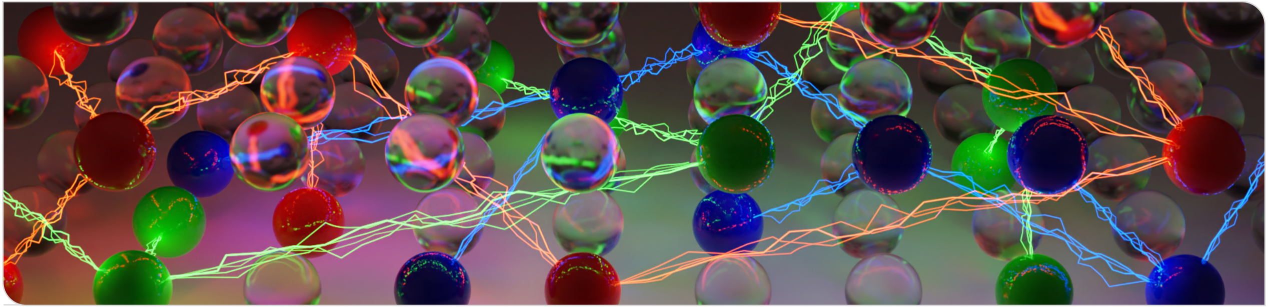
Practical SAT Solving

Lecture 7

Markus Iser, Dominik Schreiber, Tomáš Balyo | June 3, 2024



Recap

Lecture 6: Modern SAT Solving 2

- Efficient Unit Propagation
- Clause Forgetting
- Modern Decision Heuristics: VSIDS & Co.

Today

Preprocessing

Conflict-driven Clause Learning (CDCL) Algorithm

Last Time

- Efficient Unit Propagation
- Clause Forgetting
- Modern Decision Heuristics

Today

- Preprocessing

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     BRANCHING
13 return SAT
```

Preprocessing

Preprocessing takes place between problem encoding and its solution.

Preprocessing is . . .

- a form of reencoding a problem: to fix bad encodings
- a form of reasoning itself: inprocessing

Conjecture: Smaller problems are easier to solve $\Longrightarrow$ Try to reduce the size of the formula.

Classic Preprocessing Techniques

- Subsumption
- Self-subsuming Resolution
- (Bounded) Variable Elimination (BVE)

Preprocessing: Subsumption

A clause C is subsumed by D iff $D \subseteq C$.

Subsumed clauses can be removed from the formula without changing satisfiability: $\forall D \subseteq C, D \models C$

Example

$\{a, b\}$ subsumes $\{a, b, c\}$ and $\{a, b, d\}$

Preprocessing: Subsumption

A clause C is subsumed by D iff $D \subseteq C$.

Subsumed clauses can be removed from the formula without changing satisfiability: $\forall D \subseteq C, D \models C$

Example

$\{a, b\}$ subsumes $\{a, b, c\}$ and $\{a, b, d\}$

Implementation 1: Forward Subsumption

Select clause C and check if it is subsumed by any other clause $D \subseteq C$.

- Temporarily mark all literals in C as unsatisfied, and use one-watched literal data-structure to find subsumed clauses
- **Optimization 1:** Watch literals with the fewest occurrences
- **Optimization 2:** Keep literals sorted and perform merge-sort style subset check

Preprocessing: Subsumption

Implementation 2: Backward Subsumption

Select clause D and check if it subsumes any other clause $C \supseteq D$.

Learned clauses are never subsumed but can subsume other clauses, e.g., recently learned clauses

- **Optimization 1:** Check the clauses of the variable with the fewest occurrences (scales to large formulas)
- **Optimization 2:** Use signatures to skip the majority of subsumption checks (cf. Bloom filters)

Algorithm 2: Signature-based Subsumption Check

```
// Initialization:
for $clause \in formula$ do
    clause.signature = 0
    for $lit \in *clause$ do
        clause.signature |= 1ull $<< (id(lit) \% 64)$
// Subsumption Check:
if $D.signature \; \& \; invert(C.signature) == 0$ then
    // Check if $D$ subsumes $C$
```

Preprocessing: Self-Subsuming Resolution

Applicable if the resolvent of C and another clause D subsumes C.

If $C \otimes_x D \subseteq C$ then C can be replaced by $C \otimes_x D$.

Example

Let $\otimes_f$ be the resolution operator on variable f.

$C := \{\neg b, \neg e, f, \neg h\}$ $D := \{\neg b, \neg e, \neg f\}$ $E := C \otimes_f D = \{\neg b, \neg e, \neg h\}$

$\longrightarrow$ Replace C by E (“clause strengthening”)

Implementation

- **Integrate with subsumption:** Allow at most one literal of D to occur negated in C
- **Variant:** On-the-fly subsumption/strengthening of reason clauses during conflict analysis

Preprocessing: Bounded Variable Elimination

Let $S_x, S_{\overline{x}} \subseteq F$ be the sets of all clauses containing x resp. $\overline{x}$, and let $R = \{C \otimes_x D \mid C \in S_x, D \in S_{\overline{x}}\}$ be the set of all resolvents on x. The formulas F and $F' := (F \setminus (S_x \cup S_{\overline{x}})) \cup R$ are equisatisfiable but not equivalent.

Most important preprocessing technique in practical SAT solving

Bounded Variable Elimination (BVE)

Eliminate variable only if the formula size does not increase (too much).

- **Note 1:** Variables of removed clauses have to be rescheduled for further elimination attempts
- **Note 2:** Resolvent can trigger further subsumptions and vice versa
- Particularly effective in presence of functional definitions (cf. Tseitin encoding)
- **Variant:** Incrementally Relaxed BVE: Increase bound each round if formula size did not increase too much
- **Optimizations:** Perform check only for bounded clause size, resolvent size, or variable occurrence count

Preprocessing: Blocked Clause Elimination (BCE)

A clause $\{x\} \cup C$ is blocked in F by x if either x is pure in F or
for every clause $\{\neg x\} \cup D$ in F the resolvent $C \cup D$ is a tautology.

$\rightarrow$ Dead ends in the resolution graph, no proof beyond this point.

Blocked clause elimination (BCE) has a unique fixpoint, and preserves satisfiability.

Example

$F := (a \vee b) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee c)$

First clause is not blocked, second is blocked by both a and $\neg c$, third is blocked by c.

- Effectiveness of BVE can be increased by interleaving it with BCE
- Relationship with circuit-level simplification techniques
- **Generalization: Covered Clauses**
  A clause is covered if it can be turned into a blocked clause by adding a covered literal. A literal x is covered by a clause C, if it contains a literal y such that all non-tautological resolvents of C on y contain x.

Preprocessing: Solution Reconstruction

Many preprocessing techniques remove clauses or variables from a formula in a mere satisfiability-preserving way, such that the solution to the preprocessed formula might not be a solution to the original formula.

Reconstruction Algorithm

Keep track of eliminated variables (BVE) and clauses (BCE) in a solution reconstruction stack S, and if a model is found, use it to reconstruct a solution to the original formula.

Algorithm 3: Solution Reconstruction

```
Data: Assignment $A$, Stack $S$
1 while $S$ is not empty do
2     remove the last literal-clause pair $(l, C)$ from $S$;
3     if $C$ is not satisfied by $A$ then
4         $A := (A \setminus \{l = 0\}) \cup \{l = 1\}$
5 If variables remain unassigned in $A$, then assign them an
  arbitrary value.
```

Recap.

Preprocessing: Classic Techniques

- Subsumption and Self-subsuming Resolution
- Bounded Variable Elimination
- Blocked Clause Elimination
- Solution Reconstruction

Next Up

Relationship between preprocessing techniques and gate encodings

Preprocessing: Relationship with Gate Encodings

Tseitin encoding E of a gate with output o, function g, and input literals $x_1, \dots, x_n$:

$$E \equiv o \leftrightarrow g(x_1, \dots, x_n)$$

Properties of Gate Encodings

Let a Tseitin encoding $E \equiv o \leftrightarrow g(x_1, \dots, x_n)$ be given, and let $A(X) := \{T \cup \{\overline{x} \mid x \in X \setminus T\} \mid T \in 2^X\}$ denote the set of all assignments to variables in X.

For each input assignment $I \in A(\{x_1, \dots, x_n\})$,

a) there exists at least one output assignment $O \in \{o, \overline{o}\}$ such that $I \cup O \models E$ (left-totality)

b) there exists at most one output assignment $O \in \{o, \overline{o}\}$ such that $I \cup O \models E$ (right-uniqueness)

$\rightarrow$ The output is uniquely determined by the input, such that either $I, o \models E$ and $I, \overline{o} \not\models E$ or vice versa.

Preprocessing: Relationship with Gate Encodings

From the left-totality it follows that a Tseitin encoding E is a satisfiable set of blocked clauses.

Left-Totality of Gate Encodings

Let a Tseitin encoding $E \equiv o \leftrightarrow g(x_1, \ldots, x_n)$ be given, it holds that

a) for each clause $C \in E$, either $o \in C$ or $\overline{o} \in C$

Proof: The existence of a clause $C \in E$ such that $o \notin \text{vars}(C)$ would contradict left-totality, because the assignment falsifiying C, falsifies E for any assignment to o.

b) and all resolvents $R \in E_o \otimes_o E_{\overline{o}}$ are tautological.

Proof: The existence of a non-tautological resolvent $R \in E_o \otimes_o E_{\overline{o}}$ would contradict left-totality, because $E \models R$ and $o \notin \text{vars}(R)$, such that the assignment falsifying R, falsifies E for any assignment to o.

Preprocessing: Relationship with Gate Encodings

From the left-totality it follows that a Tseitin encoding E is a satisfiable set of blocked clauses.

Example (Tseitin encoding $E \equiv o \leftrightarrow x \land y$)

Let a Tseitin encoding $E := \big\{\{\neg o, x\}, \{\neg o, y\}, \{o, \neg x, \neg y\}\big\} \equiv o \leftrightarrow x \land y$ be given, it holds that

a) all resolvents in $E_o \otimes_o E_{\overline{o}} = \big\{\{x, \neg x, \neg y\}, \{y, \neg x, \neg y\}\big\} \equiv \top$ are tautological,

b) and Blocked Clause Elimination (BCE) would remove all clauses from E.

Questions:

- What does BCE do to $F = \big\{\{o\}\big\} \cup E$?
- What does BCE do to $F = \big\{\{\neg o\}\big\} \cup E$?
- What does BCE do to $F = \big\{\{q\}, \{\neg q, o, p\}, \{\neg q, \neg o, \neg p\}\big\} \cup E$?

Preprocessing: Relationship with Gate Encodings

Resolving the clauses of a gate encoding on the output literal o results in a set of tautological clauses.

Idea: Optimized Variable Elimination for Gate Encodings E

Let a formula $F = E \cup R$ with gate clauses E and remainder R be given.
Apply variable elimination as follows:

$$\begin{aligned}
(E_x \cup R_x) \otimes (E_{\overline{x}} \cup R_{\overline{x}}) &\equiv (E_x \otimes R_{\overline{x}}) \cup (R_x \otimes E_{\overline{x}}) \cup (R_x \otimes R_{\overline{x}}) \cup (E_x \otimes E_{\overline{x}}) \\
&\equiv (E_x \otimes R_{\overline{x}}) \cup (R_x \otimes E_{\overline{x}}) \cup (R_x \otimes R_{\overline{x}}) && (E_x \otimes E_{\overline{x}} \equiv \top) \\
&\equiv (E_x \otimes R_{\overline{x}}) \cup (R_x \otimes E_{\overline{x}}) && ((E_x \otimes R_{\overline{x}}) \cup (R_x \otimes E_{\overline{x}}) \models R_x \otimes R_{\overline{x}})
\end{aligned}$$

Proof Idea: Each clause $c \in R_x \otimes R_{\overline{x}}$, derived by resolving $c_x \in R_x$ and $c_{\overline{x}} \in R_{\overline{x}}$, can also be derived by resolving clauses in $R_{\overline{x}} \otimes E_x$ and $E_{\overline{x}} \otimes R_x$.

Preprocessing: Unit Propagation, Probing

Propagation-based Redundancy

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

- **Failed Literal Probing**
  If $F \wedge x \vdash_{UP} \bot$, then $F \models \neg x$.
  $\Longrightarrow$ add $\{\neg x\}$ to F
- **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 $C \setminus \{x\}$
- **Asymmetric Tautology Elimination (ATE)**
  If $F \setminus C \wedge \overline{C} \vdash_{UP} \bot$, then $F \models C$.
  $\Longrightarrow$ remove C from F

Preprocessing: Unit Propagation, Probing

Optimizations of Probing Techniques

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

- **Avoidance of Redundant Propagations**
  Sort literals and clauses in a formula to simulate a trie, and reuse propagations that share the same prefix. the binary implication graph and application of the *parenthesis theorem*.

- **Variants: Distillation / Vivification**
  Interleave assignments and propagations to detect ATs / ALs early.

Probing: 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). All learned clauses are RUP at the moment of their learning. RUP checking is basic proof checking.

- **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. Modern proof systems are based on RATs.

Recap.

Recap.

- Classic Preprocessing Techniques: Subsumption, Self-subsuming Resolution, Bounded Variable Elimination, Blocked Clause Elimination
- Relationship between Preprocessing Techniques and Gate Encodings
- Probing Techniques: Failed Literal Probing, Asymmetric Literal Elimination, Asymmetric Tautology Elimination
- Relationship between Probing Techniques and Proof Checking

Next Up

Scheduling of Preprocessing Techniques, Inprocessing, Autarky Reasoning

Preprocessing: Scheduling of Preprocessing Techniques

At a point where one technique is unable to make further progress, another technique might be applicable and even modify the problem in a way that the first technique can make further progress.

Scheduling of Preprocessing Techniques

- **Heuristic Limits**
  Bound the number of applications of a technique.
- **Scheduling of Techniques**
  Non-trivial, benefit of techniques depends on the formula.
- **Interleaving of Techniques**
  Apply techniques in a round-robin fashion.
- **Inprocessing**
  Interleave search and preprocessing.

Inprocessing

Idea: Interleave search and preprocessing

- Preprocessing can be extremely beneficial: most solvers in SAT competitions use bounded variable elimination, subsumption and self-subsuming resolution
- Problem: Many preprocessing techniques, though polynomial, require considerable time
- Possible Solution:
  - Interrupt preprocessing techniques after some time
  - Resume on restart
  - Limit preprocessing time in relation to search
- **Discussion:** What are the problems that can arise in practice when SAT instances are solved incrementally?

Inprocessing: Autarkies

Autarky reasoning is used by state-of-the-art SAT solvers (cf. `Kissat`) to remove clauses from a formula.

Autarky-based Clause Removal

Let a formula F and a partial assignment A be given.

- A clause $C \in F$ is touched by A if it contains the negation of a literal assigned in A
- A clause $C \in F$ is 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.
All clauses touched by an autarky can be removed.

Discussion: How to obtain partial assignments to probe for autarky-based clause removal?

Autarky-based Clause Removal

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

Techniques that do not Reduce the Formula Size

Next Time

- **Resolution Calculus** (a.k.a. Clause Learning)
- **Tseitin's Extension Rule**: Introduce definitions of new variables as a conjunction of existing literals (a.k.a. Bounded Variable Addition (BVA)). Some formulas have refutations of exponential size in the resolution calculus, but of polynomial size in extended resolution, e.g., pigeonhole formulas, mutilated chessboard, ...

  $\longrightarrow$ `SBVA-CaDiCaL`: Winner of SAT Competition 2023
- **Symmetry Breaking Predicates**: Exclusion of Symmetric Solutions

  $\longrightarrow$ `BreakId-Kissat`: Special Price at SAT Competition 2023
- **PReLearning**: Preprocessing adds specific Propagation Redundant (PR) clauses

  $\longrightarrow$ `KissatMAB-Prop`: Winner of SAT Competition 2023 on UNSAT instances