Knowledge Compilation for Incremental and Checkable Stochastic Boolean Satisfiability

Che Cheng [1*] Yun-Rong Lauren Luo [2*] Jie-Hong Roland Jiang [1]

[1] National Taiwan University (NTU), Taipei, Taiwan

[2] University of Michigan, Ann Arbor, MI, USA

[*] Contributed equally

The 33rd International Joint Conference on Artificial Intelligence

Motivation 1: Knowledge Compilation for SSAT

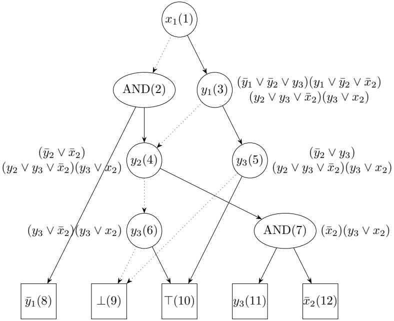
SSAT: $Q_1 v_1, \ldots, Q_n v_n . \phi$, $Q_i \in \{\rotatebox[origin=c]{180}{R}^p, \exists\}$, where $\rotatebox[origin=c]{180}{R}^p\, r$ denotes that $\Pr[r = \top] = p$

- Randomized variant of QBF
- Semantics: maximum satisfying probability $\Pr[\Phi]$

Motivation 1: Knowledge Compilation for SSAT

SSAT: $Q_1 v_1, \ldots, Q_n v_n . \phi$, $Q_i \in \{ᴚ^p, \exists\}$, where $ᴚ^p\, r$ denotes that $\Pr[r = \top] = p$

- Randomized variant of QBF
- Semantics: maximum satisfying probability $\Pr[\Phi]$



$x_1(1)$
AND(2)
$y_1(3)$
$(\bar{y}_1 \vee \bar{y}_2 \vee y_3)(y_1 \vee \bar{y}_2 \vee \bar{x}_2)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$(\bar{y}_2 \vee \bar{x}_2)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$y_2(4)$
$y_3(5)$
$(\bar{y}_2 \vee y_3)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$(y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$y_3(6)$
AND(7)
$(\bar{x}_2)(y_3 \vee x_2)$
$\bar{y}_1(8)$
$\bot(9)$
$\top(10)$
$y_3(11)$
$\bar{x}_2(12)$

- Observation: the trace of a run of the SSAT solver **SharpSSAT** [FJ23] is a *dec-DNNF* [DM02].

$\Longrightarrow$ Why not compile the dec-DNNF and use it?

Motivation 2: Incremental and Checkable SSAT

- Different but similiar SSAT formulas may result in an identical **SharpSSAT** trace

  $\Longrightarrow$ let **SharpSSAT** re-use the trace and avoid repeated searches

  $\Longrightarrow$ *Incremental SSAT*

Motivation 2: Incremental and Checkable SSAT

- Different but similiar SSAT formulas may result in an identical **SharpSSAT** trace

  $\Longrightarrow$ let **SharpSSAT** re-use the trace and avoid repeated searches

  $\Longrightarrow$ *Incremental SSAT*

- The trace is the footprint of a **SharpSSAT** run

  $\Longrightarrow$ a proof/certificate for the run that can be independently checked

  $\Longrightarrow$ *Checkable SSAT*

Contribution 1: Knowledge Compilation for SSAT

- Lift dec-DNNF compilation to SSAT:

WMC

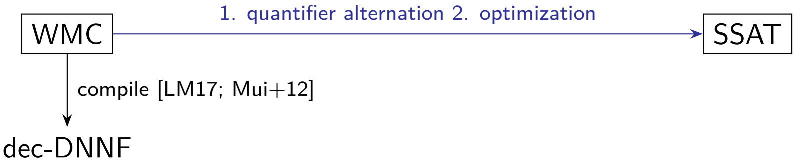
compile [LM17; Mui+12]

dec-DNNF

SSAT

Contribution 1: Knowledge Compilation for SSAT

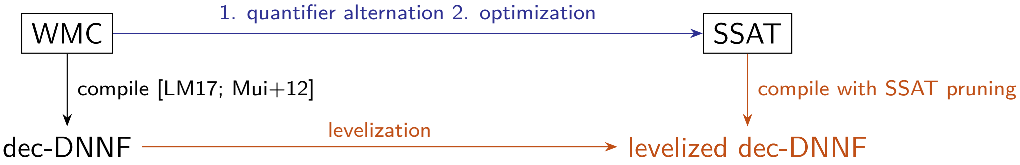
- Lift dec-DNNF compilation to SSAT:
  1. decision orders should follow the quantifier alternation levels in SSAT prefix.
  2. SSAT decision pruning may occur in SSAT solving due to the optimization nature of SSAT



Contribution 1: Knowledge Compilation for SSAT

- Lift dec-DNNF compilation to SSAT:
  1. decision orders should follow the quantifier alternation levels in SSAT prefix.
  2. SSAT decision pruning may occur in SSAT solving due to the optimization nature of SSAT

- Contribution: propose *levelized dec-DNNF* and compilation with SSAT pruning.



WMC

1. quantifier alternation 2. optimization

SSAT

compile [LM17; Mui+12]

compile with SSAT pruning

dec-DNNF

levelization

levelized dec-DNNF

Contribution 1A: Levelized Dec-DNNF

SSAT formula: $\Phi = \reflectbox{R}^{0.4}\, x_1, \exists\, y_1, \exists\, y_2, \exists\, y_3, \reflectbox{R}^{0.6}\, x_2 . \phi$

quantifier alternation levels: $x_1 \prec y_1 = y_2 = y_3 \prec x_2$

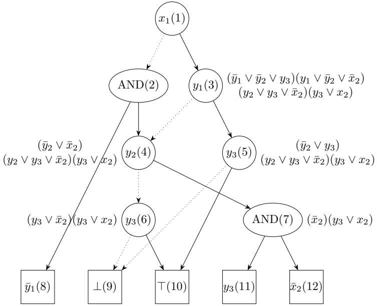

$x_1(1)$
AND(2)
$y_1(3)$
$(\bar{y}_1 \vee \bar{y}_2 \vee y_3)(y_1 \vee \bar{y}_2 \vee \bar{x}_2)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$(\bar{y}_2 \vee \bar{x}_2)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$y_2(4)$
$y_3(5)$
$(\bar{y}_2 \vee y_3)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$(y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$y_3(6)$
AND(7)
$(\bar{x}_2)(y_3 \vee x_2)$
$\bar{y}_1(8)$
$\bot(9)$
$\top(10)$
$y_3(11)$
$\bar{x}_2(12)$

Figure: Levelized dec-DNNF G for Φ.

Definition (Levelized Dec-DNNF)

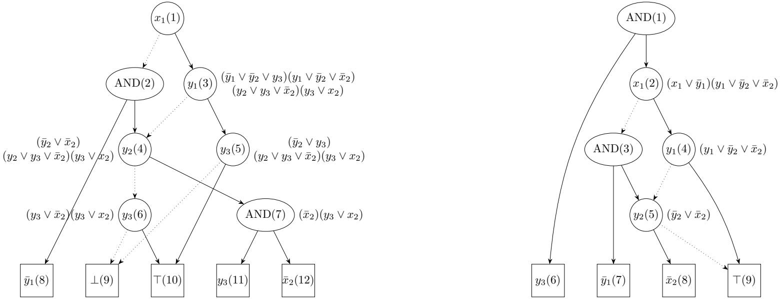
A dec-DNNF G s.t. for decision nodes $N_1 \leq N_2$ in G, their decision variables satisfy $v_1 \preceq v_2$

Theorem (SSAT Evaluation)

$\Pr[\Phi]$ *can be evaluated with* G *in one reversed topological traversal.*

Contribution 1B: Compilation with SSAT Pruning

Incorporate *all* pruning techniques[1] in the SSAT solver **SharpSSAT** [FJ23]
$\implies$ a more compact levelized dec-DNNF[2] with the same SSAT evaluation result.



$x_1(1)$
AND(2)
$y_1(3)$
$(\bar{y}_1 \vee \bar{y}_2 \vee y_3)(y_1 \vee \bar{y}_2 \vee \bar{x}_2)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$(\bar{y}_2 \vee \bar{x}_2)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$y_2(4)$
$y_3(5)$
$(\bar{y}_2 \vee y_3)$
$(y_2 \vee y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$(y_3 \vee \bar{x}_2)(y_3 \vee x_2)$
$y_3(6)$
AND(7)
$(\bar{x}_2)(y_3 \vee x_2)$
$\bar{y}_1(8)$
$\bot(9)$
$\top(10)$
$y_3(11)$
$\bar{x}_2(12)$
AND(1)
$x_1(2)$
$(x_1 \vee \bar{y}_1)(y_1 \vee \bar{y}_2 \vee \bar{x}_2)$
AND(3)
$y_1(4)$
$(y_1 \vee \bar{y}_2 \vee \bar{x}_2)$
$y_2(5)$
$(\bar{y}_2 \vee \bar{x}_2)$
$y_3(6)$
$\bar{y}_1(7)$
$\bar{x}_2(8)$
$\top(9)$

(a) Without pure literal detection.

(b) With pure literal detection.

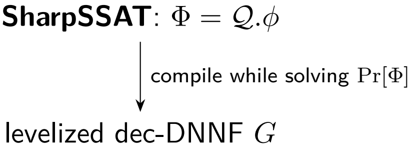
Figure: Levelized dec-DNNF graphs.

[1] pure literal detection and existential early return

[2] Whenever pruning occurs at a decision node, replace each unexplored node with the constant node $\bot$ or the other explored node.

Contribution 2: Incremental SSAT Evaluation with Levelized Dec-DNNF

- **SharpSSAT** compiles the SSAT $\Phi = \mathcal{Q}.\phi$ into a levelized dec-DNNF G while solving $\Pr[\Phi]$



SharpSSAT: $\Phi = \mathcal{Q}.\phi$

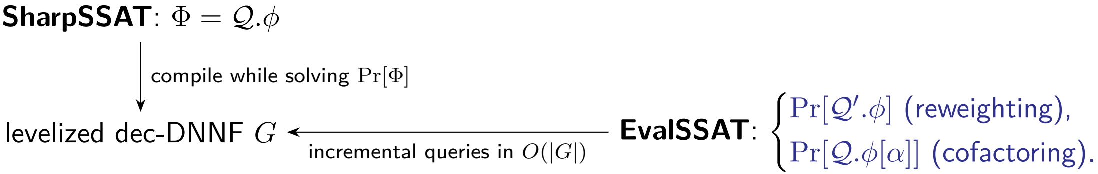
compile while solving $\Pr[\Phi]$

levelized dec-DNNF G

[3] Cofactoring cannot be correctly computed with SSAT pruning enabled.

Contribution 2: Incremental SSAT Evaluation with Levelized Dec-DNNF

- **SharpSSAT** compiles the SSAT $\Phi = \mathcal{Q}.\phi$ into a levelized dec-DNNF G while solving $\Pr[\Phi]$
- **EvalSSAT** performs linear incremental queries on G for maximum satisfying probability of:
  - a reweighting $\mathcal{Q}.\phi \mapsto \mathcal{Q}'.\phi$: differ from Φ in the probabilities of the randomized variables
  - a *cofactoring*[3] $\mathcal{Q}.\phi \mapsto \mathcal{Q}.\phi[\alpha]$: variables in the matrix are substituted with Boolean constants



SharpSSAT: $\Phi = \mathcal{Q}.\phi$

compile while solving $\Pr[\Phi]$

levelized dec-DNNF G

incremental queries in $O(|G|)$

EvalSSAT: $\begin{cases} \Pr[\mathcal{Q}'.\phi] \text{ (reweighting)}, \\ \Pr[\mathcal{Q}.\phi[\alpha]] \text{ (cofactoring)}. \end{cases}$

[3] Cofactoring cannot be correctly computed with SSAT pruning enabled.

Contribution 3: SSAT Validation with Levelized Dec-DNNF

- We develop an SSAT proof framework **cert-SSAT** based on the model counting proof framework **CPOG** [Bry+23] to validate the correctness of $\Pr[\Phi]$ reported by **SharpSSAT**.
- **cert-SSAT** validates $\Pr[\Phi]$ by sandwiching: $\mathsf{LB}(\Pr[\Phi]) = \Pr[\Phi] = \mathsf{UB}(\Pr[\Phi])$[4]

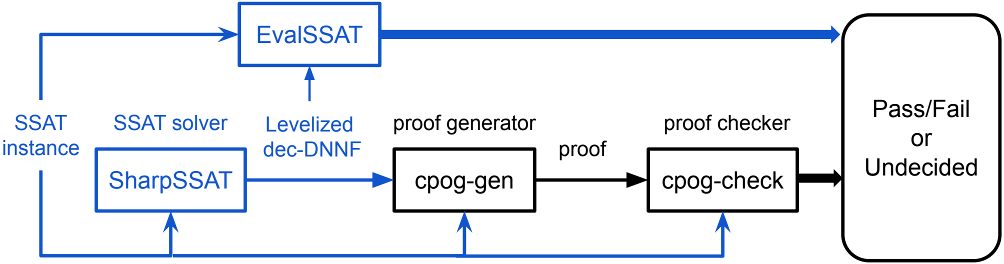


Figure: Toolchain flow for **cert-SSAT**.

[4] (1) Compiles two levelized dec-DNNFs G_l, G_u from SSAT $\Phi = \mathcal{Q}.\phi$; (2) Proves $(G_l \rightarrow \phi) \wedge (\phi \rightarrow G_u)$; (3) Proves **EvalSSAT**$(\Phi, G_l) = \Pr[\Phi] =$ **EvalSSAT**(Φ, G_u)

Experimental Results 1: Incremental SSAT Evaluation

KC: solving $\mathrm{Pr}[\Phi]$ incrementally with knowledge compilation[5]

Baseline: solving $\mathrm{Pr}[\Phi]$ for each individual query from scratch[6]

[5] Time limit: 1000 sec for compilation and 100 sec for each query

[6] Time limit: 200 sec for each query

[7] SSAT benchmark: https://github.com/NTU-ALComLab/ClauSSat

Experimental Results 1: Incremental SSAT Evaluation

KC: solving $\Pr[\Phi]$ incrementally with knowledge compilation[5]

Baseline: solving $\Pr[\Phi]$ for each individual query from scratch[6]

1. Reweighting task: for each SSAT formula[7], randomly generate 10 reweighting queries
   - # instances solved: **KC: 2220** $>$ Baseline: 2212
   - PAR2: **KC: 154.22** $<$ Baseline: 157.36
2. Cofactoring task: for each SSAT formula, randomly generate 10 cofactoring queries
   - # instances solved: **Baseline: 2361** $>$ KC: 1870
   - PAR2: **Baseline: 140.03** $<$ KC: 193.71

[5] Time limit: 1000 sec for compilation and 100 sec for each query

[6] Time limit: 200 sec for each query

[7] SSAT benchmark: https://github.com/NTU-ALComLab/ClauSSat

Experimental Results 2: SSAT Validation

- Out of 236 solvable SSAT instances for **SharpSSAT**[8], **cert-SSAT**[9] validates LB($\mathrm{Pr}[\Phi]$) for 205 instances (86.9%) and validates $\mathrm{Pr}[\Phi]$ for 190 instances (80.5 %)[10]

[8] SSAT benchmark: https://github.com/NTU-ALComLab/ClauSSat

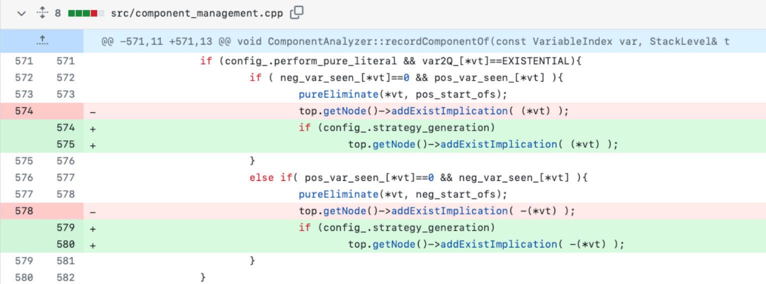
[9] Time limit: **SharpSSAT/EvalSSAT:** 1000 sec, **cpog-gen/cpog-check:** 2500 sec

[10] Validating lower and upper bounds are independent and validating a lower bound is easier.

Experimental Results 2: SSAT Validation

- Out of 236 solvable SSAT instances for **SharpSSAT**[8], **cert-SSAT**[9] validates $\mathrm{LB}(\Pr[\Phi])$ for 205 instances (86.9%) and validates $\Pr[\Phi]$ for 190 instances (80.5 %)[10]

- **cert-SSAT** helps discover one bug of **SharpSSAT**!



```
8 src/component_management.cpp

@@ -571,11 +571,13 @@ void ComponentAnalyzer::recordComponentOf(const VariableIndex var, StackLevel& t
                 if (config_.perform_pure_literal && var2Q_[*vt]==EXISTENTIAL){
                         if ( neg_var_seen_[*vt]==0 && pos_var_seen_[*vt] ){
                                 pureEliminate(*vt, pos_start_ofs);
-                                top.getNode()->addExistImplication( (*vt) );
+                                if (config_.strategy_generation)
+                                        top.getNode()->addExistImplication( (*vt) );
                         }
                         else if( pos_var_seen_[*vt]==0 && neg_var_seen_[*vt] ){
                                 pureEliminate(*vt, neg_start_ofs);
-                                top.getNode()->addExistImplication( -(*vt) );
+                                if (config_.strategy_generation)
+                                        top.getNode()->addExistImplication( -(*vt) );
                         }
                 }
```

[8] SSAT benchmark: https://github.com/NTU-ALComLab/ClauSSat

[9] Time limit: **SharpSSAT/EvalSSAT:** 1000 sec, **cpog-gen/cpog-check:** 2500 sec

[10] Validating lower and upper bounds are independent and validating a lower bound is easier.

Conclusions

- Contributions:
  - Proposed a dec-DNNF-based knowledge compilation technique for SSAT
  - Proposed incremental and checkable SSAT solving scenarios and demonstrated their effectiveness

Conclusions

- Contributions:
  - Proposed a dec-DNNF-based knowledge compilation technique for SSAT
  - Proposed incremental and checkable SSAT solving scenarios and demonstrated their effectiveness
- Future directions:
  - More applications for levelized dec-DNNF compilers
  - More types of incremental SSAT queries
  - Extend **cert-SSAT** to support SSAT preprocessor and DSSAT solver [CJ23]

References I

[Bry+23] Randal E. Bryant et al. "Certified Knowledge Compilation with Application to Verified Model Counting". In: *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*. Vol. 271. 2023, 6:1–6:20.

[CJ23] Che Cheng and Jie-Hong R. Jiang. "Lifting (D)QBF preprocessing and solving techniques to (D)SSAT". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 2023, pp. 3906–3914.

[DM02] Adnan Darwiche and Pierre Marquis. "A Knowledge Compilation Map". In: *Journal of Artificial Intelligence Research* 17.1 (2002), pp. 229–264.

[FJ23] Yu-Wei Fan and Jie-Hong R. Jiang. "SharpSSAT: A Witness-Generating Stochastic Boolean Satisfiability Solver". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 2023, pp. 3949–3958.

[LM17] Jean-Marie Lagniez and Pierre Marquis. "An Improved Decision-DNNF Compiler". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. AAAI Press, 2017, pp. 667–673.

[Mui+12] Christian Muise et al. "Dsharp: Fast d-DNNF Compilation with SharpSAT". In: *Proceedings of the Canadian Conference on Advances in Artificial Intelligence*. Springer-Verlag, 2012, pp. 356–361.