RELAXING DEDUCTIVE AND INDUCTIVE REASONING IN RELATIONAL LEARNING

MARTIN SVATOŠ

Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

A thesis submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy (Ph.D.)
Study Programme No. P2612 – Electrical Engineering and Information Technology
Branch No. 2612V025 – Information Science and Computer Engineering

SUPERVISOR:
Ing. Ondřej Kuželka, Ph. D.

February 2024

Martin Svatoš: *Relaxing Deductive and Inductive Reasoning in Relational Learning,* a thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Ph.D.),
Study Programme No. P2612 – Electrical Engineering and Information Technology
Branch No. 2612V025 – Information Science and Computer Engineering,

SUPERVISORS:
Ing. Ondřej Kuželka, Ph. D.

LOCATION:
Prague, Czech Republic

TIME FRAME:
2016 – February 2024

To my brother
Jiří
and my beloved
Ráď'a
for making me a better person.

ABSTRACT

Relational learning employs first-order logic for the representation of both data and models. The advantages of this representation arise from the transparent explainability of models and the ability to model a wide variety of tasks, including, for example, the well-known structural alert or knowledge base completion. However, it also poses some challenges when one wants to learn a model relying on an accurate hypothesis, which is quite a challenging problem since the space of hypotheses, i.e., logical formulae, is large and, in turn, makes the learning expensive. Utilizing learned hypotheses in a non-naive way is a challenging problem since learning on real-life data commonly leads to imperfect hypotheses.

This thesis addresses these two challenges. Namely, we focus on pruning the space of logical sentences in two separate scenarios. Firstly, we study the problem of generating integer sequences that can be described using first-order logic. We show that our novel approach, which resembles relational pattern mining, is fruitful and scalable by pruning the space of logical formulae. Besides producing a database as large as one-seventh of the well-known On-Line Encyclopedia of Integer Sequences (OEIS), our approach was also able to generate new descriptions of several entries in OEIS. Secondly, we present a novel general technique for pruning hypotheses space by utilizing domain knowledge, which has to be learned at first and, hence, requires additional computational time. However, our experiments show that the method is faster than the standard isomorphism-based pruning and produces fewer hypotheses without sacrificing the completeness of the search.

Finally, we consider the problem of utilizing imperfect rules for the knowledge base completion task, for which the standard entailment is too brittle. We propose a novel extension of an existing fragment-based inference algorithm by stratifying rules, which we learn using a heuristically driven top-down beam search suited for the inference algorithm. Our experiments show that the inference method is inferior to some others and, when accompanied by the heuristic rule learner, is more cautious than standard rule-based approaches applied in the inductive setting.

Keywords relational learning, mathematical discovery, rule learning, sentence space pruning, inference

ABSTRAKT

Relační učení využívá predikátovou logiku prvního řádu pro reprezentaci dat i modelů. Neoddiskutovatelnou výhodou tohoto formalismu je snadné vysvětlení naučených modelů a možnost reprezentovat širokou paletu problémů, kterými jsou například toxicita molekul nebo doplňování znalostní báze. Na druhou stranu, tento formalismus také přináší jistá úskalí a to zejména v případech aplikování predikátových logických hypotéz. Jejich samotné hledá je náročné kvůli velikému prostoru logických formulí, které plyne z predikátové logiky. Použití hypotéz ve formě predikátové logiky na reálných datech je těžký úkol sám o sobě, protože taková data často obsahují šum a vedou k ne zcela přesným hypotézám.

Tato práce cílí na dva výše zmíněné problémy. Zaměřuje se na prořezávání prostoru sentencí ve dvou oddělených úlohách. Cílem první úlohy je generování celočíselných sekvencích, které mohou být popsány pomocí predikátové logiky. Naše výsledky ukazují, že námi navržený nový přístup řešení této úlohy, který připomíná relační dolování vzorů, funguje díky prořezávání prostoru logických formulí. Náš přístup zatím vygeneroval databází celočíselných sekvencí o velikosti jedné sedminy velice populární *On-Line Encyclopedia of Integer Sequences* (OEIS) a dokonce byl schopen vygenerovat sekvence, které již v této populární databázi jsou. Cílem druhé úlohy je prořezávání prostoru hypotéz v relačních doménách, pro což jsme vytvořili novou metodu, která využívá znalosti obsažené v datech, např. symetrii určitých relací. Pravdou je, že tato znalost musí být nejdříve naučena z dat, nicméně, naše experimenty ukázaly, že i tato část výpočtu je kompenzována efektivitou tohoto prořezávání, které je rychlejší a produkuje méně logických hypotéz aniž by došlo k ohrožení úplnosti prohledávaného prostoru.

Posledním problémem, který v této práci řešíme, je použití nedokonalých hypotéz pro doplňování znalostní báze. Jejich přímočaré použití pomocí klasického odvozování z predikátové logiky prvního řádu je v tomto případě málo robustní. Proto jsme navrhli nové rozšíření metody založené na fragmentech dat, pro kterou jsme vyvinuli inferenční algoritmus. Naše rozšíření spočívá v prioritizaci pravidel, která se učíme pomocí heuristického shora-dolů paprskového prohledávání vyvinutého speciálně pro naší inferenční metodu. Experimenty ukazují, že naše inferenční metoda je v případě použití stejných pravidel lepší než ostatní inferenční metody a zároveň, ve spojení s heuristicky naučenými pravidly, je opatrnější než standardní pravidlové přístupy v indukčním nastavení.

Klíčová slova relační učení, matematické objevy, učení pravidel, prořezávání prostoru sentencí, inference

PUBLICATIONS

Some contents of this thesis have appeared previously in the following publications:

[1] **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 2023, 19th-25th August 2023, Macao, SAR, China*. ijcai.org, 2023, pp. 3338–3346. DOI: 10.24963/ijcai.2023/372. URL: https://doi.org/10.24963/ijcai.2023/372. WoS: 0, Scopus: 1, Google: 1.

[2] **Martin Svatoš**, Steven Schockaert, Jesse Davis, and Ondřej Kuželka. "STRiKE: Rule-Driven Relational Learning Using Stratified k-Entailment." In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. Ed. by Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 1515–1522. DOI: 10.3233/FAIA200259. URL: https://doi.org/10.3233/FAIA200259. WoS: 0, Scopus: 1, Google: 7.

[3] **Martin Svatoš**, Gustav Šourek, Filip Zelezný, Steven Schockaert, and Ondřej Kuželka. "Pruning Hypothesis Spaces Using Learned Domain Theories." In: *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*. Ed. by Nicolas Lachiche and Christel Vrain. Vol. 10759. Lecture Notes in Computer Science. Springer, 2017, pp. 152–168. DOI: 10.1007/978-3-319-78090-0_11. URL: https://doi.org/10.1007/978-3-319-78090-0%5C_11. WoS: 0, Scopus: 2, Google: 5.

[4] Gustav Šourek, **Martin Svatoš**, Filip Železný, Steven Schockaert, and Ondřej Kuželka. "Stacked Structure Learning for Lifted Relational Neural Networks." In: *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*. Ed. by Nicolas Lachiche and Christel Vrain. Vol. 10759. Lecture Notes in Computer Science. Springer, 2017, pp. 140–151. DOI: 10.1007/978-3-319-78090-0_10. URL: https://doi.org/10.1007/978-3-319-78090-0%5C_10. WoS: 0, Scopus: 4, Google: 12.

Any parts of the original papers reused verbatim in this thesis have been included with the approval of the co-authors.

ACKNOWLEDGMENTS

Foremost, my warmest acknowledgment belongs to Ondřej Kuželka for his endless well of ideas and support during my studies. Besides, I thank Gustav Šír and Filip Železný, who co-supervised me during my freshman years, and all of the co-authors, namely Steven Schockaert, Jesse Davis, Jan Tóth, Peter Jung, Yuyi Wang, and Giuseppe Marra. Finally, I would like to thank Peter Ryšavý, František Malinka, and Jáchym Barvínek for technical talks in the *loft* office.

I acknowledge the support of the following grants: Czech Science Foundation (projects no. 17-26999S, 20-19104Y, 23-07299S), OP VVV project *CZ.02.1.01/0.0/0.0/16_019/0000765* "Research Center for Informatics", Czech Technical University students grants (SGS17/189/OHK3/3T/13, SGS20/178/OHK3/3T/13, SGS23/184/OHK3/3T/13), and a generous donation from X-Order Lab.
Computatianl resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures", and OP VVV project *CZ.02.1.01/0.0/0.0/16_019/0000765* "Research Center for Informatics"

CONTENTS

ACRONYMS

AUC-PR	area under the precision-recall curve
CHED	cumulative Hamming error distance
FOL	first-order logic
GNN(s)	graph neural network(s)
HB	Herbrand base
ILP	inductive logic programming
LRNN(s)	lifted relational neural network(s)
MAP	maximum a posteriori probability
MLN(s)	Markov logic network(s)
OEIS	On-Line Encyclopedia of Integer Sequences
PAC	probably approximately correct
PCA	partial completeness assumption
PR	precision-recall
RL	relational learning
SRL	statistical relational learning
WL	Weisfeiler-Lehman labeling procedure
(W)FOMC	(weighted) first-order model counting

Part I

INTRODUCTION

1 INTRODUCTION

Inductive and deductive reasoning have attracted philosophers' minds since the age of Aristotle. However, one could argue that it is pretty common in a man's life. For example, consider a person who likes rainbows. After he had observed a dozen rainbows, he suddenly realized that a rainbow often occurs after rain.[1] The man actually induces this hypothesis from some observations.[2] Then, this rainbow-liker can watch the sky after every rain, hoping to see a rainbow. That is something he deduces from his hypothesis. Of course, the hypothesis is not perfect.[3] Nonetheless, it is the standard way of human drive to understand things.

Whereas the above-mentioned example could be formulated in several representations, e.g., propositional or modal logic, the choice of *suitable* representation is arguably the most important one in machine learning tasks. A typical scenario for a machine learning practitioner is to search for enough expressive representation while maintaining scalability for a task at hand. This thesis is focused on relational learning that uses function-free subset of first-order logic (FOL) for representation. In particular, this is suitable for structured data – these are everywhere around us, e.g., molecules and knowledge bases, which can be encoded as (hyper)graphs. Despite its restriction w.r.t. FOL, some of the reasoning processes fall into the NP-complete complexity class, e.g., θ-subsumption, which makes the reasoning computationally challenging; still, this is better than the undecidability of pure FOL. Although the field is relatively mature, both inductive and deductive reasoning parts of relational learning still contain challenging problems. One of the standard issues is the vast space of first-order logical formulae, e.g., Horn rules, that comes directly from the expressiveness of the representation. To deal with this issue, most of the methods employ some language bias, i.e., they consider only a subset of all possible hypotheses, which clearly makes them incomplete in finding all possible solutions.[4] Another issue arises in the presence of real-life datasets since logical formulae are effective in capturing hard constraints, e.g., integrity constraints in databases, but are not suited for handling exceptions and noisy data, which is typical for real-life data. Hence, the latter problem is often solved in a closely related field of statistical relational learning (SRL) that has emerged over the years to bridge the paradigms of relational learning and statistical learning.

The aim of this thesis is to investigate the possibility of relaxing deductive and inductive reasoning in relational learning. In particular, we focus on pruning the space of logical formulae to make the inductive reasoning part faster. Further, we focus on deductive reasoning in the presence of imperfect rules.

1 There are languages where *rainbow* contains the word *rain*, e.g., English and German; however, this does not hold all languages, e.g., French.
2 Most likely, he does not know the term *induction*.
3 Indeed, this is only partially true due to the fact that to see a rainbow, a viewer's position w.r.t. the Sun has to be taken into account.
4 Thought, some of such restrictions remove clearly redundant hypotheses, which is favorable.

1.1 A BRIEF OVERVIEW OF RELATIONAL LEARNING

Despite the current progress in artificial intelligence, which is mainly driven by deep learning, logic has been the main tool since the early days of artificial intelligence. This was partially due to the long tradition of mathematical logic and the ability of logic to capture problems and reason about them. However, since the 1980s [78], researchers have started to shift from propositional logic to FOL for its more expressive representation suitable for various problems. The rise of relational learning, which combines machine learning with the representation based on FOL, started with the wave of inductive logic programming (ILP) in the 1990s [83].

Contrary to relational learning, ILP, which was extensively studied in the past few decades, allows hypotheses to contain functions. The general task in ILP can be formulated as follows: given a training set of positive $\mathcal{E}^+$ and negative $\mathcal{E}^-$ examples, learn a theory $\mathcal{T}$ that *discriminates* these two sets. The exact definition of examples and the *discrimination* relation differ from one learning setting to another. One of the typical ILP learning settings is learning from entailment, which employs the following definition: an *example* is in the form of a clause, $\mathcal{T}$ is a clausal theory, and the *discriminator relation* is logical entailment. Then, the task is to learn $\mathcal{T}$ such that

$$\begin{aligned} &\mathcal{T} \models e && \forall e \in \mathcal{E}^+ \\ &\mathcal{T} \not\models e && \forall e \in \mathcal{E}^- \end{aligned}$$

Another quite popular learning setting is learning from interpretation where an *example* is in the form of a Herbrand interpretation, $\mathcal{T}$ is a clausal theory, and the *discriminator relation* is satisfied if and only if an example is a model of $\mathcal{T}$.

$$\begin{aligned} &e \models \mathcal{T} && \forall e \in \mathcal{E}^+ \\ &e \not\models \mathcal{T} && \forall e \in \mathcal{E}^- \end{aligned}$$

The main difference between these two is that in the latter case, the interpretation fully specifies the example. ILP and both of these learning settings were studied and used for a number of applications besides toy-like problems [14]. One of the most known applications is the task of finding a sub-structure that occurs in dangerous molecules while being absent from the rest of the molecules at hand [118]. Another interesting application of ILP was the HR system [25] dedicated to mathematical discovery.

Problems within the above-mentioned learning settings usually reduce to searching for a set of logical theories, i.e., a set of logic formulae. Since the space of formulae is inherently enormous, various approaches to pruning are usually employed. The most typical approach is to apply *language* bias, which considers only some syntactical forms of formula to be acceptable, as well as semantics-based biases [85], e.g., minimal support of a hypothesis, usage of variables, etc. Many, but not all, approaches are interested in Horn clauses, which can be easily interpreted as rules and hence are often called *rule learners*. The past decades brought many of these algorithms with various search strategies, e.g., top-down, bottom-up, (in)complete, having either ideal or optimal refinement operator, etc. Besides the evolution of underlying engines, starting with Prolog-based methods [117], through in-memory-based [39], all up to database-based [22], there was also a shift in the focus of application through the decades. The most typical applications are program synthesis [84], predicate invention [26], knowledge graph completion [75], and data mining [33]. However, these problems are often tackled by researchers from respective fields by enriching their language to (a subset) of FOL, which consequently means that relational learning, ILP, and SRL often overlap with other approaches on some tasks. A natural example of such overlap is *mining multi-relational data mining* [36], which can be seen as the intersection of ILP and data mining.

Unfortunately, ILP is not suited for all kinds of problems due to several reasons. Firstly, the learning task does not incorporate any measure of performance on unseen data. Secondly, the crisp logic does not handle uncertainty and noise well; these are often present in real-life data. Hence, one can argue that it is applicable only to idealized scenarios. There are limited ways ILP can compensate for this. A straightforward way is to allow some degree of misclassification of learned hypotheses [84] or to apply *propositionalization*. The latter paradigm is based on transforming the problem into an attribute-value learning setting and using various methods from that area. The transformation can be simply viewed as constructing a vector of features for each training example where features are learned using first-order pattern mining algorithms [33].

However, the above-mentioned does not imply that we cannot reason with relational representation under uncertainty. On the contrary, this is the main goal of statistical relational learning (SRL) [108], which often enhances clauses with weights and learns a probability distribution of the data. Many methods have been developed over the years, e.g., Markov Logic Networks (MLNs) [111], Bayesian logic programs [49], and ProbLog [31]. The learning task is typically split into two main components – *structure* and *parameters* learning. The structure learning mimics the ILP search for hypothesis [51] and, hence, suffers from the same issues we discussed for hypotheses learning. In the second component, the parameters of a model are learned as an optimization task that maximizes (pseudo-)likelihood of the data.

There are other approaches with more or less restricted FOL language beside and within relational learning, e.g., OWL and RDF. We discuss relevant related work for each task we solve in Part iii at the end of every chapter.

1.2 PROBLEM STATEMENT

The problem statement of this thesis is, as the title suggests, to *exploit the relaxation of deductive and inductive reasoning in relational learning*. Indeed, this sounds like a general problem; hence, we narrowed the topic to be more specific. In particular, we will be interested in how, if even possible, the relaxation of either one of these helps to

- *scale up the process of sentence space traversal*, and
- *make the inference process robust in the presence of imperfect rules*.

Rather than focusing on developing a general theory, we will study these problems in isolation. Namely, the first two proposed methods, presented in Chapters 4 and 5, prune the space of logical formulae to provide better scalability for particular problems while preserving the completeness of the search algorithms. The latter question is investigated in Chapter 6, where we present an extension of a fragment-based inference method and design a heuristic rule learner specially for it. For each of these, we will tweak either standard induction, deduction, or both reasoning processes to achieve the goal.

1.3 STRUCTURE OF THE THESIS

The thesis is structured as follows. After this introduction Chapter 1, Part ii covers the basic notation and concepts in Chapter 2, followed by Chapter 3, which presents algorithms that repeat as sub-routines in the main content of this work. Part iii contains three chapters, each proposing a novel method for a given problem. Chapter 4 presents a method for discovering combinatorial integer sequences using a fragment of first-order logic. Chapter 5 proposes a method that prunes hypothesis space automatically using domain knowledge of the data. Chapter 6 devises an inference algorithm suited for reasoning with imperfect rules. In Part iv, we briefly discuss the applications of Chapter 4 and Chapter 5 in Chapter 7 and Chapter 8, respectively. In Part v, the contribution of the proposed method is discussed in Chapter 9. The appendix in Appendices A to C covers some technical details and additional material for Chapter 4.

Part II

THEORETICAL BACKGROUND

2 THEORETICAL FOUNDATION

This chapter contains theoretical foundations for the thesis.

2.1 FIRST-ORDER LOGIC

We work with a function-free subset of first-order logic. The language is defined by a finite set of *constants* $\mathcal{C}$, a finite set of *variables* $\mathcal{V}$ and a finite set of *predicates* $\mathcal{P}$. An *atom* has the form $P(t_1, \dots, t_k)$ with k-arity predicate symbol $P/k \in \mathcal{P}$ and terms $t_i \in \mathcal{C} \cup \mathcal{V}$. A *literal* is an atom or its negation. A *formula* is an atom and a literal. More complex formulae may be formed from existing formulae by logical connectives, e.g., $\wedge$, $\vee$, and $\Rightarrow$, or by surrounding them with a universal ($\forall x$), an existential ($\exists x$), or a counting ($\exists^{=k}$) quantifier where $x \in \mathcal{V}$. A variable x in a formula is called $\mathtt{free}$ if the formula contains no quantification over x. A formula is called a $\mathtt{sentence}$ if it contains no free variables. A formula is called *ground* if it contains no variables. Predicates start with an uppercase, while variables and constants start with a lowercase. Variables are denoted by a single letter from the end of the alphabet and usually preceded by a quantifier at the start of a sentence, e.g., $\forall x\ \varphi(x)$. Constants are denoted by a word or a single letter from the start of the alphabet, e.g., *ann* or *c*.

A clause α is a universally quantified disjunction of literals $\forall x_1 ... \forall x_n\ \tau_1 \vee ... \vee \tau_k$, such that $x_1, .., x_n$ are the only variables occurring in the literals $\tau_1, ..., \tau_k$. For ease of presentation, we will sometimes identify a clause α with the corresponding set of literals $\{\tau_1, ..., \tau_k\}$ and $|\alpha|$ denoting the number of literals of α. The set of variables occurring in a clause α is written as *vars*(α), the set of all terms as *terms*(α), and the set of all constants *const*(α). For a clause α, we define the *sign flipping* operation as

$$\widetilde{\alpha} \overset{\mathrm{def}}{=} \bigvee_{\tau \in \alpha} \tilde{\tau},$$

where $\tilde{\tau} = \neg\tau$ and $\widetilde{\neg\tau} = \tau$ for an atom τ. In other words, the sign flipping operation simply replaces each literal by its negation. A clause is said to be connected if it cannot be written as a disjunction of two non-empty clauses.

Example 1. *Consider clauses*

$$\alpha = \forall x \forall y\ P_1(x) \vee P_2(y) \vee P_3(x, y, a)$$
$$\beta = \forall x \forall y\ P_1(x) \vee \neg P_2(y)$$

then

$$\begin{aligned}
const(\alpha) &= \{a\} \\
const(\beta) &= \emptyset \\
vars(\alpha) &= vars(\beta) = \{x, y\} \\
terms(\alpha) &= \{a, x, y\} \\
terms(\beta) &= \{x, y\} \\
|\alpha| &= 3 \\
|\beta| &= 2 \\
\widetilde{\alpha} &= \forall x \forall y\ \neg P_1(x) \lor \neg P_2(y) \lor \neg P_3(x, y, a) \\
\widetilde{\beta} &= \forall x \forall y\ \neg P_1(x) \lor P_2(y)
\end{aligned}$$

Both α and β are sentences since there is no free variable. Also, note that α is a connected clause while β is not.

Fact is a ground positive literal. A *definite rule* is a clause which has exactly one positive literal; note that facts are special cases of definite rules. A *hard constraint* is a clause that has no positive literals. To improve readability, we will usually write a definite rule

$$\forall x_1, ..., \forall x_m\ \neg b_1 \lor \cdots \lor \neg b_n \lor h$$

as

$$\forall x_1, ..., \forall x_m\ b_1 \land \cdots \land b_n \Rightarrow h.$$

As usual, we call $b_1 \land ... \land b_n$ the body of the rule and h the head. Range-restricted rules satisfy $vars(h) \subseteq vars(b_1 \land ... \land b_n)$. We often refer to a set of sentences Φ as a *rule set* or *theory* if it contains definite rules or sentences, respectively.

Example 2. *The fact*

$$\mathsf{Smokes}(alice)$$

asserts that alice smokes, the range-restricted rule

$$\forall x \forall y\ \mathsf{Smokes}(x) \land \mathsf{Friends}(x, y) \Rightarrow \mathsf{Smokes}(y)$$

asserts that anyone who is friends with someone who smokes is also a smoker, and the hard constraint

$$\forall x\ \neg \mathsf{Friends}(x, x)$$

states that no one can be friends with themselves.

A substitution θ is a mapping from variables to terms. For a clause α, we write $\alpha\theta$ for the clause $\{\tau\theta \mid \tau \in \alpha\}$, where $\tau\theta$ is obtained by replacing each occurrence in τ of a variable x by the corresponding term $\theta(x)$. A grounding substitution is a substitution in which each variable is mapped to a constant. Clearly, if θ is a grounding substitution, then for any literal τ it holds that $\tau\theta$ is ground. For a clause α and a set of clauses Γ, we say that $G_\Gamma(\alpha)$ is the set of all grounding substitutions θ that can be constructed from variables occurring in α and constants in Γ.

Example 3. *Let* Γ *be a set of facts and* α *be a set of clauses*

$$\Gamma = \{\mathsf{Happy}(\mathit{ann}), \mathsf{Happy}(\mathit{liz})\},$$
$$\alpha = \forall x \forall y\ \mathsf{Friends}(x, y) \vee \mathsf{Happy}(y).$$

Then $G_\Gamma(\alpha)$ *is equal to*

$$\begin{aligned}&\{\{x \mapsto \mathit{ann}, y \mapsto \mathit{ann}\},\\ &\ \{x \mapsto \mathit{ann}, y \mapsto \mathit{liz}\},\\ &\ \{x \mapsto \mathit{liz}, y \mapsto \mathit{ann}\},\\ &\ \{x \mapsto \mathit{liz}, y \mapsto \mathit{liz}\}\}.\end{aligned}$$

For example, applying the second substitution, we obtain $\mathsf{Friends}(\mathit{ann}, \mathit{liz}) \vee \mathsf{Happy}(\mathit{liz})$.

If α and β are clauses then we say that α θ-*subsumes* β (denoted $\alpha \preceq_\theta \beta$) if and only if there is a substitution θ such that $\alpha\theta \subseteq \beta$. If $\alpha \preceq_\theta \beta$ and $\beta \preceq_\theta \alpha$, we call α and β θ-*equivalent* (denoted $\alpha \approx_\theta \beta$). Note that the $\approx_\theta$ relation is indeed an equivalence relation, i.e., it is reflexive, symmetric and transitive. A clause α is said to be θ-*reducible* if there exists a clause β such that $\alpha \approx_\theta \beta$ and $|\beta| < |\alpha|$.

Clauses α and β are said to be *isomorphic* (denoted $\alpha \approx_{iso} \beta$) if there exists an injective substitution θ such that $\alpha\theta = \beta$.[1] We say that α OI-subsumes β (denoted $\alpha \preceq_{OI} \beta$ [37]) if there is an injective substitution such that $\alpha\theta \subseteq \beta$. Note that α is isomorphic to β if and only if $\alpha \preceq_{OI} \beta$ and $\beta \preceq_{OI} \alpha$.

Example 4. *Let us consider the following four clauses:*

$$\begin{aligned}\alpha_1 &= \forall x \forall y\ P_1(x, y) \vee \neg P_2(x, y)\\ \alpha_2 &= \forall x \forall y \forall z\ P_1(x, y) \vee \neg P_2(x, y) \vee \neg P_2(x, z)\\ \alpha_3 &= \forall x \forall y \forall w\ P_1(x, y) \vee \neg P_2(x, y) \vee \neg P_2(x, w)\\ \alpha_4 &= \forall x \forall y\ P_1(x, y) \vee \neg P_3(x, y)\end{aligned}$$

Then we can verify that $\alpha_1 \approx_\theta \alpha_2 \approx_\theta \alpha_3$, *and thus also* $\alpha_i \preceq_\theta \alpha_j$ *for* $i, j \in \{1, 2, 3\}$). *While some of these relations are easy to see, e.g.,* $\alpha_1 \preceq_\theta \alpha_2$, *some are little bit more advanced, e.g.,* $\alpha_3 \preceq_\theta \alpha_1$ *with* $\theta = \{x \mapsto x, y \mapsto y, w \mapsto y\}$.

We also have $\alpha_1 \not\approx_{iso} \alpha_2$, $\alpha_1 \not\approx_{iso} \alpha_3$, *as well as* $\alpha_i \not\preceq_\theta \alpha_4$ *and* $\alpha_4 \not\preceq_\theta \alpha_i$ *for any* $i \in \{1, 2, 3\}$. *The only two isomorphic clauses are* α_2 *and* α_3, *i.e.,* $\alpha_2 \approx_{iso} \alpha_3$, *since the variable* w *can map to the variable* z *and vice versa.*

Finally, we also have $\alpha_1 \preceq_{OI} \alpha_i$ *for* $i \in \{1, 2, 3\}$, $\alpha_2 \preceq_{OI} \alpha_3$, *and* $\alpha_3 \preceq_{OI} \alpha_2$. *The clauses* α_2 *and* α_3 *are* θ-*reducible.*

For a set of constants $\mathcal{C}$, $\Phi[\mathcal{C}]$ denotes the set of clauses obtained from Φ by removing all clauses that contain a constant $c \notin \mathcal{C}$.

Example 5. *Let* Φ *be a set of clauses and* $\mathcal{C}$ *be a set of constants*

$$\Phi = \{\mathsf{Friends}(\mathit{alice}, \mathit{bob}), \mathsf{Smokes}(\mathit{alice})\},$$
$$\mathcal{C} = \{\mathit{alice}\}.$$

Then

$$\Phi[\mathcal{C}] = \{\mathsf{Smokes}(\mathit{alice})\}.$$

1 This defines *variable isomorphism* among clauses. We will use this term heavily.

As is customary in computer science, we adopt the *Herbrand semantics* [44] with a finite domain. We use HB to denote the *Herbrand base*, i.e., the set all ground atoms. We use ω to denote a *possible world*, i.e., any subset of HB. Elements of a possible world are assumed to be true, all others are assumed to be false. A clause $\alpha = \{\tau_1, ..., \tau_n\}$ is satisfied by a possible world ω, written

$$\omega \models \alpha,$$

if for each grounding substitution θ, it holds that $\{\tau_1\theta, ..., \tau_n\theta\} \cap \omega \neq \emptyset$. In particular, note that a ground literal τ is satisfied by ω if $\tau \in \omega$. The satisfaction relation $\models$ is extended to (sets of) propositional combinations of clauses in the usual way. If $\omega \models \mathcal{A}$, for $\mathcal{A}$ a propositional combination of clauses, we say that ω is a model of $\mathcal{A}$. If $\mathcal{A}$ has at least one model, we say $\mathcal{A}$ is satisfiable. For two sets of formulae Φ and Υ, we write $\Phi \models \Upsilon$ if every model of Φ is also model of Υ. If $\Phi \models \Upsilon$ and $\Upsilon \models \Phi$, we write $\Phi \models\mid \Upsilon$. Note that if $\alpha \preceq_\theta \beta$ for clauses α and β, then $\alpha \models \beta$ but the converse does not hold in general. Finally, $\top$ and $\bot$ are used to denote tautology and contradiction, respectively.

Note that classical entailment is tractable when restricted to facts and definite rules. Then, we can use forward chaining to compute the set of facts that are entailed by a given set of clauses. It also means that the inference process can be easily explained to users who are not familiar with formal logic.

Example 6. *Forward chaining iteratively expands the given set of facts by applying (groundings of) rules whose body is satisfied by the facts that have already been derived. More precisely, if all the literals in a ground rule's body are included in the set of facts, then its head can be added to the set of facts. This process is repeated until no further facts can be derived. To illustrate this, consider the following rule set:*

$$\begin{aligned}\Phi = \{&\forall x \forall y \forall z\ \mathsf{BornIn}(x, y) \wedge \mathsf{PartOf}(y, z) \Rightarrow \mathsf{BornIn}(x, z),\\ &\forall x \forall y\ \mathsf{BornIn}(x, y) \wedge \mathsf{Country}(y) \Rightarrow \mathsf{Nationality}(x, y)\}\end{aligned}$$

and the following set of facts:

$$\mathcal{E}_1 = \{\mathsf{BornIn}(\textit{alice}, \textit{sdC}), \mathsf{Country}(\textit{spain}), \mathsf{PartOf}(\textit{sdC}, \textit{spain})\}.$$

We can use the first rule to derive

$$\mathsf{BornIn}(\textit{alice}, \textit{spain}).$$

Together with the fact $\mathsf{Country}(\textit{spain})$ *from* $\mathcal{E}_1$, *we can now apply the second rule to derive*

$$\mathsf{Nationality}(\textit{alice}, \textit{spain}).$$

At this point, no further facts can be derived.

2.2 WEIGHTED FIRST-ORDER MODEL COUNTING

In one part of this thesis, we will require access to fast computation of model counts of first-order logic formulae. For that, we will make use of the weighted first-order model counting (WFOMC) problem [136].

Definition 1. *(Weighted First-Order Model Counting) Let* φ *be a sentence over some relational language* $\mathcal{L}$*. Let* HB *denote the Hebrand base of* $\mathcal{L}$ *over some domain of size* $n \in \mathbb{N}$*. Let* $\mathcal{P}$ *be the set of the predicates of the language* $\mathcal{L}$ *and let* $\mathsf{pred} : \mathsf{HB} \mapsto \mathcal{P}$ *map each atom to its corresponding predicate symbol. Let* $w : \mathcal{P} \mapsto \mathbb{R}$ *and* $\overline{w} : \mathcal{P} \mapsto \mathbb{R}$ *be a pair of weightings assigning a positive and a negative weight to each predicate in* $\mathcal{L}$*. We define*

$$\mathsf{WFOMC}(\varphi, n, w, \overline{w}) = \sum_{\omega \subseteq \mathsf{HB} : \omega \models \varphi} \prod_{l \in \omega} w(\mathsf{pred}(l)) \prod_{l \in \mathsf{HB} \setminus \omega} \overline{w}(\mathsf{pred}(l)).$$

Example 7. *Consider the sentence*

$$\varphi = \forall x\, \neg \mathsf{Edge}(x, x)$$

and the weights $w(\mathsf{Edge}) = \overline{w}(\mathsf{Edge}) = 1$*. Since all the weights are unitary, we simply count the number of models of* φ*. We can interpret the sentence as follows: Each constant of the language is a vertex. Each atom* $\mathsf{Edge}(a, b) \in \mathsf{HB}$ *with* $\{a, b\} \subseteq \mathcal{C}$ *denotes an edge from a to b. Furthermore, the sentence prohibits reflexive atoms, i.e, loops. Overall, the models of* φ *will be all directed graphs without loops on* n *vertices. Hence, we obtain*

$$\mathsf{WFOMC}(\varphi, n, w, \overline{w}) = 2^{n^2 - n}.$$

Example 8. *Consider the sentence*

$$\varphi = \exists x\, \mathsf{Heads}(x)$$

and the weights $w(\mathsf{Heads}) = 4, \overline{w}(\mathsf{Heads}) = 1$*. Now, we can consider each domain element to be the result of a coin flip. The sentence requires that there is at least one coin flip with the value of "heads" (there exists a constant* $a \in \mathcal{C}$ *such that* $\mathsf{Heads}(a)$ *is an element of the model). Suppose we have* $i > 0$ *"heads" in the model. Then, the model's weight will be* $4^i \cdot 1^{n-i} = 4^i$ *and there will be* $\binom{n}{i}$ *such models. Therefore,*

$$\mathsf{WFOMC}(\varphi, n, w, \overline{w}) = \sum_{i=1}^{n} 4^i \cdot \binom{n}{i} = 5^n - 1.$$

2.2.1 *WFOMC in the Two-Variable Fragment*

Limiting the number of variables in each sentence to at most two we obtain a language known as $\mathbf{FO}^2$. This fragment of first-order logic allows computing WFOMC in time polynomial in the domain size [134, 135]. We provide a brief overview of that tractability result.

When computing WFOMC in a lifted manner, we seek to avoid grounding the problem as much as possible. Grounding first-order sentences often exponentially enlarges the problem and inherently leads to many symmetrical subproblems.

Example 9. *Consider the sentence*

$$\varphi = \forall x\, \mathsf{Smokes}(x) \Rightarrow \mathsf{Cancer}(x).$$

Grounding the sentence on the domain of size $n \in \mathbb{N}$ *will produce a conjunction of* n *implications. Each of those implications will have three models with* *atoms completely different from the atoms in models of the other implications. Moreover, there will be bijections between the models of different implications. Overall, we could have computed the model count in a much simpler way. For one particular constant, there will be three distinct models. Since there are n constants, the final model count will be 3^n.*

To avoid repeating the computations on such symmetrical instances, we aim to decompose the WFOMC problem into mutually independent parts with each needed to be solved only once. *Cells* of a logical sentence whose WFOMC is to be computed allow such decomposition.

Definition 2 (Cell). *A cell of an* ***FO***^{2} *formula* φ *is a maximal consistent conjunction of literals formed from atoms in* φ *using only a single variable.*

Example 10. *Consider the formula*

$$\varphi = \forall x \forall y \; \mathsf{Smokes}(x) \land \mathsf{Friends}(x, y) \Rightarrow \mathsf{Smokes}(y).$$

Then there are four cells:

$$\begin{aligned}
C_1(x) &= \mathsf{Smokes}(x) \land \mathsf{Friends}(x, x),\\
C_2(x) &= \neg\mathsf{Smokes}(x) \land \mathsf{Friends}(x, x),\\
C_3(x) &= \neg\mathsf{Smokes}(x) \land \neg\mathsf{Friends}(x, x),\\
C_4(x) &= \mathsf{Smokes}(x) \land \neg\mathsf{Friends}(x, x).
\end{aligned}$$

To simplify the WFOMC computation, we condition on cells in the following way:

$$\begin{aligned}
\psi_{ij}(x, y) &= \varphi(x, y) \land \varphi(y, x) \land C_i(x) \land C_j(y),\\
\psi_k(x) &= \varphi(x, x) \land C_k(x).
\end{aligned}$$

And we compute

$$\begin{aligned}
r_{ij} &= \mathsf{WMC}(\psi_{ij}(a, b), w', \overline{w}'),\\
w_k &= \mathsf{WMC}(\psi_k(a), w, \overline{w}),
\end{aligned}$$

where WMC is simply the propositional version of WFOMC, $\{a, b\} \subseteq \mathcal{C}$ and the weights $(w', \overline{w}')$ are the same as $(w, \overline{w})$ except for the atoms appearing in the cells conditioned on. Those weights are set to one, since the weights of the unary and binary reflexive atoms are already accounted for in the w_k terms. Note that $r_{ij} = r_{ji}$.

Now, assuming there are p distinct cells, we can write

$$\mathsf{WFOMC}(\varphi, n, w, \overline{w}) = \sum_{\mathbf{k} \in \mathbb{N}^p : |\mathbf{k}| = n} \binom{n}{\mathbf{k}} \prod_{i, j \in [p] : i < j} r_{ij}^{(\mathbf{k})_i (\mathbf{k})_j} \prod_{i \in [p]} r_{ii}^{\binom{(\mathbf{k})_i}{2}} w_i^{(\mathbf{k})_i}. \tag{1}$$

2.2.1.1 *Lifting Skolemization*

However, the approach above is only applicable for universally quantified **FO**^{2} sentences. To get rid of existential quantifiers in the input formula, we can utilize specialized *Skolemization*. In general, the procedure eliminates existential quantifiers by introducing new (fresh) *Skolem* predicates Sk with $w(\mathsf{Sk}) = 1$ and $\overline{w}(\mathsf{Sk}) = -1$ [9]. Specifically, given a sentence

$$\varphi = \forall x \exists y \ \alpha(x, y)$$

where α is a quantifiers-free disjunction of literals, the method introduces a fresh predicate Sk, i.e.

$$\forall x \ (\exists y \ \alpha(x, y) \Rightarrow \mathsf{Sk}(x))$$

which is equal to

$$\varphi' = \forall x \forall y \ \neg\alpha(x, y) \lor \mathsf{Sk}(x),$$

and sets up the weights, i.e., $w(\mathsf{Sk}) = 1$ and $\overline{w}(\mathsf{Sk}) = -1$. Then, it holds that

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

where

$$\begin{aligned} w(\mathsf{Edge}) &= w'(\mathsf{Edge}), & w'(\mathsf{Sk}) &= 1, \\ \overline{w}(\mathsf{Edge}) &= \overline{w}'(\mathsf{Edge}), & \overline{w}'(\mathsf{Sk}) &= -1. \end{aligned}$$

Informally, this approach is based on inclusion and exclusion of model counts; we refer to [9] for justification. This process is repeated until all existential quantifiers are eliminated.

Due to Equation (1) combined with the specialized Skolemization procedure, WFOMC can be evaluated in time polynomial in n for any $\mathbf{FO}^2$ sentence.

Example 11. *Consider the sentence*

$$\varphi = \forall x \exists y \ \mathsf{Edge}(x, y).$$

Applying the above mentioned Skolemization, we obtain

$$\varphi' = \forall x \forall y \ \neg\mathsf{Edge}(x, y) \lor \mathsf{Sk}(x)$$

with $w(\mathsf{Sk}) = 1$ *and* $\overline{w}(\mathsf{Sk}) = -1$.

Example 12. *Consider the sentence*

$$\varphi = \forall x \exists y \ \mathsf{Edge}(x, y) \lor \mathsf{Black}(y).$$

Applying the above mentioned Skolemization, we obtain

$$\varphi' = \forall x \forall y \ \neg(\mathsf{Edge}(x, y) \land \mathsf{Black}(y)) \lor \mathsf{Sk}(x)$$

with $w(\mathsf{Sk}) = 1$ *and* $\overline{w}(\mathsf{Sk}) = -1$, *which can be expanded to a set of sentences, i.e.*

$$\begin{aligned} \Phi' = \{ & \forall x \forall y \ \neg\mathsf{Edge}(x, y) \lor \mathsf{Sk}(x), \\ & \forall x \forall y \ \neg\mathsf{Balck}(y) \lor \mathsf{Sk}(x)\}. \end{aligned}$$

2.2.2 *WFOMC in the Two-Variable Fragment with Counting Quantifiers*

Although the language of $\mathbf{FO}^2$ permits a polynomial-time WFOMC computation, its expressive power is naturally quite limited. The search for a larger logical fragments still permitting a polynomial complexity is a subject of active research. One possibility to extend the $\mathbf{FO}^2$ while preserving its tractable property is by adding *counting quantifiers*. Such language is known as $\mathbf{C}^2$ and its tractability was shown by [54].

Counting quantifiers are a generalization of the traditional existential quantifier. For a variable $x \in \mathcal{V}$, we allow usage of a quantifier of the form $\exists^{=k}x$, where $k \in \mathbb{N}$.[2] Satisfaction of formulae with counting quantifiers is defined naturally. For example, $\exists^{=k}x\ \psi(x)$ is satisfied in ω if there are exactly k constants $\{ c_1, c_2, \dots, c_k \} \subseteq \mathcal{C}$ such that $\omega \models \psi(c_i)$ if and only if $1 \leqslant i \leqslant k$.

To handle counting quantifiers, [54] suggested evaluating WFOMC repeatedly on many *points*. The values would be subsequently used in a polynomial interpolation. Instead, we work with symbolic weights, which is, for all our purposes, equivalent to the polynomial interpolation. We simply obtain the would-be-interpolated polynomial directly.[3]

When we have a sentence with counting quantifiers whose WFOMC is to be computed, first thing to do is to convert the counting quantifiers to the traditional existential quantifier. That can be achieved using yet another syntactic construct known as *cardinality constraints*. We allow the formula to contain an *atomic formula* of the form $(|P| = k)$,[4] where $P \in \mathcal{P}$ is a predicate and $k \in \mathbb{N}$. Intuitively speaking, a cardinality constraint enforces that all models of a sentence contain exactly k atoms with the predicate P.

Example 13. *Consider the sentences*

$$\begin{aligned} \varphi &= \forall x \exists^{=1} y \ \mathsf{Edge}(x,y), \\ \varphi' &= (\forall x \exists y \ \mathsf{Edge}(x,y)) \land (|E| = n). \end{aligned}$$

Then it holds that

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

for any weights $(w, \overline{w})$.

Using transformations such as the one shown in Example 13, WFOMC of a $\mathbf{C}^2$ sentence φ can be reduced to WFOMC of the sentence

$$\varphi' = \psi \land \bigwedge_{i=1}^{m} (|P_i| = k_i),$$

where ψ is an $\mathbf{FO}^2$ sentence. Then, for each cardinality constraint $(|P_i| = k_i)$, we define $w'(P_i) = x_i$, where x_i is a new symbolic variable. For predicates

2 [54] actually proved the tractability for a more general version of the counting quantifiers, i.e., $\exists^{\bowtie k}x$, where $\bowtie \in \{ <, \leqslant, =, \geqslant, > \}$. However, the counting with inequalities does not scale very well – in fact, even the equalities turn out to be computationally challenging – so we only work with equality in our counting quantifiers.

3 In Definition 1, WFOMC is defined for real-valued weights only. However, the extension to (multivariate) polynomials is natural and does not break anything.

4 Similarly to counting quantifiers, cardinality constraints can be generalized to $(|P| \bowtie k)$ with $\bowtie \in \{<, \leqslant, =, \geqslant >\}$. See [54] for the full treatment.

$Q \in \mathcal{P}$, which do not occur in any cardinality constraint, we leave the positive weight unchanged, i.e., $w'(Q) = w(Q)$.

Finally, we are ready to compute $\mathsf{WFOMC}(\psi, n, w', \overline{w})$. The result will be a multivariate polynomial over the symbolic variables introduced for each cardinality constraint. However, only one of its monomials will carry the information about the actual WFOMC of the original $\mathbf{C}^2$ sentence.[5] Namely, the monomial

$$A \cdot \prod_{i=1}^{m} x_i^{e_i}$$

such that $e_i = k_i$ for each cardinality constraint $(|P_i| = k_i)$.

Now, we can report the final WFOMC result of the original $\mathbf{C}^2$ sentence φ. Nevertheless, we must still account for the positive weights that were replaced by symbolic variables when dealing with cardinality constraints. Hence,

$$\mathsf{WFOMC}(\varphi, n, w, \overline{w}) = A \cdot \prod_{i=1}^{m} w(P_i)^{k_i}.$$

Example 14. *Consider the sentence*

$$\varphi = \forall x \exists^{=1} y \; \mathsf{Edge}(x, y),$$

domain of size $n = 5$ *and* $w(\mathsf{Edge}) = \overline{w}(\mathsf{Edge}) = 1$. *Let us compute* $\mathsf{WFOMC}(\varphi, n, w, \overline{w})$.

First, we get rid of the counting quantifier:

$$\varphi' = (\forall x \exists y \; \mathsf{Edge}(x, y)) \wedge (|\mathsf{Edge}| = 5)$$

Second, we introduce a symbolic variable x *as the positive weight of the* Edge *predicate:*

$$w'(\mathsf{Edge}) = x$$

Finally, we evaluate $\mathsf{WFOMC}(\forall x \exists y \; \mathsf{Edge}(x, y), n, w', \overline{w})$ *and extract the coefficient of the term where* x *is raised to the fifth power:*

$$\mathsf{WFOMC}(\varphi, n, w, \overline{w}) = 3125.$$

Let us check the obtained result. We can interpret the formula φ *as a directed graph with each vertex having exactly one outgoing edge. For each vertex, there are* n *vertices that it could be connected to. Hence, there are* n^n *such graphs. For* $n = 5$, *we obtain* $5^5 = 3125$.

2.3 LEARNING SETTINGS

In this section, we describe the learning settings used in this thesis in detail.

2.3.1 *Learning from Interpretations*

In the classical setting of *learning from interpretations* [107], examples are interpretations and hypotheses are clausal theories, i.e., conjunctions of clauses.

5 When using counting quantifiers $\exists^{=k}$ with $k > 0$, we also need to take care of overcounting, which is described in detail in [54].

An example e is said to be *covered* by a hypothesis α if e is a model of α; we denote this by

$$e \models \alpha.$$

Given a set of positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$, the training task is then to find a hypothesis α from some class of hypotheses $\mathcal{A}$ which optimizes a given scoring function, e.g., training error. For ease of presentation, we will restrict ourselves to classes $\mathcal{A}$ of hypotheses in the form of clausal theories without constants, as constants can be emulated by unary predicates (since we do not consider functions). Note that examples are fully specified in this setting, i.e., there is no missing information since the truth value of each fact must be known.

2.3.2 *Knowledge Base Completion*

In the knowledge base completion task, there is a single training example $\mathcal{E}$ that fully describes the world, i.e., a set of facts that are true, while the rest are false. Then, the task is to learn a model, using $\mathcal{E}$, that predicts missing facts from Υ, i.e., to decide whether each single missing fact[6] in Υ holds or not. Both structures $\mathcal{E}$ and Υ share the same set of predicates.[7]

In this thesis, we will be interested in the *inductive* setting of the problem; that is a setup where $\mathcal{E}$ and Υ do not share any constant (entity), i.e., $\mathit{const}(\mathcal{E}) \cap \mathit{const}(\Upsilon) = \emptyset$. On the contrary, a popular version of this problem is knowledge graph completion in the *transductive* setting where both $\mathcal{E}$ and Υ may share constants and sometimes are even equal, i.e., $\mathcal{E} = \Upsilon$; hence, the training example is imperfect in this case. This is closely related to the open world assumption, e.g., [39], whereas we consider the training example to be fully specified, i.e., closed world assumption. The knowledge graph problem itself is a restriction that allows only binary relations,[8] whereas we allow arbitrary non-zero n-ary relations, e.g., unary and ternary predicates.

6 With respect to constants and predicates occurring in Υ.

7 For readers familiar with description logic: this setup differs from the one you are used to – see, there is no T-Box, and the A-Box can contain predicates with arbitrary arity.

8 With the facts usually being described as a triplet of *object-relation-subject* instead of FOL language.

3 AUXILIARY ALGORITHMS

This chapter describes several algorithms that are used in the main part of thesis as common routines.

3.1 θ-SUBSUMPTION ENGINE AND ISOMORPHISM CHECKING

Deciding θ-subsumption between two clauses is an NP-complete problem. It is closely related to constraint satisfaction problems with finite domains and tabular constraints [32], conjunctive query containment [19] and homomorphism of relational structures. The formulation of θ-subsumption as a constraint satisfaction problem has been exploited in the ILP literature for the development of fast θ-subsumption algorithms [56, 71]. CSP solvers can also be used to check whether two clauses are isomorphic, by using the primal CSP encoding described in [71] together with an *alldifferent* constraint [45] over CSP variables representing logical variables.

To efficiently check if a clause φ has an isomorphic counterpart in a set of clauses Φ, i.e.

$$\exists \varphi' \in \Phi : \ \varphi \approx_{iso} \varphi',$$

we use the above-mentioned CSP encoding accompanied with an optimization based on the Weisfeiler-Lehman (WL) labeling procedure [140]. We use its directed hypergraph variant, where terms play the role of hyper-vertices and literals the role of directed hyper-edges and enrich the respective clauses by unary literals with predicates representing the labels obtained by the Weisfeiler-Lehman procedure, which also helps the CSP solver to reduce its search space. This procedure provides us with a hash that we further utilize to minimize the number of isomorphism calls. Therefore, we represent Φ by a multi-list, i.e., a map where the key is a hash of the WL procedure and the value is a list of sentences. This is, in particular, favorable for the above-mentioned problem since it allows us to test isomorphism only between pairs of sentences that share the same hash. We will employ this approach as a routine for checking whether an isomorphic sentence already exists within a set of sentences for some sentence φ, because this is more efficient than testing every single sentence $\varphi' \in \Phi$ against φ.

3.2 COVERING RELATION $\models$

To evaluate the covering relation of a hypothesis φ on an example e, i.e.

$$e \models \varphi,$$

which is the case in Section 2.3.1, we can use a θ-subsumption solver as follows. Since we restrict ourselves to hypotheses in the form of clausal theories, each hypothesis φ can be written as a conjunction of clauses

$$\varphi = \alpha_1 \wedge \cdots \wedge \alpha_n.$$

Clearly, $e \not\models \varphi$ if there is an i in $\{1, \dots, n\}$ such that

$$e \models \neg\alpha_i,$$

which holds precisely when

$$\alpha_i \preceq_\theta \neg(\bigwedge e).$$

This means that we can exploit fast θ-subsumption algorithms for checking the covering relation in this setting.

Example 15. *Let us consider the following example, inspired by the Michalski's East-West trains datasets [120]:*

$$\begin{aligned} e = \{&\mathsf{EastBound}(\mathit{car1}), \mathsf{HasCar}(\mathit{car1}), \mathsf{HasLoad}(\mathit{car1}, \mathit{load1}),\\ &\mathsf{BoxShape}(\mathit{load1}), \neg\mathsf{EastBound}(\mathit{load1}), \neg\mathsf{HasCar}(\mathit{load1}),\\ &\neg\mathsf{HasLoad}(\mathit{load1}, \mathit{car1}), \neg\mathsf{HasLoad}(\mathit{load1}, \mathit{load1}),\\ &\neg\mathsf{HasLoad}(\mathit{car1}, \mathit{car1}), \neg\mathsf{BoxShape}(\mathit{car1})\}\end{aligned}$$

and two hypotheses φ_1 and φ_2

$$\begin{aligned} \varphi_1 &= \forall x \forall y\ \mathsf{HasLoad}(x, y) \wedge \mathsf{BoxShape}(y) \Rightarrow \mathsf{EastBound}(x),\\ \varphi_2 &= \forall x \forall y\ \neg\mathsf{EastBound}(x) \vee \neg\mathsf{HasLoad}(x, y).\end{aligned}$$

To check if $e \models \varphi_i$, for $i = 1, 2$, using a θ-subsumption solver, we construct

$$\begin{aligned} \neg(\bigwedge e) = &\neg\mathsf{EastBound}(\mathit{car1}) \vee \neg\mathsf{HasCar}(\mathit{car1}) \vee\\ &\vee \neg\mathsf{HasLoad}(\mathit{car1}, \mathit{load1}) \vee \mathsf{BoxShape}(\mathit{load1}) \vee\\ &\vee \mathsf{EastBound}(\mathit{load1}) \vee \mathsf{HasCar}(\mathit{load1}) \vee\\ &\vee \mathsf{HasLoad}(\mathit{load1}, \mathit{car1}) \vee \mathsf{HasLoad}(\mathit{load1}, \mathit{load1}) \vee\\ &\vee \mathsf{HasLoad}(\mathit{car1}, \mathit{car1}) \vee \mathsf{BoxShape}(\mathit{car1})\end{aligned}$$

It is then easy to check that $\varphi_1 \not\preceq_\theta \neg(\bigwedge e)$ and $\varphi_2 \preceq_\theta \neg(\bigwedge e)$, from which it follows that $e \models \varphi_1$ and $e \not\models \varphi_2$.

In practice, when using a θ-subsumption solver to check $\alpha_i \preceq_\theta \neg(\bigwedge e)$, it is usually beneficial to flip the signs of all the literals, i.e., to instead check $\widetilde{\alpha_i} \preceq_\theta \bigvee e$, which is clearly equivalent. This is because θ-subsumption solvers often represent negative literals in interpretations implicitly to avoid excessive memory consumption,[1] relying on the assumption that most predicates in real-life datasets are sparse.

3.3 THEOREM PROVING USING SAT SOLVERS

In this thesis, we require access to an efficient theorem prover for clausal theories. Since we restrict ourselves to function-free theories without equality, we can rely on a simple theorem-proving procedure based on propositionalization, which is a consequence of the following well-known result[2] [87].

1 This is true for the θ-subsumption solver based on [56] which we use in our implementation.
2 The formulation of Hebrand's theorem used here is taken from notes by Cook and Pitassi: http://www.cs.toronto.edu/~toni/Courses/438/Mynotes/page39.pdf.

Theorem 1 (Herbrand's Theorem)**.** *Let $\mathcal{L}$ be a first-order language without equality and with at least one constant symbol, and let $\mathcal{T}$ be a set of clauses. Then $\mathcal{T}$ is unsatisfiable if and only if there exists some finite set $\mathcal{T}_0$ of $\mathcal{L}$-ground instances of clauses from $\mathcal{T}$ that is unsatisfiable.*

Here $\alpha\theta$ is called an $\mathcal{L}$-ground instance of a clause α if θ is a grounding substitution that maps each variable occurring in α to a constant from the language $\mathcal{L}$.

In particular, to decide if $\mathcal{T} \models \alpha$ holds, where $\mathcal{T}$ is a set of clauses and α is a clause (without constants and function symbols), we need to check if $\mathcal{T} \wedge \neg\alpha$ is unsatisfiable. Since Skolemization preserves satisfiability, this is the case if and only if

$$\mathcal{T} \wedge \neg\alpha_{\mathrm{Sk}}$$

is unsatisfiable, where $\neg\alpha_{\mathrm{Sk}}$ is obtained from $\neg\alpha$ using Skolemization. Let us now consider the restriction $\mathcal{L}_{\mathrm{Sk}}$ of the considered first-order language $\mathcal{L}$ to the constants appearing in α_{Sk}, or to some auxiliary constant s_0 if there are no constants in α_{Sk}. From Herbrand's theorem, we know that $\mathcal{T} \wedge \neg\alpha_{\mathrm{Sk}}$ is unsatisfiable in $\mathcal{L}_{\mathrm{Sk}}$ if and only if the grounding of this formula w.r.t. the constants from $\mathcal{L}_{\mathrm{Sk}}$ is satisfiable, which we can efficiently check using a SAT solver. Moreover, it is easy to see that $\mathcal{T} \wedge \neg\alpha_{\mathrm{Sk}}$ is unsatisfiable in $\mathcal{L}_{\mathrm{Sk}}$ if and only if this formula is unsatisfiable in $\mathcal{L}$. Indeed, if $\mathcal{T} \wedge \neg\alpha_{\mathrm{Sk}}$ is unsatisfiable in $\mathcal{L}$, then there is a set of corresponding $\mathcal{L}$-ground instances of clauses that are unsatisfiable. If we replace each constant appearing in these ground clauses which does not appear in α_{sk} by an arbitrary constant that does appear in α_{sk}, then the resulting set of ground clauses must still be inconsistent, since $\mathcal{T}$ does not contain any constants and there is no equality in the language, meaning that $\mathcal{T} \wedge \neg\alpha_{\mathrm{Sk}}$ cannot be satisfiable in $\mathcal{L}_{\mathrm{Sk}}$.

In practice, it is not always necessary to completely ground the formula $\mathcal{T} \wedge \neg\alpha_{\mathrm{Sk}}$. It is often beneficial to use an incremental grounding strategy similar to cutting plane inference in Markov logic [112]. To check if a clausal theory $\mathcal{T}$ is satisfiable, this method proceeds as follows.

STEP 0: start with an empty Herbrand interpretation $\mathcal{H}$ and an empty set of ground formulae $\mathcal{G}$.

STEP 1: check which groundings of the formulae in $\mathcal{T}$ are not satisfied by $\mathcal{H}$ (e.g. using a CSP solver). If there are no such groundings, the algorithm returns $\mathcal{H}$, which is a model of $\mathcal{T}$. Otherwise the groundings are added to $\mathcal{G}$.

STEP 2: use a SAT solver to find a model of $\mathcal{G}$. If $\mathcal{G}$ does not have any model then $\mathcal{T}$ is unsatisfiable and the method finishes. Otherwise replace $\mathcal{H}$ by this model and go back to Step 1.

Part III

PROPOSED METHODS

4 DISCOVERING INTEGER SEQUENCES VIA FIRST-ORDER LOGIC

Every hard problem in mathematics has something to do with combinatorics.

— Lennart Carleson [97]

Integer sequences are omnipresent in mathematics. A wide range of questions can be asked when a research is investigating one. The most fundamental question is

What is the next element of the sequence?

among others, followed by

Is there a closed formula to express the sequence?
Has the sequence usage in any area?

etc. A simple look-up database of integer sequences enhanced with a description, formulae, properties, usages, and other features would help researchers in such situations.

A vital representative of such a database is the *On-Line Encyclopedia of Integer Sequences* (OEIS) [91], which has been in construction for more than the last 50 years. OEIS has grown to more than 367 thousand of entries,[1] where a single entry is often accompanied by different formulae, a list of properties, and notes about occurrences of that particular sequence in different fields. However, the content of this database is a clear reflection of what researchers deemed to be interesting over the decades, which is of course a subjective thing and may even shift during decades.

Many quite natural sequences are not contained in OEIS. For instance, as observed by [8], it contains sequences counting 2-regular graphs properly colored by 2 colors, but not 2-regular graphs properly colored by 3 colors. There are many similar examples of sequences missing from OEIS, which might be potentially useful for some users. This is also the motivation for the work presented in this chapter, which is a method that constructs a database of combinatorial sequences, i.e., sequences that count named objects of size n that have some given property, which are the subject of interest of enumerative combinatorics [119]. Examples of such combinatorial sequences include sequences counting: subsets of an n-element set, graphs on n vertices, connected graphs on n vertices, trees on n vertices, permutations on n elements without fixpoints, etc. In particular, we focus on combinatorial sequences of structures *that can be described using a first-order logic sentence*.

The following section outlines the construction of the database. The main focus of this chapter – an effective generation of first-order sentences – one of two cornerstones, which is needed to create the database in a reasonable time, is described in Section 4.2.

1 To this date, January 2024.

4.1 CONSTRUCTING THE SEQUENCE DATABASE

Our database of integer sequences is firmly intertwined with first-order logic formulae since, as we said earlier, we are interested in combinatorial sequences that can be encoded using first-order logic sentences. We propose a two-stage method to construct the database in a fully automatic way. The first stage generates first-order logic sentences, and the second computes an integer sequence for a given sentence.

When two sentences produce the same integer sentence,[2] we call one of them *redundant*. Naturally, we want to keep the database as compact as possible, ideally eliminating all *redundant* sentences. A naive approach would consist of filtering *redundant* sentences when inserting a new entry into the database. We show that there is a set of techniques that is beneficial to utilize during the sentence generation process to construct the database in a non-naive way. Our approach saves computational resources and is able to fill in the database in a reasonable time compared to the naive approach. This sentence generation process is described in Section 4.2, while the rest of this section describes the second phase.

This task can also be formulated as a relational pattern mining problem: *Mine first-order logic formulae that correspond to integer sequences while obtaining as many as possible unique integer sequences in a reasonable amount of time.* However, the comparison to OEIS is a much more natural motivation. This connection of paradigms of integer sequences and first-order logic is usually the subject of (lifted) first-order model counting [103], which we also utilize to achieve our goal.

4.1.1 *Computing the Integer Sequences*

Given a first-order logic sentence, we need to compute a number sequence such that its k-th member is the model count of a relational sentence on the domain of size k. The set of domain sizes for which the sequence member would be non-zero is called a *spectrum* of the sentence. *Spectrum* of a logical sentence φ is the set of natural numbers occurring as the size of some finite model of φ [15]. Since the sequence that we seek builds, in some sense, on top of the spectrum, and since the sequence can also be described as the result of the combinatorial interpretation of the original sentence, we dub the sequence *combinatorial spectrum* of the sentence.

Definition 3 (Combinatorial Spectrum)**.** *Combinatorial spectrum of a logical sentence φ, denoted as $\mathfrak{S}(\varphi)$, is a sequence of model counts of φ on finite domains of sizes taking on values $1, 2, 3, 4, \ldots$*

Example 16. *Consider the sentence*

$$\varphi = \exists x \; H(x).$$

Then, all subsets of HB are a model of φ except for the empty set. Hence, for a domain of size n, there will be $2^n - 1$ models, i.e,

$$\mathfrak{S}(\varphi) = 1, 3, 7, 15, \ldots$$

2 A proper definition follows in the text later.

Defining combinatorial spectra in this way allows us to delegate the second phase, i.e., computing an integer sequence given a first-order logical sentence, to existing methods from lifted first-order model counting [134] if we restrict the language. Model counting of the $\mathbf{FO}^2$ fragment of first-order logic is computable in polynomial time [17].[3] The same holds for any $\mathbf{C}^2$ sentence as discussed in Section 2.2.2. Contrary, this does not hold for $\mathbf{FO}^3$ [9].

To conclude this, we can compute the combinatorial spectrum of any $\mathbf{C}^2$ sentence in polynomial time, which is an important observation that we will use during the construction of our database.

4.2 GENERATING THE FIRST-ORDER LOGIC SENTENCES

There are at least two Scott normal forms for $\mathbf{C}^2$ appearing in the literature [43, 105]. However, these normal forms were designed only to guarantee equisatisfiability of $\mathbf{C}^2$ sentences and their normal forms. Since they do not guarantee *combinatorial equivalence*, they would not be directly useful for us in the sentence generation process. Instead, we are interested in $\mathbf{C}^2$ sentences in the following syntactic form:

$$\bigwedge_{i=1}^{M} Q_{i,1}x\, Q_{i,2}y\ \alpha_i(x,y) \wedge \bigwedge_{i=1}^{M'} Q_i x\ \alpha_i(x) \tag{2}$$

where $Q_i, Q_{i,1}, Q_{i,2} \in \{\forall, \exists, \exists^{=1}, \dots, \exists^{=K}\}$, each $\alpha_i(x,y)$ is a quantifier-free disjunction of literals containing only the logical variables x and y, and, similarly, each $\alpha_i(x)$ is a quantifier-free disjunction of literals containing only the logical variable x. The integers K, M and M' are parameters.

Example 17 (Syntactical sentences)**.** *Consider the following $\mathbf{C}^2$ sentences:*

$$\varphi_1 = (\forall x\ \neg R(x,x)) \wedge (\forall x \forall y\ \neg R(x,y) \vee R(y,x))$$

$$\varphi_2 = (\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x))$$

$$\varphi_3 = (\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall y \exists^{=1} x\ R(x,y)) \wedge (\forall x\ \neg R(x,x))$$

Each of these φ_i is an $\mathbf{C}^2$ of the form Equation (2)*.*

Note that φ_1 encodes the number undirected graphs with n and no loop, i.e.

$$\mathfrak{S}(\varphi_1) = 1, 2, 8, 64, \dots.$$

Sentence φ_2 encodes a permutation, leading its combinatorial spectrum to be values of $n!$, i.e.

$$\mathfrak{S}(\varphi_2) = 1, 2, 6, 24 \dots.$$

Finally, φ_3 encodes a derangement with the combinatorial spectrum

$$\mathfrak{S}(\varphi_3) = 0, 1, 2, 9, \dots.$$

Example 17 shows several sentences of the syntactical form we are interested in. However, these sentences are not taken by random. They do model very well-known combinatorics problems. For instance, recall that derangement can be used to solve the Secret Santa problem [64]. This example was

3 See Section 2.2.1 for details.

aimed to demonstrate the suitability of Equation (2) for our task. The combinatorial spectrum of any sentence of this syntactical form is also computable in polynomial time, which is essential to construct our database in a reasonable time.

However, it is no secret that Equation (2) does not cover all combinatorial spectra of $\mathbf{C}^2$ sentences. In fact, even $\mathbf{FO}^2$ sentences generate infinitely many integer sequences. Take, for instance, the sentences φ_k of the form

$$\varphi_k = \exists x \bigvee_{i=1}^{k} P_i(x).$$

Their combinatorial spectra are

$$\mathfrak{S}(\varphi_k) = \left((2^k)^n - 1\right)_{n=1}^{\infty}.$$

Hence, we have infinitely many combinatorial spectra even for these simple sentences – one for each $k \in \mathbb{N}$. Nevertheless, in our opinion, this hardly matters because our task is not to generate all combinatorial sequences of $\mathbf{C}^2$ sentences – this would not be feasible anyways because the number of different integer sequences generated as spectra of $\mathbf{C}^2$ sentences is infinite.[4] Instead, what we want to achieve is to generate as many *interesting* integer sequences as possible within a limited time budget.

The transformation presented in [54] allows one to reduce the computation of model counts of any $\mathbf{C}^2$ sentence to a computation with sentences that are in the form of Equation (2), it requires some of the predicates to have negative weights. We do not allow negative weights in the generated sentences because they make the post-hoc combinatorial explanation of the sentences significantly more difficult.

Note that in the rest of this chapter, we will slightly abuse terminology and use the term *clause* for the quantified disjunctions of literals in the form $Q_1 x Q_2 y \ \alpha_i(x, y)$ and $Qx \ \alpha_i(x)$. This step is done to provide a compact readability rather than wordy phrase *quantified disjunction of literals* every single time. Since this goes against the standard terminology, we will write quantifiers as much as possible to stress the difference.

4.3 SENTENCE REDUNDANCY

For any integer sequence, there may be many sentences that generate it. The most trivial example, based on semantics, is contradiction because the combinatorial spectrum of contradiction is, by definition, a sequence of zeros. See the following example for a syntax-based counterpart.

Example 18. *Let $\mathcal{P}$ be a set of n unary predicates, i.e.*

$$\mathcal{P} = \{P_1/1, P_2/1, P_3/1, \dots, P_n/1\}.$$

Then, we have at least n formulae corresponding to combinatorial spectra of the form $2^n - 1$, i.e.

$$\mathfrak{S}(\exists x \ P_i(x)) = 1, 3, 7, 15, \dots$$

4 This is a distinction from standard pattern mining task in which one usually wants to enumerate all patterns within some language restrictions with non-zero occurrence in the data, i.e., support, which are usually finite. On contrary, in our task any $\mathbf{C}^2$ sentence corresponds to some integer sequence, e.g., a trivial one as $0, 0, 0, 0, \dots$.

for $1 \leqslant i \leqslant n$.[5] *Thus, we can construct many first-order logic formulae of the same spectrum when the language contains multiple predicates of the same arity.*

Note that there may be more than n *formulae corresponding to this particular combinatorial spectrum – not all of them have the said syntactical version, e.g.,* $(\exists x\, P_1(x)) \wedge (\forall x\, P_1(x) \vee P_2(x)) \wedge (\forall x\, \neg P_1(x) \vee P_2(x))$.

This poses a problem in the database construction as the database could be overhauled by *redundant* sentences. Therefore, we aim to avoid generating *redundant* sentences.

Definition 4. *Given a collection of sentences* Φ*, a sentence* $\varphi \in \Phi$ *is considered* redundant *if there is another sentence* $\varphi' \in \Phi$ *and* $\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi')$.

Further, a redundant sentence may be either *safe-to-delete* or *not-safe-to-delete*. These two subcategories serve different purposes. The first, *safe-to-delete*, contains sentences that can be pruned from the sentence space without destroying its completeness. On the contrary, sentences in the second category, *not-safe-to-delete*, cannot be erased from the sentence space, however, we do not have to compute their spectra.[6] Whether a redundant sentence φ is *safe-to-delete* or *not-safe-to-delete* depends on the search strategy used. Although the latter category does not narrow the sentence space, it can still lead to a tremendous saving of computational resources since it allows to skip computation of some combinatorial spectra.

The problem of deciding whether two sentences have the same combinatorial spectrum is no easier than checking whether they are equivalent, which can be seen as follows. Let one of the sentences be a contradiction. Checking whether the other sentence has the same combinatorial spectrum, i.e., $0, 0, 0, 0, \ldots$, is equivalent to checking whether it is also a contradiction. This is only a complexity lower bound, but it already shows that checking the equivalence of combinatorial spectra of $\mathbf{C}^2$ sentences is NEXPTIME-hard, which follows from the classical results on the complexity of satisfiability checking in $\mathbf{C}^2$ [104]. [7] Therefore, we will only search for sufficient conditions for when two sentences generate the same spectrum.

4.4 TRAVERSING THE SENTENCE SPACE

Our sentences-generation method works in an iterative-deepening mode. It starts generating all sentences of a language $\mathcal{L}_k$. When all possible sentences of $\mathcal{L}_k$ are generated, it continues with $\mathcal{L}_{k+1}$. A language $\mathcal{L}_k$ is an instantiation of an extension of Equation (2) as follows:

$$\bigwedge_{i=1}^{M} Q_{i,1}x\; Q_{i,2}y\; \alpha_i^J(x,y) \wedge \bigwedge_{i=1}^{M'} Q_i x\; \alpha_i^J(x) \tag{3}$$

5 Note that for a sentence φ we compute the combinatorial spectrum w.r.t. predicates occurring in φ.

6 In other words, sentences from this category are not *evaluated*. Whereas in standard rule-learning *evaluating* would mean computing *support*, *accuracy*, etc., it translates to computing combinatorial spectra in our task.

7 The exact complexity of deciding whether two $\mathbf{C}^2$ sentences generate the same combinatorial spectra remains an interesting open problem.

where $Q_{i,1}, Q_{i,2}, Q_i$ are quantifiers from the set of quantifiers $\mathcal{K}$. A quantifiers-free disjunction of literals, i.e., $\alpha_i^J(x,y)$ and $\alpha_i^J(x)$, contains at most J literals. There are at most M' and M disjunctions with one and two variables, respectively. Finally, literals in disjunctions contain only predicates from $\mathcal{P}$. Iterative-deepening is then run over all of these parameters, including various number of unary and binary predicates in $\mathcal{P}$. This mimics enumerating all sentences that are *easier to encode* in Occam's razor style – using fewer literals, predicates, and quantifiers sooner, followed by longer sentences with an increasing number of predicates later in the generation process. Specifically, the core algorithm running inside the iterative-deepening one has to solve the following problem:

Given: A set of predicates $\mathcal{P}$, set of quantifiers $\mathcal{K}$, and integers M, M', J.

Generate: The set of sentences Φ of the form Equation (3) while Φ having as few redundant sentences as possible.

The transition from Equation (2) to Equation (3) is a technicality that restricts the language we are working with since the former unrestricted syntactical form would lead to infinite set of *basic* clauses $\mathcal{A}$, hence not deriving a single sentence at all.[8]

4.4.1 *Description of the Algorithm*

From the high-level perspective, the algorithm may be described as follows: firstly, the algorithm generates all disjunctions of literals with at most J literals and predicates from $\mathcal{P}$, combining these with all available quantifiers. Secondly, the algorithm gradually constructs literal-wise longer and longer sentences by extending shorter sentences it generated before with all available clauses from the first step, starting with an empty clause as the very first sentence.

Algorithm 1 describes the aforementioned approach in detail. The algorithm starts by generating all possible clauses that can be constructed w.r.t. the parameters; this is wrapped inside the function ***allClauses*** on line 1. It is a straightforward process that we leave out intentionally for ease of presentation. Then, *safe-to-delete* clauses are removed from $\mathcal{A}$ (line 2). The initialization ends with assigning an empty sentence to Φ (line 3). It is important to distinguish sets $\mathcal{A}$ and Φ, with the former containing clauses, i.e., basic building blocks, and the latter containing sentences, i.e., the final product that is, in turn, used to construct longer sentences.

At the end of the i-th cycle of the main for loop (lines 4-16), Φ contains all possible sentences from the language with at most i literals. This is done in the following two steps. In the first step, the algorithm extends Φ with all clauses with exactly i literals from $\mathcal{A}$ (line 5). In the second step, it selects all sentences φ with exactly l literals and refines them with all clauses α such that the final refinement, i.e., $\varphi \wedge \alpha$, has exactly i literals (lines 7-14). Unless the refinement is *safe-to-delete*, it is added into Φ (line 11). Finally, *redundant* sentences are not returned by the algorithm (line 17). Note that

8 Other approaches may overcome this issue but will most likely suffer from unrestricted $\mathcal{P}$ as well.

only *not-safe-to-delete* filtering is used as all *safe-to-delete* sentences[9] were already deleted through execution of the algorithm.

Algorithm 1 Pseudocode of Clause-wise Sentence Generator

Parameter: Set of predicates $\mathcal{P}$, set of quantifiers $\mathcal{K}$, and integers M, M', J.
Output: Set of (non-redundant) sentences with at most $(M + M') \cdot J$ literals.

```
1: $\mathcal{A} \leftarrow$ ***allClauses***$(M, M', K, J, \mathcal{P})$
2: $\mathcal{A} \leftarrow \{\alpha \in \mathcal{A} \mid \alpha$ *is not redundant*$\}$
3: $\Phi \leftarrow \{\emptyset\}$
4: **for** $i \in [1, 2, 3, \dots, (M + M') \cdot J]$ **do**
5:   $\Phi \leftarrow \Phi \cup \{\alpha \in \mathcal{A} \bigm| |\alpha| = i\}$
6:   **for** $l \in [1, 2, 3, \dots, i - 1]$ **do**
7:     **for** $\varphi \in \{\varphi \in \Phi \bigm| |\varphi| = i - l\}$ **do**
8:       **for** $\alpha \in \{\alpha \in \mathcal{A} \bigm| |\alpha \wedge \varphi| = i\}$ **do**
9:         $\varphi' \leftarrow \varphi \wedge \alpha$
10:        **if** $\varphi'$ *is not safe-to-delete* **then**
11:          $\Phi \leftarrow \Phi \cup \{\varphi'\}$
12:        **end if**
13:      **end for**
14:    **end for**
15:  **end for**
16: **end for**
17: **return** $\{\varphi \in \Phi \mid \varphi$ *is not redundant*$\}$
```

One could argue that redundancy of a sentence was defined w.r.t. a set of sentences. Indeed, however, testing redundancy w.r.t. Φ, i.e., all generated sentences, would be inefficient for most methods. In general, we can say that the *not-safe-to-delete* filtering on line 17 is done w.r.t. Φ, while *safe-to-delete* on line 10 is done w.r.t. sentences generated at i-the iteration of the main loop. Finally, there are methods that assume the witness was generated earlier, i.e., is shorter, and thus check each sentence in isolation; these can appear on lines 2, 10 and 17.

The vanilla algorithm, i.e., with no redundancy-checking technique, enumerates all sentences w.r.t. some language restriction, which brings us to the question why we have not opted for an optimal refinement operator, but used an ideal one instead. From the discussion on sentences' spectra equivalence in Section 4.3 it follows that the an optimal refinement operator would be costly.

Next, we describe an illustrating example. After that, we give a sketch of correctness.

4.4.1.1 *An Illustration*

Let us now proceed with an example with no redundancy checking technique and with a very restricted language to keep the example reasonable small.

9 With respect to used redundancy-checking techniques.

Assume $\mathcal{P} = \{P/1\}$, $\mathcal{K} = \{\exists\}$, $M = 2$, $M' = 2$, $J = 2$. First, we have to generate all clauses satisfying this setup,[10] i.e.

$$\begin{aligned}\mathcal{A} = \{&\alpha_1 = \exists x\ P(x),\\ &\alpha_2 = \exists x\ \neg P(x),\\ &\alpha_3 = \exists x \exists y\ P(x) \vee P(y),\\ &\alpha_4 = \exists x \exists y\ P(x) \vee \neg P(y),\\ &\alpha_5 = \exists x \exists y\ \neg P(x) \vee P(y),\\ &\alpha_6 = \exists x \exists y\ \neg P(x) \vee \neg P(y)\}.\end{aligned}$$

After the initiation, Φ will contain only an empty sentence,[11] i.e.

$$\Phi_0 = \{\emptyset\}$$

After the first iteration of the outer for-loop starting at line 4, Φ will newly contain sentences with exactly one literal, i.e.

$$\Phi_1 = \{\alpha_1, \alpha_2\}$$

After the second iteration, the set of sentences will be extended with sentences with exactly two literals, i.e.

$$\Phi_2 = \{\alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_1 \wedge \alpha_2\}$$

Note that there are only three unique spectra among these sentences, i.e.

$$\begin{aligned}\mathfrak{S}(\alpha_1) = \mathfrak{S}(\alpha_2) = \mathfrak{S}(\alpha_3) = \mathfrak{S}(\alpha_6) &= 1, 3, 7, 15 \ldots\\ \mathfrak{S}(\alpha_4) = \mathfrak{S}(\alpha_5) &= 2, 4, 8, 16, \ldots\\ \mathfrak{S}(\alpha_1 \wedge \alpha_2) &= 0, 2, 6, 14, \ldots\end{aligned}$$

Even though the algorithm started with six basic building blocks and finished with seven sentences, there are only three unique spectra.[12] Since computing a combinatorial spectrum is a resource-demanding operation, invoking this computation as little as possible would be beneficial. In addition, pruning the sentence space would be even more beneficial since it would lower the number of generated sentences. Lines 2, 10, and 17 play a crucial role for handling both of these issues.

4.4.1.2 *Sketch of Correctness*

Firstly, we will discuss the algorithm in its vanilla setting with no redundancy checking technique applied during the generation process. Let $\mathcal{A} = \{\alpha_1, \ldots, \alpha_l\}$ be the set of all possible clauses given $\mathcal{K}$, $\mathcal{P}$, M, M', and J. In particular, soundness, follows from the refinement step – each sentence φ,

10 While generating other isomorphic clauses is possible, e.g., $\alpha_1' = \exists y\ P(y)$, it would be inefficient since refining a sentence φ with a variable-isomorphic clause would lead to isomorphic clauses, i.e. $\varphi \wedge \alpha_1 \approx_{iso} \varphi \wedge \alpha_1'$.

11 For which we do not compute its combinatorial spectrum since it does not contain any predicate.

12 This is not surprising since we used the algorithm with its vanilla setup with no redundancy-checking method.

that is added into the set Φ, consists of clauses in $\mathcal{A}$, i.e., $\varphi = \alpha_1 \wedge \cdots \wedge \alpha_n$. To justify completeness, let Φ_i be the set of sentences generated after the i-th iteration of the main for loop. Φ_i contains all sentences with at most i literals w.r.t. clauses in $\mathcal{A}$, which can be checked by inspection of the algorithm.

The soundness of the algorithm is also easy to show in the case of sound *not-safe-to-delete* techniques, which follows from the fact that these techniques do not prune the sentence space. Their application resembles post-processing filtering, which forbids adding sentences that produce the same combinatorial spectrum and.

In general, *safe-to-delete* techniques detect a class of sentences Φ' which are redundant, i.e., all sentences from the class produce the same combinatorial spectrum but not vice versa. After the detection, a single sentence from this class, a witness, is left in the sentence space while the rest is removed. Rather than making a general proof showing that refinements of every sentence in Φ' lead to the same set of combinatorial spectra, we discuss the correctness of each redundancy-checking method in isolation.

4.4.2 *Sentence Redundancy Techniques*

In this section, we discuss several techniques that decide whether a sentence is redundant or not. We stress upfront that the methods presented here will not guarantee detecting all redundancies. On the other hand, these methods will be sound – they will not mark non-redundant sentences as redundant. Some of the techniques will mark a sentence as redundant, but they will not give us a witness for the redundancy, i.e., other sentences with the same combinatorial spectrum. This will be the case for techniques that guarantee that the witness is a shorter sentence (in the number of literals), which must have been generated earlier; thus, we will know that by pruning the longer redundant sentences, we will not affect the completeness of the search. We will use the terms *safe-to-delete* and *not-safe-to-delete* for these methods in the same way we use them for sentences. Sometimes, we call these techniques *pruning* since it is a standard term in symbolic search.

4.4.2.1 *Isomorphic Sentences*

Earlier, we said that the problem we are solving in this chapter resembles a pattern-mining scenario. Indeed, the first redundancy-checking technique is an extension of a usual component for the enumeration of non-isomorphic patterns, e.g., used in [88], which works by extending the definition of sentences isomorphism. Let us formally define *variable*-isomorphism of two clauses:

Definition 5 (Isomorphism of clause)**.** *We say that the clause* $Q_1 x Q_2 y \ \varphi(x,y)$ *is isomorphic to the clause* $Q_1' x Q_2' y \ \psi(x,y)$ *if one of the following conditions holds:*

1. *If* $Q_1 = Q_1' = Q_2 = Q_2'$ *and there exists a bijection* $f : \{x,y\} \to \{x,y\}$ *such that the set of literals of* $f(\varphi(x,y))$ *is the same as the set of literals of* $\psi(x,y)$.
2. *If* $Q_1 = Q_1' \neq Q_2 = Q_2'$ *and the set of literals of* $\varphi(x,y)$ *is the same as the set of literals of* $\psi(x,y)$.

Naturally, we firstly need to extend this definition to sentences.

Definition 6 (Variable-isomorphism of sentences)**.** *We say that two sentences of the form Equation* (2) *are isomorphic if, for every clause from one sentence, we can find a clause from the other sentence which it is isomorphic to.*

Finally, we extend this to our case where isomorphism of sentences is checked with respect to predicate names and variables.

Definition 7 (Variable & predicate-isomorphism of sentences)**.** *We say that two sentences of the form Equation* (2) *are isomorphic by renaming of predicates if there exists a bijection between their predicates that makes the two sentences isomorphic according to the definition of isomorphism Definition 6.*

Definition 7 can be utilized to induce classes of isomorphism over sentences. It is easy to see that given a class of isomorphic sentences Φ, each sentence $\varphi \in \Phi$ produces the same combinatorial spectrum; this holds since we can consistently rename predicates and variables in one sentence to get another in the same class.

This method is *safe-to-delete* and thus can be used to prune the sentence space. The reasoning behind this follows the usage of enumerating non-isomorphic patterns in [88]. Whereas there, the meaning and a support of pattern is the same no matter the legal renaming of variables, here the combinatorial spectrum does not rely on particular names of predicates. A more rigorous justification, for instance, would define a refinement downward operator ($\rho_{\mathcal{A}}$) and prove that the sets of refinements for two isomorphic sentences φ_1 and φ_2 are isomorphic, i.e., $\rho_{\mathcal{A}}(\varphi_1)$ and $\rho_{\mathcal{A}}(\varphi_2)$ are isomorphic.

Example 19. *Having* $M = 0$, $M' = 1$, $\mathcal{P} = \{P_0/1, P_1/1\}$, *and the set quantifiers* $\mathcal{K} = \{\forall, \exists\}$, *the algorithm's content of* Φ *after the first iteration is*[13]

$$\{\exists x\ P_0(x), \exists x\ \neg P_0(x), \forall x\ P_0(x), \forall x\ \neg P_0(x)\}.$$

The vanilla algorithm would have Φ *twice that size.*

After computation of combinatorial spectra of sentences in Φ, *we obtain two unique sequences, i.e.*

$$\mathfrak{S}(\exists x\ P_0(x)) = \mathfrak{S}(\exists x\ \neg P_0(x)) = 1, 3, 7, 15, \ldots$$

$$\mathfrak{S}(\forall x\ P_0(x)) = \mathfrak{S}(\forall x\ \neg P_0(x)) = 1, 1, 1, 1, \ldots$$

Note that to effectively check the isomorphism of two sentences,[14] we can deploy the hash-based approach discussed in Section 3.1. However, in practice, the checking of these extended classes of isomorphism[15] can be solved more efficiently using a *canonical*-based description of a sentence, which is discussed in Appendix A.1.1.

13 We denote each class of isomorphism by the corresponding lexicographically smallest member.

14 And isomorphism-based methods that are yet to come.

15 The methods that follow – *Negations* and *Permuting arguments*.

4.4.2.2 *Negations*

From the Example 19 we can observe that negation signs can sometimes be flipped. Indeed, this holds when we flip the signs consistently – if we flip negation signs for all occurrences of all literals of some predicate in a sentence, the combinatorial spectrum stays the same. The rationale behind this is the *signs flip* can be seen as a predicate renaming; in the end, it is up to us how we interpret a predicate, e.g., whether $\mathsf{P}(c)$ says that c is *red* or *black* node of a red-black tree. Therefore, we extend Definition 7 with negation signs.

Definition 8 (Negation signs-isomorphism)**.** *We say that two sentences of the form Equation* (2) *are isomorphic if one can be obtained from the other by negating all occurrences of literals of some predicates.*

This definition induces yet another class of equivalences that can be again utilized to prune the sentence space. The justification is analogical to the one in Section 4.4.2.1. Thus, we mark this method as *safe-to-delete*.

Example 20. *Continuing with the setup from Example 19, i.e.,* $M = 0$, $M' = 1$, $\mathcal{P} = \{P_0/1, P_1/1\}$, *and the set quantifiers* $\mathcal{K} = \{\forall, \exists\}$, *the algorithm's content of* Φ *after the first iteration is*

$$\{\exists x\ P_0(x), \forall x\ P_0(x)\},$$

further halving the size of Φ.

Example 21. *Consider the following sentences:*

$$\varphi_1 = (\forall x \forall y\ R(x,y) \vee P(x)) \wedge (\exists x\ P(x)),$$
$$\varphi_2 = (\forall x \forall y\ \neg R(x,y) \vee \neg P(x)) \wedge (\exists x\ \neg P(x)),$$
$$\varphi_3 = (\forall x \forall y\ \neg R(x,y) \vee P(x)) \wedge (\exists x\ \neg P(x)).$$

Then, it is easy to see that φ_1 *and* φ_2 *are* negation signs-*isomorphic while* φ_3 *is not isomorphic to neither of them.*

4.4.2.3 *Permuting Arguments*

In the previous method, we claimed that the interpretation of a predicate is solely in our hands. Here, we apply this argument to the interpretation of binary relations. Suppose we have two sentences α and β as follows:

$$\alpha = \forall x \exists y\ R(x,y),$$
$$\beta = \forall x \exists y\ R(y,x).$$

α can be interpreted as modelling directed graphs in which no vertex has out-degree 0 and β as modelling directed graphs in which no vertex has in-degree 0. This interpretation was based on our decision to interpret $R(x,y)$ as an edge from x to y, yet we could have also interpreted it as an edge from y to x. However, a change in the interpretation does not change the combinatorial spectrum of the sentence, which, again, does not depend on how we interpret the sentence. Therefore, we extend the previous definition of isomorphism with arguments flip.

Definition 9 (Arguments flip-isomorphism)**.** *We say that two sentences of the form Equation* (2) *are isomorphic if we can obtain one from the other by flipping arguments all literals of some binary predicates.*

As in the two previous cases, this technique induces an equivalence class that can be utilized to prune the sentence space. Hence, it is *safe-to-delete*.

4.4.2.4 *Tautologies and Contradictions*

Any sentence containing a tautological clause is redundant because a shorter sentence with the same combinatorial spectrum always exists. Consider a sentence φ that contains a tautological clause, i.e.

$$\varphi = \varphi' \wedge \top,$$

then

$$\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi')$$

since

$$\varphi \models \varphi'.$$

We can get rid of tautological quantified clauses by filtering them out from the set of refinements $\mathcal{A}$, i.e. at line 2 of Algorithm 1.

Similarly, every contradiction is redundant because every refinement leads to a contradiction with the combinatorial spectrum of all zeros. Therefore, once a contradictory sentence is generated in the algorithm, it is not added to the set of generated sentences Φ. This removes all sequences of the form $0, 0, 0, 0, \ldots$ from the generation process, which is not a problem, as this particular integer sequence is not interesting.

Both of these approaches are *safe-to-delete*.

4.4.2.5 *Trivial Constraints*

Contrary to the previous method, this one uses a deduction in a relaxed manner. Firstly, let us denote *trivial constraint* as a universally quantified single-literal clause that has only one model; for example, $\forall x\ P(x)$ and $\forall x \forall y\ R(x, y)$. Then, every sentence containing a trivial constraint is redundant since we can replace all occurrences of all literals, either P or R depending on the trivial constraint, by $\top$, thus obtaining a shorter sentence with the same combinatorial spectrum.

This method is *safe-to-delete*. Hence, the algorithm will not refine a sentence containing a trivial constraint nor add one to the set of generated sentences Φ.

Example 22. *Let* $\varphi = (\exists x\ \neg P_0(x) \vee P_1(x)) \wedge (\forall x\ P_0(x))$. *The latter clause is a trivial constraint, thus we replace all occurrences of literal* $P_0/1$ *with* $\top$, *obtaining*

$$\varphi' = (\exists x \perp \vee P_1(x)) \wedge (\forall x\ \top) = \exists x\ P_1(x).$$

4.4.2.6 *Subsumption and Entailment*

In the previous method, we exploited a simplified version of deduction. Hence, a natural question arises – can we inject more complex reasoning into the redundancy-checking process? For this, we employ several well-known techniques from logic. Namely, we utilize θ-subsumption, entailment, and their relationship, i.e., entailment can be tested using the θ-subsumption for universally quantified clauses, that is

$$(\alpha \preceq_\theta \beta) \implies (\alpha \models \beta). \tag{4}$$

Further, let α and β be two clauses. If α and β are logically equivalent, then their combinatorial spectra are the same, i.e.

$$(\alpha \models\mid \beta) \implies (\mathfrak{S}(\alpha) = \mathfrak{S}(\beta)) \tag{5}$$

which follows from the entailment relation.

θ-reducibility: Let α be non-θ-reducible universally quantified clause and β be θ-reducible universally quantified clause. Then β is redundant if they both are θ-equivalent since, in that case, they produce the same spectrum. See, from

$$\alpha \approx_\theta \beta$$

we derive

$$\mathfrak{S}(\alpha) = \mathfrak{S}(\beta)$$

using Equation (4) and Equation (5). Thus, the technique is *safe-to-delete* and we filter out θ-reducible clauses from the basic ones in $\mathcal{A}$ on line 2 of Algorithm 1.

Example 23. *Consider the following universally quantified clause α and β:*

$$\alpha = \forall x\ R(x,x),$$
$$\beta = \forall x \forall y\ R(x,y) \lor R(x,x).$$

It is sufficient to drop β and keep only α for refinements.

Usually, θ-reducibility is defined over universally quantified clauses, which, unfortunately, cover only a subset of our sentences. Therefore, we extend this idea to $\mathbf{FO}^2$ utilizing the specialized Skolemization described in Section 2.2.1.1. That is, a *clause* with an existential quantifier φ is firstly skolemized, which yields a set of universally quantified clauses Φ'. We say that φ is θ-reducible if there is a θ-reducible clause φ' in Φ', or if there are two sentences φ'_1 and φ'_2 in Φ' such that $\varphi'_1 \preceq_\theta \varphi'_2$. The justification and application follow the standard case of θ-reducibility.

Example 24. *Let $\varphi = \forall x \exists y\ R(x,x) \lor R(x,y)$. Applying the specialized Skolemization from Section 2.2.1.1 we obtain*

$$\Phi' = \{\forall x\ \neg R(x,x) \lor Sk(x),$$
$$\forall x \forall y\ \neg R(x,y) \lor Sk(x)\}$$

with weights $w(Sk) = 1$ and $\overline{w}(Sk) = -1$. It is easy to see that the second sentence in Φ' θ-subsumes the first one with substitution

$$\theta = \{x \mapsto x, y \mapsto x\}.$$

Inter-clause subsumption: While the standard θ-reducibility case can be seen as an intra-clause application of θ-subsumption, the extension to existentially quantified case showed an inter-clause application. Now, we discuss reasoning along this line of application. Suppose we have a sentence that contains two clauses, i.e.

$$\varphi = Q_1 x Q_2 y\ \Psi(x, y) \wedge Q_1 x Q_2 y\ \Gamma(x, y).$$

If $\Psi \preceq_\theta \Gamma$ with $\theta = \{x \mapsto x,\ y \mapsto y\}$ then φ is redundant since $\mathfrak{S}(\varphi)$ is generated by a shorter sentence, namely $\varphi' = Q_1 x Q_2 y\ \Psi(x, y)$. Again, a proof follows from Equations (4) and (5). Thus, the algorithm marks as *safe-to-delete* all sentences that contain two quantified clauses where one θ-subsumes the other.

Example 25. *Consider following clauses*

$$\alpha = \forall x \forall y\ R_1(x, y) \vee R_2(y, y),$$
$$\beta = \forall x \forall y\ R_1(x, y).$$

Then, the sentence $\alpha \wedge \beta$ *is redundant.*

Entailment: Recall that satisfiability checking in $\mathbf{C}^2$ is NEXPTIME-hard, so decidable but time-consuming in practice. Our goal is to fill up a database with sentence-spectrum pairs in a limited time. Therefore, we prefer to relax reasoning to cases where redundancy-checking can be done quickly instead of applying full logical reasoning. The core idea is to relax entailment using θ-subsumption given variations of quantifiers. For example, let

$$\alpha = \forall x\ \Psi(x),$$
$$\beta = \exists x\ \Psi(x),$$

then sentence containing $\alpha \wedge \beta$ is redundant since

$$\alpha \models \beta$$

which we can check using θ-subsumption and a condition on quantifiers. A dozen of similar combinations, for instance

$$\alpha = \forall x \forall y\ \Psi(x, y),$$
$$\beta = \forall x \exists y\ \Psi(x, y)$$

can be derived using the same reasoning. Although this may seem like a *naive* or *hardcoded* simplification, it allows us to scale the sentence-generation process while preserving some limited reasoning.

Similarly, we may remove some $\mathbf{C}^2$ sentences with a relaxed deduction reasoning. For example, $\exists x\ \Psi(x) \wedge \exists^{=1} x\ \Psi(x)$ is redundant since $\exists^{=1} x\ \Psi(x) \models \exists x\ \Psi(x)$. However, it is important to stress that when computing a combinatorial spectrum, we leave out the empty universe, i.e. the zero-sized entity case. Another example would be $\exists^{=1} x\ \Psi(x) \wedge \exists^{=2} x\ \Psi(x)$ which is a contradiction.

We mark all of these three *safe-to-delete* techniques as θ* because it is not a plain θ-subsumption as one might think, nor full logical reasoning.

4.4.2.7 *Cell Graph Isomorphism*

Another redundancy-checking method is built on a concept from the area of lifted inference, originally intended for a more efficient evaluation of Equation (1). In [17], the authors introduced a special structure called a *cell graph* to help them compute WFOMC faster.

Definition 10 (Cell Graph)**.** *A cell graph* G_φ *of a sentence* φ *is a complete graph* (V, E) *such that*

1. V *is the set of cell labels* $\{ 1, 2, \dots, p \}$,
2. *each node* $i \in V$ *has a label* w_i,
3. *each edge, including loops, from node* i *to* j *has a label* r_{ij}.

As one can observe from Equation (1), the WFOMC computation is fully determined by the terms r_{ij} and w_k. That remains unchanged even with the counting quantifiers, since then, only the symbolic result of Equation (1) is further searched for particular monomials. Hence, the computation is fully determined by a cell graph, which contains all the r_{ij} and w_k values.

Building on that observation, we propose a pruning technique based on two cell graphs being isomorphic. If cell graphs of two sentences are isomorphic, then their WFOMC results will be the same, and consequently, their combinatorial spectra will be the same as well. We formalize those claims below.

We start by discussing the simpler case where all weights are real-valued. That is enough to apply this pruning method to $\mathbf{FO}^2$ sentences, and then we extend it to the case with symbolic weights, which is needed for correct handling of sentences from $\mathbf{C}^2$.

First we define what we mean by cell graph isomorphism.

Definition 11 (Cell Graph Isomorphism, for graphs with real-valued weights)**.** *Let* G *and* G' *be two cell graphs where each edge* $\{i, j\} \in E(G)$ *(*$\{i', j'\} \in E(G')$*, respectively) is labeled by a real-valued weight* r_{ij} *(*$r'_{i'j'}$*, respectively) and each vertex* $i \in V(G)$ *(*$i' \in V(G')$*) is labeled by a real number* w_i *(*$w'_{i'}$*). We say that* G *and* G' *are isomorphic if there exists a bijection* $f : V(G) \to V(G')$ *such that* $w'_i = w_{f(i)}$ *and* $r'_{ij} = r_{f(i),f(j)}$ *for all* $i, j \in V(G)$.

In order to exploit the isomorphism, we will exploit the following interesting property of Equation (1).

Remark 1. *Let* p *be again the number of cells in Equation* (1) *and let*

$$f : [p] \to [p]$$

be a bijection. Then the following equality holds:

$$\sum_{\boldsymbol{k} \in \mathbb{N}^p : |\boldsymbol{k}| = n} \binom{n}{\boldsymbol{k}} \prod_{i,j \in [p] : i < j} r_{ij}^{(\boldsymbol{k})_i (\boldsymbol{k})_j} \prod_{i \in [p]} r_{ii}^{\binom{(\boldsymbol{k})_i}{2}} w_i^{(\boldsymbol{k})_i} =$$
$$\sum_{\boldsymbol{k} \in \mathbb{N}^p : |\boldsymbol{k}| = n} \binom{n}{\boldsymbol{k}} \prod_{i,j \in [p] : i < j} r_{f(i),f(j)}^{(\boldsymbol{k})_i (\boldsymbol{k})_j} \prod_{i \in [p]} r_{f(i),f(i)}^{\binom{(\boldsymbol{k})_i}{2}} w_{f(i)}^{(\boldsymbol{k})_i}.$$

In other words, permuting the cells, while preserving the structure of the weights, does not change the resulting value.

Next we state the result which will justify using the cell graph isomorphism method for $\mathbf{FO}^2$ sentences.

Theorem 2. *Let* φ *and* ψ *be two* $\mathbf{FO}^2$ *sentences with weights* $(w, \overline{w})$ *and* $(w', \overline{w}')$*, respectively, and let* G_φ *and* G_ψ *be their respective cell graphs. If* G_φ *is isomophic to* G_ψ*, then*

$$\mathsf{WFOMC}(\varphi, n, w, \overline{w}) = \mathsf{WFOMC}(\psi, n, w, \overline{w})$$

for any domain size $n \in \mathbb{N}$.

Proof sketch. The proof follows from the following observation: Let f be the bijection $f : V(G_\varphi) \to V(G_\psi)$ preserving weights, which must exist from the definition of cell graph isomorphism. Consider the equation for computing WFOMC of a sentence from its cell graph, Equation (1). If we apply the bijection f on the cell indices (i's and j's from the equation), it will turn the equation for computing WFOMC of φ to the one for ψ (again because f is weight-preserving bijection). It follows from Remark 1 that these two must be the same and therefore WFOMC of φ and ψ must be equal for any domain size n. $\square$

Next we extend the cell graph isomorphism method to $\mathbf{C}^2$ sentences. For that, we first need to extend the definition of cell graphs and of cell graph isomorphism.

Definition 12 (Cell Graph with Cardinality Constraints)**.** *A cell graph* G_φ *of a sentence* φ *is a pair* (G, C) *consisting of:*

1. *A complete graph* $G = (V, E)$ *such that*
   a) V *is the set of cell labels* $\{1, 2, \ldots, p\}$,
   b) *each node* $i \in V$ *has a label* w_i,
   c) *each edge, including loops, from node* i *to* j *has a label* r_{ij}.

   Here the weights w_i *and* r_{ij} *are, in general, multivariate polynomials.*
2. *A set* C *of monomials representing the cardinality constraints.*

Example 26. *Consider the sentence*

$$\varphi = (\forall x \, \neg E(x, x)) \wedge (\forall x \forall y \, \neg E(x, y) \vee E(y, x)) \wedge (|E| = 10)$$

which models undirected graphs with 5 edges. There is only one cell which is consistent with φ *for this sentence,* $\neg E(x, x)$*. As we already saw in Example 14, to encode the cardinality constraint* $|E| = 5$*, we need to introduce the symbolic weight* $w(E) = x$*. The cell graph then consists of the graph given by the the weights* $w_1 = 1$, $r_{1,1} = 1 + x^2$*, and of the singleton set* $C = \{x^{10}\}$*, representing the cardinality constraint.*

Now we are ready to state the definition of cell graph isomorphism for $\mathbf{FO}^2$ sentences with cardinality constraints (which is all we need to encode $\mathbf{C}^2$ sentences).

Definition 13 (Cell Graph Isomorphism, for graphs with symbolic weights and cardinality constraints)**.** *Let* (G, C) *and* (G', C') *be two cell graphs with cardinality constraints where each edge* $\{i, j\} \in E(G)$ *(*$\{i', j'\} \in E(G')$*, respectively) is labeled by a multivariate polynomial* r_{ij} *(*$r'_{i'j'}$*, respectively) and each vertex* $i \in V(G)$ *(*$i' \in V(G')$*) is labeled by a multivariate polynomial* w_i *(*$w'_{i'}$*). We say that* G *and* G' *are isomorphic if there exists a bijection* $f : V(G) \rightarrow V(G')$ *and another bijection* g *mapping variables occurring in the polynomials in* (G, C) *to variables occurring in the polynomials in* (G', C') *which satisfy the following conditions:*

1. $w'_i = g(w_{f(i)})$,
2. $r'_{ij} = g(r_{f(i),f(j)})$ *for all* $i, j \in V(G)$,
3. $C' = g(C)$.

The above definition is more complicated than the one for cell graphs of $\mathbf{FO}^2$ sentences because we need to make sure that when we discover an isomorphism of the cell graph, it will not "break" the cardinality constraints.

Finally we are ready to formally show that cell graph isomorphism can be used also for $\mathbf{C}^2$ sentences.

Theorem 3. *Let* φ *and* ψ *be two* $\boldsymbol{C}^2$ *sentences and* φ' *and* ψ' *be their encoding into* $\boldsymbol{FO}^2$ *sentences with cardinality constraints. Let* $(G_{\varphi'}, C_{\varphi'})$ *and* $(G_{\psi'}, C_{\psi'})$ *be their respective cell graphs with constraints. If* $(G_{\varphi'}, C_{\varphi'})$ *is isomorphic to* $(G_{\psi'}, C_{\psi'})$*, then*

$$\mathsf{WFOMC}(\varphi, n, w, \overline{w}) = \mathsf{WFOMC}(\psi, n, w, \overline{w})$$

for any domain size $n \in \mathbb{N}$ *and any weights* $(w, \overline{w})$.

Proof. The proof is a straightforward extension of the proof of Theorem 2. □

Example 27. *Consider the following sentences:*

$$\varphi_1 = \forall x\ P(x),$$
$$\varphi_2 = \forall x\ \forall y R(x, y).$$

It is easy to see that they share the same combinatorial spectrum, i.e.

$$\mathfrak{S}(\varphi_i) = 1, 1, 1, 1, \ldots$$

Their cell graphs are isomorphic; each of these cell graphs consist of a single node with all weights set to 1.

In contrast to the previous isomorphism-based pruning techniques, this one is *not-safe-to-delete* since it does not use the syntactical description of a sentence.

4.4.2.8 *Decomposable Sentences*

Definition 4 defines redundancy of a sentence within a set of sentences. We intentionally extend this definition to inter-sentences setup:

Definition 14. *We also consider a sentence $\varphi \in \Phi$ redundant if there are two other sentences φ', φ'' such that $\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi') \cdot \mathfrak{S}(\varphi'')$, where the product $\cdot$ is taken element-wise.*

This method is based on the following observation, which is well-known among others in lifted inference literature [134]: let $\varphi = \varphi_1 \wedge \varphi_2$ be a first-order logic sentence. If φ_1 and φ_2 use disjoint sets of predicates, then it is not hard to show that

$$\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi_1) \cdot \mathfrak{S}(\varphi_2)$$

where the product is taken element-wise and both $\mathfrak{S}(\varphi_1)$ and $\mathfrak{S}(\varphi_2)$ are understood to be computed only over the languages consisting of the predicates contained in φ_1 and φ_2, respectively.[16]

Example 28. *Consider the following sentences:*

$$\begin{aligned} \varphi_1 &= \exists x\ P_0(x), \\ \varphi_2 &= \exists^{=2} x\ P_1(x), \\ \varphi_3 &= \varphi_1 \wedge \varphi_2. \end{aligned}$$

Then, we have the following combinatorial spectra:

$$\begin{aligned} \mathfrak{S}(\varphi_1) &= 1, 3, 7, 15, \ldots \\ \mathfrak{S}(\varphi_2) &= 0, 1, 3, 6, \ldots \\ \mathfrak{S}(\varphi_3) &= 0, 3, 21, 90, \ldots \end{aligned}$$

Note that $\mathfrak{S}(\varphi_3)$ can be also computed as $\mathfrak{S}(\varphi_1) \cdot \mathfrak{S}(\varphi_2)$.

This method is *safe-to-delete* since it restricts the language of sentences. Those eliminated sentences can be generated in a post-processing step anyway.

4.4.2.9 *Reflexive Atoms*

If a sentence φ contains atoms of some binary predicate R only in the form $R(x, x)$ or $R(y, y)$ then all the ground atoms $R(i, j)$, where i and j are domain elements and $i \neq j$, are unconstrained by φ. It follows that

$$\mathfrak{S}(\varphi) = \mathfrak{S}(\varphi') \cdot \mathfrak{S}(\varphi'')$$

where $\varphi' = \forall x\ \neg R(x, x)$ and φ'' is a sentence obtained by replacing all occurrences of $R(x, x)$ by $P_R(x)$ and occurrences of $R(y, y)$ by $P_R(y)$ where P_R is a fresh predicate. Here, φ' accounts for all possible configurations of the atoms $R(i, j)$ with arguments $i \neq j$.

16 Indeed, this resembles the most typical language bias – connected clauses – employed in most FOL rule and patter mining approaches. There, however, a clause is a single connected component w.r.t. variables, whereas we consider a set of disjunctions w.r.t. predicates.

Example 29. *Consider* $\varphi = (\forall x\ R(x,x)) \wedge (\exists x\ P(x))$. *Then,x we construct* φ' *and* φ'' *according to the prescription above, i.e.*

$$\varphi' = \forall x\ \neg R(x,x),$$
$$\varphi'' = (\forall x\ P_R(x)) \wedge (\exists x\ P(x)).$$

Then we have the following spectra:

$$\begin{aligned} \mathfrak{S}(\varphi') &= 1, 4, 64, 4096, \ldots \\ \mathfrak{S}(\varphi'') &= 1, 3, 7, 15, \ldots \\ \mathfrak{S}(\varphi) = \mathfrak{S}(\varphi') \cdot \mathfrak{S}(\varphi'') &= 1, 12, 448, 61440, \ldots \end{aligned}$$

It follows that such a sentence φ is *not-safe-to-delete*. Deleting such a sentence would destroy the completeness of the search, but adding it to the database is redundant as there is a sentence generating the same spectrum having one more unary predicate.

4.4.2.10 *Remark on Interestingness*

Our effort in this chapter is biased towards a database containing a fragment of combinatorial integer sequences. Nevertheless, one might think of some of such sequences *uninteresting*. For instance, consider a sequence of all but one zero, i.e.

$$0, 1, 0, 0, \ldots.$$

While detecting all sentences that correspond to this particular combinatorial spectrum may prove to be hard,[17] we can, at least, remove a fragment of sentences that trivially corresponds to such a combinatorial spectrum. For instance, it is easy to construct the sentence

$$(\forall x\ P(x)) \wedge (\exists^{=2} x\ P(x))$$

that produces the spectrum above and is very easy to check. However, if this spectrum is really uninteresting is left out on a user.

4.5 EXPERIMENTS

In this section, we experimentally evaluate the effectiveness of the proposed approach for constructing the database of integer sequences described above within a reasonable amount of time. The goal of the following evaluation is to address the following questions:[18]

1. How do the proposed redundancy-checking methods impact the construction of the integers database?
2. Does the proposed approach of integer database construction discover new integer sequences?

17 Intuition for this stems from our argument on deciding whether two sentences produce the same combinatorial spectrum in Section 4.3.

18 In addition, Appendix B contains one more experiment focused on the comparison of the redundancy-checking techniques in isolation, i.e., without the spectra computation part.

4.5.1 *Filling the Database of Integer Sequences*

To answer the first question, we ran Algorithm 1 with two language restrictions, namely $\mathbf{FO}^2$ and $\mathbf{C}^2$ setups, to evaluate the effectiveness of the techniques described in Section 4.4.2 against a vanilla approach, which would implement the most natural redundancy-checking techniques. The metric of interest for this experiment is the time needed to construct the database, i.e., both sentence generation and combinatorial spectra computation time.

We set a time limit of five minutes for the computation of combinatorial spectra per sentence. We restricted the language of sentences in order to make a fair comparison; the language restrictions follow: a single unary and binary predicate, sentences with at most 2 clauses, each one having at most 5 literals, i.e., $\mathcal{P} = \{R/2, P/1\}$, $M = 2$, $M' = 2$, $J = 5$, and $\mathcal{K}_{\mathbf{FO}^2} = \{\forall, \exists\}$ and $\mathcal{K}_{\mathbf{C}^2} = \{\forall, \exists, \exists^{=1}\}$ for $\mathbf{FO}^2$ and $\mathbf{C}^2$ experiment, respectively. We further forbid tuples of quantifiers where one is in the form $\exists^{=k}x$ and the other is either $\exists^{=l}y$ or $\exists y$, i.e. $\exists x \exists^{=k} y$, $\exists^{=k} x \exists y$, and $\exists^{=k} x \exists^{=l} y$. When computing combinatorial spectra, these three combinations do not scale well, so we could not compute their combinatorial spectra within the time limit we used for filling in the database. For the same reason, we restricted the number of literals to at most 1 in a clause with a counting quantifier. Each sentence generator was executed with 51GB of memory and 48 hours of computation time.

In order to prevent a deadlock in the computation process, we relaxed the method *Tautologies & Contradiction* to act as a soft filter – if a theorem prover does not check a sentence within a 30 second limit, the sentence is processed as usual.[19]

We start with a baseline consisting of just the method that filters out sentences that are isomorphic (using the standard notion of isomorphism used in pattern mining literature [88], which does not consider renaming predicates) and with pruning of *decomposable sentences* – these are the very essentials any reasonable method would probably implement. Then, we enhance the baseline with *Tautologies & Contradiction*. Similarly, we add a single pruning technique on top of the previous one in the following order: *Isomorphic Sentences*, *Negations*, *Permuting Arguments*, *Reflexive Atoms*, *Trivial Constraints*, $\theta *$, and *Cell Graph Isomorphism*.

Our aim with these experiments was to assess the effect of our proposed pruning techniques. The results are depicted in Figure 1 and Figure 2 for $\mathbf{FO}^2$ and $\mathbf{C}^2$, respectively. The pruning techniques help to scale up the database-filling process in two ways. Whereas the naive approach, e.g., *baseline*, generates many sentences fast, soon consuming all available memory,[20] *safe-to-delete* techniques lower the memory requirements significantly. All pruning techniques consume some computation time, but that is negligible compared to the time needed for computing combinatorial spectra, which is the most time-demanding part of the task; e.g., compare Figure 1c with Figure 1b. Since the pruning methods, including those which are *not-safe-to-delete*, reduce the number of computations of combinatorial spectra, their use quickly pays off, as can be seen from Figure 1b and Figure 2b which show the estimated time to fill in the database. The most basic methods that do not use

19 *Cell Graph Isomorphism* yields contradictory sentences as a byproduct when computing valid cells.

20 Hence not making it into the final layer of sentences with 10 literals.

the full set of our pruning techniques, i.e., *basline* and *Tautologies & Contradictions*, generate an extremely high number of (mostly redundant) sentences whose spectra computation would take thousands of hours. Therefore, we only estimated the runtime by computing the spectra for a random sample of sentences for these methods. Using all available pruning techniques yields the fastest database construction time.

4.5.2 *An Initial Database Construction*

To answer the second question, we used our algorithm to generate an initial database of combinatorial sequences. We let the sentence generator run for 5 days to obtain a collection of sentences and their combinatorial spectra on a machine with 500GB RAM and 128 processors (we used multi-threading). As in the previous experiment, we used a five-minute time limit for the combinatorial spectrum computation of a sequence. The result was a database containing over $26,000$ unique integer sequences. We further split our discussion into two cases – generated sequences contained in OEIS and brand-new sequences.

Sequences in OEIS: For each of the sequences in our database, we queried OEIS to determine if the sequence matches a sequence that is already in OEIS. We found that 301 of the sequences[21] were present in OEIS – this makes $\approx 1.2\%$ of the sequences we generated. This may not sound like much, but it is undoubtedly non-negligible. This number serves as a check that our database contains some known combinatorial problems. In addition, our goal was to generate primarily new sequences, not to cover the OEIS database,[22] which is not doable by our approach because of two reasons. Firstly, our database and OEIS are counterparts since the former contains a subset of combinatorial integer sequences, while the latter contains both combinatorial and non-combinatorial integer sequences. Secondly, our approach does not generate all combinatorial integer sequences, e.g., *decomposable sentences* is one of them and the syntactical form. Several interesting generated sequences that happened to be in OEIS are shown in Table 1.

An example of an interesting sequence is the last one in Table 1. This sequence had no combinatorial characterization in OEIS before we published our work [125]. We can obtain such a characterization from the $\mathbf{C}^2$ sentence that generated it:[23]

$$(\forall x \forall y\ P(x) \lor \neg P(y) \lor \neg R(x,y)) \land (\forall x \exists^{=1} y\ R(x,y)).$$

This can be interpreted as follows: *We are counting configurations consisting of a function* $r : [n] \to [n]$ *and a set* $p \subseteq [n]$ *that satisfy that if* $y = r(x)$ *and* $y \in p$ *then* $x \in p$. While this may not be a profound combinatorial problem,

21 See a larger sample in Appendix C, or browse a snapshot of our database at https://fluffy.jung.ninja.

22 An interesting task on its own which is the aim of [27, 40, 67].

23 For easier readability, we replaced the predicate $R/2$ by its negation, which does not change the spectrum.

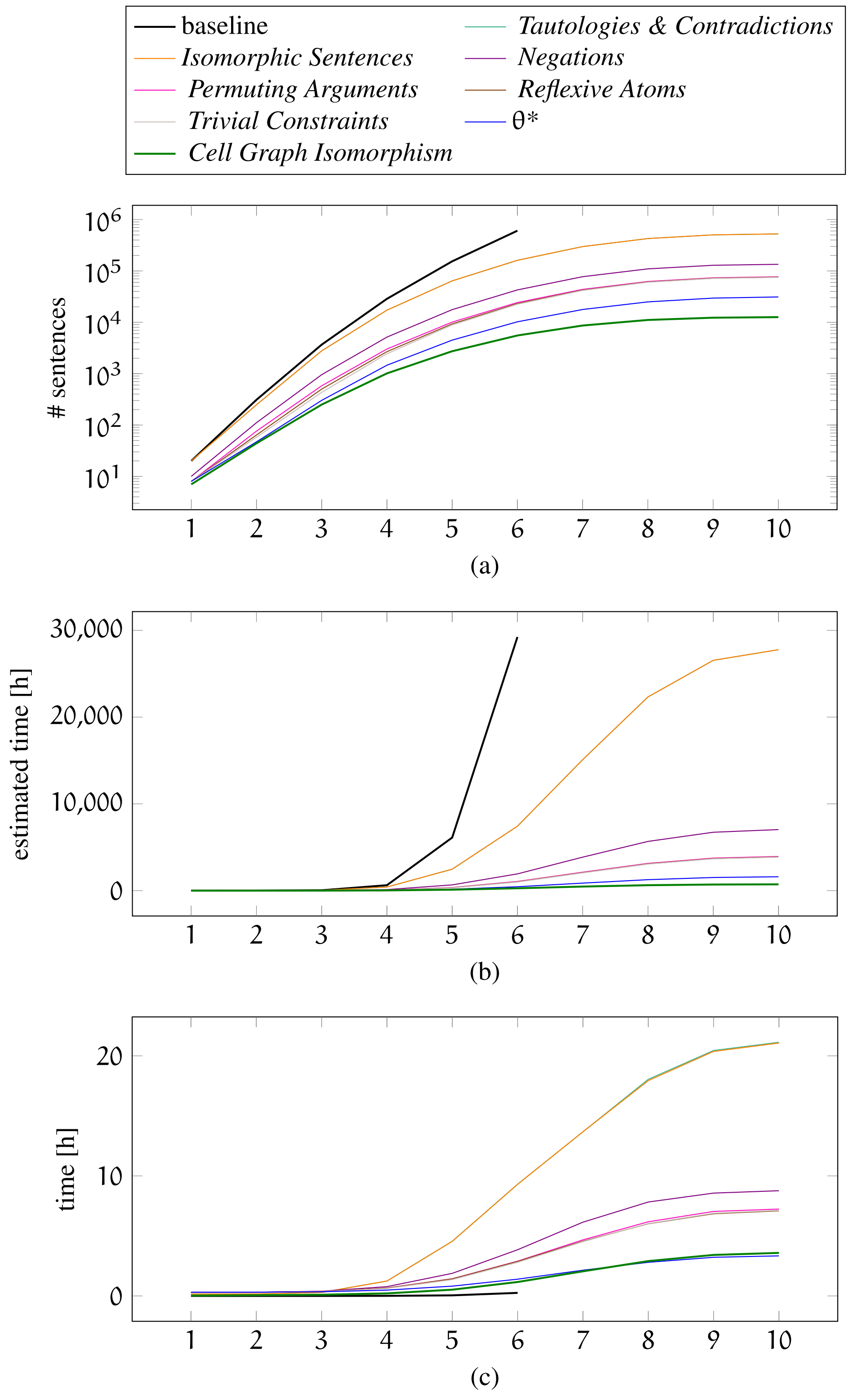

Figure 1: Cumulative number of $\mathbf{FO}^2$ sentences (a) with at most x literals, the expected time to fill in the database (b), and the time needed to generate sentences (c). At most 5 literals per clause, at most 2 clauses per sentence, one unary, and one binary predicate.

it provides a combinatorial interpretation of the sequence at hand. We would not be able to find this description without our database. Besides others, this case shows that our approach can enrich descriptions of some entries in OEIS.

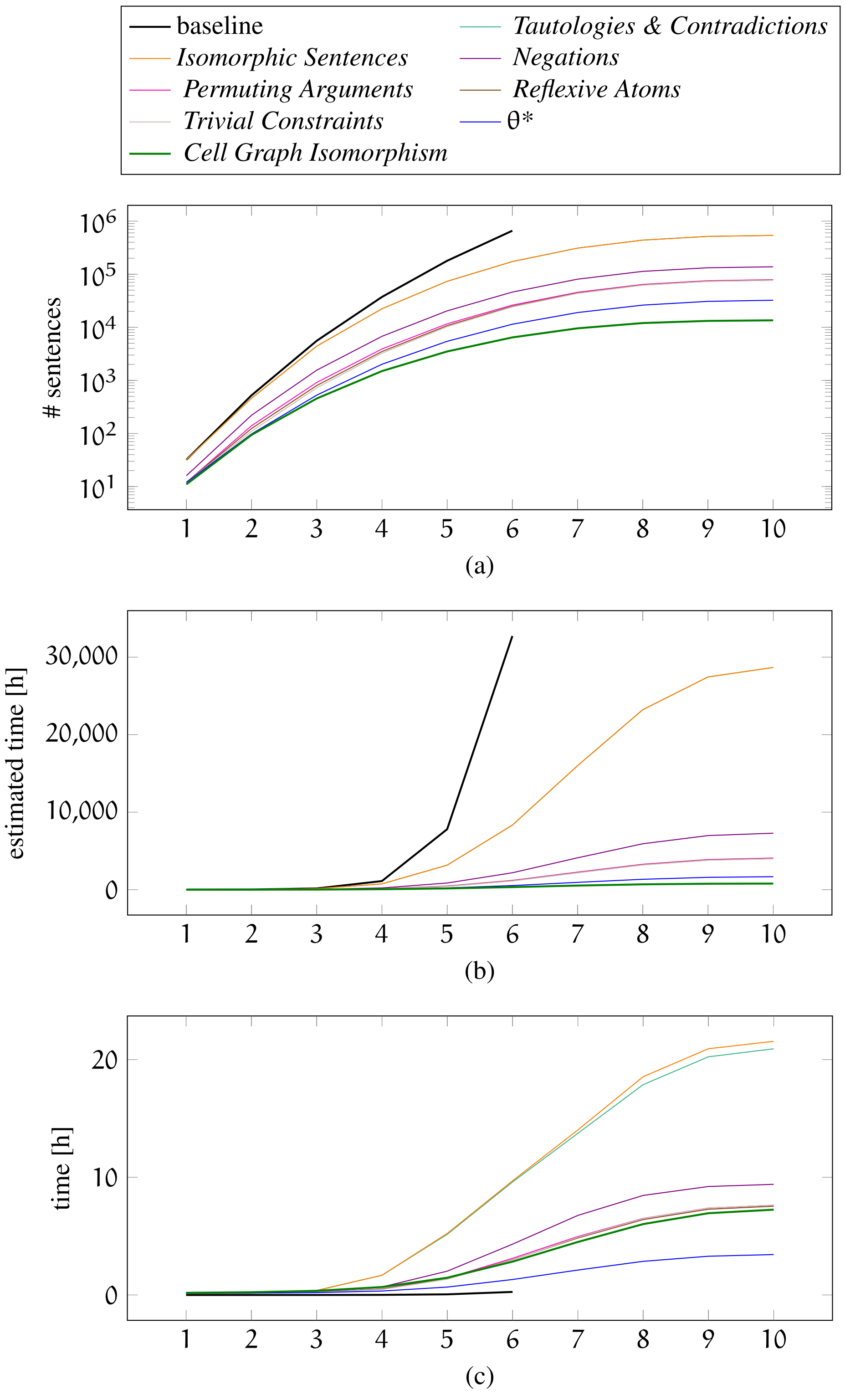

Figure 2: Cumulative number of $\mathbf{C}^2$ sentences (a) with at most x literals, the expected time to fill in the database (b), and the time needed to generate sentences (c). At most 5 literals per clause, at most 2 clauses per sentence, quantifiers $\forall, \exists, \exists^{=1}$, one unary, and one binary predicate.

New sequences: The rest of the database in this experiment, roughly $25,000$ combinatorial integer sequences, is what we hoped to get in the first place. Let us discuss several examples of those. The first of these examples is the sequence

Table 1: A sample of sequences that are combinatorial spectra of sentences generated by our algorithm that also appear in OEIS.

Sentence	**OEIS ID**	**OEIS name**
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\forall x \forall y\ R(x,x) \lor R(x,y) \lor \neg R(y,x))$	A85	Number of self-inverse permutations on n letters, also known as involutions; number of standard Young tableaux with n cells.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x))$	A142	Factorial numbers: $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot ... \cdot n$ (order of symmetric group S_n, number of permutations of n letters).
$(\exists x \exists y\ P(x) \lor R(x,y)) \land (\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x))$	A165	Double factorial of even numbers: $(2n)!! = 2^n * n!$.
$(\forall x\ R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(x,y)) \land (\forall x \exists^{=1} y\ \neg R(y,x))$	A166	Subfactorial or rencontres numbers, or derangements: number of permutations of n elements with no fixed points.
$(\forall x \forall y\ R(x,y) \lor \neg R(y,x)) \land (\exists x\ R(x,x)) \land (\forall x \exists^{=1} y \neg R(x,y))$	A1189	Number of degree-n permutations of order exactly 2.
$(\forall x\ R(x,x)) \land (\exists x\ R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(y,x)) \land (\exists^{=1} x \forall y\ R(x,y))$	A37184	Functional digraphs with 1 node not in the image.
$(\exists x\ \neg R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(y,x))$	A45531	Number of sticky functions: endofunctions of $[n]$ having a fixed point.
$(\forall x \forall y\ P(x) \lor R(x,y)) \land (\forall x \forall y\ \neg P(x) \lor R(y,x))$	A47863	Number of labeled graphs with 2-colored nodes where black nodes are only connected to white nodes and vice versa.
$(\forall x \forall y\ R(x,y) \lor \neg R(y,y)) \land (\exists x \forall y\ R(x,y)) \land (\exists x\ \neg R(x,x)) \land (\exists^{=1} x\ \neg R(x,x))$	A58877	Number of labeled acyclic digraphs with n nodes containing exactly $n-1$ points of in-degree zero.
$(\forall x \forall y\ P_0(x) \lor P_1(y) \lor P_2(y)) \land (\forall x\ P_0(x) \lor \neg P_1(x))$	A85350	Binomial transform of poly-Bernoulli numbers A027649.
$(\forall x\ R(x,x)) \land (\forall x \exists y\ \neg R(x,y)) \land (\forall x \exists y\ \neg R(y,x))$	A86193	Number of $n \times n$ matrices with entries in $\{0,1\}$ with no zero row, no zero column and with zero main diagonal.
$(\forall x \forall y\ P_0(x) \lor R(y,y)) \land (\exists x \exists y\ P_0(x) \lor P_1(x) \lor R(x,y))$	A88668	Number of nXn matrices over GF(2) with characteristic polynomial $x^{n-1} * (x-1)$.
$(\exists x \forall y\ R(x,y) \lor \neg R(y,x)) \land (\forall x\ R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(x,y))$	A246189	Number of endofunctions on $[n]$ where the smallest cycle length equals 2.
$(\forall x \forall y\ P(x) \lor \neg P(y) \lor R(x,y)) \land (\forall x \exists^{=1} y\ \neg R(x,y))$	A290840	$a(n) = n! \cdot [x^n] \frac{\exp(n \cdot x)}{1+LambertW(-x)}$

$$0, 0, 6, 72, 980, 15360, \ldots$$

generated by the sentence

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

We can interpret it as counting the number of functions $r : [n] \to [n]$ without fixed points and with image not equal to $[n]$. Another example is the sequence

$$1, 7, 237, 31613, 16224509, 31992952773, \ldots$$

which corresponds to the sentence

$$(\forall x \exists y\ R(x, y)) \land (\exists x \forall y\ R(x, y) \lor R(y, x))$$

and counts directed graphs on n vertices in which every vertex has non-zero out-degree and there is a vertex that is connected to all other vertices (including to itself) by either an outgoing or incoming edge. Yet another example is the sequence

$$1, 5, 127, 12209, 4329151, 5723266625, \ldots$$

corresponding to the sentence

$$(\forall x \exists y\ R(x, y)) \land (\exists x \forall y\ R(x, y))$$

which counts directed graphs where every vertex has non-zero out-degree and at least one vertex has out-degree n, which is also the same as the number of binary matrices with no zero rows and at least one row containing all ones. These examples correspond to the simpler structures in the database, there are others which are more complex (and also more difficult to interpret). For example, another sequence

$$0, 3, 43, 747, 22813, 1352761, \ldots$$

constructed by our algorithm, given by the sentence

$$\begin{aligned} &(\forall x\ \neg R(x, x)) \land (\forall x \forall y\ \neg R(x, y) \lor R(y, x)) \land \\ &\land (\exists x \forall y\ \neg R(x, y) \lor \neg P(y)) \land (\exists x \exists y\ R(x, y)) \end{aligned}$$

which counts undirected graphs without loops with at least one edge and with vertices labeled by two colors, red and black (red corresponding to $P(x)$, and black corresponding to $\neg P(x)$) such that there is at least one vertex not connected to any of the red vertices (note that this vertex can itself be red). We could keep on listing similar sequences,[24] but we believe the handful we showed here gives a sufficient idea about the kind of sequences one could find in the database constructed by our system.

4.5.3 *Conclusion*

The latter experiment shows that the idea of discovering interesting combinatorial sentences using first-order logic works in practice. Although we constructed only a tiny initial database, we made a small contribution to the well-established OEIS. The former experiment shows that the proposed redundancy-checking techniques scale up the sentence-generation process and save computational resources in order of magnitudes compared to a naive approach.

24 Continuing at this pace, this would take roughly $8,000$ more pages.

4.6 RELATED WORK

There is a long tradition of research focusing on automated discovery in mathematics dating back to the AM program [63] through the well-known HR system [23, 24] and current approaches [28, 47, 81, 132]. Usually, methods from this field aim to either discover new concepts in mathematics or construct new conjectures about some objects of interest automatically, e.g., groups, graphs, programs, and prove them. The HR system [25] was used to generate new integers from initial knowledge that, e.g., contains numbers, operations between numbers, and a set of rules. Several recent works use OEIS sequences as inputs for program synthesis, e.g., QSynt [40] or [27, 67]. However, these works have different biases and are orthogonal to ours since their initial aim is to synthesize programs or formulae covering entries in OEIS.[25][26]

To the best of our knowledge, there has been almost no prior work on automated generation of combinatorial sequences, with the work [2], which focuses on sequences generated by permutations avoiding certain patterns, being an exception. However, there are works that intersect with the work presented in this chapter in certain aspects. Our bias towards combinatorics puts us very close to the works on lifted inference [9, 42, 54, 103, 134, 135]. In fact, our approach would not be possible without lifted inference as it constructs the bridge between first-order logic and integer sequences since we use the algorithms and the concept of cell graphs from [17]. Although the detection of isomorphic sentences can be seen as an extension of standard isomorphic pruning from pattern-mining literature [88], it is similar, in spirit, to the techniques presented in [16]. There, however, the main focus lies on propositional logic problems, whereas we use these techniques for problems with first-order logic sentences.

The closest line of work at the intersection of combinatorics and artificial intelligence are the works [124] and [131]. However, those works do not attempt to generate new sequences or new combinatorics results, as they mostly aim at solving textbook-style combinatorial problems, which is still a highly non-trivial problem too, though.

4.7 FUTURE WORK

To the best of our knowledge, this is the first work along the lines of *automatically combinatorics* interwinding first-order logic sentences and integer sequences. In this chapter, we focused only on a single component of the whole environment – generation of logical sentences. Each component of the database construction can be extended, and the database can be used for other machine-learning tasks. We briefly outline a few extensions of the database construction process.

Techniques for lifted first-order model counting are a vital part of our approach. Hence, progress in this area can extend the number of entries in the database with almost no modification at all. This is usually done by proving a particular class of sentences to be domain liftable, e.g., by adding tree axiom [133] or linear order axiom [130].

25 For instance, the OEIS's sequence A290840, which we discussed in Section 4.5, corresponds to the output of https://loda-lang.org/edit/?oeis=290840 Loda program.

26 See Chapter 7 for demonstration of learned models by these methods and ours.

The other component of our approach, which was the main topic of this chapter, touched its limits after a few weeks of computation, leading to a database with $161,000$ sentences and $52,000$ unique integer sequences. While our method enumerated all sentences in a given syntactical form, an incomplete approach, e.g., utilizing reinforcement learning [40], sounds like the next natural step that could overcome the current limitations of our approach. In fact, any similar extension, depth-breath search, literal-wise refinements, etc., can build up on the pruning techniques described in this chapter. Recall that these techniques have to be adjusted to their usage; for example, some techniques may change to *not-safe-to-delete* from our *safe-to-delete*, and vice versa. A more profound investigation of entailment-based pruning, i.e., incorporating more complex, or even full, reasoning on the intra-sentence level, is left for future work as we relaxed this technique to preserve scalability.

5 SENTENCE SPACE PRUNING USING DOMAIN KNOWLEDGE

Domain knowledge is your compass in any domain.

— Richard Warepam [138]

In the previous chapter, we deployed a standard sentence space traversal algorithm with a dozen, less or more complicated, pruning methods in order to generate integer sequences in a fully automatic way. A natural question arises:

Having some knowledge of the sentence space, can we prune the space automatically?

In this chapter, we answer this question positively in a single scenario that covers standard ILP systems. A lack of domain knowledge is not an issue since we learn domain knowledge by observing regularities in the data.

In this chapter, we work within Section 2.3.1 learning setup, i.e., learning from interpretation.[1] Let us start with two motivating examples.

Example 30. *Let us consider the following two sentences for some target concept* C*:*

$$\alpha = \forall x \; \mathsf{Animal}(x) \wedge \mathsf{Cod}(x) \Rightarrow \mathsf{C}(x),$$
$$\beta = \forall x \; \mathsf{Fish}(x) \wedge \mathsf{Cod}(x) \Rightarrow \mathsf{C}(x).$$

Intuitively, these two sentences are equivalent since every cod *is a* fish *and every* fish *is an* animal*. Yet, standard rule-mining systems would need to consider both of these sentences separately because* α *and* β *are not isomorphic, not* θ*-equivalent, and neither of them* θ*-subsumes the other.*

Example 31. *Problems with redundant hypotheses abound in datasets of molecules, which are widespread in the ILP literature. For instance, consider the following two sentences:*

$$\alpha = \forall x \forall y \forall z \; \mathsf{Carb}(y) \wedge \mathsf{Bond}(x,y) \wedge \mathsf{Bond}(y,z) \wedge \mathsf{Hydro}(z) \Rightarrow \mathsf{C}(x),$$
$$\beta = \forall x \forall y \forall z \; \mathsf{Carb}(x) \wedge \mathsf{Bond}(x,y) \wedge \mathsf{Bond}(z,y) \wedge \mathsf{Hydro}(z) \Rightarrow \mathsf{C}(x).$$

These two sentences intuitively represent the same molecular structures (a carbon and a hydrogen both connected to the same atom of unspecified type). They differ only in the syntactical form of bond *direction, which is, in reality, undirected. Again, however, their equivalence cannot be detected without the domain knowledge that bonds in molecular datasets are symmetric.*[2]

1 Contrary to the previous chapter, universally quantified clauses, and variable-based isomorphism (Section 2.1) is enough.

2 In the physical world, bonds do not necessarily have to be symmetric, e.g., there is an obvious asymmetry in polar bonds. However, it is a common simplification in data mining on molecular datasets to assume that bonds are symmetric.

These two examples correspond to the scenario this chapter aims for. In a nutshell, our approach consists of two steps: i) learning a domain theory that covers the data, e.g., learning that *bond* relation is symmetric, ii) utilizing that knowledge to prune the space of sentences, e.g., saying that α and β from Example 31 are equal w.r.t. bond symmetry. We solve the first step straightforwardly and focus mainly on the second.

5.1 SATURATIONS

The main technical ingredient of our approach is the following notion of *saturation*.

Definition 15 (Saturation of a clause)**.** *Let $\mathcal{B}$ be a clausal theory and α a clause (without constants or function symbols). If $\mathcal{B} \not\models \alpha$, we define the saturation of α w.r.t. $\mathcal{B}$ to be the maximal clause α' satisfying:*

1. *$vars(\alpha') = vars(\alpha)$*
2. *$\mathcal{B} \wedge \alpha'\theta \models \alpha\theta$*

for any injective grounding substitution θ. If $\mathcal{B} \models \alpha$, we define the saturation of α w.r.t. $\mathcal{B}$ to be $\top$.

When $\mathcal{B}$ is clear from the context, we will simply refer to α' as the saturation of α.

Definition 15 naturally leads to a straightforward procedure for computing the saturation of a given clause. Let $\mathcal{L} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be the set of all literals which can be constructed using variables from α and predicate symbols from $\mathcal{B}$ and α. Let θ be an arbitrary injective grounding substitution; note that we can indeed take θ to be arbitrary because $\mathcal{B}$ and α do not contain constants. If $\mathcal{B} \not\models \alpha$, the saturation of α is given by the following clause:

$$\bigvee\{\tau \in \mathcal{L} \mid \mathcal{B} \models \neg\tau\theta \vee \alpha\theta\}. \tag{6}$$

This means in particular that we can straightforwardly use the SAT based theorem proving method from Section 3.3 to compute saturations. The fact that Equation (6) correctly characterizes the saturation can be seen as follows. If α' is the saturation of α then $\mathcal{B} \wedge \alpha'\theta \models \alpha\theta$ by definition, which is equivalent to

$$\mathcal{B} \wedge \neg(\alpha\theta) \models \neg(\alpha'\theta).$$

We have

$$\begin{aligned}\neg(\alpha'\theta) &= \bigwedge\{\tilde{\tau}\theta \mid \mathcal{B} \wedge \neg(\alpha\theta) \models \tilde{\tau}\theta\} \\ &= \bigwedge\{\tilde{\tau}\theta \mid \mathcal{B} \wedge \tau\theta \models \alpha\theta\}\end{aligned}$$

and thus

$$\alpha'\theta = \bigvee\{\tau\theta \mid \mathcal{B} \wedge \tau\theta \models \alpha\theta\}.$$

Finally, since θ is injective, we have[3]

$$\alpha' = (\alpha'\theta)\theta^{-} = \bigvee\{\tau \mid \mathcal{B} \wedge \tau\theta \models \alpha\theta\}.$$

3 Note that we are slightly abusing notation here, as θ^{-1} is not a substitution.

Example 32. *Let us consider the following theory*

$$\mathcal{B} = \{\forall x \forall y \ \mathsf{Friends}(x, y) \Rightarrow \mathsf{Friends}(y, x)\}$$

which expresses the fact that friendship is a symmetric relation and a clause

$$\alpha = \forall x \forall y \ \mathsf{Friends}(x, y) \Rightarrow \mathsf{Happy}(x).$$

To find the saturation of this clause, we first need a suitable injective substitution θ*; let us take* $\theta = \{x \mapsto c_1, y \mapsto c_2\}$*. Then we have*

$$\begin{aligned}\mathcal{B} \cup \neg(\alpha\theta) &= \mathcal{B} \cup \{\mathsf{Friends}(c_1, c_2) \wedge \neg \mathsf{Happy}(c_1)\} \\ &\models \mathsf{Friends}(c_1, c_2) \wedge \mathsf{Friends}(c_2, c_1) \wedge \neg \mathsf{Happy}(c_1),\end{aligned}$$

After negating the latter formula and inverting the substitution (noting that it is injective) we get the following saturation

$$\alpha' = \forall x \forall y \ \mathsf{Friends}(x, y) \wedge \mathsf{Friends}(y, x) \Rightarrow \mathsf{Happy}(x).$$

Now, let us consider another clause

$$\beta = \forall x \forall y \ \mathsf{Friends}(x, y) \Rightarrow \mathsf{Happy}(y).$$

This clause is not isomorphic to α*. However, it is easy to see that its saturation*

$$\beta' = \forall x \forall y \ \mathsf{Friends}(x, y) \wedge \mathsf{Friends}(y, x) \Rightarrow \mathsf{Happy}(y)$$

is isomorphic to the saturation α' *of* α*.*

The next proposition will become important later as it will allow us to replace clauses by their saturations when learning from interpretations.

Proposition 1. *If* α' *is a saturation of* α *w.r.t.* $\mathcal{B}$ *then* $\mathcal{B} \wedge \alpha' \models \alpha$*.*

Proof. We have $\mathcal{B} \wedge \alpha' \models \alpha$ if and only if $\mathcal{B} \wedge \alpha' \wedge \neg\alpha$ is unsatisfiable. Skolemizing $\neg\alpha$, this is equivalent to $\mathcal{B} \wedge \alpha' \wedge \neg(\alpha\theta_{Sk})$ being unsatisfiable, where θ_{Sk} is a substitution representing the Skolemization. As in Section 3.3, we find that the satisfiability of $\mathcal{B} \wedge \alpha' \wedge \neg(\alpha\theta_{Sk})$ is also equivalent to the satisfiability of the grounding of $\mathcal{B} \wedge \alpha' \wedge \neg(\alpha\theta_{Sk})$ w.r.t. the Skolem constants introduced by θ_{Sk}. In particular, this grounding must contain the ground clause $\alpha'\theta_{Sk}$. From the definition of saturation, we have that $\mathcal{B} \wedge \alpha'\theta_{Sk} \wedge \neg(\alpha\theta_{Sk}) \models \bot$, note that θ_{Sk} is injective. It follows that $\mathcal{B} \wedge \alpha' \wedge \neg\alpha \models \bot$, and thus also $\mathcal{B} \wedge \alpha' \models \alpha$. □

The next proposition shows that saturations cover the same examples as the clauses from which they were obtained, when $\mathcal{B}$ is a domain theory that is valid for all examples in the dataset.

Proposition 2. *Let* $\mathcal{B}$ *be a clausal theory such that for all examples* e *from a given dataset it holds that* $e \models \mathcal{B}$*. Let* α *be a clause and let* α' *be its saturation w.r.t.* $\mathcal{B}$*. Then for any example* e *from the dataset we have* $(e \models \alpha) \Leftrightarrow (e \models \alpha')$*.*

Proof. From the characterization of saturation in Equation (6), it straightforwardly follows that $\alpha \models \alpha'$, hence $e \models \alpha$ implies $e \models \alpha'$.

Conversely, if $e \models \alpha'$, then we have $e \models \mathcal{B} \wedge \alpha'$, since we assumed that $e \models \mathcal{B}$. Since we furthermore know from Proposition 1 that $\mathcal{B} \wedge \alpha' \models \alpha$, it follows that $e \models \alpha$. □

Finally, we define *positive* and *negative saturations*, which only add positive or negative literals to clauses. Among others, this will be useful in settings where we are only learning Horn clauses or hard constraints.

Definition 16. *A positive (resp. negative) saturation of* α *is defined as*

$$\alpha'' = \alpha \cup \{\tau \in \alpha' \mid \tau \textit{ is a positive (resp. negative) literal}\}$$

where α' *is a saturation of* α*.*

Proposition 1 and Proposition 2 are also valid for positive or negative saturations; their proofs can be straightforwardly adapted. When computing the positive (resp. negative) saturation, we can restrict the set $\mathfrak{L}$ of candidate literals to the positive (resp. negative) ones. This can speed up the computation of saturations significantly.

5.2 SEARCHING THE SPACE OF SATURATIONS

In this section, we show how saturations can be used together with refinement operators to search the space of clauses ordered by OI-subsumption, which we use to partially order the constructed sentences.[4] Specifically, we show that if we have a refinement operator that can completely generate some set of clauses then we can use the same refinement operator, in combination with a procedure for computing saturations, to generate the set of all saturations of the considered set of clauses. Since this set of saturations is typically smaller than the complete set of clauses (as many clauses can lead to the same saturated clauses), this is already beneficial for reducing the size of the sentence space. In Section 5.3, we show that it also allows us to very quickly prune clauses equivalent w.r.t. domain theory. First we give a definition of *refinement operator* [126].

Definition 17 (Refinement operator)**.** *Let* $\mathcal{L}$ *be a first-order language. A refinement operator*[5] *on the set* $\mathcal{A}$ *of all* $\mathcal{L}$*-clauses is a function* $\rho : \mathcal{A} \to 2^{\mathcal{A}}$ *such that for any* $\alpha \in \mathcal{A}$ *and any* $\beta \in \rho(\alpha)$ *it holds* $\alpha \preceq_{OI} \beta$*. A refinement operator* ρ *is complete if for any two clauses* α *and* β *such that* $\alpha \preceq_{OI} \beta$*, a clause* γ *isomorphic to* β *(*$\beta \approx_{iso} \gamma$*) can be obtained from* α *by repeated application of the refinement operator (i.e.* $\gamma \in \rho(\rho(\dots\rho(\alpha)\dots))$*).*

Most works define refinement operators w.r.t. θ-subsumption instead of OI-subsumption [126]. We need the restriction to OI-subsumption as a technical condition for Proposition 3 below. It should be noted, however, that our results remain valid for many refinement operators that are not specifically based on OI-subsumption, including all refinement operators that only add new literals to clauses. Also, note that we do not use OI-subsumption as a covering operator but only to structure the space of sentences. Therefore, there is no loss in what sentences can be learned.

The following definition formally introduces the combination of refinement operators and saturations.

4 We do not use OI-subsumption to check the entailment relation.

5 What we call refinement operator in this chapter is often called downward refinement operator. Since we only consider downward refinement operators here, we omit the word downward.

Definition 18 (Saturated refinement operator)**.** *Let $\mathcal{L}$ be a first-order language. Let ρ be a refinement operator on the set $\mathcal{A}$ of all $\mathcal{L}$-clauses containing at most n variables. Let $\mathcal{B}$ be a clausal theory. Let $\sigma_{\mathcal{B}} : \mathcal{A} \to \mathcal{A}$ be a function that maps a clause α to its saturation α' w.r.t. $\mathcal{B}$. Then the function $\rho_{\mathcal{B}} = \sigma_{\mathcal{B}} \circ \rho$ is called the saturation of ρ w.r.t. $\mathcal{B}$.*

Clearly, the saturation of a refinement operator w.r.t. some clausal theory $\mathcal{B}$ is a refinement operator as well. However, it can be the case that ρ is complete, whereas its saturation is not. As we will show next, this is not a problem for completeness w.r.t. the given theory $\mathcal{B}$ in the sense that saturations of all clauses from the given class $\mathcal{A}$ are guaranteed to be eventually constructed by the combined operator, when ρ is a complete refinement operator.

Proposition 3. *Let $\mathcal{L}$ be a first-order language. Let ρ be a complete refinement operator on the set of $\mathcal{L}$-clauses $\mathcal{A}$, $\mathcal{B}$ be clausal theory, $\sigma_{\mathcal{B}}$ a function that maps a clause α to its saturation α' w.r.t. $\mathcal{B}$ and let $\rho_{\mathcal{B}}$ be the saturation of ρ w.r.t. $\mathcal{B}$. Let $\alpha \in \mathcal{A}$ be a clause, $\mathcal{S}_\alpha$ and let $\mathcal{S}_\alpha^{\mathcal{B}}$ be the sets of clauses that can be obtained from α by repeated application of ρ and $\rho_{\mathcal{B}}$, respectively. Then for any clause $\beta \in \mathcal{S}_\alpha$ there is a clause $\beta' \in \mathcal{S}_\alpha^{\mathcal{B}}$ such that $\sigma_{\mathcal{B}}(\beta) \approx_{iso} \beta'$.*

Proof. We first note that if $A \preceq_{OI} B$ then $\sigma_{\mathcal{B}}(A) \preceq_{OI} \sigma_{\mathcal{B}}(B)$ (assuming an extended definition of OI-subsumption such that $A \preceq_{OI} \top$ for any A), which follows from the monotonicity of the entailment relation $\models$. Let us define $\mathcal{X} = \{\sigma_{\mathcal{B}}(A) \mid A \in \mathcal{S}_C\}$. Note that $\mathcal{X}$ and $\mathcal{S}_C^{\mathcal{B}}$ are not defined in the same way; $\mathcal{X}$ is the set of saturations of clauses in $\mathcal{S}_C$, whereas $\mathcal{S}_C^{\mathcal{B}}$ is the set of clauses that can be obtained by the saturated refinement operator $\rho_{\mathcal{B}}$ from the clause C. We need to show that these two sets are *equivalent*. Clearly, $\mathcal{S}_C^{\mathcal{B}} \subseteq \mathcal{X}$. To show the other direction, let us assume (for contradiction) that there is a clause $X \in \mathcal{X}$ for which there is no clause $Y \in \mathcal{S}_C^{\mathcal{B}}$ which is isomorphic to X. Let us assume that X is a minimal clause with this property, meaning that for any clause X' contained in the set $\mathcal{Z}_X = \{Z \in \mathcal{X} \mid Z \preceq_{OI} X, X \not\approx_{iso} Z\}$ there is a clause $Y' \in \mathcal{S}_C^{\mathcal{B}}$ which is isomorphic to X'. Clearly, if there is one such clause X then there is also a minimal one which follows from the fact that all the considered clauses are finite and $\preceq_{OI}$ is a partial order. Let us take a clause $X' \in \mathcal{Z}_X$ which is maximal[6] w.r.t. the ordering induced by $\preceq_{OI}$ and let Y' be the respective isomorphic clause from $\mathcal{S}_C^{\mathcal{B}}$. Then $\rho(Y')$ must contain a clause Y'', $Y' \not\approx_{iso} Y''$, that OI-subsumes X, which follows from completeness of the refinement operator ρ. However, then $\sigma_{\mathcal{B}}(Y'')$ must be contained in $\mathcal{S}_C^{\mathcal{B}}$. It must also hold that $\sigma_{\mathcal{B}}(Y'') \preceq_{OI} \sigma_{\mathcal{B}}(X) = X$. Here, $\sigma_{\mathcal{B}}(Y'') \preceq_{OI} \sigma_{\mathcal{B}}(X)$ follows from the already mentioned observation that if $A \preceq_{OI} B$ then $\sigma_{\mathcal{B}}(A) \preceq_{OI} \sigma_{\mathcal{B}}(B)$, and the equality $\sigma_{\mathcal{B}}(X) = X$ follows from the idempotence of $\sigma_{\mathcal{B}}$, noting that X is already a saturation of some clause. However, this is a contradiction with the maximality of X' and the corresponding Y'. □

6 If we ordered the set of clauses by θ-subsumption instead of OI-subsumption then there would not have to exist a maximal clause with this property.

5.3 PRUNING ISOMORPHIC SATURATIONS

When searching the space of clauses or, in particular, saturations of clauses, we should avoid searching through isomorphic clauses. It is easy to see that the sets of clauses generated by a (saturated) complete refinement operator ρ from two isomorphic clauses α and α' will contain clauses that are isomorphic, i.e., for any clause in the first set there will be an isomorphic clause in the second set and vice versa. Therefore, it is safe to prune isomorphic clauses during the search.

During sentence space traversal, searching algorithms keep some data structures, usually an open and closed lists, in order not to evaluate a clause multiple times. However, many of the clauses that are stored in such data structures will be equivalent, even if they are not isomorphic. Existing methods, even if they were removing isomorphic clauses during the search,[7] have to consider each of these equivalent clauses separately, which may significantly affect their performance. This is where using saturations of clauses w.r.t. some domain knowledge is most useful because it can replace the different implicitly equivalent clauses by their saturation. Therefore, we first compute the saturation of α and prune it if α' is isomorphic to a clause in an open or closed list.[8]

Example 33. *Let us again consider the two clauses from Example 30:*

$$\alpha = \forall x \forall y\ \mathsf{Animal}(x) \land \mathsf{Cod}(x) \Rightarrow \mathsf{C}(x),$$
$$\beta = \forall x \forall y\ \mathsf{Fish}(x) \land \mathsf{Cod} \Rightarrow \mathsf{C}(x).$$

Suppose that the theory $\mathcal{B}$ encodes the taxonomy of animals consists of the following clauses

$$\mathcal{B} = \{\forall x\ \mathsf{Cod}(x) \Rightarrow \mathsf{Fish}(x),$$
$$\forall x\ \mathsf{Fish}(x) \Rightarrow \mathsf{Animal}(x)\}.$$

Computing the saturations of α and β w.r.t. $\mathcal{B}$, we obtain

$$\alpha' = \forall x\ \mathsf{Animal}(x) \land \mathsf{Cod}(x) \land \mathsf{Fish}(x) \Rightarrow \mathsf{C}(x),$$
$$\beta' = \forall x\ \mathsf{Fish}(x) \land \mathsf{Cod}(x) \land \mathsf{Animal}(x) \Rightarrow \mathsf{C}(x),$$

which are isomorphic. Therefore both of them can be replaced by the same saturations while the corresponding algorithm keeps searching the sentence space.

Similarly, as shown above, for the two clauses from Example 30, saturations could be used to detect the equivalence of the two clauses from Example 31 w.r.t. the corresponding domain knowledge theory $\mathcal{B}$.[9]

In addition to equivalence testing, saturations can be used to filter trivial sentences, i.e., hypotheses covering every example, without explicitly computing their coverage on the dataset, which can be very costly on large datasets,

7 For instance, Farmr [88] or RelF [57] remove isomorphic clauses (or conjunctive patterns), but many existing ILP systems do not attempt removing isomorphic clauses.

8 Or any other structure a search algorithm uses.

9 Assuming $\mathcal{B} = \{\forall x \forall y\ \mathsf{Bond}(x,y) \Rightarrow \mathsf{Bond}(y,x)\}$, saturations of both clauses are isomorphic to $\forall x \forall y\ \mathsf{Carb}(y) \land \mathsf{Bond}(x,y) \land \mathsf{Bond}(y,x) \land \mathsf{Bond}(y,z) \land \mathsf{Bond}(z,y) \land \mathsf{Hydro}(z) \Rightarrow \mathsf{C}(x)$. The computation follows the same process as in Example 32.

or in scenarios where the evaluation of a sentence is complex, e.g., together with a theory. We illustrate this use of saturations in the following example.

Example 34. *Consider a domain theory*

$$\mathcal{B} = \{\forall x\ \neg \mathsf{Professor}(x) \vee \neg \mathsf{Student}(x)\}$$

which contains a hard constraint stating that no one can be both a student and a professor. Let us also consider a sentence

$$\alpha = \forall x\ \mathsf{Professor}(x) \wedge \mathsf{Student}(x) \Rightarrow \mathsf{Employee}(x).$$

If the domain theory $\mathcal{B}$ is correct, α should cover all examples from the dataset and is thus trivial. Accordingly, the saturation of α contains every literal, and is in particular equivalent to $\top$.

Finally, we note that it can be shown straightforwardly that the pruning method based on saturations does not affect the completeness of the sentence search methods considered here. Preservation of completeness follows, first, from the monotonicity of deduction in classical logic and, second, from the fact that we use isomorphism for pruning and not θ-equivalence.

5.3.1 *Why Relative Subsumption is Not Sufficient*

Although the motivation behind relative subsumption [102] is similar to ours, relative subsumption has two main disadvantages that basically disqualify it for the purpose of pruning the sentence space. The first problem is that pruning sentences that are equivalent w.r.t. relative subsumption may not guarantee the completeness of the search. This is the same issue as with pruning based on plain θ-subsumption, which, unlike pruning based on isomorphism, may lead to incompleteness of the search. Note that this is already the case in the more restricted setting of graph mining under homomorphism [110]. The second issue with relative subsumption is that it would need to be tested for all pairs of candidate sentences, whereas the pruning based on saturations and isomorphism testing allows us to use the more efficient hashing strategy based on the Weisfeiler-Lehman procedure.

5.4 LEARNING DOMAIN THEORIES

The domain theories that we want to use for pruning sentence space can be learned from the given training dataset $\mathcal{E}$. Every clause α in such a learned domain theory should satisfy

$$e \models \alpha \quad \forall e \in \mathcal{E}. \tag{7}$$

Further, we are interested in relatively small and compact theories to make fast computations. Hence, only clauses satisfying Equation (7) are added into the domain theory $\mathcal{B}$ only if there is no other clause from the theory that would θ-subsume them. The algorithm we use solves the following problem:

Given: A set of example $\mathcal{E}$ and an integer d.

Compute: The domain theory $\mathcal{B}$ such that each $\beta \in \mathcal{B}$ has at most d literals and there are not any two distinct $\beta_i, \beta_j \in \mathcal{B}$ such that $\beta_i \preceq_\theta \beta_j$.

Algorithm 2 describes how we construct such theories using a level-wise search procedure, starting with an empty domain theory (line 1). The level-wise procedure maintains a list of candidate clauses $\mathcal{A}_i$ (modulo isomorphism – lines 12-14)[10] with i literals. If a clause γ, from the list of candidate clauses, covers all examples (line 8), then it is removed from the list, and if there is no clause in the domain theory which θ-subsumes γ, then γ is also added to the domain theory (line 9). Each of the remaining clauses in the list, i.e., those which do not cover all examples in the dataset, are then extended in all possible ways by the addition of a literal in the next iteration of the main loop (line 5). This is repeated until a threshold on the maximum number of literals is reached. The covering of examples by the candidate clauses is checked using θ-subsumption as outlined in Section 3.2.

Algorithm 2 Pseudocode of Level-wise Domain-Theory Learner

Parameter: Set of examples $\mathcal{E}$ and integer d.
Output: Domain theory $\mathcal{B}$.

```
1:  $\mathcal{B} \leftarrow \emptyset$
2:  $\mathcal{A}_0 \leftarrow \{\emptyset\}$
3:  for $i \in \{1, 2, 3, \ldots, d\}$ do
4:      $\mathcal{A}_i \leftarrow \emptyset$
5:      for $\alpha \in \mathcal{A}_{i-1}$ do
6:          for $\gamma \in$ refinements$(\alpha)$ do
7:              if $\forall e \in \mathcal{E} : e \models \gamma$ then
8:                  if $\forall \beta \in \mathcal{B} : \beta \not\preceq_\theta \gamma$ then
9:                      $\mathcal{B} \leftarrow \mathcal{B} \cup \{\gamma\}$
10:                 end if
11:             else
12:                 if $\forall \delta \in \mathcal{A}_i : \gamma \not\approx_{iso} \delta$ then
13:                     $\mathcal{A}_i \leftarrow \mathcal{A}_i \cup \{\gamma\}$
14:                 end if
15:             end if
16:         end for
17:     end for
18: end for
19: return $\mathcal{B}$
```

It is worth pointing out that if we restrict the domain theories, e.g., to contain only clauses of length at most two or only Horn clauses, the saturation process will be guaranteed to run in polynomial time, which follows from the polynomial-time solvability of 2-SAT and Horn-SAT.

10 Note that for checking isomorphism within a set of clauses, we use the hash-based approach discussed in Section 3.1, not an exhaustive testing of all pairs.

5.5 INTEGRATING SATURATIONS WITH EXISTING ALGORITHMS

In this section, we describe two separate applications of saturation-based pruning within more or less straightforward algorithms used in the literature. We evaluate the effectiveness of our pruning method later in Section 5.6.

5.5.1 *Level-wise Feature Construction*

We consider the feature construction using a simple exhaustive level-wise algorithm, which works similarly to the Warmr frequent pattern-mining algorithm [33]. It takes three parameters by default: a set of examples $\mathcal{E}$, the maximum depth d, and the maximum number of covered examples t (also called "maximum frequency"). Due to the connection with saturation, it also takes domain theory $\mathcal{B}$. It returns all connected[11] clauses that can be obtained by saturating clauses containing at most d literals and which cover at most t examples. Unlike in frequent conjunctive pattern-mining where minimum frequency constraints are natural when mining in the setting of learning from interpretations, the analogue of the minimum frequency is the maximum frequency constraint.[12]

The level-wise algorithm, which is depicted in Algorithm 3, expects as input a list of interpretations (examples) $\mathcal{E}$, domain theory $\mathcal{B}$, and parameters t and d. It starts with an empty clause (line 1). Then it proceeds in the level-wise manner (lines 4-18) by extending clauses stored in the previous level in all possible ways by negative literals (line 5). Negative saturations (line 6) of those clauses are used for isomorphic pruning (lines 9-11).[13] Finally, only clauses with coverage at most t are stored for the next iteration (line 13). The algorithm ends after exceeding all d levels.

We restricted ourselves to mining clauses which contain only negative literals. This essentially corresponds to mining positive conjunctive queries, which is arguably the most typical scenario.

5.5.2 *Saturating Domain Theory Learner*

Inference algorithms, e.g., classical logical entailment or MLNs, often utilize hard constraints that are an essential part of a domain theory. During inference, hard constraints forbid the algorithms to predict things that apparently do not hold. For instance, recall Example 34 where domain theory encoded that *no one is a student and a professor at the same time*. We also need domain theory to mine features with our proposed saturation-based pruning. One could easily ask the following question aloud:

Can we use domain theory to learn domain theory?

Indeed, boosting the domain theory learner from Section 5.4 is relatively easy. The idea is to saturate i-th layer of the search, i.e., clauses with i literals, using domain theory learned up to that layer, i.e., $(i-1)$-th level. It is important

11 If a clause is connected, then its saturation is also connected.

12 Frequent conjunctive pattern mining can be emulated in our setting. It is enough to notice that the clauses that we construct are just negations of conjunctive patterns.

13 Again, this is just a pseudocode. We use the hash-based approach discussed in Section 3.1.

Algorithm 3 Pseudocode of Level-wise Feature Construction Algorithm

Parameter: Set of examples $\mathcal{E}$, domain theory $\mathcal{B}$, and integers t, d.

Output: Clauses Φ with at most d literals and covering at most t examples.

```
 1: $\mathcal{A}_0 \leftarrow \{\emptyset\}$
 2: $\mathfrak{L}^- \leftarrow$ all possible negative literals
 3: $\Phi \leftarrow \mathcal{A}_0$
 4: for $i \in \{1, 2, 3, \ldots, d\}$ do
 5:     $\mathcal{A}_i \leftarrow \bigcup_{\alpha \in \mathcal{A}_{i-1}} \{\alpha \cup \{\tau\} \mid \tau \in \mathfrak{L}^- \wedge \tau \notin \alpha\}$
 6:     $\mathcal{A}_i \leftarrow \{\sigma_{\mathcal{B}}(\alpha) \mid \alpha \in \mathcal{A}_i\}$
 7:     $\Gamma \leftarrow \emptyset$
 8:     for $\alpha \in \mathcal{A}_i$ do
 9:         if $\forall \gamma \in \Gamma : \gamma \not\approx_{iso} \alpha$ then
10:             $\Gamma \leftarrow \Gamma \cup \{\alpha\}$
11:         end if
12:     end for
13:     $\mathcal{A}_i \leftarrow \{\gamma \in \Gamma \mid |\{e \in \mathcal{E} \mid e \models \gamma\}| \leqslant t\}$
14:     if $\mathcal{A}_i$ is empty then
15:         break
16:     end if
17:     $\Phi \leftarrow \Phi \cup \mathcal{A}_i$
18: end for
19: return $\Phi$
```

to note that the algorithm still traverses the space of sentences rather than the space of saturations, which is an important difference compared to the previous feature-construction scenario. Otherwise, some sentences equivalent[14] to some domain theory clause could be expanded multiple times. We do not show here the whole pseudocode since it is a straightforward extension of Algorithm 2.

5.6 EXPERIMENTS

In this section, we evaluate the usefulness of the proposed saturation-based pruning method. The goal of our empirical evaluation is to answer the following question:

1. Does our proposed saturation-based pruning method make sentence mining faster than standard isomorphism-based pruning?

We evaluate this question on real datasets in two separate cases for which we already outlined algorithms in Section 5.5. The first case emulates a standard pattern-mining scenario, while the second uses the saturation-based pruning within the domain theory learner.

14 With respect to domain theory of the data.

5.6.1 *Feature Construction*

In this scenario, we follow the evaluation of the exhaustive level-wise feature construction algorithm described in Section 5.5.1. We measure runtime and the total number of clauses returned by the level-wise algorithm without saturations and with saturations. Both algorithms are exactly the same, the only difference being that the second algorithm first learns a domain theory, using the approach described in Section 5.4, and then uses it for computing the saturations. Note, in particular, that both algorithms use the same isomorphism filtering. Therefore, any differences in computation time must be directly due to the use of saturations.

The experiment was evaluated on a standard molecular dataset KM20L2 from the NCI GI 50 dataset collection [109]. This dataset contains 1207 examples (molecules) and 94263 facts; the representation uses 40 unary predicates to describe atoms, e.g., carbon or hydrogen, and 5 binary predicates to encode different bond types, e.g., double or aromatic bound. The learned domain theories were restricted to only clauses with at most 2 literals. We set the maximum number of covered examples equal to the number of examples in the dataset minus 1. Then, we also performed an experiment where we set it equal to the number of examples in the dataset minus 50. We set the maximum time limit to 10 hours.[15]

The results of the experiments are shown in Figure 3. The pruning method based on saturations turns out to pay off when searching for longer clauses where it improves the baseline by approximately an order of magnitude and allows it to search for longer clauses within the given time limit. When searching for smaller clauses, the runtime is dominated by the time for learning the domain theory, which is why the baseline algorithm is faster in that case. The number of generated clauses, which is directly proportional to memory consumption, also becomes orders of magnitude smaller when using saturations for longer clauses. Note that for every clause constructed by the baseline algorithm, there is an equivalent clause constructed by the algorithm with saturation-based pruning. The learned domain theory can be described in two terms: *bond symmetry* and *mutexes*. The former is something we already discussed in Example 31, e.g.

$$\forall x \forall y \; \mathsf{Bond}(x,y) \Rightarrow \mathsf{Bond}(y,x).$$

The latter encodes a variety of mutexes among types of atoms (predicates in general). For example, an atom (entity) cannot be a carbon and a hydrogen at the same time, i.e.

$$\forall x \; \neg\mathsf{Carb}(x) \vee \neg\mathsf{Hydro}(x).$$

Similar results, i.e., the effectiveness of the pruning method and learned domain theories, were also observed on the rest of the NCI molecular dataset, as they all describe molecules with very similar structures. Hence, we report only the result on KM20L2.

15 A side-note: a clause in this setup can be viewed as a negated conjunctive pattern. Thus, the first setup corresponds to a minimum frequency constraint of 1. Analogically, the second setup corresponds to a minimum frequency constraint of 50.

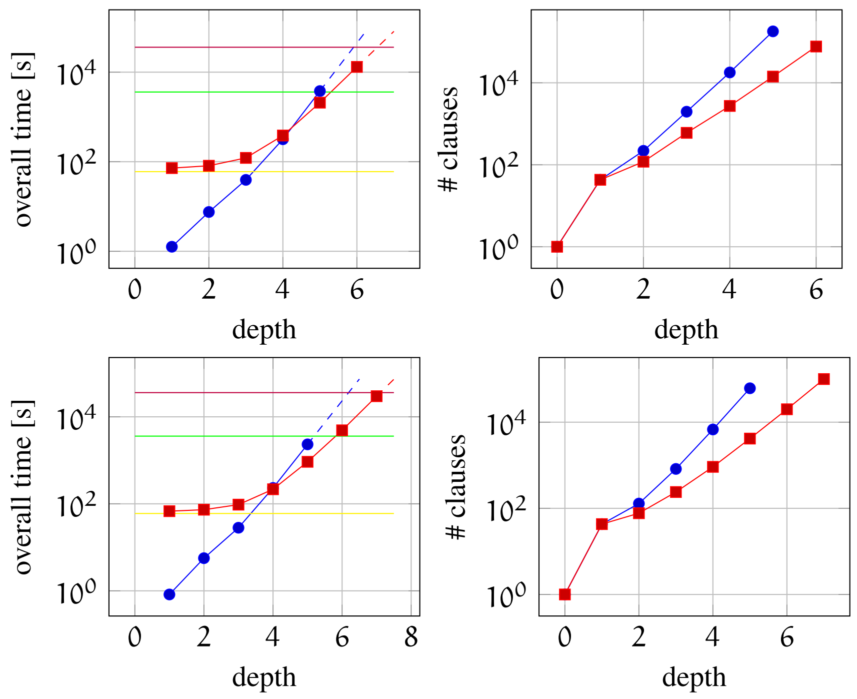

Figure 3: **Left:** Runtime of the level-wise algorithm using saturations for pruning (red) and without using saturations (blue). **Right:** Number of clauses constructed by the algorithm using saturations (red) and without using saturations (blue). **Top:** Results for maximal number of covered examples equal to dataset size minus 1 and **bottom** for this parameter set to dataset size minus 50, which corresponds to minimum frequency of 50. One minute, one hour, and 10 hours are highlighted by yellow, green, and purple horizontal lines, respectively. Runtimes are extrapolated by exponential function and shown in dashed lines.

5.6.2 *Boosting Domain Theory Learner*

In this scenario, we are interested in applying the saturation-based pruning to the domain theory mining algorithm described in Section 5.4 and extended in Section 5.5.2. Similar to the previous experiment, we execute the domain theory learning algorithm with the two pruning techniques – the only difference is that one setup uses saturation-based pruning while the other only uses isomorphic-based pruning. Hence, any difference in the performance comes directly from saturation-based pruning. Further, we restrict the constraints to be connected with up to 4 literals, with at most 1 positive literal,[16] and at most 15 variables. Again, we measure the runtime and the number of evaluated clauses.

Since all NCI datasets have similar underlying structures (bond symmetry and mutexes), we decided to use different datasets from different domains, numbers of predicates, etc. Therefore, we chose the standard SRL datasets and created four new ones. For the former, we selected UWCS [111], describing relations in a Department of Computer Science, Proteins [3], describing proteins and interactions between them, and IMDB [79], describing movie-related persons with relations to movies their appeared in or created. In ad-

16 This results in rules we can easily understand.

dition, we created several new datasets, which are more artificial than the already mentioned.

Firstly, we created two datasets from quasigroups and loops by generating 35 examples of the order from 5 to 40 for each structure. In detail, for order i, we generate i constants and describe only those relations between three constants that hold in the particular structure (e.g. $P/3$). Similarly, we encoded the first 100 Moufangs loops [86].

Secondly, we created a dataset composed of two consecutive steps from plans found by a planning algorithm on the Gripper domain, which is in the standard benchmarks of classical planning [128]. We used the PDDL notation for describing the world; therefore, we call this dataset PDDL-Grip. Recall that in classical planning, there is a domain, i.e., a set of first-order rules describing valid actions and facts in the world, and an instance, i.e., a particular description of the initial and the goal state. The planning algorithm task is to find a plan, e.g., a set of actions, leading from the initial to the goal state. Learning domain theory from this PDDL-Grip dataset actually corresponds to learning constraints in the domain file. We can check the validity of the learned domain theories on the newly created datasets because we know the underlying rules from which the examples were generated. This contrasts real-world datasets like UWCS and IMDB where we do not know the underlying structure.

Note that these datasets contain unary predicates, as well as binary, e.g., UWCS, and sometimes ternary predicates, e.g., PDDL-Grip.

The aggregated results, showing the number of evaluated clauses and the runtime from these runs for each dataset, are in Figure 4. Saturated-based pruning lowered the number of evaluated sentences in all datasets except the quasigroup one. Saturated-based pruning was faster on all datasets except for UWCS. This is caused by the fact that there are five hundred constraints of length two, which slows the computation of saturations. There was a tie on the IMDB dataset, but with such a short runtime of one minute compared to other datasets, we do not make any conclusions here.

This experiment showed another application of saturated-based pruning. The effectiveness, however, always depends on the underlying structure of the data.

5.6.3 *Conclusion*

We evaluated the proposed saturation-based pruning method in two sentence-mining cases and different application levels. Except for a single dataset, a method with saturation-based pruning outperformed its vanilla isomorphism-based pruning. These results clearly suggest the usefulness of this kind of pruning, which could be applied to various relational methods that use some first-order sentences.

5.7 RELATED WORK

The idea of utilizing some kind of knowledge for pattern mining has been tackled by many researchers [1, 4] with the utmost goal of i) constraint the search space to scale up the search process, ii) driving the search to *interest-*

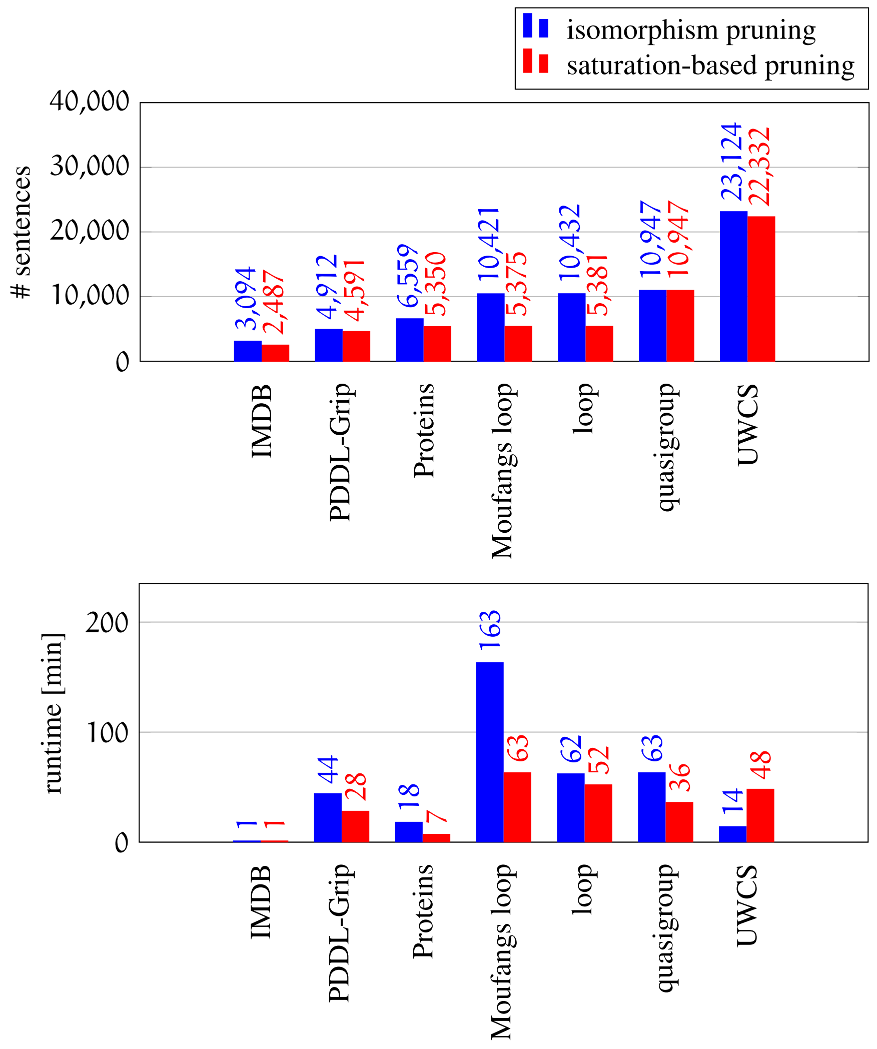

Figure 4: Number of evaluated sentences (top) and runtime in minutes (bottom) for domain theory learner with isomorphism (blue) and saturation-based pruning (red) on several datasets. Domain theories of the following form were sought: connected clauses with at most 4 literals, at most one 1 positive literal, and at most 15 variables.

ing discoveries, iii) or both at the same time. Naturally, the very essence of knowledge differs from paper to paper as some use regularities in the data (as we do), large ontologies, or even interaction with human experts [82, 92]. We will discuss only the first two cases.

However, most papers that inject domain knowledge into the rule-mining process consider only propositional (associative) rules. These methods often utilize domain knowledge in a pre-processing step [143], e.g., narrowing the number of attributes, or in a post-processing step to shrink large rule sets [11, 98]. Only a few approaches utilize domain knowledge, usually in the form of ontology or taxonomy of attributes, in the search process [69, 70, 142], as our approach does, or learn statistics from the data to drive the search [66]. Background knowledge was used to reduce the space of hypotheses in the Progol 4.4 system [84], which uses Plotkin's relative clause reduction. Note that the latter is a method for *removing* literals from bottom clauses, whereas, in contrast, our method is based on *adding* literals to clauses. Hence, the Progol 4.4 strategy is orthogonal to the methods presented in this chapter. Another key difference is that our approach is able to learn the domain knowledge from the training data, whereas all the other ILP approaches use predefined background knowledge.

Nonetheless, the works most related to our approach are those relying on a special case of Plotkin's relative subsumption [102] called generalized subsumption [18]. Generalized subsumption was among others used in [68]. However, we already discussed the reasons why relative subsumption is not suitable for pruning in Section 5.3.1.

The most common approach for learning first-order logical rules uses some heuristic search strategy with an ideal downward refinement operator [126] – our method is suited to accompany methods that follow this idea. However, to narrow the sentence space, these approaches usually deploy language biases [85] that are predefined or customizable by a user. For instance, rule-mining algorithms developed for RDF data, i.e., knowledge graphs, such as AMIE [39], typically employ *closed paths* language bias in which the variables in the head of a horn rule are chained through the body's literals. Even though most of these approaches [39, 76] learn only from the A-Box,[17] i.e., facts as we do, there are some exceptions that also utilize T-Box, i.e., RDF schema. Most notably, OP [21] and RARL [101] use the T-Box to learn rules that are then checked w.r.t. the A-Box; similarly, SWARM [7] exploits a single relationship from T-Box. However, SWARM mines semantically-enriched association rules rather than first-order logic rules. Contrary to these, our approach does not use predefined background knowledge but learns domain theory from the data. Nonetheless, it is important to note that we assume examples to be fully specified, i.e., closed world assumption, while RDF-based rule learning methods typically do not. In addition, a pruning technique was developed [139] for a very similar task we solved in Section 5.5.1 with the author's suggestion to develop a language bias based on regularities in data, which is very similar to our work.

Finally, our approach is not limited to definite clauses, which is also why we do not use SLD resolution. On the other hand, as our method is rooted in first-order logic (due to the fact that we use the learning from interpretations setting) and not directly in logic programming, it lacks some of the expressive power of logic programming.

5.8 FUTURE WORK

In the experiments, we showed two possible applications of the saturation-based pruning method. One more application is briefly discussed in Chapter 8. Hence, we will turn the discussion of future work to two possible extensions of this pruning method.

A natural case for extension of this method would be the propositionalization technique in ILP since it essentially mines features, i.e., conjunctive queries, from the data. While traversing the feature space, saturation-based pruning can utilize more than one domain theory to produce saturations. Hence, the algorithm would learn domain theories specific to different

17 This, among others, implies that the learned rule sets are typically quite huge and various aggregation techniques are being developed, e.g., [96].

example coverages. However, the exact strategy to connect this idea with a feature-mining algorithm is left for future work.

Computing of the saturations can be time-consuming since the presented approach is quite general. One of the drawbacks comes from the fact that we first generate all possible clause extensions and saturate all of these after that. Translating at least part of learned domain knowledge straightly into the refinement operator would be beneficial as it could save computational resources. A hot candidate for an easy *domain-to-refinement* translation would be the utilization of mutexes.

6 BOUNDED REASONING USING STRATIFIED k-ENTAILMENT

I would suggest "prioritised" entailment is a better fit. I do admit, "Prike" is not such a catchy name.

— Anonymous reviewer

Besides other applications, the pruning method proposed in the previous chapter can be used for learning Horn rules. In turn, these can be utilized to predict missing facts from a knowledge base. However, the most natural application of these rules using standard entailment may lead to the derivation of false positive facts. Consider the well-known example of smokers and friends, i.e.

$$\forall x \forall y \; \mathsf{Smokes}(x) \land \mathsf{Friends}(x, y) \Rightarrow \mathsf{Smokes}(y),$$

and a knowledge base of persons in which friendship connects each two people either directly or indirectly using a common friend of a friend, and so on. Finally, consider a single smoker in the knowledge base, i.e.

$$\mathcal{E} = \{\mathsf{Smokes}(\mathit{ann}), \mathsf{Friends}(\mathit{ann}, \mathit{bob}), \mathsf{Friends}(\mathit{bob}, \mathit{christin}), \\ \mathsf{Friends}(\mathit{christin}, \mathit{danny}), \dots\}.$$

In such case, the rule is quite accurate, since the implication is violated only for the grounding

$$\mathsf{Smokes}(\mathit{ann}) \land \mathsf{Friends}(\mathit{ann}, \mathit{bob}) \Rightarrow \mathsf{Smokes}(\mathit{bob}).$$

Unsurprisingly, applying this rule leads to marking every person in the knowledge base as a smoker.

To deal with this issue, we follow, in this chapter, the approach from [53] called k-entailment. The method was proposed to overcome the issue by breaking down long inference chains containing dozens of constants, which is done by relaxing the standard entailment and allowing it to reason only over a subset of constants at a time.

However, the method was studied only from a theoretical point of view, i.e., deriving PAC-style[1] guarantees on the number of incorrect derived facts, with no focus at all on practical inference algorithms or evaluation. Hence, in this chapter, we propose an inference algorithm as well as an extension of the method to our rule-based setting. We also propose a heuristic rule learner that suits the proposed inference method. In this chapter, we are working under the learning setting described in Section 2.3.2, i.e., having a fully described training example $\mathcal{E}$ and an incomplete test example Υ.

1 Probably approximately correct (PAC).

6.1 k-ENTAILMENT

We start with a description of k-entailment, which was introduced in [53]. Intuitively, a fact can be k-entailed from a given rule base if it has a derivation involving at most k constants. This captures the intuition that, in most domains, the knowledge about one specific constant typically only directly affects what we can derive about a small set of closely related constants. For instance, when modelling a social network, using k-entailment means that the knowledge we have about a given user only directly affects what we derive about the users in their neighborhood. By using 2-entailment, we thus restrict the impact of any errors in our knowledge about some user to his or her direct friends. Similarly, using 3-entailment would mean that our knowledge about the friends of a user's friends might also be affected, but not the wider network. In contrast, when using classical logic, a single error might, in principle, affect the entire network. Formally, k-entailment is defined as follows.

Definition 19 (k-entailment)**.** *Let* k *be a non-negative integer,* $\mathcal{E}$ *a set of facts,* Φ *a set of rules and* Γ *a set of constraints. We say that a fact* υ *is* k*-entailed by* $\mathcal{E} \cup \Phi \cup \Gamma$*, denoted*

$$\mathcal{E} \cup \Phi \cup \Gamma \models_k \upsilon$$

if there is some $\mathcal{C} \subseteq const(\mathcal{E} \cup \Phi \cup \Gamma)$ *such that:*

1. $|\mathcal{C}| \leqslant k$
2. $(\mathcal{E} \cup \Phi \cup \Gamma)[\mathcal{C}]$ *is consistent*
3. $(\mathcal{E} \cup \Phi)[\mathcal{C}] \models \upsilon$

Note that, apart from limiting the impact that a given piece of knowledge can have, in the presence of constraints, k-entailment also blocks all inferences involving sets of constants $\mathcal{C}$ that make $(\mathcal{E} \cup \Phi \cup \Gamma)[\mathcal{C}]$ inconsistent. This is illustrated in the next example.

Example 35. *Let us consider the following:*

$$\begin{aligned}
\Phi &= \{\forall x\ \mathsf{Giraffe}(x) \Rightarrow \mathsf{Animal}(x),\\
&\qquad \forall x \forall y\ \mathsf{Friends}(x,y) \Rightarrow \mathsf{Friends}(y,x),\\
&\qquad \forall x \forall y\ \mathsf{Friends}(x,y) \Rightarrow \mathsf{Human}(x)\}\\
\Gamma &= \{\forall x\ \neg\mathsf{Human}(x) \vee \neg\mathsf{Animal}(x)\}\\
\mathcal{E} &= \{\mathsf{Giraffe}(liz), \mathsf{Friends}(ann, liz)\}.
\end{aligned}$$

Then $\mathsf{Animal}(liz)$ *is the only fact which is* 2*-entailed by* $\mathcal{E} \cup \Phi \cup \Gamma$*, besides those in* $\mathcal{E}$*. In particular, note that we can infer neither* $\mathsf{Human}(liz)$ *nor* $\mathsf{Human}(ann)$ *because* $(\mathcal{E} \cup \Phi \cup \Gamma)[\{ann, liz\}]$ *is not logically consistent. Intuitively, the constraint in* Γ *and rules in* Φ *assert that animals do not have friends and that one cannot be an animal and a human at the same time. This means that, assuming the rules were perfect,* $\mathsf{Giraffe}(liz)$ *and* $\mathsf{Friends}(ann, liz)$ *cannot both be correct. Since we do not want the potential errors to spread through our knowledge base, by using* 2*-entailment we avoid inferences that involve ann and liz at the same time.*

In Example 35, we observed what type of theory, i.e., set of rules, is used in k-entailment – definite rules and constraints. The distinction between Φ and Γ is that the former contains (most likely) imperfect rules that infer something. Contrary, Γ encodes constraints that hold in the data, for instance, mutual-exclusivity.[2] These may cause inconsistency during the inference procedure, affecting the whole knowledge base when we apply standard entailment.

6.1.1 *Properties of* k*-entailment*

In [53], authors derive PAC-style guarantees for k-entailment in relational domains. First, we define the *accuracy* of a set of rules and constraints $\Phi \cup \Gamma$, inspired by [53], in order to show these guarantees in our setting. Note that for a set of k constants $\mathcal{C}'$, we can think of $\mathcal{E}[\mathcal{C}']$ as a possible world involving these k constants. The *accuracy* of a set of clauses is then simply the percentage of these possible worlds in which the clauses are satisfied. This is formalized in the following definition.

Definition 20 (Accuracy)**.** *Let $\mathcal{E}$ be a set of facts, Φ be a set of rules and Γ be a set of constraints. Let $\mathcal{C} = const(\mathcal{E} \cup \Phi \cup \Gamma)$ and $[\mathcal{C}]^k$ be the set of all size-k subsets of $\mathcal{C}$. Then, for a given positive integer k, the accuracy of $\Phi \cup \Gamma$ on $\mathcal{E}$ is defined as:*

$$\mathbf{Acc}_{\mathcal{E},k}(\Phi \cup \Gamma) = \frac{1}{|[\mathcal{C}]^k|} \sum_{\mathcal{C}' \in [\mathcal{C}]^k} \mathbb{1}(\mathcal{E}[\mathcal{C}'] \models (\Phi \cup \Gamma)[\mathcal{C}'])$$

The next proposition links the *accuracy* of a set of rules and constraints $\Phi \cup \Gamma$ to the number of errors produced by k-entailment; it is a direct counterpart of Proposition 6 from [53].

Proposition 4. *Let $\mathcal{E}'$ and $\mathcal{E}$ be sets of facts such that $\mathcal{E}' \subseteq \mathcal{E}$, Φ be a set of rules and Γ a set of constraints. Furthermore let $\mathcal{F}$ be the set of all positive ground literals of an a-arity predicate P/a, $a \leqslant k$, which are k-entailed by $\mathcal{E}' \cup \Phi \cup \Gamma$ but are not contained in $\mathcal{E}$. Then*

$$|\mathcal{F}| \leqslant (1 - \mathbf{Acc}_{\mathcal{E},k}(\Phi \cup \Gamma))\, |\mathcal{C}|^k k^a$$

where

$$\mathcal{C} = const(\mathcal{E}' \cup \Phi \cup \Gamma).$$

Note that the bound grows with the size of $\mathcal{C}$. In particular, even for a fixed value of $\mathbf{Acc}_{\mathcal{E},k}(\Phi \cup \Gamma)$, the bound grows with the size of the test data.

To see the significance of the above proposition, let $\mathcal{E}'$ represent the set of facts that we are given as evidence and let $\mathcal{E}$ be the set of all facts. The proposition bounds the number of incorrect facts that are derived when using k-entailment. Note, however, that this bound depends on $\mathbf{Acc}_{\mathcal{E},k}(\Phi \cup \Gamma)$, whereas in practice $\mathcal{E}$ is not known. This characterization is useful because we are typically able to estimate $\mathbf{Acc}_{\mathcal{E},k}(\Phi \cup \Gamma)$ sufficiently precisely from the given training data. This is formalized in the next proposition, which is an adaptation of Theorem 10 from [53] to our setting.

2 This is in contrast to standard definite logic programs where constraints only determine whether a solution exists or not without any applying to inference or solutions' filtering.

Proposition 5. *Let $\mathcal{E}$ be a set of facts and let $0 \leqslant n \leqslant |const(\mathcal{E})|$ and $0 \leqslant k \leqslant n$ be integers. Let $\mathcal{C}$ be sampled uniformly from $[const(\mathcal{E})]^n$ and let $\mathcal{E}' = \mathcal{E}[\mathcal{C}]$. Then we have*

$$P\left[|\mathbf{Acc}_{\mathcal{E}',k}(\Phi \cup \Gamma) - \mathbf{Acc}_{\mathcal{E},k}(\Phi \cup \Gamma)| \geqslant \varepsilon\right] \leqslant 2\exp\left(-2\left\lfloor \frac{n}{k} \right\rfloor \varepsilon^2\right).$$

One can combine Proposition 4 and Proposition 5 to get various learning guarantees.[3]

The significance of the insights from [53] for this chapter is that they justify why our rule learner is able to evaluate candidate rules based on how accurate they are on small fragments of the training data (assuming that this training data was sampled from a reasonable distribution). Our rule learning approach in Section 6.5 will take advantage of this insight.

Furthermore, unlike entailment from classical logic, k-entailment is non-monotonic. We will discuss this property more in Section 6.4.

6.2 INFERENCE ALGORITHM

Naively following the definition of k-entailment would require enumerating all subsets with at most k constants in the given domain $\mathcal{C}$ and determining which positive literals can be entailed from each of them. This section shows how to implement k-entailment in a much more efficient way. In particular, we present an approach based on a forward-chaining procedure with additional book-keeping of set-wise minimal sets of constants. Specifically, the algorithm solves the following problem:

Given: *A set of range-restricted definite rules $\Phi = \{\alpha_1, \dots, \alpha_l\}$, constraints $\Gamma = \{\beta_1, \dots, \beta_{l'}\}$ and facts $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$.*

Compute: *The set of positive literals k-entailed by $\Phi \cup \Gamma \cup \mathcal{E}$.*

We assume w.l.o.g. that Φ does not contain any facts, noting that including a fact α in Φ is equivalent to including α in $\mathcal{E}$.

6.2.1 *Description of the Algorithm*

The algorithm begins by initializing auxiliary variables (lines 1-6) and a hash table *Support*, which associates literals (keys) to sets of subsets of constants (values). When the algorithm terminates, *Support*$[\upsilon]$ will contain all minimal sets of constants $\mathcal{C}$, up to size k, for which $\mathcal{E}[\mathcal{C}] \cup \Phi \cup \Gamma \models \upsilon$. The set of k-entailed literals will thus correspond to the key set of *Support*, which we denote by ***Keys***(*Support*). To find these sets $\mathcal{C}$, the algorithm alternatingly adds new candidate subsets of constants and then filters those which turn out to be inconsistent. To detect inconsistencies efficiently, the algorithm maintains the set *Incons* which is initially empty. Throughout the execution of the algorithm, it is used to store all encountered subsets I of $\mathcal{C}$ where $\mathcal{E}[I] \cup \Phi \cup \Gamma$ is found to be logically inconsistent.

The main loop (lines 7-28) of the algorithm starts with solving the following sub-problem: given a set of ground positive literals $\mathcal{E}$ and a rule $\alpha = (b_1 \wedge \dots \wedge b_l \Rightarrow a)$ or a constraint $\beta = \neg(b_1 \wedge \dots \wedge b_l)$, find the set of

3 Further details on confidence bounds that one can obtain in this setting for accuracy can be found in [53].

all groundings θ such that $(b_1 \wedge \dots \wedge b_l)\theta \subseteq \mathcal{E}$. We will denote the corresponding set of ground rules and constraints by ***Active***$(\alpha, \mathcal{E})$. In other words, this set contains those ground rules and constraints whose body is satisfied by the literals that we have derived so far (line 8). It can be efficiently computed using a conjunctive query in a relational database, similar to how such relational database engines are used in Markov logic inference [89, 112].

Next, the algorithm removes each active rule α for which there is some $I \in$ *Incons* such that $I \subseteq$ *const*(α) (line 9). The rules that are removed from $\mathcal{A}$ in this step correspond to those whose body can only be satisfied by starting from a fragment of $\mathcal{E}$ that is inconsistent with $\Phi \cup \Gamma$ or – more precisely – those which have previously been found to be inconsistent.

In the next step, the algorithm iterates over the remaining active rules (lines 10-23). For each such rule $(b_1 \wedge \dots \wedge b_m) \Rightarrow h \in \mathcal{A}$ it combines the support sets of the ground literals $b_1, \dots, b_m$ and stores them in a set $\mathcal{U}$. In particular, if we have $S_1 \in$ *Support*$[b_1]$, this intuitively means that we can derive b_1 using a fragment of $\mathcal{E}$ that only involves the constants from S_1, and similar for $b_2, \dots, b_m$. Hence, thanks to the given rule, this means that we can also derive its head h using a fragment that contains all constants from $S = S_1 \cup \dots \cup S_m$. Of these sets S, we only keep those which have at most k elements (line 13) and which are minimal w.r.t. set inclusion (line 14). Next, the procedure ***FindInconsistent*** aims to detect subsets of constants $\mathcal{C}$ that occur in *Support*$[h']$, for some literal h', such that $\mathcal{E}[\mathcal{C}] \cup \Phi \cup \Gamma$ is inconsistent. To do this efficiently, we take advantage of the fact that each iteration of the main loop corresponds to an iteration of a modified forward chaining procedure. Rather than checking whether $\mathcal{E}[\mathcal{C}] \cup \Phi \cup \Gamma$ is inconsistent, we thus check whether such an inconsistency has been derived so far. This is done in two steps, which are not shown in the pseudocode. First, we determine all the sets of constants $\mathcal{C}$ whose consistency needs to be (re-)checked. This is the case for all supersets of the elements from $\mathcal{U}$ (including the elements from $\mathcal{U}$ themselves). For the second step, let us write *Support*$^{-1}[S]$ for the set of all literals h' for which *Support*$[h']$ contains a subset of S. Note that *Support*$^{-1}[S]$ intuitively corresponds to the set of literals for which our forward chaining procedure has already established that they can be derived from $\mathcal{E}[S] \cup \Phi \cup \Gamma$. For every superset S of an element from $\mathcal{U}$, the procedure checks if *Support*$^{-1}[S]$ is consistent with the constraints in Γ; it returns all sets S for which this is not the case. Finally, these sets of constants are removed from *Support* (line 17). If the set of support sets for some literal h' in ***Keys***(*Support*) then becomes empty, this literal is removed from the key set of the hash table (line 19). Finally, all support sets found as inconsistent are added to the set *Incons* (line 22).

Next, the algorithm first updates the current state of the evidence $\mathcal{E}$ to contain exactly the facts which occur as keys in the hash table *Support* (line 25) and then it either repeats the main loop from line 7 if *Support* has been modified in the current iteration, or it finishes and returns derived facts $\mathcal{E}$ together with the initial evidence $\mathcal{E}_0$ (line 29).

Next, we describe an illustrating example. Thereafter, we give a sketch of correctness.

Algorithm 4 Pseudocode of k-Entailment Inference Algorithm

Parameter: Set of range-restricted rules Φ, constraints Γ, evidence $\mathcal{E}$ and the integer k.

Output: All positive facts that are k-entailed by $\Phi \cup \Gamma \cup \mathcal{E}$.

```
1: $\mathcal{E}_0 \leftarrow \mathcal{E}$
2: $Support \leftarrow \emptyset$
3: $Incons \leftarrow \emptyset$
4: for $v \in \mathcal{E}$ do
5:     $Support[v] \leftarrow const(v)$
6: end for
7: while $\mathtt{true}$ do
8:     $\mathcal{A} \leftarrow \cup_{\alpha \in \Phi} \boldsymbol{Active}(\alpha, \mathcal{E})$
9:     $\mathcal{A} \leftarrow \{\alpha \in \mathcal{A} \mid \forall I \in Incons : I \not\subseteq const(\alpha)\}$
10:    for $(b_1 \wedge \cdots \wedge b_n \Rightarrow h) \in \mathcal{A}$ do
11:        $\mathcal{S}_{Cart} \leftarrow Support[b_1] \times \cdots \times Support[b_m]$
12:        $\mathcal{U} \leftarrow \{S_1 \cup \cdots \cup S_m \mid (S_1, \ldots, S_m) \in \mathcal{S}_{Cart}\}$
13:        $Support[h] \leftarrow Support[h] \cup \{S \in \mathcal{U} \mid |S| \leqslant k\}$
14:        $Support[h] \leftarrow \{\mathcal{S}_1 \in Support[h] \mid \forall \mathcal{S}_2 \in Support[h] : \mathcal{S}_2 \not\subset \mathcal{S}_1\}$
15:        $\mathcal{I} \leftarrow \boldsymbol{FindInconsistent}(h, Support, \Gamma)$
16:        for $h \in$ Keys$(Support)$ do
17:            $Support[h] \leftarrow Support[h] \setminus \mathcal{I}$
18:            if $Support[h] = \emptyset$ then
19:                remove $h$ from $Support$
20:            end if
21:        end for
22:        $Incons \leftarrow Incons \cup \mathcal{I}$
23:    end for
24:    $\mathcal{E} \leftarrow \boldsymbol{Keys}(Support)$
25:    if If $Support$ was not changed in the last iteration then
26:        break
27:    end if
28: end while
29: return $\mathcal{E} \cup \mathcal{E}_0$
```

6.2.2 *An Illustration*

Here, we exemplify one run of the algorithm. Let continue with Φ, Γ and $\mathcal{E}$ from Example 35, i.e.

$$\begin{aligned}
\Phi = \{&\forall x\ \mathsf{Giraffe}(x) \Rightarrow \mathsf{Animal}(x),\\
&\forall x \forall y\ \mathsf{Friends}(x,y) \Rightarrow \mathsf{Friends}(y,x),\\
&\forall x \forall y\ \mathsf{Friends}(x,y) \Rightarrow \mathsf{Human}(x)\},\\
\Gamma = \{&\forall x\ \neg \mathsf{Human}(x) \vee \neg \mathsf{Animal}(x)\},\\
\mathcal{E} = \{&\mathsf{Giraffe}(\mathit{liz}), \mathsf{Friends}(\mathit{ann}, \mathit{liz})\},\\
k =\ & 3.
\end{aligned}$$

which talks about the giraffe *liz* and her friend *ann*. Running the initialization procedure will result in

$$\begin{aligned} \mathit{Support} = \{&\mathsf{Giraffe}(\mathit{liz}) : \{\{\mathit{liz}\}\}, \\ &\mathsf{Friends}(\mathit{ann}, \mathit{liz}) : \{\{\mathit{ann}, \mathit{liz}\}\}\}. \end{aligned}$$

Evaluating line 8, we find the set $\mathcal{A}$, which is the set of ground rules whose bodies are true in the current $\mathcal{A}$:

$$\begin{aligned} \mathcal{A} = \{&\mathsf{Giraffe}(\mathit{liz}) \Rightarrow \mathsf{Animal}(\mathit{liz}), \\ &\mathsf{Friends}(\mathit{ann}, \mathit{liz}) \Rightarrow \mathsf{Friends}(\mathit{liz}, \mathit{ann})\}, \\ &\mathsf{Friends}(\mathit{ann}, \mathit{liz}) \Rightarrow \mathsf{Human}(\mathit{ann})\}. \end{aligned}$$

After the inner loop on lines 10-23 is executed, the state of the hash table becomes

$$\begin{aligned} \mathit{Support} = \{&\mathsf{Giraffe}(\mathit{liz}) : \{\{\mathit{liz}\}\}, \\ &\mathsf{Friends}(\mathit{ann}, \mathit{liz}) : \{\{\mathit{ann}, \mathit{liz}\}\}, \\ &\mathsf{Animal}(\mathit{liz}) : \{\{\mathit{liz}\}\}, \\ &\mathsf{Friends}(\mathit{liz}, \mathit{ann}) : \{\{\mathit{ann}, \mathit{liz}\}\}, \\ &\mathsf{Human}(\mathit{ann}) : \{\{\mathit{ann}, \mathit{liz}\}\}\}. \end{aligned}$$

Next, in the procedure ***FindInconsistent***, none of the support sets will be found inconsistent in this iteration because there is no inconsistency with the constraints Γ. The algorithm sets $\mathcal{E} = \textbf{\textit{Keys}}(\mathit{Support})$ and continues from line 7.

In the second iteration, the algorithm then adds $\mathsf{Human}(\mathit{liz}) : \{\{\mathit{ann}, \mathit{liz}\}\}$ to *Support*. Next, on line 15, the algorithm finds the support set $\{\mathit{ann}, \mathit{liz}\}$ inconsistent. After the filtering step, the state of the hash table *Support* becomes

$$\begin{aligned} \mathit{Support} = \{&\mathsf{Animal}(\mathit{liz}) : \{\{\mathit{liz}\}\}, \\ &\mathsf{Giraffe}(\mathit{liz}) : \{\{\mathit{liz}\}\}\}. \end{aligned}$$

In the third (and in this case final) iteration, the state of the hash table *Support* remains the same and the algorithm finishes, returning the set

$$\{\mathsf{Animal}(\mathit{liz}), \mathsf{Giraffe}(\mathit{liz}), \mathsf{Friends}(\mathit{ann}, \mathit{liz})\}.$$

Note that the fact $\mathsf{Friends}(\mathit{ann}, \mathit{liz})$ is added from the original evidence set.

6.3 SKETCH OF CORRECTNESS AND RUNTIME

If we had $\Gamma = \emptyset$, then the correctness of the algorithm would follow almost immediately from the same arguments that show the correctness of the forward chaining procedure for classical logical reasoning. In general, when $\Gamma \neq \emptyset$, we can proceed as follows. It is easier to analyze a version of the algorithm without the filtering of non-minimal support sets. We can check that omitting this step would not affect the correctness of the algorithm and that the number of iterations of this modified algorithm would not be lower than that of the full algorithm. Hence, we will analyze this simpler algorithm below.

Termination: Let A be the maximum arity among the relations from $\mathcal{P}$. Let us define

$$N_i = |\mathcal{P}| \cdot k^{A+1} \cdot |\mathit{Incons}_i| + \sum_{\upsilon \in \mathbf{Keys}(\mathit{Support}_i)} |\mathit{Support}_i[\upsilon]|$$

where Incons_i and $\mathit{Support}_i$ are states of the respective data structures at the beginning of the i-th iteration of the main loop. Then, for all $i > 1$ it holds $N_{i-1} < N_i$; in particular, this is true because we ignore filtering of non-minimal support sets. Since the set $\mathcal{C}$ is finite, there may be only a finite number of iterations of the algorithm, in particular at most

$$|\mathcal{P}| \cdot k^{A+1} \cdot |\mathcal{C}|^k.$$

This also gives us a polynomial bound on the number of iterations of the main loop of the algorithm.

Soundness and Completeness: Showing soundness and completeness for our inference algorithm in a completely rigorous way would be tedious and not really illuminating. Thus, we only provide a brief justification. Soundness is easy to check. In particular, whenever a fact υ is derived using the inference algorithm, there must be a classical proof of it on a fragment of evidence, given as input, with at most k constants (this can be seen easily by inspection of the algorithm). Now, it could still be the case that υ is derived from a fragment that is inconsistent with $\Phi \cup \Gamma$. However, if that was the case, this fragment would have been detected by the procedure ***FindInconsistent*** and it would have been removed together with all its occurrences among support sets stored in the hash table *Support*.

Completeness is not difficult to check either. If there is a fragment of $\mathcal{E}$ that is consistent with $\Phi \cup \Gamma$ and a fact υ can be derived from it, then it can also be derived from it using forward chaining (this follows from the same result for classical logic). Since the fragment is consistent, neither the fragment itself nor any of its subsets can be present in the set *Incons* at any time. It follows that the fact υ must be derived by the algorithm.

6.4 STRATIFIED k-ENTAILMENT

Unlike classical logic, k-entailment is not monotonic. In particular,

$$\mathcal{E} \cup \Phi \cup \Gamma \models_k \alpha$$

does not imply that

$$\mathcal{E} \cup \mathcal{E}' \cup \Phi \cup \Gamma \models_k \alpha,$$

for $\mathcal{E}$, $\mathcal{E}'$ sets of ground literals, Φ a set of rules and Γ a set of constraints. This is illustrated in the next well-known example from the non-monotonic reasoning literature [121].

Example 36. *Consider the following rules and constraints*

$$\begin{aligned}\Phi = \{&\forall x\ \mathsf{Bird}(x) \Rightarrow \mathsf{Flies}(x),\\ &\forall x\ \mathsf{Penguin}(x) \Rightarrow \mathsf{Bird}(x)\},\\ \Gamma = \{&\forall x\ \neg\mathsf{Penguin}(x) \vee \neg\mathsf{Flies}(x)\}.\end{aligned}$$

For $\mathcal{E} = \{\mathtt{Bird}(\textit{tweety})\}$, we can derive $\mathtt{Flies}(\textit{tweety})$ using k*-entailment. However, when we additionally know that Tweety is a penguin, i.e. if our evidence is given by*

$$\mathcal{E}' = \{\mathtt{Bird}(\textit{tweety}), \mathtt{Penguin}(\textit{tweety})\},$$

we can no longer derive $\mathtt{Flies}(\textit{tweety})$ using k*-entailment because*

$$(\Phi \cup \Gamma \cup \mathcal{E}')[\{\textit{tweety}\}]$$

is inconsistent.

This non-monotonic behavior is in itself not problematic. Indeed, it is standard practice in artificial intelligence to use non-monotonic reasoning when dealing with rules that may have exceptions, and probabilistic reasoning is also non-monotonic. However, the behavior of k-entailment in the presence of conflicts is arguably too cautious – most frameworks for non-monotonic reasoning would still derive $\mathtt{Bird}(\textit{tweety})$ from $\mathtt{Penguin}(\textit{tweety})$ in the previous example. This can be achieved by taking into account that the rule

$$\forall x\ \mathtt{Penguin}(x) \Rightarrow \mathtt{Bird}(x)$$

is more reliable than the other rule, either by inducing an ordering on the set of rules automatically [41, 99] or by relying on an explicitly given ordering of these rules [10].

We will follow the latter strategy to obtain a refinement of k-entailment, which we call *stratified* k-entailment or *STRiKE*.

Definition 21 (Stratified k-Entailment)**.** *Let $\Lambda = (\alpha_1, \ldots, \alpha_m)$ be a list of rules and constraints and let $\mathcal{E}$ be a set of facts. We say that a fact υ is* k*-entailed at level i from Λ and $\mathcal{E}$ if there exists some $j \leqslant i$ s.t. $\{\alpha_1, \ldots, \alpha_j\} \cup \mathcal{E} \models_k \upsilon$.*

The general intuition is that the rules and constraints in Λ are ordered based on how confident we are in them, i.e., α_1 is the most confident rule or constraint.

The use of stratified k-entailment serves two purposes. First, ordering the rules and constraints permits ordering the predictions made by stratified k-entailment according to how confident we are in them. This brings k-entailment closer to approaches such as AMIE and MLNs, which also provide confidence values for the predicted facts. In applications, this allows us to tune the trade-off between precision and recall. Second, by taking the ordering of the rules and constraint into account, we can avoid the situation from Example 36, where a less reliable rule was blocking the conclusion of a more reliable rule. This is illustrated in the next example.

Example 37. *Consider the following list:*

$$\begin{aligned}\Lambda = (&\forall x\ \mathtt{Penguin}(x) \Rightarrow \mathtt{Bird}(x),\\ &\forall x\ \neg\mathtt{Penguin}(x) \vee \neg\mathtt{Flies}(x),\\ &\forall x\ \mathtt{Bird}(x) \Rightarrow \mathtt{Flies}(x))\end{aligned}$$

and let $\mathcal{E} = \{\mathtt{Penguin}(\textit{tweety})\}$. Then, one can check that $\mathtt{Bird}(\textit{tweety})$ is k*-entailed at level 1 whereas with standard* k*-entailment we could not derive $\mathtt{Bird}(\textit{tweety})$ at all.*

Note that the inference mechanism in the previous example is similar in spirit to the one from possibilistic logic [35]. However, the ordering of formulas in possibilistic logic plays a different role than in stratified k-entailment – it is used in possibilistic logic to avoid entailment becoming trivial in the face of inconsistencies, but in our setting, the use of k-entailment already prevents entailment from becoming trivial. Furthermore, standard possibilistic logic only considers propositional formulas. The set of entailed facts in possibilistic logic is also unordered when the standard approach to inconsistency handling is used. Specifically, a formula is entailed from a possibilistic logic knowledge base if and only if it can be classically entailed from the set of formulas above the so-called inconsistency level, i.e., the first level i such that $\{\alpha_1, ..., \alpha_i\}$ is inconsistent.

Importantly, the PAC-type guarantees that follow from Proposition 4 and Proposition 5 also hold for stratified k-entailment. This follows easily from the fact that they must hold for every $\Phi_i = \{\alpha_1, \dots, \alpha_i\}$ where $i \leqslant m$. Obviously, it holds that

$$\mathbf{Acc}_{\mathcal{E},k}(\Phi_i) \leqslant \mathbf{Acc}_{\mathcal{E},k}(\Phi_j)$$

when $j \leqslant i$. Using the union bound, e.g. as Proposition 2 in [48], the accuracy of Φ on $\mathcal{E}$ can be bounded using the accuracies of the individual rules as

$$\mathbf{Acc}_{\mathcal{E},k}(\Phi_i) \geqslant 1 - \sum_{l=1}^{i} (1 - \mathbf{Acc}_{\mathcal{E},k}(\alpha_i)).$$

Hence, as long as the rules are accurate, we can obtain guarantees on the number of errors that stratified k-entailment will make.

Finally, the extension of the algorithm from Section 6.2 to the stratified case of k-entailment is as easy as executing the algorithm with a gradually increasing number of rules, i.e., $\{\alpha_1, \dots, \alpha_i\}$.

6.5 A HEURISTIC RULE LEARNER

We introduce a heuristic algorithm for learning rules that are suitable for reasoning with stratified k-entailment. At a high level, the algorithm performs a top-down beam search through the space of definite rules, using a refinement operator [85] that adds one literal at a time to the rules in the beam. This is a standard strategy in relational learning and inductive logic programming systems [85, 106]. The algorithm's key novelty is how it heuristically scores the candidate rules.

Definition 22 (Precision of a rule). *Let $\alpha = b_1 \wedge \dots \wedge b_m \Rightarrow h$ be a rule and let $\mathcal{E}$ a set of facts. We define* precision *of rule α as*

$$\psi_{\wedge}(\alpha) = \frac{|H_\alpha \cap \mathcal{E}|}{|H_\alpha|}$$

where

$$H_\alpha = \{h\theta \mid \mathcal{E} \models (b_1 \wedge \dots \wedge b_m)\theta,\ \theta \in G_{\mathcal{E}}(\alpha)\}.$$

In other words, $\psi_{\wedge}(\alpha)$ corresponds to the percentage of facts predicted by the rule α that we know to be true.

We then create a sorted list of rules Λ in descending order of their precision, i.e.

$$\Lambda = (\alpha_1, \dots, \alpha_l)$$

where

$$\psi_\Lambda(\alpha_i) \geqslant \psi_\Lambda(\alpha_{i+1}).$$

Λ defines a probabilistic model that assigns to any fact υ the probability

$$p(\upsilon) = \max\{\psi_\Lambda(\alpha_i) \,|\, 1 \leqslant i \leqslant l, \upsilon \in H_{\alpha_i}\}.$$

Thus, this simple model assumes that the probabilities of the facts are independent. However, we recall that this is still just a heuristic for selecting the rules, not the final model for prediction. The advantage of this probabilistic view is that it naturally allows us to learn rules by selecting those that most improve the log-likelihood of the training data, i.e.

$$\sum_{\upsilon \in \mathcal{E}} \log p(\upsilon) + \sum_{\upsilon \in \overline{\mathcal{E}}} \log(1 - p(\upsilon)).$$

Our beam-search selects rules based on optimizing this score. An essential advantage of this approach is that the values ψ (precision) are computed based on the whole dataset rather than on what remains after removing the covered subsets of the dataset. This should improve the robustness of the algorithm.

This strategy contrasts with the standard covering strategy used in inductive logic programming [84], which is brittle and has problems with imbalanced data. Our rule learning strategy exploits the properties of *stratified* k-entailment, especially its ability to predict facts with different levels of certainty. In the classical inductive logic programming setting, one would typically optimize accuracy (or a closely related measure) and would, therefore, often only end up with high-precision rules. In contrast, since stratified k-entailment ranks the predicted facts by confidence, it can also work well with less precise rules (although the usefulness of such lower-confidence predictions clearly depends on the application).

In order to force the rule learning algorithm to discover non-trivial relationships, we give it subsampled data. Consider, for instance, that the training complete dataset contains facts of the form $\mathsf{Friends}(x, y)$. Then it would be difficult to learn any rules beyond symmetry, i.e.

$$\forall x \forall y \; \mathsf{Friends}(x, y) \Rightarrow \mathsf{Friends}(y, x),$$

as no other rule for predicting friendship would improve the log-likelihood. However, by subsampling the data we can break these symmetries, which means that other rules may be found that improve the log-likelihood. For instance, assume that the fact $\mathsf{Friends}(\mathit{christin}, \mathit{anna})$ is missing from the subsampled dataset but that these facts

$$\{\mathsf{Friends}(\mathit{anna}, \mathit{bob}),$$
$$\mathsf{Friends}(\mathit{bob}, \mathit{christin}),$$
$$\mathsf{Friends}(\mathit{ann}, \mathit{christin})\}$$

are present in $\mathcal{E}$. Then, the rule learner might add the rule

$$\forall x \forall y \forall z \; \mathsf{Friends}(x, y) \wedge \mathsf{Friends}(y, z) \Rightarrow \mathsf{Friends}(x, z)$$

which might improve log-likelihood because it can be used to predict

$$\mathsf{Friends}(\textit{anna}, \textit{christin}).$$

In contrast, without subsampling, the fact $\mathsf{Friends}(\textit{anna}, \textit{christin})$ can simply be predicted from $\mathsf{Friends}(\textit{christin}, \textit{anna})$ using the symmetry rule, hence there would not be any reason for adding the transitivity rule.

Finally, note that given a list of rules $\Lambda = (\alpha_1, \ldots, \alpha_m)$, there often exists a probability-assigning function $\psi'_\Lambda(.) \neq \psi_\Lambda(.)$ that will lead to a better log-likelihood score; in fact, an optimal one can be obtained using the geometric programming formulation from [55]. Following that approach would change the meaning of a rule's weight – the weight of a rule α_i would, in the simplest case, correspond to the probability of facts that are predicted by α_i and not predicted by $\alpha_1, \ldots, \alpha_{i-1}$. Using the fixed distribution $\psi_\Lambda(.)$, the case we follow, has several practical advantages. First, it means that all rules and their weights can be understood in isolation from the other rules and their weights – a rule's weight gives a lower bound on the probability of the facts it predicts. Second, it means that adding more rules at some point stops improving the log-likelihood even if there are some not yet used rules, which we think of as a form of heuristic regularization.

6.6 EXPERIMENTS

In this section, we experimentally evaluate our proposed approach stratified k-entailment in isolation and as a whole system together with our proposed heuristic rule learner. The goal of our empirical evaluation is to address the following two questions:

1. How does our proposed STRiKE reasoning method compare to other forms of inference?
2. How does our approach compete with state-of-the-art methods?

We investigate these questions in the same order. In all the experiments reported in this section, we set the parameter of (stratified) k-entailment to $k = 5$.

6.6.1 *Evaluation of STRiKE Inference*

To address the first question, we compare the following four approaches to reasoning with a fixed set of relational rules:

MLN-MAP uses RockIt [90] to perform MAP-inference in a Markov logic network learned using the default structure learner from the Alchemy package.

PosLog is the possibilistic logic inference approach from [55].

k**-entailment** is the k-entailment algorithm proposed in Section 6.1.

STRiKE is our stratified k-entailment proposed in Section 6.4.

1S is the one-step prediction, which returns max-aggregated predictions produced by a single iteration of forward chaining. It does not use constraints.

We compare these five approaches on the UWCSE and Proteins data set, which were used in [55].We use the same sets of rules, constraints, and MLNs as in [55]; hence all methods use the same theory.

We follow the same experimental protocol as [55]. In particular, we randomly divide the constants into two disjoint sets of equal size. The training set consists of atoms containing only the constants from the first set, and the test set contains only the constants from the second set. We then predict the set of facts given evidence sets of increasing size using the four inference methods and compute the Hamming error, which measures the size of the symmetric difference between the predicted set of facts and the set of facts in the test set. We then report the cumulative differences (CHED) between the errors of the models and a baseline, which is MAP-inference in Markov logic networks.

The results are shown in Figure 5. A specific inference method outperforms MAP inference in MLNs for a range of evidence set-sizes if the curve is increasing and performs worse otherwise. Steeper curves signify large differences in performance.

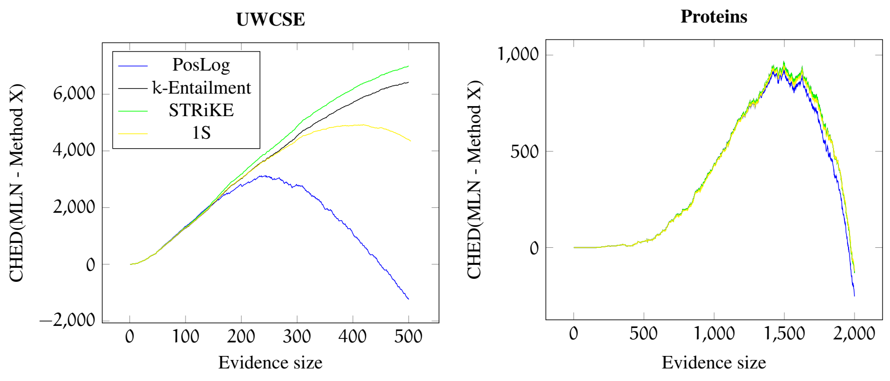

Figure 5: Comparison of the performance for inference given a fixed rule set for the UWCS (left) and Proteins dataset (right). The results show the cumulative Hamming error as a function of the size of the evidence set between the predictions made by PosLog, (2) k-entailment, (3) STRiKE, and (4) one-step versus MLN-MAP. Increasing slopes mean that a specific inference approach outperforms MLN-MAP.

In this case, STRiKE performed best among the considered methods. For UWCSE, it improves MLNs consistently, across the entire range of evidence sizes, in contrast to the possibilistic logic strategy which is only effective for sufficiently small evidence sets. On the proteins dataset, we can see that MLNs perform best for large evidence sets; the difference between STRiKE and possibilistic logic is also smaller on this dataset. The relatively weak performance of this variant shows that having constraints can be beneficial. On the Proteins dataset, on the other hand, there was no difference between the variants with and without constraints.

To better understand why MLNs outperformed STRiKE for large evidence sets on the Proteins dataset, we perform the following experiment. We select the n most confident rules from the rule set and then report the cumulative

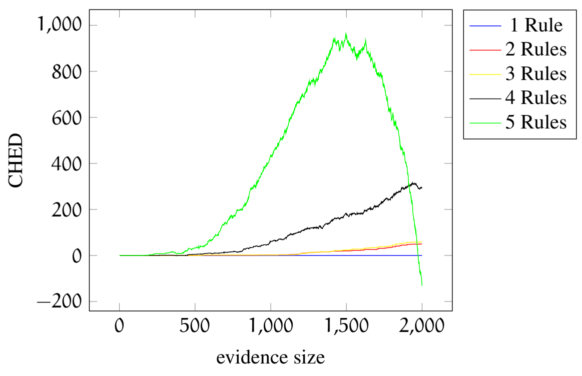

Figure 6: The cumulative Hamming error as a function of the size of the evidence set between the predictions made by STRiKE versus MLN-MAP on the Proteins dataset for rule sets with 1, 2, 3, 4, and 5 best rules. Increasing slopes mean that STRiKE outperforms MLN-MAP.

Hamming error of predictions made by STRiKE and MLN-MAP when using the simplified rule set. Figure 6 shows the results for this experiment when using only top $n = \{1, 2, 3, 4, 5\}$ rules. Having all but the least certain rule already leads to STRiKE outperforming MLNs over the entire range of evidence set sizes.

6.6.2 *Completing a Knowledge Graph*

To answer the second question, we compared our proposed method with AMIE 3 [59] and AnyBURL [75] on the knowledge graph completion task. We follow the same experimental protocol as in Section 6.6.1 but report the area under the precision-recall curve (AUC PR) because it is better suited for evaluating weighted predictions. We ran this experiment on three datasets – Nations, Kinships, and UMLS [3]. We use the following acronyms:

AMIE 3 is a rule-learning method that uses one-step inference.

AnyBURL is a rule-based method for knowledge-graph completion using one-step inference.

STRiKE is our inference method from Section 6.4 with rules learned using a beam search with the heuristic proposed in Section 6.5.

When a fact can be derived with multiple confidence using one-step inference, we take the maximal, i.e., max-aggregation, since it performs better than noisy-or in practice [96]. It is important that all of these methods were designed with different goals in mind, but all of them can be, and historically were, used for knowledge graph completion. We allow only constant-free range-restricted rules with object-identity constraints for all of them. Having constants in rules is meaningless in an inductive setting.

The algorithms from AMIE family[4] were developed to mine rules on large knowledge graphs without sacrificing the completeness of the search. They proved to be suited for the task, and, in turn, the rules they learned started to

4 AMIE, AMIE+, AMIE 3

be used to complete knowledge graphs by applying one-step prediction. As such, comparing AMIE 3 with its default setup leads to unfair competition.[5] Therefore, we ran AMIE 3 with the following settings: the maximum number of literals in a rule was set to 3,[6] recursivity limit was set to 4, the minimum size of the relations to be considered as head relations was set to 10, the minimum support was set to 2, the minimum head coverage was set to 0.05, and the minimum value of PCA confidence was set to 0.01. We used PCA confidence as a rule's weight, which was done in the original paper [39].[7]

The algorithms from the AnyBURL family were developed to learn rules by generalizing sampled paths in the knowledge graph, hence acting as a heuristic and incomplete rule-learning procedure. However, AnyBURL learns many rules; therefore, we selected only those with confidence with at least 0.5. AnyBURL was executed with the default setting – 7 threads and a maximum runtime of 100 seconds.

To learn a theory for STRiKE, we subsampled the dataset uniformly to obtain two evidence sets of equal size. Then, we used our rule learning method from Section 6.5 with the following settings: beam size was set to 4, the number of iterations of beam search was set to 5 per a predicate in the head, minimum coverage was set to 1, the maximum number of literals in the rule's bodies was set to 3, minimum precision of 0.5, and the maximal number of variables within a rule was limited to 5. We learned all possible connected constraints using the domain-theory learner from Section 5.4 with at most 2 variables and 2 literals on the training data.

Our method learns only a handful of rules, while AMIE's and AnyBURL's theories consist of thousands. Table 2 shows the exact number of rules learned by each method on each dataset. Recall that our method utilizes constraints as well; hence, we show the number of definite rules and constraints, whereas the other methods contain only definite rules.

Table 2: Number of learned definite rules and constraints in learned theories with AMIE 3, AnyBURL, and our rule learner.

	AMIE 3	AnyBURL	our rule learner	
	# rules	# rules	# rules	# constraints
Nations	122904	1295227	40	1476
Kinships	13083	25366	57	467
UMLS	8010	42353	103	1254

The results are shown in Figure 7. The left column displays AUC-PR of the methods as a function of evidence size. The right column displays illustrative PR curves for different evidence sizes on the three datasets; these allow us to gain better insight into the behavior of the methods. Here, we also added STRiKE without any constraints for comparison. As can be seen from these plots, using constraints helps STRiKE to obtain better precision but at the cost of decreasing the recall, which is to be expected. In fact, constraints are making STRiKE too cautious. STRiKE also scales up poorly when no

5 The default setup leads to learning only a few rules, which usually derive only a few facts.
6 This value was selected for all datasets because AMIE 3 did not finish on the Nations dataset within a day.
7 All three methods were ran in the same environment with 16GB of memory.

constraints are provided; this is caused by too many fragments that need to be evaluated. Hence, we only show this version on the smallest dataset – Nations – see Figure 7a.

The order of methods stays unchanged through all evidence sizes on Nations. This is different for the other two larger datasets, where we can see that at some point in the evidence size, STRiKE starts to perform better than the other methods. To explain this, we can examine precision-recall plots from which we may observe the trend that STRiKE provides more precise predictions than the other methods. AMIE's and AnyBURL's precisions suffer from deriving too many false positive facts; the size of false positives can be orders of magnitudes higher than those predicted by STRiKE. In turn, this raises their recall. In theory, we can try to increase recall by increasing k in STRiKE, which would make the inference process longer.

The empirical evaluation done in this section suggests that our approach is more cautious than standard knowledge-graph completion methods, which suffer from huge rule sets they rely on. Naturally, one would try to select rules carefully rather than mining all of them – which is exactly what AnyBURL does – or focus on a robust aggregation of multiple rules that derive the same fact – which is the active field of research [13, 96]. Once more, we stress that these two systems were developed and are often evaluated in a transductive setting of the same task where rules involving constants may boost performance. In addition, note that the most popular evaluation metrics for knowledge graph completion are query-based – mean reciprocal rank (MRR) and Hits@k. In other words, a model is evaluated using a handful of queries that only target the first (head) or second (tail) argument of a partially grounded fact, i.e., who is the most probable friend of *ann* – $\mathtt{Friend}(ann, ?)$.[8][9] This is a reasonable metric for large knowledge graphs; however, it favors neural methods since symbolic methods threaten all non-derived, i.e., non-entailed, facts as having the same score [5]. Typically, the set of queries is fixed with the work of [93] being an exception.

Finally, we abstain from a detailed runtime analysis since we know from the start that our reasoning procedure would be more complex than single-step prediction. We only briefly mention time requirements for each part of the knowledge graph completion. Considering the rule learning part – AnyBURL was always the fastest learner, using up to two minutes. AMIE usually ran up to one hour, and our heuristic rule learner with up to two hours. Considering the runtime of the inference part depicted in Figure 8 – AMIE was fastest because it had fewer rules than AnyBURL, which followed. STRiKE was slowest, clearly, due to much more complex reasoning than the one-step inference used in the other systems – clearly, this does not hold for Nations where the constraints were too tight and forbidding almost all derivations.

6.6.3 *Conclusions*

During these experiments, we showed that the proposed rule-based inference is more effective than MLNs, the possibilistic logic strategy from [55], and one-step prediction when all inferences were run with the same theories. Also,

8 The giraffe *liz* is the correct answer.

9 Although it is possible to ask what is the most probable relation that holds between *ann* and *liz*, i.e., $?(ann, liz)$, such queries are usually not present in the test set.

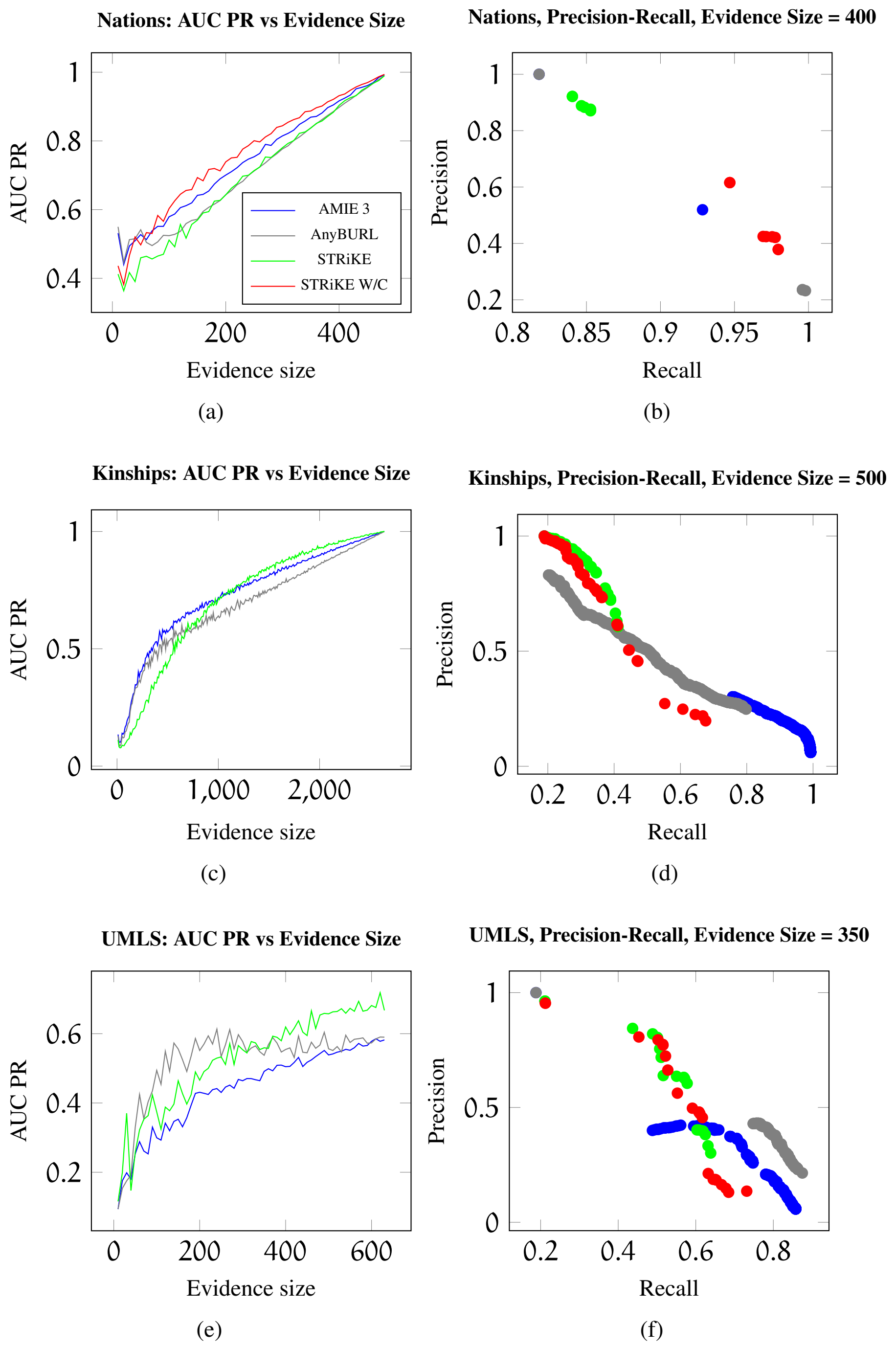

Figure 7: Comparison of AMIE 3, AnyBURL and STRiKE one Nations (top), Kinships (middle), and UMLS (bottom) datasets. **Left:** Area under precision-recall curve as a function of evidence size. **Right:** precision-recall plots for selected evidence size. *STRiKE W/C* is our method without constraints in the theory.

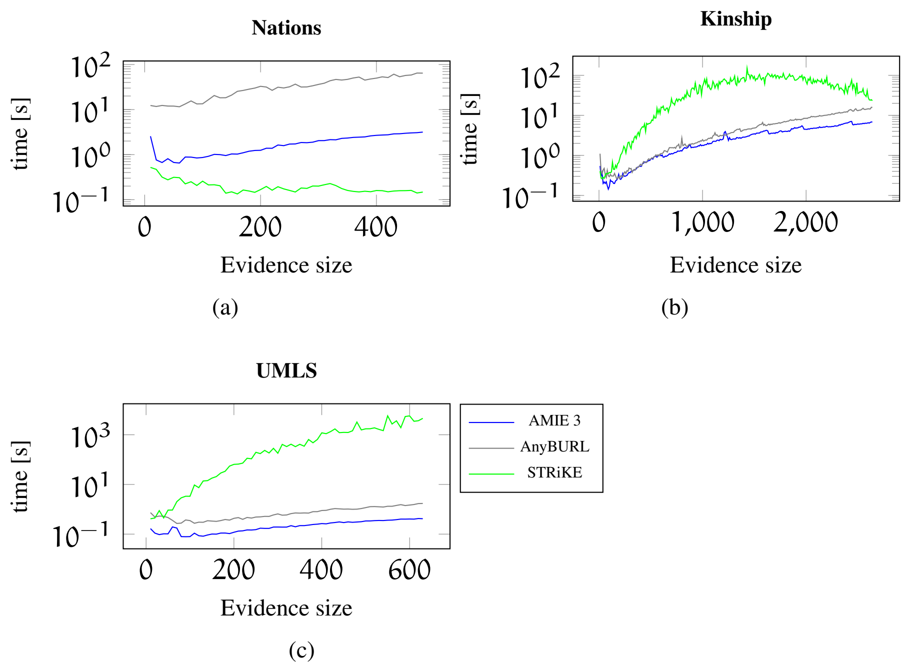

Figure 8: Comparison of AMIE 3, AnyBURL and STRiKE inference time one Nations (top left), Kinships (top right), and UMLS (bottom) datasets in seconds.

our proposed heuristic rule learner produces relatively small theories that, when accompanied by stratified k-entailment, tend to provide more cautious predictions than the current system, with the drawback of failing to achieve the same recall.

6.7 RELATED WORK

Our approach can be seen as an alternative to Markov Logic Networks [111]. MLNs address the brittleness of classical logic by using weighted rules that define a probabilistic graphical model. That also implies a complex relationship between the weights of rules at hand, which can no longer be described as intuitive as confidence. Among others, this means that MLNs can be used for marginal inference, which our method does not support. However, learning MLNs is challenging, and their effectiveness for the knowledge base completion task is not well understood. The same holds to a large extent for other statistical relational learning systems, e.g., ProbLog [31] and HL-MRF [6] as well.

Our work is also related to approaches for learning relational rules from data. The most closely related work falls in the area of structure learning for statistical relational learning, e.g., TDSL [50] for MLNs, or SAYU [29] for Bayesian networks. These approaches usually employ a beam search and typically evaluate a candidate rule's usefulness in the context of the current model. However, for formalisms like Markov logic, unless one considers a restricted model class, it is intractable to select rules that maximize the log-likelihood of the data. For this reason, an approximate measure, such as pseudo-likelihood, is often used instead, e.g., [50]. Our approach to rule learning is quite different from traditional approaches to inductive logic programming, e.g., Aleph [117] or FOIL [106], which typically employ a cover-removal style approach that scores each rule independently by looking at, for example, the difference in the number of positive and negative examples a rule covers. Similarly, more recent approaches to mining relational rules from knowledge graphs, e.g., AMIE [39] and [147] continue this tradition of evaluating the usefulness of each rule in isolation.

Knowledge graph completion [80, 114], a restriction of knowledge base completion to only binary relations, is an active field of research with many methods developed in the past decade. On the contrary, our approach is applicable beyond binary relations, which also holds for MLNs.[10] It is possible to transform the knowledge base to one with only binary predicates – the most straightforward way is to create an id for each fact and tie up that id with all arguments by a fresh set predicates.[11] However, many current systems such as AMIE and AnyBURL employ *closed rules* language bias, i.e., enforcing each variable to appear at least twice,[12] or even more strict version of connected closed paths rules, both of which are obviously not suited for handling such transformed knowledge bases. These symbolic methods arose around several mining paradigms[13] – Apriori-like exhaustive rule mining [38, 39, 59, 137, 144], path ranking-based (PRA) [60] that learns from (sampled) paths in the knowledge graph [75–77], as well as heuristic-based searches [95, 145], e.g., by utilizing embeddings RLvLR [94], or reinforcement learning [20]. Note that from these, only RuDiK [95] learns constraints, i.e., *negative* rules that should express integrity constraints in the knowledge graph. Their application and possible knowledge graph repairement[14] are left to the user since they are not used in the inference step, whereas we use constraints within the inference phase.

Besides the language bias, there are several more distinctions between our method and those developed for the knowledge graph task. Firstly, this problem typically assumes an open world while we assume a fully observed training example. Secondly, the weight of a rule may be confidence, similar to our approach, or its approximation when its exact value is difficult to compute [75]. Several assumptions may be used to compute confidence in particular scenarios, e.g., the partial completeness assumption (PCA)[15] [39][16] for function-like relations and [100, 147]. Thirdly, these symbolic approaches usually learn thousands of Horn rules which, in turn, may infer a single fact with multiple weights. Most natural aggregation of these are taking the maximum or employing noisy-or [38] which performs poorly in practice [96]. Recent methods (learn to) prune a rule set before inference, e.g., ScaLeKB [22] and SAFRAN [96], or learn a latent representation to aggregate the predicted

10 For example, UWCS dataset, used in Section 6.6.1, contains unary and ternary predicates.

11 For example, $\mathrm{R}(a,b,c)$ would transform to a set of binary facts $\{\mathrm{R}_1(id_1,a), \mathrm{R}_2(id_1,b), \mathrm{R}_3(id_1,c)\}$.

12 Although this is not precisely correct for AnyBURL – it allows a type of rule with a single variable occurring only once, i.e., U_d – the consequence remains the same.

13 Indeed, these methods often stem from the data mining community since they are oriented for RDF-like triplets, i.e., a set of facts (A-Box), while usually ignoring RDF schema, i.e., T-Box, if even present for the data. Methods that arose from description logic would try to incorporate RDF schema by default.

14 This behavior brings RuDiK close to ORE for OWL [62].

15 Which would have to be acommodated for knowledge bases with relations of arbitrary arity.

16 In [34], this is also called as *local closed world assumption*.

weights [13]. Probabilistic interpretation of these aggregation method is an active field of research [12]. On the contrary, our approach is biased toward sound inference mechanisms.

Besides neural-based approaches such as Neural LP [141] and Lifted Relational Neural Networks [115], popular frameworks for knowledge graph completion are based on embeddings and graph neural networks (GNNs). Although embedding-based approaches gained a lot of attention and many methods were developed over the past decade,[17] these are usually applicable only to the transductive setting, e.g., RotatE [122] and ComplEx-N3 [58]. Contrary to this limitation, GNNs-based methods such as GraIL [127] and NBFNet [146] operate within the inductive setting. Finally, several hybrid approaches used both rules and embeddings, e.g., [5, 74, 113]. While some of these methods share closed paths bias, e.g., [127], we abstain from a detailed review of these approaches since they usually lack the interpretability of a rule-based paradigm.

17 However, many of the early works had flawed evaluation protocols with *filtered* and *non-filtered* Hits@k and unfair tie-breaking [123]. Since then, several frameworks were developed to tackle this issue, e.g., PyKEEN or LibKGE.

Part IV

APPLICATIONS

7 ONE INTEGER SEQUENCE, MULTIPLE EXPLANATIONS

In Chapter 4, we proposed a method that discovers new, unseen, combinatorial integer sequences that have a combinatorial interpretation. Motivated by the results that showed that some of these sequences are already contained in OEIS, we turn our attention in this chapter to a small demonstration of explanations created by different methods that solve a similar task. Namely, these are:

QSynt [40] – reinforcement learning-based model that synthesizes programs for integer sequences, and

Sequence Machine [67] – a generator of stacked programs that generate integer sequences.

Indeed, all the approaches, including ours, construct prescriptions of integer sequences, however, each one of them was developed with a different goal. Specifically, our approach is biased toward enumerating all combinatorial sequences with descriptions in first-order logic. Contrary, QSynt produces programs in a heuristic, i.e., incomplete, way. Similarly, Sequence Machine generates programs that may even be stacked,[1] in an assembly-like language. On the contrary, our approach nor QSynt uses stacking of formulae or programs. The aim of this chapter is to show a few OEIS's sequences with corresponding prescriptions made by each of these algorithms rather than a rigorous comparative study.[2]

7.1 DIFFERENT PRESCRIPTIONS

As we mentioned above, we abstain from a profound comparison of the methods as well as their languages;[3] we will discuss only operators that occur in shown programs. We will select few sequences from OEIS, show their description from OEIS,[4] and discuss programs found by each approach.

7.1.1 *A Simple Combinatorial Sequence*

Consider the OEIS sequence A165, i.e.

$$2, 8, 48, 384, 3840, \dots$$

which is given by *the double factorial of even number*, i.e.

$$a(n) = (2n)!! = 2^n * n! \tag{8}$$

1 *Stacking* here means an arbitrary program from an archive of previously generated programs may be used as a subroutine.

2 For the ease of presentation, we consider domain $1, 2, 3, 4, \dots$ as we did in Chapter 4.

3 We refer to [40] for QSynt's and [65] for Sequence Machine's language, respectively.

4 The description is available on the webpage https://oeis.org/Axyz where *Axyz* is the A-identifier of a sequence.

Our approach generated the sentence

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

which can be described as follows: *We are counting the Cartesian product of two sets – permutations of* $[n]$ *and subsets of* $[n]$. Indeed, Equation (8) expresses the number of such products. Note that the set of subsets contains an empty set since $\forall x \exists^{=1} y\ R(x,y) \models \exists x \exists y\ R(x,y)$, and hence the middle disjunction entails the first one.[5]

QSynt produced $\mathrm{loop}((y+y) * x, x, 1)$ which can be rewritten, for clarity, as a recurrent function[6]

$$f(n) = \begin{cases} 1 & \text{if } n \leqslant 0 \\ (n+n) * f(n-1). & \end{cases}$$

Clearly, this is equivalent to $\prod_{i=0}^{n-1} 2(n-i)$ and hence to Equation (8) as well.

Sequence Machine produced assembly-like program depicted in Algorithm 5 that can be rewritten to a pseudocode in a higher-order language for clarity, i.e., Algorithm 6. The computation of the latter algorithm can be expressed as $\prod_{i=0}^{n-1} (2n - 2i)$ which is equal to Equation (8).

Algorithm 5 Sequence Machine Program for A165

```
1: mov $1, 1
2: mul $0, 2
3: lpb $0
4: mul $1, $0
5: sub $0, 2
6: lpe
7: mov $0, $1
```

Algorithm 6 Higher-order Pseudocode for Algorithm 5

```
1: $v_1 \leftarrow 1$
2: $v_0 \leftarrow 2 * n$
3: while $v_0 \geqslant 0$ do
4:     $v_1 \leftarrow v_1 * v_0$
5:     $v_0 \leftarrow v_0 - 2$
6: end while
7: return $v_1$
```

7.1.2 *Yet Another Combinatorial Sequence*

Next, consider the OEIS's sequence A126883, i.e.

$$1, 7, 63, 1023, 32767, \ldots$$

with quite wordy description: $a(n) = (2^0) * (2^1) * (2^2) * (2^3) * \cdots * (2^n) - 1 = 2^{T(n)} - 1$ *where* $T(n) = \mathrm{A000217}(n)$ *is the* n*-th triangular number.*[7]

5 One could argue that such a sentence is in fact redundant since it does not differ semantically from $(\exists x\ P(x) \lor \neg P(x)) \land (\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x))$, i.e., a sentence that is decomposable. However, to solve this case, we would need to i) integrate a more complex reasoning in the redundancy checking phase, which we left for future work, and ii) extend the database with a post-processing step that would combine different sentences to get their element-wise product, e.g., to combine $\exists x\ P(x) \lor \neg P(x)$ and $(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x))$.

6 $f(n)$ is actually an instantiation of a QSynt's language construct that corresponds to functional-like definition of recursive function with the first argument, e.g., $(y+y) * n$, being a function of two arguments.

7 The mentioned sequence A000217 is defined as follows: $a(n) = \mathrm{binomial}(n+1, 2) = \frac{n*(n+1)}{2} = 0 + 1 + 2 + \cdots + n$.

Our approach generated the sentence

$$(\forall x \forall y \; R(x, y) \vee \neg R(y, x)) \wedge (\exists x \exists y \; R(x, y))$$

that can be described as *computing the number of undirected graphs with n nodes having at least one edge*. Note that these graphs can contain loops; thus there are $2^n * 2^{\frac{n(n-1)}{2}} - 1$ such graphs with n nodes, which is equivalent to the description mentioned above.

QSynt produced $\mathrm{loop2}(x * y, y + y, x, 1, 2) - 1$ that can be rewritten as Algorithm 7 since the $\mathrm{loop2}$ construct is defined as follows:

$$\mathrm{loop2}(f, g, a, b, c) = \begin{cases} b & \text{if } c \leqslant 0 \\ \mathrm{loop2}(f, g, a-1, f(b, c), g(b, c)). \end{cases}$$

Following the computation, the resulting value can be expressed as $2^{\frac{n(n+1)}{2}} - 1$ and hence is equal to the expression we derived above for our approach.

Algorithm 7 Pseudocode for QSynth's $\mathrm{loop2}(x * y, y + y, x, 1, 2) - 1$

```
1: $b \leftarrow 1$
2: $c \leftarrow 2$
3: while $n > 0$ do
4:     $b \leftarrow b * c$
5:     $c \leftarrow c + c$
6:     $n \leftarrow n - 1$
7: end while
8: return $b - 1$
```

Sequence Machine produced program depicted in Algorithm 8 that can be rewritten into a single formula $2^{\binom{-n}{2}} - 1$ where the binomial number is extended for negative values [52]. Obviously, this approach profits from its language bias that already contains a combinatorial function, while the other approaches do not.

Algorithm 8 Sequence Machine Program for A126883

```
1: sub $1, $0
2: bin $1, 2
3: mov $0, 2
4: pow $0, $1
5: sub $0, 1
```

7.1.3 *A Sequence with Negative Numbers*

Finally, consider the OEIS's A33999 sequence, i.e.

$$-1, 1, -1, 1, -1, \ldots$$

which can be encoded as $a(n) = (-1)^n$.

Although our approach, as described in Chapter 4, did not generate this particular sequence, it is possible to make a few adjustments in order to do so.

For example, computing the number weighted of models of sentence $\forall x\ P(x)$ with the positive weight set -1, i.e., $w(P) = -1$,[8] produces this sequence. However, the explanation then is not so natural.[9]

QSynt produced $\mathrm{loop}(0 - x, x, 1)$ that downplays to $\prod_{i=1}^{n} -1$.

Sequence Machine produced Algorithm 9, which translates into $\binom{-1}{n}$, again utilizing the extended combinatorial function.

Algorithm 9 Sequence Machine Program for A33999

```
mov $1, −1
bin $1, 0
mov $0, 1
```

7.2 CONCLUSION

We could keep listing more examples like these, e.g., the sequence A290840 discussed in Section 4.5.2, for some more time; however, these few above should be enough for a reader to get the intuition. Finally, from the two first integer sequences, we observe that a single approach is not always the most comprehandsible for all sequences.

8 The model count of this particular sentence does not depend on the negative weight since there is only a single model with all atoms being positive.

9 Which is also why we abstained from different weights setups in this work.

8 EXPLOITING BOND SYMMETRY FOR RULE LEARNING

Motivated by the effectiveness of the domain theory-based pruning we presented in Chapter 5, we briefly outline, in this chapter, a single application and stress the advantage it brings for a particular set of datasets, namely those from the NCI [109] molecular datasets collection. As in Chapter 5, we are working within the learning from interpretation learning setup (Section 2.3.1).

8.1 INSPECTING LEARNED DOMAIN THEORY

As we already outlined in Section 5.6, when learning a domain theory on an NCI dataset, we derived two types of rules: i) mutex constraints, which says that an atom can be only of a single type, e.g.

$$\forall x\ \neg \mathsf{Carb}(x) \vee \neg \mathsf{Hydro}(x),$$

and ii) bond symmetry, e.g.

$$\forall x \forall y\ \mathsf{Bond}(x,y) \Rightarrow \mathsf{Bond}(y,x).$$

Implementation of the mutex constraint into a refinement is quite straightforward. This kind of optimization, i.e., forbidding some type of refinements w.r.t. a clause α, is particularly favorable in cases where evaluating a single sentence is resource demanding, e.g., computing coverage of a large dataset, since we know ahead, thanks to the domain theory, that such refinement is useless. Therefore, we turn our attention to the bond-related domain theory. To see the advantage of utilization of this knowledge, consider the following example.

Example 38. *Consider a Horn rule*

$$\alpha = \forall x \forall y\ \mathsf{Hydro}(y) \wedge \mathsf{Bond}(x,y) \Rightarrow \mathsf{Carb}(x).$$

Then, α is equivalent to

$$\beta = \forall x \forall y\ \mathsf{Hydro}(y) \wedge \mathsf{Bond}(y,x) \Rightarrow \mathsf{Carb}(x)$$

w.r.t. bond symmetry, i.e.

$$\mathcal{B} = \{\forall x \forall y\ \mathsf{Bond}(x,y) \Rightarrow \mathsf{Bond}(y,x)\}$$

which we already saw in Example 31.

Now, consider refining the body of α with new bond literal containing a fresh variable, i.e.

$$\gamma_1 = \forall x \forall y \forall z\ \mathsf{Bond}(z,y) \wedge \mathsf{Hydro}(y) \wedge \mathsf{Bond}(x,y) \Rightarrow \mathsf{Carb}(x),$$
$$\gamma_2 = \forall x \forall y \forall z\ \mathsf{Bond}(y,z) \wedge \mathsf{Hydro}(y) \wedge \mathsf{Bond}(x,y) \Rightarrow \mathsf{Carb}(x).$$

Then, unsurprisingly, both γ_1 and γ_2 are also equivalent w.r.t. $\mathcal{B}$.

Adjusting a refinement operator to utilize the bond symmetry knowledge is easy – instead of adding a single bond literal, its symmetric counterpart is also added. This also holds for double, triple, quadruple, and aromatic bond.

Example 39. *Consider the bond symmetry knowledge from Example 38 and path-like rule with* $n-1$ *body* bond *literals, e.g.*

$$\forall x_1 \dots \forall x_n \ \mathrm{Bond}(x_1, x_2) \wedge \mathrm{Bond}(x_2, x_3) \wedge \dots \wedge \mathrm{Bond}(x_{n-1}, x_n) \Rightarrow C(x_n).$$

Then, there are $2^{n-1}-1$ *more rules*[1] *that are equivalent to this rule w.r.t. domain theory. See, adding an arbitrary atom* $\neg\mathrm{Bond}(x_{i-1}, x_i)$ *leads to the same equivalence class.*

8.2 UTILIZING BOND SYMMETRY

Among others, the beam search strategy is popular for a heuristic traversal of the search space, e.g., [51]. Utilization of bond symmetry is particularly practical for this kind of problem since it removes domain-equivalent clauses that would otherwise occupy the beam and skew the search. For instance, consider γ_1 and γ_2 from Example 38; evaluating both of these would lead to the same value, e.g., support or accuracy. Having both of these in a beam would narrow the search space traversed by the search procedure.

Motivated by this knowledge, we enhanced a refinement operator with the bond symmetry for structure learning of Lifted Relational Neural Networks (LRNN) [115], a template-based method that combines function-free first-order logic and neural networks. Specifically, it is built around the concept of weighted Horn rules where the weights correspond to weights in a neural network. A ground neural network is constructed w.r.t. a learning example by computing the least Herbrand model[2] of the set of Horn rules given a set of facts, i.e., the example, and constructing weight connections among the facts in the Herbrand model which correspond to learnable weights that are, thereafter, learned by stochastic gradient descent. Hence, we applied beam search enhanced with the refinement operator and learned LRNN by altering the structure learning phase, i.e., learning Horn rules, and weight learning, i.e., the neural network part. This approach proved to be effective as the learned LRNN outperformed hand-crafted templates, i.e., logical theories, on which the method relies.

In detail, the structure learning algorithm initially starts only with a set of predicates $\mathcal{P}$ that are in the training data. In each iteration, it deploys a top-down beam search to learn a Horn rule with a fresh predicate in the head, i.e., $b_1 \wedge \dots \wedge b_n \Rightarrow h$ where predicate in the head h does not occur in $\mathcal{P}$.[3] After the beam search terminates, the set of predicates $\mathcal{P}$ is extended by the fresh predicate in h. Thus, in the next iteration, the algorithm may use this predicate inside new rules and, hence, stack the invented predicates. Note that the newly invented predicates correspond to induced soft (hierarchical) concepts. Thus, this particular approach can be seen as a statistic predicate invention with the ability to induce hierarchical latent concepts.[4]

1 With the same prefix $\forall x_1 \dots \forall x_n$.
2 For example by employing forward chaining as shown in Example 6.
3 We do not allow recursive rules, so body literals can contain predicates only from the previous iteration.
4 We refer to [116] for a detailed description.

Part V

CONCLUSION

9 CONCLUSION

Here, we sum up the contribution of the proposed methods in Part iii and discuss possible future work in the following section.

9.1 THESIS CONTRIBUTION

In general, we proposed two techniques for sentence space pruning. The common property of these lies in their preservation of completeness of the search procedures they were designed for. In the first case, which resembles a specialized pattern mining scenario and is described in Chapter 4, a set of task-dependent pruning techniques was developed to discover integer sequences with combinatorial explanations. The set of pruning techniques proved to be an order of magnitudes faster than a vanilla version. We showed that our approach is fruitful by generating a database of first-order formulae and corresponding integer sequences, although we used quite restrictive language bias. Compared to other approaches in the line of mathematical discovery, e.g., HR system [24], we set up the goal of investigating a subset of integer sequences, whereas other methods usually focus on generating and explaining more general sequences. Our direction of generation of integer sequences from scratch is orthogonal to the current approaches, e.g., [27, 40] which learn a model that, in turn, generates integer sequences or their prescription. We contributed to the OEIS as few entries were enriched with our first-order formula description[1] without our direct involvement, which clearly shows the potential benefit of our approach for the wider scientific community.[2]

In Chapter 5, we presented a more general sentence space pruning method that can be deployed in many rule-learning algorithms in the literature. Our experiments show that utilizing domain theory can be beneficial for exhaustive searches even though our method needs to learn domain theory at the beginning. The method differs from the current systems since they often rely on some predefined language bias, e.g., closed paths, which are typically not devised for a particular dataset. The output of these methods is postprocessed, in the case of the large learned rule set, while our approach uses the regularity in the data during the learning process. The methods OP [21] and RARL [101] are clear exceptions from the mainstream since they employ some domain knowledge (from the T-Box) for rule learning, whereas the rest uses only the data (A-Box).[3] In particular, other systems could benefit by utilizing our approach in domains where regularities are common, e.g., some relations being symmetric, as in Chapter 8.

1 For example, A290840.

2 Historically, papers presented the number of newly derived integer sequences that were accepted into OEIS. However, we feel that this is not proper anymore in the current age of large-scale integer sequence generation. Similarly, [67] could easily overfill OEIS.

3 Although we use only data, we firstly learn regularities in them and exploit that knowledge later.

In Chapter 6, we presented a method suited for knowledge base completion in the presence of imperfect rules. Our approach, based on the stratification of rules and accompanied by a restricted version of logical entailment, usually derives fewer incorrect facts than standard rule-based methods that rely on large rule sets with an even more restricted version of logical entailment [76]. Besides other differences with the current approaches, such as the learning setting, e.g., inductive vs transductive setup, knowledge graph vs base, query-based evaluation vs ground truth, our approach is primarily focused towards cautious inference with a specially tailored rule learner. This fills the gap in the current state of the art, which is usually concerned with the simple application of large rule sets. Recently, a cautious aggregation of these large rule sets became an active field of research [13].

Each method can be used in isolation, including the heuristic rule learner and stratified k-entailment from Chapter 6. However, the sentence state pruning method from Chapter 5 can be used together with methods from Chapter 6.

9.2 FUTURE WORK

The methods from this thesis can be used in various machine-learning tasks.[4] The most straightforward application of a method presented in this thesis is the usage of saturation-based pruning from Chapter 5 for rule learning. While we have demonstrated its effectiveness on exhaustive searches and discuss its advantages in beam search strategy, it can be deployed in current rule learning systems, e.g., [59], as a post-processing step for filtering rules that are equivalent w.r.t. domain theory to obtain a more compact representation of a learned rule set. In particular, this would find usage in the knowledge graph completion task since current systems, e.g., [76], need to aggregate predictions made by large rule sets; a pruning w.r.t. domain theory would help to remove redundant ones. Besides possible applications, the method, or at least a part of it, could be injected into a refinement operation on the implementation level in order to make it more efficient, as we discussed in Section 5.8.[5]

In Chapter 4, we investigated a way of generating integer sequences, which resulted in two contributions: i) a set of pruning techniques,[6] ii) a database of combinatorial integer sequences. The latter can be used for various machine-learning tasks. For instance, a model biased towards combinatorial integer sequences can be learned with [27] or [40] using our database as an input. In theory, our database could be merged with OEIS.[7] This would be beneficial for learning a language for the description of combinatorial problems since our database consists of first-order logical formulae, which is a native language only for logicians, whereas OEIS contains descriptions of sequences in natural language.

4 Here, we discuss possible applications and extensions of the proposed methods. Notes on more specific extensions of the methods are described at the end of each chapter.

5 Indeed, scalability is a strong argument nowadays since there are many quite fast algorithms that learn thousands of rules in the blink of an eye.

6 See Section 4.7 for a detailed discussion of possible future work regarding the pruning techniques.

7 Hence constructing an *OEIS+* as few reviewers hinted out.

Finally, Chapter 6 sheds new light on the field of relational inference. Hence, one may follow our path by presenting newer versions of altered logical entailment. One such representative is *voting entailment* [53], which was developed with the same underlying principle as k-entailment.

Part VI

APPENDIX

A IMPLEMENTATION DETAILS

This section contains a few implementation details for all methods that were presented through Part iii. The core algorithms are available at https://github.com/martinsvat together with experiment setups, datasets, and results. A snapshot of our database from Chapter 4 is accessible at https://fluffy.jung.ninja.[1] For some parts, we used third-party libraries. However, the main algorithms are implemented in Java. We heavily relied on, and are grateful for, the following libraries:

1. *Prover9* [72] – for theorem proving in Section 4.4.2.4.
2. *FastWFOMC.jl* [129] – to compute cell graphs and generate combinatorial spectra.
3. θ-subsumption engine [56] – to check the coverage of examples.
4. *Sat4j library* [61] – in connection with the above, this allowed us to prove theories using an incremental grounding solver.
5. AUC-PR [30] – to evaluate experiments in Section 6.6.

A.1 NOTES ON OPTIMIZATIONS

This section contains a few implementation details of the methods in Part iii.

A.1.1 *Notes on Isomorphism*

For the vast majority of sentence isomorphism checking, we used the hash-based approach discussed in Section 3.1 to lower the number of isomorphism calls. In fact, the task of isomorphism-based redundancy-checking techniques, i.e., *predicates*, *negations*, and *permuting arguments* Sections 4.4.2.1 to 4.4.2.3, can be encoded as isomorphism of hypergraphs (ensuring that a binary predicate can map only to another binary predicate, etc.). This is rather a technical and straightforward application of isomorphism. However, in practice, it proved to be too expensive to compute and was easily outperformed by *canonical* representation of a sentence. The latter approach comes from standard graph mining [46], where the canonical representation of graphs is used to forbid the evaluation of isomorphic graphs multiple times. Therefore, we developed a branch-and-bound algorithm that computes lexicographically minimal sentence given a $\mathbf{C}^2$ sentence of the form Equation (2) w.r.t. variable and predicate names, flipping negation signs, and permuting arguments of binary relations (according to the definitions in Sections 4.4.2.1 to 4.4.2.3). This approach proved to scale much better, even though it is not easier to compute than the isomorphism of hypergraphs. As a byproduct, we obtain sentences in a canonical representation throughout our integer sequences database.

1 If this link does not work anymore, please check the aforementioned Github page.

Example 40. *Consider the sentence*

$$\varphi = \exists x \exists y\ \neg P_0(x) \vee P_1(y).$$

Then, the canonical, i.e., lexicographical minimal, counter part is

$$\varphi' = \exists x \exists y\ P_0(x) \vee P_1(y).$$

While the result should not be surprising, several mappings θ' *derive this lexicographically minimal counterpart of* φ*. For instance, the following mappings transform* φ *into* φ'*:*

$$\begin{aligned}\theta'_1 = \{&P_0/1 \mapsto \neg P_0/1,\\ &\neg P_0/1 \mapsto P_0/1\},\\ \theta'_2 = \{&P_0/1 \mapsto \neg P_1/1,\\ &\neg P_0/1 \mapsto P_1/1,\\ &P_1/1 \mapsto P_0/1,\\ &x \mapsto y,\\ &y \mapsto x\}.\end{aligned}$$

The first mapping swaps negation signs of $P_0/1$*, whereas the second swaps and renames both variables, predicates, and swaps negation signs for the original* $P_0/1$ *predicate.*

There are several factors that, when combined, helped the canonical-based approach to outperform the isomorphism of hypergraphs, which was usually overwhelmed with too many variables in the isomorphism-checking problem, i.e., there is a single variable for each predicate name, negation flip, binary relation order, etc. One of these factors is an effective pruning of unfavorable branches in the computation of the canonical version since there are only a few ways to map the original sentence into its canonical version. Secondly, comparing two canonical sentences is as hard as comparing the equality of two lines of text. Thirdly, canonical counterparts can be computed in parallel for a set of sentences, whereas testing the isomorphism of two hypergraphs relies on some engine and its capability of parallel executions.

Finally, we note that the canonical approach only worked well for the abovementioned problem. We have not experimented with developing an algorithm that would compute canonical representation for cell graphs. Although the problem is the same in nature, the input – a (hyper)graph – differs since it represents a complete weighted graph. Hence, it would be much more efficient to utilize some specialized programs for finding the canonical representation of a graph, e.g., nauty [73].

A.1.2 *Effective Usage of Resources*

To scale up the process of either sentence generation in Chapter 4 and k-entailment in Chapter 6 we utilized bit sets. In the former case, bit sets were used to precompute calls that would be quite repetitive – this concerns mostly entailment and θ-subsumption-based methods from Section 4.4.2.6. For instance, let α and β be clauses such that

$$\alpha \wedge \beta$$

is redundant. Then, every sentence φ that contains β is redundant when refined by α, i.e.

$$\alpha \wedge \varphi,$$

as well as any refinement of φ.

In the case of Chapter 6, we used bit sets to represent subsets of constants. This was beneficial for two purposes: i) lower memory consumption, and ii) fast checking of *subset* relation between two sets of constants, i.e. $\mathcal{C}_1 \subseteq \mathcal{C}_2$.

B ADDITIONAL SCALABILITY OF SENTENCE GENERATOR

In Chapter 4, we were focused purely on building up a database of sentences and their combinatorial spectra. Here, we present one more experiment of the proposed pruning techniques, however, only to compare their effectiveness and runtime. We abstain here from computing combinatorial spectra as such, which, as we saw in Section 4.5.1, is arguable the most resource demanding part of the database building process.

We will use a slightly changed protocol from Section 4.5.1. Specifically, we are concerned only with the set of quantifiers containing a counting quantifier, i.e., $\mathcal{K} = \{\forall, \exists, \exists^{=1}\}$, since such sentence space is larger than pure $\mathbf{FO}^2$, and a richer set of predicates, i.e., $\mathcal{P} = \{\mathsf{P}_0/1, \mathsf{P}_1/1, \mathsf{R}_0/2, \mathsf{R}_1/2\}$. We run three different setups, namely:

- at most 3 literals per disjunction, at most 3 disjunctions per clause
- at most 4 literals per disjunction, at most 3 disjunctions per clause
- at most 4 literals per disjunction, at most 4 disjunctions per clause

The rest of the protocol stays unchanged.[1] Actually, the language bias in Section 4.5.1 was too strict, so not all redundancy checking technique could take place in the process.

The results are depicted in Figure 9; the order from top to bottom follows the order of the above-mentioned language restrictions. We show only the cumulative number of generated sentences (left) and the required runtime (right).[2] Although the baseline is quite fast, it generates too many sentences. In turn, it quickly ran into memory issues, e.g., the bottom figure. The post-process filtering based on the isomorphism of cell graphs is time-demanding; hence, it did not make it to the same level as the other techniques.[3] The rest of the methods behave the same as in the initial experiment of construction of a small initial database in Section 4.5.1.

1 This means 51GB memory, 48 hours of computation, 30 seconds as proving limit, and restriction of disjunction with counting quantifiers.

2 Since we are not concerned with the time needed to compute corresponding spectra.

3 Nonetheless, if we also consider the time needed to compute combinatorial spectra, we would fill in the database faster than using only the other methods.

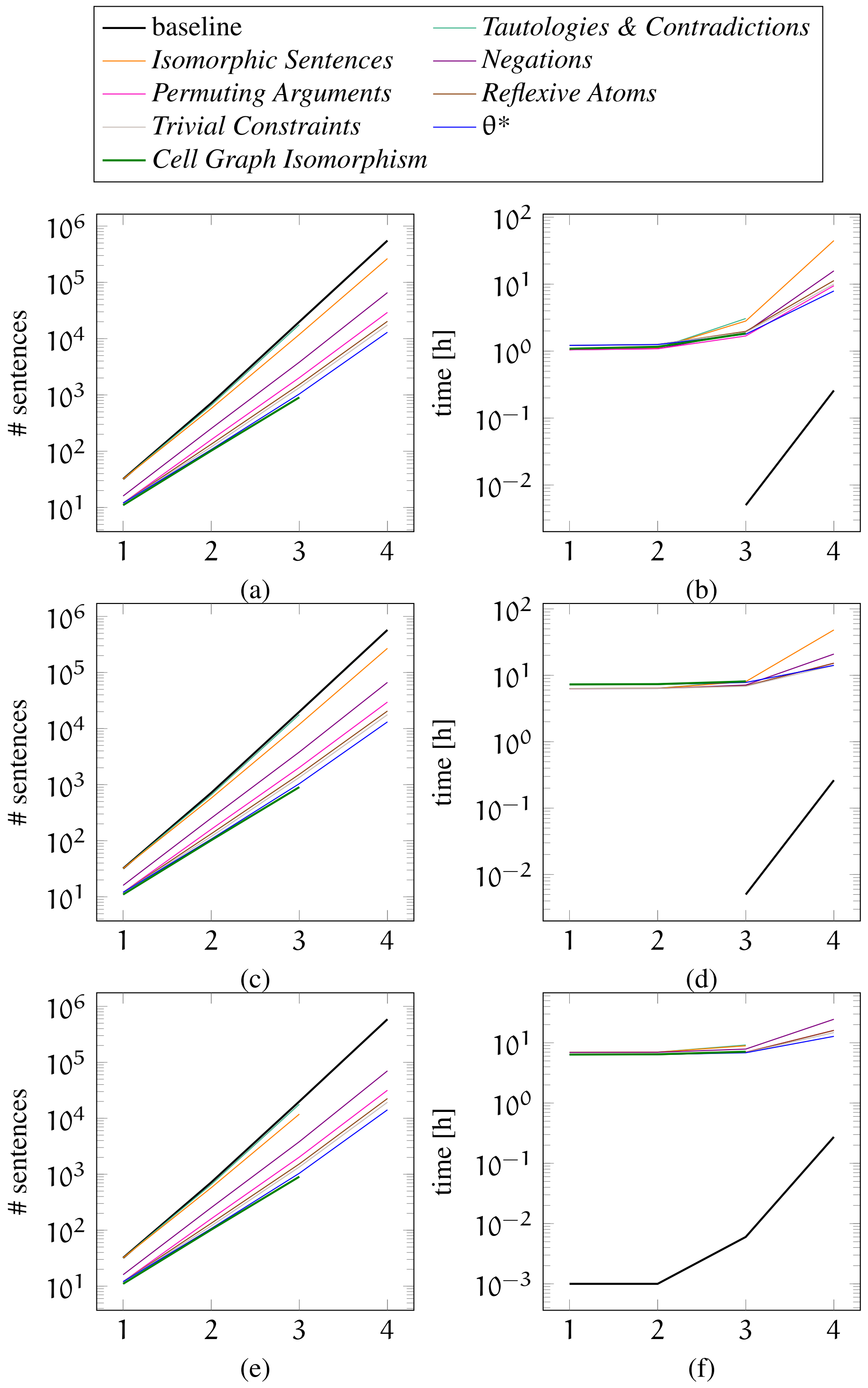

Figure 9: Cumulative number of $\mathbf{C}^2$ sentences (left) with at most x literals and the time needed to generate them (right). At most 2 unary and 2 binary predicates, $\mathcal{K} = \{\forall, \exists, \exists^{=1}\}$, at most 3 (top and middle) and 4 (bottom) literals per clause, and at most 3 (top) and 4 (middle and bottom) disjunctions per sentence.

C COMBINATORIAL INTEGER SEQUENCES FOUND IN OEIS

Table 3 is an extension of Table 1 from Chapter 4. It contains a sample of 152 integer sequences that we generated during the experiment discussed in Section 4.5.2. The entries are sorted in increasing order of OEIS identifiers.

Table 3: A larger sample of OEIS sequences that corresponds to some $\mathbf{C}^2$ sentence we generated during the experiment in Section 4.5.2.

Sentence	**OEIS ID**	**OEIS name**
$(\exists^{=1}x\ P(x))$	A27	The positive integers. Also called the natural numbers, the whole numbers or the counting numbers, but these terms are ambiguous.
$(\forall x\ P(x) \lor \neg P(x)) \land (\forall x\ \neg P(x) \lor P(x))$	A79	Powers of 2: $a(n) = 2^n$.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\forall x \forall y\ R(x,y) \lor \neg R(y,x))$	A85	Number of self-inverse permutations on n letters, also known as involutions; number of standard Young tableaux with n cells.
$(\forall x \exists y\ \neg R(y,x)) \land (\exists x \exists y\ \neg R(x,y)) \land (\forall x \exists^{=1} y\ \neg R(x,y))$	A142	Factorial numbers: $n! = 1 * 2 * 3 * 4 * ... * n$ (order of symmetric group S_n, number of permutations of n letters).
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\forall x \forall y\ R(x,y) \lor R_1(x,y))$	A165	Double factorial of even numbers: $(2n)!! = 2^n * n!$.
$(\forall x\ \neg P(x) \lor \neg R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(x,y))$	A169	Number of labeled rooted trees with n nodes: n^{n-1}.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\exists^{=1} x\ R(x,x))$	A240	Rencontres numbers: number of permutations of $[n]$ with exactly one fixed point.
$(\forall x\ P(x) \lor P_1((x))$	A244	Powers of 3: $a(n) = 3^n$.
$(\forall x \forall y\ R(x,y) \lor \neg R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(y,x))$	A248	expansion of e.g.f. $exp(x * exp(x))$.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\forall x \forall y\ R(x,x) \lor \neg R(x,y) \lor \neg R(y,x))$	A266	expansion of e.g.f. $\frac{exp(-\frac{x^2}{2})}{1-x}$.
$(\exists x \exists y R(x,y)) \land (\forall x \exists^{=1} y\ R(x,y))$	A312	$a(n) = n^n$; number of labeled mappings from n points to themselves (endofunctions).

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x\ P(x) \lor P_1((x)) \land (\forall x\ P(x) \lor P_2(x))$	A351	Powers of 5: $a(n) = 5^n$.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\forall x \forall y\ \neg R(x,x) \lor P(x) \lor R(x,y))$	A354	expansion of e.g.f. $\frac{exp(-x)}{1-2*x}$.
$(\forall x \exists y\ P(x) \lor P_1((y)) \land (\forall x \forall y\ P_1((x) \lor \neg P_1((y) \lor P_2(x))$	A400	Powers of 6: $a(n) = 6^n$.
$(\forall x\ P(x) \lor P_1((x) \lor P_2(x))$	A420	Powers of 7: $a(n) = 7^n$.
$(\forall x \exists y\ R(x,y)) \land (\forall x\ R(x,x) \lor P(x)) \land (\forall x \exists^{=1} y\ R(y,x))$	A522	Total number of ordered k-tuples ($k = 0..n$) of distinct elements from an n-element set: $a(n) = \sum_{k=0..n} \frac{n!}{k!}$.
$(\forall x\ P(x) \lor P_1((x) \lor P_2(x)) \land (\forall x\ P(x) \lor P_3(x))$	A1020	Powers of 11: $a(n) = 11^n$.
$(\forall x\ P(x) \lor P_1((x) \lor P_2(x) \lor P_3(x))$	A1024	Powers of 15.
$(\forall x \forall y\ R(x,y) \lor \neg R(y,x)) \land (\exists x R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(x,y))$	A1189	Number of degree-n permutations of order exactly 2.
$(\forall x \exists^{=1} y\ \neg R(y,x)) \land (\exists^{=1} x \forall y R(x,y))$	A1804	$a(n) = n! * C(n,2)$.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\forall x\ \neg R(x,x) \lor P(x) \lor P_1((x))$	A1907	expansion of e.g.f. $\frac{exp(-x)}{(1-4*x)}$.
$(\forall x\ P(x) \lor P_1((x)) \land (\exists^{=1} x\ \neg P(x)) \land (\exists^{=1} x\ \neg P_1((x))$	A2378	Oblong (or promic, pronic, or heteromecic) numbers: $a(n) = n * (n+1)$.
$(\forall x\ R(x,x) \lor \neg R(x,x))$	A2416	$a(n) = 2^{n^2}$.
$(\exists x\ \neg R(x,x)) \land (\exists x \forall y\ \neg R(y,x))$	A5019	The number of n X n (0,1)-matrices with a 1-width of 1.
$(\forall x\ P(x) \lor P_1((x)) \land (\forall x \forall y\ \neg P(x) \lor P_2(y))$	A5056	$a(n) = 3^n + 2^n - 1$.
$(\forall x \forall y\ P(x) \lor P_1((y)) \land (\exists x\ P(x)) \land (\forall x \forall y\ P(x) \lor P_2(y))$	A5367	$a(n) = 2 * (2^n + 1) * (2^{n+1} - 1)$.
$(\forall x \forall y\ R(x,y) \lor \neg R(y,x))$	A6125	$a(n) = 2^{\frac{n*(n-1)}{2}}$.
$(\forall x \forall y\ R(x,y) \lor \neg R(y,x)) \land (\forall x \exists y\ R(x,y)) \land (\exists x R(x,x))$	A6129	$a(0)$, $a(1)$, $a(2)$, ... satisfy $\sum_{k=0..n} a(k) * \mathrm{binomial}(n,k) = 2^{\mathrm{binomial}(n,2)}$, for $n \geqslant 0$.

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x\ \neg R(x,x)) \wedge (\forall x \forall y\ \neg R(x,y) \vee R(y,x)) \wedge (\forall x \forall y\ \neg R(x,y) \vee P(x))$	A6896	$a(n)$ is the number of hierarchical linear models on n labeled factors allowing 2-way interactions (but no higher order interactions); or the number of simple labeled graphs with nodes chosen from an n-set.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\forall x \forall y\ R(x,y) \vee \neg R(y,x))$	A6898	$a(n) = \sum_{k=0..n} C(n,k) * 2^{\frac{k*(k+1)}{2}}$.
$(\forall x \forall y\ R(x,x) \vee R(y,y)) \wedge (\forall x \exists^{=1} y\ \neg R(x,y))$	A7778	$a(n) = n^{n+1}$.
$(\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x)) \wedge (\forall x\ R(x,x) \vee P(x) \vee P_1((x))$	A10845	$a(n) = 3 * n * a(n-1) + 1$, $a(0) = 1$.
$(\forall x \exists y\ P(x) \vee R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x)) \wedge (\exists^{=1} x\ \neg P(x)) \wedge (\exists^{=1} x\ \neg R(x,x))$	A11379	$a(n) = n^2 * (n+1)$.
$(\forall x\ P(x) \vee P_1((x) \vee P_2(x)) \wedge (\forall x\ \neg P(x) \vee P_3(x))$	A11557	Powers of 10: $a(n) = 10^n$.
$(\forall x\ \neg R(x,x)) \wedge (\forall x \exists y\ R(x,y) \vee P(x) \vee P_1((y)) \wedge (\forall x \forall y\ \neg R(x,y) \vee R(x,x))$	A20515	$a(n) = 4^n - 2^n + 1$.
$(\forall x \exists y\ P(x) \vee P_1((y)) \wedge (\exists x\ P(x) \vee P_2(x) \vee P_3(x))$	A20518	10th cyclotomic polynomial evaluated at powers of 2.
$(\exists x\ P(x) \vee P_1((x)) \wedge (\exists x\ \neg P(x) \vee P_2(x))$	A20540	$a(n) = 8^{n+1} - 2^{n+2}$.
$(\forall x \forall y\ P(x) \vee P(y) \vee P_1((x) \vee \neg P_1((y))$	A27649	a(n) = $2 * (3^n) - 2^n$.
$(\forall x \forall y\ P(x) \vee P_1((y)) \wedge (\forall x \forall y\ \neg P(x) \vee P_2(y))$	A33484	$a(n) = 3 * 2^n - 2$.
$(\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x)) \wedge (\forall x \forall y\ R(x,x) \vee P(x) \vee \neg P(y))$	A33540	$a(n+1) = n * (a(n) + 1)$ for $n \geqslant 1$, $a(1) = 1$.
$(\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x\ R(x,x) \vee P(x)) \wedge (\forall x\ \neg R(x,x)) \wedge (\exists^{=1} x \forall y\ \neg R(y,x))$	A37184	Functional digraphs with 1 node not in the image.
$(\exists x R(x,x)) \wedge (\exists x \exists y R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x))$	A45531	Number of sticky functions: endofunctions of $[n]$ having a fixed point.
$(\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x)) \wedge (\exists x \exists y\ R(x,y) \vee P(x) \vee P_1((x))$	A47053	$a(n) = 4^n * n!$.
$(\forall x \forall y\ R(x,y) \vee R(y,x))$	A47656	$a(n) = 3^{\frac{n^2-n}{2}}$.

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x \forall y\ P(x) \lor R(x,y)) \land (\forall x \forall y\ \neg P(x) \lor R(y,x))$	A47863	Number of labeled graphs with 2-colored nodes where black nodes are only connected to white nodes and vice versa.
$(\forall x \exists y\ \neg R(x,y)) \land (\forall x \exists y\ \neg R(y,x))$	A48291	Number of $0, 1$ n X n matrices with no zero rows or columns.
$(\forall x\ P(x) \lor P_1((x) \lor P_2(x)) \land (\forall x \forall y\ \neg P(x) \lor \neg P_1((y))$	A48473	$a(0) = 1, a(n) = 3 * a(n-1) + 2$; $a(n) = 2 * 3^n - 1$.
$(\exists x\ \neg R(x,x)) \land (\forall x \exists^{=1} y\ R(x,y))$	A48861	$a(n) = n^n - 1$.
$(\forall x\ R(x,x)) \land (\forall x \exists y\ R(x,y))$	A53763	$a(n) = 2^{n^2-n}$.
$(\forall x \forall y\ R(x,y) \lor R_1(x,y)) \land (\exists x R(x,x)) \land (\forall x\ \neg R_1(x,x))$	A53764	$a(n) = 3^{n^2-n}$.
$(\forall x \exists y\ \neg R(y,x))$	A55601	Number of n X n binary matrices with no zero rows.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x\ R(x,x) \lor P(x)) \land (\exists x\ \neg P(x))$	A55869	$a(n) = (n+1)^n - n^n$.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\exists^{=1} x\ R(x,x))$	A55897	$a(n) = n * (n-1)^{n-1}$.
$(\forall x\ P(x) \lor P_1((x)) \land (\exists x\ P(x)) \land (\exists x\ P_1((x))$	A58481	$a(n) = 3^n - 2$.
$(\forall x\ P(x) \lor P_1((x)) \land (\exists x\ \neg P(x)) \land (\exists^{=1} x\ \neg P_1((x))$	A58877	Number of labeled acyclic digraphs with n nodes containing exactly $n-1$ points of in-degree zero.
$(\forall x \forall y\ P(x) \lor P_1((y)) \land (\exists x\ \neg P(x) \lor P_2(x))$	A59153	$a(n) = 2^{n+2} * (2^{n+1} - 1)$.
$(\forall x\ \neg R(x,x) \lor \neg R_1(x,x)) \land (\forall x\ R_1(x,x) \lor P(x))$	A60757	$a(n) = 4^{n^2}$.
$(\forall x \forall y\ R(x,y) \lor R_1(x,y)) \land (\forall x \forall y\ R(x,y) \lor R_2(x,y))$	A60758	$a(n) = 5^{n^2}$.
$(\forall x \exists y\ P(x) \lor P_1((y)) \land (\exists x\ \neg P(x))$	A60867	$a(n) = (2^n - 1)^2$.
$(\forall x \exists y\ R(x,y)) \land (\exists x \exists y\ \neg R(x,y)) \land (\forall x \exists^{=1} y\ \neg R(y,x))$	A61190	$a(n) = n^n - n$.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\forall x\ \neg R(x,x) \lor P(x)) \land (\exists^{=1} x\ \neg P(x))$	A62119	$a(n) = n! * (n-1)$.
$(\exists x \exists y P(x) \lor R(x,y)) \land (\forall x \exists^{=1} y\ \neg R(x,y))$	A62971	$a(n) = (2 * n)^n$.

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x \exists y\ R(x,y)) \land (\exists x \forall y\ \neg R(y,x) \lor \neg R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x))$	A66052	Number of permutations in the symmetric group S_n with order $\geqslant 3$.
$(\forall x \forall y\ P(x) \lor R(x,y)) \land (\forall x \exists y\ \neg P(x) \lor R(y,x)) \land (\forall x \exists^{=1} y\ R(y,x))$	A66068	$a(n) = n^n + n$.
$(\forall x \forall y\ P(x) \lor P_1((y)) \land (\exists x\ P(x)) \land (\forall x \forall y\ \neg P_1((x) \lor P_2(y))$	A68156	G.f.: $\frac{(x+2)*(x+1)}{(x-1)*(x-2)} = \sum_{n \geqslant 0} a(n) * \frac{x}{2}^n$.
$(\forall x \forall y\ P(x) \lor R(x,y)) \land (\exists x \exists y\ \neg R(x,y)) \land (\forall x \exists^{=1} y\ \neg R(y,x))$	A72034	$a(n) = \sum_{k=0..n} \mathrm{binomial}(n,k) * k^n$.
$(\forall x\ \neg R(x,x)) \land (\forall x \forall y\ \neg R(x,y) \lor R(y,x)) \land (\forall x \forall y\ \neg R(x,y) \lor P(x) \lor P(y))$	A79491	Numerator of $\sum_{k=0..n} \frac{\mathrm{binomial}(n,k)}{2^{\frac{k*(k-1)}{2}}}$.
$(\forall x \forall y\ P(x) \lor \neg P(y) \lor P_1((x) \lor P_2(y))$	A81626	$2 * 6^n - 4^n$.
$(\forall x \forall y\ R(x,y) \lor R_1(x,y)) \land (\forall x \forall y\ R(x,y) \lor \neg R_1(y,x))$	A81955	$a(n) = 2^r * 3^s$ where $r = \frac{n(n+1)}{2}$ and $s = \frac{n(n-1)}{2}$.
$(\forall x \forall y\ P(x) \lor P_1((x) \lor P_2(y)) \land (\forall x \forall y\ P(x) \lor \neg P_2(y))$	A83319	$4^n + 3^n - 2^n$.
$(\forall x \forall y\ P(x) \lor P_1((x) \lor P_2(y)) \land (\forall x\ P(x) \lor P_2(x))$	A83320	$a(n) = 5^n + 4^n - 3^n$.
$(\forall x\ P(x) \lor P_1((x)) \land (\forall x \exists y\ P(x) \lor \neg P_1((y)) \land (\exists x\ P(x))$	A83323	$a(n) = 3^n - 2^n + 1$.
$(\forall x\ \neg R(x,x)) \land (\forall x \forall y\ \neg R(x,y) \lor R(y,x)) \land (\forall x \forall y\ \neg R(x,y) \lor R_1(x,y)) \land (\forall x \forall y\ R_1(x,y) \lor \neg R_1(y,x))$	A83667	Number of antisymmetric binary relations on a set of n labeled points.
$(\forall x \forall y\ P(x) \lor P_1((y) \lor P_2(y)) \land (\forall x\ P(x) \lor \neg P_1((x))$	A85350	Binomial transform of poly-Bernoulli numbers A027649.
$(\forall x \forall y\ P(x) \lor P(y) \lor P_1((x) \lor P_2(y))$	A85352	expansion of $\frac{1-4x}{(1-5x)(1-6x)}$.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x\ R(x,x) \lor P(x)) \land (\forall x\ P(x) \lor P_1((x))$	A85527	$a(n) = (2n+1)^n$.
$(\forall x\ \neg P(x) \lor R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(x,y))$	A85528	a(n) = $(2 * n + 1)^{n+1}$.
$(\forall x \forall y\ R(x,y) \lor R_1(x,y)) \land (\exists x R(x,x)) \land (\forall x\ \neg R_1(x,x)) \land (\forall x \exists^{=1} y\ R_1(x,y))$	A85532	$(2n)^{n+1}$.

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x\ R(x,x)) \wedge (\forall x \exists y\ \neg R(x,y)) \wedge (\forall x \exists y\ \neg R(y,x))$	A86193	Number of n X n matrices with entries in $0,1$ with no zero row, no zero column and with zero main diagonal.
$(\forall x\ \neg R(x,x)) \wedge (\forall x \exists y\ R(x,y))$	A86206	Number of n X n matrices with entries in $0,1$ with no zero row and with zero main diagonal.
$(\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x)) \wedge (\forall x \forall y\ \neg R(x,x) \vee P(y)) \wedge (\exists^{=1} x\ P(x))$	A86325	Let $P(1) = 0$, $P(2) = 1$, $P(k) = P(k-1) + \frac{P(k-2)}{k-2}$; then $a(n) = n! * P(n)$.
$(\forall x \forall y\ P(x) \vee R(y,y)) \wedge (\exists x \exists y P(x) \vee P_1((x) \vee R(x,y))$	A88668	Number of n X n matrices over GF(2) with characteristic polynomial $x^{n-1} * (x-1)$.
$(\forall x\ P(x) \vee \neg R(x,x)) \wedge (\exists^{=1} x\ \neg P(x)) \wedge (\forall x \exists^{=1} y\ R(x,y))$	A89205	$a(n) = n^n * (n-1)$.
$(\forall x \exists y\ P(x) \vee P_1((y)) \wedge (\forall x \forall y\ P_1((x) \vee P_2(y))$	A92440	$a(n) = 2^{(n+1)} - 2^{n+1} + 1$.
$(\forall x \forall y\ P(x) \vee \neg P(y)) \wedge (\forall x \forall y\ P(x) \vee P_1((x) \vee P_2(y))$	A93069	$a(n) = (2^n + 1)^2 - 2$.
$(\forall x \forall y\ R(x,y) \vee \neg R(y,x)) \wedge (\forall x \forall y\ R(x,x) \vee P(y)) \wedge (\exists^{=1} x\ \neg R(x,x))$	A95340	Total number of nodes in all labeled graphs on n nodes.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\forall x \exists^{=1} y\ \neg R(y,x)) \wedge (\exists^{=1} x \forall y R(x,y))$	A98916	Permanent of the n X n (0,1)-matrices with ij-th entry equal to zero iff $(i = 1, j = 1)$,$(i = 1, j = n)$,$(i = n, j = 1)$ and $(i = n, j = n)$.
$(\forall x \forall y\ R(x,y) \vee \neg R(y,x)) \wedge (\forall x \forall y\ R(x,y) \vee R_1(x,y))$	A99168	$a(n) = 3^n * 5^{binomial(n,2)}$.
$(\forall x \forall y\ P(x) \vee P_1((x) \vee P_2(y)) \wedge (\forall x \forall y\ \neg P(x) \vee P_2(y))$	A99393	$a(n) = 4^n + 2^n - 1$.
$(\forall x\ P(x) \vee P_1((x)) \wedge (\forall x \exists y\ \neg P(x) \vee \neg P_1((y)) \wedge (\exists x\ \neg P(x))$	A101052	Number of preferential arrangements of n labeled elements when only $k \leqslant 3$ ranks are allowed.
$(\forall x \forall y\ R(x,y) \vee R_1(x,y)) \wedge (\forall x \forall y\ \neg R(x,y) \vee \neg R(y,x))$	A109345	$a(n) = 5^{\frac{n^2-n}{2}}$.
$(\forall x\ \neg R(x,x)) \wedge (\forall x \forall y\ \neg R(x,y) \vee R_1(x,y)) \wedge (\forall x \forall y\ R_1(x,y) \vee R(y,x))$	A109354	$a(n) = 6^{\frac{n^2-n}{2}}$.
$(\forall x\ \neg R(x,x)) \wedge (\forall x \forall y\ \neg R(x,y) \vee R(y,x) \vee R_1(x,y)) \wedge (\forall x \forall y\ R_1(x,y) \vee R(x,y))$	A109493	$a(n) = 7^{\frac{n^2-n}{2}}$.

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x\ \neg R(x,x)) \wedge (\forall x \forall y\ \neg R(x,y) \vee R_1(x,y)) \wedge (\forall x \forall y\ R_1(x,y) \vee R_1(y,x))$	A109966	$a(n) = 8^{\frac{n^2-n}{2}}$.
$(\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x \forall y\ R(x,x) \vee \neg R(y,y))$	A110567	$a(n) = n^{n+1} + 1$.
$(\forall x \forall y\ R(x,y) \vee \neg R(y,x)) \wedge (\exists x R(x,x))$	A122743	Number of normalized polynomials of degree n in $GF(2)[x,y]$.
$(\forall x \forall y\ R(x,y) \vee \neg R(y,x)) \wedge (\exists x \exists y R(x,y))$	A126883	$a(n) = (2^0) * (2^1) * (2^2) * (2^3)...(2^n) - 1 = 2^{T(n)} - 1$ where $T(n) = A000217(n)$ is the n-th triangular number.
$(\forall x\ \neg R(x,x)) \wedge (\forall x \forall y\ \neg R(x,y) \vee R(y,x)) \wedge (\forall x \forall y\ \neg R(x,y) \vee P(x)) \wedge (\forall x \forall y\ P(x) \vee \neg P(y))$	A126884	$a(n) = (2^0) * (2^1) * (2^2) * (2^3)...(2^n) + 1 = 2^{T_n} + 1$ (cf. A000217).
$(\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x)) \wedge (\forall x \forall y\ R(x,x) \vee P(y))$	A127986	$a(n) = n! + 2^n - 1$.
$(\exists^{=1} x\ \neg R(x,x))$	A128406	$a(n) = (n+1) * 2^{n*(n+1)}$.
$(\forall x \exists y\ P(x) \vee P_1((y)) \wedge (\forall x \exists y\ \neg P_1((x) \vee P_2(y)) \wedge (\exists x\ \neg P_2(x))$	A128831	Number of n-tuples where each entry is chosen from the subsets of $1,2,3$ such that the intersection of all n entries is empty.
$(\forall x \forall y\ R(x,y) \vee \neg R(y,x)) \wedge (\forall x\ R(x,x) \vee P(x))$	A132727	$a(n) = 3 * 2^{n-1} * a(n-1)$ with $a(0) = 1$.
$(\forall x \forall y\ P(x) \vee P_1((x) \vee R(x,y)) \wedge (\forall x\ P(x) \vee P_1((x))$	A133460	$3^n * 2^{n^2}$.
$(\forall x \exists y\ R(x,y)) \wedge (\exists x R(x,x)) \wedge (\exists x\ \neg R(x,x)) \wedge (\forall x \exists^{=1} y\ R(y,x))$	A133798	$a(n) = A002467(n) - 1$.
$(\forall x \forall y\ P(x) \vee P(y) \vee R(x,y)) \wedge (\forall x\ R(x,x))$	A134485	Row sums of triangle $A134484(n,k) = 2^{[n(n-1)-k(k-1)]} * C(n,k)$.
$(\forall x \forall y\ P(x) \vee P_1((x) \vee \neg P_1((y)) \wedge (\forall x\ P(x) \vee P_2(x))$	A135160	$a(n) = 5^n + 3^n - 2^n$.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\forall x \forall y\ P(x) \vee R(y,x))$	A135748	$a(n) = \sum_{k=0..n} \mathrm{binomial}(n,k) * 2^{k^2}$.
$(\forall x \forall y\ P(x) \vee R(x,y) \vee R(y,x)) \wedge (\forall x \forall y\ \neg P(x) \vee \neg R(x,y))$	A135755	$a(n) = \sum_{k=0..n} C(n,k) * 3^{[\frac{k*(k-1)}{2}]}$.
$(\forall x \forall y\ R(x,y) \vee \neg R(x,x)) \wedge (\forall x \forall y\ \neg R(x,x) \vee R(y,x))$	A135756	$a(n) = \sum_{k=0..n} C(n,k) * 2^{k*(k-1)}$.
$(\forall x \forall y\ \neg P(x) \vee \neg R(y,x))$	A136516	$a(n) = (2^n + 1)^n$.
$(\forall x \exists y\ P(x) \vee P_1((y)) \wedge (\exists x\ P(x) \vee \neg P_1((x) \vee P_2(x))$	A145641	Numbers whose binary representation is the concatenation of n 1's, n 0's and n 1's.

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x \forall y\ P(x) \vee \neg P(y) \vee P_1((x)) \wedge (\forall x\ P(x) \vee P_2(x))$	A155588	$a(n) = 5^n + 2^n - 1^n$.
$(\forall x\ P(x) \vee P_1((x)) \wedge (\forall x \exists y\ \neg P(x) \vee \neg P_1((y) \vee P_2(y))$	A155597	$a(n) = 6^n - 2^n + 1$.
$(\forall x \exists y\ P(x) \vee P_1((y) \vee P_2(y))$	A155599	$a(n) = 8^n - 2^n + 1^n$.
$(\forall x \forall y\ P(x) \vee \neg P(y) \vee P_1((x)) \wedge (\forall x\ \neg P_1((x) \vee P_2(x))$	A155602	$4^n + 3^n - 1$.
$(\forall x\ P(x) \vee P_1((x)) \wedge (\forall x \exists y\ P(x) \vee P_2(y)) \wedge (\exists x\ P(x))$	A155611	$6^n - 3^n + 1$.
$(\forall x\ P(x) \vee P_1((x) \vee P_2(x)) \wedge (\exists x \forall y P(x) \vee \neg P_1((y))$	A155612	$7^n - 3^n + 1$.
$(\forall x \exists y\ P(x) \vee P_1((y)) \wedge (\exists x\ P(x) \vee P_2(x))$	A155629	$a(n) = 8^n - 4^n + 1^n$.
$(\forall x \exists^{=1} y\ R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x)) \wedge (\forall x \forall y\ R(x,y) \vee R_1(x,y)) \wedge (\exists^{=1} x\ \neg R_1(x,x))$	A161937	The number of indirect isometries that are derangements of the $(n-1)$-dimensional facets of an n-cube.
$(\forall x \forall y\ R(x,x) \vee \neg R(y,x))$	A165327	E.g.f: $\sum_{n \geqslant 0} 2^{n(n-1)} * \exp(2^n * x) * \frac{x^n}{n!}$.
$(\exists x\ P(x) \vee P_1((x)) \wedge (\exists x\ \neg P(x)) \wedge (\exists x\ P(x) \vee \neg P_1((x))$	A170940	$4^n - 2^n - 2$.
$(\forall x \forall y\ P(x) \vee P_1((y)) \wedge (\exists x\ P(x)) \wedge (\forall x\ \neg P_1((x) \vee P_2(x))$	A171270	a(n) is the only number m such that $m = pi(1^{\frac{1}{n}}) + pi(2^{\frac{1}{n}}) + ... + pi(m^{\frac{1}{n}})$.
$(\forall x \exists y\ \neg R(x,y) \vee \neg R(y,x)) \wedge (\exists x \exists y\ \neg R(x,y))$	A173403	Inverse binomial transform of A002416.
$(\exists x \forall y\ R(x,y) \vee \neg R(y,y)) \wedge (\exists x \exists y\ \neg R(x,y)) \wedge (\forall x \exists^{=1} y\ R(y,x))$	A176043	$a(n) = (2*n-1)*(n-1)^{n-1}$.
$(\forall x \exists y\ \neg P(x) \vee R(x,y))$	A180602	$(2^{n+1} - 1)^n$.
$(\forall x \exists y\ P(x) \vee P_1((y)) \wedge (\forall x \exists y\ P_1((x) \vee P(y))$	A191341	$a(n) = 4^n - 2*2^n + 3$.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\forall x \forall y\ R(x,y) \vee P_1((y))$	A196460	e.g.f.: $A(x) = \sum_{n \geqslant 0} (1+2^n)^n * \exp((1+2^n)*x) * \frac{x^n}{n!}$.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\forall x \forall y\ \neg R(x,y) \vee R_1(x,y))$	A202989	E.g.f: $\sum_{n \geqslant 0} 3^{(}n^2) * \exp(3^n * x) * \frac{x^n}{n!}$.
$(\forall x\ P(x) \vee P_1((x)) \wedge (\forall x \exists y\ \neg P(x) \vee R(x,y))$	A202990	E.g.f: $\sum_{n \geqslant 0} 3^n * 2^{n^2} * \exp(-2 * 2^n * x) * \frac{x^n}{n!}$.
$(\forall x \forall y\ R(x,y) \vee R_1(x,y)) \wedge (\forall x \exists y\ R(x,y)) \wedge (\forall x \exists y\ \neg R(x,y) \vee \neg R_1(x,y))$	A202991	E.g.f: $\sum_{n \geqslant 0} 3^{n^2} * \exp(-2 * 3^n * x) * \frac{x^n}{n!}$.

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x \forall y\ R(x,y) \lor R(y,x)) \land (\exists x \exists y\ \neg R(x,y))$	A206601	$3^{\frac{n(n+1)}{2}} - 1$.
$(\forall x \exists^{=1} y\ R(x,y)) \land (\forall x \exists^{=1} y\ R(y,x)) \land (\forall x \forall y\ \neg R(x,x) \lor R(x,y) \lor \neg R(y,y))$	A212291	Number of permutations of n elements with at most one fixed point.
$(\forall x\ P(x) \lor P_1((x)) \land (\forall x \exists y\ \neg P(x) \lor P_2(y)) \land (\exists x\ \neg P_2(x))$	A212850	Number of n X 3 arrays with rows being permutations of $0..2$ and no column j greater than column j-1 in all rows.
$(\forall x \forall y\ R(x,y) \lor \neg R(y,x)) \land (\forall x \forall y\ R(x,x) \lor \neg R(y,y))$	A217994	$a(n) = 2^{\frac{2+n+n^2}{2}}$.
$(\forall x\ \neg R(x,x)) \land (\forall x \forall y\ \neg R(x,y) \lor R(y,x)) \land (\forall x \forall y\ \neg R(x,y) \lor P(x) \lor \neg P(y))$	A226773	Number of ways to select a simple labeled graph on n nodes and then select a subset of its connected components.
$(\forall x \forall y\ R(x,y) \lor R_1(x,y)) \land (\forall x \exists y\ R(x,y)) \land (\forall x \exists y\ R(y,x)) \land (\exists x \exists y R(x,y))$	A230879	Number of 2-packed n X n matrices.
$(\forall x \exists y\ R(y,x) \lor R_1(x,y))$	A241098	$(4^n - 1)^n$.
$(\forall x\ \neg R(x,x)) \land (\forall x \forall y\ \neg R(x,y) \lor P(x) \lor P_1((x)) \land (\forall x \forall y\ P(x) \lor \neg R(y,x))$	A243918	$a(n) = \sum_{k=0..n} \mathrm{binomial}(n,k) * (1+2^k)^k$.
$(\exists x \forall y\ R(x,y) \lor \neg R(y,x)) \land (\forall x\ R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(x,y))$	A246189	Number of endofunctions on $[n]$ where the smallest cycle length equals 2.
$(\forall x \forall y\ P(x) \lor R(x,y)) \land (\forall x \exists y\ P(x) \lor \neg R(y,x))$	A251183	$a(n) = \sum_{k=0..n} \mathrm{binomial}(n,k) * (-1)^{n-k} * (2^k+1)^k$.
$(\forall x \forall y\ P(x) \lor R(x,y)) \land (\forall x \forall y\ R(x,y) \lor P_1((x))$	A251657	$a(n) = (2^n + 3)^n$.
$(\forall x \forall y\ P(x) \lor P_1((y)) \land (\exists x\ P(x) \lor P_2(x)) \land (\exists x\ P_1((x))$	A267816	Decimal representation of the n-th iteration of the "Rule 221" elementary cellular automaton starting with a single ON (black) cell.
$(\forall x \exists y\ R(x,y) \lor R_1(x,y)) \land (\forall x\ R(x,x)) \land (\forall x \exists^{=1} y\ \neg R(x,y)) \land (\forall x \exists^{=1} y\ \neg R_1(x,y))$	A281997	$a(n) = (n-1)^n * n^n$.
$(\forall x \exists y\ R(x,y)) \land (\forall x \exists y\ \neg R(x,y)) \land (\forall x \exists y\ R(y,x)) \land (\forall x \exists y\ \neg R(y,x))$	A283624	Number of $0,1$ n X n matrices with no rows or columns in which all entries are the same.

Table 3 – continued from previous page

Sentence	OEIS ID	OEIS name
$(\forall x \exists y\ R(x,y)) \wedge (\exists x \forall y\ \neg R(y,x) \vee P(y)) \wedge (\forall x \forall y\ P(x) \vee \neg P(y))$	A287065	Number of dominating sets on the n X n rook graph.
$(\exists x \forall y\ R(x,y) \vee P(y)) \wedge (\forall x \exists^{=1} y\ R(y,x)) \wedge (\exists^{=1} x\ \neg R(x,x))$	A317637	$a(n) = n * (n+1) * (n+3)$.
$(\forall x \forall y\ R(x,y) \vee \neg R(y,x)) \wedge (\forall x \exists y\ R(x,y))$	A322661	Number of graphs with loops spanning n labeled vertices.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\forall x \exists y\ \neg P(x) \vee \neg R(y,x))$	A324306	G.f.: $\sum_{n \geqslant 0} \frac{(2^n+1)^n * x^n}{(1+2^n*x)^{n+1}}$.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\forall x \forall y\ R(x,y) \vee R_1(x,y))$	A326555	$a(n) = (2^n + 3^n)^n$ for $n \geqslant 0$.
$(\forall x\ P(x) \vee P_1((x) \vee P_2(x)) \wedge (\exists x\ P(x)) \wedge (\exists x\ P_1((x))$	A337418	Number of sets (in the Hausdorff metric geometry) at each location between two sets defined by a complete bipartite graph $K(3,n)$ (with n at least 3) missing two edges, where the removed edges are not incident to the same vertex in the 3 point part but are incident to the same vertex in the other part.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\exists x\ P(x)) \wedge (\forall x \exists y\ \neg R(y,x))$	A337527	G.f.: $\sum_{n \geqslant 0} \frac{(2^n+1)^n * x^n}{(1+(2^n+1)*x)^{n+1}}$.
$(\forall x\ P(x) \vee P_1((x)) \wedge (\forall x \forall y\ \neg P(x) \vee R(x,y))$	A337851	$a(n) = (2^n + 2)^n$.
$(\forall x \forall y\ P(x) \vee R(x,y)) \wedge (\forall x \exists y\ R(x,y) \vee P_1((x))$	A337852	$a(n) = (2^{n+1} + 1)^n$.

BIBLIOGRAPHY

[1] CB Abhilash and Kavi Mahesh. “Ontology-based data interestingness: A state-of-the-art review.” In: *Natural Language Processing Journal* (2023), p. 100021.

[2] Michael H. Albert, Christian Bean, Anders Claesson, Émile Nadeau, Jay Pantone, and Henning Ulfarsson. *Combinatorial Exploration: An algorithmic framework for enumeration*. https://arxiv.org/abs/2202.07715. 2022. arXiv: 2202.07715 `[math.CO]`.

[3] Alchemy. *Alchemy: Open Source AI*. [Online; accessed 09-September-2018]. 2018. URL: http://alchemy.cs.washington.edu/.

[4] Sarabjot S Anand, David A Bell, and John G Hughes. “The role of domain knowledge in data mining.” In: *Proceedings of the fourth international conference on Information and knowledge management*. 1995, pp. 37–43.

[5] Akash Anil, Víctor Gutiérrez-Basulto, Yazmín Ibañéz-García, and Steven Schockaert. “Inductive Knowledge Graph Completion with GNNs and Rules: An Analysis.” In: *arXiv preprint arXiv:2308.07942* (2023).

[6] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. “Hinge-Loss Markov Random Fields and Probabilistic Soft Logic.” In: *Journal of Machine Learning Research* 18.109 (2017), pp. 1–67.

[7] Molood Barati, Quan Bai, and Qing Liu. “SWARM: an approach for mining semantic association rules from semantic web data.” In: *PRICAI 2016: Trends in Artificial Intelligence: 14th Pacific Rim International Conference on Artificial Intelligence, Phuket, Thailand, August 22-26, 2016, Proceedings 14*. Springer. 2016, pp. 30–43.

[8] Jáchym Barvínek, Timothy van Bremen, Yuyi Wang, Filip Železný, and Ondřej Kuželka. “Automatic Conjecturing of P-Recursions Using Lifted Inference.” In: *Inductive Logic Programming - 30th International Conference, ILP 2021, Proceedings*. Ed. by Nikos Katzouris and Alexander Artikis. Vol. 13191. Lecture Notes in Computer Science. Springer, 2021, pp. 17–25. DOI: 10.1007/978-3-030-97454-1_2. URL: https://doi.org/10.1007/978-3-030-97454-1%5C_2.

[9] 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. Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 313–328. ISBN: 9781450327572. DOI: 10.1145/2745754.2745760. URL: https://doi.org/10.1145/2745754.2745760.

[10] Salem Benferhat, Claudette Cayrol, Didier Dubois, Jérôme Lang, and Henri Prade. “Inconsistency Management and Prioritized Syntax-Based Entailment.” In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. 1993, pp. 640–647.

[11] Abdelaziz Berrado and George C Runger. “Using metarules to organize and group discovered association rules.” In: *Data mining and knowledge discovery* 14 (2007), pp. 409–431.

[12] Patrick Betz, Stefan Lüdtke, Christian Meilicke, and Heiner Stuckenschmidt. “On the aggregation of rules for knowledge graph completion.” In: *arXiv preprint arXiv:2309.00306* (2023).

[13] Patrick Betz, Christian Meilicke, and Heiner Stuckenschmidt. “Supervised knowledge aggregation for knowledge graph completion.” In: *European Semantic Web Conference*. Springer. 2022, pp. 74–92.

[14] Mikhail Bongard. “Pattern Recognition.” In: *Spartan Books* (1970).

[15] Egon Börger, Erich Grädel, and Yuri Gurevich. “The classical decision problem.” In: Springer Science & Business Media, 2001, p. 48.

[16] Timothy van Bremen, Vincent Derkinderen, Shubham Sharma, Subhajit Roy, and Kuldeep S. Meel. “Symmetric Component Caching for Model Counting on Combinatorial Instances.” In: *Thirty-Fifth AAAI Conference on Artificial Intelligence*. AAAI Press, 2021, pp. 3922–3930.

[17] Timothy van Bremen and Ondřej Kuželka. “Faster lifting for two-variable logic using cell graphs.” In: *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*. Vol. 161. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 1393–1402. URL: https://proceedings.mlr.press/v161/bremen21a.html.

[18] Wray L. Buntine. “Generalized Subsumption and Its Applications to Induction and Redundancy.” In: *Artif. Intell.* 36.2 (1988), pp. 149–176.

[19] Chandra Chekuri and Anand Rajaraman. “Conjunctive query containment revisited.” In: *Theor. Comput. Sci.* 239.2 (2000), pp. 211–229.

[20] Lihan Chen, Sihang Jiang, Jingping Liu, Chao Wang, Sheng Zhang, Chenhao Xie, Jiaqing Liang, Yanghua Xiao, and Rui Song. “Rule mining over knowledge graphs via reinforcement learning.” In: *Knowledge-Based Systems* 242 (2022), p. 108371.

[21] Yang Chen, Sean Goldberg, Daisy Zhe Wang, and Soumitra Siddharth Johri. “Ontological pathfinding.” In: *Proceedings of the 2016 International Conference on Management of Data*. 2016, pp. 835–846.

[22] Yang Chen, Daisy Zhe Wang, and Sean Goldberg. “ScaLeKB: scalable learning and inference over large knowledge bases.” In: *The VLDB Journal* 25.6 (2016), pp. 893–918.

[23] Simon Colton. “Refactorable numbers-a machine invention.” In: *Journal of Integer Sequences* 2.99.1 (1999), p. 2.

[24] Simon Colton. “The HR program for theorem generation.” In: *International Conference on Automated Deduction*. Springer. 2002, pp. 285–289.

[25] Simon Colton, Alan Bundy, and Toby Walsh. “Automatic invention of integer sequences.” In: *AAAI/IAAI*. 2000, pp. 558–563.

[26] Andrew Cropper, Rolf Morel, and Stephen H Muggleton. “Learning Higher-Order Programs through Predicate Invention.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 09. 2020, pp. 13655–13658.

[27] Stéphane D’Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and Francois Charton. “Deep symbolic regression for recurrence prediction.” In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 4520–4536. URL: https://proceedings.mlr.press/v162/d-ascoli22a.html.

[28] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, et al. “Advancing mathematics by guiding human intuition with AI.” In: *Nature* 600.7887 (2021), pp. 70–74.

[29] J. Davis, E. Burnside, I.C. Dutra, D. Page, and V. Santos Costa. “An Integrated Approach to Learning Bayesian Networks of Rules.” In: *16th ECML*. Springer, 2005, pp. 84–95.

[30] Jesse Davis and Mark Goadrich. “The relationship between Precision-Recall and ROC curves.” In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[31] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. “ProbLog: A Probabilistic Prolog and Its Application in Link Discovery.” In: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*. 2007, pp. 2462–2467.

[32] Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[33] Luc Dehaspe and Luc De Raedt. “Mining Association Rules in Multiple Relations.” In: *Inductive Logic Programming, 7th International Workshop, ILP-97*. 1997, pp. 125–132.

[34] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. “Knowledge vault: A web-scale approach to probabilistic knowledge fusion.” In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 601–610.

[35] D. Dubois, J. Lang, and H. Prade. “Possibilistic logic.” In: *Handbook of Logic in Artificial Intelligence and Logic Programming*. Ed. by D. Nute D. Gabbay C. Hogger J. Robinson. Vol. 3. Oxford University Press, 1994, pp. 439–513.

[36] Sašo Džeroski. *Relational data mining*. Springer, 2010.

[37] Stefano Ferilli, Nicola Fanizzi, Nicola Di Mauro, and Teresa MA Basile. “Efficient θ-subsumption under object identity.” In: *AI*IA Workshop, 2002)*. 2002, pp. 59–68.

[38] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. “Fast rule mining in ontological knowledge bases with AMIE+.” In: *The VLDB Journal* 24.6 (2015), pp. 707–730.

[39] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. “AMIE: association rule mining under incomplete evidence in ontological knowledge bases.” In: *Proceedings of the 22nd International Conference on World Wide Web*. 2013, pp. 413–422.

[40] Thibault Gauthier, Miroslav Olšák, and Josef Urban. *Alien Coding*. 2023. DOI: 10.48550/ARXIV.2301.11479. URL: https://arxiv.org/abs/2301.11479.

[41] Hector Geffner and Judea Pearl. “Conditional Entailment: Bridging two Approaches to Default Reasoning.” In: *Artif. Intell.* 53.2-3 (1992), pp. 209–244.

[42] Vibhav Gogate and Pedro M. Domingos. “Probabilistic Theorem Proving.” In: *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*. Ed. by Fábio Gagliardi Cozman and Avi Pfeffer. AUAI Press, 2011, pp. 256–265. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article%5C_id=2263%5C&proceeding%5C_id=27.

[43] Erich Grädel and Martin Otto. “On logics with two variables.” In: *Theoretical computer science* 224.1-2 (1999), pp. 73–113.

[44] Timothy Hinrichs and Michael Genesereth. *Herbrand Logic*. Tech. rep. LG-2006-02. http://logic.stanford.edu/reports/LG-2006-02.pdf. Stanford, CA: Stanford University, 2006.

[45] Willem Jan van Hoeve. “The alldifferent Constraint: A Survey.” In: *CoRR* cs.PL/0105015 (2001). URL: http://arxiv.org/abs/cs.PL/0105015.

[46] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. “An apriori-based algorithm for mining frequent substructures from graph data.” In: *Principles of Data Mining and Knowledge Discovery: 4th European Conference, PKDD 2000 Lyon, France, September 13–16, 2000 Proceedings 4*. Springer. 2000, pp. 13–23.

[47] Moa Johansson and Nicholas Smallbone. “Conjectures, tests and proofs: An overview of theory exploration.” In: *arXiv preprint arXiv:2109.03721* (2021).

[48] Brendan Juba. “Implicit Learning of Common Sense for Reasoning.” In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*. 2013, pp. 939–946.

[49] Kristian Kersting and Luc De Raedt. “Towards combining inductive logic programming with Bayesian networks.” In: *International Conference on Inductive Logic Programming*. Springer. 2001, pp. 118–131.

[50] Stanley Kok and Pedro Domingos. “Learning the Structure of Markov Logic Networks.” In: *Proceedings of the 22nd ICML*. Bonn, Germany: ACM Press, 2005, pp. 441–448.

[51] Stanley Kok and Pedro Domingos. “Statistical predicate invention.” In: *Proceedings of the 24th International Conference on Machine Learning*. 2007, pp. 433–440.

[52] MJ Kronenburg. "The binomial coefficient for negative arguments." In: *arXiv preprint arXiv:1105.3689* (2011).

[53] Ondrej Kuzelka, Yuyi Wang, Jesse Davis, and Steven Schockaert. "PAC-Reasoning in Relational Domains." In: *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*. 2018, pp. 927–936.

[54] Ondřej Kuželka. "Weighted First-Order Model Counting in the Two-Variable Fragment With Counting Quantifiers." In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 1281–1307.

[55] Ondřej Kuželka, Jesse Davis, and Steven Schockaert. "Induction of Interpretable Possibilistic Logic Theories from Relational Data." In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. 2017, pp. 1153–1159.

[56] Ondřej Kuželka and Filip Železný. "A Restarted Strategy for Efficient Subsumption Testing." In: *Fundam. Inform.* 89.1 (2008), pp. 95–109.

[57] Ondřej Kuželka and Filip Železný. "Block-wise construction of tree-like relational features with monotone reducibility and redundancy." In: *Mach. Learn.* 83.2 (2011), pp. 163–192.

[58] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. "Canonical tensor decomposition for knowledge base completion." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2863–2872.

[59] Jonathan Lajus, Luis Galárraga, and Fabian Suchanek. "Fast and exact rule mining with AMIE 3." In: *The Semantic Web: 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings 17*. Springer. 2020, pp. 36–52.

[60] Ni Lao, Tom Mitchell, and William Cohen. "Random walk inference and learning in a large scale knowledge base." In: *Proceedings of the 2011 conference on empirical methods in natural language processing*. 2011, pp. 529–539.

[61] D. Le Berre and A. Parrain. "The SAT4J library, release 2.2." In: *Journal on Satisfiability, Boolean Modeling and Computation* 7 (2010), pp. 50–64.

[62] Jens Lehmann and Lorenz Bühmann. "ORE-a tool for repairing and enriching knowledge bases." In: (2010), pp. 177–193.

[63] Douglas Bruce Lenat. *AM: an artificial intelligence approach to discovery in mathematics as heuristic search.* Stanford University, 1976.

[64] Leo Liberti and Franco Raimondi. "The secret santa problem." In: *International Conference on Algorithmic Applications in Management*. Springer. 2008, pp. 271–279.

[65] *Loda language*. [Online; accessed 01-January-2024]. 2024. URL: https://loda-lang.org/.

[66] Ali Mirza Mahmood and Mrithyumjaya Rao Kuppa. "A novel pruning approach using expert knowledge for data-specific pruning." In: *Engineering with Computers* 28 (2012), pp. 21–30.

[67] Jon Maiga. *Sequence Machine*. https://sequencedb.net. [Online; accessed 01-January-2024]. 2024.

[68] Donato Malerba. "Learning Recursive Theories in the Normal ILP Setting." In: *Fundam. Inform.* 57.1 (2003), pp. 39–77.

[69] František Malinka, Filip Železný, and Jiří Kléma. "Finding semantic patterns in omics data using concept rule learning with an ontology-based refinement operator." In: *BioData mining* 13 (2020), pp. 1–22.

[70] J. Mallen and M. Bramer. "CUPID-an iterative knowledge discovery framework." In: *Expert Systems* 94 (1994).

[71] Jérôme Maloberti and Michèle Sebag. "Fast Theta-Subsumption with Constraint Satisfaction Algorithms." In: *Machine Learning* 55.2 (2004), pp. 137–174.

[72] W. McCune. "Prover9 and Mace4." 2010. URL: http://www.cs.unm.edu/~mccune/prover9/.

[73] Brendan D McKay and Adolfo Piperno. "Practical graph isomorphism, II." In: *Journal of symbolic computation* 60 (2014), pp. 94–112.

[74] Christian Meilicke, Patrick Betz, and Heiner Stuckenschmidt. "Why a naive way to combine symbolic and latent knowledge base completion works surprisingly well." In: *3rd Conference on Automated Knowledge Base Construction*. 2021.

[75] Christian Meilicke, Melisachew Wudage Chekol, Patrick Betz, Manuel Fink, and Heiner Stuckeschmidt. "Anytime bottom-up rule learning for large-scale knowledge graph completion." In: *The VLDB Journal* (2023), pp. 1–31.

[76] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. "Anytime Bottom-Up Rule Learning for Knowledge Graph Completion." In: *IJCAI*. 2019, pp. 3137–3143.

[77] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. "Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion." In: *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I 17*. Springer. 2018, pp. 3–20.

[78] Ryszard S Michalski. "A theory and methodology of inductive learning." In: *Machine learning*. Elsevier, 1983, pp. 83–134.

[79] Lilyana Mihalkova and Raymond J Mooney. "Bottom-up learning of Markov logic network structure." In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 625–632.

[80] George A Miller. "WordNet: a lexical database for English." In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[81] Omar Montano-Rivas, Roy McCasland, Lucas Dixon, and Alan Bundy. "Scheme-based theorem discovery and concept invention." In: *Expert systems with applications* 39.2 (2012), pp. 1637–1646.

[82] Mahesh Motwani, JL Rana, and RC Jain. "Use of domain knowledge for fast mining of association rules." In: *Proceedings of the International Multi-Conference of Engineers and Computer Scientists*. 2009.

[83] Stephen Muggleton. “Inductive logic programming.” In: *New generation computing* 8.4 (1991), pp. 295–318.

[84] Stephen Muggleton. “Inverse Entailment and Progol.” In: *New Gen. Comput.* 13.3&4 (1995), pp. 245–286.

[85] Stephen Muggleton and Luc De Raedt. “Inductive logic programming: Theory and methods.” In: *The Journal of Logic Programming* 19 (1994), pp. 629–679.

[86] Gábor P Nagy and Petr Vojtěchovskỳ. “Computing with quasigroups and loops in GAP.” In: (2005).

[87] Monty Newborn. *Automated theorem proving - theory and practice*. Springer, 2001. ISBN: 978-0-387-95075-4.

[88] Siegfried Nijssen and Joost N. Kok. “Faster Association Rules for Multiple Relations.” In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*. Ed. by Bernhard Nebel. Morgan Kaufmann, 2001, pp. 891–896.

[89] Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. “Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS.” In: *PVLDB* 4.6 (2011), pp. 373–384.

[90] Jan Noessner, Mathias Niepert, and Heiner Stuckenschmidt. “RockIt: Exploiting Parallelism and Symmetry for MAP Inference in Statistical Relational Models.” In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. 2013.

[91] OEIS Foundation Inc. *The On-Line Encyclopedia of Integer Sequences.* Published electronically at http://oeis.org. Accessed: 2023-05-23. 2023.

[92] Andrei Olaru, Claudia Marinica, and Fabrice Guillet. “Local mining of association rules with rule schemas.” In: *2009 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE. 2009, pp. 118–124.

[93] Pouya Ghiasnezhad Omran, Kerry Taylor, Sergio Rodriguez Mendez, and Armin Haller. “Active knowledge graph completion.” In: *Information Sciences* 604 (2022), pp. 267–279.

[94] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. “Scalable Rule Learning via Learning Representation.” In: *IJCAI*. 2018, pp. 2149–2155.

[95] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. “Rudik: Rule discovery in knowledge bases.” In: *Proceedings of the VLDB Endowment* 11.12 (2018), pp. 1946–1949.

[96] Simon Ott, Christian Meilicke, and Matthias Samwald. “SAFRAN: An interpretable, rule-based link prediction method outperforming embedding models.” In: *arXiv preprint arXiv:2109.08002* (2021).

[97] Igor Pak. *What is Combinatorics?* [Online; accessed 01-January-2024]. 2024. URL: https://www.math.ucla.edu/~pak/hidden/papers/Quotes/Combinatorics-quotes.htm.

[98] Nicolas Pasquier, Claude R Pasquier, Laurent Brisson, and Martine Collard. “Mining gene expression data using domain knowledge.” In: *International Journal of Software and Informatics (IJSI)* 2.2 (2008), pp. 215–231.

[99] Judea Pearl. “System Z: A Natural Ordering of Defaults with Tractable Applications to Nonmonotonic Reasoning.” In: *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge*. 1990, pp. 121–135.

[100] Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. “Completeness-aware rule learning from knowledge graphs.” In: *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I 16*. Springer. 2017, pp. 507–525.

[101] Giuseppe Pirrò. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 03. 2020, pp. 2975–2982.

[102] Gordon D Plotkin. “A note on inductive generalization.” In: *Machine intelligence* 5.1 (1970), pp. 153–163.

[103] David Poole. “First-order probabilistic inference.” In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. Ed. by Georg Gottlob and Toby Walsh. Morgan Kaufmann, 2003, pp. 985–991. URL: http://ijcai.org/Proceedings/03/Papers/142.pdf.

[104] Ian Pratt-Hartmann. “Complexity of the two-variable fragment with counting quantifiers.” In: *Journal of Logic, Language, and Information* (2005), pp. 369–395.

[105] Ian Pratt-Hartmann. “Data-complexity of the two-variable fragment with counting quantifiers.” In: *Information and Computation* 207.8 (2009), pp. 867–888.

[106] J. Ross Quinlan and R. Mike Cameron-Jones. “Induction of logic programs: FOIL and related systems.” In: *New Generation Computing* 13.3-4 (1995), pp. 287–312.

[107] Luc De Raedt. “Logical Settings for Concept-Learning.” In: *Artif. Intell.* 95.1 (1997), pp. 187–201.

[108] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. “Statistical relational artificial intelligence: Logic, probability, and computation.” In: *Synthesis lectures on artificial intelligence and machine learning* 10.2 (2016), pp. 1–189.

[109] Liva Ralaivola, Sanjay J. Swamidass, Hiroto Saigo, and Pierre Baldi. “Graph kernels for chemical informatics.” In: *Neural Netw.* 18.8 (2005), pp. 1093–1110.

[110] Jan Ramon, Samrat Roy, and Daenen Jonny. “Efficient homomorphism-free enumeration of conjunctive queries.” In: *Preliminary Papers ILP 2011*. 2011, p. 6.

[111] Matthew Richardson and Pedro Domingos. “Markov logic networks.” In: *Machine Learning* 62 (2006), pp. 107–136.

[112] Sebastian Riedel. "Improving the Accuracy and Efficiency of MAP Inference for Markov Logic." In: *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*. 2008, pp. 468–475.

[113] Tim Rocktäschel and Sebastian Riedel. "End-to-end Differentiable Proving." In: *Proceedings of the Annual Conference on Neural Information Processing Systems*. 2017, pp. 3791–3803.

[114] Edward W Schneider. "Course Modularization Applied: The Interface System and Its Implications For Sequence Control and Data Analysis." In: (1973).

[115] Gustav Šourek, Vojtěch Aschenbrenner, Filip Železný, Steven Schockaert, and Ondřej Kuželka. "Lifted Relational Neural Networks: Efficient Learning of Latent Relational Structures." In: *J. Artif. Intell. Res.* 62 (2018), pp. 69–100.

[116] Gustav Šourek, Martin Svatoš, Filip Železný, Steven Schockaert, and Ondřej Kuželka. "Stacked Structure Learning for Lifted Relational Neural Networks." In: *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*. Ed. by Nicolas Lachiche and Christel Vrain. Vol. 10759. Lecture Notes in Computer Science. Springer, 2017, pp. 140–151. DOI: 10.1007/978-3-319-78090-0_10. URL: https://doi.org/10.1007/978-3-319-78090-0%5C_10.

[117] Ashwin Srinivasan. *The Aleph Manual*. 2001.

[118] Ashwin Srinivasan and Ross D King. "Using inductive logic programming to construct structure-activity relationships." In: *Predictive toxicology of chemicals: experiences and impact of AI tools (Papers from the 1999 AAAI Spring Symposium)*. AAAI Press Menlo Park, CA. 1999, pp. 64–73.

[119] Richard P. Stanley. "What is enumerative combinatorics?" In: *Enumerative combinatorics*. Springer, 1986, pp. 1–63.

[120] Robert E Stepp and Ryszard S Michalski. "Conceptual clustering: Inventing goal-oriented classifications of structured objects." In: *Machine learning: An artificial intelligence approach* 2 (1986), pp. 471–498.

[121] Christian Strasser and G Aldo Antonelli. "Non-monotonic Logic." In: (2001).

[122] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. "Rotate: Knowledge graph embedding by relational rotation in complex space." In: *arXiv preprint arXiv:1902.10197* (2019).

[123] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. "A re-evaluation of knowledge graph completion methods." In: *arXiv preprint arXiv:1911.03903* (2019).

[124] Simon Suster, Pieter Fivez, Pietro Totis, Angelika Kimmig, Jesse Davis, Luc De Raedt, and Walter Daelemans. "Mapping probability word problems to executable representations." In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 3627–3640.

[125] 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 2023, 19th-25th August 2023, Macao, SAR, China*. ijcai.org, 2023, pp. 3338–3346. DOI: 10.24963/ijcai.2023/372. URL: https://doi.org/10.24963/ijcai.2023/372.

[126] Alireza Tamaddoni-Nezhad and Stephen Muggleton. "The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause." In: *Machine Learning* 76.1 (2009), pp. 37–72.

[127] Komal Teru, Etienne Denis, and Will Hamilton. "Inductive relation prediction by subgraph reasoning." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9448–9457.

[128] *The Fourth International Conference*. [Online; accessed 09-September-2018]. AIPS 1998 competition. 2018. URL: http://www.cs.cmu.edu/~aips98/.

[129] Jan Tóth. *Fast WFOMC*. https://github.com/jan-toth/FastWFOMC.jl/. 2023.

[130] Jan Tóth and Ondřej Kuželka. "Lifted inference with linear order axiom." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 10. 2023, pp. 12295–12304.

[131] Pietro Totis, Jesse Davis, Luc De Raedt, and Angelika Kimmig. "Lifted Reasoning for Combinatorial Counting." In: *Journal of Artificial Intelligence Research* 76 (2023), pp. 1–58.

[132] Josef Urban and Jan Jakubuv. "First neural conjecturing datasets and experiments." In: *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings 13*. Springer. 2020, pp. 315–323.

[133] Timothy Van Bremen and Ondřej Kuželka. "Lifted inference with tree axioms." In: *Artificial Intelligence* 324 (2023), p. 103997.

[134] Guy Van den Broeck. "On the Completeness of First-Order Knowledge Compilation for Lifted Probabilistic Inference." In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., 2011, pp. 1386–1394. ISBN: 9781618395993.

[135] 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. Vienna, Austria: AAAI Press, 2014, pp. 111–120. ISBN: 1577356578.

[136] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. "Lifted probabilistic inference by first-order knowledge compilation." In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Toby Walsh. AAAI Press/International Joint Conferences on Artificial Intelligence, 2011, pp. 2178–2185. URL: http://starai.cs.ucla.edu/papers/VdBIJCAI11.pdf.

[137] Zhichun Wang and Juanzi Li. “RDF2Rules: Learning rules from RDF knowledge bases by mining frequent predicate cycles.” In: *arXiv preprint arXiv:1512.07734* (2015).

[138] Richard Warepam. *Why “Domain Knowledge” is IMPORTANT for Data Science Jobs?* https://medium.com/dare-to-be-better/why-domain-knowledge-is-important-for-data-science-jobs-8195a5d3e84e. [Online; accessed 01-January-2024]. 2023.

[139] Irene Weber. “Levelwise search and pruning strategies for first-order hypothesis spaces.” In: *Journal of Intelligent Information Systems* 14 (2000), pp. 217–239.

[140] Boris Weisfeiler and AA Lehman. “A reduction of a graph to a canonical form and an algebra arising during this reduction.” In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968), pp. 12–16.

[141] Fan Yang, Zhilin Yang, and William W Cohen. “Differentiable learning of logical rules for knowledge base reasoning.” In: *Advances in neural information processing systems* 30 (2017).

[142] Monika Žáková and Filip Železnỳ. “Exploiting term, predicate, and feature taxonomies in propositionalization and propositional rule learning.” In: *European Conference on Machine Learning*. Springer. 2007, pp. 798–805.

[143] Martin Zeman, Martin Ralbovskỳ, Vojtěch Svátek, and Jan Rauch. “Ontology-driven data preparation for association mining.” In: *Online http://keg. vse. cz/onto-kdd-draft. pdf* (2009).

[144] Václav Zeman, Tomáš Kliegr, and Vojtěch Svátek. “RDFRules: Making RDF rule mining easier and even more efficient.” In: *Semantic web* 12.4 (2021), pp. 569–602.

[145] Qiang Zeng, Jignesh M Patel, and David Page. “Quickfoil: Scalable inductive logic programming.” In: *Proceedings of the VLDB Endowment* 8.3 (2014), pp. 197–208.

[146] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. “Neural bellman-ford networks: A general graph neural network framework for link prediction.” In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29476–29490.

[147] Kaja Zupanc and Jesse Davis. “Estimating rule quality for knowledge base completion with the relationship between coverage assumption.” In: *Proceedings of the Web Conference 2018*. 2018, pp. 1–9.

AUTHOR'S PUBLICATIONS

C.1 PUBLICATIONS RELATED THE TOPIC OF THIS THESIS

List of publications presented for the purpose of dissertation defense with citations from Web of Science, Scopus, and Google Scholar listed as of February 18th, 2024.

C.1.1 *Conference papers*

[1] **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 2023, 19th-25th August 2023, Macao, SAR, China*. ijcai.org, 2023, pp. 3338–3346. DOI: 10.24963/ijcai.2023/372. URL: https://doi.org/10.24963/ijcai.2023/372. WoS: 0, Scopus: 1, Google: 1.

[2] **Martin Svatoš**, Steven Schockaert, Jesse Davis, and Ondřej Kuželka. "STRiKE: Rule-Driven Relational Learning Using Stratified k-Entailment." In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. Ed. by Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 1515–1522. DOI: 10.3233/FAIA200259. URL: https://doi.org/10.3233/FAIA200259. WoS: 0, Scopus: 1, Google: 7.

[3] **Martin Svatoš**, Gustav Šourek, Filip Zelezný, Steven Schockaert, and Ondřej Kuželka. "Pruning Hypothesis Spaces Using Learned Domain Theories." In: *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*. Ed. by Nicolas Lachiche and Christel Vrain. Vol. 10759. Lecture Notes in Computer Science. Springer, 2017, pp. 152–168. DOI: 10.1007/978-3-319-78090-0_11. URL: https://doi.org/10.1007/978-3-319-78090-0%5C_11. WoS: 0, Scopus: 2, Google: 5.

[4] Gustav Šourek, **Martin Svatoš**, Filip Železný, Steven Schockaert, and Ondřej Kuželka. "Stacked Structure Learning for Lifted Relational Neural Networks." In: *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*. Ed. by Nicolas Lachiche and Christel Vrain. Vol. 10759. Lecture Notes in Computer Science. Springer, 2017, pp. 140–151. DOI: 10.1007/978-3-319-78090-0_10. URL: https://doi.org/10.1007/978-3-319-78090-0%5C_10. WoS: 0, Scopus: 4, Google: 12.

C.2 OTHER AUTHORS' PUBLICATIONS

C.2.1 *Conference papers*

[1] Jan Tozicka, Jan Jakubuv, **Martin Svatoš**, and Antonin Komenda. "Recursive Polynomial Reductions for Classical Planning." In: *TWENTY-SIXTH INTERNATIONAL CONFERENCE ON AUTOMATED PLANNING AND SCHEDULING (ICAPS 2016).* Proceedings of the International Conference on Automated Planning and Scheduling. 26th International Conference on Automated Planning and Scheduling (ICAPS), Kings Coll, London, ENGLAND, JUN 12-17, 2016. 2016, pp. 317–325. WoS: 2, Scopus: 4, Google: 8.

C.2.2 *Workshop papers*

[1] **Martin Svatoš**, Gustav Šourek, and Filip Železný. "Revisiting Neural-Symbolic Learning Cycle." In: *Proceedings of the 2019 International Workshop on Neural-Symbolic Learning and Reasoning (NeSy 2019), Annual workshop of the Neural-Symbolic Learning and Reasoning Association, Macao, China, August 12, 2019.* Ed. by Derek Doran, Artur S. d'Avila Garcez, and Freddy Lécué. 2019. WoS: 0, Scopus: 0, Google: 2.