

Towards Practical First-Order Model Counting

Ananth K. Kidambi¹ Guramrit Singh¹ **Paulius Dilkas**^{2,3}
Kuldeep S. Meel^{4,2}

¹IIT Bombay, India

²University of Toronto, Canada

³Vector Institute, Canada

⁴Georgia Tech, USA

SAT 2025

Motivation

Example Setting

- ▶ Let Δ be a set of cardinality $n \in \mathbb{N}_0$
- ▶ Suppose we want to count all $P \subseteq \Delta^2$ (as a function of n) that are:
 - ▶ functions,
 - ▶ bijections,
 - ▶ partial orders,
 - ▶ symmetric,
 - ▶ transitive,
 - ▶ etc.

Motivation

Example Setting

- ▶ Let Δ be a set of cardinality $n \in \mathbb{N}_0$
- ▶ Suppose we want to count all $P \subseteq \Delta^2$ (as a function of n) that are:
 - ▶ functions,
 - ▶ bijections,
 - ▶ partial orders,
 - ▶ symmetric,
 - ▶ transitive,
 - ▶ etc.
- 👎 Propositional model counting ($\#SAT$) is $\#P$ -complete
- 👍 But many of these counting problems have **efficient solutions**
- ▶ And we can find them using **first-order model counting**
 - ▶ i.e., reasoning about sets, subsets, and arbitrary elements without **grounding** them

More Formally: What Is the Input?

Example Input Sentence

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z$$

Many-Sorted Function-Free First-Order Logic with Equality

More Formally: What Is the Input?

Example Input Sentence

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z$$

Many-Sorted Function-Free First-Order Logic with Equality

More Formally: What Is the Input?

Example Input Sentence

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z$$

Many-Sorted **Function-Free** First-Order Logic with Equality

More Formally: What Is the Input?

Example Input Sentence

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z$$

Many-Sorted Function-Free First-Order Logic with Equality

More Formally: What Is the Input?

Example Input Sentence

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z$$

Many-Sorted Function-Free First-Order Logic with Equality

- ▶ Any number of variables and constants
- ▶ \exists and \forall quantifiers can be nested arbitrarily deeply
- ▶ All domains are finite
 - ▶ Solutions are functions that take domain sizes as inputs
- ▶ Of course, not all valid inputs have tractable solutions

More Formally: What Is the Input?

Example Input Sentence

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z$$

Many-Sorted Function-Free First-Order Logic with Equality

- ▶ Any number of variables and constants
- ▶ \exists and \forall quantifiers can be nested arbitrarily deeply
- ▶ All domains are finite
 - ▶ Solutions are functions that take domain sizes as inputs
- ▶ Of course, not all valid inputs have tractable solutions

First-Order Model Counting (FOMC)

- ▶ Each predicate acts like a **subset**
 - ▶ of a domain or a Cartesian product of domains
- ▶ Goal: count **combinations of subsets** that satisfy the sentence

Exact Algorithms for FOMC

Predecessors of This Work

- ▶ **FORCLIFT** (Van den Broeck et al. 2011)
 - ▶ knowledge compilation to **FO d-DNNF**
- ▶ **CRANE** (Dilkas and Belle 2023)
 - ▶ knowledge compilation to FO d-DNNF + **directed cycles**
 - ▶ extends **FORCLIFT** with support for:
 - ▶ more input sentences and
 - ▶ recursive solutions

Exact Algorithms for FOMC

Predecessors of This Work

- ▶ **FORCLIFT** (Van den Broeck et al. 2011)
 - ▶ knowledge compilation to **FO d-DNNF**
- ▶ **CRANE** (Dilkas and Belle 2023)
 - ▶ knowledge compilation to **FO d-DNNF + directed cycles**
 - ▶ extends **FORCLIFT** with support for:
 - ▶ more input sentences and
 - ▶ recursive solutions

Some Other Approaches

- ▶ **L2C** (Kazemi and Poole 2016)
 - ▶ knowledge compilation to **C++** code
- ▶ **Alchemy** (Gogate and Domingos 2016)
 - ▶ **DPLL**-style search
- ▶ **FastWFOMC** (van Bremen and Kuželka 2021)
 - ▶ based on **cell enumeration**

Previous Work: CRANE (Dilkas and Belle 2023)

- ▶ A knowledge compilation approach:
 - ▶ Sentences \rightarrow labelled digraphs \rightarrow function-defining equations
- ▶ Two variants: greedy search and breadth-first search (BFS)

Previous Work: CRANE (Dilkas and Belle 2023)

- ▶ A knowledge compilation approach:
 - ▶ Sentences \rightarrow labelled digraphs \rightarrow function-defining equations
- ▶ Two variants: greedy search and breadth-first search (BFS)

An Example Solution for Counting Bijections

$$f(m, n) = \sum_{l=0}^n \binom{n}{l} (-1)^{n-l} g(l, m),$$
$$g(l, m) = g(l-1, m) + mg(l-1, m-1)$$

Previous Work: CRANE (Dilkas and Belle 2023)

- ▶ A knowledge compilation approach:
 - ▶ Sentences \rightarrow labelled digraphs \rightarrow function-defining equations
- ▶ Two variants: greedy search and breadth-first search (BFS)

An Example Solution for Counting Bijections

$$f(m, n) = \sum_{l=0}^n \binom{n}{l} (-1)^{n-l} g(l, m),$$
$$g(l, m) = g(l-1, m) + mg(l-1, m-1)$$

Issues We Are Going to Address

Completeness: recursive functions (like g) have no base cases

Usability: how do I compute, e.g., $f(7, 7)$? (C++ to the rescue!)

The Workflow of CRANE2 (1/2)

1. Use **CRANE** to compile sentence ϕ into a set of equations \mathcal{E}

The Workflow of CRANE2 (1/2)

1. Use **CRANE** to compile sentence ϕ into a set of equations \mathcal{E}
2. Simplify them, e.g.,

$$g(l, m) = \sum_{k=0}^m [0 \leq k \leq 1] \binom{m}{k} g(l-1, m-k)$$

becomes

$$g(l, m) = g(l-1, m) + mg(l-1, m-1)$$

The Workflow of CRANE2 (1/2)

1. Use **CRANE** to compile sentence ϕ into a set of equations \mathcal{E}
2. Simplify them, e.g.,

$$g(l, m) = \sum_{k=0}^m [0 \leq k \leq 1] \binom{m}{k} g(l-1, m-k)$$

becomes

$$g(l, m) = g(l-1, m) + mg(l-1, m-1)$$

3. (\Rightarrow) Identify a **sufficient set of base cases** of all recursive functions
 - ▶ e.g., $\{g(0, m), g(l, 0)\}$

The Workflow of CRANE2 (2/2)

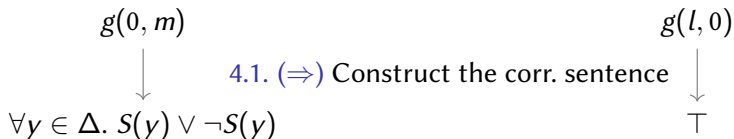
4. For each base case:

$$g(0, m)$$

$$g(l, 0)$$

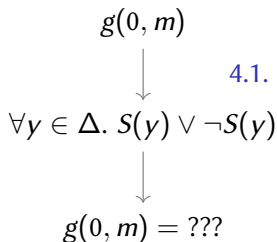
The Workflow of CRANE2 (2/2)

4. For each base case:



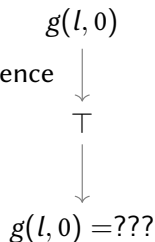
The Workflow of CRANE2 (2/2)

4. For each base case:



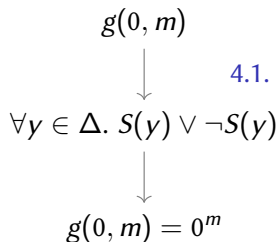
4.1. (\Rightarrow) Construct the corr. sentence

4.2. Recurse



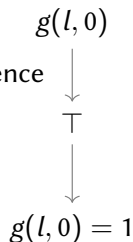
The Workflow of CRANE2 (2/2)

4. For each base case:



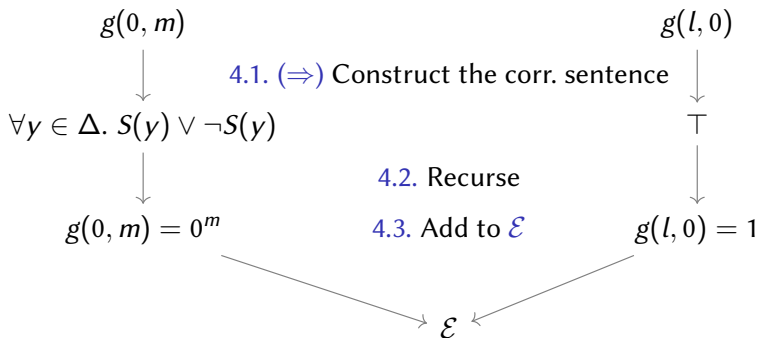
4.1. (\Rightarrow) Construct the corr. sentence

4.2. Recurse



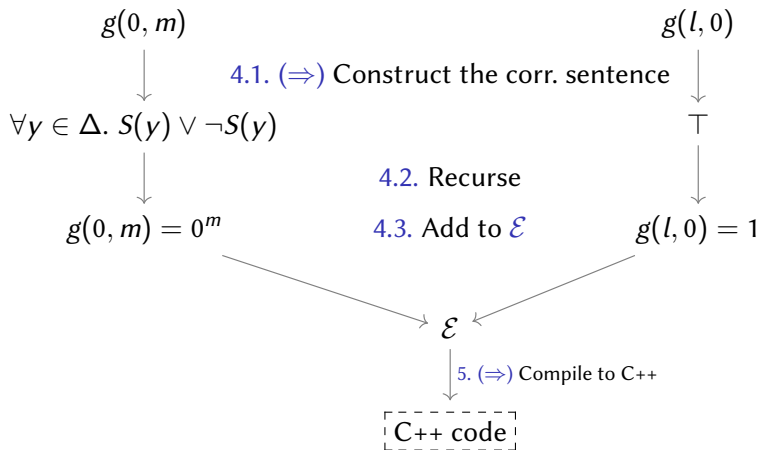
The Workflow of CRANE2 (2/2)

4. For each base case:



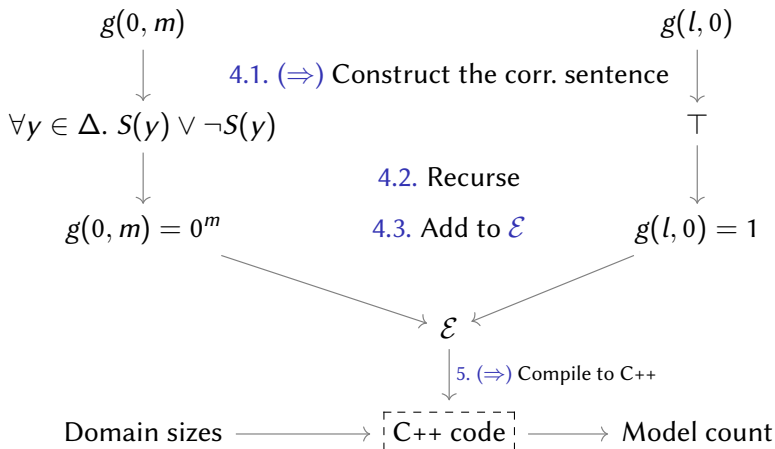
The Workflow of CRANE2 (2/2)

4. For each base case:



The Workflow of CRANE2 (2/2)

4. For each base case:



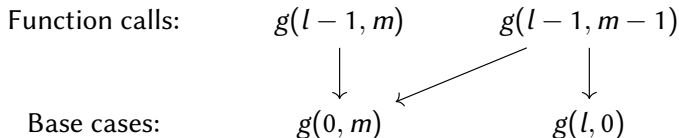
Finding (a Sufficient Set of) Base Cases

Outline

1. For every **function call**:
 - 1.1 For every **argument** of the form *var* – *const*:
 - 1.1.1 Replace the **signature parameter** with 0, 1, ..., *const* – 1
 - 1.2 For every **argument** of the form *const*:
 - 1.2.1 Replace the corresponding signature parameter with *const*

Example

The **signature** of *g* is *g(l, m)*.



No Infinite Cycles

Theorem

*The **evaluation** of a recursive function always **terminates**.*

No Infinite Cycles

Theorem

*The **evaluation** of a recursive function always **terminates**.*

Proof (hints).

- ▶ There exists a **topological ordering** of functions
- ▶ All function calls follow the **structure** from the previous slide
- ▶ Some common-sense assumptions about the **evaluation order** and previous work



From a Base Case to a Sentence

From Previous Work (Dilkas and Belle 2023)

- ▶ **CRANE** associates each function f with a sentence ϕ such that $\text{CRANE}(\phi)$ produces the definition of f
- ▶ And there is a bijection between the **parameters** of f and the **domains** of ϕ

Example

- ▶ Base case: $g(0, m)$

From a Base Case to a Sentence

From Previous Work (Dilkas and Belle 2023)

- ▶ **CRANE** associates each function f with a sentence ϕ such that $\text{CRANE}(\phi)$ produces the definition of f
- ▶ And there is a bijection between the **parameters** of f and the **domains** of ϕ

Example

- ▶ Base case: $g(0, m)$
 $\begin{array}{cc} \Gamma & \Delta \\ \updownarrow & \updownarrow \end{array}$

From a Base Case to a Sentence

From Previous Work (Dilkas and Belle 2023)

- ▶ **CRANE** associates each function f with a sentence ϕ such that $\text{CRANE}(\phi)$ produces the definition of f
- ▶ And there is a bijection between the **parameters** of f and the **domains** of ϕ

Example

- ▶ Base case: $g(\overset{\Gamma}{\downarrow}0, \overset{\Delta}{\downarrow}m)$
- ▶ **Part** of the sentence of g :

$$\forall x \in \Gamma. \forall y \in \Delta. S(y) \vee \neg P(x, y) \quad (1)$$

From a Base Case to a Sentence

From Previous Work (Dilkas and Belle 2023)

- ▶ **CRANE** associates each function f with a sentence ϕ such that $\text{CRANE}(\phi)$ produces the definition of f
- ▶ And there is a bijection between the **parameters** of f and the **domains** of ϕ

Example

- ▶ Base case: $g(\overset{\Gamma}{\downarrow}0, \overset{\Delta}{\downarrow}m)$
- ▶ **Part** of the sentence of g :

$$\forall x \in \Gamma. \forall y \in \Delta. S(y) \vee \neg P(x, y) \quad (1)$$

- ▶ $g(0, \dots)$ means we need to simplify (1) by assuming $|\Gamma| = 0$

From a Base Case to a Sentence

From Previous Work (Dilkas and Belle 2023)

- ▶ **CRANE** associates each function f with a sentence ϕ such that $\text{CRANE}(\phi)$ produces the definition of f
- ▶ And there is a bijection between the **parameters** of f and the **domains** of ϕ

Example

- ▶ Base case: $g(\overset{\Gamma}{\downarrow}0, \overset{\Delta}{\downarrow}m)$
- ▶ **Part** of the sentence of g :

$$\forall x \in \Gamma. \forall y \in \Delta. S(y) \vee \neg P(x, y) \quad (1)$$

- ▶ $g(0, \dots)$ means we need to simplify (1) by assuming $|\Gamma| = 0$
- ▶ Result: $\forall y \in \Delta. S(y) \vee \neg S(y)$ (**Smoothing**)

The Structure of the Resulting C++ Program

initialise $\text{Cache}_{g(0,m)}$, $\text{Cache}_{g(l,0)}$, Cache_g , and Cache_f ;

The Structure of the Resulting C++ Program

initialise $\text{Cache}_{g(0,m)}$, $\text{Cache}_{g(l,0)}$, Cache_g , and Cache_f ;

Function $g_{0,m}(m)$: ...

Function $g_{l,0}(l)$: ...

The Structure of the Resulting C++ Program

initialise $\text{Cache}_{g(0,m)}$, $\text{Cache}_{g(l,0)}$, Cache_g , and Cache_f ;

Function $g_{0,m}(m)$: ...

Function $g_{l,0}(l)$: ...

Function $g(l, m)$:

if $(l, m) \in \text{Cache}_g$ **then return** $\text{Cache}_g(l, m)$;

if $l = 0$ **then return** $g_{0,m}(m)$;

if $m = 0$ **then return** $g_{l,0}(l)$;

$r \leftarrow g(l-1, m) + mg(l-1, m-1)$;

$\text{Cache}_g(l, m) \leftarrow r$;

return r ;

Function $f(m, n)$: ...

Function **Main**:

$(m, n) \leftarrow \text{ParseCommandLineArguments}()$;

return $f(m, n)$;

► Friends & Smokers

$$(\forall x, y \in \Delta. S(x) \wedge F(x, y) \rightarrow S(y)) \wedge (\forall x \in \Delta. S(x) \rightarrow C(x))$$

Benchmarks

► Friends & Smokers

$$(\forall x, y \in \Delta. S(x) \wedge F(x, y) \rightarrow S(y)) \wedge (\forall x \in \Delta. S(x) \rightarrow C(x))$$

► Functions

$$(\forall x \in \Gamma. \exists y \in \Delta. P(x, y)) \wedge \\ (\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \rightarrow y = z)$$

Benchmarks

► Friends & Smokers

$$(\forall x, y \in \Delta. S(x) \wedge F(x, y) \rightarrow S(y)) \wedge (\forall x \in \Delta. S(x) \rightarrow C(x))$$

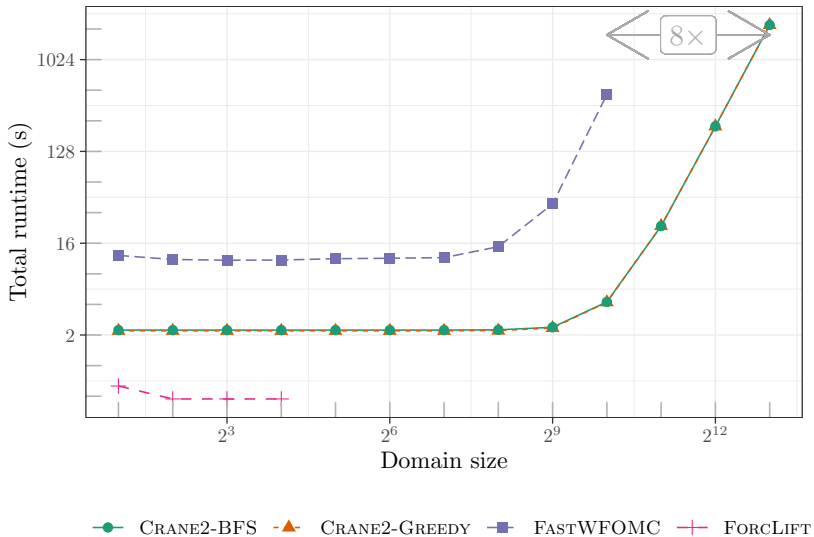
► Functions

$$(\forall x \in \Gamma. \exists y \in \Delta. P(x, y)) \wedge \\ (\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \rightarrow y = z)$$

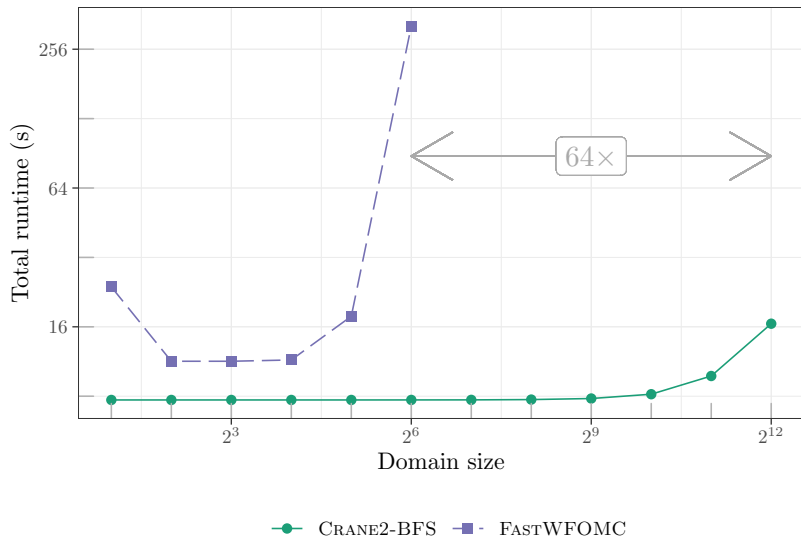
► Bijections

$$(\forall x \in \Gamma. \exists y \in \Delta. P(x, y)) \wedge \\ (\forall y \in \Delta. \exists x \in \Gamma. P(x, y)) \wedge \\ (\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \rightarrow y = z) \wedge \\ (\forall x, z \in \Gamma. \forall y \in \Delta. P(x, y) \wedge P(z, y) \rightarrow x = z)$$

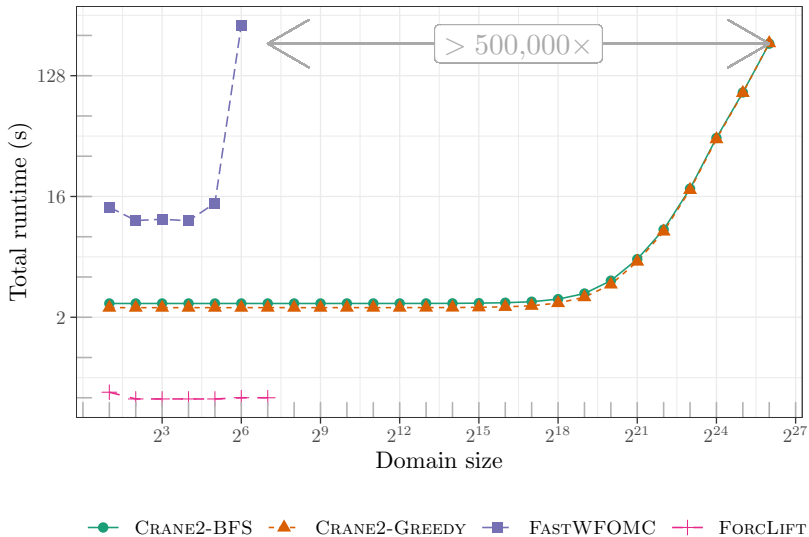
Friends & Smokers



Bijections



Functions



Summary & Future Work

Contributions

Completeness: recursive solutions now come with **base cases**

Usability: compilation to C++ programs

Scalability compared to other FOMC algorithms

- ▶ 8 to 500,000 times higher domain sizes

Summary & Future Work

Contributions

Completeness: recursive solutions now come with **base cases**

Usability: compilation to C++ programs

Scalability compared to other FOMC algorithms

- ▶ 8 to 500,000 times higher domain sizes

Future Work

- ▶ Support for weighted counting (trivial)
- ▶ Experiments on a large set of benchmarks
- ▶ Completeness for fragments of first-order logic
- ▶ Fine-grained complexity