CDCL and Proof Production for (UN)SAT

Sankalp Gambhir

May 20, 2025

The Problem

We have a language, of a countable set of atoms $A = \{x, y, z, \ldots\}$, and ways to combine them $x \wedge y$, $x \vee y$, and $\neg x$ with the usual interpretations. These terms form a Boolean algebra.

The Problem

We have a language, of a countable set of atoms $A = \{x, y, z, \ldots\}$, and ways to combine them $x \wedge y$, $x \vee y$, and $\neg x$ with the usual interpretations. These terms form a Boolean algebra.

A *literal* is an atom x or its negation $\neg x$.

The Problem

We have a language, of a countable set of atoms $A = \{x, y, z, \ldots\}$, and ways to combine them $x \wedge y$, $x \vee y$, and $\neg x$ with the usual interpretations. These terms form a Boolean algebra.

A *literal* is an atom x or its negation $\neg x$.

A *clause* is a disjunction of literals $C = \neg x \vee y \vee \neg z \vee \ldots$.

The Problem

We have a language, of a countable set of atoms $A = \{x, y, z, \dots\}$, and ways to combine them $x \wedge y$, $x \vee y$, and $\neg x$ with the usual interpretations. These terms form a Boolean algebra.

A *literal* is an atom x or its negation $\neg x$.

A *clause* is a disjunction of literals $C = \neg x \vee y \vee \neg z \vee \dots$.

A *formula in conjunctive normal form (CNF)* is a conjunction of clauses $F = C_1 \wedge C_2 \wedge \dots$.

The Problem

The *(Clausal) Boolean Satisfiability Problem (Clausal SAT)* is the problem of determining whether a given formula F in CNF is satisfiable, i.e., whether there exists a mapping $\sigma : A \to \{0, 1\}$ (or $\mathbb{B}$) such that $F\sigma = 1$ under the usual interpretation of the two-element Boolean algebra.

Typically, we are also interested in producing such a mapping, if it exists.

We looked at interpreting the SAT problem very plainly as a reading of the definition:

```
def checkSat(f: Formula[Prop]): SatResult[Prop] =
  f.frees.toSet.subsets
    .find(f.evaluateUnder) match
    case Some(s) => Sat(s)
    case None => Unsat
```

Recap

We tried to come up with a way to frame this as a decision procedure that checks variables one by one, wherein we got a branching procedure:

```
def checkSat(f: CNF[Prop]): SatResult[Prop] =
  def rec(clauses: Set[Clause[Prop]], choices: List[Literal[Prop]], frees:
 List[Prop]): SatResult[Prop] =
    if clauses.isEmpty then Sat(choices)
    else if clauses.exists(_.isEmpty) then Unsat
    else if frees.isEmpty then Unsat
    else
      // decide
      val x = frees.head
      val rest = frees.tail
      // branch
      lazy val pos =
        rec(clauses.substitute(x, true), Pos(x) :: choices, rest)
      lazy val neg =
        rec(clauses.substitute(x, false), Neg(x) :: choices, rest)
      pos.orElse(neg)
  rec(f.clauses, Nil, f.frees)

```

What is our search space?

Proof Production

Extension of the original problem: with an answer to the SAT problem, can you also supply a *certificate* for the answer that can be efficiently checked?

Extension of the original problem: with an answer to the SAT problem, can you also supply a *certificate* for the answer that can be efficiently checked?

On SAT:

Proof Production

Extension of the original problem: with an answer to the SAT problem, can you also supply a *certificate* for the answer that can be efficiently checked?

On SAT: The model is a certificate. We can substitute the model into the formula and check in linear time whether the formula indeed evaluates to true.

Extension of the original problem: with an answer to the SAT problem, can you also supply a *certificate* for the answer that can be efficiently checked?

On SAT: The model is a certificate. We can substitute the model into the formula and check in linear time whether the formula indeed evaluates to true.

On UNSAT:

Proof Production

Extension of the original problem: with an answer to the SAT problem, can you also supply a *certificate* for the answer that can be efficiently checked?

On SAT: The model is a certificate. We can substitute the model into the formula and check in linear time whether the formula indeed evaluates to true.

On UNSAT: Note that if $f_1 \iff f_2$, then $\forall\sigma : A \to \mathbb{B}.\ f_1\sigma = f_2\sigma$.

If $\nexists\sigma. f\sigma = 1$, then $f \iff \bot$. Since $\bot \Rightarrow f$ always holds, our 'no' is actually stating that $f \Rightarrow \bot$.

A certificate for this is a proof of $f \Rightarrow \bot$, or a *refutation*.

Where do proofs squeeze in?

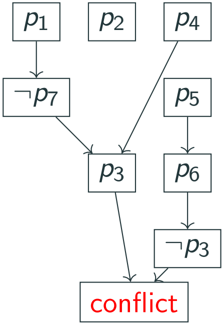
```scala
def checkSat(f: CNF[Prop]): SatResult[Prop] =
  def rec(clauses: Set[Clause[Prop]], choices: List[Literal[Prop]], frees: List[Prop]): SatResult[Prop] =
    if clauses.isEmpty then Sat(choices)
    else if clauses.exists(_.isEmpty) then Unsat
    else if frees.isEmpty then Unsat
    else
      // decide
      val x = frees.head
      val rest = frees.tail
      // branch
      lazy val pos =
        rec(clauses.substitute(x, true), Pos(x) :: choices, rest)
      lazy val neg =
        rec(clauses.substitute(x, false), Neg(x) :: choices, rest)
      pos.orElse(neg)
  rec(f.clauses, Nil, f.frees)

```

Using Partial Information

c_1	$\neg p_4 \vee p_5$
c_2	$\neg p_4 \vee p_3 \vee p_7$
c_3	$\neg p_5 \vee p_6$
c_4	$\neg p_3 \vee \neg p_6$
c_5	$p_4 \vee p_7 \vee \neg p_5$
c_6	$p_5 \vee p_3$
c_7	$p_5 \vee \neg p_3 \vee \neg p_2$
c_8	$\neg p_1 \vee \neg p_2$

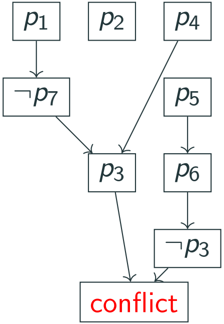
Using Partial Information

c_1	$\neg p_4 \vee p_5$
c_2	$\neg p_4 \vee p_3 \vee p_7$
c_3	$\neg p_5 \vee p_6$
c_4	$\neg p_3 \vee \neg p_6$
c_5	$p_4 \vee p_7 \vee \neg p_5$
c_6	$p_5 \vee p_3$
c_7	$p_5 \vee \neg p_3 \vee \neg p_2$
c_8	$\neg p_1 \vee \neg p_2$



Using Partial Information

c_1	$\neg p_4 \lor p_5$
c_2	$\neg p_4 \lor p_3 \lor p_7$
c_3	$\neg p_5 \lor p_6$
c_4	$\neg p_3 \lor \neg p_6$
c_5	$p_4 \lor p_7 \lor \neg p_5$
c_6	$p_5 \lor p_3$
c_7	$p_5 \lor \neg p_3 \lor \neg p_2$
c_8	$\neg p_1 \lor \neg p_2$



We can say more concretely, $\neg p_1 \lor \neg p_4$, and jump back.

CDCL

CDCL

That was CDCL, thanks and bye.

CDCL

That was CDCL, thanks and bye.

Conflict-driven clause learning

Proof production for CDCL

Question 1: what is a conflict? What are we saying, precisely, when we discover a conflict?

Proof production for CDCL

Question 1: what is a conflict? What are we saying, precisely, when we discover a conflict?

Question 2: what about the little inferences we made at each step?

Proof production for CDCL

Question 1: what is a conflict? What are we saying, precisely, when we discover a conflict?

Question 2: what about the little inferences we made at each step?

Question 3: how do we compose these together?

Proof production for CDCL

Question 1: what is a conflict? What are we saying, precisely, when we discover a conflict?

Question 2: what about the little inferences we made at each step?

Question 3: how do we compose these together?

Question 4: how do we stop? (or even, do we?)

Thank you!