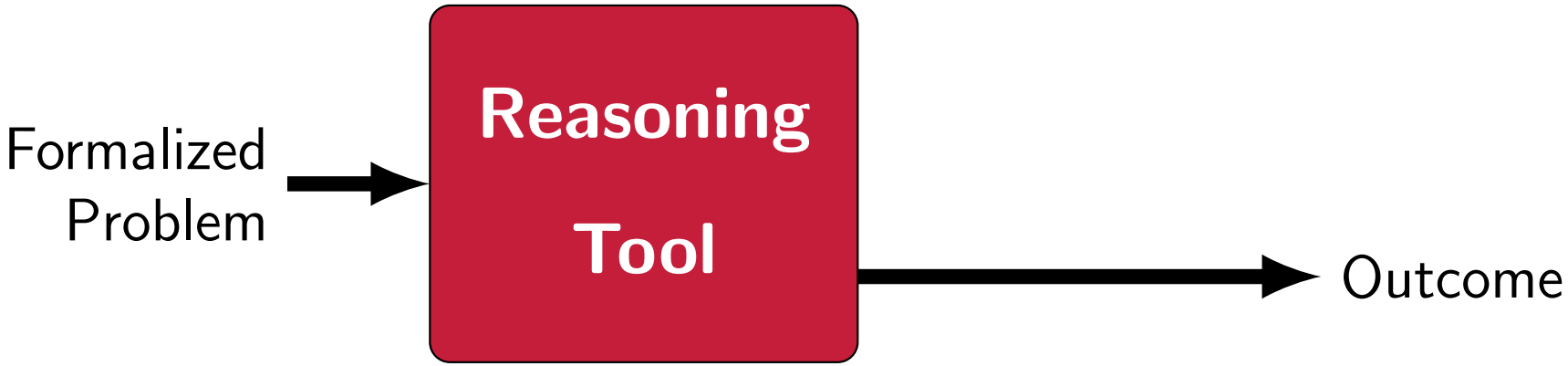
Certified Knowledge Compilation with Application to Verified Model Counting

Randal E. Bryant
Wojciech Nawrocki
Jeremy Avigad
Marijn J. H. Heule

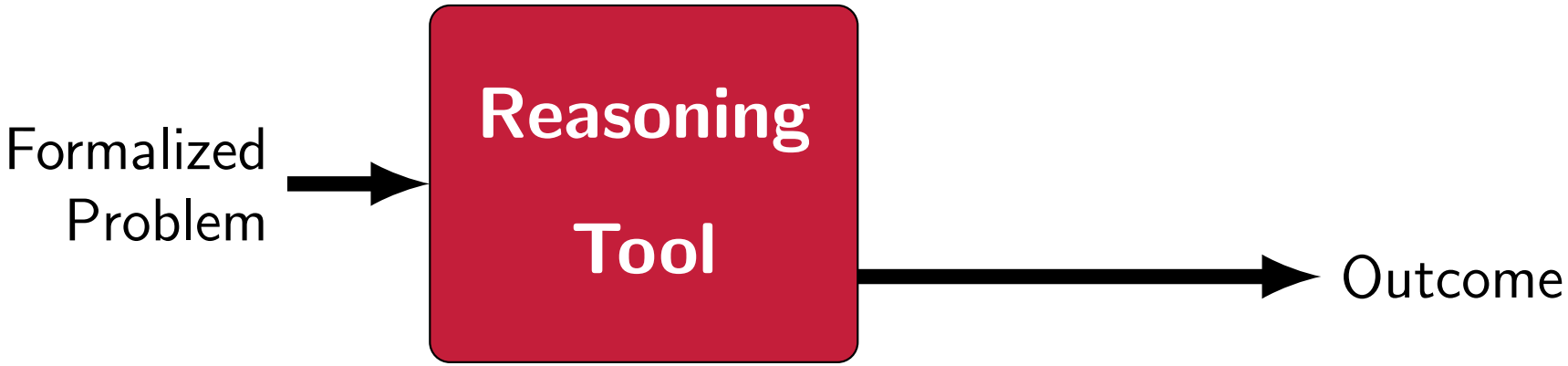
Carnegie Mellon University

SAT, 2023

Motivation: Automated Reasoning Programs



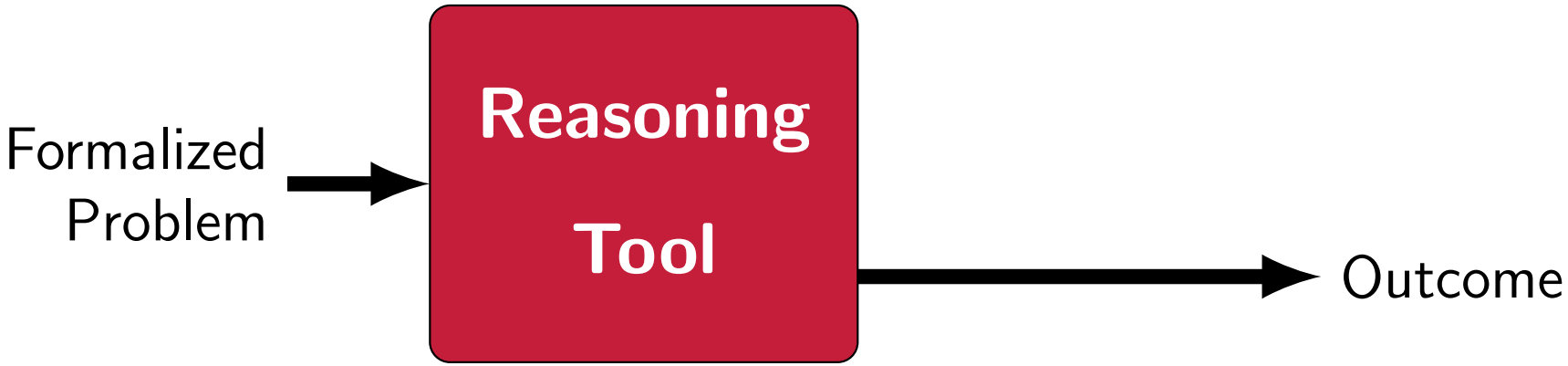
Motivation: Automated Reasoning Programs



Standard Tools

- Lingering doubt about whether result can be trusted
- If find bug in tool, must rerun all prior verifications

Motivation: Automated Reasoning Programs



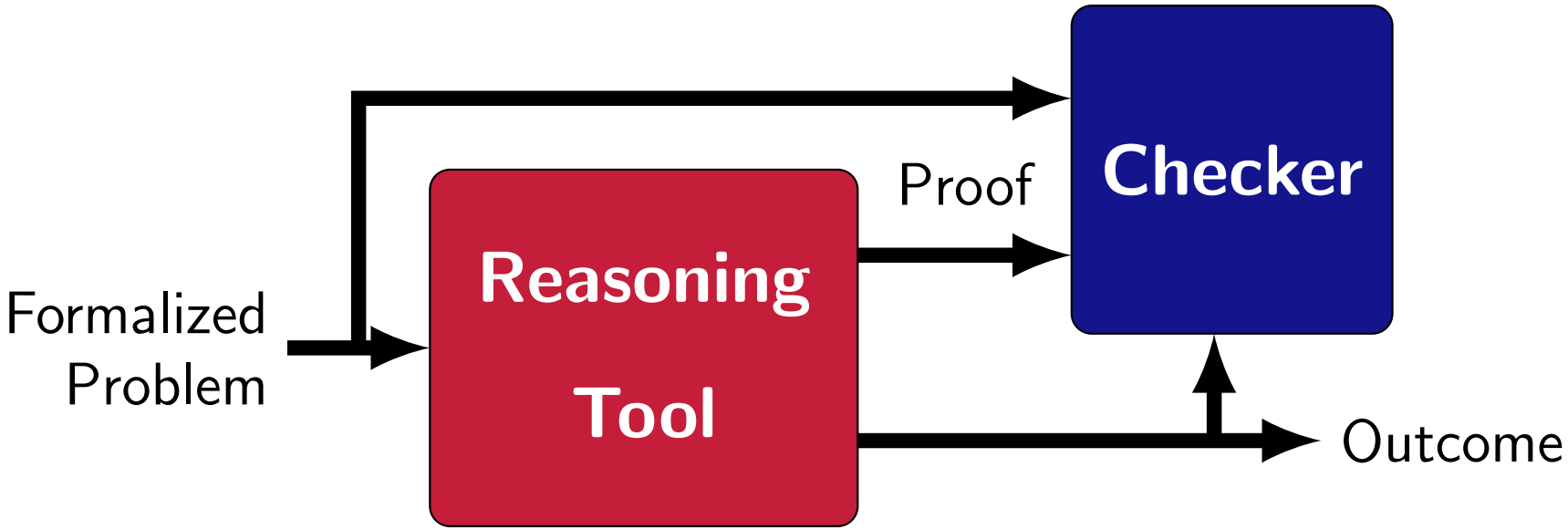
Standard Tools

- Lingering doubt about whether result can be trusted
- If find bug in tool, must rerun all prior verifications

Formally Verified Tools

- Hard to develop
- Hard to make scalable

Proof-Generating Automated Reasoning Programs



Proof-Generating Tools

- Verify individual executions, not entire program
- Can have bugs in tool but still trust result
- Can we trust the checker?

  Ideal: formally verified

Model Counting

Formula ϕ

$$[\overline{x}_1 \vee x_3 \vee \overline{x}_4] \wedge$$
$$[\overline{x}_1 \vee \overline{x}_3 \vee x_4] \wedge$$
$$[x_1 \vee x_3 \vee \overline{x}_4] \wedge$$
$$[x_1 \vee \overline{x}_3 \vee x_4] \wedge$$
$$[\overline{x}_1 \vee \overline{x}_2]$$

Models $\mathcal{M}(\phi)$

$$\{\overline{x}_1, \overline{x}_2, \overline{x}_3, \overline{x}_4\} \quad \{\overline{x}_1, \overline{x}_2, x_3, x_4\}$$
$$\{\overline{x}_1, x_2, x_3, x_4\} \quad \{x_1, \overline{x}_2, x_3, x_4\}$$
$$\{\overline{x}_1, x_2, \overline{x}_3, \overline{x}_4\} \quad \{x_1, \overline{x}_2, \overline{x}_3, \overline{x}_4\}$$

Definitions

- Input variables $x_1, x_2, \ldots, x_n$
- Assignment: $\alpha = \{\ell_1, \ell_2, \ldots, \ell_n\}$ with each $\ell_i \in \{x_i, \overline{x}_i\}$
- Models: $\mathcal{M}(\phi)$ is set of satisfying assignments for formula ϕ

Model Counting

Formula ϕ		**Models** $\mathcal{M}(\phi)$	
$[\overline{x}_1 \vee x_3 \vee \overline{x}_4]$	$\wedge$		
$[\overline{x}_1 \vee \overline{x}_3 \vee x_4]$	$\wedge$	$\{\overline{x}_1, \overline{x}_2, \overline{x}_3, \overline{x}_4\}$	$\{\overline{x}_1, \overline{x}_2, x_3, x_4\}$
$[x_1 \vee x_3 \vee \overline{x}_4]$	$\wedge$	$\{\overline{x}_1, x_2, x_3, x_4\}$	$\{x_1, \overline{x}_2, x_3, x_4\}$
$[x_1 \vee \overline{x}_3 \vee x_4]$	$\wedge$	$\{\overline{x}_1, x_2, \overline{x}_3, \overline{x}_4\}$	$\{x_1, \overline{x}_2, \overline{x}_3, \overline{x}_4\}$
$[\overline{x}_1 \vee \overline{x}_2]$			

Definitions

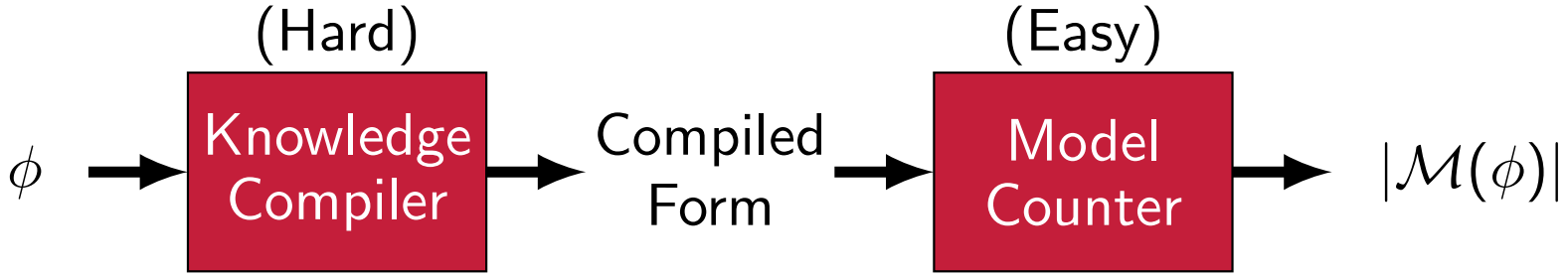
- Input variables $x_1, x_2, \ldots, x_n$
- Assignment: $\alpha = \{\ell_1, \ell_2, \ldots, \ell_n\}$ with each $\ell_i \in \{x_i, \overline{x}_i\}$
- Models: $\mathcal{M}(\phi)$ is set of satisfying assignments for formula ϕ

Model Counting Problem

- Given formula ϕ, compute $|\mathcal{M}(\phi)|$
- Challenging: #SAT more difficult than SAT

Knowledge Compilation

- Darwiche [DarMar-2002]

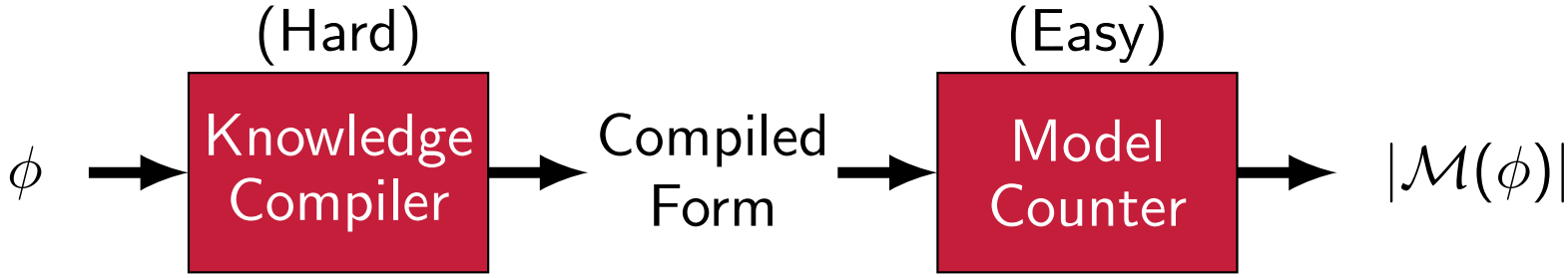


Convert CNF formula into more tractable representation

- Potentially exponential size
- Model counting polynomial in size of representation

Knowledge Compilation

- Darwiche [DarMar-2002]



Convert CNF formula into more tractable representation

- Potentially exponential size
- Model counting polynomial in size of representation

Concerns:

- Is the compiled form logically equivalent to the input formula?
- Is the counting computed correctly?

(Weighted) Model Counting

- Assign weight $w(x_i)$ to each input variable x_i
  - $0.0 < w(x_i) < 1.0$
- Define $w(\overline{x}_i) = 1 - w(x_i)$
  - Write as $\sim w(x_i)$
- Weighted count $\Delta(\phi, w)$ of formula ϕ:

$$\Delta(\phi, w) \quad = \quad \sum_{\alpha \in \mathcal{M}(\phi)} \prod_{\ell_i \in \alpha} w(\ell_i)$$

Standard Model Counting

- $w(x_i) = w(\overline{x}_i) = 1/2$ for all i
- $\Delta(\phi, w)$ gives density of function
  - Fraction of assignments that satisfy ϕ
- Scale by 2^n to get model count

Partitioned-Operation Formulas

Allowed Operations

- **Product:** $\phi_1 \wedge^{\mathsf{p}} \phi_2$, where $\mathcal{D}(\phi_1) \cap \mathcal{D}(\phi_2) = \emptyset$
  - $\mathcal{D}(\phi)$: Set of variables occuring in ϕ
- **Sum:** $\phi_1 \vee^{\mathsf{p}} \phi_2$, where $\mathcal{M}(\phi_1) \cap \mathcal{M}(\phi_2) = \emptyset$
- **Negation:** $\neg\phi$

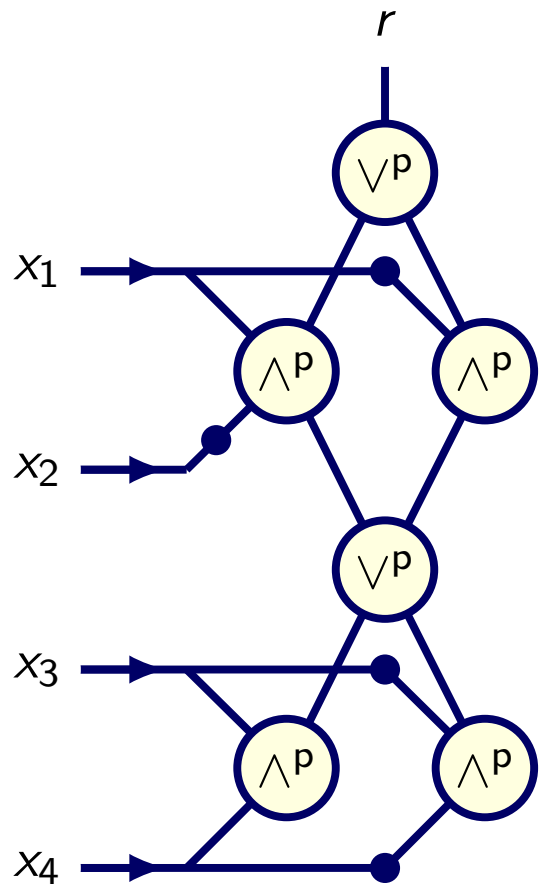
Weighted Count of Partitioned Formula

$$\begin{aligned} \Delta(\phi_1 \wedge^{\mathsf{p}} \phi_2,\, w) &= \Delta(\phi_1, w) \times \Delta(\phi_2, w) \\ \Delta(\phi_1 \vee^{\mathsf{p}} \phi_2,\, w) &= \Delta(\phi_1, w) + \Delta(\phi_2, w) \\ \Delta(\neg\phi,\, w) &= \sim \Delta(\phi, w) \end{aligned}$$

Partitioned-Operation Graph (POG)

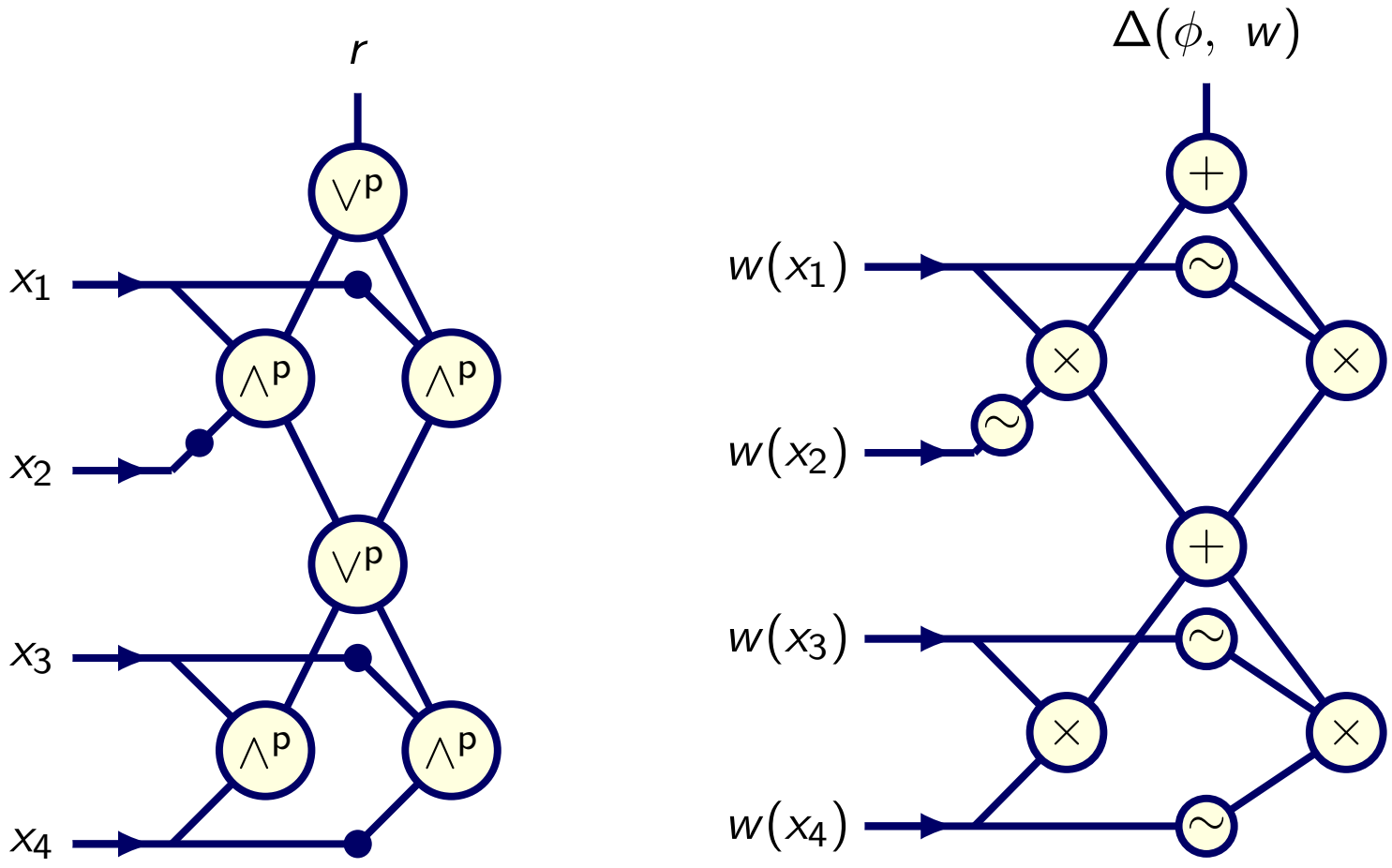
Formula ϕ

$$
\begin{aligned}
&[\overline{x}_1 \vee x_3 \vee \overline{x}_4] \quad \wedge \\
&[\overline{x}_1 \vee \overline{x}_3 \vee x_4] \quad \wedge \\
&[x_1 \vee x_3 \vee \overline{x}_4] \quad \wedge \\
&[x_1 \vee \overline{x}_3 \vee x_4] \quad \wedge \\
&[\overline{x}_1 \vee \overline{x}_2]
\end{aligned}
$$



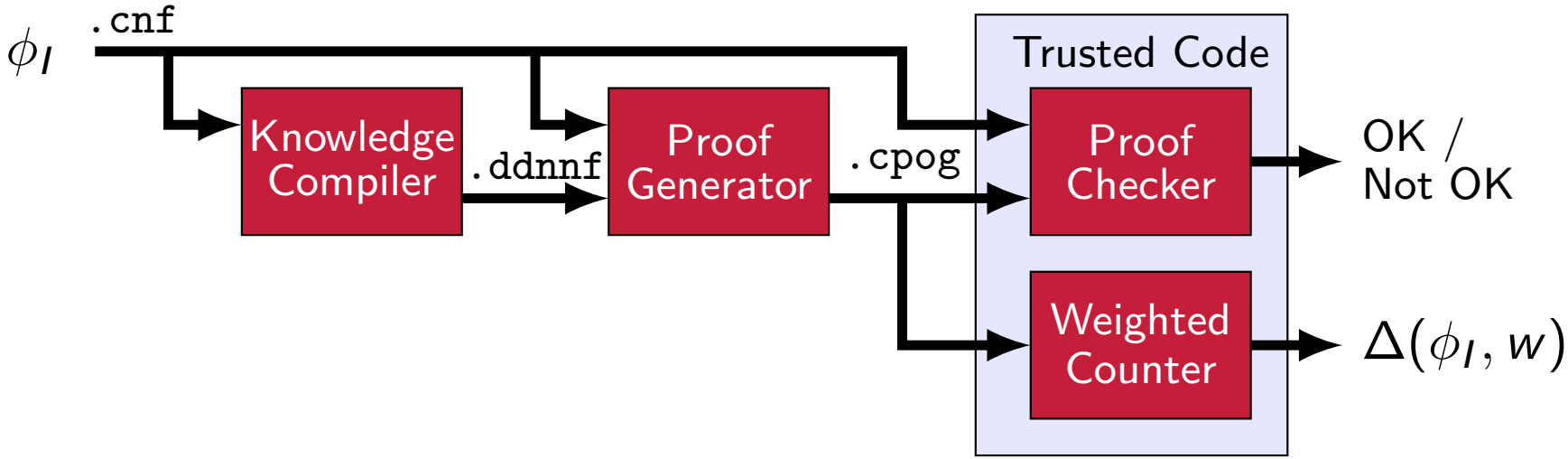
- Directed graph representation of partitioned-operation formula
- Each edge can be negative or positive

Weighted Count of POG



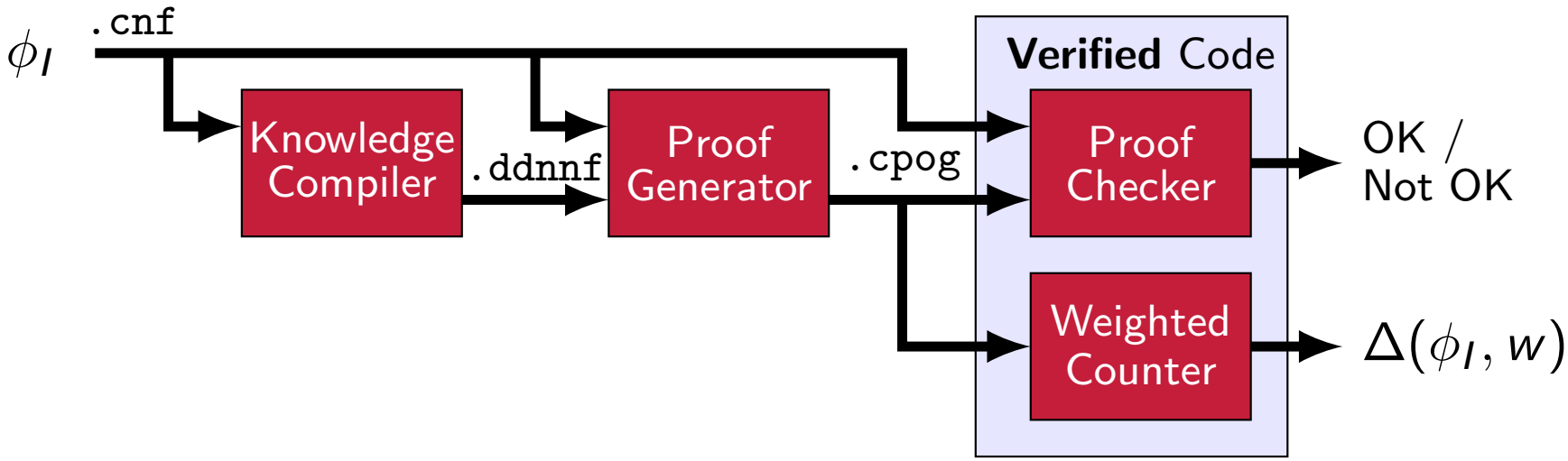
- Evaluation: Number of operations linear in graph size

Certifying Toolchain



- Knowledge Compiler (D4 [LagMar-2017]): Convert CNF into representation using only partitioned operations
- Proof Generator: Generate file combining POG definition + equivalence proof
- Proof Checker: Validate proof file
- Weighted Counter: Compute standard or weighted model count

Verifying the Trusted Code



Using the Lean 4 theorem prover [DemUlr-2021]

- Soundness of proof system
  - Helped us identify unsoundness in our prototype proof rules
- Verified proof checker
  - Around 6× slower than one implemented in C
- Verified weighted counter

Formally Verified Theorems

Theorem (Proof framework and checker correctness)

If the CPOG proof checker has assembled POG P starting from input formula ϕ_I, and the final check succeeds, then ϕ_I is logically equivalent to the formula ϕ_P represented by P.

Theorem (Correctness of efficient weighted counter)

For any POG P, the weighted counter executed on P with weights w returns $\Delta(\phi_P, w)$.

Related Work: CD4

CD4: Certifying D4 [CapLagMar-2021]

- Modified version of D4
  - Generates annotated output + clausal proof in DRAT format
  - Verify with checker + DRAT-TRIM [HeuHunWet-2013]
- Experiments: Scales very well

Limitations:

- Proof framework tied closely to compiler implementation
- No formal proof of soundness
  - Found exploitable weakness

Related Work: MICE

MICE: Proof framework for top-down model counters
[FicHecRol-2022]

- Modify model counter or generate proof from D4 output
- Generates series of proof obligations
- Experiments: Scaling problems when many shared subgraphs

Limitations:

- Only verifies standard model counting
- Proof framework based on specific class of model counters
- No formal proof of soundness
- No verified checker

Importance of Formal Verification

Claim:

Any proof framework that has not been mechanically verified is unsound

- Borne out by our own experience

CPOG File: Declaration + Proof

Clausal Representation of POG θ_P

- Tseitin encoding of POG operations
  - Extension variable u for each operation node **u**
  - Node **u** with k children characterized by $k + 1$ defining clauses
- Each child indicated by literal
  - Positive or negated argument
  - Input variable or result from other operation
- Unit clause $[r]$ for root node **r**

Proof Steps

- Sequence of clause additions and deletions

CPOG Proof Objective

$$\phi_I(X) \iff \exists! \, Z \; \theta_P(X, Z)$$

- ϕ_I: Input formula
- θ_P: POG formula
  - Defining clauses for POG
  - Unit clause $[r]$ for root literal
- Z: extension variables for the POG operations
- For any assignment α to X:
  - Defining clauses induce unique extension α^* to $X \cup Z$
  - α satisfies ϕ_I if and only if $\alpha^*(r) = 1$

Proof Methodology

- Transform ϕ_I to θ_P
  - Via sequence of equivalence-preserving proof steps

CPOG Example: Formula

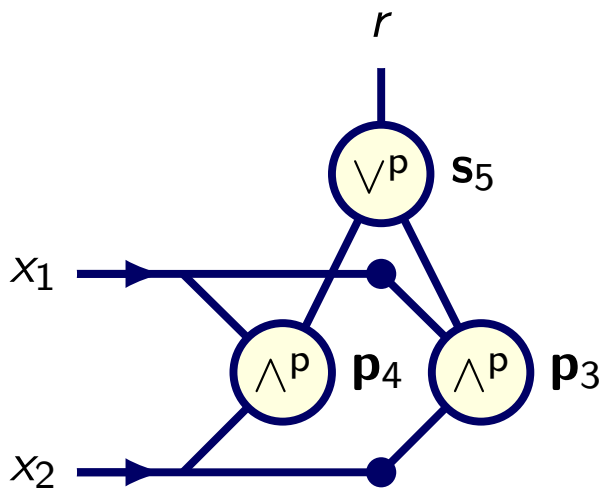
- Encode formula $x_1 \leftrightarrow x_2$.
- CNF representation:

$$[x_1 \vee \overline{x}_2] \wedge [\overline{x}_1 \vee x_2]$$

Clause Database

ID	Literals	Explanation
1	1 -2	Input
2	-1 2	Input

CPOG Example: POG Declaration



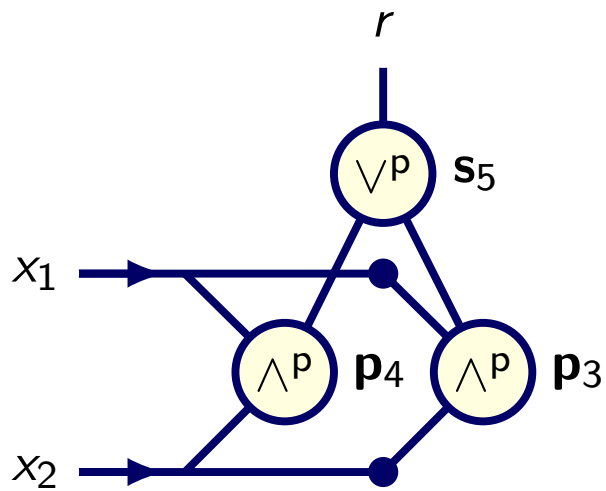
CPOG Declarations

```
3   p 3 -1 -2   0
```

Clause Database

ID	Literals	Explanation
1	1 -2	Input
2	-1 2	Input
3	3 1 2	$\mathbf{p}_3$
4	-3 -1	
5	-3 -2	

CPOG Example: POG Declaration



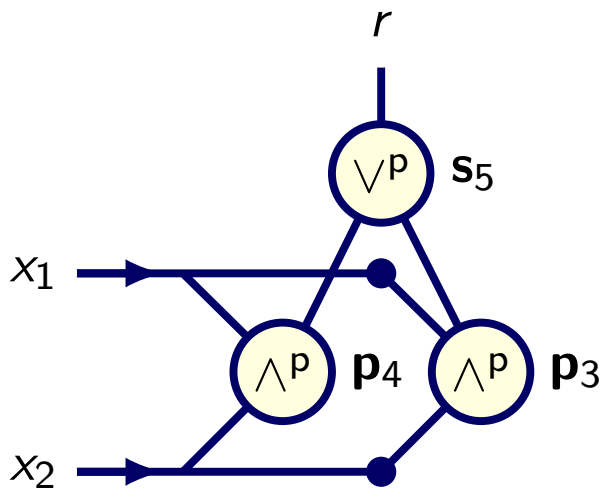
CPOG Declarations

```
3   p 3 -1 -2   0
6   p 4 1 2     0
```

Clause Database

ID	Literals	Explanation
1	1 -2	Input
2	-1 2	Input
3	3 1 2	$\mathbf{p_3}$
4	-3 -1	
5	-3 -2	
6	4 -1 -2	$\mathbf{p_4}$
7	-4 1	
8	-4 2	

CPOG Example: POG Declaration



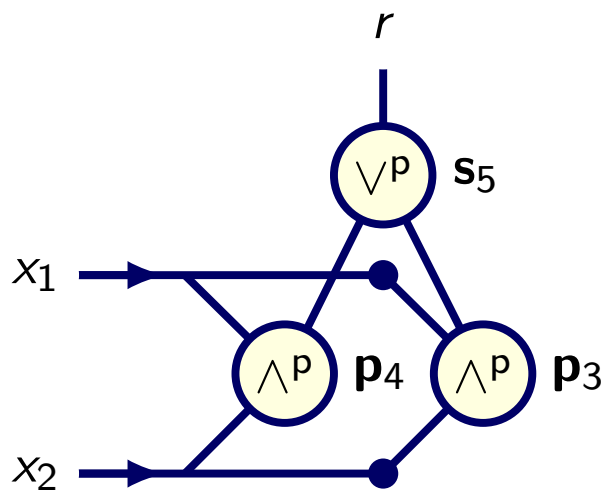
CPOG Declarations

```
3   p 3 -1 -2      0
6   p 4 1 2        0
9   s 5 3 4 4 7    0
```

Clause Database

ID	Literals	Explanation
1	1 -2	Input
2	-1 2	Input
3	3 1 2	$\mathbf{p}_3$
4	-3 -1	
5	-3 -2	
6	4 -1 -2	$\mathbf{p}_4$
7	-4 1	
8	-4 2	
9	-5 3 4	$\mathbf{s}_5$
10	5 -3	
11	5 -4	

CPOG Example: POG Declaration



CPOG Declarations

```
9  s 5 3 4 4 7  0
```

- Sum declaration must justify mutual exclusion
- Resolving clauses 4 and 7 gives $\overline{p}_3 \vee \overline{p}_4$.

Clause Database

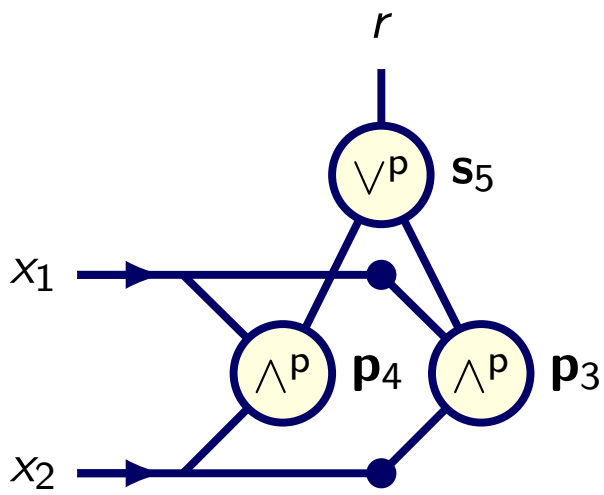
ID	Literals	Explanation
1	1 -2	Input
2	-1 2	Input
3	3 1 2	$\mathbf{p}_3$
4	*-3 -1*	
5	-3 -2	
6	4 -1 -2	$\mathbf{p}_4$
7	*-4 1*	
8	-4 2	
9	-5 3 4	$\mathbf{s}_5$
10	5 -3	
11	5 -4	

CPOG Proof Structure: Forward Implication

$$\phi_I(X) \implies \exists! Z\, \theta_P(X, Z)$$

- Add clauses by reverse unit propagation (RUP)
- Terminating with unit clause $[r]$
- Any assignment satisfying ϕ_I (when extended) causes the POG to evaluate to true

CPOG Example: Forward Implication



CPOG Assertions

```
12   a -2 5    0   11 1 6       0
13   a 5       0   10 12 2 3    0
```

- Must give justifying RUP sequences
- Finish with unit clause asserting root literal

1	1 -2	Input
2	-1 2	Input
3	3 1 2	p_3
4	-3 -1	
5	-3 -2	
6	4 -1 -2	p_4
7	-4 1	
8	-4 2	
9	-5 3 4	s_5
10	5 -3	
11	5 -4	
12	-2 5	
13	5	Root literal

Forward Proof Generation Methods

Monolithic

- Single call to proof-generating SAT solver
- Experimentally: Scales to POGs with $\sim 10^6$ defining clauses

Structural

- Top-down recursion on POG structure
- Avoid exponential expansion by defining and applying lemmas
  - Can express within CPOG structure
- Experimentally: Scales to POGs with $\sim 10^8$ defining clauses

CPOG Proof Structure: Reverse Implication

Reverse Implication Proof

$$\exists! \, Z \; \theta_P(X, Z) \implies \phi_I(X)$$

- Delete clauses by RUP
  - Deleted clause implied by remaining ones
- Only clausal representation of POG θ_P remains at end

CPOG Example: Reverse Implication

CPOG Deletions

```
d  12  11 1 6  0
```

- All deletions must give justifying RUP sequence

1	1 -2	Input
2	-1 2	Input
3	3 1 2	$\mathbf{p}_3$
4	-3 -1	
5	-3 -2	
6	4 -1 -2	$\mathbf{p}_4$
7	-4 1	
8	-4 2	
9	-5 3 4	$\mathbf{s}_5$
10	5 -3	
11	5 -4	
12		*Deleted*
13	5	Root literal

CPOG Example: Reverse Implication

CPOG Deletions

```
d  12  11 1 6     0
d   1  13 5 7 9   0
```

- All deletions must give justifying RUP sequence

1		*Deleted*
2	-1 2	Input
3	3 1 2	$\mathbf{p}_3$
4	-3 -1	
5	-3 -2	
6	4 -1 -2	$\mathbf{p}_4$
7	-4 1	
8	-4 2	
9	-5 3 4	$\mathbf{s}_5$
10	5 -3	
11	5 -4	
12		*Deleted*
13	5	Root literal

CPOG Example: Reverse Implication

CPOG Deletions

```
d  12  11 1 6     0
d   1  13 5 7 9   0
d   2  13 4 8 9   0
```

- All deletions must give justifying RUP sequence

1		*Deleted*
2		*Deleted*
3	3 1 2	$\mathbf{p}_3$
4	-3 -1	
5	-3 -2	
6	4 -1 -2	$\mathbf{p}_4$
7	-4 1	
8	-4 2	
9	-5 3 4	$\mathbf{s}_5$
10	5 -3	
11	5 -4	
12		*Deleted*
13	5	Root literal

Proof Result

Initial Clause Database: ϕ_I

ID	Literals	Explanation
1	1 -2	Input
2	-1 2	Input

Final Clause Database: θ_P

ID	Literals	Explanation
3	3 1 2	$\mathbf{p}_3$
4	-3 -1	
5	-3 -2	
6	4 -1 -2	$\mathbf{p}_4$
7	-4 1	
8	-4 2	
9	-5 3 4	$\mathbf{s}_5$
10	5 -3	
11	5 -4	
13	5	Root literal

- Transformed input formula ϕ_I into POG formula θ_P
  - Via equivalence-preserving proof steps

CPOG Checking Requirements

- Partitioned product: $\mathcal{D}(\phi_1) \cap \mathcal{D}(\phi_2) = \emptyset$
  - Syntactic check of dependencies
- Partitioned sum: $\mathcal{M}(\phi_1) \cap \mathcal{M}(\phi_2) = \emptyset$
  - Check of mutual-exclusion RUP sequence
- Clause addition and deletion
  - Check of RUP sequence

Experimental Evaluation

Benchmark Problems:

- 180 unique formulas from 2022 unweighted and weighted model counting competitions

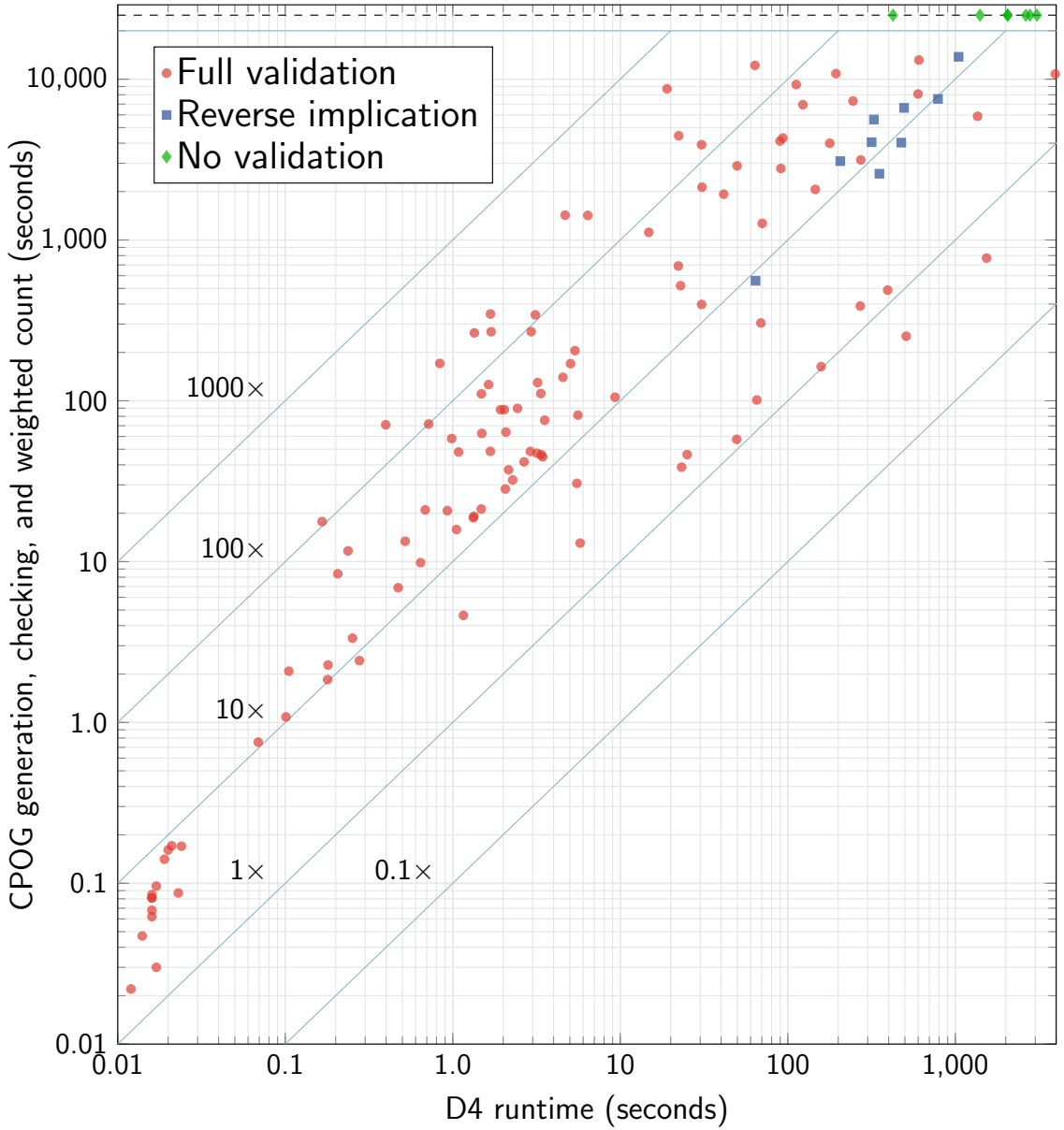
D4 Execution

- Time limit of 4000 seconds
- Completed 124 problems
- Converted to POGs ranging from 304 to 2,761,457,765 defining clauses

Running our toolchain

- Limited CPOG generation to 10,000 seconds
- Full proofs for 108 problems
- Reverse implication proofs for 9 more
- No proofs of 7

Experimental Results: Toolchain Runtime



Ratios

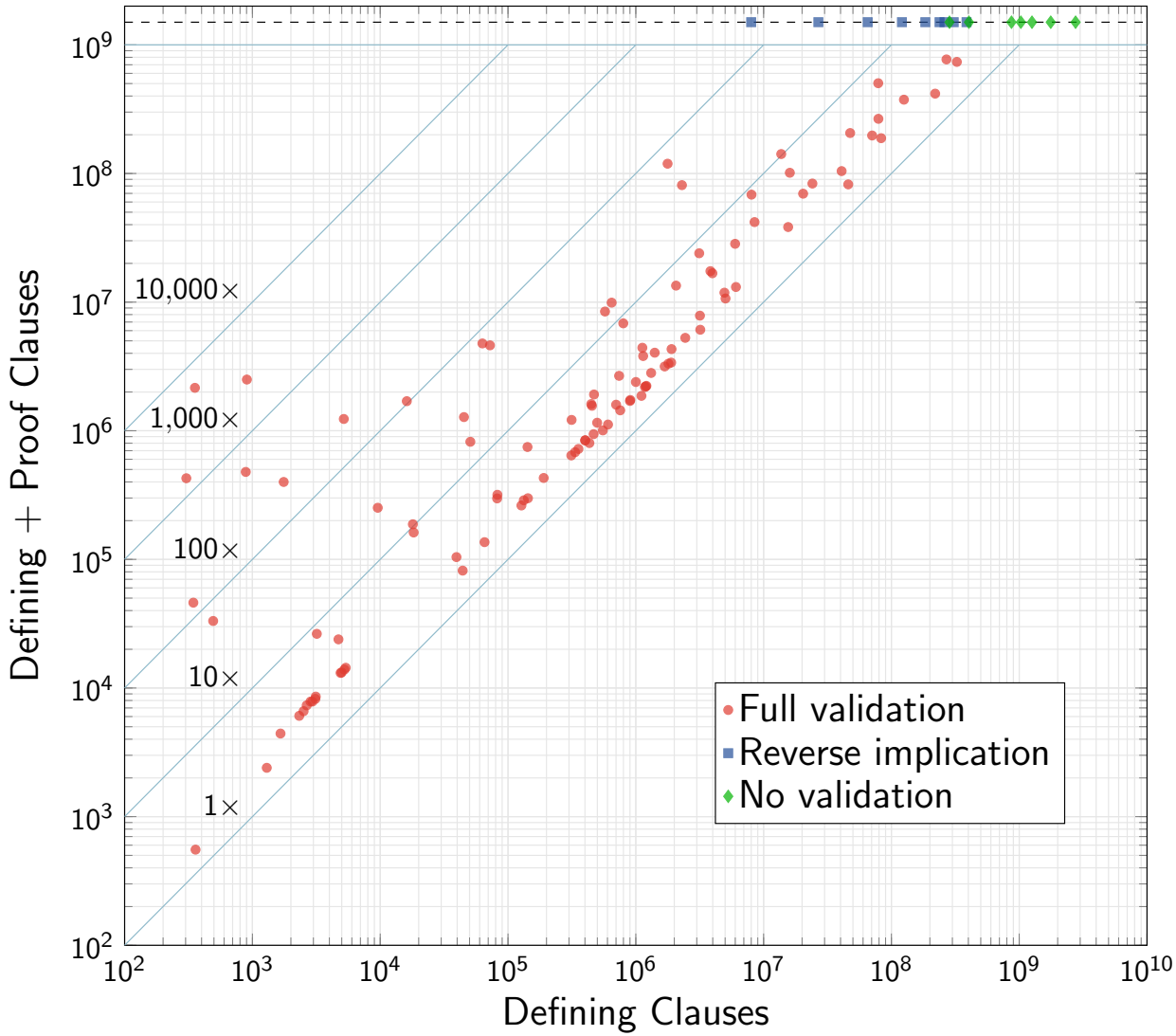
Min
0.5×

Harmonic mean
5.9×

Median
16.4×

Max
465.8×

Experimental Results: CPOG Sizes



Ratios

Min
1.5×

Harmonic mean
3.1×

Median
2.7×

Max
6073.0×

Final Thoughts

Observations

- Toolchain can handle all but largest outputs from D4
- Framework is very general
  - E.g., can generate, prove, and apply lemmas without any extensions
  - Not tied to particular compilation method

Future Work

- Improve speed and capacity of toolchain
- Handle outputs from other knowledge compilers
- Certification of other automated reasoning tools

Supplementary Information

Code

https://github.com/rebryant/cpog

Documentation

https://doi.org/10.5281/zenodo.7966174

- Worked example
- More details on algorithms
- More details on formal verification
- Lots of experimental results

References

CapLagMar-2021 F. Capelli, J.-M. Lagniez, and P. Marquis, "Certifying top-down decision-DNNF compilers", AAAI, 2021

DarMar-2002 A. Darwiche and P. Marquis, "A knowledge compilation map," JAIR, 2002

DemUlr-2021 L. de Moura and S. Ulrich, "The Lean 4 theorem prover and programming language," CADE, 2021

FicHecRol-2022 J. Fichte, M. Hecher, and V. Roland, "Proofs for propositional model counting," SAT 2022.

HeuHunWet-2013 M. J. H. Heule, W. A. Hunt, Jr., N. Wetzler, "Trimming while checking clausal proofs," FMCAD, 2013.

LagMar-2017 J.-M. Lagniez and P. Marquis, "An improved decision-DNNF compiler," IJCAI, 2017