KoSAT: Pure Kotlin CDCL SAT Solver

MENTORS : Dmitry Ivanov & Stepan Kochemazov & Konstantin Chukharev
BY: Nikolay Stepanov

Boolean Satisfiability Problem

The problem of determining if there exists an interpretation that satisfies a given Boolean formula

Applications:

- ***Bounded Model Checking***
- ***Software*** & Hardware ***Verification***
- Automated Theorem Proving
- Finite Mathematics
- … and a lot of other NP-hard problems

Programs for solving SAT problem are called ***SAT solvers***.

KoSAT - Pure Kotlin CDCL SAT Solver

Most solvers are written with performance in mind. While mechanical sympathy drastically improves performance, it tends to make code less readable, and not well-suited for ***educational purposes***.

- KoSAT is written in a high-level language: ***Kotlin***
- ***Hackable*** without much field-specific knowledge
- Runs in different ***environments*** (e.g. JVM, JS)
- Compares with modern solvers

Customers

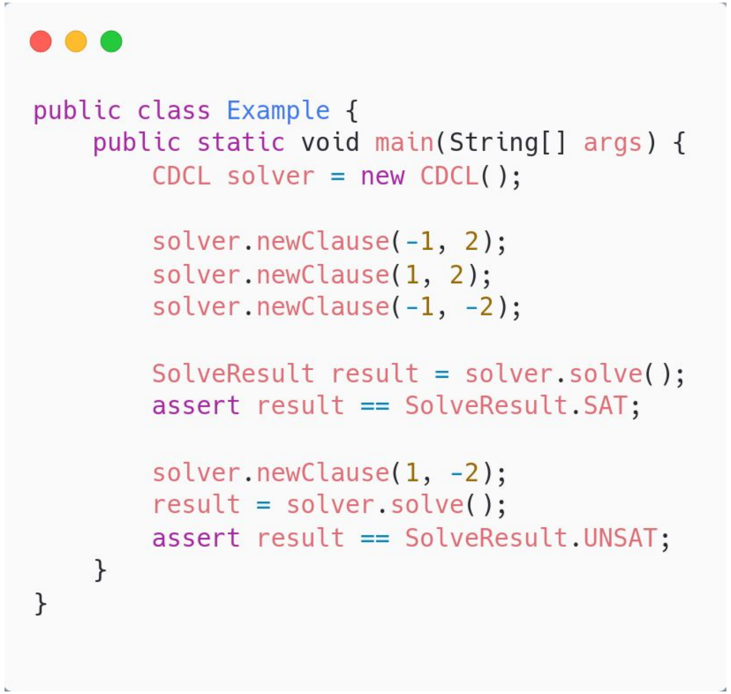
- Educators
  - will find KoSAT visualization tool useful for ***teaching*** the CDCL algorithm
- Researchers
  - will find KoSAT easy to ***modify*** and ***experiment*** with
- Engineers
  - will find KoSAT easy to ***use*** *the solver within their product.*

What have been done?

I picked up the project from an internship a year ago. Over the past two months, numerous features have been implemented:

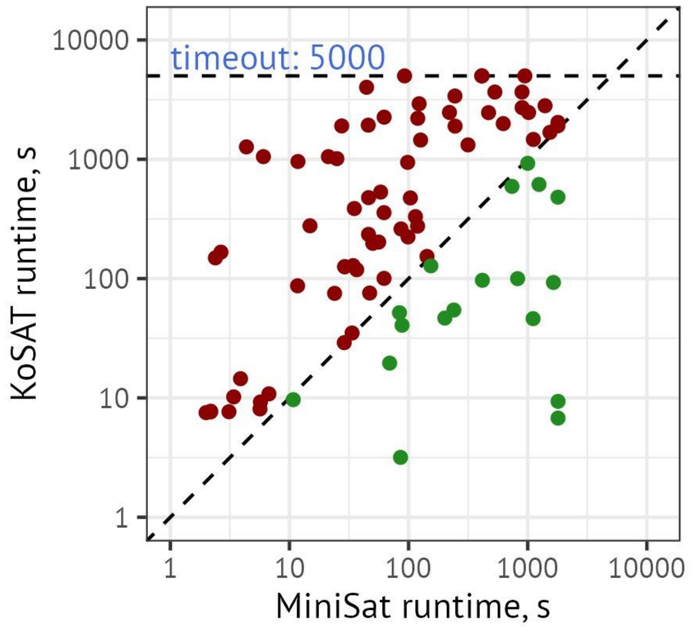
- Bounded variable elimination (with lots of extra stuff)
- Proof generation
- Failed Literal Probing
- Equivalent Literal Substitution
- Reconstruction Stack
- On-the-fly hyper-binary resolution

Use KoSAT

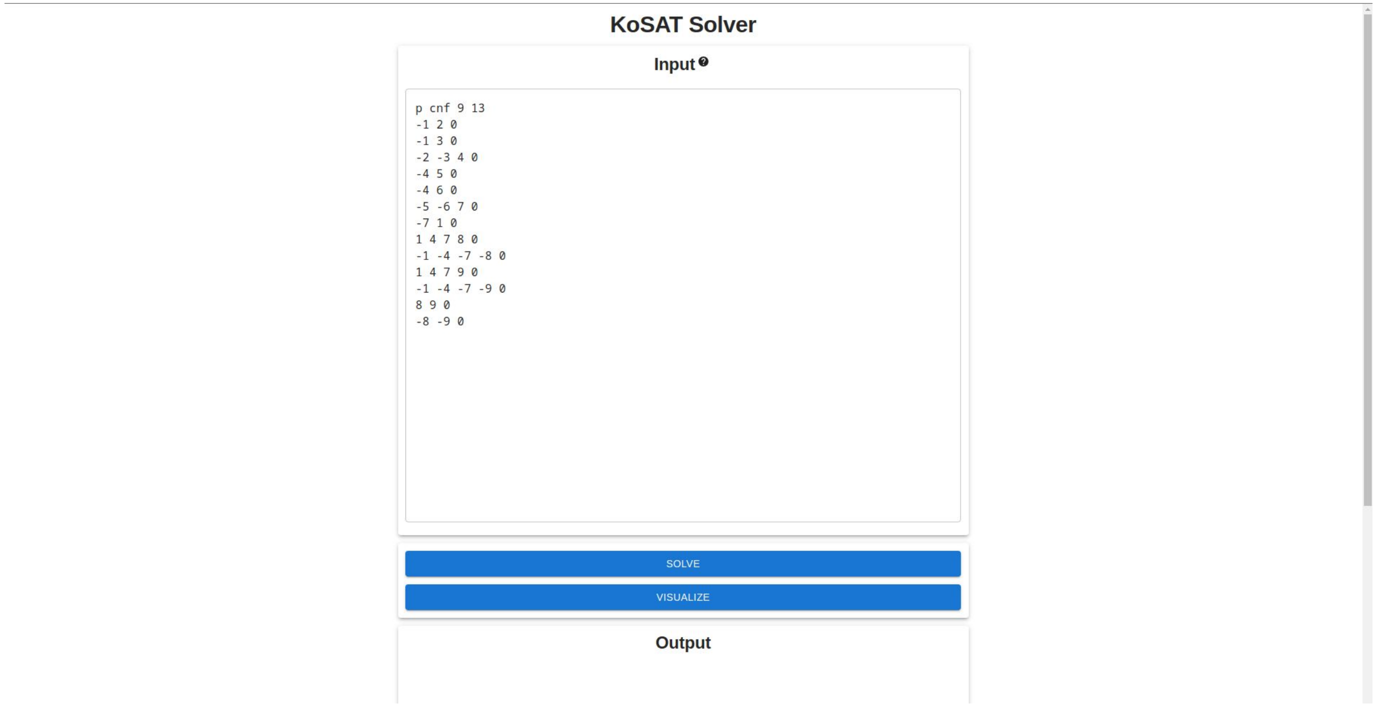


```
public class Example {
    public static void main(String[] args) {
        CDCL solver = new CDCL();

        solver.newClause(-1, 2);
        solver.newClause(1, 2);
        solver.newClause(-1, -2);

        SolveResult result = solver.solve();
        assert result == SolveResult.SAT;

        solver.newClause(1, -2);
        result = solver.solve();
        assert result == SolveResult.UNSAT;
    }
}
```

Performance

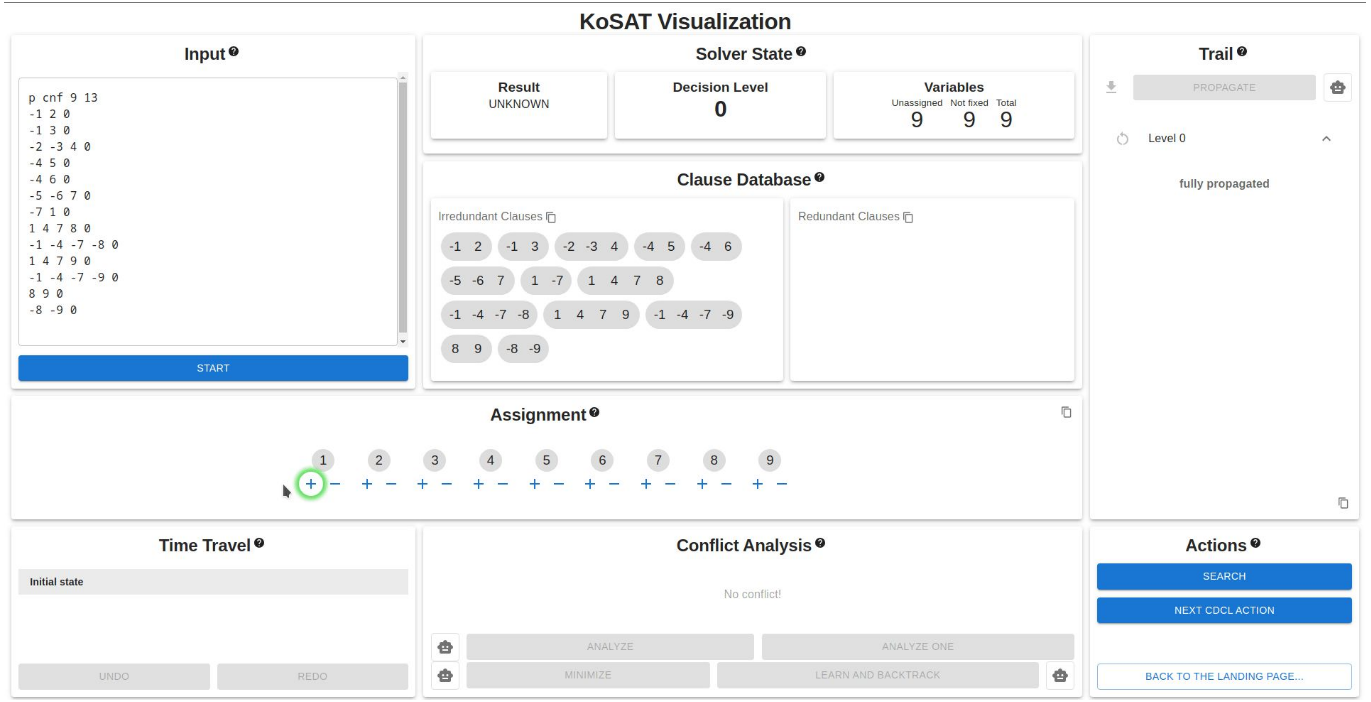


Web Application — Web Interface



```
p cnf 9 13
-1 2 0
-1 3 0
-2 -3 4 0
-4 5 0
-4 6 0
-5 -6 7 0
-7 1 0
1 4 7 8 0
-1 -4 -7 -8 0
1 4 7 9 0
-1 -4 -7 -9 0
8 9 0
-8 -9 0
```

Web Application — Visualization tool



KoSAT Visualization

Input

```
p cnf 9 13
-1 2 0
-1 3 0
-2 -3 4 0
-4 5 0
-4 6 0
-5 -6 7 0
-7 1 0
1 4 7 8 0
-1 -4 -7 -8 0
1 4 7 9 0
-1 -4 -7 -9 0
8 9 0
-8 -9 0
```

START

Solver State

Result	Decision Level	Variables: Unassigned	Not fixed	Total
UNKNOWN	0	9	9	9

Clause Database

Irredundant Clauses

-1 2 | -1 3 | -2 -3 4 | -4 5 | -4 6 | -5 -6 7 | 1 -7 | 1 4 7 8 | -1 -4 -7 -8 | 1 4 7 9 | -1 -4 -7 -9 | 8 9 | -8 -9

Redundant Clauses

Assignment

1 2 3 4 5 6 7 8 9

Trail

PROPAGATE

Level 0

fully propagated

Time Travel

Initial state

UNDO REDO

Conflict Analysis

No conflict!

ANALYZE | ANALYZE ONE | MINIMIZE | LEARN AND BACKTRACK

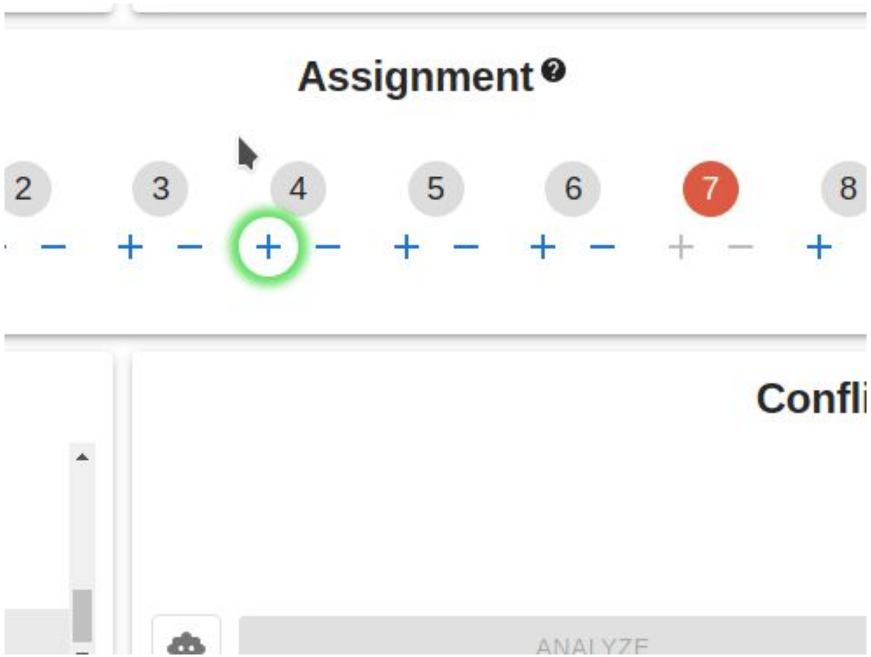
Actions

SEARCH

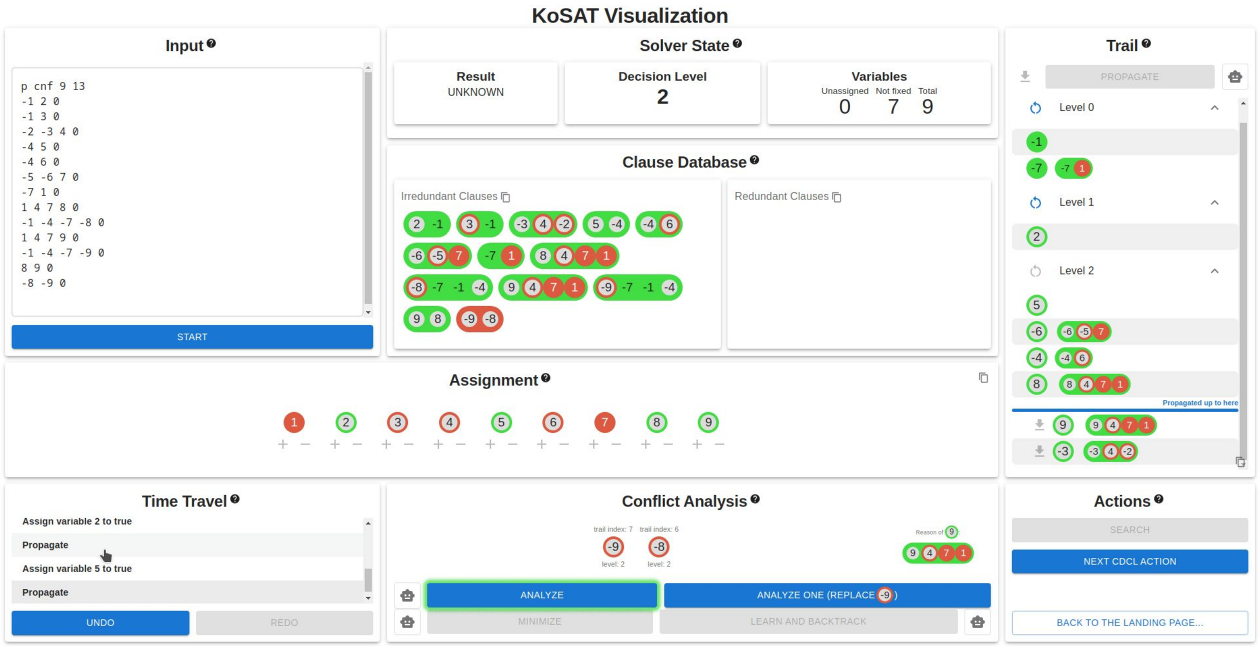
NEXT CDCL ACTION

BACK TO THE LANDING PAGE...

Web Application — In App Docs



Web Application — Time Travel



KoSAT Visualization

Input

```
p cnf 9 13
-1 2 0
-1 3 0
-2 -3 4 0
-4 5 0
-4 6 0
-5 -6 7 0
-7 1 0
1 4 7 8 0
-1 -4 -7 -8 0
1 4 7 9 0
-1 -4 -7 -9 0
8 9 0
-8 -9 0
```

Solver State

Result	Decision Level	Variables: Unassigned	Variables: Not fixed	Variables: Total
UNKNOWN	2	0	7	9

Time Travel

Assign variable 2 to true
Propagate
Assign variable 5 to true
Propagate

Web Application — Try it out!

https://www.utbot.org/kosat/

https://github.com/UnitTestBot/kosat

Stepanov Nikolay

@elteammate

elteammate@gmail.com

2023