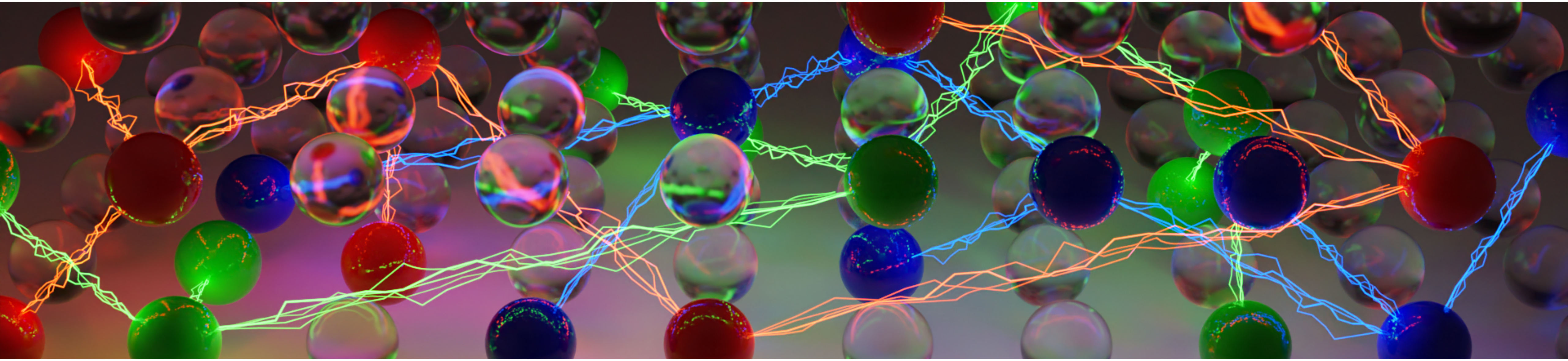

Practical SAT Solving

Lecture 6 – Preprocessing

Markus Iser, Dominik Schreiber | May 26, 2025

Conflict-driven Clause Learning (CDCL)

Recap: Last Time

- Efficient Unit Propagation
- Clause Forgetting
- Modern Decision Heuristics

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

```
if not PREPROCESSING then return UNSAT
while A is not complete do
    UNIT PROPAGATION
    if A falsifies a clause in F then
        if decision level is 0 then return UNSAT
        else
            (clause, level) ← CONFLICT-ANALYSIS
            add clause to F and backtrack to level
            continue
    if RESTART then backtrack to level 0
    if CLEANUP then forget some learned clauses
    BRANCHING
return SAT
```

Conflict-driven Clause Learning (CDCL)

Recap: Last Time

- Efficient Unit Propagation
- Clause Forgetting
- Modern Decision Heuristics

Today

- Preprocessing

Algorithm 2: CDCL(CNF Formula F, &Assignment A $\leftarrow \emptyset$)

```
if not PREPROCESSING then  return UNSAT
while A is not complete do
    UNIT PROPAGATION
    if A falsifies a clause in F then
        if decision level is 0 then  return UNSAT
        else
            (clause, level) ← CONFLICT-ANALYSIS
            add clause to F and backtrack to level
            continue
    if RESTART then  backtrack to level 0
    if CLEANUP then  forget some learned clauses
    BRANCHING
return SAT
```

Preprocessing

Preprocessing takes place between problem encoding and its solution.

Preprocessing is . . .

- a form of reencoding a problem
- a form of reasoning itself

Classic Preprocessing Techniques

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

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

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 $\{a, b, d\}$

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 $D \subseteq C$.

How to check if there exists a clause D that subsumes C?

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 `propagate()` to find subsuming clauses.

- **Optimization 1:** Use one-watched literal data-structure
- **Optimization 2:** Watch literals with the fewest occurrences
- **Optimization 3:** Keep literals sorted and perform merge-sort style subset check

Subsumption

Implementation 2: Backward Subsumption

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

Learned clauses can subsume other clauses.

Algorithm 3: Signature-based Subsumption

`// Initialization:`

for *clause* $\in$ *formula* **do**

  clause.signature = 0

  for *lit* $\in *$*clause* **do**

    clause.signature |= 1ull $<<$ $\big(id(lit)\%64\big)$

`// Subsumption Check:`

if $D.signature \,\&\, invert(C.signature) == 0$ **then**

  `// Check if` D `subsumes` C

Subsumption

Implementation 2: Backward Subsumption

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

Learned clauses can subsume other clauses.
How about the other way around?

Algorithm 4: Signature-based Subsumption

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

Subsumption

Implementation 2: Backward Subsumption

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

Learned clauses can subsume other clauses.

- **Optimization 1:** Only check the clauses of the variable with the fewest occurrences (scales to large formulas, might miss some subsumptions)
- **Optimization 2:** Use signatures to skip the majority of subsumption checks (cf. Bloom filters)

Algorithm 5: Signature-based Subsumption

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

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.

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

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

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.

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

$\longrightarrow$ 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

Bounded Variable Elimination (BVE)

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.

Bounded Variable Elimination (BVE)

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.

Bounded Variable Elimination (BVE)

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

- **Note 1:** Variables of removed clauses can be rescheduled for further elimination attempts
- **Note 2:** Resolvent can trigger further subsumptions and vice versa
- **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

BVE is particularly effective in presence of functional definitions (cf. Tseitin encoding)

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.

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
- Together with BVE: relationship with circuit-level simplification techniques

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
- Together with BVE: relationship with circuit-level simplification techniques

Generalization: Covered Clauses

A clause C is covered if it can be turned into a blocked clause by adding a covered literal.
A literal x is covered in C, if C contains a literal y such that all non-tautological resolvents of C on y contain x.

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 needs some processing in order to 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.

The order is important, such that the last literal-clause pair (l, C) in S needs to be the first to be processed.

Algorithm 6: Solution Reconstruction

Data: Assignment A, Stack S

while *S is not empty* **do**
  remove the last literal-clause pair (l, C) from S;
  if *C is not satisfied by A* **then**
    $A := (A \setminus \{l = 0\}) \cup \{l = 1\}$

If variables remain unassigned in A, then assign them an arbitrary value.

Recap.

Classic Techniques

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

Next Up

Relationship between preprocessing techniques and gate encodings

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 $X := \{x_1, \dots, x_n\}$.

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

1. there exists at least one output assignment $O \in \{o, \overline{o}\}$ such that $I \cup O \models E$ (left-totality)
2. 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.

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

1. 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.

2. 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.

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 $E \equiv o \leftrightarrow x \wedge y$)

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

1. 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,
2. 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 $F = \big\{\{q\}, \{\neg q, o, p\}, \{\neg q, \neg o, \neg p\}\big\} \cup E$?

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 $E_{\overline{x}} \otimes R_x$.

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 Bounds**
  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.

Recap.

Recap.

- Classic Preprocessing Techniques:
  Subsumption, Self-subsuming Resolution, Bounded Variable Elimination, Blocked Clause Elimination
- Relationship between Preprocessing Techniques and Gate Encodings
- Scheduling of Preprocessing Techniques

Next Session

Propagation-based Redundancy Notions and Proof Systems