Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Weighted Model Counter for Some Domain-Liftable Languages

Bc. Václav Kůla

Supervisor: Ing. Jan Tóth
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kůla Václav**	Personal ID number:	**491883**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence**		

II. Master’s thesis details

Master’s thesis title in English:

Weighted Model Counter for Some Domain-Liftable Languages

Master’s thesis title in Czech:

Počítání vážených modelů pro některé doménově-liftovatelné jazyky

Guidelines:

1. Familiarize yourself with the Weighted First-Order Model Counting (WFOMC) problem and domain-liftability defined in terms of it.
2. Familiarize yourself with some of the recently identified domain-liftable languages. Namely, the two-variable fragment with cardinality constraints, the two-variable fragment with counting quantifiers (C2) and C2 with the linear order axiom. Optionally others.
3. Design a grounding scheme for those languages. If there are several suitable approaches, compare their advantages and disadvantages.
4. Design and implement algorithms for solving WFOMC on the propositional level over the studied domain-liftable languages using the designed grounding scheme and available propositional model counters.
5. Propose a set of problems that can be used to compare efficiency of your solution with others.
6. Using the dataset, compare efficiency of your propositional method with efficiency of lifted techniques presented in the literature. Visualize the results.

Bibliography / sources:

1. Van den Broeck, G. 2011. On the Completeness of First-Order Knowledge Compilation for Lifted Probabilistic Inference. Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS’11, 1386–1394.
2. Kuželka, O. 2021. Weighted First-Order Model Counting in the Two-Variable Fragment With Counting Quantifiers. Journal of Artificial Intelligence Research, 70: 1281–1307.
3. Tóth, J.; and Kuželka, O. 2023. Lifted Inference with Linear Order Axiom. Proceedings of the AAAI Conference on Artificial Intelligence, 37(10), 12295-12304.
4. Graedel, E.; Otto, M.; and Rosen, E. 1997. Two-variable logic with counting is decidable. In Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science, 306–317. IEEE.
5. van Bremen, T.; and Kuželka, O. 2021b. Lifted Inference with Tree Axioms. Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, 599–608.
6. Van den Broeck, G., Kersting, K., Natarajan, S., and Poole, D. 2021. An Introduction to Lifted Probabilistic Inference. MIT Press

Name and workplace of master’s thesis supervisor:

Ing. Jan Tóth   Department of Computer Science FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **01.02.2024**   Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Jan Tóth Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Ing. Jan Tóth, for his invaluable guidance, patience, and feedback throughout this endeavor. This work would not be possible without his collaboration. I would also like to acknowledge my family for their unwavering support throughout my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 23, 2024

Prohlašuji, že jsem předloženou práci vypracoval samostantně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 23. května 2024

Abstract

This thesis studies weighted first-order model counting. Weighted first-order model counting can be used to solve many practical problems and has been shown to be domain-liftable for certain first-order logic fragments. In this work, we implement a grounder of first-order logic sentences from these fragments, and use available SAT solver and weighted model counters to compare their effectiveness to the lifted-methods. We create a dataset to test the effectiveness of both lifted and non-lifted methods and provide experimental results of their performance over said dataset. Lastly, we use these results to argue that the state-of-the-art lifted methods vastly outperform their non-lifted counterparts, as well as provide a few observations about non-lifted methods.

Keywords: weighted first-order model counting, weighted model counting, grounding, cardinality constraints, first-order logic, Herbrand semantics

Supervisor: Ing. Jan Tóth

Abstrakt

Tato práce studuje počítání vážených modelů v logice prvního řádu. Počítání vážených modelů v logice prvního řádu může být použito k řešení mnoha praktických problémů a bylo dokázáno, že pro určité logické fragmenty je možné počítat ho v čase polynomiálním ve velikosti domény. V této práci implementujeme grounder sentencí z těchto fragmentů logiky prvního řádu, a použijeme existující SAT řešič a počítače vážených modelů na porovnání jejich výkonu s polynomiálními metodami. Připravíme sadu sentencí, na kterých testujeme efektivitu polynomiálních i nepolynomiálních metod a prezentujeme výsledky experimentů na této sadě. Nakonec těmito výsledky podložíme, že nejmodernější polynomiální metody jsou mnohonásobně výkonnější než nepolynomiální metody.

Klíčová slova: počítání vážených modelů v logice prvního řádu, počítání vážených modelů, grounding, omezení kardinality, logika prvního řádu, Herbrandovská semantika

Překlad názvu: Počítání vážených modelů pro některé doménově-liftovatelné jazyky

Contents

Figures

Tables

Chapter 1

Introduction

In this work, we tackle the problem of Weighted First-Order Model Counting via grounding the formula and counting its models using a model counter or iterative calls to a SAT solver. Weighted First-Order Model Counting is an interesting problem in computer science as it finds many applications in computer science. Some applications of WFOMC are probabilistic inference in Markov Logic Networks [Beame et al., 2015], enumeration problems in combinatorics, and discovering interesting integer sequences [Svatoš et al., 2023] since all these problems can be reduced to WFOMC. Computing WFOMC in certain fragments of first-order logic is possible to do in time polynomial in the size of the domain. We call these fragments *domain-liftable*, and the methods that compute WFOMC in time polynomial in the size of the domain *lifted methods*. It has been proven that the first-order logic fragment with at most 2 variables, denoted $\mathbf{FO}^2$, is *domain-liftable* [Van den Broeck et al., 2014], that the first-order logic fragment with at most 2 variables and cardinality constraints, denoted $\mathbf{FO}^2 + \mathbf{CC}$, and the first-order logic fragment with at most 2 variables and counting quantifiers, denoted $\mathbf{C}^2$, are *domain-liftable* [Kuželka, 2021], and also the first-order logic fragment with at most 2 variables, counting quantifiers, and linear order axiom, denoted $\mathbf{C^2} + \mathbf{Linear}$ is *domain-liftable* [Tóth and Kuželka, 2023]. These qualities together make WFOMC a tool, that can be used to effectively solve many practical problems. While much effort has been put into developing lifted methods capable of solving WFOMC, there is little to no naive baseline for the $\mathbf{FO^2} + \mathbf{CC}$, $\mathbf{C^2}$ and $\mathbf{C^2} + \mathbf{Linear}$ fragments to compare the effectiveness of these methods to. In this thesis, we aim to provide such a baseline along with a dataset to compare lifted methods with a naive baseline and also aim to show the necessity for lifted methods, should WFOMC be used to tackle some of the problems mentioned previously.

Aside from providing a baseline to compare lifted methods to, the naive methods can compute the weighted model count of formulas from any logical fragment, albeit it is extremely time-consuming to solve the problem for even very small domains. We do not explore the usability of the method presented in this thesis to solve problems beyond the fragments mentioned above.

Chapter 2

Preliminaries

In this chapter, we intend to familiarize the reader with most of the background we will use in this work.

2.1 First-Order Logic

In this work, we consider a function-free subset of first-order logic. It consists of a finite set of constants Δ, a finite set of variables V, and a finite set of predicates P. An atom is a tuple of a predicate and a number of constants from Δ and variables from V equal to its arity. A literal is an atom or its negation. A formula consists of a literal, or more complex formulas and their negations connected by logical connectives. It can also be quantified by an existential quantifier $\exists x$ or a universal quantifier $\forall x$. A formula in which all variables are quantified is called a sentence. A formula that contains no variables is called ground. In this work, we expect all formulas to be fully quantified, as we can only ground sentences. Therefore, any time we mention a formula, we mean a sentence.

We adopt the Herbrand semantics [Hinrichs and Genesereth, 2006] with a finite domain. We consider a finite Herbrand Universe HU of all constants and a Herbrand Base HB of all ground atoms. There exists a bijection between the Herbrand Universe and the set of constants, so we will be denoting the Hrebrand Universe Δ. A possible world is any subset of HB, with all ground atoms present in the possible world considered to be true and all ground atoms not present in the possible world considered to be false. A possible world is a model of a formula if the formula is satisfied in the possible world.

We also use the tautological equivalence symbol $\boxminus$. A fromula Φ and a formula Ψ are tautologically equivalent, denoted $\Phi \boxminus \Psi$, if and only if for any possible world ω if holds that, if $\omega \models \Phi$ then $\omega \vDash \Psi$, and if $\omega \not\models \Phi$ then $\omega \not\models \Psi$.

2.2 Tseytin Transformation

Tseytin transformation [Tseitin, 1983] is used to produce a formula in conjunctive normal form that is linear in size of the original formula. It works by replacing subformulas in the original formula with new helper atoms and introducing rules for the new helper atoms such that an equisatisfiable formula in conjunctive normal form is created. To get the equisatisfiable formula in conjunctive normal form, we combine the rules we obtained during the transformation and also the last predicate we obtained. The rules for replacing subformulas usually are as follows:

$$Z \iff (P \wedge Q) \models\!\!\!\dashv (\neg P \vee \neg Q \vee Z) \wedge (P \vee \neg Z) \wedge (Q \vee \neg Z)$$
$$Z \iff (P \vee Q) \models\!\!\!\dashv (P \vee Q \vee \neg Z) \wedge (\neg P \vee Z) \wedge (\neg Q \vee Z)$$

With P, Q and Z being predicates of arity 0. Additional rules can be used, but these two are sufficient for our purposes, as we on use Tseytin transformation on formulas, where the only logical connectives are $\wedge$ and $\vee$. We can also expand upon these rules to get more efficient transformation as follows:

$$Z \iff \bigwedge_{i=1\ldots n} P_i \models\!\!\!\dashv (Z \bigvee_{i=1\ldots n} \neg P_i) \bigwedge_{i=1\ldots n} (\neg Z \vee P_i)$$
$$Z \iff \bigvee_{i=1\ldots n} P_i \models\!\!\!\dashv (\neg Z \bigvee_{i=1\ldots n} P_i) \bigwedge_{i=1\ldots n} (Z \vee (\neg P_i))$$

With Z and $P_1 \ldots P_n$ being predicates of arity 0. The reason for introducing these new rules is efficiency, as our structure allows combining conjunctions and disjunctions and we can introduce fewer new helper atoms this way.

Since the goal of our work is to count the models of an input sentence and Tseytin transformation introduces new helper atoms, we need to address how to handle a few things. First, when introducing new helper atoms, it is important to know whether these new helper atoms may lead to a change in the number of models of the new formula.

Definition 2.1 (Equicountablity)**.** We consider two formulas ϕ and ψ to be equicountable, if for a Herbrand Universe of any size, it holds that number of models of ϕ and ψ is the same.

We will use this extended definition of equicountability, as we work with first-order logic, rather than the definition in Kuiter et al. [2023], which defines equicountability for propositional logic.

Proposition 2.2. *Tseytin transformation is equicountable, meaning that the number of models does not change when its new helper atoms are introduced [Kuiter et al., 2023].*

Even though Tseytin in equicountable, we still monitor these new helper atoms and distinguish them from atoms original to the formula. When forbidding found solutions, we then only add original atoms to the clause that forbids repeated findings of an already found model. This is due to the longer clause being less informative and worse for the performance of the SAT solver. Also, by only caring about the original atoms when forbidding already found models, we prevent new helper atoms from adding new models, even if the adding of the formula with new helper atoms was not equicountable. Lastly, when solving with weights, we need to set the weights of the new helper atoms to one for positive weights and one for negative weights as that way, these new helper atoms do not change the resulting weighted model count.

2.3 SAT and #SAT Problems

Satisfiability problem (SAT) is a very well-known problem of deciding whether a formula in conjunctive normal form is satisfiable. This problem was proved to be NP-complete [Cook, 1971]. Since then, SAT solvers made huge progress and can solve very large SAT problems, and usually do so very fast. We use a SAT solver to solve a #SAT problem in this work, which is a counting problem of how many models of a formula exist as a subset of a Herbrand Base HB. The #SAT problem was introduced in Valiant [1979]. We define both these problems below.

Definition 2.3 (CNF). A propositional formula ϕ is in conjunctive normal form if it consists of a conjunction of subformulas χ_i, and all subformulas χ_i are either atoms, negations of atoms or disjunctions of atoms or their negations.

Definition 2.4 (SAT). A SAT or satisfiability problem is a decision problem whether a model of a formula ϕ in conjunctive normal form exists as a subset of the Herbrand Base HB.

Definition 2.5 (#SAT). A #SAT or Model Counting is the problem of how many models of a formula ϕ in conjunctive normal form exist as a subset of the Herbrand Base HB.

2.4 Weighted Model Counting and Weighted First-Order Model Counting

Weighted Model Counting (WMC) is a generalization of the #SAT problem. First, we need to mention, that WMC is a problem from propositional logic. While we deal with first-order logic in this work, after grounding a first-order sentence, we can consider each ground atom to be a proposition and use this technique to count the models of a first-order sentence. This is important as we will be using a WMC solver to count the weighted models of grounded first-order formulas later. We use the definition of WMC presented in Dilkas and Belle [2021]

Definition 2.6 (Weighted Model Counting). A WMC instance is a tuple (ϕ, X, w), where X is the set of literals, ϕ is a propositional formula in conjunctive normal form over X and $w : \{x|x \in X\} \cup \{\neg x|x \in X\} \to \mathbb{R}$ is a weight function. Then

$$WMC(\phi, X, w) = \sum_{\omega \models \phi} \prod_{x \models \omega} w(x) \prod_{x \not\models \omega} w(\neg x).$$

Weighted First-Order Model Counting (WFOMC) is a generalization of WMC to first-order logic. It is the problem we tackle in this work. It was first described by Broeck et al. [2011]. We use the definition of this problem that can be found below.

Definition 2.7 (Weighted First-Order Model Counting)**.** Let ϕ be a sentence over some relational language L. Let HB be a Herbrand Base of L over a domain of size $n \in \mathbb{N}$. Let P be a set of predicates of the language L and let $p : HB \to P$ map each atom to its predicate symbol. Let $w : P \to \mathbb{R}$ and $\bar{w} : P \to \mathbb{R}$ be weight functions assigning positive and negative weight to each predicate in L. Then

$$WFOMC(\phi, n, w, \bar{w}) = \sum_{\omega \subseteq HB, \omega \models \phi} \prod_{l \in \omega} w(p(l)) \prod_{l \notin \omega} \bar{w}(p(l)).$$

2.5 Logical Fragments

In mathematical logic, a fragment of a logical language or theory is a subset of this logical language obtained by imposing syntactical restrictions on the language [Bradley and Manna, 2007]. Focusing on a fragment of logic can make certain calculations tractable, which would otherwise be intractable. In this work, we mention several fragments of first-order logic.

First-order logic fragment with at most 2 variables, denoted $\mathbf{FO^2}$ is a fragment of first-order logic in which at most two variables are used. It should be noted that in this fragment any predicate of arity $n \geq 3, n \in \mathbb{N}$ can be replaced by a predicate of arity 2 as shown in Gradel and Vardi [1997].

Another fragment we work with is the first-order logic with at most 2 variables and cardinality constraints, denoted $\mathbf{FO^2} + \mathbf{CC}$. Cardinality constraints are an additional syntactic construct, used for example in Svatoš et al. [2023] to increase the expressional power of first-order logic. We use the following definition of cardinality constraints.

Definition 2.8 (Cardinality constraint)**.** A cardinality constraint is an expression in the form of $(|P| \bowtie k)$ where P is a predicate and $k \in \mathbb{N}$ and $\bowtie \in \{\leq, =, \geq\}$.

A possible world ω satisfies a given cardinality constraint $(|P| \bowtie k)$ if and only if the number of ground atoms $P_i(x_1, ..., x_a)$ that are true in the possible world is $\bowtie k$. A formula in $\mathbf{FO^2} + \mathbf{CC}$ takes the form of

$$\Phi = \Psi \wedge (|P| \bowtie k)$$

where Ψ is a formula from $\mathbf{FO^2}$, $k \in \mathbb{N}$ and $\bowtie \in \{\leq, =, \geq\}$.

Another fragment we work with is the first-order logic with at most 2 variables and counting quantifiers. Counting quantifiers are a generalization of the traditional existential quantifier. They add to the expressivity of the $\mathbf{FO^2}$ fragment and the fragment of first-order logic with at most two variables with counting quantifiers is usually denoted $\mathbf{C^2}$. The $\mathbf{C^2}$ fragment has more expressive power while still being computationally tractable, as was shown in Kuželka [2021].

Definition 2.9 (Counting Quantifier)**.** A counting quantifier is an expression in the form of $\phi = \exists^{\bowtie k} x\ \psi(x), k \in \mathbb{N}, \bowtie \in \{\leq, =, \geq, \neq\}$.

A possible world ω is a model of ϕ, if and only if there are exactly l constants $\{A_1, ..., A_l\}$ from a Herbrand Universe Δ such that $\omega \models \psi(A_i)$ if and only if $1 \leq i \leq l$ and $l \bowtie k$.

Another fragment we work with is the first-order logic with at most two variables and linear order axiom. This fragment is also domain-liftable, as is shown in Tóth and Kuželka [2023]. The linear order axiom enforces a total order, which means a reflexive, anti-symetric, transitive and strongly connected relation. In our case, linear order is such a relation applied to the Herbrand Universe Δ. An example of such a relation would be a predicate $\leq/2$. We then require the following to be true:

- $\forall x \ \leq(x, x)$
- $\forall x \forall y \ \leq(x, y) \lor \leq(y, x)$
- $\forall x \forall (y \ \leq(x, y) \land \leq(y, x)) \implies (x = y)$
- $\forall x \forall y \forall z \ (\leq(x, y) \land \leq(y, z)) \implies \leq(x, z)$

The last fragment we mention is the universally quantified first-order logic with at most 2 variables and cardinality constraints, denoted $\mathbf{UFO^2} + \mathbf{CC}$. This fragment consists of sentences in the form of $\forall x \forall y\ \Psi(x)$ and possibly some cardinality constraints.

To make it easier to describe possible worlds later in this work, we use the following definition of evaluation function. Evaluation function is often used in propositional logic, for example in Bailleux and Boufkhad [2003].

Definition 2.10 (Evaluation function)**.** Evaluation function $e_\omega : HB \mapsto \{0, 1\}$ is a function that maps a ground atom $A \in HB$ to values 0 if and only if the ground atom A is not true in a possible world ω and to 1 if and only if the ground atom A is true in a possible world ω.

2.6 WMC solvers

In this work, we want to compare the performance of various non-lifted methods of counting the weighted models of a first-order sentence to the lifted methods. The non-lifted methods work by grounding the formula and using a WMC solver to count the models of the grounded formula. There are several known methods of WMC solving.

One method is to iteratively call a SAT solver and count the models this way. We implement this method using the CryptoMiniSAT [Soos et al., 2009] solver. In Chapter 5, we can see that this approach does not scale very well, as it does not leverage any information about the WMC problem.

A different approach is to construct a support structure, usually a directed acyclic graph, from which the models can be counted. Both the solvers we use, [Muise et al., 2012], and Dudek et al. [2020], follow this approach. In

Chapter 5, we can see that this approach can outperform the naive iterative SAT-solving approach. However, it also does not scale past fairly small instances.

2.7 On-line Encyclopedia of Integer Sequences

OEIS [OEIS Foundation Inc., 2024] or The On-Line Encyclopedia of Integer Sequences is an online encyclopedia in which integer sequences are stored. Users can search for sequences by their number or by inputting a few numbers of a sequence. If the latter is used, OEIS shows up all integer sequences starting with the numbers the user puts into the search bar. Since we generate a sort of integer sequences when we count the modes of a sentence on Herbrand Universes of size $1, 2 \ldots n$, OEIS is a great tool to help us. If the sequence we generate by counting the models of a sentence appears in OEIS, we can confirm the meaning of the sentence we are counting the models of.

Chapter 3

Approach

For #SAT solving, we consider a Herbrand universe Δ of size n and a formula ϕ. We first create a Herbrand base HB. The size of this Herbrand base is in $O(p \cdot A^n)$ with $n = |\Delta|$, p being the number of predicates, and A being the highest arity among the predicates we consider. This potentially leads to an exponential blowup, but as long as we limit ourselves to the two variable fragment, any predicate of arity higher than two can be equivalently expressed by predicates of arity 2 at the cost of increasing the size of the formula [Gradel and Vardi, 1997]. This means that in the two variable fragment, the Herbrand base HB is polynomial in size with respect to $|\Delta|$. With the Herbrand base constructed, we ground the formula ϕ and either iteratively find solutions with a SAT solver or use a dedicated WMC solver to count the weighted models of the input formula. When using a SAT solver to count the models, with each model found, we add a clause that forces a different model to be found until the problem is no longer satisfiable. This process is finite as there are at most $2^{|HB|}$ solutions with $|HB|$ being the size of the Herbrand base. We repeat this process for Herbrand universes of size $1, 2, \ldots$ until searching for models is no longer feasible in a reasonable time. Since the number of possible worlds is $2^{|HB|}$, growing exponentially, we will not be experimenting past very small instances as we need the search to run in a reasonable time. The time we consider reasonable for gathering data in Chapter 5 for this work is five minutes. It is a time window long enough to gather information on what part of the program is taking too long and show the reason it is happening while also allowing the experiments to be reproduced and rerun without too much of a time burden. It should also be noted that Algorithm 1 only counts the models and not the weighted model count. This can be done by multiplying the positive or negative weights of each atom based on its presence in the model found. Since we already need to check the model for all atoms to forbid a model from being found again, it would only add a constant to the runtime with each model we find. To provide complete information, we also include Algorithm 2 that solves WMC rather than #SAT 2.

Algorithm 1 #Sat naive

```
input: grounded formula $\phi$ in CNF
output: number of models of $\phi$

$models \leftarrow 0$
while satisfiable $\phi$ do
    find model $\omega$ of $\phi$
    $\chi = (\bigvee_{L\in\omega} \neg L) \vee (\bigvee_{L\notin\omega} L)$
    $\phi \leftarrow \phi \wedge \chi$
    $models \leftarrow models + 1$
end while
return $models$
```

Algorithm 2 WMC solved iteratively

```
input: grounded formula $\phi$ in CNF, positive weighs $w$, negative weights $\overline{w}$
output: WMC of $\phi$

$wmc \leftarrow 0$
while satisfiable $\phi$ do
    find model $\omega$ of $\phi$
    $\chi = (\bigvee_{L\in\omega} \neg L) \vee (\bigvee_{L\notin\omega} L)$
    $\phi \leftarrow \phi \wedge \chi$
    $wmc \leftarrow wmc + \Pi_{L\in\omega} w(L) \cdot \Pi_{L\notin\omega} \overline{w}(L)$
end while
return $wmc$
```

3.1 Grounding Sentences

The idea our work follows is based around grounding a sentence in order to count its models. More precisely, we need to ground a sentence and obtain its conjunctive normal form, as most SAT and #SAT solvers expect their input to be a ground formula in conjunctive normal form. Since we work with a function-free subset of first-order logic, the approach to grounding consists of only four steps. We first remove any implications and equivalences in the formula, then redistribute negations, then ground all quantifiers that are present in the formula, and lastly use Tseytin transformation [Tseitin, 1983] to obtain a conjunctive normal form of the formula.

The grounding process starts with removing all implications and equivalences from the formula. Since our application allows chained implications and chained equivalences to be put together into a single subformula to save a bit of space, we use three rules to remove implications and equivalences:

$$(\Phi_1 \iff \Phi_2 \iff \dots \iff \Phi_k) \boxminus (\bigwedge_{i\in\mathbb{N},i\leq k} (\Phi_i) \vee \bigwedge_{i\in\mathbb{N},i\leq k} (\neg\Phi_i))$$

and

$$(\Phi \implies \Psi) \boxminus (\neg\Phi \vee \Psi)$$

and

$$(\Phi_1 \implies \Phi_2 \implies \ldots \implies \Phi_k) \boxminus (\neg\Phi_1 \vee (\bigwedge_{j \in \mathbb{N}, 2 \leq j \leq k} \Phi_j))$$

These rules can be used to remove all equivalences and implications in the input formula and once these are removed, no additional implications or equivalences are introduced. This reduces the number of rules that are needed later when using the Tseytin transformation to produce a formula in conjunctive normal form.

The next step is to remove all negations that are effecting a subformula other than a single atom. We use the following rules until all negations are pushed in front of an atom.

$$\neg\exists x\ \phi(x) \boxminus \forall x\ \neg(\phi(x))$$

$$\neg\forall x\ \phi(x) \boxminus \exists x\ \neg(\phi(x))$$

and the De Morgan Laws [Hurley, 2013]:

$$(\neg \bigwedge_{i \in \mathbb{N}} (\phi_i)) \boxminus (\bigvee_{i \in \mathbb{N}} (\neg\phi_i))$$

$$(\neg \bigvee_{i \in \mathbb{N}} (\phi_i)) \boxminus (\bigwedge_{i \in \mathbb{N}} (\neg\phi_i))$$

Using the rules shown above repeatedly, we can obtain a formula, that is logically equivalent to any input formula while only using conjunctions and disjunctions and with all negations effecting only atoms.

The next step is to ground the formula. Grounding is done from the outermost quantifier, making its way inwards until no quantifiers are left. The process consists of iterating 2 rules:

$$\exists x\ \phi \boxminus \bigvee_{A \in \Delta} \phi(x = A)$$

and

$$\forall x\ \phi \boxminus \bigwedge_{A \in \Delta} \phi(x = A)$$

where $=$ is unification of variable x with constant $A \in \Delta$. This is only possible because we use the Herbrand semantics. This leads to growth in the size of the formula proportional to $|\Delta|^q$ where $|\Delta|$ is the size of the Herbrand Universe and q is the number of nested quantifiers.

After these steps are finished, we need to produce a formula in conjunctive normal form, as most SAT solvers, #SAT solvers, and weighted model counters expect a formula in conjunctive normal form as their input. This could be done naively, by distributing atoms in conjunctions and disjunctions to create a formula in conjunctive normal form. Choosing the naive approach, we would use the distributive property of conjunctions and disjunctions:

$$(\Phi \land \Psi) \lor (\Gamma \lor \Lambda) \models\!\!\!\!\dashv (\Phi \lor \Gamma) \land (\Phi \lor \Lambda) \land (\Psi \lor \Gamma) \land (\Psi \lor \Lambda)$$

This, however, produces a formula that is exponential in size of the original formula. To avoid that, we use the Tseytin transformation Tseitin [1983], which produces a formula in conjunctive normal form that is linear in size of the original formula. Using this approach, we can produce a grounded formula in conjunctive normal form that is polynomial in the size of the original formula.

3.2 Cardinality Encodings

Cardinality constraints are a powerful tool to create more expressive fragment of first-order logic. Why we care about cardinality constraints is because the first-order logic fragment with at most two variables, $\mathbf{FO^2}$ enriched by cardinality constraints, usually denoted $\mathbf{FO^2} + \mathbf{CC}$ is computationally tractable as was shown in Kuželka [2021]. In this work, we need to ground cardinality constraints in order to be able to count the models of a sentence from the $\mathbf{FO^2} + \mathbf{CC}$ fragment. To make referencing atoms that we will be working with easier, we introduce a set $G = \{G_i = P(x_1, \ldots, x_a) \mid x_1, \ldots, x_a \in \Delta\}$ for some predicate P/a that occurres in cardinality constraint $|P| \bowtie k$.

There are many ways to ground a cardinality constraint and it is important to choose a way that suits the problem. One of the early ways to ground a cardinality constraint in the form of $|P| \leq k$ is to explicitly not allow any combinations of atoms equal to $k + 1$ to be true simultaneously [Ansótegui and Manyà, 2004]. This is achieved by adding the following to the formula in CNF:

$$\bigwedge_{H \subseteq G, |H| = k+1} \bigvee_{H_i \in H} \neg H_i$$

This approach requires no additional atoms to be introduced, but requires $\binom{|P|}{k+1}$ clauses. This is very useful for $k = 1$, as it only produces $|P| \cdot (|P| - 1)/2$ clauses, but it becomes inefficient very fast for any larger k. Since we also allow cardinality constraints in the form of $|P| \geq k$ and $|P| = k$, we also present how to encode those. For cardinality constraints in the form of $|P| \geq k$, for $k \geq 1$ we explicitly require that a combination of at least k atoms is true for any valid model. This is achieved by adding the following to the formula in CNF:

$$\bigwedge_{H \subseteq G, |H| = (|P| - k + 1)} \bigvee_{H_i \in H} H_i$$

This produces $\binom{|P|}{|P| - k + 1}$ new clauses, running into similar problem as the encoding for cardinality constraint in the form of $|P| \leq k$. Lastly, in the case of cardinality constraint in the form of $|P| = k$, we would treat it as both of the cardinality constraints in the forms of $|P| \leq k$ and $|P| \geq k$, which acts as a cardinality constraints $|P| = k$ (and also adds a large number of clauses to the formula in CNF). Since in this work, we expect to work with cardinality constraints with k greater than one, we do not consider such encoding as useful, since it does not preserve the polynomial size of the ground form we have kept so far. Even though such an encoding does not introduce any new helper atoms, it is not beneficial enough to use it over more refined encodings we introduce next.

When comparing cardinality constraint encodings, there are several factors to consider. One is the number of new clauses and helper atoms the encoding introduces to the formula, as a longer formula slows down the SAT solver search. Another criterion we can consider is the efficiency of the encoding. We base our definition of an efficient encoding on the definition presented in Bailleux and Boufkhad [2003]. However, as the original definition is used in propositional logic, we need to change it so that we can use it with Herbrand logic.

Definition 3.1 (Efficient encoding)**.** Let Δ be a Herbrand universe, P/a a predicate of arity a and $G = \{G_i = P(x_1, ..., x_a) | x_1, \dots, x_a \in \Delta\}$ be a set of all ground atoms constructed using the predicate P/a and atoms from the universe Δ. Then, for an encoding of a cardinality constraint to be efficient, it must hold that for any model ω:

- for an encoding of a cardinality constraint in the form of $|P| \leq k$, once we obtain the information that $e_\omega(G_i) = 1$ for a set of indices $I = \{i \in \mathbb{N} \mid i \leq |G|\}, |I| = k$, we get $e_\omega(G_j) = 0$ for any atom $G_j, j \notin I$ by unit propagations.
- for an encoding of a cardinality constraint in the form of $|P| \geq k$, once we obtain the information that $e_\omega(G_i) = 0$ for a set of indices $I = \{i \in \mathbb{N} \mid i \leq |G|\}, |I| = |G| - k$, we get $e_\omega(G_j) = 1$ for any atom $G_j, j \notin I$ by unit propagations.

We chose 2 encodings that differ in all of the criteria to test what is more important when counting models. The ladders encoding is efficient, but it introduces $O(nk)$ new helper atoms to the formula and it introduces $O(nk)$ new clauses to the formula. The parallel counter encoding is not efficient, but it only introduces $O(n)$ new helper atoms to the formula and it introduces $O(n)$ new clauses to the formula Ansótegui and Manyà [2004].

3.3 Ladders Encoding

Our ladder encoding is inspired by a ladder encoding described in Ansótegui and Manyà [2004]. However, while the encoding proposed in the article can only be used to resolve cardinality constraints in the form of $|P| \leq 1$, we modify it to work for any cardinality constraint in the form $|P| \leq k$, $|P| = k$, or $|P| \geq k$ with $k \in \mathbb{N}$. We call it ladder encoding because when encoding a cardinality constraint, we create a set of ladders and use it to enforce the cardinality constraint. We present an example of this to showcase this encoding before describing how we add it to the formula we want to ground.

Example 3.2. Consider a formula $\Phi = \Psi \wedge (|P| = 2)$ and a Herbrand universe $\Delta, |\Delta| = 4$, where P is a predicate of arity 1. We then add 10 new helper atoms $L_{1,1}, \ldots, L_{1,5}, L_{2,1}, \ldots L_{2,5}$, and use the following structure:

$e_\omega(P(1)) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$
$e_\omega(P(2)) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$
$e_\omega(P(3)) = 1$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$
$e_\omega(P(4)) = 0$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 1$
	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$

Table 3.1: Ladder Encoding Example

In this case, for a possible world to be a model, in order for an atom to have the evaluation $e_\omega(P(x)) = 1$, it needs to have a corresponding tuple $(e_\omega(L_{i,x}) = 0, e_\omega(L_{i,x+1}) = 1)$, for some index i. In other words, for some ladder consisting of helper atoms $L_{i,1}, \ldots, L_{i,n}$, the tuple $(e_\omega(L_{i,x}) = 0, e_\omega(L_{i,x+1}) = 1)$ is the deciding factor whether the corresponding atom $P(x)$ is true or not.

We first describe the ladder encoding we used as an inspiration. This encoding can only be used for cardinality constraints in the form of $|P| \leq 1$, but we modify it to work for any cardinality constraint in the form of $|P| \bowtie k$. To create a ladder encoding of a cardinality constraint in the form of $|P| \leq 1$, we do the following. First, we create a set of new atoms $L = \{L_i \mid i \in \mathbb{N}, i \leq (|G| + 1)\}$. Next, we require that a set of clauses

$$\{\neg L_i \vee L_{i+1} \mid i \leq |G|\}$$

is appended to the formula. This set guarantees that in a model ω, there is at most a single occurrence of $e_\omega(L_i) = 0, e_\omega(L_{i+1}) = 1, i < |L|$. Lastly we require that two sets of clauses

$$\{\neg G_i \vee L_{i+1} \mid i \leq |G|\}$$

and

$$\{\neg G_i \vee \neg L_i \mid i \leq |G|\}$$

are also appended to the formula. These two sets guarantee that for any model ω it holds that $e_\omega(G_i) = 1$ is possible only if $e_\omega(L_i) = 0$ and $e_\omega(L_{i+1}) = 1$ which can only happen at most once.

Now, we propose a ladder encoding that allows cardinality constraints in the form of $|P| \bowtie k,\ k \in \mathbb{N},\ \bowtie \in \{\leq, =, \geq\}$. We assume a set of $(n+1) \cdot k$ new atoms $L = \{L_{j,i} \mid j \in \mathbb{N}, j \leq k, i \in \mathbb{N}, i \leq |G| + 1\}$ and require that for cardinality constraints in the form of $|P| \leq k$, we require that a possible world ω is a model only if

$$e_\omega(G_i) = 1 \implies (e_\omega(L_{j,i}) = 0 \wedge e_\omega(L_{j,i+1}) = 1)$$

for some $j \leq k$, where j denotes that this implication is true in ladder j. For a cardinality constraint in the form of $|P| \geq k$ we require a reversed implication

$$e_\omega(G_i) = 1 \impliedby (e_\omega(L_{j,i}) = 0 \wedge e_\omega(L_{j,i+1}) = 1)$$

for any possible world ω to be a model. Lastly, for a cardinality constraint in the form of $|P| = k$, we combine both implications.

This could be easily encoded using the formulas:

$$G_i \implies \bigvee_{j \leq k} (\neg L_{j,i} \wedge L_{j,i+1})$$

respectively

$$G_i \impliedby \bigvee_{j \leq k} (\neg L_{j,i} \wedge L_{j,i+1})$$

However, the first formula in such an encoding would present 2 problems. Firstly, it would not be efficient. It would also either lead to an exponential blowup if we distributed it to obtain a formula in conjunctive normal form, or would require us to use Tseytin transform to obtain a formula in conjunctive normal form. Since neither approach is desirable, we use a different set of claues to encode the ladder encoding that forces the same requirements.

Here, we present the clauses that form this encoding. We present both a disjunction and implication form of the clauses for clarity.

$$\{\neg L_{i,1} \mid \forall i \leq k\} \tag{3.1}$$

This set of unit clauses forces all helper atoms at the bottom of a ladder to be false. We use this alongside Clauses 3.2 to guarantee that in each ladder, the sequence of $e_\omega(L_{j,i}) = 0, e_\omega(L_{j,i+1}) = 1$ appears.

$$\{L_{i,n+1} \mid \forall i \leq k\} \tag{3.2}$$

This set of unit clauses forces all helper atoms at the top of a ladder to be true. We use this alongside Clauses 3.1 to guarantee that in each ladder, the sequence of $e_\omega(L_{j,i}) = 0, e_\omega(L_{j,i+1}) = 1$ appears.

$$\begin{gathered}\{\neg L_{j,i} \vee L_{j,i+1} \mid \forall j \leq k, \forall i \leq n\} \\ \{L_{j,i} \implies L_{j,i+1} \mid \forall j \leq k, \forall i \leq n\}\end{gathered} \tag{3.3}$$

This set of clauses guarantees that in each ladder, a sequence of $e_\omega(L_{j,i}) = 0, e_\omega(L_{j,i+1}) = 1$ is found at most once (and alongside Clauses 3.1 and 3.2 exactly once).

$$\begin{gathered}\{L_{j,i} \vee \neg L_{j,i+1} \vee \neg L_{j+1,i+1} \mid \forall j \leq k-1, \forall i \leq n\} \\ \{(\neg L_{j,i} \wedge L_{j,i+1}) \implies \neg L_{j+1,i+1} \mid \forall j \leq k-1, \forall i \leq n\}\end{gathered} \tag{3.4}$$

This set of clauses forces the sequence of $e_\omega(L_{j,i}) = 0, e_\omega(L_{j,i+1}) = 1$ to happen in each ladder j at a different index i_j. This is potentially unnecessary for cardinality constraints in the form of $|P| \leq k$, but we use it for all cardinality constraints nonetheless.

$$\begin{gathered}\{\neg L_{j+1,i} \vee L_{j,i} \mid \forall j \leq k-1, \forall i \leq n+1\} \\ \{L_{j+1,i} \implies L_{j,i} \mid \forall j \leq k-1, \forall i \leq n+1\}\end{gathered} \tag{3.5}$$

This set of clauses guarantees that the sequence $e_\omega(L_{j,i}) = 0, e_\omega(L_{j,i+1}) = 1$ is found in lower index j for a lower index i. This allows for easier handling of the ladders.

$$\begin{gathered}\{\neg G_i \lor \neg L_{k,i} \mid \forall i \leq n\} \\ \{G_i \implies \neg L_{k,i} \mid \forall i \leq n\}\end{gathered} \tag{3.6}$$

This set of clauses is only used for cardinality constraints in the form of $|P| \leq k$ or $|P| = k$. Due to the structure we forced on the ladders by Clauses 3.1 to 3.5, each atom G_i can only be true if $L_{k,i}$ is not true.

$$\begin{gathered}\{\neg G_i \lor L_{1,i+1} | \forall i \leq n\} \\ \{G_i \implies L_{1,i+1} | \forall i \leq n\}\end{gathered} \tag{3.7}$$

This set of clauses is only used for cardinality constraints in the form of $|P| \leq k$ or $|P| = k$. Due to the structure we forced on the ladders by Clauses 3.1 to 3.5, each atom G_i can only be true if $L_{1,i+1}$ is true.

$$\begin{gathered}\{\neg G_i \lor \neg L_{j,i} \lor L_{j+1,i+1} \mid \forall i \leq n, \forall j \leq k-1\} \\ \{(L_{j,i} \land \neg L_{j+1,i+1}) \implies \neg G_i \mid \forall i \leq n, \forall j \leq k-1\}\end{gathered} \tag{3.8}$$

This set of clauses is only used for cardinality constraints in the form of $|P| \leq k$ or $|P| = k$. Intuition behind this set of clauses is as follows. In each ladder for G_i, one of three possible outcomes happens. $e_\omega(L_{j,i+1}) = 1$ and $e_\omega(L_{j,i}) = 0$, allowing $e_\omega(G_i) = 1$. Or $e_\omega(L_{j,i+1}) = 0$, meaning G_i has to be allowed in a ladder with lower j. Or $e_\omega(L_{j,i}) = 1$, meaning G_i has to be allowed in a ladder with greater j. In case $e_\omega(L_{j+1,i+1}) = 0$ and $e_\omega(L_{j,i}) = 1$, G_i would need to be allowed by ladder $l \in \mathbb{N}, j < l < j+1$. No such l exists hence $e_\omega(G_i) = 0$.

$$\begin{gathered}\{G_i \lor L_{j,i} \lor \neg L_{j,i+1} \mid \forall i \leq n, \forall j \leq k\} \\ \{(\neg L_{j,i} \land L_{j,i+1}) \implies G_i \mid \forall i \leq n, \forall j \leq k\}\end{gathered} \tag{3.9}$$

This set of clauses is only used for cardinality constraints in the form of $|P| \geq k$ and $|P| = k$. This clause is very simple, each sequence of $e_\omega(L_{j,i+1}) = 1$ and $e_\omega(L_{j,i}) = 0$ requires that $e_\omega(G_i) = 1$. Since we know k different such sequences are found in the ladders, we guarantee at least k indexes such that $e_\omega(G_i) = 1$ exist.

Now, we prove that the ladder encoding is efficient and provide a worked example. We start by proving that the encoding of cardinality constraint in the form of $|P| \leq k$ is efficient.

Proof. Suppose a set G of ground atoms produced from predicate P/a, a cardinality constraint $|P| \leq k$, a set of indices $I \subseteq \{1, 2, \ldots, |G|\}$, $|I| = k$, and a set of helper atoms L encoding a ladder encoding. Suppose a possible world in which $e_\omega(G_i) = 1$ for all $i \in I$. Then we need to be able to obtain $e_\omega(G_j) = 0$ for all $j \notin I$ only from unit clauses. We provide an example in which $|G| = 6$, $I = \{2, 4, 5\}$. We start by using all the unit clauses from Clauses 3.1 and 3.2 to obtain $e_\omega(L_{i,|G|+1}) = 1$ and $e_\omega(L_{i,1}) = 0$ for all $1 \le i \le k$. A partial possible world from our example can be seen in Table 3.2. In a partial possible world, we might not know the truth values of some atoms. We fill in truth values of all the atoms progressively as we obtain them from unit clauses.

$e_\omega(G_1) =$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) = 1$	$e_\omega(L_{1,2}) =$	$e_\omega(L_{2,2}) =$	$e_\omega(L_{3,2}) =$
$e_\omega(G_3) =$	$e_\omega(L_{1,3}) =$	$e_\omega(L_{2,3}) =$	$e_\omega(L_{3,3}) =$
$e_\omega(G_4) = 1$	$e_\omega(L_{1,4}) =$	$e_\omega(L_{2,4}) =$	$e_\omega(L_{3,4}) =$
$e_\omega(G_5) = 1$	$e_\omega(L_{1,5}) =$	$e_\omega(L_{2,5}) =$	$e_\omega(L_{3,5}) =$
$e_\omega(G_6) =$	$e_\omega(L_{1,6}) =$	$e_\omega(L_{2,6}) =$	$e_\omega(L_{3,6}) =$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.2: Efficiency Proof Part 1, Step 1

Next, we can use Clauses 3.6 and 3.3 to obtain $e_\omega(L_{1,i+1}) = 1$ for all $i \ge min(I)$. We can also use Clauses 3.7 and 3.3 to obtain $e_\omega(L_{k,i}) = 0$ for all $i \le max(I)$. In our example, what we know can be seen in Table 3.3

$e_\omega(G_1) =$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) = 1$	$e_\omega(L_{1,2}) =$	$e_\omega(L_{2,2}) =$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) = 0$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) =$	$e_\omega(L_{3,3}) = 0$
$e_\omega(G_4) = 1$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) =$	$e_\omega(L_{3,4}) = 0$
$e_\omega(G_5) = 1$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) =$	$e_\omega(L_{3,5}) = 0$
$e_\omega(G_6) =$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) =$	$e_\omega(L_{3,6}) =$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.3: Efficiency Proof Part 1, Step 2

Next, we can use Clauses 3.8 and 3.3 to obtain $e_\omega(L_{2,i+1}) = 1$ for all $i \ge j$, j is the *second* lowest element from I. In our example, we use Clause 3.8 in which $e_\omega(G_4) = 1$ and $e_\omega(L_{1,4}) = 1$, which we knew already, gives us a unit clause that requires $e_\omega(L_{2,5}) = 1$. This can be seen in Table 3.4

$e_\omega(G_1) =$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) = 1$	$e_\omega(L_{1,2}) =$	$e_\omega(L_{2,2}) =$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) =$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) =$	$e_\omega(L_{3,3}) = 0$
$e_\omega(G_4) = 1$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) =$	$e_\omega(L_{3,4}) = 0$
$e_\omega(G_5) = 1$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$	$e_\omega(L_{3,5}) = 0$
$e_\omega(G_6) =$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$	$e_\omega(L_{3,6}) =$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.4: Efficiency Proof Part 1, Step 3

This can be repeated l times, using Clauses 3.8 and 3.3 to obtain that

$e_\omega(L_{l,i+1}) = 1$ for all $i \geq j$, j is the *l-th* lowest element from I. In our example, we repeat this one time and obtain what can be seen in Table 3.5

$e_\omega(G_1) =$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) = 1$	$e_\omega(L_{1,2}) =$	$e_\omega(L_{2,2}) =$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) =$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) =$	$e_\omega(L_{3,3}) = 0$
$e_\omega(G_4) = 1$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) =$	$e_\omega(L_{3,4}) = 0$
$e_\omega(G_5) = 1$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$	$e_\omega(L_{3,5}) = 0$
$e_\omega(G_6) =$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$	$e_\omega(L_{3,6}) = 1$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.5: Efficiency Proof Part 1, Step 4

Next, we again use Clauses 3.8 and 3.3 to obtain that $e_\omega(L_{k-1,i}) = 0$ for all $i \leq j$, where j is the 2nd highest element from I. In our example, we use Clause 3.8 and from $e_\omega(G_4) = 1$ and $e_\omega(L_{3,5}) = 0$ we obtain a unit clause that gives us $e_\omega(L_{2,4}) = 0$. This can be seen in Table 3.6

$e_\omega(G_1) =$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) = 1$	$e_\omega(L_{1,2}) =$	$e_\omega(L_{2,2}) = 0$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) =$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$	$e_\omega(L_{3,3}) = 0$
$e_\omega(G_4) = 1$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 0$	$e_\omega(L_{3,4}) = 0$
$e_\omega(G_5) = 1$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$	$e_\omega(L_{3,5}) = 0$
$e_\omega(G_6) =$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$	$e_\omega(L_{3,6}) = 1$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.6: Efficiency Proof Part 1, Step 5

We can again repeat this process, which allows us to use Clauses 3.8 and 3.3 to obtain that $e_\omega(L_{k-l,i}) = 0$ for all $i \leq j$, where j is the *l-th* highest element from I. Using this in our example, we obtain truth values of all the helper atoms and can use Clauses 3.1, 3.2, and 3.8 to obtain truth values of all atoms in G. The final evaluation of our example can then be seen in Table 3.7 Therefore, we obtained that $e_\omega(G_i) = 0$, $\forall\, i \notin I$ only using unit

$e_\omega(G_1) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) = 0$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$	$e_\omega(L_{3,3}) = 0$
$e_\omega(G_4) = 1$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 0$	$e_\omega(L_{3,4}) = 0$
$e_\omega(G_5) = 1$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$	$e_\omega(L_{3,5}) = 0$
$e_\omega(G_6) = 0$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$	$e_\omega(L_{3,6}) = 1$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.7: Efficiency Proof Part 1, Step 6

clauses. □

We also need to prove that the ladder encoding is efficient for cardinality constraints $|P| \geq k$.

Proof. Suppose a set G of ground atoms produced from predicate P/a, a cardinality constraint $|P| \geq k$, a set of indices $I \subseteq \{1, 2, \dots, |G|\},\ |I| = |G| - k$, and a set of helper atoms L encoding a ladder encoding. Suppose a possible world in which $e_\omega(G_i) = 0$ for all $i \in I$. Then we need to be able to obtain $e_\omega(G_j) = 1$ for all $j \notin I$ only from unit clauses. We provide an example in which $|G| = 6,\ k = \{1, 3, 6\}$. We start by using all the unit clauses from Clauses 3.1 and 3.2 to obtain $e_\omega(L_{i,|G|+1}) = 1$ and $e_\omega(L_{i,1}) = 0$ for all $1 \leq i \leq k$. A possible world from our example can be seen in Table 3.8.

$e_\omega(G_1) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) =$	$e_\omega(L_{1,2}) =$	$e_\omega(L_{2,2}) =$	$e_\omega(L_{3,2}) =$
$e_\omega(G_3) = 0$	$e_\omega(L_{1,3}) =$	$e_\omega(L_{2,3}) =$	$e_\omega(L_{3,3}) =$
$e_\omega(G_4) =$	$e_\omega(L_{1,4}) =$	$e_\omega(L_{2,4}) =$	$e_\omega(L_{3,4}) =$
$e_\omega(G_5) =$	$e_\omega(L_{1,5}) =$	$e_\omega(L_{2,5}) =$	$e_\omega(L_{3,5}) =$
$e_\omega(G_6) = 0$	$e_\omega(L_{1,6}) =$	$e_\omega(L_{2,6}) =$	$e_\omega(L_{3,6}) =$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.8: Efficiency Proof Part 2, Step 1

Next, we use all unit clauses obtained from Clauses 3.9, to obtain from $e_\omega(G_i) = 0$ that $e_\omega(L_{i,j}) = e_\omega(L_{i,j+1)}$ for all i, j, where we know the truth value of either $L_{i,j}$ or $L_{L_{i,j+1}}$. In our example, this gives the information that can be seen in Table 3.9.

$e_\omega(G_1) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) =$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) = 0$	$e_\omega(L_{1,3}) =$	$e_\omega(L_{2,3}) =$	$e_\omega(L_{3,3}) =$
$e_\omega(G_4) =$	$e_\omega(L_{1,4}) =$	$e_\omega(L_{2,4}) =$	$e_\omega(L_{3,4}) =$
$e_\omega(G_5) =$	$e_\omega(L_{1,5}) =$	$e_\omega(L_{2,5}) =$	$e_\omega(L_{3,5}) =$
$e_\omega(G_6) = 0$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$	$e_\omega(L_{3,6}) = 1$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.9: Efficiency Proof Part 2, Step 2

When there are no unit clauses from Clauses 3.9, we use unit clauses produced from Clauses 3.4. In our example, we obtain $e_\omega(L_{1,5}) = 1$ from $e_\omega(L_{1,6}) = 1$ and $e_\omega(L_{2,6}) = 1$, and $e_\omega(L_{2,5}) = 1$ from $e_\omega(L_{2,6}) = 1$ and $e_\omega(L_{3,6}) = 1$. In our example, we also use the fact that $e_\omega(L_{1,5}) = 1$ and $e_\omega(L_{2,5}) = 1$ and use that to get $e_\omega(L_{1,4}) = 1$ via the unit clause from Clause 3.9 and get $e_\omega(L_{1,3}) = 1$. This can be seen in Table 3.10.

Now we can use Clauses 3.9 to obtain that $e_\omega(G_2) = 1$. Also, we can use Clauses 3.4 to obtain $e_\omega(L_{2,3}) = 0$. We can see the usage of that (along with some use of Clauses 3.3 and 3.5) in Table 3.11.

Last, we use Clauses 3.9 and 3.4 to obtain all the truth values. This can be seen in Table 3.12. This shows that by only using unit clauses, we can obtain all the truth values for all the atoms. Therefore, the ladder encoding of cardinality constraints is efficient. □

$e_\omega(G_1) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) =$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) = 0$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) =$	$e_\omega(L_{3,3}) =$
$e_\omega(G_4) =$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) =$	$e_\omega(L_{3,4}) =$
$e_\omega(G_5) =$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$	$e_\omega(L_{3,5}) =$
$e_\omega(G_6) = 0$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$	$e_\omega(L_{3,6}) = 1$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.10: Efficiency Proof Part 2, Step 3

$e_\omega(G_1) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) = 0$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$	$e_\omega(L_{3,3}) = 0$
$e_\omega(G_4) =$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) =$	$e_\omega(L_{3,4}) =$
$e_\omega(G_5) =$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$	$e_\omega(L_{3,5}) =$
$e_\omega(G_6) = 0$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$	$e_\omega(L_{3,6}) = 1$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.11: Efficiency Proof Part 2, Step 4

$e_\omega(G_1) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$	$e_\omega(L_{3,1}) = 0$
$e_\omega(G_2) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$	$e_\omega(L_{3,2}) = 0$
$e_\omega(G_3) = 0$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$	$e_\omega(L_{3,3}) = 0$
$e_\omega(G_4) = 1$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 0$	$e_\omega(L_{3,4}) = 0$
$e_\omega(G_5) = 1$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$	$e_\omega(L_{3,5}) = 0$
$e_\omega(G_6) = 0$	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$	$e_\omega(L_{3,6}) = 1$
	$e_\omega(L_{1,7}) = 1$	$e_\omega(L_{2,7}) = 1$	$e_\omega(L_{3,7}) = 1$

Table 3.12: Efficiency Proof Part 2, Step 5

One problem the ladders encoding presents is that for any cardinality constraint other than one in the form of $|P| = k$, which means for cardinality constraints in the form of $|P| \leq k$ and $|P| \geq k$, the formula resulting from adding the clauses to encode the cardinality constraint is not equicountable with the original formula. This is because for these cardinality constraints, the helper atoms in the ladders we introduce create more models with the same original atoms as we show in Examples 3.5 and 3.6 We solve this problem by only counting solutions where at least one truth value of an atom from the original formula is different. We cannot to do this with dedicated WMC solvers though, so we must not use this encoding for WMC solvers unless for cardinality constraints in the form of $|P| = k$.

Lemma 3.3. *When grounding cardinality constraints and introducing new helper atoms, we can show equicountablity as follows.*

Proof Sketch 3.1 (Lemma 3.3)**.** *Consider a formula:*

$$\phi = \psi \wedge (|P_i| \leq k)$$

where ψ is a formula in conjunctive normal form and a formula:

$$\phi' = \psi \wedge \chi$$

where χ is a formula in conjunctive normal form encoding the cardinality constraint $(|P| \leq k)$. Consider a set $\Omega = \{\omega_i | \omega_i \models \phi\}$ and a set $\Omega' = \{\omega'_i | \omega'_i \models \phi'\}$. Then we can use a bijective proof to show that $|\Omega| = |\Omega'|$. If the cardinality constraint encoding is correct, we can suppose that the mapping $f^{-1} : \Omega' \mapsto \Omega$ is trivial, as to obtain ω from ω', we simply remove all ground atoms, that do not appear in ψ from ω' and obtain ω. Therefore, when proving equisatisfiablity using a bijective proof, we will only show a mapping $f : \Omega \mapsto \Omega'$, supposing the inverse is created trivially as shown above.

The idea of the presented lemma is that as long as we can show that for any constellation of atoms original to formula ϕ that is a model, only one constellation of new helper and original atoms that is a model of ϕ' exists. We can show that is the case for ladders encoding of cardinality constraints in the form of $|P| = k$, and provide counterexamples for cardinality constraints in the form of $|P| \leq k$ and $|P| \geq k$.

We start by showing that for a cardinality constraint in the form of $|P| = k$ the ladders encoding produces an equicountable formula after grounding.

Example 3.4. Consider a Herbrand universe Δ, $|\Delta| = 5$, and a simple formula

$$\phi = |P| = 2.$$

One model ω can be seen in Table 3.13.

We can show that

$$e_\omega(P(2)) = 1 \rightsquigarrow e_\omega(L_{1,3}) = 1,\ e_\omega(L_{1,4}) = 1,\ e_\omega(L_{1,5}) = 1,\ e_\omega(L_{1,6}) = 1$$

from Clauses 3.7 and 3.3.

$e_\omega(P(1)) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$
$e_\omega(P(2)) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$
$e_\omega(P(3)) = 1$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$
$e_\omega(P(4)) = 0$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 1$
$e_\omega(P(5)) = 0$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$
	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$

Table 3.13: Model Example, Ladder Encoding Equicountablity

Also,

$$e_\omega(P(3)) = 1 \rightsquigarrow e_\omega(L_{2,3}) = 0,\ e_\omega(L_{2,2}) = 0,\ e_\omega(L_{2,1}) = 0$$

from Clauses 3.6 and 3.3.

Then

$$(e_\omega(P(2)) = 1 \wedge e_\omega(L_{2,3}) = 0) \rightsquigarrow e_\omega(L_{1,2}) = 0,\ e_\omega(L_{1,1}) = 0$$

from Clauses 3.8 and 3.3.

Lastly

$$(e_\omega(P(3)) = 1 \wedge e_\omega(L_{1,3}) = 1) \rightsquigarrow e_\omega(L_{2,4}) = 1,\ e_\omega(L_{2,5}) = 1,\ e_\omega(L_{2,6}) = 1$$

again from Clauses 3.8 and 3.3.

This shows that only from the constellation of original atoms, we can get one and only one constellation of the new helper atoms. We do not present a proof of the existence of a mapping M from truth values of the original atoms to the truth values of the new helper atoms, as in Proof 3.3, we prove that for a cardinality constraint in the form of $|P| \leq k$ and only a partial information about the original atoms (including that there are k atoms with truth value one, that is important, however, for any cardinality constraint in the form of $|P| = k$, there need to be k atoms with truth value one), we get one and only one constellation of truth values of the new helper atoms.

Example 3.5. Here we provide a counter-example to show that ladders encoding of a cardinality constraint in the form $|P| \leq k$ is not equicountable to the original formula. Consider a Herbrand universe of size five and a simple formula

$$\phi = |P| \leq 2.$$

One model ω can be seen in Table 3.14.

Another model ω can be seen in Table 3.15.

It can be seen that there are new models introduced by adding new helper atoms.

Example 3.6. Here we provide a counter-example to show that ladders encoding of a cardinality constraint in the form $|P| \geq k$ is not equicountable to the original formula. Consider a Herbrand universe of size five and a simple formula

$$\phi = |P| \geq 2.$$

$e_\omega(P(1)) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$
$e_\omega(P(2)) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$
$e_\omega(P(3)) = 0$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$
$e_\omega(P(4)) = 0$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 1$
$e_\omega(P(5)) = 0$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$
	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$

Table 3.14: Ladder Encoding Equicountablity Counterexample 1, Model 1

$e_\omega(P(1)) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$
$e_\omega(P(2)) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$
$e_\omega(P(3)) = 0$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$
$e_\omega(P(4)) = 0$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 0$
$e_\omega(P(5)) = 0$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$
	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$

Table 3.15: Ladder Encoding Equicountablity Counterexample 1, Model 2

$e_\omega(P(1)) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$
$e_\omega(P(2)) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$
$e_\omega(P(3)) = 1$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$
$e_\omega(P(4)) = 1$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 1$
$e_\omega(P(5)) = 1$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$
	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$

Table 3.16: Ladder Encoding Equicountablity Counterexample 2, Model 1

$e_\omega(P(1)) = 0$	$e_\omega(L_{1,1}) = 0$	$e_\omega(L_{2,1}) = 0$
$e_\omega(P(2)) = 1$	$e_\omega(L_{1,2}) = 0$	$e_\omega(L_{2,2}) = 0$
$e_\omega(P(3)) = 1$	$e_\omega(L_{1,3}) = 1$	$e_\omega(L_{2,3}) = 0$
$e_\omega(P(4)) = 1$	$e_\omega(L_{1,4}) = 1$	$e_\omega(L_{2,4}) = 0$
$e_\omega(P(5)) = 1$	$e_\omega(L_{1,5}) = 1$	$e_\omega(L_{2,5}) = 1$
	$e_\omega(L_{1,6}) = 1$	$e_\omega(L_{2,6}) = 1$

Table 3.17: Ladder Encoding Equicountablity Counterexample 2, Model 2

One model ω can be seen in Table 3.16.

Another model ω can be seen in Table 3.17.

It can be seen that there are new models introduced by adding new helper atoms.

3.4 Parallel Counter Encoding

The next encoding we propose is a parallel counter encoding based on the parallel counter encoding presented in Ansótegui and Manyà [2004]. To encode a cardinality constraint using the parallel counter encoding, we need to construct a binary adder to add the input atoms and then construct a comparator circuit that compares the sum of the variables to the desired

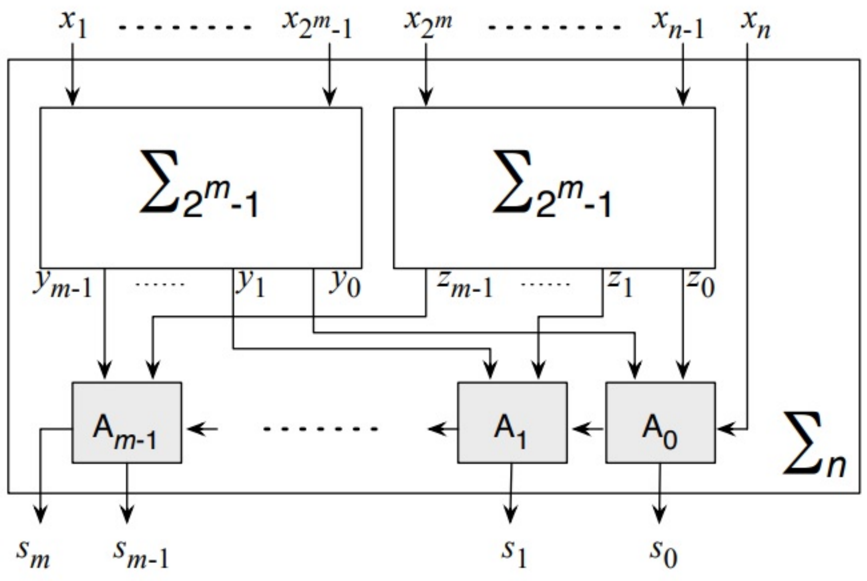

Figure 3.1: Parallel counter

k for a cardinality constraint $|P| \bowtie k$. However, the encoding presented in Ansótegui and Manyà [2004] only encodes cardinality constraints in the form of $|P| \leq k$. Therefore, we first present the encoding from Ansótegui and Manyà [2004] and then our expanded encoding for any cardinality constraint $|P| \bowtie k$.

In order to implement a parallel counter encoding, we need to first construct a binary adder. To construct the adder, we first recursively divide the input variables into halves. In Ansótegui and Manyà [2004], the number of input atoms is conveniently in the form of $2^m - 1, m \in \mathbb{N}$. Numbers in this form are easy to divide into halves for the adder as can be seen in Figure 3.1 with each recursion starting with a number of $2^{m-1} - 1$ input atoms. The paper does not include any information as how different numbers of input atoms are treated, but we show how we treat this procedure to incorporate any number of input atoms in our version later in this chapter. The recursion stops when three input atoms remain at which point a full binary adder is constructed. In case only two input atoms remain (which can happen if the number of input atoms is not in the form of $2^m - 1, m \in \mathbb{N}$), only a half adder is constructed. A half adder is constructed by introducing two new helper atoms, one for the sum of the adder and the other for the carry. The two input atoms entering the half adder will be called A and B in the clauses that are to be added and two new helper atoms will be created for each adder, Sum and $Carry$. The half adder then introduces three new clauses to the CNF formula:

- $A \lor B \lor Sum$
- $\neg A \lor B \lor Sum$
- $\neg A \lor \neg B \lor Carry$

In the case of a full adder, we consider three input atoms, called A, B, and C, and introduce two new helper atoms for each adder Sum and $Carry$. Then, the following seven clauses are added:

- $\neg A \lor B \lor C \lor Sum$
- $A \lor \neg B \lor C \lor Sum$
- $A \lor B \lor \neg C \lor Sum$

These three clauses work so that if one of the input atoms is true, then exactly two are true or Sum is true (or, since this is only one way implication, exactly two input atoms are true and Sum is also true).

- $\neg A \lor \neg B \lor \neg C \lor Sum$

This clause works so that if all three input atoms are true, Sum is also true.

- $\neg A \lor \neg B \lor Carry$
- $\neg A \lor \neg C \lor Carry$
- $\neg B \lor \neg C \lor Carry$

These three clauses work so that if 2 or more input atoms are true, $Carry$ is also true.

As we can see, there are only implications to the Sum and Carry helper atoms, and no reversed implications. This is even mentioned in the original paper, saying that the implications above are the only ones needed. That is true, however, it is only true for cardinality constraints in the form of $|P| \leq k$. For the other forms we allow in this work, these implications are not sufficient.

After the adders that add the original atoms are created, the next layer of adders is created from newly introduced helper atoms. In each layer, a one bit longer output sum is created, by chaining adders of bits of the previous sums. The output of the full procedure is the sum of all the input atoms.

Once we obtain the full sum of the input atoms, we need to construct the comparator circuit. The comparator circuit for a cardinality constraint in the form of $|P| = k$ is the easiest, as we simply take a binary representation of the constraint number k and for each bit, we add a unit clause setting the truth value of the respective output sum bit to one if the bit in the binary representation of k is one and to zero if the binary representation bit is zero.

The comparator circuit for cardinality constraint in the form of $|P| \leq k$ consists of several clauses. To construct the less than or equal comparator circuit, we consider a list $L = (l_0, ..., l_m)$, where l_0 is the lowest bit of binary representation of the number k and l_m is the highest. We also consider the output sum of the parallel counter, a list $S = (s_0, ..., s_m)$. We consider these two lists to have the same number of elements m. In case either of the numbers has fewer bits in binary representations, all elements of the respective list at position $i > j$, where j is the highest bit in the binary representation of either number, are considered to be zero. The clauses added to form the comparator circuit then have the form of

$$\neg s_i \bigvee_{i<j<m, l_j=0} \neg s_j,$$

for all i such that $l_i = 0$.

The comparator circuit for a cardinality constraint in the form of $|P| \geq k$ is similar to the comparator circuit for a cardinality constraint in the form of $|P| \leq k$ described above. We again consider a list $L = (l_0, ..., l_m)$, and a list $S = (s_0, ..., s_m)$ of the sum of the output of the adder. If either list is smaller, we pad it as above. The clauses to be added to form the comparator circuit then have the form of

$$s_i \bigvee_{i<j<m, l_j=1} s_j,$$

for all i such that $l_i = 1$.

However, even though we describe these comparator circuits here, the version of the parallel counter encoding from Ansótegui and Manyà [2004] can only encode cardinality constraints in the form of $|P| \leq k$. That is because in the adders, only implications for the *Sum* and *Carry* are used, instead of full equivalences. This means that the adders only output an upper bound on the number of input atoms rather than their exact sum. This led to adding the reverse implications in our version, to allow for correctly constructing the parallel counter encoding for forms other than $|P| \leq k$. We describe the additional clauses alongside our way of splitting the input atoms into the adders below.

When splitting an arbitrary number of input atoms $n \in \mathbb{N}$, we first need to recursively split them until a base case is reached. In our work, base cases are any input of two input atoms to five input atoms. These numbers are not chosen at random, as if we treat all these numbers of input atoms as base cases, we never run into a split of only one input atom, which is what we are trying to prevent. Then, for an arbitrary number of input atoms n, we execute Algorithm 3. To prevent confusion, we use upper index to index different lists, lower index to index elements of lists and the + symbol means a concatenation of lists. The procedure `add_sums` that is used creates a chain of adders adding the two input list of atoms and using the last atom as a carry to input to the first adder, so it can be created as a full adder. The procedure `half_adder` creates a half adder adding two atoms and returning a list with a binary representation of the result. The procedure `full_adder` behaves similarly, taking three atoms and returning a list with a binary representation of the result.

Algorithm 3 Add atoms

```
input: list L of atoms to add
output: list representing a binary representation of the number of true input atoms

$n \leftarrow |L|$
if $n = 2$ then
    $sum \leftarrow$ half_adder($l_1, l_2$)
    return $sum$
end if
if $n = 3$ then
    $sum \leftarrow$ full_adder($l_1, l_2, l_3$)
    return $sum$
end if
if $n = 4$ then
    $subsum^1 \leftarrow$ half_adder($l_1, l_2$)
    $subsum^2 \leftarrow$ half_adder($l_3, l_4$)
    $subsum^3 \leftarrow$ half_adder($subsum^1_1, subsum^2_1$)
    $sum \leftarrow$ full_adder($subsum^1_2, subsum^2_2, subsum^3_2$) $+ subsum^3_1$
    return $sum$
end if
if $n = 5$ then
    $subsum^1 \leftarrow$ half_adder($l_1, l_2$)
    $subsum^2 \leftarrow$ half_adder($l_3, l_4$)
    $sum \leftarrow$ add_sums($subsum^1, subsum^2, l_5$)
    return $sum$
end if
$split \leftarrow argmax_{m \in \mathbb{N}}(2^m < n - 1)$
$split \leftarrow split - 1$
$subsum^1 \leftarrow$ add_atoms(($l_1, ..., l_{split}$))
$subsum^2 \leftarrow$ add_atoms(($l_{split+1}, ..., l_{n-1}$))
$sum \leftarrow$ add_sums($subsum^1, subsum^2, l_n$)
return $sum$
```

Another change we make to the cardinality constraint encoding presented in the original paper is that we require a full equation with the sum, rather than just a one way implication. As was said, this is required for cardinality constraints in the form of $|P| \geq k$ and in the form of $|P| = k$. Or rather, the full equality is only required for cardinality constraints in the form of $|P| = k$, while only the reversed implications not presented in the original paper would be required for a cardinality constraint in the form of $|P| \geq k$. We however always include the full implication, as it should have only a negligible effect on performance. So the half adder that adds two input atoms A and B in our work consists of two new helper atoms sum and carry and the following clauses:

- $A \lor \neg B \lor Sum$
- $\neg A \lor B \lor Sum$
- $\neg A \lor \neg B \lor \neg Sum$
- $A \lor B \lor \neg Sum$

These 4 clauses together deal with any possible combination of truth values of the input atoms and the resulting truth value of Sum.

- $\neg A \lor \neg B \lor Carry$
- $A \lor \neg Carry$
- $B \lor \neg Carry$

These 3 clauses specify the truth value of $Carry$. The first forces $Carry$ to be true if both the input atoms are true and the remaining 2 force $Carry$ to be false unless both the input atoms are true.

An adder in this form outputs the exact binary representation of the sum of the input atoms.

The full adder, which adds three input atoms A, B, and C again creates two new helper atoms, Sum, and $Carry$. Then it adds the following clauses:

- $\neg A \lor B \lor C \lor Sum$
- $A \lor \neg B \lor C \lor Sum$
- $A \lor B \lor \neg C \lor Sum$
- $\neg A \lor \neg B \lor \neg C \lor Sum$
- $\neg A \lor \neg B \lor C \lor \neg Sum$
- $\neg A \lor B \lor \neg C \lor \neg Sum$
- $A \lor \neg B \lor \neg C \lor \neg Sum$
- $A \lor B \lor C \lor \neg Sum$

These 8 clauses deal with any possible constellation of truth values of input atoms A, B, and C, and what the resulting truth value of Sum is. We need 8 clauses for that, as $2^3 = 8$, so there are 8 possible truth evaluations of 3 atoms.

- $\neg A \lor \neg B \lor Carry$
- $\neg A \lor \neg C \lor Carry$

- $\neg B \lor \neg C \lor Carry$
- $A \lor B \lor \neg Carry$
- $A \lor C \lor \neg Carry$
- $B \lor C \lor \neg Carry$

We only need 6 clauses for the $Carry$, as any combination of 2 true input atoms mean that $Carry$ is also true and any combination of 2 false input atoms mean that $Carry$ is also false.

This approach might seem inefficient and we could reduce the number of clauses by only using the implications that are required for the adder, either only the implications that lead to a positive sum and carry for the cardinality constraints in the form of $|P| \leq k$, or only the implications that lead to a negative sum and carry for the cardinality constraints in the form of $|P| \geq k$. However, in our case the effect on the speed with which we can search for models is minimal. Also, since for a cardinality constraint in the form of $|P| = k$, we still need to add all the clauses above.

When deciding whether to add full equivalences or only half equivalences, we should also consider if the encoding is equicountable. In the case of half equivalences, we can easily find an example showing that it is not equicountable.

Example 3.7. Consider a simple formula $\phi = \exists x P(x) \land |P| \leq 1$, where P/1 is a predicate and a Herbrand Universe $|\Delta| = 2$. Then we get a ground form from parallel counter encoding with half equivalnces:

$$(P(1) \lor P(2)) \land (P(1) \lor \neg P(2) \lor Sum) \land (\neg P(1) \lor P(2) \lor Sum) \land$$

$$(\neg P(1) \lor \neg P(2) \lor Carry) \land (\neg Carry)$$

where sum and carry are new helper predicates of arity zero. This formula has the following models:

$$\{\{\}, \{P(1), Sum\}, \{P(2), Sum\}, \{Sum\}\}$$

We can clearly see that the last model of $\{Sum\}$ is a model that was introduced by introducing new variables.

However, when we use full equivalences, we can show that by grounding a cardinality constraint we get a new equicountable formula.

Lemma 3.8.

$$\phi = \psi \land |P| \leq k$$

and a formula

$$\phi' = \psi \land \chi$$

where ψ is a formula in conjunctive normal form and χ is a parallel counter encoding of the cardinality constraint $|P| \bowtie k$ using full equivalences is equicountable.

Proof Sketch 3.2. *To show that ϕ and ϕ' are equicountable, according to Lemma 3.3, we need to show that a mapping $f : \Omega \mapsto \Omega'$ where $\Omega = \{\omega_i | \omega_i \models \phi\}$ and $\Omega' = \{\omega'_i | \omega'_i \models \phi'\}$ exists. A parallel counter encoding consists of half and full adders that add either input atoms or outputs of previous adders. Since we use full equivalences, we can show all possible outcomes for any adder. Consider a half adder with input atoms A and B and output atoms Sum and $Carry$. Then only the following sets are subsets of any model:*

$$\{\}, \{A, Sum\}, \{B, Sum\}, \{A, B, Carry\}$$

Now we consider a full adder with input atoms A, B, and C and output atoms Sum and $Carry$. Then, only the following sets are subsets of any model:

$$\{\}, \{A, Sum\}, \{B, Sum\}, \{C, Sum\},$$

$$\{A, B, Carry\},$$

$$\{A, C, Carry\}, \{B, C, Carry\},$$

$$\{A, B, C, Sum, Carry\}$$

Since we never introduce a new helper atom outside of the adders and each adder only allows one output for any input, we can see that any model $\omega \in \Omega$ maps to exactly one model from $\omega' \in \Omega'$.

3.5 Counting Quantifiers

Counting quantifiers are a generalization of the traditional existential quantifier. They add to the expressivity of the $\mathbf{FO^2}$ fragment and the fragment of first-order logic with at most two variables with counting quantifiers is usually denoted $\mathbf{C^2}$. The $\mathbf{C^2}$ fragment has more expressive power while still being computationally tractable, as was shown in Kuželka [2021]. While lifted methods focus mainly on the counting quantifier in the form of $\exists^{=k} x\psi(x), k \in \mathbb{N}$, and transform other forms of the counting quantifier to the form of $\exists^{=k} x\psi(x), k \in \mathbb{N}$, our approach of grounding the problem allows us to use any of the counting quantifiers from Definition 2.9. We will show that while grounding, it is much appreciated to be able to deal with any of the forms above and also show that once a solid infrastructure to ground cardinality constraints is established, grounding counting quantifiers is fairly simple.

When grounding counting quantifiers, we can use a simple rule that works as follows:

$$\exists^{=k}\ x\phi(x) \boxminus (\forall x\ (C(x) \iff \phi(x)) \wedge (|C| = k))$$

It is extremely important to mention that this rule can only be used if the counting quantifier is not preceded by any other quantifier or a negation. This approach also works for any other form of the counting quantifier as follows:

$$\exists^{\leq k} x\ \phi(x) \boxminus (\forall x\ (C(x) \iff \phi(x)) \wedge (|C| \leq k))$$

$$\exists^{\geq k} x\ \phi(x) \Vdash (\forall x\ (C(x) \iff \phi(x)) \wedge (|C| \geq k))$$

We use a special way to ground the counting quantifier $\exists^{\neq k}$

$$\exists^{\neq k} x\ \phi(x) \Vdash \forall x\ (((C(x) \iff \phi(x)) \wedge (|C| \geq k+1))$$
$$\vee((D(x) \iff \phi(x)) \wedge (|D| \leq k-1)))$$

We said previously that we do not introduce any new equivalences into the formula once we remove equivalences from the original in the beginning. This holds, as instead of the equivalence, we use a conjunction of two disjunctions. We provide an example below, just to be clear, all the equivalences added are handled the same way as in the example:

$$\exists^{\leq k} x\ \phi(x) \Vdash ((\forall x\ (\neg C(x) \vee \phi(x))) \wedge (\forall x\ (C(x) \vee \neg\phi(x))) \wedge (|C| \leq k))$$

However, when describing grounding the counting quantifiers, we will use the equivalence form as it is much more readable.

We said above that this approach of grounding only works when the quantifier is not preceded by a different quantifier or a negation. When a negation precedes a counting quantifier, we can use a simple rule of flipping its comparison sign as follows:

$$(\neg\exists^{=k} x\ \phi(x)) \Vdash (\exists^{\neq k} x\ \phi(x))$$
$$(\neg\exists^{\leq k} x\ \phi(x)) \Vdash (\exists^{\geq k+1} x\ \phi(x))$$
$$(\neg\exists^{\geq k} x\ \phi(x)) \Vdash (\exists^{\leq k-1} x\ \phi(x))$$
$$(\neg\exists^{\neq k} x\ \phi(x)) \Vdash (\exists^{=k} x\ \phi(x))$$

When a quantifier precedes a counting quantifier, we can ground it as follows:

$$(\forall x \exists^{=k} y\ \phi(x,y)) \Vdash (\bigwedge_{A_i \in \Delta} (\bigwedge_{B_i \in \Delta} (C_{A_i}(B_i) \iff \phi(A_i, B_i)) \wedge (|C_{A_i}| = k)))$$

where Δ is a Herbrand Universe and $C_{A_i}/1$ are $|\Delta|$ new predicates. It is extremely important to mind the correct placement of the parenthesis around subformulas, as even a slight misplacement might change the meaning of the formula. Similarly, we ground the existential quantifier preceding the counting quantifier:

$$(\exists x \exists^{=k} y\ \phi(x,y)) \Vdash (\bigvee_{A_i \in \Delta} (\bigwedge_{B_i \in \Delta} (C_{A_i}(B_i) \iff \phi(A_i, B_i)) \wedge (|C_{A_i}| = k)))$$

where Δ is a Herbrand Universe and $C_{A_i}/1$ are $|\Delta|$ new predicates. The most complicated constellation happens when two counting quantifiers are following one another. Then, the grounding looks as follows:

$$(\exists^{=l} x\ \exists^{=k} y \phi(x,y)) \Vdash$$
$$(\bigwedge_{A_i \in \Delta} (C(A_i) \iff (\bigwedge_{B_i \in \Delta} (D_{A_i}(B_i) \iff \phi(A_i, B_i))$$
$$\wedge(|D_{A_i}| = k))) \wedge (|C| = l))$$

We mentioned earlier that when grounding counting quantifiers, it is extremely important to mind the correct placement of parenthesis and to create the correct equivalence in the formula. We now show an example of how even a slight mistake can lead to an incorrect model count.

Example 3.9. Consider a sentence

$$\exists^{=1} x \forall y\ R(x,y)$$

in a Herbrand Universe Δ of size two, where R is a predicate of arity two. Then the correct grounding looks as follows:

$$\forall x\ (C(x) \iff \forall y\ R(x,y)) \land (|C| = 1)$$

which is grounded to

$$\bigwedge_{A \in \{1,2\}} (C(A) \iff \bigwedge_{B \in \{1,2\}} R(A,B)) \land (|C| = 1)$$

If we consider an encoding of the cardinality constraint that does not introduce any new atoms (such an encoding exists and was described previously), we can see that all models of such a sentence are:

$$\{\{C(1), R(1,1), R(1,2)\}, \{C(1), R(1,1), R(1,2), R(2,1)\},$$
$$\{C(1), R(1,1), R(1,2), R(2,2)\}, \{C(2), R(2,1), R(2,2)\},$$
$$\{C(2), R(1,1), R(2,1), R(2,2)\}, \{C(2), R(1,2), R(2,1), R(2,2)\}\}$$

Now we show what happens for an incorrect grounding where the universal quantifier is left out of the newly introduced equivalence:

$$\forall x \forall y\ (C(x) \iff R(x,y)) \land (|C| = 1)$$

which is grounded to

$$\bigwedge_{A \in \{1,2\}} \bigwedge_{B \in \{1,2\}} (C(A) \iff R(A,B)) \land (|C| = 1)$$

Then only two models of such a sentence exist:

$$\{\{C(1), R(1,1), R(1,2)\}, \{C(2), R(2,1), R(2,2)\}\}$$

This shows that we need to be very careful when grounding counting quantifiers in the way we do, to avoid making such a mistake. In a small example like the one presented, the mistake is easy to spot, however, for more complicated sentences, such a mistake might not be as easy to spot.

3.5.1 Lifted Approach to Counting Quantifiers

Another way to ground counting quantifiers is the way used in the lifted methods that solve WFOMC. This way uses a number of lemmas which are neatly and compactly aggregated in Tóth and Kuželka [2024]. Notice that the paper presents a conversion from $\mathbf{C}^2$ to $\mathbf{UFO^2} + \mathbf{CC}$. In this work, however, we do not require the formula to be universally quantified as long as the formula is a sentence. We now present the lemmas as they are presented in the paper.

Lemma 1 in the paper is a skolemization procedure first presented in Van den Broeck et al. [2014], and a modified version from Beame et al. [2015] is used.

Lemma 3.10. *Let $\Gamma = Q_1 x_1 Q_2 x_2, ..., Q_k x_k \phi(x_1, ..., x_k)$ be a first-order sentence in prenex normal form with each quantifier Q_i being either $\forall$ or $\exists$ and ϕ being a quantifier-free formula. Denote j the first position of $\exists$. Let $\mathbf{x} = (x_1, ..., x_{j-1})$ and $\phi(\mathbf{x}, x_j) = Q_{j+1} x_{j+1} ... Q_k x_k \phi$. Set*

$$\Gamma' = \forall \mathbf{x} \left((\exists x_j \ \phi(x, x_j)) \implies A(\mathbf{x}) \right)$$

where A is a fresh predicate. Then, for any $n \in \mathbb{N}$ and any weights $(w, \overline{w})$ with $w(A) = 1$ and $\overline{w}(A) = -1$, it holds that

$$WFOMC(\Gamma, n, w, \overline{w}) = WFOMC(\Gamma', n, w, \overline{w})$$

This lemma is needed to remove existential quantifiers. This is needed for lifted methods, but not for our grounding method.

Lemma 2 in the paper is a technique to remove negations without distributing them inside the formula. This lemma was presented in Beame et al. [2015]. This lemma could be used even when grounding to remove free variables.

Lemma 3.11. *Let $\neg\psi(\mathbf{x})$ be a subformula of a first-order logic sentence Γ with k free variables $\mathbf{x} = (x_1, ..., x_k)$ Let C/k and D/k be two fresh predicates with $w(C) = w(C) = w(D) = 1$ and $w(D) = -1$. Denote Γ' the formula obtained from Γ by replacing the subformula $\neg\psi(\mathbf{x})$ with $C(\mathbf{x})$. Let $\Upsilon = (\forall \mathbf{x} \, C(\mathbf{x}) \vee D(\mathbf{x})) \wedge (\forall \mathbf{x} \, C(\mathbf{x}) \vee \psi(\mathbf{x})) \wedge (\forall \mathbf{x} \, D(\mathbf{x}) \vee \psi(\mathbf{x}))$. Then, it holds that*

$$WFOMC(\Gamma, n, w, \overline{w}) = WFOMC(\Gamma' \wedge \Upsilon, n, w, \overline{w})$$

With these two lemmas, we can now start with how to remove counting quantifiers from a formula. Lemma 3 in the paper removes a single counting quantifier and was first presented in Kuželka [2021].

Lemma 3.12. *Let Γ be a first-order logic sentence. Let ψ be a $\mathbf{C}^2$ sentence such that $\psi = \exists^{=k} x R(x)$. Let $\psi' = (|R| = k)$ be a cardinality constraint. Then, it holds that*

$$WFOMC(\Gamma \wedge \psi, n, w, \overline{w}) = WFOMC(\Gamma \wedge \Psi', n, w, \overline{w})$$

Next lemma deals with a case of $\forall\exists^{=k}$. This lemma was originally presented in Kuželka [2021].

Lemma 3.13. *Let Γ be a first-order logic sentence. Let Ψ be a $\mathbf{C}^2$ sentence such that $\psi = \forall x \exists^{=k} \, R(x, y)$. Let Υ be an $\mathbf{FO}^2 + \mathbf{CC}$ sentence defined as*

$$\Upsilon = (|R| = k \cdot n)$$

$$\wedge(\forall x \forall y\ R(x,y) \iff \bigvee_{i, 1\leq i \leq k} f_i(x,y))$$
$$\wedge \bigwedge_{1\leq i<j\leq k} (\forall x \forall y\ f_i(x,y) \implies \neg f_j(x,y))$$
$$\wedge \bigwedge_{1\leq i\leq k} (\forall x \exists y\ f_i(x,y))$$

where $f_i/2$ are fresh predicates not appearing anywhere else. Then, it holds that

$$WFOMC(\Gamma \wedge \Psi, n, w, \overline{w}) = \frac{1}{k!^n} WFOMC(\Gamma \wedge \Upsilon, n, w, \overline{w})$$

Next is the last lemma used to obtain a $\mathbf{UFO^2}$ sentence from a $\mathbf{C}^2$ sentence. It was originally presented in Kuželka [2021].

Lemma 3.14. *Let Γ be a first-order logic sentence. Let Ψ be a $\mathbf{C}^2$ sentence such that $\Psi = \forall x\ A(x) \vee (\exists^{=k} R(x,y))$. Define $\Upsilon = \Upsilon_1 \wedge \Upsilon_2 \wedge \Upsilon_3 \wedge \Upsilon_4$ such that*

$$\Upsilon_1 = \forall x \forall y\ \neg A(x) \implies (R(x,y) \iff B_R(x,y))$$
$$\Upsilon_2 = \forall x \forall y\ ((A(x) \wedge B_R(x,y)) \implies U_R(y))$$
$$\Upsilon_3 = (|U_R| = k)$$
$$\Upsilon_4 = \forall x \exists^{=k}\ y B_R(x,y)$$

where U, $R/1$, and $B_R/2$ are fresh predicates not appearing anywhere else. Then, it holds that

$$WFOMC(\Gamma \wedge \Psi, n, w, \overline{w}) = \frac{1}{\binom{n}{k}} WFOMC(\Gamma \wedge \Upsilon, n, w, \overline{w})$$

Notice that in all these cases, only counting quantifiers in the form of $\exists^{=k}$ are used. That is because the mentioned lifted methods only work with counting quantifiers in the form of $\exists^{=k}$. That is because both the counting quantifier in the form of $\exists^{\leq k}$ and $\exists \geq k$ can be transformed to the form of $\exists^{=k}$. A sketch of how that is done is that counting quantifiers in the form of $\exists^{\leq k} x P(x)$ can be transformed to

$$\bigvee_{1\leq i\leq k} (\exists^{=k} P(x)).$$

The counting quantifiers in the form of $\exists^{\geq k} x P(x)$ can be transformed into

$$\neg \bigvee_{1\leq i\leq k-1} (\exists^{=k} P(x)).$$

Tóth and Kuželka [2024] also provide an algorithm describing how to apply all the lemmas above to obtain a $\mathbf{UFO^2} + \mathbf{CC}$ sentence from a $\mathbf{C}^2$ sentence.

This concludes the description of how lifted methods deal with counting quantifiers. We had the option to implement this method as well, however, we opted not to do so. That is because it would put a huge strain on the solvers by introducing many new solutions. We can see that lemma 3.14 introduces

Algorithm 4 Convert $\mathbf{C}^2$ to $\mathbf{UFO}^2$

```
Algorithm Convert $\mathbf{C}^2$ to $\mathbf{UFO}^2$
Input: Sentence $\Gamma \in \mathbf{C}^2$
Output: Sentence $\Gamma' \in \mathbf{UFO}^2 + \mathbf{CC}$
for all sentence $\exists^{=k}x\ \Psi(x)$ in $\Gamma$ do
    Apply Lemma 3.12
end for
for all sentence $\forall x \exists^{=k}y\ \Psi(x,y)$ in $\Gamma$ do
    Apply Lemma 3.13
end for
for all subformula $\Phi(x) = \exists^{=k}y\ \Psi(x,y)$ in $\Gamma$ do
    Create new predicates R/2 and A/1
    Let $\mu \leftarrow \forall x \forall y\ R(x,y) \iff \Psi(x,y)$
    Let $\nu \leftarrow \forall x\ A(x) \iff (\exists^{=k}y\ R(x,y))$
    Apply Lemmas 3.11, 3.14 and 3.13 to $\nu$
    Replace $\Phi$ by $A(x)$
    Append $\mu \wedge \nu$ to $\Gamma$
end for
for all sentence with an existential quantifier in $\Gamma$ do
    Apply Lemma 3.10
end for
return $\Gamma$
```

$\binom{n}{k}$-times new models, where $n = |\Delta|$, where Δ is the Herbrand Universe, and k is the number in the counting quantifier. Also, lemma 3.13 introduces $k!^n$-times new models, where $n = |\Delta|$, where Δ is the Herbrand Universe, and k is the number in the counting quantifier. As we show later, the number of models plays a key role in the runtime of our implementation, so using this method would only lower the size of universes we would be able to count the models of a sentence in. Also, implementing this method would be much harder than the first method we describe in this section, requiring a lot of work for very little results. However, we need to mention the importance of this method for counting the models of a sentence in a lifted way.

3.6 Linear Order Axiom

The last fragment of first-order logic we implment the grounding of is the $\mathbf{FO^2}$ with the **Linear Order Axiom**. Linear order axiom enforces a a total order, which means a reflexive, anti-symetric, transitive and strongly connected relation. Aside from transitivity, all these properties do belong to the $\mathbf{FO}^2$ fragment. Transitivity, however, requires 3 variables, so it does not belong to the $\mathbf{FO}^2$ fragment. However, Tóth and Kuželka [2023] showed that WFOMC with linear order axiom is domain-liftable.

We will be using the predicate $\leq$ to denote a linear order. When using the predicate $\leq$ to represent a linear order, we will be also using the infix notation. We will be denoting $\Phi = \Psi \wedge \mathit{Linear}(\leq)$, where Ψ is a sentence in $\mathbf{C}^2$ and $\leq$ is a predicate of arity two. We also provide a definition.

Definition 3.15. Let Ψ be a logical sentence containing a predicate $\leq$ of arity 2. A possible model ω is a model of $\Phi = \Psi \wedge \mathit{Linear}(\leq)$ if and only if ω is a model of Ψ and $\leq$ satisfies the linear order axioms.

Now, we present the 2 ways we ground the linear order. The first way we ground the linear order is by adding clauses to the formula in conjunctive normal form that we obtained such that a predicate follows the linear order axiom. That means we need to add all the qualities of the linear order to a predicate. Since we do not require a limited number of predicates that follow the linear order, we consider a predicate P/2 that linear order is being applied to, and a Herbrand Universe Δ. This is done by adding the following clauses to the formula in conjunctive normal form:

- $$\bigwedge_{x \in \Delta} P(x, x)$$
  This is simple reflexivity, there is no problem with it.

- $$\bigwedge_{x \in \Delta} \bigwedge_{y \in \Delta, y \neq x} (P(x, y) \vee P(y, x))$$
  We require that for any combination of two variables, at least one relation is true, either P(x, y) or P(y, x).

- $$\bigwedge_{x \in \Delta} \bigwedge_{y \in \Delta, y \neq x} (\neg P(x, y) \vee \neg P(y, x))$$
  While in the previous addition of clauses, we could have omitted the requirement that $x \neq y$, this time it is necessary to only append this clause when $x \neq y$. That is because if we added the clause where $x = y$, we would get a contradiction with reflexivity immediately.

- $$\bigwedge_{x \in \Delta} \bigwedge_{y \in \Delta, y \neq x} \bigwedge_{z \in \Delta, z \neq x, z \neq y} (\neg P(x, y) \vee \neg P(y, z) \vee P(x, z))$$
  These are clauses that encode transitivity. We use three variables, which would not be possible in the two-variable fragment of first-order logic, but since our application allows it, we use this approach. Importantly, this allows for the input sentences to contain evidence about the linear order. It also still allows us to claim that the resulting formula in conjunctive normal form is polynomial in the size of the domain, as the transitivity is encoded in $O(|\Delta|^3)$ size.

Another way we can ground the linear order axiom is not to fully ground it but to *semi-ground* it. This is done by fixing a single possible linear order, solving the WFOMC problem with a fixed linear order, and then multiplying the result by $|\Delta|!$, where Δ is a Herbrand Universe. This was presented as Corollary 1 in Tóth and Kuželka [2023]. This only allows one linear order axiom in a formula, compared to the full grounding above. Also, we only allow this way of grounding for sentences without evidence, meaning with no constants. It is possible to have a sentence with constants and linear order that can be solved using the *semi-grounded* approach, however, it would be hard to detect which sentences are possible to count this way and which cannot. We provide some examples below.

Example 3.16. Consider a sentence $\Phi = (2 \leq 1) \wedge \mathit{Linear}(\leq)$ and a *semi-grounded* linear order $\{(1 \leq 1), (1 \leq 2), (2 \leq 2)\}$. Then we easily obtain a contradiction, which means there are no models of Φ, while there exists at least one model: $\{(2 \leq 2), (2 \leq 1), (1 \leq 1)\}$.

Example 3.17. Consider a sentence $\Phi = P(1) \wedge \mathit{Linear}(\leq)$ and a *semi-grounded* linear order $\{(1 \leq 1), (1 \leq 2), (2 \leq 2)\}$. This sentence contains evidence and its models can be counted using the *semi-grounded* linear order.

Example 3.18. Consider a sentence

$$\Phi = \Psi \wedge \mathit{Linear}(\leq),$$

$$\Psi = (P(1) \vee (Q(2,1) \implies (2 \leq 1))$$

and a semi-grounded linear order $\{(1 \leq 1), (1 \leq 2), (2 \leq 2)\}$. We can see that the ground atoms change what possible worlds are models drastically. Even more indirect evidence for the linear order may be introduced in a formula. Knowing this, it would require a high amount of preprocessing to distinguish formulas with evidence of which we can count the models with semi-grounded linear order and which models we cannot. Therefore, we do not allow any constants in sentences with linear order.

Chapter 4

Implementation

In this chapter, we introduce the reader to the implementation of the application we developed to solve the WFOMC problem by grounding. The application we provide is written in C, which we chose for its high performance. While C does not support object-oriented programming, we can leverage structures to obtain a pseudo polymorphism. This is useful when storing arbitrary first-order logic formulas.

4.1 Data Structures

To store arbitrary formulas, we use the following data structures

- **Object** is a pseudo superclass to represent a formula. It stores either a Formula or Atom with the information which of the former it is.
- **Formula** is a structure that consists of a list of Prepositions that affect the formula, the logical connective between its subformulas, and a list of Objects that represent its subformulas.
- **Atom** is an atomic formula, it consists of a list of Prepositions being applied to the Predicate inside, a Predicate, and a (possibly empty) list of Terms this predicate is being applied to.
- **Preposition** is a negation symbol, a universal, existential, or counting quantifier along with the variable it is being applied to, in the case of counting quantifier also with a number k of the counting quantifier.
- **Term** is a pseudo superclass for Functions, Variables, and Constants, containing any of the former and the type it stores.
- **Function** consists of a function symbol and Terms it is being applied to.
- **Variable, Constant** represent variables and constants, containing their respective names.

These data structures allow us to store any arbitrary formula ϕ. During experiments, we do not allow the use of functions, as any function of arity greater or equal to 1 makes the Herbrand base infinite, making model counting impossible.

In our implementation, cardinality constraints are not represented as a part of a formula, but rather a list of cardinality constraints is passed to the counter if any are present. Similarly, a linear order is not represented as a part of a formula, but rather a name of a predicate that needs to satisfy the linear order is passed to the counter. The reason for this is that both cardinality constraints and linear order are enforced by adding clauses after the formula has been grounded. This makes grounding the formula easier, as when grounding a formula, we need not worry about any cardinality constraints or linear order. Also, when grounding the cardinality constraints or linear order, we can create clauses in DIMACS format, the output format of the grounder, rather than creating a construct of structures that would be transformed to the DIMACS format immediately.

The DIMACS format [1] is a widely used format for storing propositional formulas in CNF. We ground formulas into the DIMACS format both in the case of solving iteratively with a SAT solver or when using dedicated WMC solver as all these solvers expect their input to be a DIMACS file.

A file in the DIMACS format is an ASCII file that stores propositional formulas in CNF. The file consists of a problem line in the format of `p cnf n m`, where `n` is the number of clauses in the formula and `m` is the number of propositional variables, and usually `n` clause lines, which are lines of positive or negative whole numbers ending with a 0. There can also be optional comment lines starting with `c`. It is not necessary to write each clause on a separate line, a clause can take several lines or several clauses divided by 0s can be on a single line, however, it is customary that each clause be one line long. A short example of a DIMACS file is a file:

```
c first comment line
p cnf 2 3
c second comment line
1 -2 0
-1 3 0
```

which encodes the formula $(x \vee \neg y) \wedge (\neg x \vee z)$. The application we implemented either solves iteratively using a SAT solver or produces a DIMACS file to be used with a WMC solver.

4.2 Solvers

In this work, we work with 3 solvers. The first is a SAT solver CryptoMiniSat [Soos et al., 2009]. This solver is a SAT solver, that we use when finding solutions by iterative calls to a SAT solver. We chose CryptoMiniSat because it offers an interface in C, which made it easy to implement in our application. Also, CrpytoMiniSat is a contender for the International SAT Competition [2]. It earned three bronze awards in the year 2018 and a golden and bronze award

[1]/https://www.satcompetition.org/2009/format-benchmarks2009.html
[2]http://www.satcompetition.org/

in the year 2016. It offers a fair bit of documentation, making its use easy while being sufficiently fast, which is shown in the experiments. Other solvers exist, for example, CaDiCaL [Biere et al., 2020], or Glucose and many others that can be found in Heule et al. [2018]. However, not many of them offer an interface in C or sufficient documentation to use them in our implementation.

Next, we use the solver DSHARP [Muise et al., 2012], which is a WMC solver. The last solver we use is the ADDMC solver [Dudek et al., 2020], which is a WMC solver that tied for first place at the Model Counting Competition of 2020 [Fichte et al., 2021]. Again, many other solvers exist and some of them can be found in Fichte et al. [2021].

We compare the performance of these model counters against the state-of-the-art model counters using the lifted approach. These model counters utilize the FastWFOMC algorithm [van Bremen and Kuželka, 2021] and the IncrementalWFOMC algorithm [Tóth and Kuželka, 2023]. FastWFOMC is an algorithm that computes WFOMC in the $\mathbf{C^2}$ fragment, while IncrementalWFOMC computes WFOMC in the $\mathbf{C^2} + \mathbf{Linear}$ fragment.

4.3 File Structure

We keep the usual file structure of C programs having an implementation file `file.c` and a header file `file.h` for each structure and each major functionality. An exemption from this is the definition of **object**, **formula**, and **atom** structures in the files `formula.c` and `formula.h`. This is to prevent circular dependency.

All functions the user should need are declared in the files `fluffbro.h`, in which they are also documented. How to use these functions can be seen in the file `main.c`, in which all examples from Chapter 5 are also implemented. Alongside the source code, we provide a `Makefile`, which can be used to compile the `main.c` into an executable binary `main`. It also has the typical options of `run`, which executes the binary, and `clean`, which removes the `.o` files. Another option of the `Makefile` is `valrun`, which compiles the code and runs Valgrind with the file. This is because we developed the code to be Valgrind safe [3].

[3]https://valgrind.org/

Chapter 5

Experiments

In this chapter, we present the results we obtained from testing different approaches to WFOMC on several examples. When comparing the different approaches, we measure the maximum size of the Herbrand universe the approach can manage and the time it takes. The lifted approach is implemented in the Julia language [Bezanson et al., 2017], so we used the package BenchmarkTools [1] to measure the execution time, which gives us the average time of many runs. We only measure runtimes on universes $|\Delta| \leq 20$ using the lifted approach, even though it is capable of going far beyond $|\Delta| = 20$. However, no other approach can go beyond $|\Delta| = 20$, so we stop there. Otherwise, we measured the time of 100 runs and chose the median, if possible, and performed fewer measurements when runtime did not allow that many runs. We counted unweighted formulas, as the only difference to weighted formulas is multiplying the weights of predicates, which would only add constant time to what we measure this way. All measurements were taken on a laptop with the Intel Core i3-8145U CPU.

5.1 Two-Variable Fragment

We first measure the performance of different solvers within the $\mathbf{FO^2}$ fragment. We chose the following problems to measure the performance of the solvers.

Graphs without isolated nodes, which are represented by the formula

$$\begin{aligned}\Phi =& \forall x\, \neg Edge(x,x) \wedge \\ & \forall x \forall y\, Edge(x,y) \implies Edge(y,x) \wedge \\ & \forall x \exists y\, Edge(x,y).\end{aligned}$$

Runtimes can be seen in Figure 5.1. The sequence generated by counting the models of this sentence can be found in OEIS Foundation Inc. [2024] as sequence `A006129`.

[1]juliaci.github.io/BenchmarkTools.jl/

n-colored graphs without isolated nodes, with the general formula

$$\begin{aligned}\Phi =&\forall x\ \neg Edge(x,x)\wedge\\ &\forall x\forall y\ Edge(x,y) \implies Edge(y,x)\wedge\\ &\forall x\exists y\ Edge(x,y)\wedge\\ &\forall x \bigvee_{1\leq i\leq n} C_i(x)\\ &\forall x\ \neg C_i \vee \neg C_j,\ 1\leq i<j\leq n\ \wedge\\ &\forall x\forall y\ Edge(x,y) \implies \neg C_i(x)\vee\neg C_i(y)\ 1\leq i\leq n.\end{aligned}$$

We chose $n \in \{2, 3, 4, 5\}$. Runtimes can be seen in Figure 5.2. The only sequence of those generated by model counting these sentences that can be found in OEIS Foundation Inc. [2024] is A052332 for $n = 2$.

Edge covers, which are represented by the formula:

$$\begin{aligned}\Phi =&\forall x\ \neg Edge(x,x)\wedge\\ &\forall x\forall y\ Edge(x,y) \implies Edge(y,x)\wedge\\ &\forall x\exists y\ CoverEdge(x,y)\\ &\forall x\forall y\ CoverEdge(x,y) \implies CoverEdge(y,x)\\ &\forall x\forall y\ CoverEdge(x,y) \implies Edge(x,y)\end{aligned}$$

Runtimes can be seen in Figure 5.3.

And vertex covers, which are represented by the formula:

$$\begin{aligned}\Phi =&\forall x\ \neg Edge(x,x)\wedge\\ &\forall x\forall\ yEdge(x,y) \implies Edge(y,x)\wedge\\ &\forall x\forall y\ (Edge(x,y) \implies CoverVertex(x)\vee CoverVertex(y))\end{aligned}$$

Runtimes can be seen in Figure 5.4. The sequence generated by counting the models of this formula can be found in OEIS Foundation Inc. [2024] as seqeunce A079491.

We can state several observations from the data in Figures 5.1 through 5.4.

First, we can see that the iterative SAT solver is worse than exponential when counting models. This is because the more solutions we find, the longer time it takes to generate new solutions. This can be seen in Figure 5.5, which shows the cumulative time it takes the iterative SAT solver to find 200,000 solutions of a graph with unisolated nodes formula with $|\Delta| = 7$.

The DSHARP model counter works very well for small domain sizes. Then, we can see that its complexity is exponential in the size of the universe. From the non-lifted model counters, DSHARP performs the best. From the runtime of n-colorable graphs, we can see that it struggles a bit with a larger equation of 5-colorable graphs.

The ADDMC model counter did not perform well. It features weird but reproducible spikes in some sizes of the universe Δ. This behavior can be

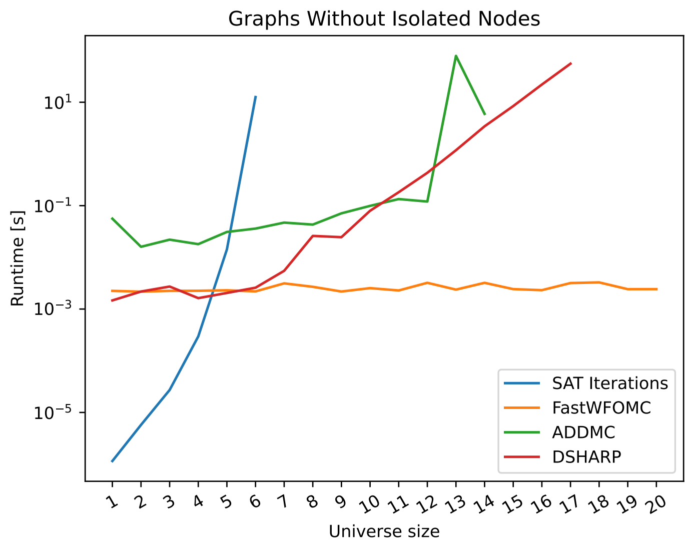

Figure 5.1: Runtime of Graphs Without Isolated Nodes

seen in Figures 5.1 and 5.4. It does not outperform the DSHARP model counter and sometimes not even the iterative SAT solver. It comes out as worst but will show more promise in other fragments.

The FastWFOMC lifted method shows that its runtime is almost constant in the size of the universe Δ. We could go far beyond $|\Delta| = 20$ and it still counts models of the formula in nearly the same time. There is a noticeable jump in runtime between the runtime of counting models of 4-colorable graphs and 5-colorable graphs in Figure 5.2. It is because the FastWFOMC complexity is polynomial in the size of the domain. Since the 5-colorable graphs have the largest formula, the time to count the models of this formula is the longest.

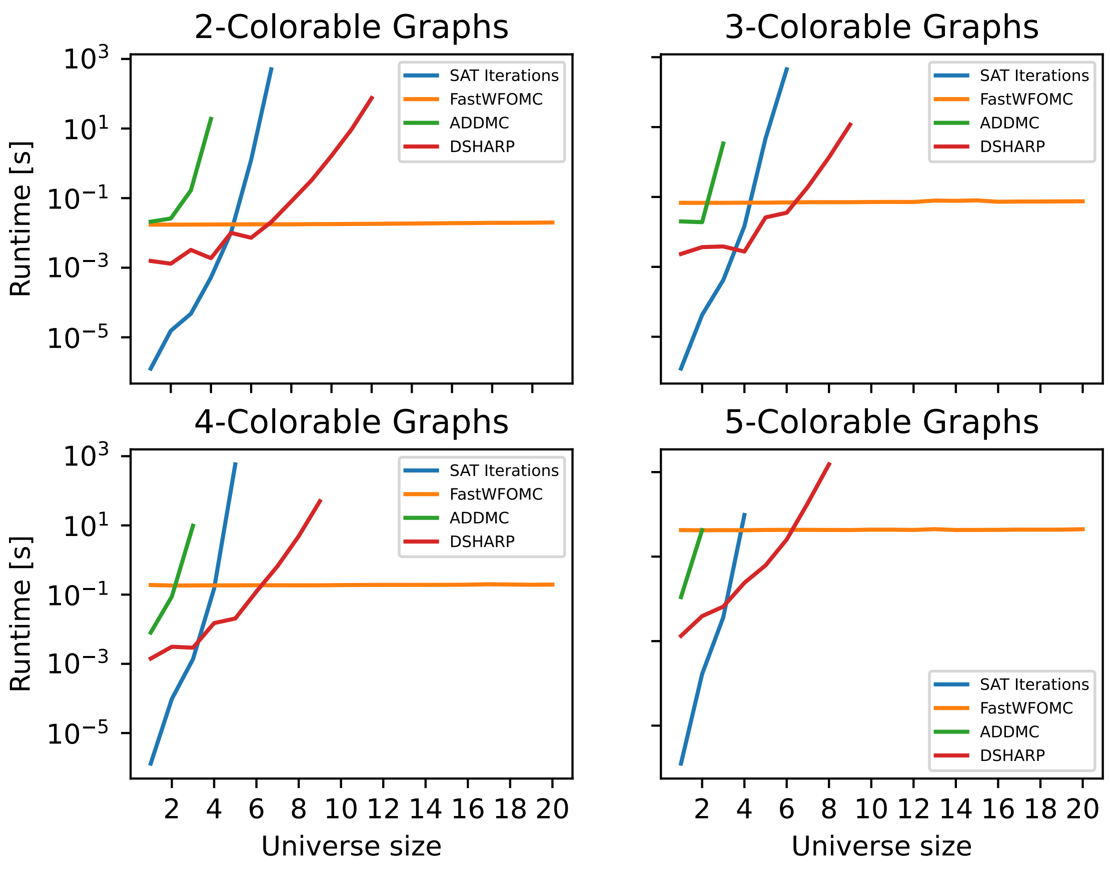

Figure 5.2: Runtime of n-Colorable Graphs

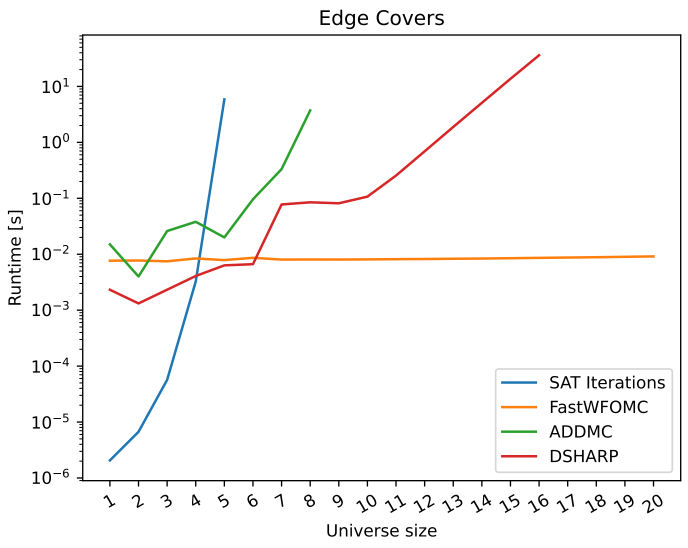

Figure 5.3: Runtime of Edge Covers

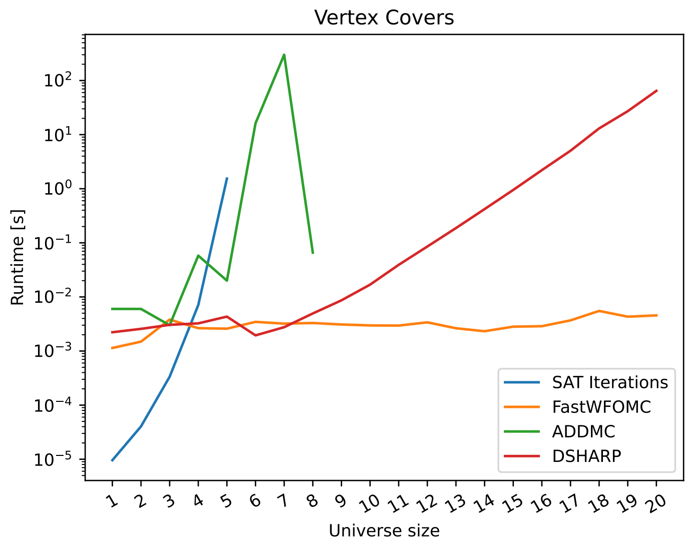

Figure 5.4: Runtime of Vertex Covers

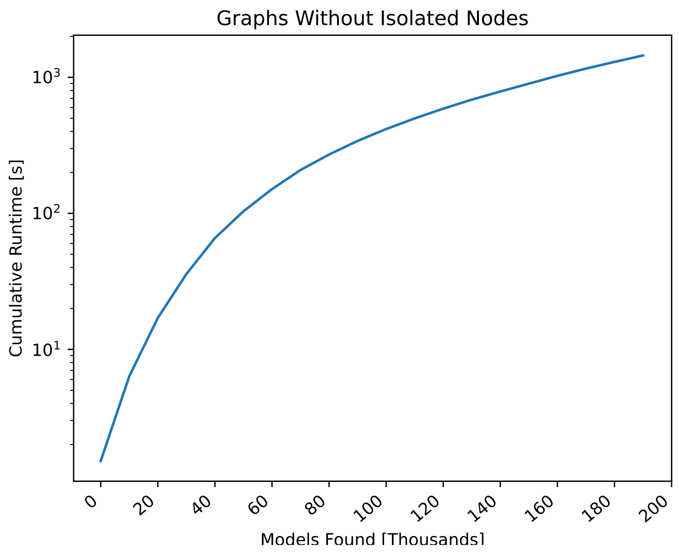

Figure 5.5: Cumulative Time of Finding Models of Graphs Without Isolated Nodes, $|\Delta| = 7$

5.2 Two-Variable Fragment With Cardinality Constraints

Next, we measured the runtime of solving problems from the $\mathbf{FO^2} + \mathbf{CC}$ fragment. We chose the following problems to measure the performance of the solvers. Since the FastWFOMC package only allows for cardinality constraints in the form of $|P| = k$, we only use this form of cardinality constraints so that we can compare the results to FastWFOMC.

Graphs with 3-edge cover and 4-edge cover, which are represented by the formula:

$$\begin{aligned}\Phi =& \forall x\ \neg Edge(x,x) \wedge \\ & \forall x \forall y\ Edge(x,y) \implies Edge(y,x) \wedge \\ & \forall x \exists y\ CoverEdge(x,y) \wedge \\ & \forall x \forall y\ CoverEdge(x,y) \implies CoverEdge(y,x) \wedge \\ & \forall x \forall y\ CoverEdge(x,y) \implies Edge(x,y) \wedge \\ & |CoverEdge| = k,\end{aligned}$$

where $k \in \{6, 8\}$. The runtimes can be seen in Figures 5.6 and 5.7. It should be noted, that for $k = 3$, model count of any universe $|\Delta| > 6$ is 0, so we only go up to $|\Delta| = 7$ and stop there. Similarly, we only go up to $|\Delta| = 9$ for $k = 4$.

Graphs with 1-vertex cover, 2-vertex cover, or 3-vertex cover. The formula representing this problem is:

$$\begin{aligned}\Phi =& \forall x\ \neg Edge(x,x) \wedge \\ & \forall x \forall y\ Edge(x,y) \implies Edge(y,x) \wedge \\ & \forall x \forall y\ Edge(x,y) \implies (CoverVertex(x) \vee CoverVertex(y)) \wedge \\ & |CoverVertex| = k,\end{aligned}$$

where $k \in \{1, 2, 3\}$. The runtimes can be seen in Figures 5.8 through 5.10. The sequences generated by this sentence can be found in OEIS Foundation Inc. [2024] as:

- `A001787` for $n = 1$.
- `A052780` for $n = 2$.
- There is no entry for $n = 3$.

In these examples, we can show several things. We can see that the iterative SAT solver is extremely responsive to the number of solutions it has to find. In Figure 5.6, it can to count the models of universe $|\Delta| = 6$, and the universe $|\Delta| = 7$ is much faster as there are no models from this universe onwards. In Figure 5.13, we see that for a very restrictive formula that has very few models, it can count the models of a universe $|\Delta| = 15$. However, in Figure

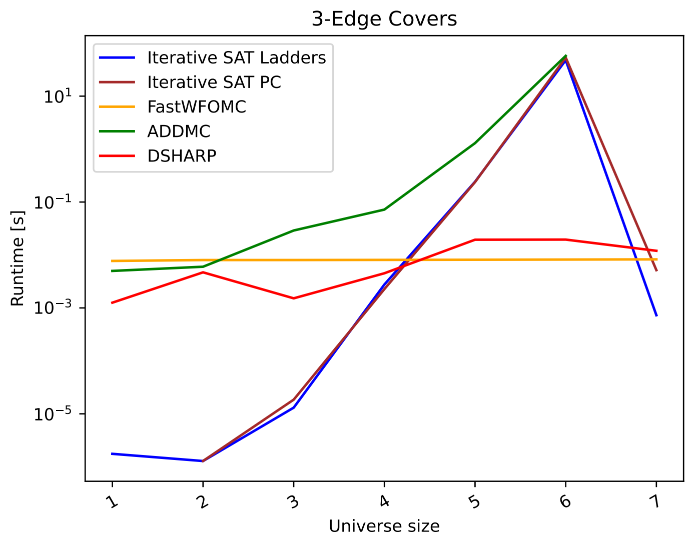

Figure 5.6: Runtime of 3-Edge Covers

5.15, we can see that for a formula that has many solutions, it can only count up to $|\Delta| = 5$. We can also compare the ladder encoding and parallel counter encoding. The comparison can be seen in Figure 5.11. The encoding that seems to be slightly better is the ladder encoding. This makes it seem that the efficiency is more important for runtime than the number of clauses or atoms.

We can see that DSHARP performs well, being the best of the non-lifted methods. We can see jumps in runtimes for 1-vertex covers and 3-vertex covers. This is because these problems have 0 models for $|\Delta| = 2 \cdot k + 1, k \in \mathbb{N}$ and DSHARP decides satisfiablity during preprocessing, skipping counting the models if the formula is unsatisfiable.

ADDMC again performs the worst of the solvers. It does not decide satisfiablity of the formula and only counts its models, which is important for the problems we chose for this fragment.

FastWFOMC still takes constant time to count models of universes of all sizes.

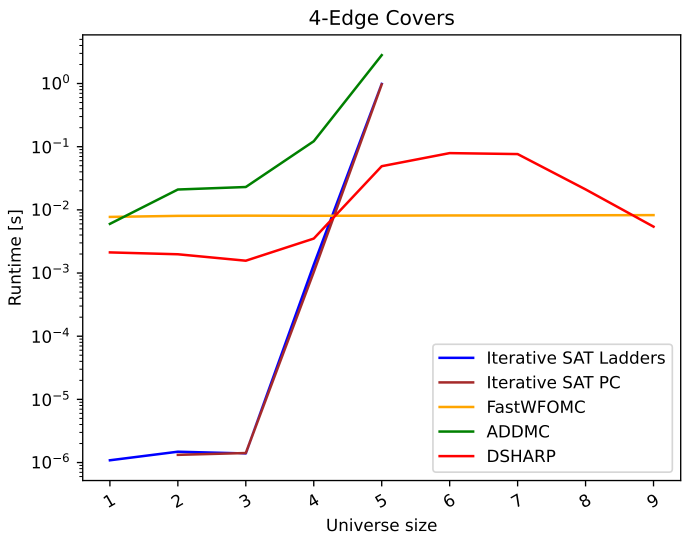

Figure 5.7: Runtime of 4-Edge Covers

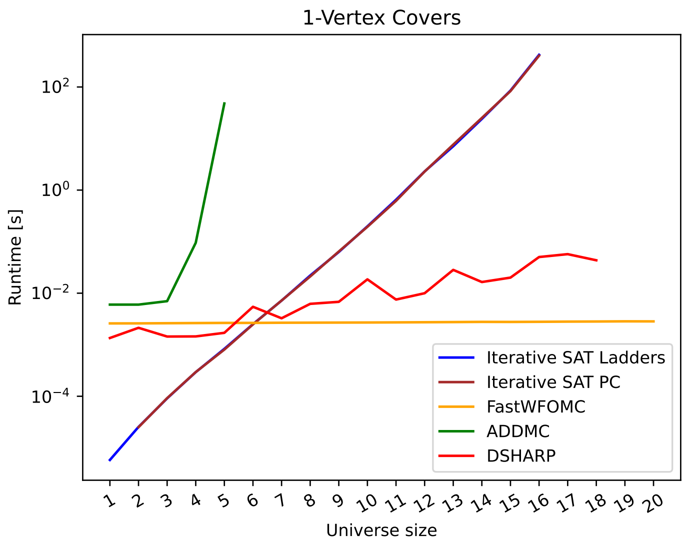

Figure 5.8: Runtime of 1-Vertex Covers

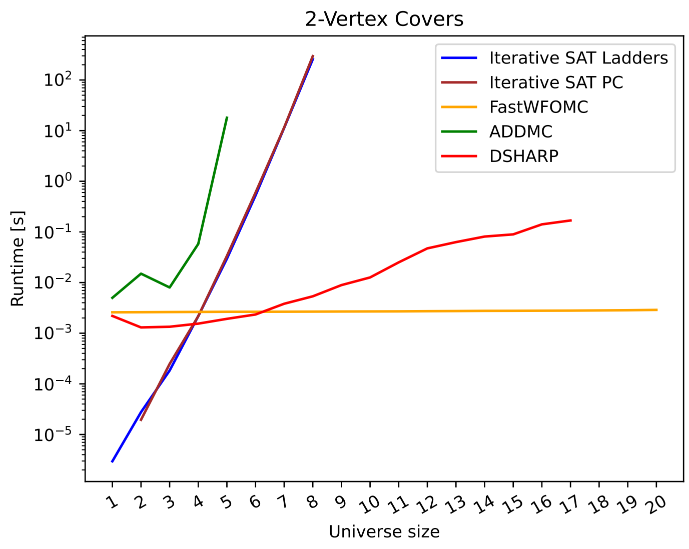

Figure 5.9: Runtime of 2-Vertex Covers

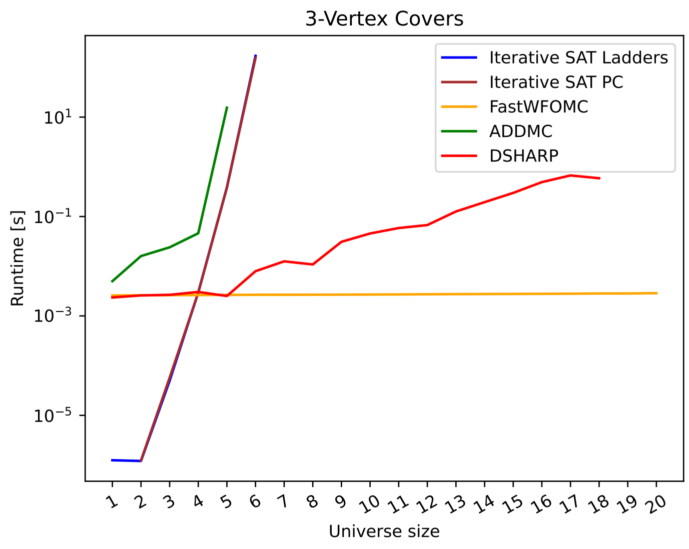

Figure 5.10: Runtime of 3-Vertex Covers

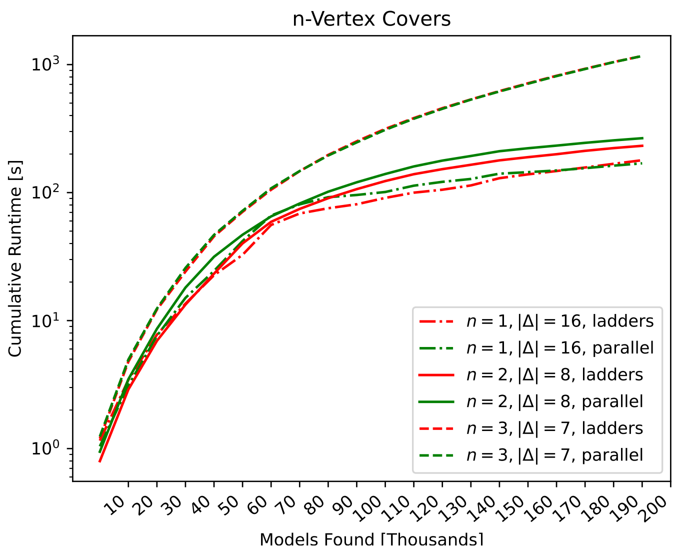

Figure 5.11: Cumulative Time to Find 200,000 Solutions of n-Vertex Covers

5.3 Two-Variable Fragment with Counting Quantifiers

The next fragment we measured the runtime of solvers is $\mathbf{C^2}$. We chose the following problems to measure the performance of the solvers.

Perfect Matching, which is represented by the formula:

$$\begin{aligned}\Phi =&\forall x\ \neg Edge(x,x)\wedge\\ &\forall x\forall y\ Edge(x,y) \implies Edge(y,x)\wedge\\ &\forall x\exists^{=1}y\ Match(x,y)\wedge\\ &\forall x\forall y\ Match(x,y) \implies Match(y,x)\wedge\\ &\forall x\forall y\ Match(x,y) \implies Edge(x,y)\end{aligned}$$

The runtimes of this formula can be seen in Figure 5.12

n-regular graphs, which are represented by the formula:

$$\begin{aligned}\Phi =&\forall x\ \neg Edge(x,x)\wedge\\ &\forall x\forall y\ Edge(x,y) \implies Edge(y,x)\wedge\\ &\forall x\exists^{=n}y\ Edge(x,y),\end{aligned}$$

where $n \in \{1,2,3\}$. The runtimes of this formula can be seen in Figures 5.13 through 5.15. The sequences generated by this sentence can be found in OEIS Foundation Inc. [2024] as:

- `A123023` for $n = 1$.
- `A001205` for $n = 2$.
- There is no entry for $n = 3$.

From the runtimes, we can again see that the unsatisfiable problems are much faster to solve for the iterative SAT solver. We also see that the ladder encoding seems to perform better. Further proof of that can be seen in Figure 5.16. We have more empyrical evidence that efficiency is the more important factor for cardinality constraint encodings.

DSHARP is still able to count the models of the formulas up to the highest $|\Delta|$ from the non-lifted methods. However, it starts to run into problems, as it cannot count the models of a universe with more than 2^{1024} possible worlds, which is equivalent to a grounded formula with 1024 ground atoms. Even if we use the parallel counter encoding to reduce the number of ground atoms in the formula, DSHARP still is not able to count past some universe sizes.

ADDMC is able to count to higher sizes of Δ, but still performs the worst of the presented solvers.

FastWFOMC still runs in nearly constant time, with the exceptions that can be seen in Figures 5.14, and 5.15. In these cases, the formula is unsolvable for $|\Delta| \leq 2$ or $|\Delta| \leq 3$ respectively and some preprocessing is likely responsible for faster detection of this fact.

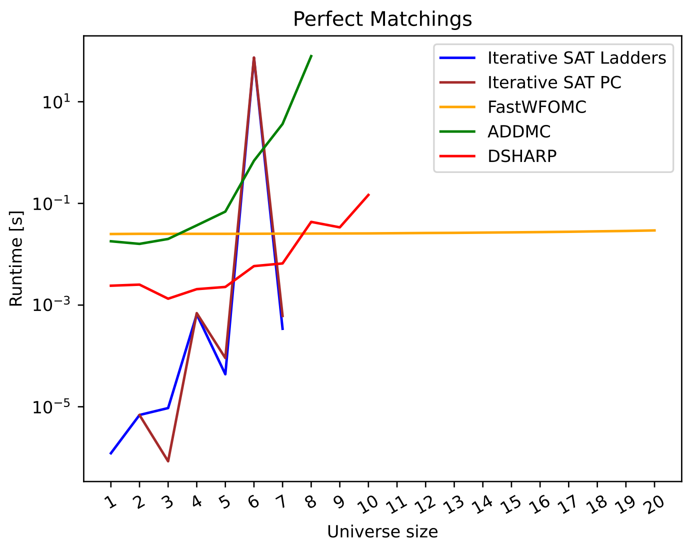

Figure 5.12: Runtime of Prefect Matching

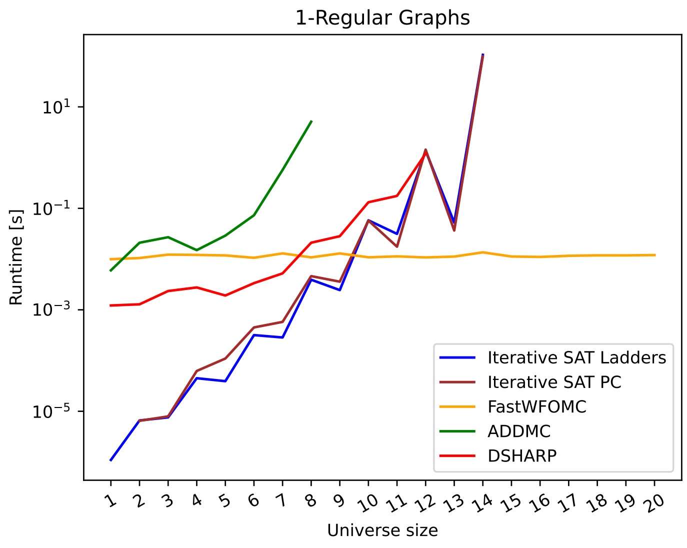

Figure 5.13: Runtime of 1-Regular Graphs

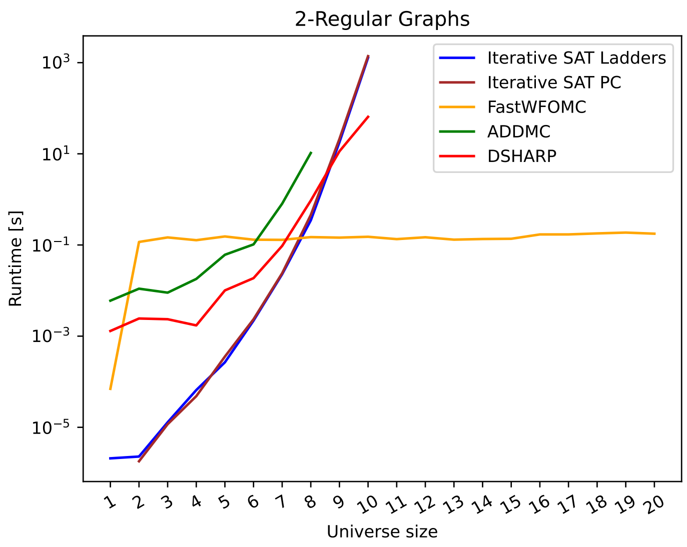

Figure 5.14: Runtime of 2-Regular Graphs

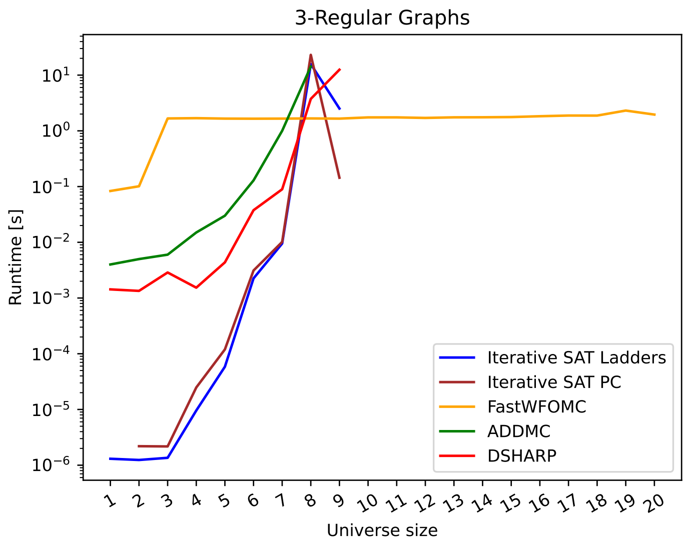

Figure 5.15: Runtime of 3-Regular Graphs

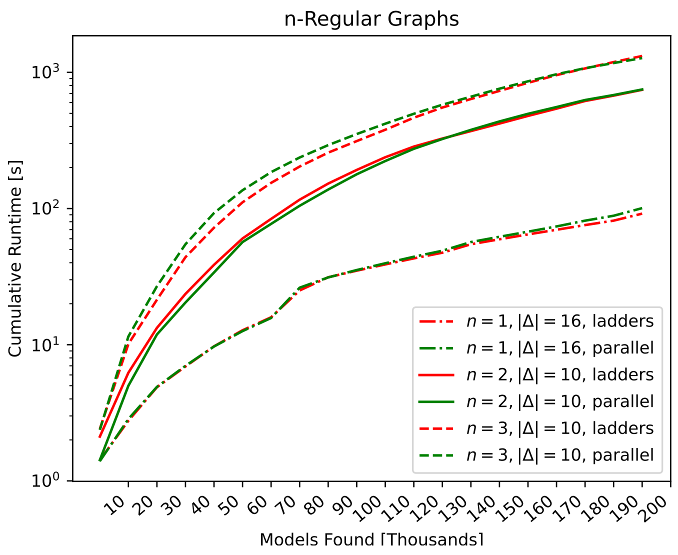

Figure 5.16: Cumulative Time to Find 200,000 Models of k-Regular Graphs

5.4 Two-Variable Fragemnt with Linear Order Axiom

The last fragment we measured the performance of the solvers on is the $\mathbf{FO^2} + \mathbf{Linear}$. In this fragment, we chose the sequence splitting problems *Head & Tail* and *Head, Tail, & Middle*. These problems split a sequence to a head and tail, or head, tail, and middle respectively.

Head & Tail

$$\begin{aligned}\Phi =& \forall x\ Head(x) \lor Tail(x) \land \\ & \forall x\ \neg Head(x) \lor \neg Tail(X) \land \\ & \forall x \forall y\ (Head(x) \land (y \leq x)) \implies Head(y) \land \\ & \forall x \forall y\ (Tail(x) \land (x \leq y)) \implies Tail(y) \land \\ & Linear(\leq)\end{aligned}$$

The runtime of this formula can be seen in Figure 5.17. The sequence generated by counting the models of this sentence can be found in OEIS Foundation Inc. [2024] as sequence `A000142`.

Head, Tail & Middle

$$\begin{aligned}\Phi =& \forall x\ Head(x) \lor Tail(x) \lor Middle(x) \land \\ & \forall x\ \neg Head(x) \lor \neg Tail(X) \land \\ & \forall x\ \neg Middle(x) \lor \neg Tail(X) \land \\ & \forall x\ \neg Head(x) \lor \neg Middle(X) \land \\ & \forall x \forall y\ (Head(x) \land (y \leq x)) \implies Head(y) \land \\ & \forall x \forall y\ (Tail(x) \land (x \leq y)) \implies Tail(y) \land \\ & Linear(\leq)\end{aligned}$$

The runtime of this formula can be seen in Figure 5.18. The sequence generated by counting the models of this sentence can be found in OEIS Foundation Inc. [2024] as sequence `A001710`.

In this fragment, we can see that all the solvers are overwhelmed by the extremely high number of models these formulas have. ADDMC again performs poorly.

The iterative SAT solver utilizing a full grounding of the linear order is again worse than exponential and times out on quite a small universe Δ. On the other hand, the iterative SAT solver utilizing a *semi-grounding* of the linear order works extremely well. It is very fast, but at $|\Delta| = 12$, we encounter an integer overflow and the results are not correct from this point onward. This could be solved by improving the implementation to use arbitrary precision for the output. Even if we achieved arbitrary precision though, we can see from the Figures 5.17 and 5.18 that the trend is that this approach would be outperformed by IterativeWFOMC for larger domains.

The IncrementalWFOMC counts models in nearly constant times and again can count the models in much larger universes than we present.

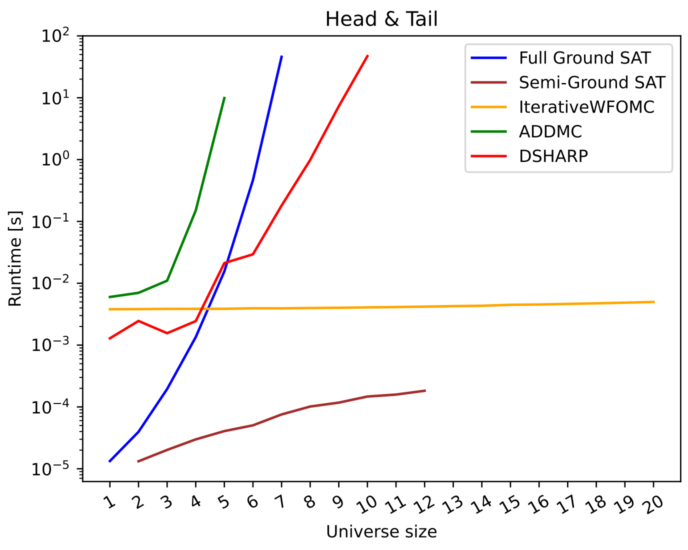

Figure 5.17: Runtime of Head Tail

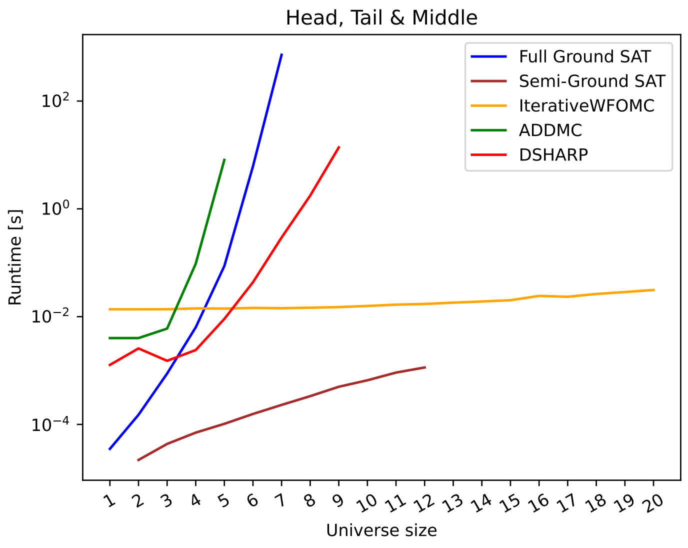

Figure 5.18: Runtime of Head Tail Middle

Chapter 6

Conclusion

Weighted first-order model counting is a powerful technique that can be used to solve several practical problems. With the progress made recently in lifted approach to WFOMC, this technique became even more powerful. However, lifted approach to WFOMC is only possible in certain first-order logical fragments. For some of these fragments, the non-lifted baseline we could compare the performance of the lifted approach to is missing. The goal of this work was to provide such a baseline and compare it to the lifted methods.

We first created an infrastructure, that allowed us to handle formulas from the required fragments and ground them. We then used several different solvers to count the models of the formula and compared their performance to the performance of the lifted approach. We also compared several approaches to grounding fragment-specific constructs.

In Chapter 5 we present the results of experimental performance testing. We can clearly see from the results that the lifted methods perform incomparably better to the non-lifted ones. This performance is what makes the lifted approach to WFOMC much more desirable and usable compared to the non-lifted approach. This proves that the effort put into developing lifted methods that solve the WFOMC problem yields great results.

The only advantage of non-lifted methods is their ability to count the models of any arbitrary formulas not being subject to any logical fragment. However, as we show in Chapter 5, the number of atoms in the grounded formula as well as the number of models makes counting the models of the formula take more time. That means that these methods would probably fail for arbitrary formulas that are more complicated than those we can solve using lifted methods. To conclude our work, we can safely say that the lifted methods vastly outperform the non-lifted baseline.

Bibliography

Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. Symmetric weighted first-order model counting. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '15, page 313–328, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450327572. doi: 10.1145/2745754.2745760. URL `https://doi.org/10.1145/2745754.2745760`.

Martin Svatoš, Peter Jung, Jan Tóth, Yuyi Wang, and Ondřej Kuželka. On discovering interesting combinatorial integer sequences. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23, 2023. ISBN 978-1-956792-03-4. doi: 10.24963/ijcai.2023/372. URL `https://doi.org/10.24963/ijcai.2023/372`.

Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. Skolemization for weighted first-order model counting. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'14, page 111–120. AAAI Press, 2014. ISBN 1577356578.

Ondřej Kuželka. Weighted first-order model counting in the two-variable fragment with counting quantifiers. *Journal of Artificial Intelligence Research*, 70:1281–1307, 03 2021. doi: 10.1613/jair.1.12320.

Jan Tóth and Ondřej Kuželka. Lifted inference with linear order axiom. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37:12295–12304, 06 2023. doi: 10.1609/aaai.v37i10.26449.

T. Hinrichs and M. Genesereth. Herbrand logic. technical report lg-2006-02, stanford university, stanford, ca. http://logic.stanford.edu/reports/LG-2006-02.pdf, 2006.

G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. ISBN 978-3-642-81955-1. doi: 10.1007/978-3-642-81955-1_28. URL `https://doi.org/10.1007/978-3-642-81955-1_28`.

Elias Kuiter, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. Tseitin or not tseitin? the impact of cnf transformations on feature-model analyses. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394758. doi: 10.1145/3551349.3556938. URL https://doi.org/10.1145/3551349.3556938.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. ISBN 9781450374644. doi: 10.1145/800157.805047. URL https://doi.org/10.1145/800157.805047.

L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(79)90044-6. URL https://www.sciencedirect.com/science/article/pii/0304397579900446.

Paulius Dilkas and Vaishak Belle. Weighted model counting with conditional weights for bayesian networks. In Cassio de Campos and Marloes H. Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 386–396. PMLR, 27–30 Jul 2021. URL https://proceedings.mlr.press/v161/dilkas21a.html.

Guy Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. *IJCAI International Joint Conference on Artificial Intelligence*, pages 2178–2185, 01 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-363.

Aaron R. Bradley and Zohar Manna. *The calculus of Computation*. Springer-Verlag Berlin Heidelberg, 2007.

P. G.; Gradel, E.; Kolaitis and M. Y. Vardi. *On the decision problem for two-variable first-order logic. Bull. Symb. Log., 3(1): 53–69.* 1997.

Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. *International Conference on Principles and Practice of Constraint Programming*, 2833:108–122, 09 2003. doi: 10.1007/978-3-540-45193-8_8.

Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. doi: 10.1007/978-3-642-02777-2_24. URL https://doi.org/10.1007/978-3-642-02777-2_24.

Christian Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric Hsu. DSHARP: Fast d-DNNF Compilation with sharpSAT. In *Canadian Conference on Artificial Intelligence*, 2012.

Jeffrey Dudek, Nguyen Phan Vu, and Moshe Vardi. Addmc: Weighted model counting with algebraic decision diagrams. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:1468–1476, 04 2020. doi: 10.1609/aaai.v34i02.5505.

OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, 2024. Published electronically at `http://oeis.org`.

Patrick Hurley. *A Concise Introduction to Logic*. Wadsworth Publishing, Belmont, CA, 12 edition, October 2013.

Carlos Ansótegui and Felip Manyà. Mapping problems with finite-domain variables into problems with boolean variables. *Lecture Notes in Computer Science*, 3542, 05 2004. doi: 10.1007/11527695_1.

Jan Tóth and Ondřej Kuželka. Complexity of weighted first-order model counting in the two-variable fragment with counting quantifiers: A bound to beat. *arXiv preprint arXiv:2404.12905*, 2024.

Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

Marijn J. H. Heule, Matti Juhani Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*, volume B-2018-1 of *Department of Computer Science Series of Publications B*. Department of Computer Science, University of Helsinki, Finland, 2018.

Johannes K. Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. *ACM J. Exp. Algorithmics*, 26, oct 2021. ISSN 1084-6654. doi: 10.1145/3459080. URL `https://doi.org/10.1145/3459080`.

Timothy van Bremen and Ondřej Kuželka. Faster lifting for two-variable logic using cell graphs. In Cassio de Campos and Marloes H. Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1393–1402. PMLR, 27–30 Jul 2021. URL `https://proceedings.mlr.press/v161/bremen21a.html`.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. URL `https://doi.org/10.1137/141000671`.

Appendix A

Enclosed Source Codes

```
│ cardinality_constraint.c
├─ cardinality_constraint.h
├─ cns.c
├─ cns.h
├─ constant.c
├─ constant.h
├─ fluffbro.c
├─ fluffbro.h
├─ formula.c
├─ formula.h
├─ function_symbol.h
├─ function_symbol.c
├─ grounder.c
├─ grounder.h
├─ hash_simple.c
├─ hash_simple.h
├─ ladder_encoding.c
├─ ladder_encoding.h
├─ linear_order_axiom.c
├─ linear_order_axiom.h
├─ list.c
├─ list.h
├─ main.c
├─ Makefile
├─ parallel_counter_encoding.c
├─ parallel_counter_encoding.h
├─ predicate.c
├─ predicate.h
├─ prenex.c
├─ prenex.h
├─ prep.c
├─ prep.h
├─ skolemize.c
└─ skolemize.h
```

```
│
├── term.c
├── term.h
├── utils.c
├── utils.h
├── variable.c
└── variable.h
```