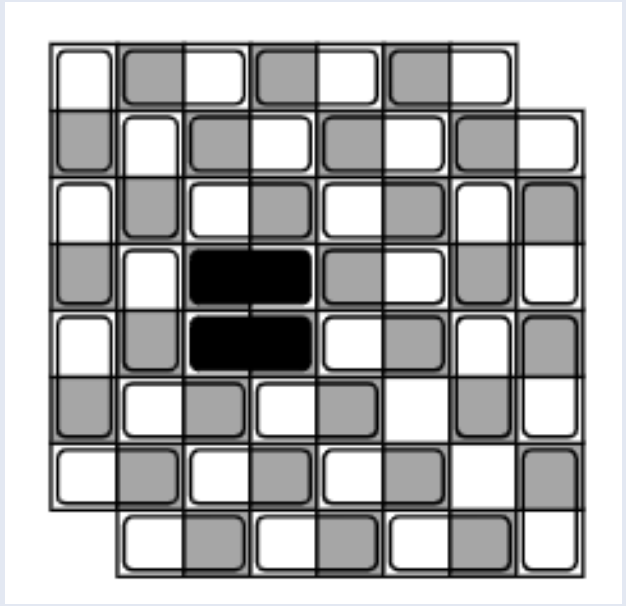
SAT Solver: Let's try and learn.



Impasse Detected



→ Resolution



→ More powerful Proof-system[2]

[2]Example taken from SAT lecture of Marijn J.H. Heule

Proof Complexity

Proof complexity is the study of the size of proofs in different proof systems.

Relationship with SAT Solving

- Static analysis without algorithmic considerations
- Questions of automizability not addressed
- Lower bounds on proof size tell us how good a SAT solver can be in the best case
- Upper bounds on proof size tell us how good a SAT solver should be

In the following: three proof systems of increasing power or generality.

1. Resolution $\rightarrow$ 2. Extended Resolution $\rightarrow$ 3. Blocked Clauses

Proof Systems – 1. Resolution

Resolution

- Preserves equivalence
- CDCL is as powerful as General Resolution
- Well-known Exponential Lower Bounds:

  For example, proofs of unsatisfiability of Pigeon Hole formulas
  are necessarily of exponential length in the resolution proof system (Haken 1985)
- Heuristic for Finding Resolution Proofs: CDCL

Proof Systems – 2. Extended Resolution

Extended Resolution (Tseitin 1966)

- Preserves satisfiability
- Extension rule:
  incorporate $x \leftrightarrow a \wedge b$ for some a, b in the formula and a new variable x
- Strictly more powerful than Resolution
- No Lower Bounds known
- (Cook 1967) polynomial sized ER proof for PH formulas

  $\Rightarrow$ exponentially shorter than any resolution proof

Proof Systems – 3. Blocked Clauses

Blocked Clauses (Kullmann 1999)

- Preserves satisfiability
- Generalization of Extended Resolution
- Allows the addition and removal of blocked clauses

  $\Rightarrow$ no need to introduce a defining equivalence explicitly
- No Lower Bounds known

Stronger Proof Systems without New Variables

Extended Resolution and Blocked Clauses gain power by introducing new variables.

Can we obtain equally short proofs *without* new variables?

Key relaxation: redundancy instead of implication

A clause C is redundant w.r.t. F if F and $F \wedge C$ are *equisatisfiable*.

$\Rightarrow$ C need **not** be implied by F; it suffices that adding it preserves satisfiability.

Example (Redundant but *not* implied)

$F := \{(a \vee b),\ (\neg a \vee \neg b)\}$ has exactly the models $\{\neg a, b\}$ and $\{a, \neg b\}$.

Adding $C := (a)$ keeps F satisfiable (model $\{a, \neg b\}$), yet $F \not\models C$
since the model $\{\neg a, b\}$ falsifies C.

Background: such systems can polynomially simulate Extended Resolution, e.g. short proofs of pigeon hole / mutilated chessboard without any new variables.

Certifying Redundancy: the Witnessing Assignment

How can we *certify* that adding C preserves satisfiability?

Redundancy via a witness

Let $\alpha := \overline{C}$ be the assignment blocked by C (it falsifies every literal of C).

C is redundant w.r.t. $F \iff$ there is a witness ω with $\omega \models C$ and $F|_\alpha \models F|_\omega$.

Notation: $F|_\beta$ removes from F all clauses satisfied by β and all literals falsified by β.

Intuition: every model of F that falsifies C can be repaired into a model of $F \wedge C$ by patching it with ω.

Example (Witness for $C = (a)$ in $F = \{(a \vee b), (\neg a \vee \neg b)\}$)

$\alpha = \{\neg a\}$, witness $\omega = \{a, \neg b\}$ (which satisfies C).

$F|_\alpha = \{(b)\}$ and $F|_\omega = \emptyset$ ($\top$), so $F|_\alpha \models F|_\omega$ holds.

Redundancy Theorem: Proof

Let F be a formula, C a non-empty clause, and $\alpha = \overline{C}$ the assignment blocked by C.
C is redundant with respect to F *if and only if* there exists an assignment ω such that ω satisfies C and $F|_\alpha \models F|_\omega$.

Proof "only if"

Assume F and $F \wedge C$ are equisatisfiable. Show that there exists an ω satisfying C and $F|_\alpha \models F|_\omega$.

If $F|_\alpha$ is unsatisfiable, then the semantic implication trivially holds.

Assume now that $F|_\alpha$ is satisfiable, implying that F is satisfiable.

Since F and $F \wedge C$ are equisatisfiable, there exists an assignment ω that satisfies both F and C.

Since ω satisfies F, it holds that $F|_\omega = \emptyset$ and so trivially $F|_\alpha \models F|_\omega$.

Redundancy Theorem: Proof

Let F be a formula, C a non-empty clause, and $\alpha = \overline{C}$ the assignment blocked by C.
C is redundant with respect to F *if and only if* there exists an assignment ω such that ω satisfies C and $F|_\alpha \models F|_\omega$.

Proof "if"

Assume there exists an assignment ω satisfying C and $F|_\alpha \models F|_\omega$. Show that F and $F \wedge C$ are equisatisfiable.

Let γ be a (total) assignment that satisfies F and falsifies C.
Since γ satisfies F, it must satisfy $F|_\alpha$ and since $F|_\alpha \models F|_\omega$, it must also satisfy $F|_\omega$.

Now, we can turn γ into a satisfying assignment γ' for $F \wedge C$ as follows:

$$\gamma'(x) = \begin{cases} \omega(x) & \text{if } x \in \text{vars}(\omega) \\ \gamma(x) & \text{otherwise} \end{cases}$$

Since ω satisfies C, γ' satisfies C.

Since γ satisfies $F|_\omega$ – and $\text{vars}(F|_\omega) \subseteq \text{vars}(\gamma) \setminus \text{vars}(\omega)$ – so γ' satisfies F.

$\Rightarrow \gamma'$ satisfies $F \wedge C$.

In Practice: Propagation Redundancy (PR)

Deciding $F|_\alpha \models F|_\omega$ is coNP-hard: it asks whether *every* assignment satisfying $F|_\alpha$ also satisfies $F|_\omega$, i.e. whether $F|_\alpha \wedge \neg F|_\omega$ is unsatisfiable.

Propagation Redundant (PR) clause

C is PR w.r.t. F with witness ω if $\omega \models C$ and $F|_\alpha \vdash_{UP} F|_\omega$.

Replace $\models$ by unit propagation $\vdash_{UP}$.

PR clauses need a witness ω. But where do we *find* one cheaply?
Up next: autarkies, a class of assignments that come with their witness for free.

Autarkies

Autarky

Let a formula F and a partial assignment A be given. A clause $C \in F$ is ...

- ... touched by A if it contains a variable assigned in A
- ... satisfied by A if it contains a literal assigned to *True* by A

An autarky is a partial assignment A such that all touched clauses are satisfied.

Let α be an autarky of F. Then, F and $F|_\alpha$ are equisatisfiable. All clauses touched by an autarky can be removed.

Edge Cases: Pure Literals and Satisfying Assignments are Autarkies
Application in Inprocessing: Kissat analyses assignments found by sprints of local search to find autarkies.

Example

The partial assignment $A = \{\neg a, \neg c\}$ is an autarky for $F := \big\{\{\neg a, b\}, \{b, \neg c\}, \{a, \neg b, \neg c\}\big\}$, with witness $\omega = A = \{\neg a, \neg c\}$.

Conditional Autarkies

A plain autarky must satisfy *all* touched clauses.
Relax it conditionally: require the autarky property only *after* fixing some literals α_{con}.

Conditional Autarkies

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

Then F and $F \wedge (\alpha_{con} \rightarrow \alpha_{aut})$ are equisatisfiable.

Example (Pruning Branches with a Conditional Autarky)

Let $F := \{\{x, y\}, \{x, \neg y\}, \{\neg y, \neg z\}\}$, and let $\alpha_{con} = \{x\}$ and $\alpha_{aut} = \{\neg y\}$.
Then $F|_{\alpha_{con}} = \{\{\neg y, \neg z\}\}$ and $\alpha_{aut} = \{\neg y\}$ is an autarky of $F|_{\alpha_{con}}$, such that $\{x\} \cup \{\neg y\}$ is a conditional autarky of F.
We can thus learn the clause $\{\neg x, \neg y\}$, with witness $\omega = \{x, \neg y\}$.

A conditional autarky *is* the redundancy witness ω for the pruning clause. Next we search for it automatically while solving.

Satisfaction Driven Clause Learning (SDCL)

Idea: Also learn clauses if no conflict is detected, but a positive reduct is satisfiable.

Positive Reduct

Let a formula F and a partial, non-conflicting assignment α be given.

The positive reduct $p(F, \alpha)$ is the formula that contains all clauses of F satisfied by α and a clause $C := \neg\alpha$.

A satisfying assignment ω of the positive reduct $p(F, \alpha)$ is a conditional autarky of F, i.e. exactly the redundancy witness ω for the pruning clause $\neg\alpha$.

$\Rightarrow$ if the positive reduct is satisfiable, then the branch α can be pruned.

Key Idea of SDCL:

While solving a formula, check the positive reducts of current assignments α for satisfiability.

If $p(F, \alpha)$ is satisfiable, prune the branch α.

Positive reducts are typically very easy to solve.

From Modern to “Post-modern” SAT Solving

Problem: Automizability, how to find such short proofs?

Solving the Problem of Automizability: Practical Approaches

- **Resolution**: Clause Learning

  → *any classic CDCL implementation*

- **Extended Resolution**: Structural Boundend Variable Addiction (SBVA).

  → `SBVA-CaDiCaL`: Winner of SAT Competition 2023

- **PReLearning**: Preprocessing adds specific Propagation Redundant (PR) clauses

  → `KissatMAB-Prop`: Winner of SAT Competition 2023 on UNSAT instances

- **Symmetry Breaking Predicates**: Exclusion of Symmetric Solutions

  → `BreakId-Kissat`: Special Price at SAT Competition 2023

Recap.

Recap.

- Propagation-based Redundancy Notions
- Proof Systems: Resolution, Extended Resolution, Blocked Clauses, Implication-based Redundancy
- Autarkies, Conditional Autarkies, and Satisfaction Driven Clause Learning (SDCL)

References: Redundancy and Propagation Redundancy

- Heule, Kiesl, Biere (2019), *Strong Extension-Free Proof Systems*
  Introduces redundancy via a witness and the redundancy theorem; defines the SPR/PR family of proof systems without new variables.
- Heule, Kiesl, Biere (2017), *Short Proofs Without New Variables*
  Defines propagation-redundant (PR) clauses and shows they admit short proofs (e.g. pigeon hole) without introducing new variables.
- Tan, Heule, Myreen (TACAS 21), `cake_lpr`
  A formally verified PR proof checker, giving machine-checked trust in PR-based unsatisfiability proofs.
- Heule et al. (2017), *PRuning Through Satisfaction*
  Introduces satisfaction-driven clause learning (SDCL) via positive reducts and conditional autarkies.
- Monien, Speckenmeyer (1985), *Solving satisfiability in less than 2^n steps*
  Origin of autark assignments, the basis for autarky-based simplification.