An Efficient Two-phase Method for Prime Compilation of Non-clausal Boolean Formulae

Weilin Luo[1], Hai Wan [1,*], Hongzhen Zhong [1], Ou Wei [2],
Biqing Fang [1] Xiaotong Song [1]

[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
[2] Department of Computer Science, University of Toronto, Toronto, Canada

Content

Content

Definition of Problem

Non-clausal Boolean Formulae:

- arbitrary form, *e.g.*, $a \wedge (c \vee b \wedge \neg a)$
- encoding methods, *e.g.*, Tseitin encoding[1]

Definition of Problem

Non-clausal Boolean Formulae:

- arbitrary form, *e.g.*, $a \wedge (c \vee b \wedge \neg a)$
- encoding methods, *e.g.*, Tseitin encoding[1]

Implicate and Prime Implicate:

- A clause λ is called an *implicate* of φ if $\varphi \models \lambda$.
- An implicate λ of φ is called *prime* if any subset $\lambda' \subsetneq \lambda$ is not an implicate of φ.

Implicant and Prime Implicant:

- A term κ is called an *implicant* of φ if $\kappa \models \varphi$.
- An implicant κ of φ is called *prime* if any subset $\kappa' \subsetneq \kappa$ is not an implicant of φ.

Definition of Problem

Prime Compilation:

- generate *all* prime implicates/implicants of a Boolean formula

Example 1 (Prime Compilation of Non-clausal Formulae)

Let $\varphi = (a \wedge b) \vee (\neg a \wedge c)$ be a non-clausal Boolean formula.

- $\neg a \wedge c$
- $a \wedge b$
- $b \wedge c$

c \ a b	0 0	0 1	1 1	1 0
0	0	0	1	0
1	1	1	1	0

Significance and Challenge

Significance:

- complete analysis for the safety-critical systems[2,3]
- widely applications, *e.g.*, knowledge compilation[4], logic minimization[5,6], digital circuit analysis and optimization[7], logic synthesis[8,9], model checking[10], and fault tree analysis[11]

Significance and Challenge

Significance:

- complete analysis for the safety-critical systems [2,3]
- widely applications, *e.g.*, knowledge compilation [4], logic minimization [5,6], digital circuit analysis and optimization [7], logic synthesis [8,9], model checking [10], and fault tree analysis [11]

Challenge:

- overlap of prime implicates/implicants
- hard for Σ_p^2 [12]

	ab	0 0	0 1	1 1	1 0
c	0	0	0	1	0
	1	1	1	1	0

Significance and Challenge

Significance:

- complete analysis for the safety-critical systems [2,3]
- widely applications, *e.g.*, knowledge compilation [4], logic minimization [5,6], digital circuit analysis and optimization [7], logic synthesis [8,9], model checking [10], and fault tree analysis [11]

Challenge:

- overlap of prime implicates/implicants
- hard for Σ_p^2 [12]

c \ a b	0 0	0 1	1 1	1 0
0	0	0	1	0
1	1	1	1	0

Developing new approaches over different paradigms remains to be an interesting research direction.

State-of-the-art Approache

The state-of-the-art (SOTA) approach [13] for compiling non-clausal formulae:

- construct a *minimal* cover represented as a set of *prime* implicates and find all prime implicants simultaneously.
- extract prime implicant from a model using QuickXplain [10] (QX).

State-of-the-art Approache

The state-of-the-art (SOTA) approach [13] for compiling non-clausal formulae:

- construct a *minimal* cover represented as a set of *prime* implicates and find all prime implicants simultaneously.
- extract prime implicant from a model using QuickXplain [10] (QX).

Advantage:

1. dual rail (DR) encoding [14] to handle overlap
   - x_v represents the literal v; $x_{\neg v}$ represents the literal $\neg v$, *e.g.*, $x_a \wedge \neg x_{\neg a}$ means $a =$ true.
2. SAT solver to improve the performance

State-of-the-art Approache

Shortcoming:

1. DR encoding needs a double number of variables.

State-of-the-art Approache

Shortcoming:

1. DR encoding needs a double number of variables.
2. It it time-consuming to construct a minimal cover.

State-of-the-art Approache

Shortcoming:

1. DR encoding needs a double number of variables.
2. It it time-consuming to construct a minimal cover.
3. To ensure correction, they require minimal or maximal models in SAT solving.

State-of-the-art Approache

Shortcoming:

1. DR encoding needs a double number of variables.
2. It it time-consuming to construct a minimal cover.
3. To ensure correction, they require minimal or maximal models in SAT solving.

We argue that:

- DR encoding is not necessary for constructing a cover.
- It not necessary to compute the minimal cover.

Content

Cover of Boolean Formulae

Definition 1

A Boolean formula Φ is a *cover* of φ if Φ is a conjunction of implicates of φ and is logically equivalent to φ. The *size* of a cover Φ, denoted by $|\Phi|$, is the sum of the size of implicates in Φ. A cover is *minimal* if all the implicates of the cover are prime.

Cover of Boolean Formulae

Definition 1

A Boolean formula Φ is a *cover* of φ if Φ is a conjunction of implicates of φ and is logically equivalent to φ. The *size* of a cover Φ, denoted by $|\Phi|$, is the sum of the size of implicates in Φ. A cover is *minimal* if all the implicates of the cover are prime.

- In order to generate *all* prime implicants of φ, it is necessary to compute a cover of φ under the DR encoding[13].
- Naturally, constructing a cover independently need not use DR encoding, which avoids enlarging the search space.
- Moreover, we can exploit the powerful polarity heuristic method for SAT solving in the first phase.

Two-phase Framework: CoAPI

Let Σ_φ be a CNF of a non-clausal Boolean formula φ, $\Sigma_{\neg\varphi}$ its negation, and $\mathcal{V}_o$ a set of original variables.

- CoAPI takes Σ_φ, $\Sigma_{\neg\varphi}$, and $\mathcal{V}_o$ as input, which is the same as the approach[13].
- CoAPI returns all prime implicants Π of φ.

Two-phase Framework: CoAPI

Let Σ_φ be a CNF of a non-clausal Boolean formula φ, $\Sigma_{\neg\varphi}$ its negation, and $\mathcal{V}_o$ a set of original variables.

- CoAPI takes Σ_φ, $\Sigma_{\neg\varphi}$, and $\mathcal{V}_o$ as input, which is the same as the approach[13].
- CoAPI returns all prime implicants Π of φ.

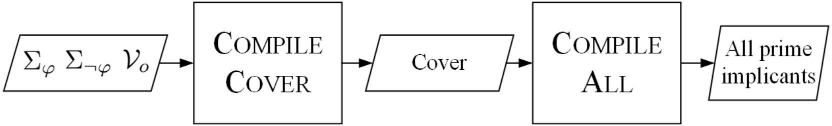

Figure 1: Framework of CoAPI.

CompileCover

We compute small implicants of $\neg\phi$ to construct a succinct cover of ϕ.

- well-known blocking clause framework
- two additional constraints as follows:
  1. extract implicates of φ
  2. small size of the implicates

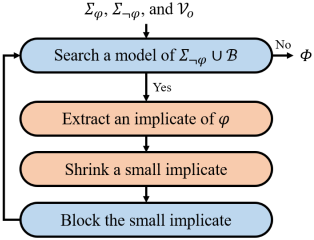

Figure 2: Algorithm of CompileCover.

CompileCover

We compute small implicants of $\neg\phi$ to construct a succinct cover of ϕ.

- well-known blocking clause framework
- two additional constraints as follows:
  1. extract implicates of φ
  2. small size of the implicates

Example 2 (Running for $\varphi = (a \wedge b) \vee (\neg a \wedge c)$)

1^{st} iteration:

1. model
   $\neg g_1 \wedge \neg g_2 \wedge \neg g_3 \wedge \neg a \wedge b \wedge \neg c$
2. implicate $a \vee \neg b \vee c$
3. small implicate $a \vee c$

c \ a b	0 0	0 1	1 1	1 0
0	0	0	1	0
1	1	1	1	0

Finally, a cover $\Phi = (a \vee c) \wedge (\neg a \vee b)$.

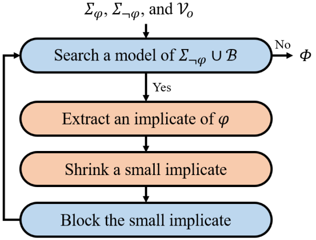

Figure 2: Algorithm of CompileCover.

CompileAll

We reformulate the prime implicants generation as the minimal model enumeration.

- similar to the work [15]
  1. limitation: a large number of the constraints
- minimal models like the work [13]

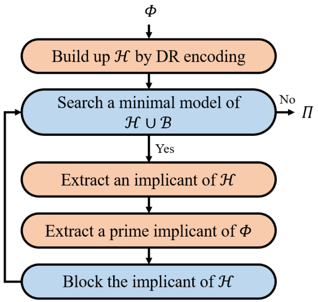

Figure 3: Algorithm of CompileAll.

CompileAll

We reformulate the prime implicants generation as the minimal model enumeration.

- similar to the work [15]
  1. limitation: a large number of the constraints
- minimal models like the work [13]

Example 3 (Running for $\Phi = (a \vee c) \wedge (\neg a \vee b)$)

DR encoding:
$\mathcal{H} = (x_a \vee x_c) \wedge (x_{\neg a} \vee x_b) \wedge (\neg x_a \vee \neg x_{\neg a}) \wedge (\neg x_b \vee \neg x_{\neg b}) \wedge (\neg x_c \vee \neg x_{\neg c})$
1^{st} iteration:

1. minimal model $\neg x_a \wedge x_{\neg a} \wedge \neg x_b \wedge \neg x_{\neg b} \wedge x_c \wedge \neg x_{\neg c}$
2. implicant of $\mathcal{H}$ $x_{\neg a} \wedge x_c$
3. prime implicant of Φ $\neg a \wedge c$

Finally, $\Pi = \{\neg a \wedge c, b \wedge c, a \wedge b\}$.

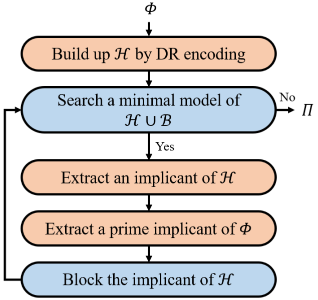

Figure 3: Algorithm of CompileAll.

Content

Small Implicants Extraction

Definition 2

Given an implicant π of $\neg\varphi$, $l \in \pi$ is *necessary* for π if it fulfills the requirement that $\bigwedge_{l_i \in \pi, l_i \neq l} l_i \wedge \Sigma_\varphi$ is satisfiable.

Small Implicants Extraction

Definition 2

Given an implicant π of $\neg\varphi$, $l \in \pi$ is *necessary* for π if it fulfills the requirement that $\bigwedge_{l_i \in \pi, l_i \neq l} l_i \wedge \Sigma_\varphi$ is satisfiable.

Remove all the unnecessary literals from a model.

- advantage: minimal size (a greatly succinct representation)
- shortcoming: time-consuming and not unclearly cost-effective in two phases
- linearly query the necessity (linear method) or QuickXplain [10] (QX)

Small Implicants Extraction

Definition 2

Given an implicant π of $\neg\varphi$, $l \in \pi$ is *necessary* for π if it fulfills the requirement that $\bigwedge_{l_i \in \pi, l_i \neq l} l_i \wedge \Sigma_\varphi$ is satisfiable.

Remove all the unnecessary literals from a model.

- advantage: minimal size (a greatly succinct representation)
- shortcoming: time-consuming and not unclearly cost-effective in two phases
- linearly query the necessity (linear method) or QuickXplain [10] (QX)

We design a bounded prime extraction method (BPE) to find a good trade-off between having an efficient algorithm and constructing a succinct cover.

Bounded Prime Extraction

We use sup and inf to dynamically measure the small size for different instances.

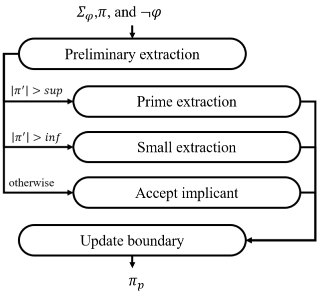

Figure 4: Algorithm of BPE.

Bounded Prime Extraction

We use sup and inf to dynamically measure the small size for different instances.

Preliminary extraction (PreE):

- extract a small implicant via SAT solver under failed assumptions[16].

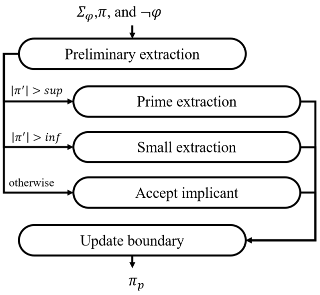

Figure 4: Algorithm of BPE.

Bounded Prime Extraction

We use sup and inf to dynamically measure the small size for different instances.

Preliminary extraction ($\textsc{PreE}$):

- extract a small implicant via SAT solver under failed assumptions[16].

Prime extraction (PE):

- reduce the size of the implicant or prove that a larger prime implicant exists.
- QX potentially requires more queries than the linear method.
- use the linear method, than using QX.

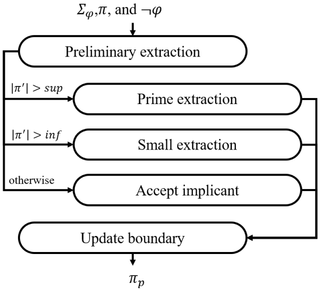

Figure 4: Algorithm of BPE.

Bounded Prime Extraction

We use sup and inf to dynamically measure the small size for different instances.

Preliminary extraction (PreE):

- extract a small implicant via SAT solver under failed assumptions[16].

Prime extraction (PE):

- reduce the size of the implicant or prove that a larger prime implicant exists.
- QX potentially requires more queries than the linear method.
- use the linear method, than using QX.

Small extraction (SE):

- IntervalExtract is like QX, while avoids discussing the case where none of the partial assignments are an implicant.

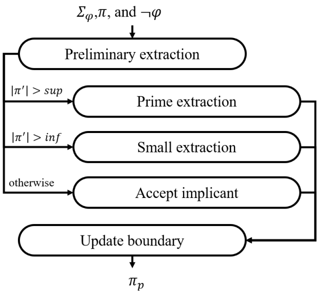

Figure 4: Algorithm of BPE.

Good Trade-off

Theorem 1

In BPE, π_p *is smaller than or equal in size to the largest prime implicant of* $\neg\varphi$.

Theorem 2

In BPE, *if* $|\pi'| > sup$, BPE *requires* $\mathcal{O}(|\pi'|)$ *SAT queries; if* $inf < |\pi'| \leqslant sup$, *requires* $\mathcal{O}(\lg \frac{|\pi'|}{|\pi_p|})$*; otherwise requires* $\mathcal{O}(1)$.

We show that BPE can achieve a good trade-off between having an efficient algorithm and constructing a succinct cover.

Content

Settings

Benchmarks

- classic benchmarks[13]: fairly large numbers of variables but without a large number of prime implicates/implicants
- industrial benchmark[11]: millions of prime implicants

Settings

Benchmarks

- classic benchmarks[13]: fairly large numbers of variables but without a large number of prime implicates/implicants
- industrial benchmark[11]: millions of prime implicants

Competitors

- `primer-a` and `primer-b`[13]: based on different search strategies
- variants of `CoAPI`
  - only to extract prime implicants
    - `CoAPI-pp`: $\textsc{PreE} + \textsc{PE}$
    - `CoAPI-qx`: QX
  - only to extract small implicants
    - `CoAPI-ps`: $\textsc{PreE} + \textsc{SE}$
    - `CoAPI-fa`: failed assumptions[17,18]
    - `CoAPI-idp`: interleaved dual propagation[19]

Settings

Benchmarks

- classic benchmarks[13]: fairly large numbers of variables but without a large number of prime implicates/implicants
- industrial benchmark[11]: millions of prime implicants

Competitors

- `primer-a` and `primer-b`[13]: based on different search strategies
- variants of `CoAPI`
  - only to extract prime implicants
    - `CoAPI-pp`: $\textsc{PreE} + \textrm{PE}$
    - `CoAPI-qx`: QX
  - only to extract small implicants
    - `CoAPI-ps`: $\textsc{PreE} + \textrm{SE}$
    - `CoAPI-fa`: failed assumptions[17,18]
    - `CoAPI-idp`: interleaved dual propagation[19]

Tasks

1. Generating prime implicates
2. Generating prime implicants

RQ1: What is the performance of `CoAPI` compared with the SOTA approaches for prime implication of non-clausal Boolean formulae?

Table 1: The number of instances computed (task (i) / task (ii)).

	QG6 (83)	*Geffe gen.* (600)	*F+PHP* (30)	*F+GT* (30)	*Total* (743)
`primer-a`	30 / 68	577 / **596**	30 / 30	28 / 30	665 / 724
`primer-b`	30 / 67	578 / **596**	30 / 30	28 / 30	666 / 723
`CoAPI-qx`	30 / 73	**589** / 593	30 / 30	26 / 30	675 / 726
`CoAPI`	30 / **82**	**589** / 595	30 / 30	**30** / 30	**679** / **737**

- `CoAPI-qx`, `primer-a`, and `primer-b` are comparable.
- `CoAPI` successfully computes more instances than `primer-a` and `primer-b`.
- Particularly, `CoAPI` increases the number of instances in *QG6* for the task (ii) and in *Geffe gen.* for the task (i).

RQ1: What is the performance of CoAPI compared with the SOTA approaches for prime implication of non-clausal Boolean formulae?

- `CoAPI` computes much faster than `primer-a` (*resp.* `primer-b`) in 664 (*resp.* 666) instances – it is at least one order of magnitude faster than `primer-a` (*resp.* `primer-b`) in 214 (*resp.* 215) instances.
- It is in 85% (*resp.* 86%) instances that `CoAPI-qx` beats `primer-a` (*resp.* `primer-b`).
- `CoAPI` is greatly better than `CoAPI-qx`.

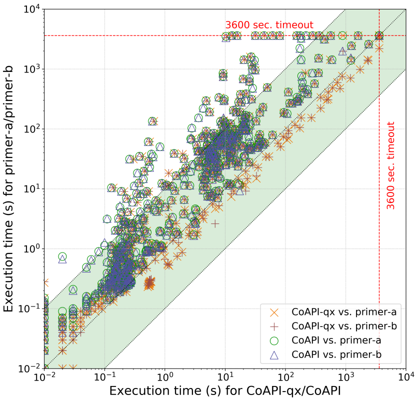

Figure 5: Generating prime implicates.

RQ1: What is the performance of `CoAPI` compared with the SOTA approaches for prime implication of non-clausal Boolean formulae?

- `CoAPI` dominates `primer-a` and `primer-b` on *QG6*, *F+PHP*, and *F+GT*.

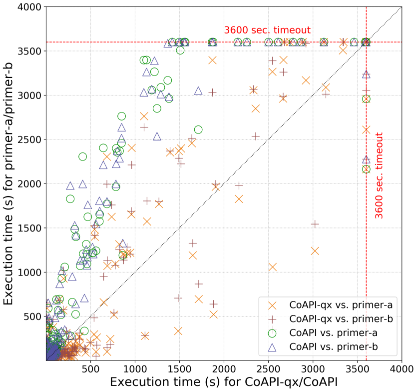

Figure 6: Generating prime implicates.

RQ1: What is the performance of `CoAPI` compared with the SOTA approaches for prime implication of non-clausal Boolean formulae?

- `CoAPI` dominates `primer-a` and `primer-b` on *QG6*, *F+PHP*, and *F+GT*.

Answer for RQ1:

- `CoAPI` achieves SOTA performances in most instances in classic benchmarks.
- The better performance of `CoAPI-qx` than the SOTA approaches confirms that the two-phase framework is efficient.

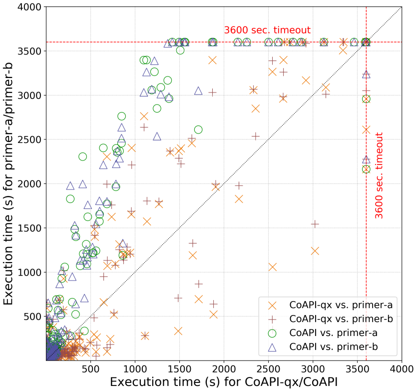

Figure 6: Generating prime implicates.

RQ2: Can CoAPI outperform the SOTA approaches in the real-world industrial benchmark?

- `CoAPI-qx` and `CoAPI` can successfully solve 16 instances which include all the 15 (*resp.* 15) instances successfully computed by `primer-a` (*resp.* `primer-b`).
- For most cases, `CoAPI` uses significantly less CPU time than `primer-a` and `primer-b`.

Table 2: CPU time (s) on the industrial benchmarks (omit all "N/A")

Instances	primer-a	primer-b	CoAPI-qx	CoAPI
chinese	0.02	0.01	**0.00**	**0.00**
das9201	164.95	163.68	145.60	**83.00**
das9202	62.69	60.21	28.68	**28.62**
das9203	0.40	0.33	**0.24**	**0.24**
das9204	2.18	**0.68**	0.70	0.83
das9205	0.25	0.29	**0.02**	0.03
das9206	256.08	331.06	**44.78**	72.56
das9208	5.96	6.35	6.06	**1.17**
edf9201	1,164.51	1,100.92	830.85	**742.67**
edf9205	86.82	**43.57**	46.68	53.16
edfpa15p	N/A	N/A	1,061.29	**1,012.33**
edfpa15r	2,755.73	2,872.39	1,087.08	**967.62**
ftr10	14.11	13.82	15.44	**1.24**
isp9603	570.37	565.98	568.08	**80.90**
isp9606	72.19	77.73	33.61	**13.69**
isp9607	452.26	540.61	273.99	**262.09**
#win	0	2	4	**12**

"N/A" indicates the results out of the CPU time limit. If the time usage is less than 0.01s, we mark it as 0.00.

RQ2: Can CoAPI outperform the SOTA approaches in the real-world industrial benchmark?

- CoAPI-qx and CoAPI can successfully solve 16 instances which include all the 15 (*resp.* 15) instances successfully computed by primer-a (*resp.* primer-b).
- For most cases, CoAPI uses significantly less CPU time than primer-a and primer-b.

Answer for RQ2:

- CoAPI has an excellent ability in generating all prime implicants on the industrial benchmark compared with the SOTA approaches.
- The better performance of CoAPI-qx confirms the advantage of the two-phase framework again.

Table 2: CPU time (s) on the industrial benchmarks (omit all "N/A")

Instances	primer-a	primer-b	CoAPI-qx	CoAPI
chinese	0.02	0.01	**0.00**	**0.00**
das9201	164.95	163.68	145.60	**83.00**
das9202	62.69	60.21	28.68	**28.62**
das9203	0.40	0.33	**0.24**	**0.24**
das9204	2.18	**0.68**	0.70	0.83
das9205	0.25	0.29	**0.02**	0.03
das9206	256.08	331.06	**44.78**	72.56
das9208	5.96	6.35	6.06	**1.17**
edf9201	1,164.51	1,100.92	830.85	**742.67**
edf9205	86.82	**43.57**	46.68	53.16
edfpa15p	N/A	N/A	1,061.29	**1,012.33**
edfpa15r	2,755.73	2,872.39	1,087.08	**967.62**
ftr10	14.11	13.82	15.44	**1.24**
isp9603	570.37	565.98	568.08	**80.90**
isp9606	72.19	77.73	33.61	**13.69**
isp9607	452.26	540.61	273.99	**262.09**
#win	0	2	4	**12**

"N/A" indicates the results out of the CPU time limit. If the time usage is less than 0.01s, we mark it as 0.00.

RQ3: Can BPE keep a good trade-off between having an efficient algorithm and constructing a succinct cover?

- `CoAPI` is faster in most instances.
- `CoAPI` constructs a more succinct cover than any other method that only extracts small implicants.
- Compared with the methods to only extract prime implicants, `CoAPI` requires fewer SAT queries, which speeds up in CompileCover.

Table 3: Results of extracting methods (task (i) / task (ii)).

	$\lvert\Sigma\rvert$[1]	#SAT[2]	T. in CC[3]	T. in CA[4]
CoAPI-pp	1.00 / 0.45	6.82 / 1.44	4.31 / 1.26	1.36 / 0.88
CoAPI-qx	1.00 / 0.33	11.38 / 1.57	4.50 / 2.27	0.96 / 1.18
CoAPI-ps	1.00 / 3.31	1.60 / 2.25	1.54 / 1.45	1.30 / 4.58
CoAPI-fa	1.00 / 93.62	0.96 / 114.19	1.73 / 700.09	1.36 / 5.73
CoAPI-idp	4.60 / 19525.32	0.75 / 609.71	2.86 / 121.83	1.24 / 7049.41

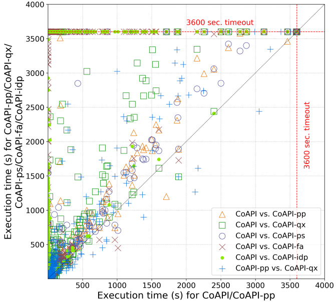

Figure 7: Generating prime implicates & prime implicants.

RQ3: Can BPE keep a good trade-off between having an efficient algorithm and constructing a succinct cover?

- `CoAPI` is faster in most instances.
- `CoAPI` constructs a more succinct cover than any other method that only extracts small implicants.
- Compared with the methods to only extract prime implicants, `CoAPI` requires fewer SAT queries, which speeds up in CompileCover.

Answer for RQ3:

- BPE provides a better trade-off between having an efficient algorithm and constructing a succinct cover.

Table 3: Results of extracting methods (task (i) / task (ii)).

	$\|\Sigma\|$[1]	#SAT[2]	T. in CC[3]	T. in CA[4]
`CoAPI-pp`	1.00 / <u>0.45</u>	6.82 / 1.44	4.31 / 1.26	1.36 / <u>0.88</u>
`CoAPI-qx`	1.00 / <u>0.33</u>	11.38 / 1.57	4.50 / 2.27	<u>0.96</u> / 1.18
`CoAPI-ps`	1.00 / 3.31	1.60 / 2.25	1.54 / 1.45	1.30 / 4.58
`CoAPI-fa`	1.00 / 93.62	<u>0.96</u> / 114.19	1.73 / 700.09	1.36 / 5.73
`CoAPI-idp`	4.60 / 19525.32	<u>0.75</u> / 609.71	2.86 / 121.83	1.24 / 7049.41

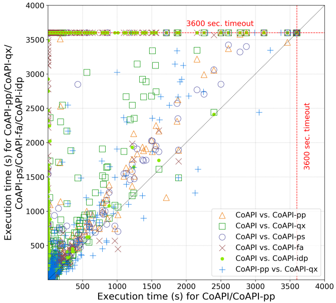

Figure 7: Generating prime implicates & prime implicants.

Content

Conclusion and Future Work

Conclusion:

1. For prime compilation of non-clausal Boolean formulae, we have proposed a two-phase approach – CoAPI, which allows us to construct a cover without using DR encoding and benefits from the polarity heuristic of SAT solving.
2. To improve the performance, we have designed a bounded prime extraction method (BPE) to dynamically extract small implicates or a prime implicates to construct a succinct cover.
3. Our experimental results show that CoAPI has dramatically pushed the limits of the performance of the state-of-the-art approaches on the classic and industrial benchmarks.
4. The results confirm that BPE provides a good trade-off between having an efficient algorithm and constructing a succinct cover.

Future work:

- We will develop optimization techniques to improve performance for the industrial benchmark.

References I

[1] G. S. Tseitin, "On the complexity of derivations in the propositional calculus," *SCMML*, pp. 234–259, 1983.

[2] S. Contini, G. Cojazzi, and G. Renda, "On the use of non-coherent fault trees in safety and security studies," *Reliab. Eng. Syst. Saf.*, vol. 93, no. 12, pp. 1886–1895, 2008.

[3] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools," *Comput. Sci. Rev.*, vol. 15, pp. 29–62, 2015.

[4] P. Marquis, "Knowledge compilation using theory prime implicates," in *IJCAI*, 1995, pp. 837–845.

[5] A. Ignatiev, A. Previti, and J. MarquesSilva, "Sat-based formula simplification," in *SAT*, 2015, pp. 287–298.

[6] J. Echavarria, S. Wildermann, and J. Teich, "Design space exploration of multi-output logic function approximations," in *ICCAD*, 2018, pp. 52–59.

[7] P. Tison, "Generalized consensus theory and applications to the minimization of boolean circuits," *IEEE Trans. on Computers*, vol. 16, no. 4, pp. 446–456, 1967.

[8] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *ICCAD*, 2013, pp. 779–786.

[9] A. Raghuvanshi and M. A. Perkowski, "Logic synthesis and a generalized notation for memristor-realized material implication gates," in *ICCAD*, 2014, pp. 470–477.

[10] A. R. Bradley and Z. Manna, "Checking safety by inductive generalization of counterexamples to induction," in *FMCAD*, 2007, pp. 173–180.

[11] W. Luo and O. Wei, "Wap: Sat-based computation of minimal cut sets," in *ISSRE*, 2017, pp. 146–151.

[12] P. Liberatore, "Redundancy in logic i: Cnf propositional formulae," *AIJ*, vol. 163, no. 2, pp. 203–232, 2005.

References II

[13] A. Previti, A. Ignatiev, A. Morgado, and J. Marques-Silva, "Prime compilation of non-clausal formulae," in *IJCAI*, 2015, pp. 1980–1987.

[14] R. E. Bryant, D. B., K. B., K. Cho, and T. Sheffler, "Cosmos: A compiled simulator for mos circuits," in *DAC*, 1987, pp. 9–16.

[15] S. Jabbour, J. Marques-Silva, L. Sais, and Y. Salhi, "Enumerating prime implicants of propositional formulae in conjunctive normal form," in *JELIA*, 2014, pp. 152–164.

[16] N. E. and N. Sörensson, "Temporal induction by incremental sat solving," *Electron. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.

[17] L. Zhang and S. Malik, "Searching for truth: Techniques for satisfiability of boolean formulas," Ph.D. dissertation, Princeton University Princeton, 2003.

[18] A. Niemetz, M. Preiner, and A. Biere, "Turbo-charging lemmas on demand with don't care reasoning," in *FMCAD*, 2014, pp. 179–186.

[19] A. Goultiaeva, M. Seidl, and A. Biere, "Bridging the gap between dual propagation and cnf-based qbf solving," in *DATE*, 2013, pp. 811–814.

Thank you for your listening!