Incremental Proofs for Bounded Model Checking

Methoden und Beschreibungssprachen zur Modellierung
und Verifikaton von Schaltungen und Systemen
February 15, 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 & Motivation

Our Contributions

Conclusion

Boolean Satisfiability Problem (SAT)

- Propositional logic

$$(a \vee \overline{b}) \wedge (a \vee b) \wedge (\overline{a} \vee \overline{b})$$

Boolean Satisfiability Problem (SAT)

- Propositional logic
- NP-complete problem: Is this set of clauses satisfiable?

$$(a \vee \overline{b}) \wedge (a \vee b) \wedge (\overline{a} \vee \overline{b})$$

Boolean Satisfiability Problem (SAT)

- Propositional logic
- NP-complete problem: Is this set of clauses satisfiable?

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

$$(a \vee \overline{b}) \wedge (a \vee b) \wedge (\overline{a} \vee \overline{b})$$

SAT-based Bounded Model Checking

- Encode HW/SW system and its properties into propositional logic

SAT-based Bounded Model Checking

- Encode HW/SW system and its properties into propositional logic
- Goal: Given safety property, uncover if there is a possible way to violate it

SAT-based Bounded Model Checking

- Encode HW/SW system and its properties into propositional logic
- Goal: Given safety property, uncover if there is a possible way to violate it
- Bound: limit the depth of the search in the state space
  - Incrementally increase the bound on the number of steps explored

SAT-based Bounded Model Checking

- Encode HW/SW system and its properties into propositional logic
- Goal: Given safety property, uncover if there is a possible way to violate it
- Bound: limit the depth of the search in the state space
  - Incrementally increase the bound on the number of steps explored

F_i satisfiable $\leftrightarrow$ there is a property violation up to i steps

Incremental SAT Problems

- In practice problems often formulated step-by-step:

Is formula F_0	satisfiable?
Is formula $F_0 \wedge F_1$	satisfiable?
Is formula $F_0 \wedge F_1 \wedge F_2$	satisfiable?

Incremental SAT Problems

- In practice problems often formulated step-by-step:

Is formula F_0	satisfiable?
Is formula $F_0 \wedge F_1$	satisfiable?
Is formula $F_0 \wedge F_1 \wedge F_2$	satisfiable?

- Sequence of decision problems where each problem is an extension or slight modification of the previous one.

Incremental SAT Problems

- In practice problems often formulated step-by-step:

Is formula F_0	under $\overline{x}_1$	satisfiable?
Is formula $F_0 \wedge F_1$	under x_1 and $\overline{x}_2$	satisfiable?
Is formula $F_0 \wedge F_1 \wedge F_2$	under no assumption	satisfiable?

- Sequence of decision problems where each problem is an extension or slight modification of the previous one.
- Assumptions: Temporary constraints that are considered in the next query and after that immediately deleted.

Incremental SAT Problems

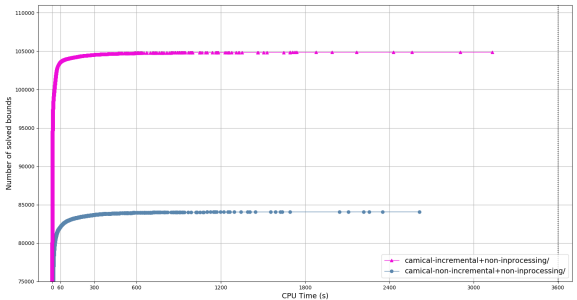
- In practice problems often formulated step-by-step:

Is formula F_0	under $\overline{x}_1$	satisfiable?
Is formula $F_0 \wedge F_1$	under x_1 and $\overline{x}_2$	satisfiable?
Is formula $F_0 \wedge F_1 \wedge F_2$	under no assumption	satisfiable?

- Sequence of decision problems where each problem is an extension or slight modification of the previous one.
- Assumptions: Temporary constraints that are considered in the next query and after that immediately deleted.
- **Incremental Solvers**: Can solve each formula with the exact same solver.
  - \+ Reuse reasoning steps instead of repeating them.

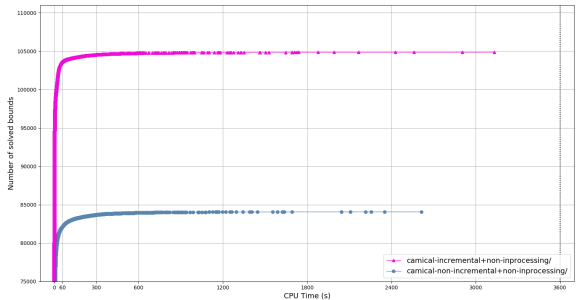
SAT-based Bounded Model Checking & Incremental SAT

+ Incremental reasoning can lead to significant speed up in BMC



SAT-based Bounded Model Checking & Incremental SAT

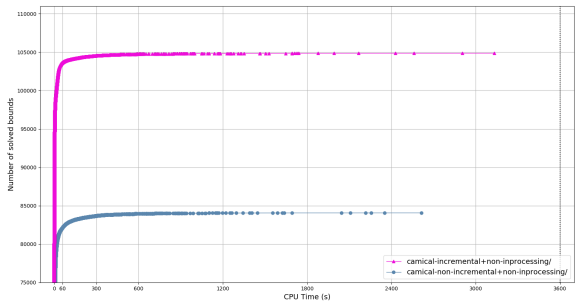
+ Incremental reasoning can lead to significant speed up in BMC



+ Verifiable results of (non-incremental) SAT solvers

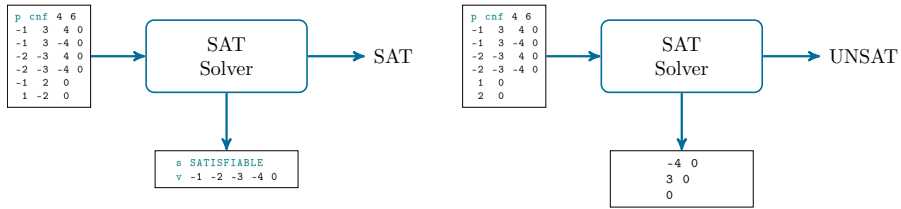
SAT-based Bounded Model Checking & Incremental SAT

+ Incremental reasoning can lead to significant speed up in BMC



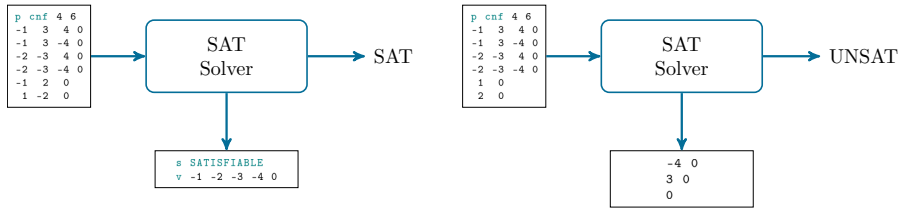
+ Verifiable results of (non-incremental) SAT solvers
− **Not all results are certified in incremental SAT solvers.**

Verifiable Results – Proofs & Solutions of SAT Solvers



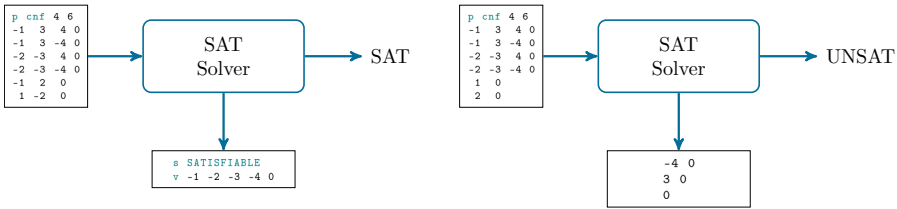
- Standardized input and output formats, guaranteed verifiable certificates.

Verifiable Results – Proofs & Solutions of SAT Solvers



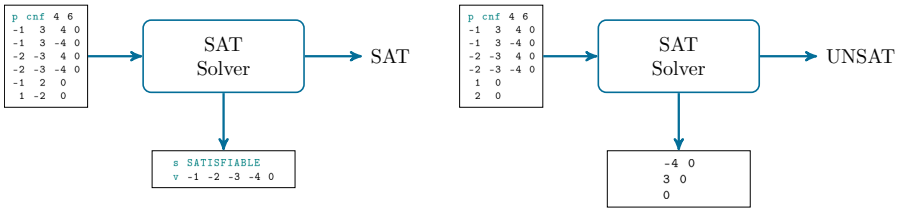
- Standardized input and output formats, guaranteed verifiable certificates.
- Solution of SAT: Satisfying truth assignment that agrees with assumptions.

Verifiable Results – Proofs & Solutions of SAT Solvers



- Standardized input and output formats, guaranteed verifiable certificates.
- Solution of SAT: Satisfying truth assignment that agrees with assumptions.
- Proof of UNSAT:
  - wo. assumptions: Derivation of the empty clause.

Verifiable Results – Proofs & Solutions of SAT Solvers



- Standardized input and output formats, guaranteed verifiable certificates.
- Solution of SAT: Satisfying truth assignment that agrees with assumptions.
- Proof of UNSAT:
  - wo. assumptions: Derivation of the empty clause.
  - with assumptions: **Not defined, no guaranties what will be derived.**

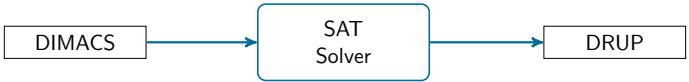
Outline

Preliminaries & Motivation

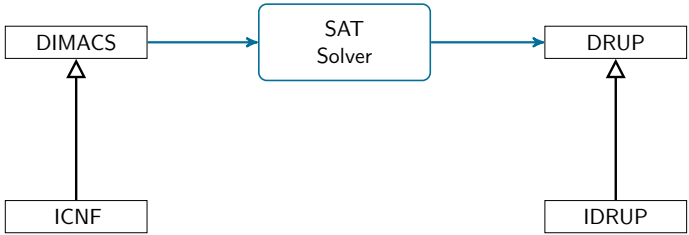
Our Contributions

Conclusion

Incremental Input and Proof Formats

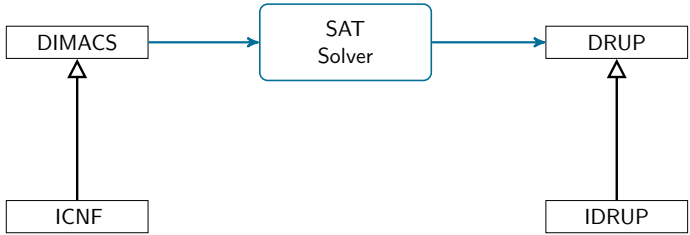


Incremental Input and Proof Formats



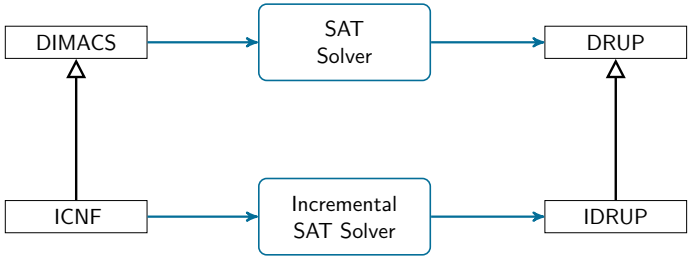
- New ICNF input format:
  - encodes complete incremental queries

Incremental Input and Proof Formats



- New ICNF input format:
  - encodes complete incremental queries
- New IDRUP proof format:
  - explicitly reasons about failed assumptions
  - supports incremental inprocessing operations

Incremental Input and Proof Formats



- New ICNF input format:
  - encodes complete incremental queries
- New IDRUP proof format:
  - explicitly reasons about failed assumptions
  - supports incremental inprocessing operations

Syntax & Semantics of the New Formats

```
<icnf>     = <comments> "p icnf\n" <lines>
<comments> = { <comment> "\n" }
<lines>    = { <comment> "\n" | <line> "\n" }
<comment>  = "c"   " " <anything-but-new-line>
<line>     = <tag> " " { <literal> " " } "0"
           | "s"   " " <status>
<tag>      = "i" | "q" | "u" | "m"
<status>   = "SATISFIABLE"
           | "UNSATISFIABLE"
           | "UNKNOWN"
<literal>  = <pos> | <neg>
<pos>      = "1" | "2" | ... | <INT_MAX>
<neg>      = "-" <pos>

<idrup> = <comments> "p idrup\n" <lines>
...
<tag> =    "i" | "q" | "u" | "m"
         | "l" | "d" | "w" | "r"
...
```

Syntax & Semantics of the New Formats

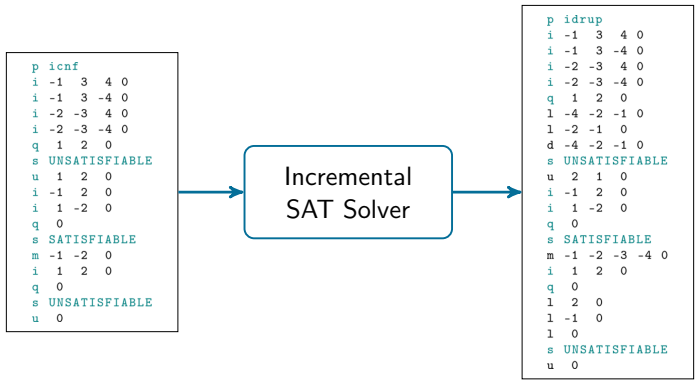
```
<icnf>     = <comments> "p icnf\n" <lines>
<comments> = { <comment> "\n" }
<lines>    = { <comment> "\n" | <line> "\n" }
<comment>  = "c"    " " <anything-but-new-line>
<line>     = <tag> " " { <literal> " " } "0"
           | "s"    " " <status>
<tag>      = "i" | "q" | "u" | "m"
<status>   = "SATISFIABLE"
           | "UNSATISFIABLE"
           | "UNKNOWN"
<literal>  = <pos> | <neg>
<pos>      = "1" | "2" | ... | <INT_MAX>
<neg>      = "-" <pos>

<idrup> = <comments> "p idrup\n" <lines>
...
<tag> =     "i" | "q" | "u" | "m"
          | "l" | "d" | "w" | "r"
...
```

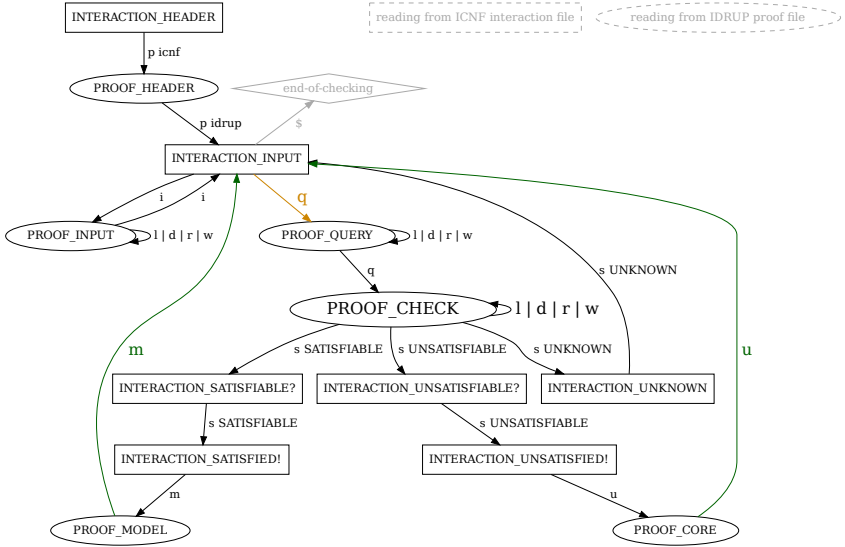
- F_A, F_P: *active* and *passive* clauses

$$(F_A^{i+1}, F_P^{i+1}) = \begin{cases} (F_A^i \cup \{L_i\}, F_P^i) & \text{if } t(L_i) = \texttt{"i"} \\ (F_A^i \cup \{L_i\}, F_P^i) & \text{if } t(L_i) = \texttt{"l"} \\ (F_A^i \setminus \{L_i\}, F_P^i) & \text{if } t(L_i) = \texttt{"d"} \\ (F_A^i \setminus \{L_i\}, F_P^i \cup \{L_i\}) & \text{if } t(L_i) = \texttt{"w"} \\ (F_A^i \cup \{L_i\}, F_P^i \setminus \{L_i\}) & \text{if } t(L_i) = \texttt{"r"} \\ (F_A^i, F_P^i) & \text{otherwise.} \end{cases}$$

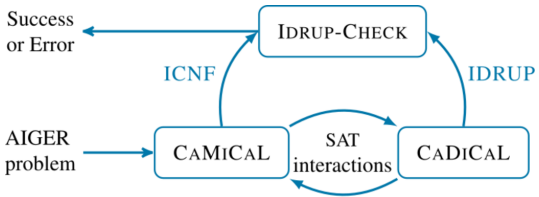
ICNF & IDRUP Example



IDRUP-CHECK – Checking Incremental Proofs

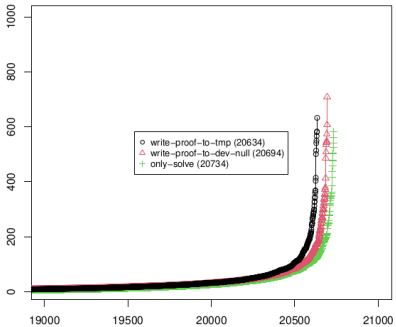


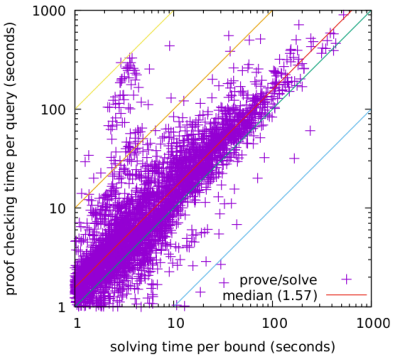
Experiments



- Hardware Model Checking Competition Benchmark set (2017), 300 instances
- Limits: 16 GB memory, 1000 second, maximum bound: k = 100
  - $\rightarrow$ at most 101 incremental SAT query for each instance

Results





- Very small overhead of proof writing
- Reasonable proof checking time ($\sim$ 2x)

Outline

Preliminaries & Motivation

Our Contributions

Conclusion

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of SAT-based Model Checkers**

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of SAT-based Model Checkers**
- `IDRUP-CHECK`: First prototype to check IDRUP proofs
  - First incremental proof checker

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of SAT-based Model Checkers**
- `IDRUP-CHECK`: First prototype to check IDRUP proofs
  - First incremental proof checker
- Promising preliminary results in HW model checker CaMiCaL

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of SAT-based Model Checkers**
- `IDRUP-CHECK`: First prototype to check IDRUP proofs
  - First incremental proof checker
- Promising preliminary results in HW model checker CaMiCaL
- Future Work:
  - more evaluation (e.g. in IC3)
  - backward proof checking, trimming
  - verify proof checker
  - incremental LRAT

Conclusion & Future Work

- Standardize input and proof format for incremental use cases of SAT solvers.
  - → Gain verifiable results
  - → **Increase trustworthiness of SAT-based Model Checkers**
- `IDRUP-CHECK`: First prototype to check IDRUP proofs
  - First incremental proof checker
- Promising preliminary results in HW model checker CaMiCaL
- Future Work:
  - more evaluation (e.g. in IC3)
  - backward proof checking, trimming
  - verify proof checker
  - incremental LRAT

Thank you!