Certifying Incremental SAT Solving

International Conference on Logic for Programming,
Artificial Intelligence and Reasoning
May 27, 2024

Katalin Fazekas[1], Florian Pollitt[2], Mathias Fleury[2], and Armin Biere[2]

[1]TU Wien, Vienna, Austria
[2]Albert–Ludwigs–University, Freiburg, Germany

Outline

Preliminaries

Certifying Incremental SAT Solving

Main Contributions

Boolean Satisfiability Problem (SAT)

- Propositional logic
- Conjunctive Normal Form (CNF): $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

Boolean Satisfiability Problem (SAT)

- Propositional logic
- Conjunctive Normal Form (CNF): $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

$$
\begin{array}{ll}
(a \vee \neg b) & \wedge \\
(a \vee b \vee c) & \wedge \\
(\neg a \vee \neg b) &
\end{array}
$$

Boolean Satisfiability Problem (SAT)

- Propositional logic
- Conjunctive Normal Form (CNF): $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

$$
\begin{aligned}
&(a \vee \neg b) && \wedge \\
&(a \vee b \vee c) && \wedge \\
&(\neg a \vee \neg b) &&
\end{aligned}
$$

- NP-complete decision problem: Is this formula satisfiable?

Boolean Satisfiability Problem (SAT)

- Propositional logic
- Conjunctive Normal Form (CNF): $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

$$
\begin{aligned}
&(a \vee \neg b) && \wedge \\
&(a \vee b \vee c) && \wedge \\
&(\neg a \vee \neg b)
\end{aligned}
$$

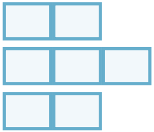
- NP-complete decision problem: Is this formula satisfiable?
- Answer: Yes, consider for example:

$$\{a = \top, b = \bot, c = \bot\}$$

Boolean Satisfiability Problem (SAT)

- Propositional logic
- Conjunctive Normal Form (CNF): $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

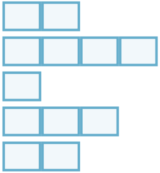
$$\begin{aligned} &(a \vee \neg b) &\wedge \\ &(a \vee b \vee c) &\wedge \\ &(\neg a \vee \neg b) & \end{aligned}$$



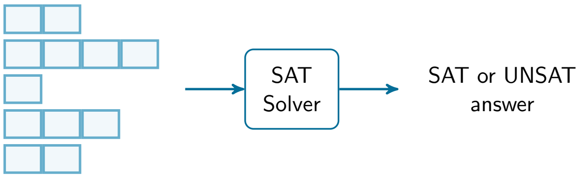
- NP-complete decision problem: Is this formula satisfiable?
- Answer: Yes, consider for example:

$$\{a = \top, b = \bot, c = \bot\}$$

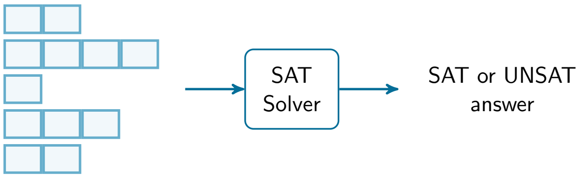
SAT Solvers



SAT Solvers

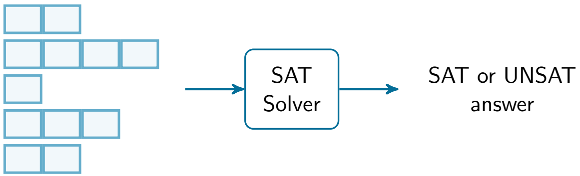


SAT Solvers



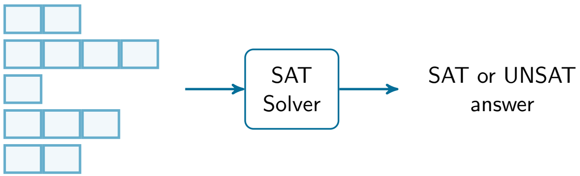
- Main approach: Conflict-Driven Clause-Learning (CDCL) algorithm

SAT Solvers



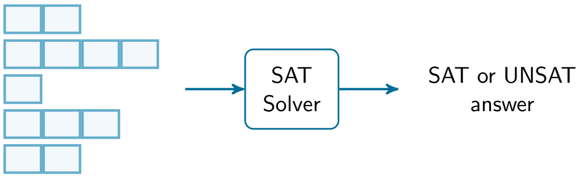
- Main approach: Conflict-Driven Clause-Learning (CDCL) algorithm
- Can scale to millions of variables and clauses

SAT Solvers



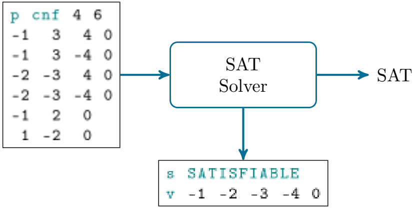
- Main approach: Conflict-Driven Clause-Learning (CDCL) algorithm
- Can scale to millions of variables and clauses
- Wide range of applications: verification, AI, solvers beyond SAT, planning ...

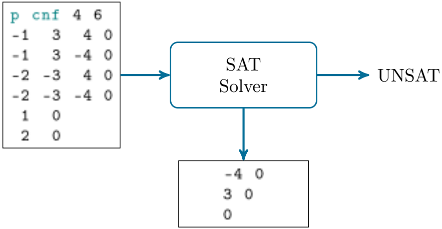
SAT Solvers



- Main approach: Conflict-Driven Clause-Learning (CDCL) algorithm
- Can scale to millions of variables and clauses
- Wide range of applications: verification, AI, solvers beyond SAT, planning . . .
- Important features:
  - Inprocessing: Efficient formula simplification techniques
  - **Verifiable result: Proofs & Solutions**

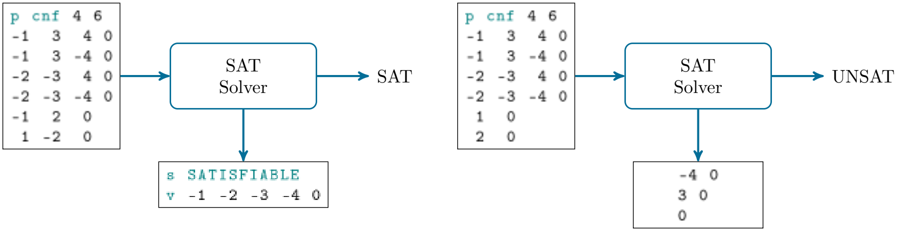
Verifiable Certificates – Proofs & Solutions of SAT Solvers





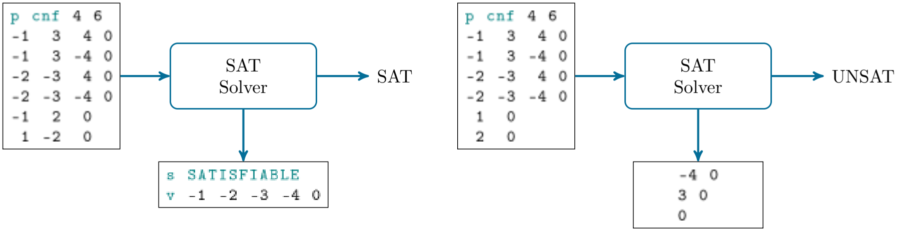
- Standardized input and output formats, guaranteed verifiable certificates.

Verifiable Certificates – Proofs & Solutions of SAT Solvers



- Standardized input and output formats, guaranteed verifiable certificates.
- Solution: Satisfying truth assignment.

Verifiable Certificates – Proofs & Solutions of SAT Solvers



- Standardized input and output formats, guaranteed verifiable certificates.
- Solution: Satisfying truth assignment.
- Proof of UNSAT: Record of all added (and deleted) clauses.
  - Derivation of the empty clause.

Example Input and Proof Formats

DIMACS

```
p cnf 4 8
1 2 -3 0
-1 -2 3 0
2 3 -4 0
-2 -3 4 0
-1 -3 -4 0
1 3 4 0
-1 2 4 0
1 -2 -4 0
```

DRUP [HeuleHW'14]

```
-3 -4 0
d -3 -1 -4 0
3 -4 0
d 2 3 -4 0
-4 0
3 0
-2 0
1 0
0
```

LRAT [CruzFHHKS-CADE'17]

```
9 -3 -4 0 5 1 8 0
9 d 5 0
10 3 -4 0 3 2 8 0
10 d 3 0
11 -4 0 9 10 0
12 3 0 11 6 7 2 0
13 -2 0 12 11 4 0
14 1 0 13 12 1 0
15 0 13 14 11 7 0
```

Incremental SAT Problems

- Sequence of SAT queries: $\langle Q_1, Q_2, \ldots, Q_n \rangle$ where $Q_i = (\Delta_i, A_i)$ for $1 \leq i \leq n$:
  - Set of clauses: Δ_i
  - Set of assumptions: A_i (temporary unit clauses)

Incremental SAT Problems

- Sequence of SAT queries: $\langle Q_1, Q_2, \ldots, Q_n \rangle$ where $Q_i = (\Delta_i, A_i)$ for $1 \leq i \leq n$:
  - Set of clauses: Δ_i
  - Set of assumptions: A_i (temporary unit clauses)

$$Q_i \text{ satisfiable } \Leftrightarrow (\wedge_{j=1}^{i} \Delta_j) \wedge A_i \text{ satisfiable}$$

Incremental SAT Problems

- Sequence of SAT queries: $\langle Q_1, Q_2, \ldots, Q_n \rangle$ where $Q_i = (\Delta_i, A_i)$ for $1 \leq i \leq n$:
  - Set of clauses: Δ_i
  - Set of assumptions: A_i (temporary unit clauses)

$$Q_i \text{ satisfiable } \Leftrightarrow (\wedge_{j=1}^{i} \Delta_j) \wedge A_i \text{ satisfiable}$$

- Example: $P = \langle Q_1, Q_2, Q_3 \rangle$:

$Q_1 = (\{C_1, C_2, C_3, C_4\}, \{a, b\}) \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge a \wedge b$ satisfiable?

Incremental SAT Problems

- Sequence of SAT queries: $\langle Q_1, Q_2, \ldots, Q_n \rangle$ where $Q_i = (\Delta_i, A_i)$ for $1 \leq i \leq n$:
  - Set of clauses: Δ_i
  - Set of assumptions: A_i (temporary unit clauses)

$$Q_i \text{ satisfiable } \Leftrightarrow (\wedge_{j=1}^{i} \Delta_j) \wedge A_i \text{ satisfiable}$$

- Example: $P = \langle Q_1, Q_2, Q_3 \rangle$:

$Q_1 = (\{C_1, C_2, C_3, C_4\}, \{a, b\}) \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge a \wedge b$ satisfiable?

$Q_2 = (\{C_5\}, \{\neg b\}) \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge \neg b$ satisfiable?

Incremental SAT Problems

- Sequence of SAT queries: $\langle Q_1, Q_2, \ldots, Q_n \rangle$ where $Q_i = (\Delta_i, A_i)$ for $1 \leq i \leq n$:
  - Set of clauses: Δ_i
  - Set of assumptions: A_i (temporary unit clauses)

$$Q_i \text{ satisfiable } \Leftrightarrow (\wedge_{j=1}^{i} \Delta_j) \wedge A_i \text{ satisfiable}$$

- Example: $P = \langle Q_1, Q_2, Q_3 \rangle$:

$$\begin{array}{lll}
Q_1 = (\{C_1, C_2, C_3, C_4\}, \{a, b\}) & \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge a \wedge b & \text{satisfiable?} \\
Q_2 = (\{C_5\}, \{\neg b\}) & \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge \neg b & \text{satisfiable?} \\
Q_3 = (\{C_6, C_7\}, \emptyset) & \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \wedge C_7 & \text{satisfiable?}
\end{array}$$

Incremental SAT Problems

- Sequence of SAT queries: $\langle Q_1, Q_2, \ldots, Q_n \rangle$ where $Q_i = (\Delta_i, A_i)$ for $1 \leq i \leq n$:
  - Set of clauses: Δ_i
  - Set of assumptions: A_i (temporary unit clauses)

$$Q_i \text{ satisfiable } \Leftrightarrow (\wedge_{j=1}^{i} \Delta_j) \wedge A_i \text{ satisfiable}$$

- Example: $P = \langle Q_1, Q_2, Q_3 \rangle$:

$$\begin{array}{lll}
Q_1 = (\{C_1, C_2, C_3, C_4\}, \{a, b\}) & \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge a \wedge b & \text{satisfiable?} \\
Q_2 = (\{C_5\}, \{\neg b\}) & \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge \neg b & \text{satisfiable?} \\
Q_3 = (\{C_6, C_7\}, \emptyset) & \mapsto C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \wedge C_7 & \text{satisfiable?}
\end{array}$$

- Several applications: Bounded Model Checking, SMT, Planning, ...

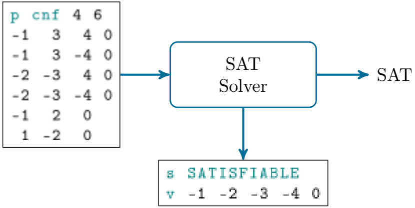
Outline

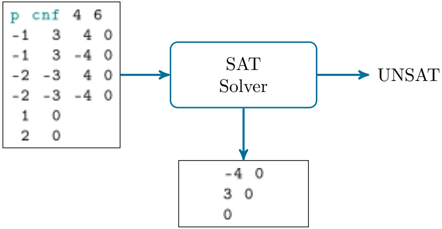
Preliminaries

Certifying Incremental SAT Solving

Main Contributions

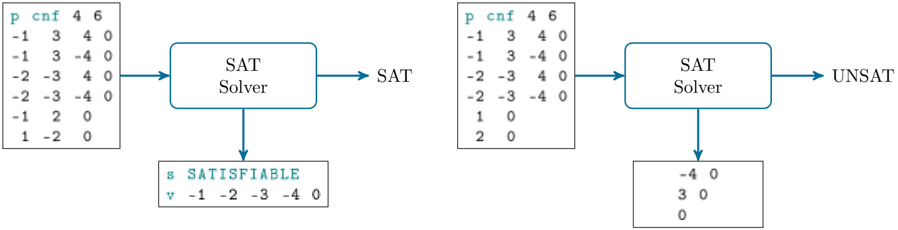
Verifiable Results of Incremental SAT Solvers





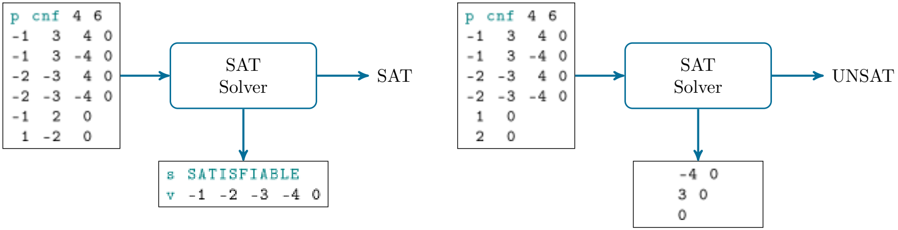
- $P = \langle Q_1, Q_2, Q_3 \rangle$

Verifiable Results of Incremental SAT Solvers



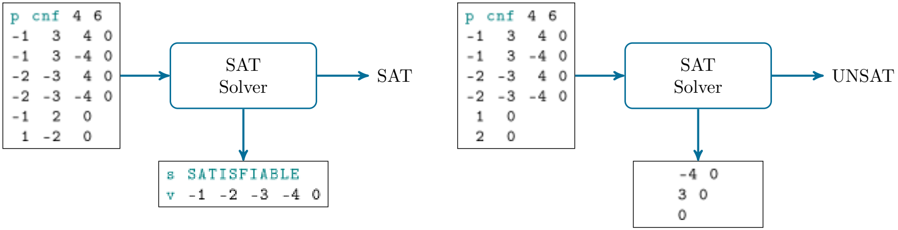
- $P = \langle Q_1, Q_2, Q_3 \rangle$
- Solution of satisfiable queries:
  - Satisfying truth assignment that agrees with assumptions.

Verifiable Results of Incremental SAT Solvers



- $P = \langle Q_1, Q_2, Q_3 \rangle$
- Solution of satisfiable queries:
  - Satisfying truth assignment that agrees with assumptions.
- Proofs of unsatisfiable queries:
  - without assumptions: Derivation of the empty clause.

Verifiable Results of Incremental SAT Solvers



- $P = \langle Q_1, Q_2, Q_3 \rangle$
- Solution of satisfiable queries:
  - Satisfying truth assignment that agrees with assumptions.
- Proofs of unsatisfiable queries:
  - without assumptions: Derivation of the empty clause.
  - with assumptions: **Not defined.**

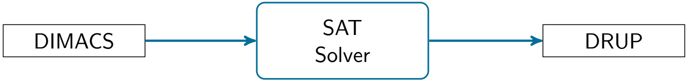
Outline

Preliminaries

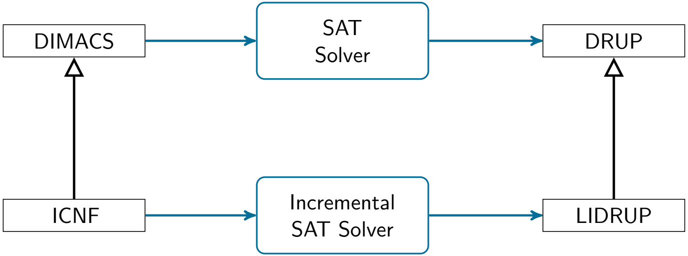
Certifying Incremental SAT Solving

Main Contributions

Incremental Input and Proof Formats



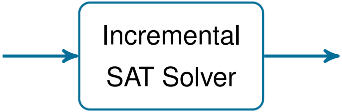
Incremental Input and Proof Formats



- New LIDRUP proof format:
  - Explicitly reasons about failed assumptions
  - Supports incremental inprocessing operations
  - Contains hints to speed up checking

LIDRUP Example

```
p icnf
i -1 3 4 0
i -1 3 -4 0
i -2 -3 4 0
i -2 -3 -4 0
q 1 2 0
```




```
p lidrup
i 1 -1 3 4 0
i 2 -1 3 -4 0
i 3 -2 -3 4 0
i 4 -2 -3 -4 0
q 1 2 0
l 5 -4 -2 -1 0 2 4 0
l 6 -2 -1 0 5 1 3 0
d 5 0
s UNSATISFIABLE
u 2 1 0 6 0
```

LIDRUP Example (cont.)

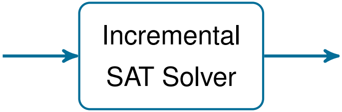
```
p icnf
i -1 3 4 0
i -1 3 -4 0
i -2 -3 4 0
i -2 -3 -4 0
q 1 2 0
s UNSATISFIABLE
f 1 2 0
```

Incremental SAT Solver

```
p lidrup
i 1 -1 3 4 0
i 2 -1 3 -4 0
i 3 -2 -3 4 0
i 4 -2 -3 -4 0
q 1 2 0
l 5 -4 -2 -1 0 2 4 0
l 6 -2 -1 0 5 1 3 0
d 5 0
s UNSATISFIABLE
u 2 1 0 6 0
```

LIDRUP Example (cont.)

```
p icnf
i -1 3 4 0
i -1 3 -4 0
i -2 -3 4 0
i -2 -3 -4 0
q 1 2 0
s UNSATISFIABLE
f 1 2 0
i -1 2 0
i 1 -2 0
q 0
```




```
p lidrup
i 1 -1 3 4 0
i 2 -1 3 -4 0
i 3 -2 -3 4 0
i 4 -2 -3 -4 0
q 1 2 0
l 5 -4 -2 -1 0 2 4 0
l 6 -2 -1 0 5 1 3 0
d 5 0
s UNSATISFIABLE
u 2 1 0 6 0
i 8 -1 2 0
i 9 1 -2 0
q 0
s SATISFIABLE
m -1 -2 -3 -4 0
```

LIDRUP Example (cont.)

```
p icnf
i -1 3 4 0
i -1 3 -4 0
i -2 -3 4 0
i -2 -3 -4 0
q 1 2 0
s UNSATISFIABLE
f 1 2 0
i -1 2 0
i 1 -2 0
q 0
s SATISFIABLE
v -1 -2 -3 -4 0
i 1 2 0
q 0
s UNSATISFIABLE
f 0
```

Incremental SAT Solver

```
p lidrup
i 1 -1 3 4 0
i 2 -1 3 -4 0
i 3 -2 -3 4 0
i 4 -2 -3 -4 0
q 1 2 0
l 5 -4 -2 -1 0 2 4 0
l 6 -2 -1 0 5 1 3 0
d 5 0
s UNSATISFIABLE
u 2 1 0 6 0
i 8 -1 2 0
i 9 1 -2 0
q 0
s SATISFIABLE
m -1 -2 -3 -4 0
i 10 1 2 0
q 0
l 11 2 0 8 10 0
l 12 -1 0 11 6 0
l 13 0 12 11 9 0
s UNSATISFIABLE
u 0 13 0
```

Syntax of LIDRUP

```
<lidrup>   =  { <comment> "\n" } <header> "\n" { <line> "\n" }
<comment>  =  "c " <any-character-but-new-line>
<header>   =  "p lidrup"
<line>     =  <comment> | <input> | <query> | <lemma> | <delete> |
              <weaken>  | <restore> | <status> | <model> | <core>
<input>    =  "i " <id> { " " <literal> } " 0"
<query>    =  "q" { " " <literal> } " 0"
<lemma>    =  "l " <id> { " " <literal> } " 0" { " " <id> } " 0"
<delete>   =  "d" { " " <id> } " 0"
<weaken>   =  "w" { " " <id> } " 0"
<restore>  =  "r" { " " <id> } " 0"
<status>   =  "s SATISFIABLE"  | "s UNSATISFIABLE" | "s UNKNOWN"
<model>    =  "m" { " " <literal> } " 0"
<core>     =  "u" { " " <literal> } " 0" { " " <id> } " 0"
<id>       =  <pos>
<literal>  =  <pos> | <neg>
<pos>      =  "1" | "2" | ... | <INT_MAX>
<neg>      =  "-" <pos>
```

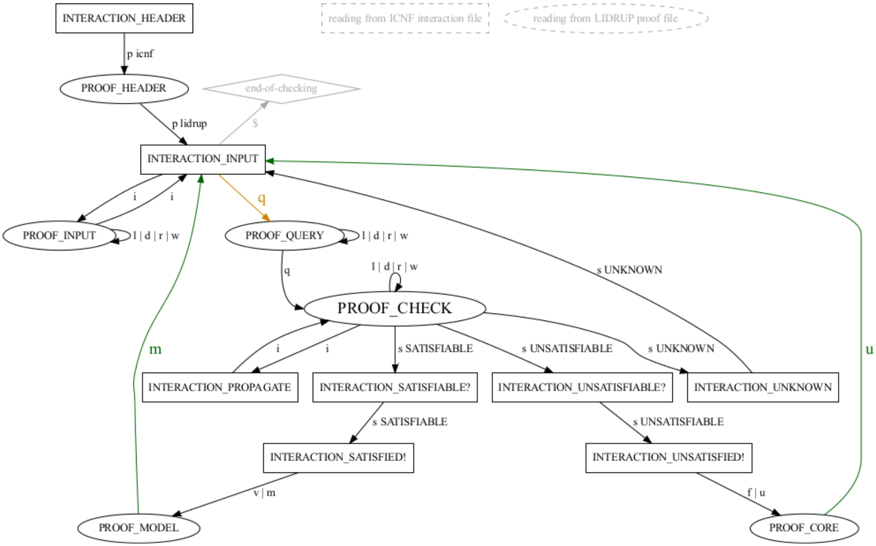
Semantics of LIDRUP

- σ: partial function that captures clause IDs
- $\mathcal{A}, \mathcal{P}$: Active and Passive sets of clauses

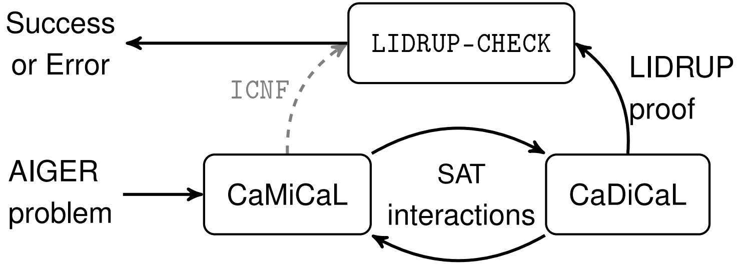
$$(\sigma', \mathcal{A}', \mathcal{P}') = \begin{cases} (\sigma \cup \{id \mapsto C\}, \mathcal{A} \cup \{id\}, \mathcal{P}) & \text{if S is `}\texttt{i}\text{' line with } id \in \mathbb{N}, \text{ clause } C \\ (\sigma \cup \{id \mapsto C\}, \mathcal{A} \cup \{id\}, \mathcal{P}) & \text{if S is `}\texttt{l}\text{' line with } id \in \mathbb{N}, \text{ clause } C \\ (\sigma \setminus \mathcal{I}, \quad \mathcal{A} \setminus \mathcal{I}, \quad \mathcal{P}) & \text{if S is `}\texttt{d}\text{' line with } \mathcal{I} \subset \mathbb{N} \text{ clause IDs} \\ (\sigma, \quad \mathcal{A} \setminus \mathcal{I}, \quad \mathcal{P} \cup \mathcal{I}) & \text{if S is `}\texttt{w}\text{' line with } \mathcal{I} \subset \mathbb{N} \text{ clause IDs} \\ (\sigma, \quad \mathcal{A} \cup \mathcal{I}, \quad \mathcal{P} \setminus \mathcal{I}) & \text{if S is `}\texttt{r}\text{' line with } \mathcal{I} \subset \mathbb{N} \text{ clause IDs} \\ (\sigma, \quad \mathcal{A}, \quad \mathcal{P}) & \text{otherwise.} \end{cases}$$

where $\sigma' = \sigma \setminus \mathcal{I}$ means that for all $n \in \mathcal{I}$: $\sigma'(n)$ is undefined, with other values unchanged.

LIDRUP-CHECK – Checking Incremental Proofs

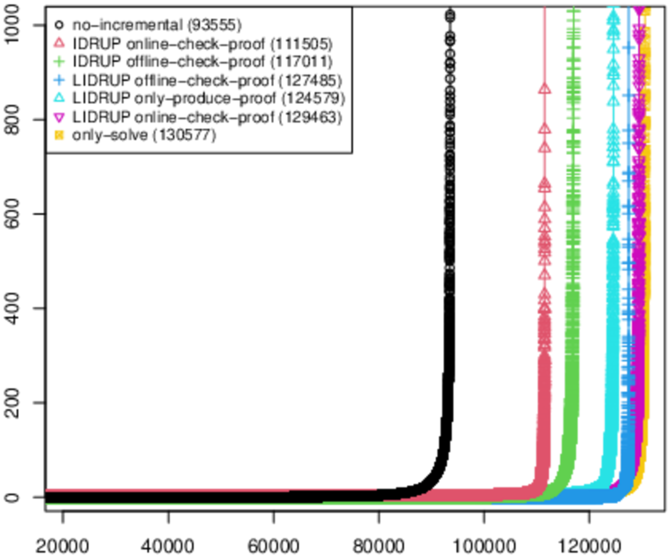


Experiments (1): Bounded Model Checking

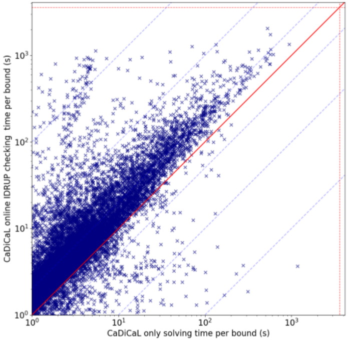


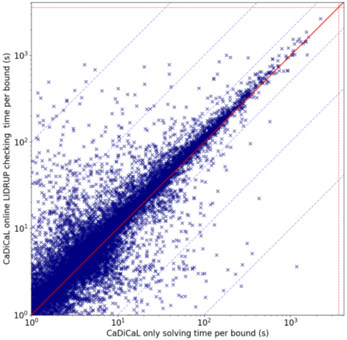
- Input: 300 AIGER models of Hardware Model Checking Competition 2017
  - 300 Incremental SAT Problems
- Maximum bound: 1000
  - Each query consists of at most 1000 unsatisfiable queries with assumptions

BMC Results – CaDiCaL



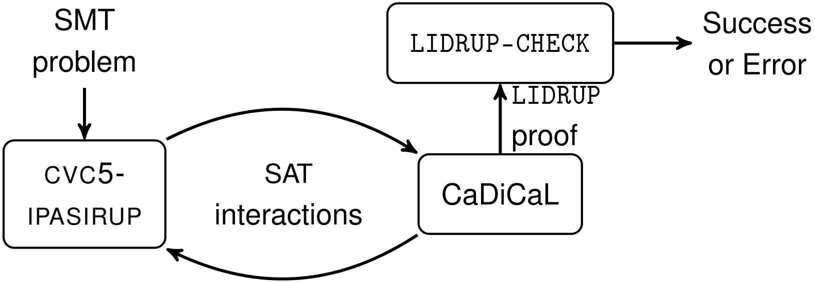
BMC Results – Checking vs. Solving times





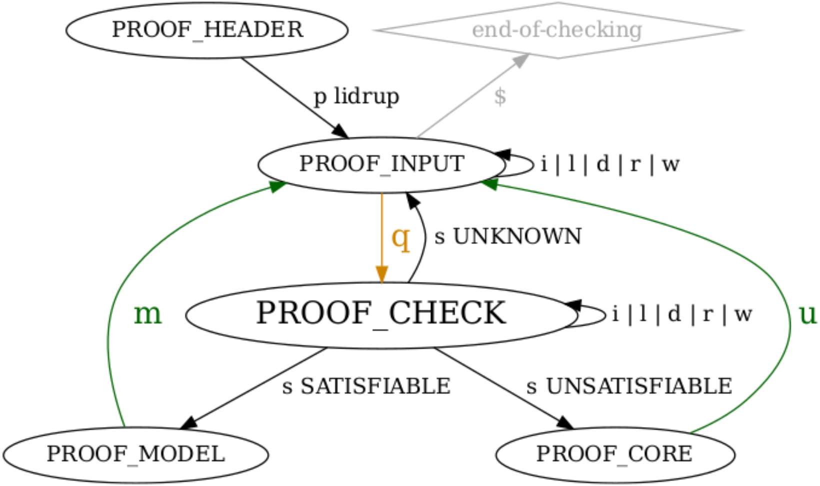
- Reasonable proof checking time
- Hints provide significant speed up in checking

Experiments (2): Satisfiability Modulo Theories



- Input: Non-Incremental and Incremental QF_LRA benchmarks of SMT-LIB
  - 1754 + 10 = 1764 Incremental SAT problems
  - 1754 + 1515 = 3269 SAT queries

LIDRUP-CHECK – Checking Incremental Proofs without Input



SMT Results: cvc5-ipasirup with CaDiCaL

Benchmark	All		Solved wo. Proof		Solved & Verified	
	Instances	Queries	Instances	Queries	Instances	Queries
QF_LRA	1754	1754	1671	1671	1650	1650
incr-QF_LRA	10	1515	2	730	2	725

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - $\rightarrow$ Gain verifiable results
  - $\rightarrow$ **Increase trustworthiness of Incremental SAT-based Tools**
  - $\rightarrow$ **Speed up proof checking via incrementality**

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of Incremental SAT-based Tools**
  - → **Speed up proof checking via incrementality**
- `LIDRUP-CHECK`: First prototype to check LIDRUP proofs
  - Incremental (online) proof checker
  - Works with and without input formula `ICNF` input

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of Incremental SAT-based Tools**
  - → **Speed up proof checking via incrementality**
- `LIDRUP-CHECK`: First prototype to check LIDRUP proofs
  - Incremental (online) proof checker
  - Works with and without input formula `ICNF` input

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of Incremental SAT-based Tools**
  - → **Speed up proof checking via incrementality**
- `LIDRUP-CHECK`: First prototype to check LIDRUP proofs
  - Incremental (online) proof checker
  - Works with and without input formula `ICNF` input
- Future Work:
  - Backward proof checking, trimming
  - Verify proof checker
  - Other proof formats (e.g. veriPB)

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of Incremental SAT-based Tools**
  - → **Speed up proof checking via incrementality**
- `LIDRUP-CHECK`: First prototype to check LIDRUP proofs
  - Incremental (online) proof checker
  - Works with and without input formula `ICNF` input
- Future Work:
  - Backward proof checking, trimming
  - Verify proof checker
  - Other proof formats (e.g. veriPB)

Thank you!