KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Declarative Modelling and Reasoning for Combinatorial Problem Solving and Argumentation under Uncertainty.

Pietro Totis

Supervisors:
Prof. dr. L. De Raedt
Prof. dr. ir. J. Davis
Prof. dr. A. Kimmig

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

March 2023

Declarative Modelling and Reasoning for Combinatorial Problem Solving and Argumentation under Uncertainty.

Pietro TOTIS

Examination committee:
Prof. dr. ir. Dirk Vandermeulen, chair
Prof. dr. L. De Raedt, supervisor
Prof. dr. ir. J. Davis, supervisor
Prof. dr. A. Kimmig, supervisor
Prof. dr. ir. W. Meert
Prof. dr. ir. T. De Laet
Prof. dr. ir. O. Kuželka
(Czech Technical University in Prague)
Prof. dr. ir. F. G. Cozman
(Escola Politécnica, Universidade de São Paulo)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

March 2023

© 2023 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Pietro Totis, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Acknowledgements

This thesis is the conclusion of a journey started four years ago, and the technical contributions presented are only a part of the story. To summarize the other part, I would like to begin with expressing my gratitude to the people that guided and helped me, along with those who made the journey more enjoyable.

First, I would like to thank my supervisors, Luc de Raedt, Jesse Davis and Angelika Kimmig. Their guidance and supervision are essential ingredients of this work, along with their patience and enthusiasm. These were clear from my very first day of the PhD, when I missed the bus stop in front of the department and arrived late to our first meeting. The challenge of making happy three supervisors at the same time could seem more of a burden than a benefit, but each of them has taught me invaluable lessons from their personal point of view.

Along with my supervisors, I am grateful to the co-authors I had the pleasure to work with during these years, who contributed to the research lines presented in this thesis. I am also grateful to Wannes Meert and Tinne de Laet, for the valuable feedback on the progress of the PhD, and, together with Ondrej Kuželka and Fabio Cozman, for the excellent comments on this manuscript. I gratefully acknowledge FWO for sponsoring the project that financially supported my doctoral studies, and the administrative staff at the department for their help throughout these years.

I would also like to thank all the colleagues in the DTAI group, for contributing to a pleasant work environment and for the enjoyable company outside office hours. There are three groups I would like to thank in particular, for giving me beautiful memories of my time in Leuven. First, my office mates: Vincent, Tim, Nitesh, Victor, and others with whom I had the pleasure to share the office. Second, the regulars of the Alma train: Kshitij, Adem, Aras and all the others I enjoyed the company of at lunchtime. Finally, the "Italian community" at DTAI, which I had the pleasure to see growing every year. Thank you, Stefano, Giuseppe, Gabriele, Paolo, Emanuele and the other fellow Italians, for making me feel a bit more at home during my stay in Leuven. A special thanks goes to Lorenzo: it is hard to summarize a friendship in a few words,

but in this case my favourite are definitely “cuddle contact”.

I am also grateful to all the new friends I met in Leuven for the fun time spent together, whether that was a beer in Oude Markt, a trip around Belgium, or playing sports. To this group also belong my classmates and teachers at CLT: finding the time to learn a language during a PhD is no easy task and their enthusiasm and engagement made it much easier. The old friends are also an important part of this journey. I am grateful for all their efforts in organizing holidays and other occasions to keep in touch and forget for a moment the difficult times. There are two friends in particular to whom I owe a huge “thank you”. Thank you Ciccio and Gioele for listening and supporting me all the way, and for being such amazing travel companions in our PhD journey.

Finally, the most wholehearted thank you goes to my family, without their love and support none of this would have been possible. Grazie mamma e papà per essere sempre stati al mio fianco in questa avventura, e grazie Marta per l’aiuto costante nel trovare la soddisfazione lungo la strada.

Abstract

Automating reasoning to solve complex problems is one of the main goals of artificial intelligence (AI). Ideally, we would like to declare *what* is the problem (modelling) and let an AI determine *how* to solve it (reasoning). Automating this process typically requires a combination of several forms of intelligence, such as logical reasoning, learning, representing and generalizing knowledge and communicating in natural language. Declarative frameworks offer many AI techniques that satisfy these requirements. They are based on the separation of the problem description from the solution implementation. The problem description is given by means of a declarative language, also called modelling language. Declarative languages provide a range of features and constructs that allow programmers to define the goals and constraints of a program, without having to worry about the solving details. Solving is done by specialized algorithms called solvers, which use a variety of techniques, such as logical reasoning, constraint propagation and search, and theorem proving, to efficiently generate solutions to any problem expressible in the declarative language. Declarative frameworks can thus reason about abstract concepts and make complex reasoning steps based on a set of predefined rules. The advantage of declarative frameworks is their generalizability and adaptability, as the system can easily be updated and modified to represent and solve new problems or to incorporate new data. This thesis discusses novel declarative modelling and reasoning techniques for two general AI problems.

The first set of contributions regards combinatorics math word problems, which ask to count the number of possible configurations of a setting sketched in natural language. They represent an interesting challenge for AI because they are often used as a benchmark for human cognition. The first contribution is the identification of key aspects of combinatorics math problems which traditional declarative frameworks fail to support adequately, thus preventing an appropriate modelling. Second, we address these limitations proposing a novel declarative language for *modelling* combinatorics math problems. Third, we pair the contributions to modelling with novel *reasoning* techniques for combinatorics math problems and we implement them in a solver tested on both real-world and artificial examples.

The second set of contributions of this thesis is related to argumentation, which is the task of structuring and determining the acceptability of arguments in a debate. Designing systems that are able to argue and persuade is a very relevant and challenging AI task, and taking into account uncertainty is fundamental in such situations. The first contribution is a novel *modelling* approach to argumentation under uncertainty based on probabilistic logic programming (PLP), a popular declarative framework for modelling and reasoning about structured, uncertain domains. Second, we introduce novel *reasoning* techniques to solve such problems, because the logic component of traditional PLP frameworks is too restricted to reason over such models. Third, we show that modelling argumentation problems with PLP gives them access to the general purpose tools and algorithms of PLP for reasoning and learning under uncertainty.

Beknopte samenvatting

Het automatiseren van redeneringen om complexe problemen op te lossen is een van de belangrijkste doelstellingen van kunstmatige intelligentie (AI). Idealiter zouden we willen aangeven *wat* het probleem is (modelleren) en AI laten bepalen *hoe* het moet worden opgelost (redeneren). Het automatiseren van dit proces vereist doorgaans een combinatie van verschillende vormen van intelligentie, zoals logisch redeneren, leren, kennis representeren en generaliseren en communiceren in natuurlijke taal. Declaratieve raamwerken bieden veel AI-technieken die aan deze eisen voldoen. Ze zijn gebaseerd op de scheiding van de probleembeschrijving en de oplossingimplementatie. De probleembeschrijving wordt gegeven door middel van een declaratieve taal, ook wel modelleringstaal genoemd. Declaratieve talen bieden een reeks functies en constructen waarmee programmeurs de doelen en beperkingen van een programma kunnen definiëren, zonder zich zorgen te hoeven maken over de oplossingsdetails. Het oplossen gebeurt door gespecialiseerde algoritmen, solvers genaamd, die een verscheidenheid aan technieken gebruiken, zoals logisch redeneren, voortplanting van beperkingen en zoeken, en het bewijzen van stellingen, om efficiënt oplossingen te genereren voor elk probleem dat in de declaratieve taal kan worden uitgedrukt. Declaratieve raamwerken kunnen dus redeneren over abstracte concepten en complexe redeneerstappen maken op basis van een reeks vooraf gedefinieerde regels. Het voordeel van declaratieve raamwerken is hun generaliseerbaarheid en aanpasbaarheid, aangezien het systeem gemakkelijk kan worden bijgewerkt en aangepast om nieuwe problemen te representeren en op te lossen of om nieuwe gegevens op te nemen. Dit proefschrift bespreekt nieuwe declaratieve modelleer- en redeneertechnieken voor twee algemene AI-problemen.

De eerste reeks bijdragen betreft combinatorische wiskundige woordproblemen, waarbij het aantal mogelijke configuraties van een in natuurlijke taal geschetste omgeving moet worden geteld. Ze vormen een interessante uitdaging voor AI omdat ze vaak worden gebruikt als benchmark voor menselijke cognitieve vaardigheden. De eerste bijdrage is de identificatie van belangrijke aspecten van combinatorische wiskundeproblemen die traditionele declaratieve kaders niet adequaat ondersteunen, waardoor een adequate modellering onmogelijk is. Ten tweede pakken we deze beperkingen aan door een

nieuwe declaratieve taal voor te stellen voor het *modelleren* van combinatorische wiskundeproblemen. Ten derde combineren we de bijdragen aan het modelleren met nieuwe *redeneertechnieken* voor combinatorische wiskundeproblemen en we implementeren ze in een oplosser die getest is op zowel echte als kunstmatige voorbeelden.

De tweede reeks bijdragen van dit proefschrift heeft betrekking op argumentatieproblemen, namelijk de taak van het structureren en bepalen van de aanvaardbaarheid van argumenten in een debat. Het ontwerpen van systemen die kunnen argumenteren en overtuigen is een zeer relevante en uitdagende AI-taak, en rekening houden met onzekerheid is in dergelijke situaties van fundamenteel belang. De eerste bijdrage is een nieuwe aanpak van argumentatie onder onzekerheid, gebaseerd op probabilistische logische programmering (PLP), een populair declaratief raamwerk voor het modelleren van en redeneren over gestructureerde, onzekere domeinen. Ten tweede introduceren we nieuwe *redeneertechnieken* om dergelijke problemen op te lossen, omdat de logische component van traditionele PLP-raamwerken te beperkt is om over dergelijke modellen te redeneren. Ten derde laten wij zien dat het koppelen van argumentatieproblemen aan PLP toegang geeft tot de algemene hulpmiddelen en algoritmen van PLP voor redeneren en leren onder onzekerheid.

List of Abbreviations

2AMC Second-level Algebraic Model Counting. 143

AI Artificial Intelligence. 1

AMC Algebraic Model Counting. 59

ASP Answer Set Programming. 117

CNF Conjunctive Normal Form. 129

CSP Constraint Satisfaction Problem. 24

MAP Maximum A Posteriori. 142

MPE Most Probable Explanation. 126

PLP Probabilistic Logic Programming. 10

QBAF Quantitative Bipolar Argumentation Frameworks. 26

WMC Weighted Model Counting. 19

Contents

Abstract iii

Beknopte samenvatting v

List of Abbreviations vii

List of Symbols ix

Contents ix

List of Figures xiii

List of Tables xv

1 Introduction 1
- 1.1 Automated Reasoning Problems in Natural Language 3
- 1.2 Declarative Programming 4
- 1.3 Motivation and Problem Statement 6
- 1.4 Thesis Contributions 9
- 1.5 Structure of the Thesis 11

2 Background 13
- 2.1 Propositional and First-order Logic 13
- 2.2 Logic Programming 14
- 2.3 Probabilistic Logic Programming 16
- 2.4 Constraint Programming 24
- 2.5 Abstract Argumentation 26

I Combinatorial Problem Solving 29

3 Combinatorics Math Word Problems: Modelling 30

3.1 Related Work . . . 33
3.1.1 Constraint Satisfaction Problems . . . 34
3.1.2 Lifted Probabilistic Inference . . . 34
3.2 Modelling Combinatorics Math Problems . . . 37
3.2.1 Multisets . . . 38
3.2.2 Configurations . . . 42
3.2.3 Constraints . . . 46
3.3 Modelling: Language Analysis . . . 50
3.4 Conclusion . . . 55

4 Combinatorics Math Problems: Reasoning 57
4.1 Related Work . . . 58
4.1.1 Constraint Satisfaction Problems . . . 58
4.1.2 Lifted Probabilistic Inference . . . 59
4.2 Applying Lifted Reasoning Principles to #CSPs . . . 62
4.3 CoSo . . . 66
4.3.1 Manipulating Domains, Multisets and Constraints . . . 68
4.3.2 Relevant Properties, Relevant Parts, Histograms . . . 69
4.4 Base Cases . . . 71
4.4.1 Level 1: Base Cases . . . 71
4.4.2 Level 2: Base Cases . . . 72
4.4.3 Level 1 and 2: Propagating Counting Constraints . . . 74
4.5 Level 1 and 2: Exchangeability and Shattering Counting Constraints . . . 75
4.5.1 Exchangeability . . . 76
4.5.2 Level 1 and 2: Shattering Counting Constraints . . . 76
4.6 Level 1: Shattering Non-repetition . . . 78
4.6.1 Level 1: Shattering Repetition Without Counting Constraints . . . 78
4.6.2 Level 1: Shattering Repetition With Counting Constraints . . . 79
4.7 Level 2: Shattering Parts . . . 80
4.7.1 Distributing Entities . . . 81
4.7.2 One Non-fixed Relevant Part: Exchangeable Variables . . . 82
4.7.3 One Non-fixed Relevant Part: Non-exchangeable Variables . . . 83
4.7.4 Many Non-fixed Relevant Parts . . . 84
4.8 Detailed Examples . . . 85
4.9 Reasoning: Solver Analysis . . . 91
4.9.1 Real-world Problems . . . 92
4.9.2 Synthetic Benchmarks . . . 94
4.10 Conclusion . . . 97

II Argumentation under Uncertainty 99

5 Probabilistic Argumentation Problems: Modelling 100

5.1 Modelling Probabilistic Argumentation with PLP 102
5.2 Related Work 109
5.3 Conclusion 113

6 Probabilistic Argumentation Models: Reasoning 115
6.1 Reasoning over SMProbLog Programs 116
6.1.1 Formal Semantics 122
6.2 The Joint Distribution of Beliefs 124
6.2.1 Epistemic Properties 127
6.3 Inference and Learning 129
6.3.1 Inference Task 129
6.3.2 Learning Task 136
6.4 Experiments 137
6.5 Related Work 145
6.6 Conclusion 149

7 Conclusion 151

A CoLa grammar 157

B Automatic Translation of CoLa to ASP and Essence 161

C Asking ChatGPT to solve combinatorics math word problems 171

Bibliography 177

List of Figures

1.1 Declarative Programming 5
1.2 Declarative Programming - Math word problems 6
1.3 Declarative Programming - Argumentation 7

2.1 Smooth d-DNNF for the implication *neighbour_calls* ← *alarm*, *neighbour_at_home* (left) and the corresponding arithmetic circuit (right). 21
2.2 Abstract argumentation framework. Edges represent attacks, nodes are arguments. 26
2.3 In probabilistic abstract argumentation framework nodes and attacks are annotated with values in $[0, 1]$ (cfr. Figure 2.2). 27

3.1 Visualization of **P1** (left) and **P2** (right). 31
3.2 On the left, the objects in Example 1 grouped by their properties. On the right, the corresponding multiset. Stars represent the colour property, black items represent the shape property. 39
3.3 Level 1 configurations map slots to objects Y. 43
3.4 Level 2 configurations map objects Y to slots 43

4.1 Solver reasoning schema. L1= level 1 configurations, L2 = level 2 configurations. Dashed arrows denote recursive calls to the solver. . . 67
4.2 Solver execution flow for **P1** 86
4.3 Solver execution flow for the first subproblem of **P1** 86
4.4 Solver execution flow for the second subproblem of **P1** 87
4.5 Solver execution flow for **P2** 89
4.6 CoSo solving time (log scale) tends to increase with the number of subproblems considered. Real-world dataset and synthetic benchmarks combined. 92
4.7 The solving time of propositional methods increases exponentially with the number of objects or the size of the configuration. 94

4.8 Solving times (log scale) on synthetic benchmarks: CoSo outperforms propositional frameworks in most cases, in particular with the increase of the number of solutions. 95
4.9 On unsatisfiable configurations CoSo outperforms propositional frameworks on most of the instances (log scale). For ASP and sharpSAT most of these instances have the largest groundings of the dataset. Legend format: *configuration type universe size/configuration size of reference*. *ms=multiset, pm=permutation, sq=sequence, cp=composition*. 96

5.1 Example 5.4 representation. Dashed (resp. solid) edges represent supports (resp. attacks) R^+ (resp. R^-). Nodes (resp. edges) are labelled with the corresponding bias (belief). 106

6.1 ProbLog2 inference schema 130
6.2 SMProbLog inference schema. (*) denotes a different version of DSHARP from ProbLog2, specific for stable model counting. 130
6.3 Compilation step of Example 6.5 (top right) and Example 6.7 (left). Models for both theories are compactly encoded as d-DNNF logical circuits. $bg = burglary$, $eq = earthquake$, $al = alarm$, $df = df$, $rt = right$. 132
6.4 Enumeration step of Example 6.5 (top right) and Example 6.7 models (left). Each node contains the corresponding (partial) models. Diamonds are union nodes, squares are Cartesian products. M is the complete list of the 5 models of Example 6.7. 133
6.5 *WMC* of Example 6.5 (top right) and Example 6.7 (left) . Each node contains the corresponding weight, logical variables do not influence the weights. Diamonds are sum nodes, squares are product nodes. . . 134
6.6 Inference time on benchmarks. 139
6.7 Mean absolute error by number of samples. 140
6.8 Mean running time by number of parameters and size of circuit (dashed line) on increasing number of observations (coloured lines). 141
6.9 Possible colorings of Example 6.25 147

7.1 Declarative Programming CoLa and CoSo 152
7.2 Declarative Programming - smProbLog 153

List of Tables

3.1 Combinatorics math problems from Dries et al. (2017). The solutions are obtained by applying counting rules based on the sizes of the groups of objects defined. 32
3.2 Twelvefold Way: basic combinatorial configurations with the corresponding counting rules. $y = |Y|, x = |X|$, $y^{\underline{x}}$ is the falling factorial, $\binom{\cdot}{\cdot}$ is the binomial coefficient, $\left\{ \cdot \atop \cdot \right\}$ is the Stirling number of the second kind, $p_y(x)$ is the integer partition of x into y parts and $[x \leq y]$ is 1 if $x \leq y$, 0 otherwise. 42
3.3 Configuration types and the corresponding CoLa syntax for a set-formula φ 45
3.4 Summary of CoLa syntax and corresponding semantics. U denotes the universe corresponding to the given objects and properties. 49

4.1 Probabilistic frameworks (Forclift, GC-FOVE) are limited in modelling by the input language and first-order logic, but implement lifted counting algorithms with limited constraint support. On the other hand CSP frameworks offer direct support for sets and constraints but not for multisets and lifted counting. 60
4.2 Operator cases without counting constraints. (n.a.= not applicable) . . 75
4.3 Operator cases with counting constraints. 75
4.4 On the 185 problems that can be encoded in CoLa, CoSo outperforms other frameworks in the number of solved problems within the timeout and average runtime. 93

5.1 Argumentation frameworks overview. 101
5.2 Argumentation frameworks comparison. $\sim$ denotes partial support . . 109
5.3 Possible models comparison of Example 5.10 112

6.1 PLP frameworks overview. 116

6.2 PLP frameworks comparison. n.a. = not applicable. 2^*= the logic semantics is two-valued stable models, but we introduce a third value for total choices with 0 stable models. 146

Chapter 1

Introduction

Artificial Intelligence (AI) research aims at developing machines that exhibit intelligent behaviour, which is usually defined as a combination of different types of capabilities. The many characterizations proposed (Legg and Hutter 2006) usually recognize logical reasoning and problem-solving skills among the cornerstones of AI. The former is a form of thinking in which premises and relations between premises are used in a rigorous manner to infer entailed (or implied) conclusions (Nunes 2012). The latter is typically a way of applying logical reasoning called "top-down" reasoning, where solutions are discovered by reasoning backwards from the goal to subgoals (R. Kowalski and Smoliar 1982). In top-down reasoning the problem is thus decomposed into subproblems whose solutions allow us to determine the solution of the original problem.

Consequently, since its inception, automating logical reasoning to solve practical problems has remained one of the core challenges of AI. The subfield of AI research concerned with this task is called Automated Reasoning. Given a high-level description of a problem, an automated reasoning system wants to find a sequence of logical steps towards the solution of the problem. The input description of the problem is composed of two parts:

1. a set of statements expressing the relevant information available (*knowledge*);
2. a specific statement indicating what information should be derived from the available knowledge (*query*).

The input is thus a representation of the relevant knowledge defining the problem, which does not specify a procedure to compute the solution. The output is the information specified by the query, optionally with the reasoning steps that led to the answer.

Many practical problems can be viewed as automated reasoning tasks if we use natural language to provide the input statements to the automated reasoning system. In particular, in this thesis we will develop automated reasoning systems for two types of problems usually formulated in natural language: mathematical word problems and argumentation problems. Math problem solving and arguing and persuading effectively are daily activities that are often considered an indicator of human intelligence, therefore they represent interesting benchmarks for artificial intelligence systems.

Mathematical word problems describe a setting in natural language, from which it is required to compute some value. For example, in probability and combinatorics math problems the query is to determine, respectively, a probability or a count.

Example 1.1. A complete cycle of a traffic light takes 80 seconds. During each cycle, the light is green for 40 seconds, amber for 10 seconds, and red for 30 seconds. At a randomly chosen time, what is the probability that the light will not be red?
To compute the probability of a colour we have to consider the time the colour is shown relative to the total time ($80s$*). The property "not red" is satisfied by the colours "green" and "amber". Since the traffic light shows only one colour at a time the time this property is shown is* $40s + 10s = 50s$*, therefore the relative time in which the property is satisfied is:* $\frac{50}{80}$.
Solution: $\frac{40+10}{80} = \frac{5}{8}$

Example 1.2. A shipment of 12 different TVs contains 3 defective ones. In how many ways can a hotel purchase 5 of these TVs and receive at least 2 of the defective ones?
To count the number of purchases we count the number of subsets of size 5. To form such sets we choose either defective or non-defective TVs. To have at least 2 defective TVs we have to choose either 2 or 3 defective TVs. Thus, we sum the results of two subproblems: one where we choose 2 defective TVs and one where we choose 3. The number of choices out of 3 defective TVs is given by the binomial coefficient: $\binom{3}{2} = 3$ *or* $\binom{3}{3} = 1$. *These choices can be combined with any selection of functioning TVs: to form a set of 5 these TVs are respectively 3 and 2, selected from 9 (*12 *TVs* -3 *defective). The choices of functioning TVs can thus be* $\binom{9}{3} = 84$ *or* $\binom{9}{2} = 36$. *The combinations of choices for defective and non-defective TVs are respectively* $\binom{3}{2} \cdot \binom{9}{3} = 252$ *and* $\binom{3}{3} \cdot \binom{9}{2} = 36$ *therefore the total possible selections are* $252 + 36 = 288$.
Solution: $\binom{3}{2} \cdot \binom{9}{3} + \binom{3}{3} \cdot \binom{9}{2} = 288$

Argumentation problems, on the other hand, present a collection of arguments, and the query is to determine the acceptable (winning) arguments, for instance the claim in the last statement of the following example:

Example 1.3. Yes, it's annoying and cumbersome to separate your rubbish properly all the time (a_1), but small gestures become natural by daily repetition (a_2). Three different

bin bags stink away in the kitchen and have to be sorted into different wheelie bins (a_3). But still Germany produces way too much rubbish (a_4) and too many resources are lost when what actually should be separated and recycled is burnt (a_5). We Berliners should take the chance and become pioneers in waste separation! (a_6)

In this thesis we address the problem of designing automated reasoning systems that can solve mathematical word problems and argumentation problems.

1.1 Automated Reasoning Problems in Natural Language

Solving mathematical word problems and argumentation problems presents two major challenges for AI: first, understanding the information provided in natural language, second, determining how the query can be derived from such information. The two main approaches to these problems are either tackling these challenges with a system that deals with the two problems at the same time, or developing two separate systems for each challenge.

The first is to use a so-called end-to-end approach, where a neural model is given the input statements and asked to predict the solution. This approach has been mainly applied to mathematical word problems (Mandal et al. 2020; Wang et al. 2017), however it suffers from the well-known limitations of deep learning techniques (Marcus 2018), among which interpretability of the errors and the explanation of the given answer (Saxton et al. 2019) and generalizability (Patel et al. 2021). In fact, Saxton et al. (2019) remark that, due to the "black-box" nature of the neural network, it is hard to interpret the reasons why a given answer is wrong (or correct), and to relate the mistake to a language understanding problem or faulty reasoning. At the same time, Patel et al. (2021) showed that in previous work most of the accuracy of the network is ascribable to learning simple heuristics even without word-order information or the question text. Recent work on pre-trained language models, such as GPT-3 (Brown et al. 2020), showed substantial progress on many challenging NLP tasks, including question answering. Such language models can provide a noticeable improvement in the explainability of the solution by generating a detailed description in natural language of the reasoning steps. However, while the quality of the generated text is so high that it is hard to determine whether it was produced by a human or not, some limitations on the understanding capability and problem-solving skills remain evident. Appendix C presents some examples of the answers provided by ChatGPT (OpenAI 2022), a recent state-of-the-art conversational language model.

The second is a so-called two-step approach, where the problem is split into the two challenges by using an intermediate formal representation. The challenge of

understanding the information provided in natural language is thus formulated as the problem of mapping the sentences in natural language to sentences in a formal language that can be easily parsed by an automated reasoning system. The challenge of deriving the answer to the query becomes the problem of developing explicit inference methods that, for any set of statements in the formal language, determine how the answer to the query is inferred. This line of thought has been more prominent in the field of argumentation, where argumentation mining (Stede and Schneider 2018) and abstract argumentation (R. A. Kowalski and Toni 1996) are two broad and popular research topics corresponding to the two steps.

The limitations of this approach lay in the choice of the intermediate formal language, which is a "bottleneck" defining what problems can and cannot be solved by the automated reasoning system (Zhang et al. 2020). In fact, the statements we can use to express the problems come with a trade-off. The effort of a two-step approach is the definition of reasoning techniques that cover all possible combinations of statements. An expressive language, such as natural language, allows us to describe any problem, but this requires very general inference methods that can answer all possible queries. Unfortunately, a solver for any problem expressible in natural language at the moment is unrealistic. By reducing the number of statements available, that is, by modulating the expressivity of the intermediate formal language, we obtain a more tractable reasoning task. For example, if we restrict our language to statements built with numbers and arithmetic operations, a simple calculator can answer all possible queries.

In this thesis we focus on a two-step approach for both types of problems at hand and improve the state of the art by pairing a more expressive language with the necessary expansion of the inference techniques required by the more general language. We will show that the available declarative languages are unsatisfactory in their level of expressivity with respect to combinatorics math problems and thus offer a poor trade-off between language expressivity and reasoning capabilities. Similarly, we will examine and improve the expressivity of probabilistic logic programming frameworks to model and reason over probabilistic argumentation problems, because existing systems restrict the intermediate language excessively to achieve this goal.

1.2 Declarative Programming

In the two-step approach, it is essential to represent the available knowledge in a complete and interpretable format for the automated reasoning system. Knowledge representation is a core component of automated reasoning because it determines the type of information that the automated reasoning system has access to and is able to process while reasoning. Because rational conclusions and solutions can only be produced by logical reasoning, the standard knowledge representation formalism for

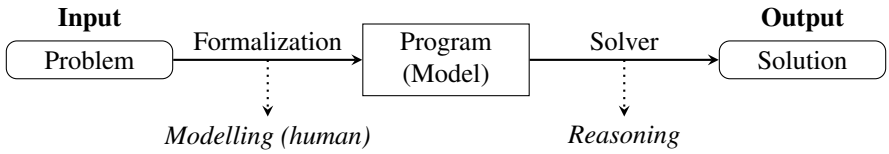


Figure 1.1: Declarative Programming approach to problem solving.

automated reasoning are propositional and first-order logic (Section 2.1). Propositional and first-order logic lay the formal foundations of many automated reasoning system, such as *logic programming* (Section 2.2), *probabilistic logic programming* (Section 2.3), and *constraint programming* (Section 2.4). These formalisms belong to the paradigm of *declarative programming*, which follows the automated reasoning principle of declaring the problem specification without specifying the details about how to find the solution.

Declarative programs (Lloyd 1994) specify an automated reasoning task (Figure 1.1), stating *what* is to be computed rather than *how* to do it. The *what* is a problem specified in some suitable logic with a high level of abstraction (the *model*), the *how* is a formal, provable, inference of the solution from the model that a solver should automatically infer (*reasoning*).

Therefore, a declarative knowledge representation differs from traditional programming paradigms, which describe a problem with an explicit procedural representation of the steps required for the resolution. Because declarative programs specify automated reasoning tasks, the efficiency of the inference process is usually guaranteed for suitable restrictions of the logic component. In fact, declarative programming systems present the trade-off between what can be represented as comprehensively as possible and what can be reasoned about as effectively as possible. On the other hand, having an unambiguous declarative language, capable of expressing the necessary domain knowledge, allows us to draw inferences, answer queries and design both knowledge bases and desired program behaviours with high modularity and generalizability. This because the same facts can be used by different automatic reasoning systems for different types of inference and the knowledge base can be expanded with new statements independently of the information already encoded.

In this thesis we address automated reasoning problems by developing declarative programming systems and discuss the corresponding trade-offs in defining the modelling language and the respective reasoning techniques. We focus on problems specified in natural language, but we delimit the class of problems supported by our declarative programming systems by defining a formal language for a specific type of problems. The formal language restricts the problems that can be modelled with our approach compared to natural language. However, this allows us to develop an effective solver. The challenge is to expand *what* can be specified in the formal language towards

the expressivity of natural language, without losing the capability of inferring *how* the problems can be effectively solved.

1.3 Motivation and Problem Statement

Solving mathematical world problems or arguing and persuading effectively are different types of problems that require different types of reasoning and solutions. However, they share the same two automated reasoning challenges: modelling and reasoning.

Step one. The first challenge is to design a domain-specific modelling language to which the relevant information can be mapped. Understanding the problem setting then entails being able to properly encode it in a formal model. Combinatorics and probability problems require to understand the description of the situation to distinguish between useful and irrelevant information for the calculation. A domain-specific language for this type of problems then requires statements to model groups of objects, their properties and how they are arranged. For instance, Example 1.2 will be encoded with the following statements:

```
labelled property tvs; #tvs = 12;
property defective; #(tvs & defective) = 3;
purchase in {tvs}; #purchase = 5;
#(purchase & defective) >= 2;
```

Similarly, argumentation problems can also contain non-argumentative information which has to be separated from the relevant argumentative text (argumentation mining). The output of argumentation mining is the identification of arguments, for instance in Example 1.3 arguments $a_1, a_2, \ldots$ and their relations, a_1 attacks a_2, a_3 supports $a_1 \ldots$ In this case, the statements should encode these relations along with additional properties of the arguments, such as associated beliefs, preferences or internal structures. For

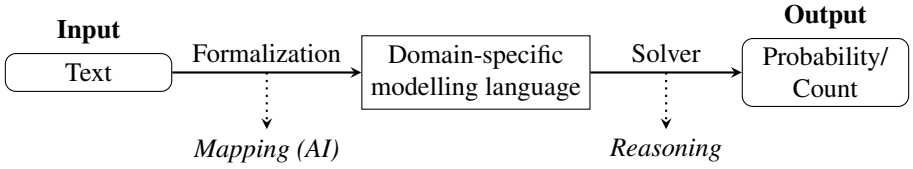


Figure 1.2: Declarative Programming approach to mathematical word problems.

instance, Example 1.3 with the addition of beliefs will be encoded with the following statements:

```
0.4::bias(a1). 0.8::bias(a2).
0.3::bias(a3). 0.7::bias(a4).
0.6::bias(a5). 0.7::bias(a6).
arg(A)← bias(A).
0.6::¬arg(a6) ← arg(a1). 0.6::arg(a4) ← arg(a5).
0.3::¬arg(a1) ← arg(a4). 0.5::arg(a1) ← arg(a3).
0.8::¬arg(a1) ← arg(a2). 0.7::¬arg(a2) ← arg(a1).
```

Step 2. The second challenge is to design a solver that derives the solution for all problems expressible in the domain-specific language. In mathematical world problems (Figure 1.2) we are interested in a strategy to derive the required count or probability from the available data. Humans are adept at doing this by identifying commonly occurring structures in the questions for which a closed-form formula exists for computing the answer. However, current AI approaches are still unable to solve combinatorial problems in this way and rely on brute-force enumeration of all possible solutions. In argumentation problems (Figure 1.3) we are interested in deriving which are the acceptable (winning) arguments. Abstract argumentation defines the restrictions to the logical component that models arguments and relations, along with the corresponding semantics of acceptability. The semantics of acceptability defines the inference rules that prove that an argument is acceptable in a formal argumentative model. While past work successfully explored the use of logic programming systems to reason about deterministic argumentation problems, the application of *probabilistic* logic programming to argumentation problems under uncertainty remains an open problem.

Mathematical word problems and argumentation problems thus require to address similar points to develop dedicated automated reasoning systems:

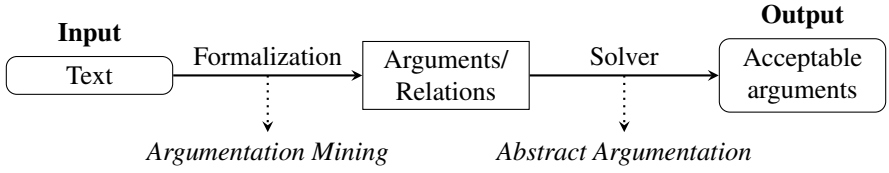


Figure 1.3: Argumentation is composed of Argumentation Mining and Abstract Argumentation.

1. **Mapping.** Both types of problems are expressed in natural language. Therefore, the variety of situations and questions that human-readable words can express has to be mapped to restricted machine-understandable logics.

2. **Modelling.** The restrictions on the logic usually present a trade-off between expressivity and tractability. Therefore, a suitable restriction on the logic has to define the problems that can be expressed. The modelling language should be as expressive as possible without losing tractability.

3. **Reasoning.** For any problem expressible in the given logic the solving technique should be inferred, possibly in an efficient way. Therefore, the modelling language has to be paired with inference algorithms that identify a solving technique for each encodable problem.

In this thesis we consider several problems ranging over all these three points.

While for probability math problems a modelling language and a corresponding solver exist (Dries et al. 2017), this is not the case for combinatorics math problems. It is thus an open question how to define a declarative approach to combinatorics math problems. We will divide this question in the two main components of the declarative approach, modelling and reasoning, as follows:

Question 1. *What are the key characteristics of a modelling language for encoding combinatorics math problems?*

Question 2. *How can models for combinatorics math problems be efficiently solved?*

Similarly, past work investigated the resolution of (deterministic) argumentation problems by means of logic programming. However, there is no argumentation system based on a declarative approach that is able to reason over uncertainty and beliefs about arguments and their relations, determining not only the acceptable arguments but also the corresponding *degree of belief*. Also in this case we fill this gap by answering two questions:

Question 3. *Can probabilistic logic programming model probabilistic argumentation problems?*

Question 4. *Can probabilistic logic programming systems efficiently solve probabilistic argumentation models?*

In this thesis we answer these questions with the following contributions.

1.4 Thesis Contributions

The contributions of the thesis follow the separation of our research questions between combinatorics math problems and probabilistic argumentation problems.

1. Lifted Reasoning for Combinatorial Counting.

The paper on which this chapter is based on was previously published as:

P. Totis, J. Davis, L. D. Raedt, and A. Kimmig (2023). "Lifted Reasoning for Combinatorial Counting". In: *J. Artif. Intell. Res.* 76, pp. 1–58. DOI: 10.1613/jair.1.14062. URL: https://doi.org/10.1613/jair.1.14062

Humans are adept at solving combinatorics math problems as they can identify commonly occurring structures in the questions for which a closed-form formula exists for computing the answer. Unfortunately, current AI approaches are still unable to solve combinatorial problems in this way. We fill this gap by developing novel AI techniques for representing and solving such problems. We address Question 1 with the following contributions:

1. We identify a class of combinatorics math problems which traditional counting techniques fail to model or solve efficiently.
2. We propose a novel declarative language for modelling this class of problems.

From the modelling problem we move to the reasoning problem and answer Question 2 with the following contributions:

3. We propose novel solving techniques bridging probabilistic inference algorithms and constraint programming.
4. We implement them in a solver that solves efficiently the class of problems under investigation.

The evaluation of our contributions on a real-world combinatorics math problems dataset and synthetic benchmarks validates the choices in the design of the modelling language and demonstrates the improved efficiency of the proposed algorithms over the existing counting techniques.

2. smProbLog: Stable Model Semantics in ProbLog and its Applications in Argumentation.

The following contributions are largely based on the paper submitted for publication:

P. Totis, A. Kimmig, and L. De Raedt (2023). "SMProbLog: Stable Model Semantics in ProbLog and its Applications in Argumentation". In: *Theory and Practice of Logic Programming*

Probabilistic Logic Programming (PLP) systems aim at being a general-purpose tool to model real-world situations under uncertainty. However, there is currently no mapping to PLP for probabilistic argumentation problems that manipulate explicitly the degrees of beliefs in arguments. We thus address the lack of a mapping by answering Question 3 with the following contribution:

1. We show how probabilistic argumentation problems can be encoded as SMProbLog programs, a novel PLP system based on ProbLog2 (Fierens et al. 2011).

Probabilistic argumentation problems represent an interesting practical application where a typical assumption of PLP does not hold. That is, in traditional PLP frameworks once the uncertainty about a given set of independent probabilistic facts is resolved, logical reasoning is deterministic and should yield exactly one interpretation of the truth values of the logical atoms in the program. This assumption is usually enforced by limiting the type of statements expressible in the PLP model. The need for a less restrictive language raises the issue of developing novel reasoning techniques to support the new types of models that the expanded modelling language allows. We address this problem by answering Question 4 with the following contributions:

2. We introduce SMProbLog, a PLP system based on the PLP language ProbLog and the PLP system ProbLog2 (Fierens et al. 2011) where the assumption of determinism in the logic rules by traditional PLP systems is removed.
3. We implement the algorithms for both inference and learning tasks in SMProbLog.
4. We show how the implementation of SMProbLog can be used as a system for inference and learning over probabilistic argumentation problems.

We evaluate our approach with experiments analyzing the computational cost of the proposed algorithms and their application to a dataset of argumentation problems.

1.5 Structure of the Thesis

The structure of the thesis follows our research questions with a chapter dedicated to each of them. Chapter 2 introduces the necessary background for the remainder of this thesis. Chapter 3 introduces the problem of solving combinatorics math problems and answers Question 1 by defining a novel modelling language for combinatorics math problems. Chapter 4 defines the reasoning techniques to efficiently solve such problems, thus answering Question 2.

Chapters 5 and 6 focus on probabilistic argumentation and the respective research questions. Chapter 5 answers Question 3 by showing a mapping from probabilistic argumentation problems to probabilistic logic programming. Chapter 6 complements this mapping with the required novel reasoning techniques that answer Question 4.

Chapter 2

Background

This chapter presents the general background information required to understand the problems and questions tackled in this thesis. Related work specific to each contribution is discussed in more detail in the following chapters. The background is organized as follows: Section 2.1 introduces propositional and first-order logic; next, we review the basics of logic programming and its semantics (Section 2.2); then, Section 2.3 gives a more detailed overview of inference and learning over probabilistic logic programs. Finally, we conclude the chapter with a brief overview of constraint programming (Section 2.4) and abstract argumentation (Section 2.5).

2.1 Propositional and First-order Logic

Propositional logic is one of the first standard formalisms for knowledge representation not only in AI but also in human history, dating back to the ancient Greek philosophers. Propositional logic encodes statements about a domain as atomic propositions corresponding to variables either true or false. For example, we can encode the statements "It is raining." and "Take the umbrella." as the atomic propositions *Rain* and *Umbrella* respectively. A set of logical operators allows us to describe the relations between the truth values of variables. This type of statement is typically used to infer formulae corresponding to true statements. For instance, we can use the propositional variables to build a statement defining the logical relation $Rain \rightarrow Umbrella$, which expresses that taking the umbrella is a logical consequence of rain. Therefore, given this relation, the statement $Rain = true$, and the query $Umbrella = ?$, an automated reasoning system for propositional logic should be able to prove from the two statements the statement $Umbrella = true$.

First-order logic extends propositional logic with structured knowledge by allowing variable objects within sentences, and quantification over them. For example, in first-order logic we can parametrize taking the umbrella to the conditions of the item, for instance by using $Broken(x)$ to denote a broken umbrella, represented by variable x. Quantification over variable objects is asserting that some property holds for *all* objects (symbol $\forall$) or *some* objects (symbol $\exists$). With quantifiers, we can express that taking the umbrella holds *for all* umbrellas that are not broken: $\forall x : Rain \wedge \neg Broken(x) \rightarrow Umbrella(x)$. First-order logic quantifiers range over a given domain, namely a (possibly infinite) set of objects to which the theory applies, for example we can use a domain of two umbrellas: A and B. From the statements $Rain = true$, $Broken(A) = true$, $Broken(B) = false$, $Umbrella(x) =?$, an automated reasoning system should thus infer $Umbrella(A) = true$, $Umbrella(B) = false$. A first-order logical theory is usually paired to a domain describing the (possibly infinite) set of objects over which quantifications range. Then we can express that taking the umbrella holds for all umbrellas that are not broken: $\forall x : Rain \wedge \neg Broken(x) \rightarrow Umbrella(x)$. First-order logic lays the formal foundations of any automated reasoning system that automates logical reasoning.

Propositional and first-order logic are a special formalism for knowledge representation and automated reasoning, because they can be used not only to specify the problem, but also to define the rules for the automated logical inference that derives the solution. This overlap is peculiar of propositional and first-order logic and does not apply to other formalisms, for instance *Constraint Programming*.

2.2 Logic Programming

Logic programming (Lloyd 1987) is a declarative framework based on the idea that first-order logic can both represent knowledge and describe the desired output. Knowledge is represented by logical statements about the domain at hand, while the output is the truth value of a logical statement. The application of the logical operators to the given knowledge allows the inference (proof) of such value. In logic programs statements are either facts, that is, assertions of true atomic propositions, or rules, that is, implications of the form $h \leftarrow b_1, b_2, \ldots, b_n$. h is called the *head* of the rule and $b_1, b_2, \ldots, b_n$ is the *body* denoting the logical conjunction of literals b_i, that is, an atomic formula or its negation. Variables in logic programming rules are implicitly universally quantified, i.e. $umbrella(X) \leftarrow broken(X), rain.$ is a statement that holds for all possible objects X. Note that in logic programs uppercase letters denote variables, lowercase letters denote constants and predicates. Replacing variables with the corresponding constant symbols is called *grounding*. An assignment of truth values to the literals of the program is called *interpretation* and when an interpretation satisfies the rules of the program it is called a *model*.

Logic programming extends first-order logic with an important procedural control of reasoning, that is, *negation as failure*. Negation as failure, denoted with *not* or $\sim$, denotes failure at proving a statement. Therefore, while classical logical negation expresses that a sentence is proven to be false, negation as failure succeeds when the truth value of a statement cannot be inferred from the available knowledge. A negated literal $\sim l$ is thus true if all possible proofs for l fail.

Example 2.1. A logic program encoding our example is as follows: a predicate `domain` represents the given set of objects (constants). Along with the facts we add the rule defining the inference rule for `umbrella`. We do not add the information that umbrella b is broken, therefore a solver for this program should infer that $\sim$`broken(b)` is true because there is no information in the program that allows us to infer `broken(b)`.

```
% Facts
domain(a).
domain(b).
rain.
broken(a).
% Rules
umbrella(X) ← domain(X), rain, ∼broken(X).
```

Negation as failure puts logic programming in the context of *non-monotonic reasoning* (Brewka et al. 1997), that is, reasoning on conclusions on the basis of incomplete information: the introduction of additional information may result in retracting previously drawn inferences.

The non-monotonic nature of logic programming introduces the issue of the logical interpretation of negation as failure (Clark 1977) and in particular the interpretation of programs containing loops of rules where two or more predicates are defined by means of interdependent negations as failure. The most popular semantics that define the models of logic programs with negation as failure are the well-founded model semantics and the stable model semantics.

Well-Founded Model Semantics. Well-founded model semantics (Gelder et al. 1991) is a three-valued semantics, that is, logical atoms can be true, false or undefined. A three-valued interpretation is a pair $I = (T,F)$ where T and F are disjoint subsets of the Herbrand base $HB(\mathscr{L})$ of the program $\mathscr{L}$, that is, the set of all ground terms that can be built with the constant and relation symbols of $\mathscr{L}$. The (ground) atoms in T (resp. F) are interpreted as *true* (resp. *false*) in I. The remaining atoms in $HB(\mathscr{L})\backslash(T \cup F)$ are said to be *undefined*. A three-valued model is thus a three-valued interpretation which satisfies all the rules of the program. The well-founded model can

be characterized as the least fixpoint of the operator W_P, an extension of the immediate consequence operator T_P. T_P can be iteratively applied to a partial interpretation I to extend I with the heads of the rules where all literals of the body are interpreted true in I. For programs without loops involving negation, the least fixpoint of T_P is a two-valued well-founded model. When this is not the case, the three-valued well-founded model is the least fixpoint of W_P, which combines T_P with the operator U_P, denoting the *greatest unfounded set* w.r.t. an interpretation I. While T_P provides information about true atoms, U_P infers the atoms that are false (more details in Gelder et al. 1991).

Stable Model Semantics. In stable model semantics (Gelfond and Lifschitz 1988) models are two-valued, but a logic program can have more than one stable model. Given a logic program $\mathscr{L}$ and a set S of atoms interpreted as $true$, called *candidate model*, stable model semantics is defined in terms of the *reduct* of a program $\mathscr{L}$ w.r.t. S. The *reduct* of P w.r.t. S, $\mathscr{L}^S$, is defined as follows: 1) for all $r \in \mathscr{L}$ remove r from $\mathscr{L}$ if $a \in S$ is negated in the body of r; 2) for all $r \in \mathscr{L}$ remove from the body of r all $\neg a$ if $a \notin S$. If S is a minimal model for $\mathscr{L}^S$ then S is a *stable model* (*answer set*) for $\mathscr{L}$.

Example 2.2. Consider the following program:

```
umbrella(a) ← ∼umbrella(b).
umbrella(b) ← ∼umbrella(a).
```

In the well-founded model semantics both `umbrella(a)` and `umbrella(b)` are interpreted as *undefined*. On the other hand, stable model semantics capture the intended meaning of the program, namely that one of the two umbrella should be taken: the reduct for $S = \{umbrella(a)\}$ is the program `umbrella(a) ← .` for which S is a minimal model, therefore S is a stable model. The same holds by replacing a with b.

2.3 Probabilistic Logic Programming

Probabilistic logic programming (PLP) extends logic programs with statements about uncertain information. In real-world scenarios the information available is often incomplete or questionable, therefore the ability to take into account uncertainty is fundamental in such situations. Uncertain knowledge is represented in probabilistic logic programs by annotating facts with probabilities, i.e. $p :: rain.$ denotes that the literal *rain* is in the knowledge base with probability p and not included with probability $1-p$.

Probabilistic inference in PLP usually consists in querying the automated reasoning framework for the probability of a statement being true.

Example 2.3. For example, we could query a PLP system for the probability of taking the umbrella given the uncertainty over the truth of *rain*.

```
% Facts
domain(a).
domain(b).
broken(a).
% Probabilistic fact
0.6::rain.
% Rules
umbrella(X) ← domain(X), rain, ∼broken(X).
% Query
query(umbrella(a)).
```

In addition to this, PLP systems, for example ProbLog (Fierens et al. 2011), typically offer several reasoning and learning algorithms peculiar of Statistical Relational Learning (De Raedt et al. 2016). Statistical Relational Learning extends probabilistic graphical models, e.g Bayesian Networks (Pearl 2006), with logical and relational expressions, as for example Markov logic (Richardson and Domingos 2006). In this setting, probabilistic inference tasks also tackled by PLP are, for example, computing marginal probabilities given some evidence and learning probabilities from (partial) interpretations. ProbLog (De Raedt et al. 2007) is a probabilistic language extending Prolog, where facts and clauses are annotated with (mutually independent) probabilities. Probabilistic logic programs based on distribution semantics, such as ProbLog, can be viewed as a "programming language" generalization of Bayesian Networks (Sato 1995).

We model uncertainty in logic programs by annotating facts with probabilities: the probabilities of facts are mutually independent and each *probabilistic fact*, i.e. $p :: f.$, corresponds to an atomic choice, that is, a choice between including f in the program (with probability p) or discarding it $(1-p)$. A *probabilistic normal logic program* $\mathscr{L} = F \cup R$ is thus a set F of facts annotated with probabilities plus a set of logic rules R. The set of facts F is disjoint from the heads of the rules R, nonetheless rule heads can be annotated as syntactic sugar: $p :: h \leftarrow b_1, \ldots, b_n.$ is equivalent to a new fact $p :: f.$ plus $h \leftarrow f, b_1, \ldots, b_n.$

Example 2.4. The well-known alarm Bayesian network (Pearl 1989) can be encoded as a ProbLog program. An alarm is triggered by either an earthquake or a burglary: if the neighbour is at home you will receive a call. The neighbour at home, the earthquake, and the burglary are events occurring with a certain probability, thus encoded as probabilistic facts. Probabilistic facts are marginally independent and therefore correspond to nodes that do not have common parents or ancestors. The logic

rules determine whether you get a call or not, and thus correspond to the conditional dependencies of the Bayesian network.

```
% Probabilistic facts
0.1::burglary.
0.2::earthquake.
0.5::neighbour_at_home.
% Rules
alarm ← earthquake.
alarm ← burglary.
neighbour_calls ← alarm, neighbour_at_home.
query(alarm).
query(neighbour_calls).
```

Distribution Semantics. A probabilistic logic program $\mathscr{L} = F \cup R$ is queried for the likelihood of atoms. The probability of an atom is commonly defined by the *distribution semantics* (Sato 1995). A *total choice* ω is a combination of atomic choices over all probabilistic facts, that is, a subset of the set of all ground facts F, i.e. $\omega \subseteq F$. We denote the set of all total choices of $\mathscr{L}$, i.e. the power set of F, with $\Omega_{\mathscr{L}}$. The probabilities of the facts define a probability distribution over the $2^{|F|}$ total choices. The possible non-probabilistic subprograms $\omega \cup R, \omega \in \Omega_{\mathscr{L}}$, which we call *possible worlds*, are obtained from $\mathscr{L}$ by including or discarding a fact according to the corresponding atomic choice. An interpretation is an assignment of a truth value to all atoms in the Herbrand base of a program. An interpretation is called a model of the theory if it satisfies all formulas in the theory.

Example 2.5. Consider the alarm example (Example 2.4), there are $2^3 = 8$ possible worlds (listed in Example 2.6), corresponding to the combinations of choices for the probabilistic facts *earthquake*, *burglary*, *neighbour_at_home*. Consider the possible world where we discard *earthquake* and include the other two: then the alarm will be triggered, i.e. *alarm* is inferred from *burglary*, and thus the neighbour calls, because both *alarm* and *neighbour_at_home* are true. The model for the possible world denoted by $\{burglary, neighbour_at_home\}$ is thus $\{burglary, neighbour_at_home, alarm, neighbour_at_home\}$.

The distribution semantics relies on a one-to-one mapping between possible worlds and models: many PLP frameworks, e.g. ProbLog, PRISM (Sato and Kameya 2008) and LPAD/CP-logic (Vennekens et al. 2009), restrict the class of valid inputs to programs where each total choice corresponds to a two-valued well-founded model (Gelder et al. 1991). The underlying assumption is that choices are modelled exclusively by means of probabilistic facts, and that they are marginally independent. Under this assumption

many algorithms and reasoning techniques have been developed to perform inference and parameter learning tasks.

Inference. The MARG inference task is the task of computing the probability of success of a query, that is, the sum of the probability of each possible world where the query is true, under the (possibly empty) given evidence. Evidence is a set of atoms whose truth value is known. In particular:

Definition 1 (The MARG Inference task)**.**
Given

- *A program $\mathscr{L}$: let G be the set of all ground (probabilistic and derived) atoms of $\mathscr{L}$.*
- *A set $E \subseteq G$ of observed atoms (evidence), along with a set e of corresponding observed truth values ($E = e$).*
- *A set $Q \subseteq G$ of atoms of interest (queries).*

Find *the marginal distribution of every query atom given the evidence, i.e. computing $P(q \mid E = e)$ for each $q \in Q$.*

In the ProbLog2 system probabilistic inference is reduced to a *weighted model counting* (WMC) problem (Cadoli and Donini 1997). Weighted model counting is the problem of computing the weight of a propositional logic formula φ, given a weight function w that assigns a positive (real) weight value to each literal v and $\neg v$ ($v \in G$). Let $MOD(\mathscr{L})$ be the set of valid interpretations for $\mathscr{L}$, the *WMC* of φ is:

$$WMC_{\mathscr{L}}(\varphi) = \sum_{M \in MOD(\mathscr{L}), M \models \varphi} \prod_{l \in M} w(l)$$

The probability $P(q|E = e)$ is computed as $WMC_{\mathscr{L}}(\varphi)$ where φ is the propositional representation of $q \wedge E = e$. The weight function is defined as follows: for all probabilistic facts $(p : f)$, $w(f) = p$ and $w(\neg f) = 1 - p$; for all atoms $a \in E$ if a is true (false) $w(a) = 1$, $w(\neg a) = 0$ (resp. $w(a) = 0$, $w(\neg a) = 1$); for all remaining (logical) atoms a, $w(a) = w(\neg a) = 1$.

Example 2.6. The possible world of Example 2.4 corresponding to {*burglary*, *neighbour_at_home*} has probability $0.1 \cdot (1 - 0.2) \cdot 0.5 = 0.04$. In order to find the probability of being called we sum the probabilities of all other possible worlds whose model contains *neighbour_calls*, namely {*earthquake*, *neighbour_at_home*} and {*burglary*, *earthquake*, *neighbour_at_home*}, which are respectively $(1 - 0.1) \cdot 0.2 \cdot 0.5 = 0.09$ and $0.1 \cdot 0.2 \cdot 0.5 = 0.01$. Therefore, $P(neighbour_calls) = WMC(neighbour_calls) = 0.04 + 0.09 + 0.01 = 0.14$.

Possible world's choices ω	Model	$P(\omega)$
$\{\}$	ω	0.36
$\{neighbour_at_home\}$	ω	0.36
$\{burglary\}$	$\omega \cup \{alarm\}$	0.04
$\{earthquake\}$	$\omega \cup \{alarm\}$	0.09
$\{neighbour_at_home, burglary\}$	$\omega \cup \{alarm, neighbour_calls\}$	**0.04**
$\{burglary, earthquake\}$	$\omega \cup \{alarm\}$	0.01
$\{earthquake, neighbour_at_home\}$	$\omega \cup \{alarm, neighbour_calls\}$	**0.09**
$\{burglary, earthquake,$ $neighbour_at_home\}$	$\omega \cup \{alarm, neighbour_calls\}$	**0.01**

The task of model counting is #P-complete in general, therefore the logic program is transformed into a representation where this task becomes tractable (Fierens et al. 2011). $\mathscr{L}$ is in fact transformed by means of knowledge compilation into a *smooth d-DNNF* formula (deterministic Decomposable Negation Normal Form) (Darwiche 2004) or a *sentential decision diagram* (Choi et al. 2013). In this thesis we focus on d-DNNFs because we base our approach on a knowledge compiler to d-DNNFs for stable model counting (Aziz et al. 2015).

Definition 2. *A* NNF *is a rooted directed acyclic graph in which each leaf node is labelled with a literal and each internal node is labelled with a disjunction or conjunction. A smooth* d-DNNF *is an NNF with the following properties:*

- *Deterministic: for all disjunctive nodes the children represent formulas pairwise inconsistent.*
- *Decomposable: the subtrees rooted in two children of a conjunction node do not have atoms in common.*
- *Smooth: all children of a disjunction node use the same set of atoms.*

On d-DNNFs the task of model counting becomes tractable. The d-DNNF is further transformed into an equivalent arithmetic circuit, by replacing conjunctions and disjunctions respectively with multiplication and summation nodes, and by replacing leaves with the weight of the corresponding literals. Arithmetic circuits allow us to efficiently perform the task of *WMC*.

Example 2.7. Consider the rule $neighbour_calls \leftarrow alarm, neighbour_at_home$. The corresponding smooth d-DNNF and arithmetic circuit are represented in Figure 2.1. The root of the circuit represents all possible assignments to the variables. The leftmost and-node corresponding to the assignment *calls* corresponds in the arithmetic circuit to the weight 0.14, the probability of it being true. Vice versa, the top and-node, corresponding to the assignment $\neg calls$, represents in the arithmetic circuit the weight

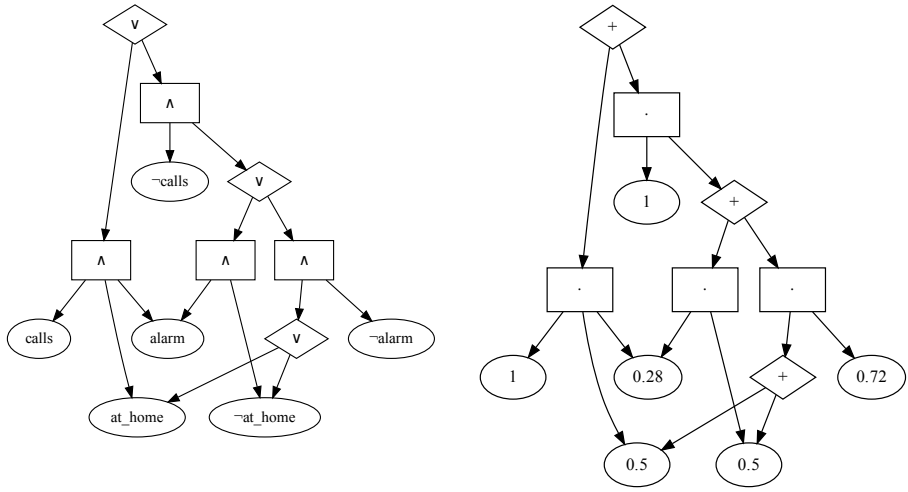


Figure 2.1: Smooth d-DNNF for the implication *neighbour_calls* ← *alarm*, *neighbour_at_home* (left) and the corresponding arithmetic circuit (right).

0.86. We simplified Example 2.4 by considering *alarm* a probabilistic fact with probability 0.28, which is the probability that can be inferred from *burglary* and *earthquake* (cfr. Example 2.6).

Algebraic Model Counting Algebraic Model Counting (AMC) is a general framework for quantitative reasoning over models that generalizes weighted model counting to the semiring setting (Kimmig, Van den Broeck, et al. 2017a).

Definition 3. *A* monoid $\mathscr{M} = (M, \otimes, e_\otimes)$ *consists of an associative binary operation* $\otimes$ *with neutral element* $e_\otimes$ *that operates on elements of* M*. A commutative* semiring $\mathscr{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$ *consists of two commutative monoids* $(S, \oplus, e_\oplus)$ *and* $(S, \otimes, e_\otimes)$ *such that* $\otimes$ *right and left distributes over* $\oplus$ *and* $e_\oplus$ *annihilates* S*, i.e.* $\forall s \in S : s \otimes e_\oplus = e_\oplus = e_\oplus \otimes s$.

Some examples of well-known commutative semirings are

- $\mathscr{P} = ([0,1], +, \cdot, 0, 1)$, the probability semiring,
- $\mathscr{S}_{\max,\cdot} = (\mathbb{R}_{\geq 0}, \max, \cdot, 0, 1)$, the max-times semiring,
- $\mathscr{S}_{\max,+} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$, the max-plus semiring,

- EU $= (\{(p, eu) \mid p \in [0,1], eu \in \mathbb{R}\}, +, \otimes, (0,0), (1,0))$, the expected utility semiring, where addition is coordinate-wise and $(a_1, b_1) \otimes (a_2, b_2) = (a_1 \cdot a_2, a_2 \cdot b_1 + a_1 \cdot b_2)$.

An AMC instance $A = (\mathscr{T}, \mathscr{S}, \alpha)$ consists of a theory $\mathscr{T}$ over variables $\mathbf{X}$, a commutative semiring $\mathscr{S}$ and a labeling function $\alpha : lit(\mathbf{X}) \to S$. The value of A is

$$AMC(A) = \bigoplus_{M \in mod(\mathscr{T})} \bigotimes_{l \in M} \alpha(l)$$

Inference in probabilistic logic programming is a task that can be formulated as an AMC task, which uses the probability semiring $\mathscr{P}$ and the logic theory from the probabilistic logic program.

Learning. Learning from interpretations of parameters in a ProbLog program is implemented in a likelihood maximization setting. In the *learning from interpretations* task the input is a program $\mathscr{L}$ where some probabilities are unknown (parameters), and a set of interpretations for $\mathscr{L}$. Interpretations can be partial, that is, the observation of truth values contains only a subset of all atoms. The goal is to estimate the value of the parameters such that the predicted values maximize the likelihood of the given interpretations:

Definition 4 (Max-Likelihood Parameter Estimation)**.**
Given

- *A program $\mathscr{L}(\mathbf{p}) = F \cup R$ where F is a set of probabilistic facts and R contains the rules describing the background knowledge. $\mathbf{p} = \langle p_1, ..., p_N \rangle$ is a set of unknown parameters attached to probabilistic facts.*
- *A set $\mathbb{I}$ of (partial) interpretations $\{I_1, ..., I_M\}$ as training examples.*

Find *the maximum likelihood probabilities $\widehat{\mathbf{p}} = \langle \widehat{p_1}, ..., \widehat{p_N} \rangle$ for the interpretations in $\mathbb{I}$. Formally,*

$$\widehat{\mathbf{p}} = \arg\max_{\mathbf{p}} P(\mathbb{I} | \mathscr{L}(\mathbf{p})) = \arg\max_{\mathbf{p}} \prod_{m=1}^{M} P(I_m | \mathscr{L}(\mathbf{p}))$$

Parameters are iteratively updated by an alternation of an expectation step (E step) which computes the expected value of the learnable parameters under the current model, and the maximization step (M step) which updates the parameters estimates to maximize the likelihood of the observed data.

Example 2.8. Consider Example 2.7 but now the probability of *alarm* is a *learnable parameter*. In ProbLog the expression $t(0.4) :: alarm.$ denotes a learnable probability which is initialized at 0.4 in the EM algorithm:

```
t(0.4)::alarm.
0.5::at_home.
calls ← alarm, at_home.
```

Given a set of interpretations reflecting the original probability distribution we can learn the probability of *0.28* :: *alarm*. even if the observation of alarm is not given. In fact, we provide three types of partial interpretations:

I_1: $\{evidence(calls,false).evidence(at_home,false).\}$ (50 examples)

I_2: $\{evidence(calls,true).evidence(at_home,true).\}$ (14 examples)

I_3: $\{evidence(calls,false).evidence(at_home,true).\}$ (36 examples)

The first corresponds to the possible worlds where *at_home* prevents the call regardless of *alarm* (50% of the examples corresponding to the probability of $at_home = false$). The second is the case where both are true and the third where *alarm* is false. The first iteration of the EM algorithm considers the frequency of each example and the probability according to the model with the probability of *alarm* initialized at 0.4. The iteration 0 then associates to I_1 the current probability of *alarm* being either true or false, that is, 0.4. On the other hand, in I_2 (resp. I_3) *alarm* has probability 1 (resp. 0) of being true (E step). The updated probability that maximizes the probability of the set of training examples is thus $\frac{50\cdot 0.4+14\cdot 1+36\cdot 0}{100} = 0.34$ (M step). Iteration 1 then repeats the E step with the updated probability 0.34 :: *alarm*. The probabilities of *alarm* in I_1, I_2 and I_3 are respectively $0.34, 1.0$ and 0 (E step) and the new updated probability is $\frac{50\cdot 0.34+14\cdot 1+36\cdot 0}{100} = 0.31$ (M step). Further iterations converge to the true probability of 0.28 :: *alarm*.

Causal effects. An alternative view of ProbLog that we exploit in this thesis is in terms of CP-logic, a logical language for representing probabilistic causal laws. Probabilistic causal laws model the probability distribution of a set of random variables that are related by a causal process, that is, the variables interact through a sequence of non-deterministic or probabilistic events. ProbLog and CP-logic are closely related as the syntax of a ProbLog program is similar to that of a CP-theory in which each rule has precisely one head atom. Moreover, the semantics of a ground ProbLog program coincides completely with the semantics of CP-Logic. This close relation makes available in ProbLog a modelling technique from CP-logic that we exploit in this thesis: the *inhibition effect* (Meert and Vennekens 2014) and *negations in the head* (Vennekens 2013).

Negation in the head gives an interpretation of the negation of heads from Answer Set Programming in the context of epistemic reasoning and logic theories defining causal

mechanisms. Each rule $\neg h \leftarrow b_1, \ldots, b_n.$ is interpreted by replacing all heads h in the program with a new atom h_{pos} and all heads $\neg h$ with a new atom h_{neg}, and the rule $h \leftarrow h_{pos}, \sim h_{neg}.$ is added. This last rule thus defines that h can be inferred from any of the causes of h (making h_{pos} true) only if there is no cause for believing in h_{neg}, the opposite of h. The inhibition effect is the resulting decrease in the probability of h when the probability of $\neg h$ (h_{neg}) increases.

The *noisy-or* effect (Meert and Vennekens 2014) describes the aggregation of the causal probabilities for an atom which appears as the head of multiple (causal) rules.

Example 2.9. We adapt the travel example from (Meert and Vennekens 2014) to describe the inhibition effect in ProbLog. Alice is travelling during a pandemic and therefore there is a chance of getting infected. This probability is influenced by two factors: first, if Alice travels with public means of transport there is some probability of being infected there. Second, if Alice is vaccinated then there is a chance that she will not be infected. We encode this last rule with the inhibition effect, which describes a decrease in the probability of infection when vaccinated.

```
0.2::riskyTravel(alice).
0.7::vaccinated(alice).
0.4::infected(alice) ← riskyTravel(alice).
0.33::¬infected(alice) ← vaccinated(alice).
```

The two rules correspond to the following rewriting:

```
0.4::infected_pos(alice) ← riskyTravel(alice).
0.33::infected_neg(alice) ← vaccinated(alice).
infected(alice)←infected_pos(alice),∼infected_neg(alice).
```

2.4 Constraint Programming

Constraint programming is a form of declarative programming oriented at solving combinatorial problems. In CSPs knowledge is represented as statements expressing relations between the variables (constraints). Constraint satisfaction problems (CSPs) define a set of decision variables, a domain of values that the variables can take, and a set of constraints defining the valid assignments for the variables. Inference in CSPs consists in finding an assignment to the variables compatible with the given constraints.

Definition 5. *A* Constraint Satisfaction Problem *(Rossi et al. 2006),* CSP*, P is a triple $P = \langle V, D, C \rangle$ where $V = \langle v_1, \ldots, v_n \rangle$ is an n-tuple of variables, $D = \langle D_1, \ldots D_n \rangle$ is a corresponding set of domains such that $v_i \in D_i$, C is a set of constraints.*

Let D_W denote the subset of D corresponding to variables W.

Definition 6. *A* constraint *is a pair* (W, R_W) *where* $W \subseteq V$ *is an m-tuple of variables in* V *and* R_W *is a relation of arity m over the Cartesian product of the domains in* D_W.

Usually constraints are expressed compactly by means of a *constraint language*, for example:

Example 2.10. Given three variables v_1, v_2, v_3 and their domain $\{1,2,3\}$, we can query a constraint satisfaction system for the valid assignments under the constraints $C = \{v_1 < 3, v_2 \neq v_3\}$. An assignment maps variables to elements from the respective domains.

Definition 7. *An* assignment *for* $W \subseteq V$ *is a function* $f : W \to D_1 \times \cdots \times D_m$ *mapping variables to elements in the respective domains:* $f(\langle v_1, \ldots, v_m \rangle) = \langle d_1, \ldots, d_m \rangle$, $d_i \in D_i, D_i \in D_W$.

An assignment that satisfies all the constraints in C is a solution of the CSP.

Definition 8. *An assignment* f satisfies *a constraint* $c = (W, R_W)$, $f(W) \vdash c$, *when* $f(W) \in R_W$. f *is a* partial *assignment when* $W \subset V$, *or* total *when* $W = V$.

If there is no solution, then the problem is *unsatisfiable*. Reasoning over CSPs reduces to a systematic search of the solution space by using inference techniques that allow an efficient exploration, such as constraint propagation, backtracking or local search (Rossi et al. 2006). Other reasoning tasks in CSPs include *optimization* problems, where the goal is to find the optimal (valid) assignment with respect to some optimization criterion, for example minimizing a variable representing a cost or maximizing a variable representing a profit. Counting Constraint Satisfaction Problems is another important reasoning task connected to CSPs.

As (weighted) model counting is the task of counting the (weighted) satisfying assignments of a (weighted) logic theory, a counting Constraint Satisfaction Problem, #CSP, is the task of counting the number of satisfying assignments of a CSP. We refer to the set of all solutions of a #CSP $P = \langle V, D, C \rangle$ as $M(V, D, C)$. The goal is then to find $MC(V, D, C) = |M(V, D, C)|$. This notation derives from probabilistic lifted inference and weighted *model* counting, where the satisfying assignment is called a (well-founded, stable) *model*.

Example 2.11. Given a #CSP $\langle V, D, C \rangle$ where $V = \langle v_1, v_2 \rangle$ and $D_1 = D_2 = \{d_1, d_2, d_3\}$, $C = \{v_2 \in \{d_1, d_2\}\}$, there are 6 possible solutions: $\langle d_1, d_1 \rangle$, $\langle d_2, d_1 \rangle$, $\langle d_3, d_1 \rangle$, $\langle d_1, d_2 \rangle$, $\langle d_2, d_2 \rangle$, $\langle d_3, d_2 \rangle$ hence $MC(V, D, C) = 6$.

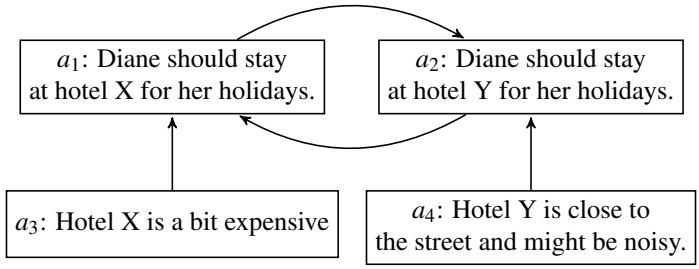


Figure 2.2: Abstract argumentation framework. Edges represent attacks, nodes are arguments.

2.5 Abstract Argumentation

An *abstract argumentation framework* (or *argument graph*) (Dung 1995) is a pair (A,R) where A is a set of arguments and $R \subseteq A \times A$ is a binary (attack) relation over A. Figure 2.2 represents an example from Hunter, Polberg, et al. (2020) of argument graph (A,R) where $A = \{a_1,a_2,a_3,a_4\}$ and $R = \{(a_1,a_2),(a_2,a_1),(a_3,a_1),(a_4,a_2)\}$.

In recent years several extensions and modifications of this formulation have been proposed in order to encode more complex argumentation reasoning systems. Among the most relevant, we find *bipolar* argument frameworks (Amgoud, Cayrol, et al. 2004), *weighted* argumentation frameworks (Amgoud, Ben-Naim, et al. 2017), and *probabilistic* argumentation frameworks (Li et al. 2011).

Bipolar argumentation frameworks introduce a *support* relation R^+ along with the attack relation R^-. A bipolar argument graph is thus a triple (A,R^-,R^+), where A is a set of arguments and $R^- \subseteq A \times A$ (resp. $R^+ \subseteq A \times A$) is a binary attack (support) relation over A.

Weighted argumentation frameworks (A,R,w) augment the traditional argument graph (A,R) with a weighting function $w : A \to [0,1]$ and belong to a general class of *gradual* argumentation frameworks (Cayrol and Lagasquie-Schiex 2005), where arguments' acceptability is evaluated on a fine-grained numerical scale or ranking.

Gradual and bipolar frameworks are combined into *Quantitative Bipolar Argumentation Frameworks* (QBAFs) (Baroni et al. 2019), where each argument has a (possibly empty) set of attackers, a (possibly empty) set of supporters, and an initial evaluation (possibly the same for all arguments) on a chosen scale. These elements contribute to a final argument evaluation, provided by a strength function on the chosen scale. A QBAF is thus a quadruple (A,R^-,R^+,τ) consisting of a finite set A of arguments, a binary (attack) relation R^- on A, a binary (support) relation R^+ on A and a total function

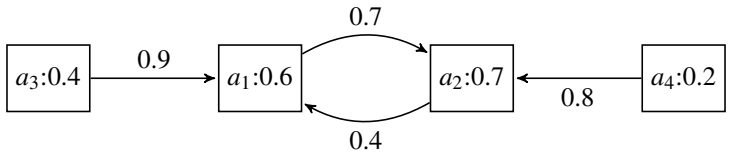


Figure 2.3: In probabilistic abstract argumentation framework nodes and attacks are annotated with values in $[0,1]$ (cfr. Figure 2.2).

$\tau : A \to I$, where I is the chosen scale. $\tau(a)$ is thus the base score of a, for any $a \in A$.

Finally, a *probabilistic argument graph* (Li et al. 2011) is a tuple (A,R,P_A,P_R) where: (A,R) is an argument graph and P_A and P_R are functions: $P_A : A \to [0,1]$ and $P_R : A \times A \to [0,1]$. Figure 2.3 describes a probabilistic argument graph derived from the example in Figure 2.2. The values assigned by the functions P_A and P_R are usually interpreted with two different approaches: the *constellations* approach and the *epistemic* approach (Hunter 2013). The constellations approach interprets the probabilities as uncertainty over the structure of the graph and models a probability distribution over subgraphs. Each subgraph is evaluated according to the traditional extension-based semantics (Dung 1995). An extension is a subset of the arguments that is acceptable w.r.t. a given criterion, therefore a distribution over acceptable arguments is derived from 1) the probability distribution over the possible subgraphs, and 2) the subset of acceptable arguments in each subgraph. On the other hand, the *epistemic* approach interprets the probabilities as a direct measure of an agent's belief in the arguments. The accepted arguments are those having probability higher than 0.5, and the argument graph is used to reason about the consistency of the agent's beliefs with respect to the relations encoded in the graph.

Part I

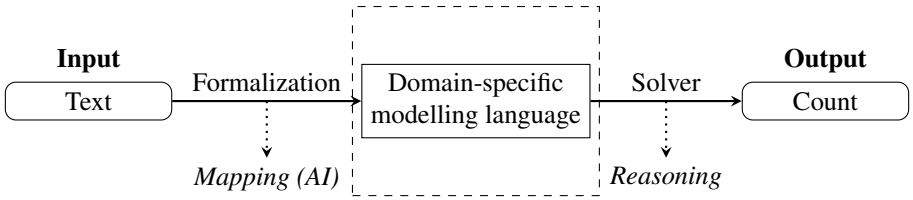
Combinatorial Problem Solving

Chapter 3

Combinatorics Math Word Problems: Modelling

The paper on which this chapter is based on was previously published as:

P. Totis, J. Davis, L. D. Raedt, and A. Kimmig (2023). "Lifted Reasoning for Combinatorial Counting". In: *J. Artif. Intell. Res.* 76, pp. 1–58. DOI: 10.1613/jair.1.14062. URL: https://doi.org/10.1613/jair.1.14062



One of the fundamental goals of AI is to outperform humans on tests associated with intelligence and advanced cognitive skills, such as puzzles (Chesani et al. 2017), games (Silver et al. 2016) and math problems (Dries et al. 2017; Mitra and Baral 2016; Roy and Roth 2018). Among math problems, combinatorics math problems have received less attention in the problem-solving AI literature despite posing interesting

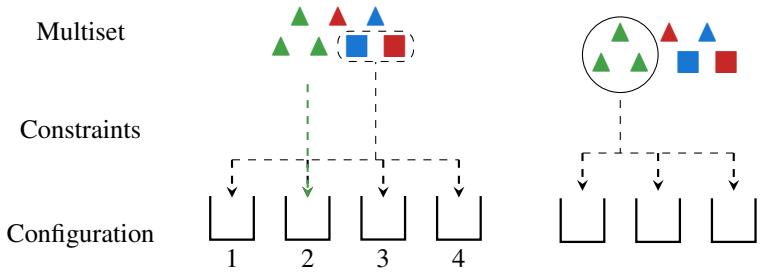


Figure 3.1: Visualization of **P1** (left) and **P2** (right).

and relevant challenges. In combinatorics math problems the task is to count how many configurations of a finite set of objects satisfy a given set of constraints. For instance:

P1. A kit of toy shapes contains five triangles and two squares. One triangle and one square are red. Another triangle and the other square are blue and the remaining triangles are green. In how many different rows of four objects can the shapes be arranged if the two squares are included and the second object is green?

P2. Given the same set of shapes as **P1**, in how many ways can the objects be divided into three (non-empty) groups such that the green objects all belong to the same group?

These examples fall into the category of math word problems (Zhang et al. 2020), which are concerned with the task of solving a math problem described in natural language. This task presents multiple challenges ranging from interpreting the text in natural language down to defining a solving procedure. Dries et al. (2017) and Suster et al. (2021) introduced a two-step approach for probability problems showing that this two-step approach can solve more problems than using an end-to-end neural model. The modelling language is formal and should allow to declaratively represent and reason about the math word problems under consideration.

Therefore, in order to apply a declarative approach to combinatorics math problems, there are two key points to address:

- Modelling: design a modelling language for modelling combinatorics problems.
- Reasoning: design a solver that is able to (efficiently) find solutions to such problems.

For combinatorics math problems, there are currently no elegant solutions for neither modelling nor reasoning. Declarative models should directly support primitives for standard concepts from combinatorics (such as permutations, repeated objects, sets,...). Furthermore, solvers should be able to count without enumerating all solutions, that is, solvers should work at the lifted level.

While combinatorial counting problems can sometimes be encoded in particular modelling languages (such as logic and CSPs) and even solved, the encodings are, as we shall show in Section 3.1, often indirect, cumbersome and complicated, or the solvers are inefficient as they enumerate all solutions. We will say that such frameworks do not directly support combinatorial counting.

Modelling combinatorics math problems requires support for three fundamental primitives: 1) multisets, 2) configurations and 3) constraints. First, multisets are fundamental for representing the objects in the domain, to model identical (repeated) objects (e.g. green triangles in **P1**). Second, configurations define how these objects are grouped together. We distinguish *level 1* configurations, where there is a single group of objects, such as a set or a permutation (e.g. "rows" in **P1**), from *level 2* configurations, where grouping is nested, such as a set of sets, also called partitions (e.g. "objects [...] divided into three (non-empty) groups" in **P2**). We also distinguish between *ordered* and *unordered* configurations, e.g. permutations or sets. Third, constraints

Problem	Multiset	Configuration	Constraints
P3. A shipment of 12 different TVs contains 3 defective ones. In how many ways can a hotel purchase 5 of these TVs and receive at least 2 of the defective ones? **Solution:** $\binom{3}{2} \cdot \binom{9}{3} + \binom{3}{3} \cdot \binom{9}{2} = 288$	set $\{TV_1, \ldots, TV_{12}\}$	set	set of size 5 with at least 2 defective TVs
P4. Fourteen construction workers are to be assigned to three different tasks. Seven workers are needed for mixing cement, five for laying bricks, and two for carrying the bricks to the brick layers. In how many different ways can the workers be assigned to these tasks? **Solution:** $\binom{14}{7} \cdot \binom{7}{5} \cdot \binom{2}{2} = 72072$	set $\{W_1, \ldots, W_{14}\}$	set of sets	3 groups of fixed sizes (7, 5, 2)
P5. In how many distinguishable ways can the letters in B A N A N A be written? **Solution:** $\frac{6!}{1!\cdot 3!\cdot 2!} = 60$	multiset $\{3 \times A, 2 \times N, 1 \times B\}$	permutation	

Table 3.1 : Combinatorics math problems from Dries et al. (2017). The solutions are obtained by applying counting rules based on the sizes of the groups of objects defined.

specify desired properties of the configuration (e.g. "the second object is green" in **P1**). Table 3.1 includes a selection of real-world examples of combinatorics math problems from the dataset of Dries et al. (2017), along with the required modelling primitives. Existing languages such as logic-based declarative languages and CSPs do not directly support the modelling of multi-sets and certain types of configurations, (cf. Section 3.1).

As we will discuss in Chapter 4, the reasoning tools for these languages are also unsatisfactory because they resort to propositional or enumeration-based counting techniques. On combinatorics math problems, these are highly inefficient because reasoning, in particular counting, is done at the level of individual objects. In contrast, humans typically exploit higher-level information about groups and independent subproblems, which is akin to probabilistic lifted inference techniques (Van den Broeck, Kersting, et al. 2021). However, they are not directly applicable to combinatorics math problems (Section 4.1.2).

Combinatorics math problems thus represent an interesting position between modelling and reasoning under constraints and the lifted counting strategies developed in the context of (probabilistic) first-order logic theories. Unfortunately, both approaches fall short w.r.t. the modelling and/or reasoning requirements, as they do not directly support the necessary primitives for combinatorics math problems. This is no surprise as lifted inference languages were designed to model joint probability distributions using first-order logic, which does not directly support multisets, configurations, and constraints. CSP languages typically have direct support for sets, constraints, and some types of configurations, but do not directly support multisets. Moreover, CSP solvers are limited too, because counting is typically done by enumeration. Lifted probabilistic inference techniques can be applied to some types of problems, but cannot exploit many high-level reasoning opportunities offered by the constraints typically used in combinatorial counting.

In this chapter we focus on the first key point of a declarative approach to combinatorics math problems: *modelling*.

3.1 Related Work

We begin with discussing the limitations of traditional declarative frameworks concerning modelling combinatorics math problems. We consider CSPs, for their constraint-oriented modelling approach, and probabilistic reasoning frameworks, for their lifted counting techniques based on grouping and symmetries. We focus on the key aspects that most combinatorics math problems share, namely, the presence of 1) multisets, that is, objects with repeated (indistinguishable) copies 2) configurations,

in particular unordered and level 2 configurations and 3) constraints on the given configuration.

3.1.1 Constraint Satisfaction Problems

Past work on CSPs offers modelling languages which directly support sets and constraints, but fails to provide efficient solving techniques for the counting problem.

Primitive types for multisets and partitions (a level 2 configuration) are missing altogether in most popular general-purpose constraint modelling languages, for instance MiniZinc (Nethercote et al. 2007). Zinc (Marriott et al. 2008) instead supports nested types, therefore partitions, but not repeated objects, thus multisets.

On the other hand, ESSENCE (Frisch, Harvey, et al. 2008) is a constraint language designed to specify combinatorial problems. ESSENCE supports a wide range of types for combinatorial problems, including multisets and partitions. There are however limitations regarding multisets. First, a variable can be of type multiset but not an object from a multiset. In fact, given a set S and a size s, we can define a variable of type multiset as any multiset of size s of objects in S (the domain is a set of multisets, level 2). However, variables cannot be of type object from a multiset of objects (the domain is a multiset, level 1). Second, unnamed types, that is, the declaration of a set from a label and a size, do not express the *indistinguishable* repetition of the same object, as multisets do, but rather a set of *interchangeable* objects that are internally labelled. This internal labelling is used in the current interpretation of Conjure (Akgün et al. 2022), solver paired with ESSENCE. To illustrate the effect of Conjure's current design choice, consider a multiset $ns = \{N,N\}$, which repeats the letter N twice. Let ns be the unnamed type in ESSENCE `letting ns be new type of size 2` which is labelled internally to two different constants, e.g. ns_1 and ns_2. Consequently, when searching for the sequences of length 2 of elements in ns, Conjure returns 4 solutions ($[ns_1,ns_1]$,$[ns_1,ns_2]$,$[ns_2,ns_1]$, $[ns_2,ns_2]$), instead of the only sequence $[N,N]$. We would like to emphasize that this is an implementation choice, and future versions of the solver (or another solver designed to work with ESSENCE) may include symmetry breaking on unnamed types that results in removing some or all of these redundant solutions.

3.1.2 Lifted Probabilistic Inference

In probabilistic lifted inference frameworks modelling combinatorics math problems is limited because variables and constants always receive a unique identifier. This prevents a direct support for multisets and unordered configurations, as we show in the following paragraphs.

Lifted probabilistic inference frameworks are based on languages that are not suited for the problem at hand, because they offer primitives to model joint probability distributions rather than combinatorial objects and constraints. In fact, these languages map to a representation based on first-order logic which has no direct support for the three key aspects of combinatorics math problems: multisets, configurations of level 2 and unordered, and constraints. First order logic frameworks reason over a *set* of constants which are always identified with a unique label, therefore multisets are not directly supported. Different variables are also uniquely identified, that is, if v_1 and v_2 represent two different constants then the assignment $v_1 = C_1, v_2 = C_2$ is different from $v_1 = C_2, v_2 = C_1$, therefore an order is induced and unordered configurations are not directly supported.

Example 3.1. Consider **P3**: we can model in first-order logic a 5-subset of 12 TVs by defining a set of 5 variables representing the selected objects. We specify the set of TVs with constants and a unary predicate $tvs/1$, e.g. $tvs(TV_1), \ldots, tvs(TV_{12})$. Then the formula

$$tvs(tv_1) \wedge \cdots \wedge tvs(tv_5) \wedge tv_1 < tv_2 \wedge tv_2 < tv_3 \wedge \cdots \wedge tv_4 < tv_5 \tag{3.1}$$

represents a subset of 5 TVs as follows. Variables $tv_1, \ldots, tv_5$ correspond to 5 TVs from the set $tvs/1$; however, variables are associated with a unique identifier, therefore additional constraints are needed to exclude the (indistinguishable) permutations of the assignments. This is done by defining an order relation $<$ over the constants to exclude for each satisfying assignment for variables tv_i the indistinguishable permutations w.r.t. $<$. As we later show, lifted counting over $<$ is not supported in all lifted reasoning frameworks. Counting the number of assignments to variables tv_i that satisfy Formula 3.1 thus corresponds to the number of subsets of 5 TVs.

Moreover, unique identifiers prevent the encoding of partitions altogether.

Example 3.2. Encoding the unordered groups in **P2** means modelling a set of sets (partition). If each subset is identified by a different predicate as in Example 3.1, e.g. $group_1/1$, $group_2/1$, $group_3/1$, then an ordering is established by the labels. Similarly, if we represent groups as a set of constants G_1, G_2, G_3, e.g. $contains(G_i, x)$ defines that group G_i contains object x, then we order each group with the identifiers of the corresponding constant. Therefore, the lack of indistinguishable objects in first order logic prevents modelling this problem.

The lack of direct support for constraints also prevents lifted reasoning opportunities, for example with aggregate constraints. Aggregate constraints depend only on the set of values assigned to the variables, e.g. if any two of the purchased TVs is defective, rather than the first, the second,... However, without dedicated language constructs the

combinations of specific assignments have to be explicitly modelled, thus preventing reasoning at the lifted level.

Example 3.3. Consider again **P3** and let *df/1* be a predicate identifying defective TVs. Following Example 3.1 we can enforce the number of defective TVs by conjoining the logic formula (3.1) with a formula specifying the possible ways that 2 out of the 5 selected TVs are defective, i.e. belong to the set *df/1*:

$$df(tv_1) \wedge df(tv_2) \wedge \neg df(tv_3) \wedge \neg df(tv_4) \wedge \neg df(tv_5)) \vee$$

$$df(tv_1) \wedge \neg df(tv_2) \wedge df(tv_3) \neg df(tv_4) \wedge \neg df(tv_5)) \vee$$

$$\cdots \vee$$

$$\neg df(tv_1) \wedge \neg df(tv_2) \wedge \neg df(tv_3) \wedge df(tv_4) \wedge df(tv_5)$$

This has to be also repeated for the remaining cases of "at least two", i.e. 3,4 and 5 defective TVs in the purchased set. This means enumerating in the model the possible configurations satisfying the counting constraint, thus preventing a lifted counting.

These limitations are inherent to the formalisms and languages based on (first-order) logic, such as Answer Set Programming, ASP (Gebser, Kaminski, Kaufmann, and Schaub 2012), Prolog (Emden and R. A. Kowalski 1976), and the conjunctive normal form (CNF) required by #SAT solvers (Thurley 2006). In particular, sets and multisets are not first-class citizens of these languages. Therefore, the aforementioned limitations in manipulating these types both as variable objects (Example 3.1) and domains (Example 3.2) arise. ASP offers counting aggregates to compactly encode the number of true literals $l_1, \ldots, l_n$ in an expression of the form `l #count {`l_1`, ..., `l_n`} u`. However, Gebser, Kaminski, Kaufmann, and Schaub (2012) remark that "if the literals do not belong to domain predicates, the value of an aggregate is not known during grounding, in which case *gringo* unwraps all possible outcomes of the aggregate's evaluation". This means that reasoning on aggregate operators over variable configurations, such as counting, still resorts to explicitly grounding out the different combinations and reasoning over individual objects. Consider for instance the following example:

Example 3.4. The following ASP program selects 3 TVs out of a set of 3 defective and 3 working TVs with a counting aggregate of defective TVs:

```
defective(tv1). defective(tv2). defective(tv3).
working(tv4).   working(tv5).   working(tv6).
tvs(X)←defective(X).
tvs(X)←working(X).
```

```
3 {purchase(N): tvs(N)} 3.
n_def(C)←C=#count{S:purchase(S),defective(S)}.
```

In the grounded program all the possible outcomes of the counting aggregate depending on the choice of purchased TVs are generated:

```
n_def(0)←0=#count{tv1:purchase(tv1);tv2:purchase(tv2);
                  tv3:purchase(tv3)}.
n_def(1)←1=#count{tv1:purchase(tv1);tv2:purchase(tv2);
                  tv3:purchase(tv3)}.
n_def(2)←2=#count{tv1:purchase(tv1);tv2:purchase(tv2);
                  tv3:purchase(tv3)}.
n_def(3)←3=#count{tv1:purchase(tv1);tv2:purchase(tv2);
                  tv3:purchase(tv3)}.
```

Therefore, counting aggregates over a variable configuration still lead to grounding out explicitly all possible outcomes $\{0,1,2,3\}$. Moreover, each combination requires considering all possible choices that lead to the specific outcome. In fact, each grounded `count` aggregate in the body is conditioned on which specific TV is chosen for purchase, i.e. which predicates are true in $\{tv1 : purchase(tv1); tv2 : purchase(tv2); tv3 : purchase(tv3)\}$.

In Prolog, where such a counting construct is not available, the user has to explicitly write these rules for the different combinations of selections and counts. As for CNF, this format expresses a *propositional* theory as the logical conjunction $C_1 \wedge \cdots \wedge C_n$ of clauses, where a clause C_i is a disjunction $v_1 \vee \cdots \vee v_m$ of a number of variables v_i or their negations $\neg v_i$. Therefore, if a first-order theory contains an exponential number of rules to encode aggregates, then the CNF would be simply a rewriting in a different (propositional) format of such logic rules.

3.2 Modelling Combinatorics Math Problems

The challenge of designing a framework for combinatorial counting based on lifted reasoning lies in recognizing in the language when and where objects can be grouped together and constraints induce symmetric (sub)problems. This process starts from the language in which the problem is expressed, therefore a modelling language with direct support for the fundamental primitives is essential for effective reasoning. In fact, as Examples 3.1 and 3.3 show, the level of abstraction of the language influences directly the opportunities for lifted reasoning on a model. Moreover, a higher level of abstraction simplifies the more general task of solving math word problems by closing

the distance between the natural and formal languages. In this section we define a modelling language for a class of combinatorics math problems: CoLa (**Co**mbinatorics math problems **La**nguage). With CoLa we define the scope of problems for which a fully lifted reasoning approach is implemented in our solver for combinatorics math problems: CoSo (**Co**mbinatorics math problems **So**lver, Chapter 4). The goal of CoLa is thus to match the scope of the lifted solver presented in Chapter 4, rather than being a general purpose language for combinatorics math problems. CoLa and CoSo directly support a class of problems general enough to cover a wide range of real-world problems (cf. Section 3.3). At the same time this class contains hard instances for both lifted reasoning and enumeration-based counting techniques, and it is thus suitable to analyze and compare the two approaches (cf. Section 4.9).

We identify the class of problems under investigation by defining a combinatorics math problem. Broadly speaking, a combinatorics math problem involves counting how many configurations of a finite set of objects satisfy a given set of constraints. The subsequent subsections specify the three characterizing components:

1. the multiset describing the atomic objects involved (Section 3.2.1);
2. the configurations corresponding to the special cases where counting rules are applicable (Section 3.2.2);
3. the constraint language for the problems at hand (Section 3.2.3).

Along with the characterizations we describe the corresponding statements in CoLa, and the corresponding semantics by mapping a sequence of statements `S = s1; s2; ...; sN;` to a counting Constraint Satisfaction Problem (summarized at the end of this section in Table 3.4).

3.2.1 Multisets

The first characterizing component is the finite set of atomic objects we reason about. As we argued in Section 3.1, expressing them as a simple set of constants here is not sufficient, because in combinatorics math problems it is relevant to refer to indistinguishable objects and their properties. In **P1**, for instance, the shapes are divided into groups with the same shape or colour, and the green triangles are indistinguishable because they have the same properties “green” and “triangle”. On the other hand, in **P3** all TVs are distinguishable, i.e. buying one rather than another is a different purchase, and there is a subset of 3 TVs with the property “defective”. To effectively model the setting of a combinatorics math problem we thus need to express 1) indistinguishability of objects and 2) subsets of objects.

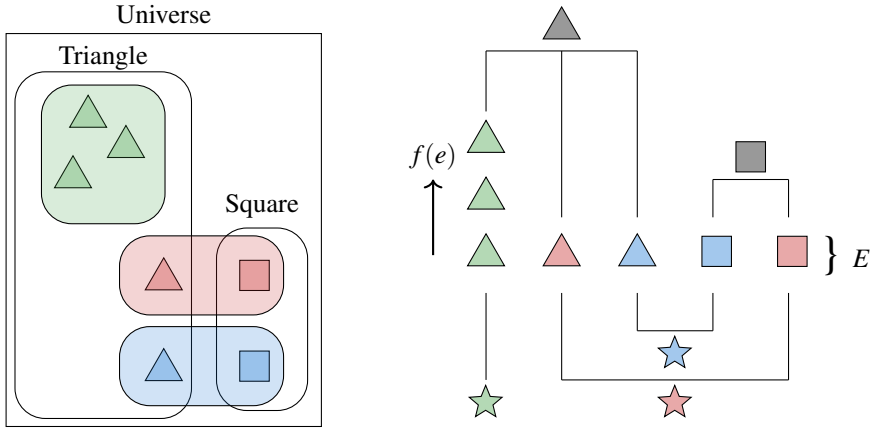


Figure 3.2: On the left, the objects in Example 1 grouped by their properties. On the right, the corresponding multiset. Stars represent the colour property, black items represent the shape property.

To address (1), we base our reasoning framework on *multisets*. A multiset is a pair (E, f) where E is a set and $f : E \to \mathbb{N}$ is a function counting the (indistinguishable) copies of each element of E in the multiset. A set is a special case of a multiset where $\forall e \in E : f(e) = 1$. We call a real-world item *object* and the corresponding mathematical representation *entity*, an element of E. Therefore, an entity $e \in E$ corresponds to $f(e)$ indistinguishable objects.

To address (2) we define a *property* as a subset of the entities: an entity e has a property $P \subseteq E$ if $e \in P$ and does not if $e \notin P$. We call the *universe* the multiset of objects along with the partitioning identified by the properties (Figure 3.2). Properties are subsets of entities because two objects cannot be indistinguishable if they do not have the same properties. If an entity e has a property P which is exclusive, i.e. $P = \{e\}$, then e corresponds to an object with no indistinguishable copies. An example of such property is the name of a person or a serial number of a TV. It follows that two objects are indistinguishable iff. they have the same properties.

Example 3.5. Consider **P1**: the multiset of entities is (E, f) where $E = \{sq_r, sq_b, tr_r, tr_b, tr_g\}$ and $f(sq_r) = 1, f(sq_b) = 1, f(tr_r) = 1, f(tr_b) = 1, f(tr_g) = 3$. The properties are: $blue = \{sq_b, tr_b\}$, $red = \{sq_r, tr_r\}$, $green = \{tr_g\}$, $square = \{sq_r, sq_b\}$ and $triangle = \{tr_r, tr_b, tr_g\}$. Figure 3.2 shows the objects as a Venn-diagram on the left and the corresponding multiset on the right, along with the corresponding properties.

A framework based on multisets is fundamentally different from traditional declarative

settings (cf. Section 3.1) where the problem's atomic objects are a *set* of constants (e.g. propositional and first-order logic) or *sets* of admissible values (e.g. the domains in CSPs). Generalizing sets to multisets is relevant because indistinguishability influences how solutions are counted, i.e. we do not count a solution as new (different) if it is obtained by replacing in a valid configuration objects with their indistinguishable copies. For example replacing any green triangle in a sequence $\langle ▲, ▲ \rangle$ with the third green triangle does not give a different solution.

The separation between the abstraction of entities and real-world objects introduces the issue of whether the user (1) should label the objects according to their indistinguishability or (2) should let the system infer the labels according to the properties specified. In the first case the user should already identify the partition into indistinguishable sets of objects induced by the properties, for instance in **P1** by recognizing that the intersection of the properties *square*, *blue* and *red* makes all squares distinguishable by means of their colour. The multiset can be then represented as an explicit enumeration of objects where distinguishable objects have different labels. This inference on small human-solvable examples is simple, and therefore it might appear more intuitive. However, this task becomes harder when enumerating an increasing number of objects and groups. Here, the second option is preferable, where the groups (properties) of objects are declared along with their cardinality, and the corresponding implicit multiset is inferred from their intersections.

We clarify the difference by describing their implementation in CoLa, where both options are supported. Regardless of whether the universe is specified explicitly or implicitly, the semantics of the CoLa statements for declaring multiset of entities and the corresponding properties is the specification of the domains D of a #CSP $MC(V,D,C)$, as we further specify in Section 3.2.2.

Enumeration In the first case, where indistinguishability is already sorted out, the universe (E, f) is declared by enumerating the objects, and indistinguishable objects receive the same label, i.e. for each entity $e \in E$ there are $f(e)$ repetitions of the label e. Properties $P \subseteq E$ are declared by enumerating the labels that correspond to the entities with the property. The keywords `universe` and `property` distinguish the two types of declaration, for example:

Example 3.6. The universe of **P1**, following Example 3.5, is declared with the CoLa statements:

```
universe shapes= {sq_r,sq_b,tr_r,tr_b,tr_g,tr_g,tr_g};
property red= {tr_r,sq_r};
property blue= {tr_b,sq_b};
property green= {tr_g};
```

```
property triangle= {tr_r,tr_b,tr_g};
property square= {sq_b,sq_r};
```

Example 3.7. We express the multiset of letters (the universe) in **P5** as:
`universe letters = {a,a,a,n,n,b}`
Problems with distinguishable objects reduce to a set declaration:

P6. In how many different ways can Ann, Bob, Carl and Dan queue at the bus stop?
`universe people = {ann, bob, carl, dan};`

Properties can also be defined by means of *set formulas*: a set formula is either a property or an expression of the form `(A & B)`, denoting the intersection $A \cap B$ where A and B are a set-formulas, or `(A + B)`, denoting the union $A \cup B$, and `¬(A)`, denoting the complement $\overline{A}$ w.r.t. the universe.

Cardinality In the second case only properties and their cardinality are declared: the universe is the union of the properties and labelling objects is a reasoning step for the solver. Instead of explicitly specifying all objects with a property P, a size constraint (cf. Section 3.2.3) of the form `#P = n`, where `n` is a natural number, specifies how many objects have the property. In this case we assume that two properties are disjoint unless a size greater than 0 is specified for their intersection. The keyword `labelled` can be prepended to the property declaration as a shortcut for expressing that all objects having the property can be distinguished from one another.

Example 3.8. The universe of **P1**, following Example 3.5, is declared with the CoLa statements:

```
property red;          property blue;
property green;        property square;
property triangle;     #green=3;
#red=2;                #blue=2;
#square=2;             #triangle=7;
#square&red=1;         #square&blue=1;
#triangle&green=3;     #triangle&red=1;
#triangle&blue=1;
```

Example 3.9. In **P6** we can express the set of people as: `labelled property people; #people=4;` The encoding for **P5** by means of cardinalities is:

```
property a; property n; property b;
#a=3; #n=2; #b=1;
```

3.2.2 Configurations

The second characterizing component of combinatorics math problems is the *configuration* in which objects are arranged. In Table 3.1 and Section 3.1 we presented some examples where a valid configuration is a selection of objects or a distribution into groups. For instance a selection (without replacement) is the row of four shapes in **P1** or the purchased TVs in **P3**. The former is ordered and the latter not. An example of distribution is the partition of the shapes into groups in **P1** or workers into tasks in **P4**.

In order to answer the question "what is a configuration?" we have to characterize these notions of selection, order, repetition and distribution. To do so, we follow the *Twelvefold-way* (Stanley 2012), which is a mathematical model based on indistinguishability for a wide range of combinatorics math problems. The configurations in the Twelvefold-way are the most common because they correspond to the special cases where the number of possible configurations can be computed from the size of the universe. The combination of two dimensions defines a configuration: constraints and the (in)distinguishability of the objects. Despite using the same characterization of configurations, our framework is more general than the Twelvefold-way because it expands both dimensions.

The *Twelvefold-way* defines a combinatorics problem as the task of counting the number of functions $g : X \to Y$ between two finite sets X and Y. As illustrated in Table 3.2, two dimensions determine different types of problems and configurations. The first is whether the function g is arbitrary (g_{any}), or constrained to be either injective (g_{inj}) or surjective (g_{sur}). Our framework expands this dimension with additional constraints over the function (Section 3.2.3). The second dimension is concerned with the indistinguishability of objects, where we have all objects distinguishable ($\neq$, a

$g : X \to Y$	g_{any}	g_{inj}	g_{sur}
$\neq$	x-sequences of Y: y^x	x-permutations of Y: $y^{\underline{x}}$	y-compositions of X: $\left\{ {x \atop y} \right\} y!$
$=^X$	x-multisubsets of Y: $\binom{y+x-1}{x}$	x-subsets of Y: $\binom{y}{x}$	y-int. compositions of X: $\binom{x-1}{x-y}$
$=_Y$	k-partitions of X: $\sum_{k=0}^{y} \left\{ {x \atop k} \right\}$	partitions of X: $[x \leq y]$	y-partitions of X: $\left\{ {x \atop y} \right\}$
$=_Y^X$	k-integer partitions of X: $\sum_{k=0}^{y} p_k(x)$	integer partitions: $[x \leq y]$	y-integer partitions of X: $p_y(x)$

Table 3.2 : Twelvefold Way: basic combinatorial configurations with the corresponding counting rules. $y = |Y|, x = |X|$, $y^{\underline{x}}$ is the falling factorial, $\binom{\cdot}{\cdot}$ is the binomial coefficient, $\left\{ {\cdot \atop \cdot} \right\}$ is the Stirling number of the second kind, $p_y(x)$ is the integer partition of x into y parts and $[x \leq y]$ is 1 if $x \leq y$, 0 otherwise.

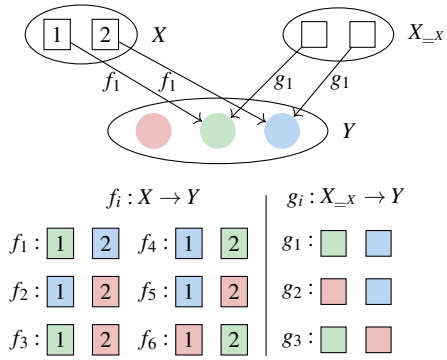


Figure 3.3: Level 1 configurations map slots to objects Y.

set), objects in X all indistinguishable and those in Y all distinguishable ($=^X$), objects in X distinguishable and those in Y indistinguishable ($=_Y$), or objects in both X and Y indistinguishable ($=_Y^X$). Our framework expands this direction by considering any multiset of objects, while the Twelvefold-way restricts either to a set, or a multiset (E, f) where all objects are identical, i.e. $|E| = 1$.

Intuitively, in the Twelvefold-way an ordered (resp. unordered) configuration is a set of labelled (resp. unlabelled) slots represented by a set of distinguishable (resp. indistinguishable) objects. There are two types of configurations. Configurations of level 1 map slots to objects, that is, each slot corresponds to exactly one object. Configurations of level 2 map objects to slots, that is, multiple objects can be mapped to the same slot, which thus represents a subset of the objects. Figures 3.3 and 3.4 represents the difference between the two types.

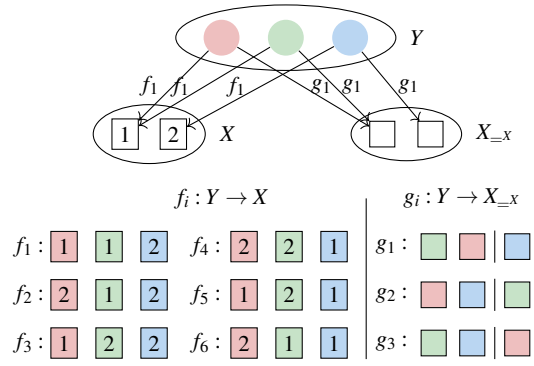


Figure 3.4: Level 2 configurations map objects Y to slots

Similarly to configurations, we also distinguish two levels of properties. Level 1 properties group together objects. Level 2 properties group subsets of objects sharing a set level feature such as the size of the set or the number of objects in the set having some level 1 property.

Example 3.10. In problem **P2** having all green objects in one group is a level 2 property because it defines that there is one part containing all three objects with the level 1 property "green". In problem **P4** we distinguish between three level 2 properties: the parts of size 7, those of size 5, and those of size 2. Suppose we could distinguish workers with the level 1 property "intern". Then, an example of level 2 property is "subsets of workers where there are 3 interns". This level 2 property is the set of all possible subsets of workers where the number of objects presenting the level 1 property "intern" is 3.

Following the Twelvefold-way, we define a configuration in our framework as a mapping between multisets. The properties of the mapping identify the characterizing components of a configuration:

Definition 9. *Given a multiset of entities E, a configuration is a pair (C, g), where $C = (S, f)$ is a multiset and g is a function between E and C (see **Level**). The fundamental properties of the configuration are:*

- ***Order.*** *A configuration is either ordered ($\forall s \in S : f(s) = 1$) or unordered ($|S| = 1$). In CoLa we denote ordered configurations with* `[·]` *and unordered configurations with* `{·}`
- ***Level.*** *Levels define how entities are grouped together: $g : C \to E$ (resp. $g : E \to C$) is a level 1 (level 2) configuration. Figures 3.3 and 3.4 visualize how inverting the role of the two sets changes a configuration from one level into another. Moreover, sets in level 2, which we call* parts *to distinguish them from the configuration name, can be empty (g_{any}) or not (g_{sur}). In CoLa we use a second level of brackets to denote level 2 configurations (cf. Table 3.3).*
- ***Repetition.*** *In Level 1 configurations entities can be repeated (g_{any}) or not (g_{inj}). In CoLa we denote configurations with the keyword* `repeated` *in front of the entity set (cf. Table 3.3).*

Table 3.3 summarizes the configurations we will consider throughout the chapter. Note that in level 2 repeating elements from one selection for a subset to another reduces to many independent subset problems, hence we do not consider it explicitly.

In CoLa we label a valid configuration with a statement of the form `label in X`, where `X` is one of the configurations in Table 3.3

	level 1		level 2
	repeated	non-repeated	non-repeated
ordered	sequence: `[repeated φ]`	permutation: `[φ]`	composition: `[{φ}]`
unordered	multisubset: `{repeated φ}`	subset: `{φ}`	partition: `{{φ}}`

Table 3.3 : Configuration types and the corresponding CoLa syntax for a set-formula φ

Example 3.11. In **P3** the purchase is unordered, therefore the configuration of purchased TVs is a *subset* (level 1), because we cannot purchase multiple times the same TV (no repetition). In CoLa we encode it as: `purchase in {tvs};` for the set `labelled property tvs; #tvs=12;`
In **P2** the indistinguishable green triangles are distributed in groups (level 2), hence do not have an order or label (unordered). Therefore, the configuration is a partition and the declaration in CoLa is: `groups in {{shapes}};` with the declaration of the universe of Example 3.6.
In **P6** four (distinguishable) objects are arranged in an ordered configuration where objects are not further grouped (level 1) and should not be repeated: a *permutation*. In CoLa: `queue in [people]`.

As for the semantics as a #CSP, the universe or set formula φ in the declaration of the configuration maps to the domains D of the variables. In fact, we use variables V to represent an object in the configuration on level 1, or a part in level 2. Therefore, the domain of a variable $v_i \in V$ is, respectively, the multiset φ or its powerset, i.e. $D_i = \varphi$ (level 1) or $D_i = \mathscr{P}(\varphi)$ (level 2). To simplify the encoding as a CSP, we introduce an additional parameter, cf, that encodes the three fundamental properties of a configuration. This parameter is syntactic and has the added benefit of bringing the description closer to the internal representation of the problem in the dedicated solver (Section 4). Therefore, from now on, the #CSP corresponding to a combinatorics math problem expressed in CoLa is $MC(V,D,C,cf)$, where cf is a symbol in $\{[||\cdot],[|\cdot],\{||\cdot\},\{|\cdot\},\{\{\cdot\}\},[\{\cdot\}]\}$ corresponding to the given configuration (the symbols $||$ and $|$ denote configurations respectively with and without repetition). The additional parameter explicitly and compactly denotes whether we are interested in unordered configurations, whereas the traditional definition of CSP defines a *tuple* (i.e., ordered set) of variables.[1] Similarly, including this parameter in the configuration constraints, allows us to separate the additional constraints C (Section 3.2.3) from those intrinsic in the configuration.

[1] We could model this aspect by explicitly defining a variant of a CSP over a set of variables rather than a tuple, or add dedicated constraints to the (low-level) constraint language to exclude the indistinguishable alternative orderings.

As we argued in Section 3.1, most declarative frameworks cannot naturally represent the configurations described in problems where unlabelled objects are involved, which are needed either for the representation of indistinguishable objects or the properties (Order) of a configuration. The Twelvefold-way is limited as well, in the encoding of multisets and constraints.

Example 3.12. **P6** is modelled by the Twelvefold-way because people are all distinguishable. On the contrary, **P5** is modelled by our definition of universe but is not by the Twelvefold-way. **P3** counts only functions that map the set of purchased TVs to at least 2 defective ones, restricting further the family of injective functions, therefore **P3** is not expressible by the Twelvefold-way.

In the Twelvefold-way, each combination of function type and distinguishability of the sets corresponds to a counting rule providing the number of mappings between the two sets.

Definition 10. *A* counting rule *is a closed-form formula that given two multi-sets representing the set of entities and a configuration, and a function g between the two ($g_{any}, g_{inj}, g_{sur}$), returns the number of functions g between the two multisets.*

Pak (2018) provides a detailed dissertation of mathematical formulas for enumerative combinatorics and their characterization in terms of computational complexity.

The problems in the Twelvefold-Way therefore do not require any particular reasoning technique since it suffices identifying the corresponding counting rule. As in **P3**, the complexity of such basic problems is increased when introducing additional constraints. As a result, the counting rules are no longer applicable, therefore lifted reasoning techniques are needed to decompose complex problems into components where counting rules are again applicable. We now define the constraints that can be added to a configuration in our framework. They allow us to analyze more sophisticated cases from the perspective of lifted reasoning, and significantly expand the number of real-world problems falling into our framework (cf. Section 3.3).

3.2.3 Constraints

The constraints are specifications of which configurations are considered valid w.r.t. the specified objects and characteristics of the configuration itself. We thus define a constraint language for entities and configurations by introducing three kinds of constraints: *size* constraints, *counting* constraints and *positional* constraints. From now on, we will use the following notation: $*$ denotes a numerical relation, i.e. $* \in \{\leq, <, >, \geq, \neq, =\}$, n denotes a natural number, i.e. $n \in \mathbb{N}$. We use this language as the constraint language for $MC(V, D, C, cf)$.

Size constraints Size constraints define either the size of a configuration or a part:

Definition 11. *Given a configuration cf, a size constraint is:*

a) a tuple $(cf, *, n)$*, which is satisfied when the size of the configuration is k and* $k * n$*. The corresponding syntax in CoLa is:* `#cf * n`.

b) a tuple $((cf, i), *, n)$ *where cf is of level 2. The constraint is satisfied when the* i^{th} *part of cf contains k entities and* $k * n$*. The corresponding syntax in CoLa is:* `#cf[i] * n`.

Let s be the size of the universe. Size constraints of type (a) define the set of valid sizes for a configuration, removing from the interval $\{1, \ldots, s\}$ the values i such that $i * n$ is false. Size constraints `#cf[i] * n`, type (b), apply to $v_i \in V$ and the corresponding declaration is simply added to C.

Example 3.13. In **P4**, let *groups* be the name of the configuration, then the number of groups of workers is expressed with: $(groups, =, 3)$. In CoLa:

```
labelled property workers;
#workers = 14;
groups in {{workers}};
#groups = 3;
```

Counting constraints Counting constraints count either entities (level 1) or parts (level 2):

Definition 12. *Given a configuration of level i, a* counting *constraint is a pair* $(\varphi, *, n)$*, where* φ *is a property of level i. If* $i = 1$ *(resp.* $i = 2$*) the constraint is satisfied when in the configuration there are k objects (resp. parts) belonging to* φ *and* $k * n$.

In CoLa counting constraints are statements of the form $\#\varphi * n$, which explicitly distinguish between the two levels for φ, which is:

1. A level 1 property counting the objects in a set-formula φ': `cf & `φ'.
2. A level 1 property counting the objects from a set-formula φ' of the i^{th} part of a composition `cf`: `cf[i] & `φ'.
3. A level 2 property, where `part` is a reserved word denoting any part of the level 2 configuration, counting the number of parts with a given:
   3.1. size, when φ is the size constraint `(#part * m)`.

3.2. number of entities from a set-formula φ', when φ is the counting constraint `(#part & φ′ * m)`.

When φ' is the universe, the expression `part & φ′` can be abbreviated to `part` and `cf[i] & φ′` to `cf[i]`.

Example 3.14. In a configuration "permutation" a counting constraint $(green, =, 3)$ states that the number of objects with the property "green") is equal to 3. In a configuration "partition" $((green, =, 3), =, 1)$ states that there is exactly one part containing three green objects (Problem **P1**).

Example 3.15. In **P4** we express the constraints over the sizes of the groups as: $((worker, =, 7), =, 1)$, $((worker, =, 5), =, 1)$, $(worker, =, 2), =, 1)$. The complete model in CoLa for **P4** is thus obtained by adding the size constraints to Example 3.13:

```
labelled property workers;
#workers = 14;
groups in {{workers}};
#groups = 3;
#(#part=7) = 1; #(#part=5) = 1; #(#part=2) = 1;
```

Example 3.16. **P3** is modelled as the universe $U = \{tvs, defective\}$, $tvs = \{tv_1, tv_2, \ldots, tv_{12}\}$, $f(tv) = 1$ for all $tv \in U$, $defective = \{tv_1, tv_2, tv_3\}$. They are arranged in a configuration cf of type subset with the size constraint $(cf, =, 5)$ and counting constraint $(defective, \geq, 2)$. In CoLa:

```
labelled property tvs;
property defective;
#tvs = 12;
#(tvs & defective) = 3;
purchase in {tvs};
#purchase = 5;
#(purchase & defective) >= 2;
```

Positional constraints Positional constraints apply to ordered configurations, where it is meaningful to refer to a particular position in the order. This kind of constraint is interesting from a reasoning perspective because, contrary to counting constraints, it requires reasoning about a specific component of the configuration, instead of a global (aggregate) feature.

Definition 13. *Given a configuration of size k and level i, a* positional *constraint is a pair* (φ, n)*, where* φ *is a property of level i, and* $n \leq k$*. The constraint is satisfied when the object or part in position n belongs to* φ*.*

Positional constraints (φ, n) for an ordered configuration `cf` of level 1 are expressions of the form `cf[i] in` φ where i is a natural number and φ is a set-formula. Similarly, for an ordered configuration `cf` of level 2 positional constraints are expressions of the form `cf[i] =` φ where i is a natural number and φ is a set-formula. Positional constraints apply to the corresponding *i-th* variable $v_i \in V$, and similarly to counting constraints are added to C.

Example 3.17. In a configuration "permutation" a positional constraint $(green, 2)$ states that the second entity in the order is green. For instance in problem **P1** we express this constraint in CoLa as: `row[2] = green;` In a configuration "composition" $((green, =, 3), 1)$ states that the first part in the order contains exactly three green objects.

Example 3.18. P6 with the additional positional constraint:

```
universe people = {ann, bob, carl, dan};
queue in [people];
#queue=4;
queue[1] in {ann, dan};
```

Syntax	Semantics	Description
`property P; #P = n`	$P = \{e_1, \ldots, e_n\}$	Add property P with n entities to U
`property P = {e1,...,en};`	$P = \{e_1, \ldots, e_n\}$	Add properties P plus $P_1, \ldots P_n$ to U
`labelled property P; #P = n`	$P_1 = \{e_1\}, \ldots, P_n = \{e_n\}$	
`cf in {{U}}` or `cf in [{U}]`	$cf = \{\{\cdot\}\}/[\{\cdot\}]$, $D_i = \mathscr{P}(U)$	Define level 2 configuration, its label and set
`cf in [repeated U]` or `cf in [U]`	$cf = [\|\|\cdot]/[\|\cdot]$, $D_i = U$	Define ordered level 1 configuration, label, set
`cf in {repeated U}` or `cf in {U}`	$cf = \{\|\|\cdot\}/\{\|\cdot\}$, $D_i = U$	Define unordered level 1 conf., label, set
`#cf * n`	$V = \langle V_1, \ldots V_i \rangle$, $i * n$	Define size(s) of configuration(s) `cf`
`#cf[i] * n`	$C = C \cup \{((V,i), *, n)\}$	Define size of the *i-th* part in composition `cf`
`cf[i] in` φ	$C = C \cup \{(\varphi, i)\}$	Define level 1 positional constraint on i
`cf[i] =` φ	$C = C \cup \{(\varphi, i)\}$	Define level 2 positional constraint on i
`#`φ `* n`	$C = C \cup \{(\varphi, *, n)\}$	Define level 1 or 2 counting constraint

Table 3.4 : Summary of CoLa syntax and corresponding semantics. U denotes the universe corresponding to the given objects and properties.

In Table 3.4 we summarize all CoLa statements and their semantics presented in this section.

3.3 Modelling: Language Analysis

To assess the expressivity of CoLa, we encoded a dataset of real-world math problems collected by (Dries et al. 2017). The dataset contains 210 combinatorics math problems, of which 185 (88%) can be expressed in CoLa, including the 106 (51%) encodable in the Twelvefold-way. The configurations are distributed as follows: 101 (54%) of the problems regard sequences and permutations, 72 sets and multisubsets (39%), 8 (5%) partitions and 4 (2%) compositions. Cola can encode 79 more problems (37%) by extending the Twelvefold-way in the two dimensions with multisets of objects and additional positional and counting constraints. The former allows us to encode 22 (12%) problems, the latter are used in 57 (31%) of the encoded problems, divided between 15 (8%) problems with positional constraints and 42 (23%) problems with counting constraints.

We now briefly analyze the most common patterns of the 25 problems that cannot be encoded in CoLa (12% of the dataset), which thus provide interesting indications about the limitations of the language that can be addressed with future work. The majority of such problems (10) regard permutations with relative position constraints: CoLa can encode absolute positions, i.e. "position i must have an entity/partition with the given properties", but not relative positions between groups, for example "entities of group A are next to each other" or "entities of group A are next to/between entities of group B", for example:

P7. Five married couples bought 10 tickets for a concert. In how many ways can they sit, in the same row, if the five men want to sit together?

P8. Nine chairs in a row are to be occupied by six students and Professors Alpha, Beta and Gamma. These three professors arrive before the six students and decide to choose their chairs so that each professor will be between two students. In how many ways can Professors Alpha, Beta and Gamma choose their chairs?

These types of problems require a more expressive constraint language for positions, and corresponding lifted counting techniques.

Other problems (4) use a configuration not included in the Twelvefold-way, that is, a circular permutation, for example:

P9. In how many ways can three men and three women sit at a round table if each woman sits in between two men?

The circularity of the permutation has an impact on the satisfiability of the relative positional constraints.

Finally, we report 3 problems where entities are numbers and constraints are expressed over arithmetic operations over them, for example

P10. Three distinguishable dice numbered 1, 2, 3, 4, 5 and 6 are thrown. In how many ways can they land and give a sum of 9?

In this case, on a language level, we would need data types (integer) and a corresponding set of operations in the constraint language (e.g. sum, or in general, arithmetic operations). On the reasoning level, the associativity, and commutativity of the sum make the variable numbers exchangeable, hence a lifted solver would be able to account for such symmetry.

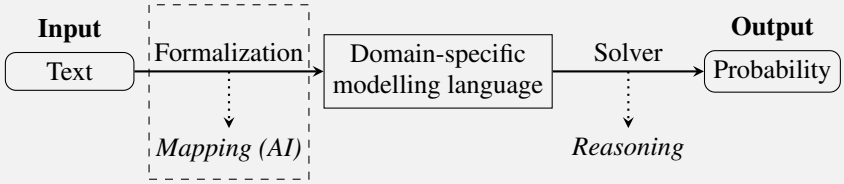
This analysis therefore shows how combinatorics problems present a wide variety of symmetries tightly related to the modelling language, where a lifted approach is possible. While CoLa and CoSo can deal with the most common cases, they fall short on tackling some less standard combinations of constraints and objects. Including more configurations and object types in the language, paired with the respective lifted reasoning techniques, is thus an interesting direction for future work. This means expanding CoLa with new language constructs in the two dimensions of the Twelvefold-way.

The first dimension concerns how the configuration is represented. Here, it is possible to consider more types of configurations such as circular permutations. The first direction is to add ad-hoc labels for the new configurations. On the reasoning side, this would require dedicated counting rules and redefining the propagation of all constraints, in particular the splitting operation, on the new configuration. However, this direction is limited in terms of modularity and expansibility of the language, because each new type of configuration requires introducing new primitives in the language. The second approach would involve expanding CoLa's primitives to allow specifying arbitrarily complex configurations in a modular way. While in the Twelvefold-way the configuration is a simple set, more sophisticated configurations define (binary) relations over the set. For example a circular permutation is no longer an order derived from labels but defines a (circular) successor relation. Relations between elements of the same set are called homogeneous (Schmidt and Ströhlein 1993). Different combinations of properties, e.g. reflexive, transitive, symmetric,..., define different types of homogeneous relations which include (un)directed graphs, orders and equivalences. On the one hand, an interesting direction is to study how exchangeability changes with respect to these properties and the resulting relation. On the other hand, introducing an homogeneous relation allows us, in principle, to refer to the relation in the constraints. For example if the configuration is a graph, we could ask that "each neighbour of a professor will be a student" rather than "each professor will be between

two students". The question is then how to count the exchangeable choices while propagating the constraints regarding the relation.

The second dimension of the Twelvefold-way involves the constraints placed on the function mapping objects to the configuration. Here, we could include new types of constraints such as the relative positions between properties. While adding new constraints is not hard, it is difficult to handle them in a lifted manner because each new constraint requires introducing new propagation and reasoning rules, both individually and in combination with other constraints. In fact, with the operators presented in Chapter 4, we will show that different combinations of constraints must be taken into account while reasoning. In general, existing constraint modelling languages provide a reference point as to which type of constraints are most useful and interesting language-wise.

Intermezzo 1: Mapping Probability Word Problems to Executable Representations



Solving math word problems automatically is an active area of research in natural language processing. It poses interesting challenges in extracting relevant entities and quantities from a concise textual narrative, and reasoning about the relationships between them to answer the question posed in the text. While several approaches for different areas of mathematics have been proposed in the past, the majority have focused on arithmetic and algebraic problems where the mathematical representation is specified as a simple equation system that can be used with a standard solver (Amini et al. 2019; Hosseini et al. 2014; Huang et al. 2016; Koncel-Kedziorski et al. 2015; Kushman et al. 2014; Ling et al. 2017; Miao et al. 2020; Roy and Roth 2018; Upadhyay and Chang 2015; Wang et al. 2017). On the other hand, probability problems that may be found in introductory textbooks for discrete mathematics, such as Example 1.1, have not received much attention.

The overall task is to obtain the answer from the text, with the formal representation as an intermediate step. Generally, the representation contains one or more multisets of objects with certain properties, one or more actions that create new multisets, and a question that imposes a constraint on the result of the actions and asks for the probability of that constraint holding.

Example 3.19. Given a set of statements for the domain of probability

problems (Dries et al. 2017), the problem from Example 1.1 is mapped to the domain-specific language as follows:

```
group(cycle).
size(cycle, 80).
property(property, [amber, green, red]).
given(exactly(10, cycle, amber)).
given(exactly(30, cycle, red)).
given(exactly(40, cycle, green)).
take(cycle, light, 1).
probability(none(light, red)).
```

Automatically constructing this representation from the natural language text is challenging for several reasons. The text may not explicitly state whether sampling (`take`) is with or without replacement, and this information may need to be inferred. Questions with co-referring expressions are common, and may contain noun ellipsis, e.g. “One is selected at random", in which the antecedent is mentioned in a previous sentence. Furthermore, problems may require access to world knowledge, e.g. distinguishing between male and female names.
Compared to general math word problems, probability word problems have a number of unique characteristics, many of which can be also found in combinatorics problems. They typically mention several multisets (e.g. “a deck”) with the objects contained (e.g. ”cards”), and the properties (e.g. “black”, “queen”) and sizes that the model needs to relate back to the corresponding objects. The problems rely heavily on Boolean operators to combine the objects’ properties, as well as specify constraints holding on these properties (e.g., by using quantifiers, such as “at least”). The Boolean operators and quantification feature much less prominently, or not at all, in other types of math word problems. These are particularly challenging in modelling, leaving ample opportunities for future work. A distinctive characteristic of probability problems is the need to construct intermediate sets with a sampling operation; the resulting sets need to be kept by the solving system and used when answering the final probability question. While predicates in several existing datasets of word problems allow a straightforward translation into arithmetic operators, probability word problems usually do not. This makes the case for modelling the intermediate meaning representation stronger.
The second step uses a dedicated solver implemented in the probabilistic programming language ProbLog (Fierens et al. 2011) to compute the final, numerical answer from the intermediate representation. In probability problems, an answer can be wrong even if its difference to the correct answer is numerically small, but such differences stand out more clearly on the problem specification level, making the latter a more appropriate modelling target.

The main focus is on predicting the intermediate representation from the problem text and a comparison with models that directly map the text to the answer. To map the word problem texts to their executable representation, the focus is on neural sequence-to-sequence (seq2seq) models. The most important findings are:

- using a pretrained language model to encode the problem text into contextualised representations leads to superior performance on the largest available probability-problem dataset NLP4PLP (Dries et al. 2017), when compared to (i) the classical seq2seq model whose encoder uses non-contextualised word embeddings; (ii) end-to-end sequential approaches foregoing the solver; (iii) the rule-based system of (Dries et al. 2017) and the simpler baselines,
- starting with the best seq2seq model that uses contextualised representations, in-domain transfer learning on the MATHQA dataset (Amini et al. 2019) containing a broader spectrum of math word problems leads to additional substantial improvements in the number of word problems answered correctly,
- although end-to-end problem answering can approximate the true probabilities, a two-step approach using the solver is crucial in obtaining solutions that are exactly correct, which are granted by the solver's reasoning algorithms for any correct encoding.

The results then suggest that mapping to a problem specification followed by the application of a dedicated solver is preferable to end-to-end modelling where the answer probabilities are predicted directly from the encoded text. The qualitative analysis of results reveals that the extraction of relevant entities and quantities from concise textual descriptions, as well as reasoning about the relationships between them are still challenging, and therefore provide possible directions for future work.

The paper on which this intermezzo is based on was previously published as:

S. Suster, P. Fivez, P. Totis, A. Kimmig, J. Davis, L. De Raedt, and W. Daelemans (2021). "Mapping probability word problems to executable representations". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Ed. by M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih. Association for Computational

Linguistics, pp. 3627–3640. DOI: 10.18653/v1/2021.emnlp-main.294. URL: https://doi.org/10.18653/v1/2021.emnlp-main.294

3.4 Conclusion

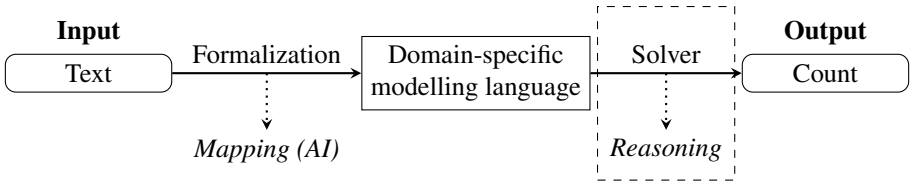
In this chapter we addressed our research question **Q1** by identifying three standard concepts of combinatorics math word problems: 1) multisets 2) configurations and 3) constraints. We proposed a novel declarative language, CoLa, that supports these fundamental primitives, because existing declarative languages present limitations in expressing these key characteristics of combinatorics math word problems. We analyzed the expressivity of CoLa by modelling the combinatorics math word problems of a real-world dataset, successfully encoding a large majority of the problems. Finally, the first intermezzo discusses the problem of automating the mapping of natural language to a domain-specific modelling language for probability problems, which necessitate primitives similar to combinatorics math word problems.

Chapter 4

Combinatorics Math Problems: Reasoning

The paper on which this chapter is based on was previously published as:

P. Totis, J. Davis, L. D. Raedt, and A. Kimmig (2023). "Lifted Reasoning for Combinatorial Counting". In: *J. Artif. Intell. Res.* 76, pp. 1–58. DOI: 10.1613/jair.1.14062. URL: https://doi.org/10.1613/jair.1.14062



Reasoning on combinatorics math problems requires the identification of independent subproblems that allow for exploiting the well-known counting formulas, for example those in the Twelvefold-way, and avoiding brute-force enumeration. For instance, **P5** is solved directly by applying the counting rule for permutations with indistinguishable elements, while **P3** and **P4** combine different subproblems, counting subsets by means of the binomial coefficient, to derive a solution. This is akin to lifted probabilistic

inference (Poole 2003; Van den Broeck 2015; Van den Broeck, Kersting, et al. 2021). The solutions for problems **P3** to **P5** are illustrations of lifted inference where the solution is derived from (a combination of) counting rules considering the size of groups of objects. Moreover, in **P3** we divide the problem into two subproblems: counting the purchases first with 2 defective TVs and then with 3.

Solvers for traditional declarative frameworks such as logic programming and CSPs are based on propositional reasoning, that is, counting at the level of individual objects that are explicitly enumerated or grounded. The number of combinations in combinatorics problems typically grows exponentially in the number of objects involved, therefore counting on a propositional level is prohibitively expensive. On the other hand, existing probabilistic lifted inference techniques are severely limited in the kind of model that they can lift. We shall discuss the difficulty in applying them to combinatorial counting in Section 4.1.

4.1 Related Work

Past work on CSPs offers modelling languages which directly support sets and constraints, but fails to provide efficient solving techniques for the counting problem. On the other hand, the problem of counting the number of valid assignments for a set of (random) variables plays a central role in probabilistic lifted inference, but modelling combinatorics math problems is limited because variables and constants always receive a unique identifier.

4.1.1 Constraint Satisfaction Problems

Constraint modelling languages are usually associated with solvers designed to answer the *satisfiability* question, i.e. "Does a solution exist?", rather than the *counting* question we are concerned with, i.e. "How many solutions exist?". While many solvers can find all solutions, usually this is done by enumeration, which is highly inefficient for combinatorial problems. For example, CONJURE (Frisch, Jefferson, et al. 2005) is a refinement system that translates a model in ESSENCE to ESSENCE', a subset of ESSENCE. ESSENCE' lacks the high-level features of ESSENCE such as the unnamed types, quantifications over variables, or nested types. Similarly to the problems expressed in Zinc or MiniZinc, ESSENCE' can then be compiled for different constraint solvers that can enumerate the satisfiable variable assignments. Tractability of enumeration has been studied in the past, for example by Greco and Scarcello (2010), where the focus is on algorithms that *enumerate* solutions with polynomial delay, that is, given an instance of size n, they require time $O(p(n))$, where $p(\cdot)$ is a polynomial, to discover if there are no solutions, or output all solutions in a way that a new solution

is computed in time $O(p(n))$ from the previous one. On the other hand, the counting question, that is, solving counting constraint satisfaction problems (#CSPs), has been approached in the past from a theoretical perspective regarding tractability (Bulatov 2013; Gottlob et al. 2000) rather than designing a solver that can efficiently answer a counting question, as we do in this chapter.

4.1.2 Lifted Probabilistic Inference

Despite being based on principles similar to the ones used by humans on combinatorics problems, lifted probabilistic frameworks struggle at offering a satisfactory approach to the reasoning task. This is true even for problems where language is not a limitation, i.e. when all objects are uniquely identifiable and the configuration is ordered. Combinatorics math problems are usually solved in two ways: 1) by applying a counting rule based on the size of the sets involved, e.g. for n objects there are $n!$ permutations or $\binom{n}{k}$ subsets of size k, or 2) by decomposing the problem into subproblems where the first method is applicable, and combining them in a count for the main problem.

This is the same principle underlying *lifted* probabilistic inference, which reasons about multiple individuals as a group when there is no additional information that makes relevant distinctions between the individuals. Lifted probabilistic inference techniques apply this principle to two of the main probabilistic inference approaches: variable elimination (Salvo Braz et al. 2005; Taghipour et al. 2012) and knowledge compilation (Van den Broeck, Taghipour, et al. 2011). Lifting can lead to polynomial complexity results for some classes of problems (Kazemi et al. 2016; Van den Broeck, Meert, et al. 2014) despite the fact that the #SAT problem (also known as propositional model counting) reduces to probabilistic inference (Cooper 1990), which is thus #P-complete (Valiant 1979). This motivates the relevance of lifted reasoning for efficient probabilistic inference. Algebraic Model Counting (Kimmig, Van den Broeck, et al. 2017b) (AMC) generalizes probabilistic inference, #SAT, and Weighted Model Counting and shows how knowledge compilation solves any AMC problem. Note that in this context the term "model" is used to denote a satisfying assignment to the variables, rather than the problem specification. We now discuss the characteristics of the main lifted reasoning frameworks, summarized in Table 4.1.

Weighted *First-Order* Model Counting (WFOMC), implemented in Forclift (Van den Broeck, Taghipour, et al. 2011), lifts a Weighted Model Counting task with a set of lifted knowledge compilation formulas. However, such formulas can lift only binary constraints of the form $v \in D$ and $t_1 = t_2$ (and their negation), where v is a variable, D is a set of constants and t_i is either a variable or a constant. It is possible to specify subgroups of constants (subdomains) to identify different properties of the objects

	Sets/ Multisets	Configu- rations	Constraint arity	Modelling	Lifted counting
Forclift	x	x	$= 2$	x	✓
WFOMC FO^2	x	x	≥ 2	x	✓
GC-FOVE	x	x	≥ 2	x	✓
CSPs	sets	✓	≥ 2	✓	x
CoLa+CoSo	✓	✓	≥ 2	✓	✓

Table 4.1 : Probabilistic frameworks (Forclift, GC-FOVE) are limited in modelling by the input language and first-order logic, but implement lifted counting algorithms with limited constraint support. On the other hand CSP frameworks offer direct support for sets and constraints but not for multisets and lifted counting.

represented by the constants, but these remain all distinguishable, thus preventing counts from multiset domains.

WFOMC with counting quantifiers (Kuzelka 2021) considers the two-variable fragment of first-order logic (FO^2) with counting quantifiers (Gradel et al. 1997) and shows that WFOMC can be extended to this class of problems for polynomial-time inference. Counting quantifiers in FO^2 in terms of modelling can play a similar role as the counting constraint shown in Example 3.3. This framework however is based on first-order logic, therefore the aforementioned limitations regarding sets and multisets apply. This means that problems involving unordered configurations (sets, partitions) or configurations with repetition (multisets) are not directly supported.

GC-FOVE (Taghipour et al. 2012) introduces lifted techniques under arbitrary constraints using *constraint trees* to represent the set of admissible assignments to variables. However, this approach can struggle with some kind of constraints, for instance the constraint tree for a non-repeated (all-different) type of configuration (that is, a non-repetition constraint on the variables), e.g. a permutation, would correspond to a tree enumerating all possible permutations of the variables, thus precluding lifted optimizations.

Probabilistic lifted reasoning principles

We briefly introduce the main lifted reasoning principles from probabilistic lifted inference that we apply to constraint satisfaction problems.

Exchangeability. Niepert and Van den Broeck (2014) argue that *exchangeability* of a finite set of random variables is one of the fundamental concepts for tractable probabilistic inference.

Definition 14. *(Niepert and Van den Broeck 2014) A set of random variables* $\boldsymbol{X} = \{X_1, X_2, \ldots X_n\}$ *is* fully exchangeable *if and only if* $Pr(X_1 = x_1, \ldots, X_n = x_n) = Pr(X_1 = x_{\pi(1)}, \ldots, X_n = x_{\pi(n)})$ *for all permutations* π *of* $\{1, \ldots, n\}$.

Exchangeability is exploited to define partitions of exchangeable variables (*variable decompositions*) to decompose the probabilistic inference problem into tractable parts. Exchangeability is closely related to the concept of *symmetries* in CSPs. Rossi et al. (2006, Chapter 10) present an overview of the different terminology associated to symmetries. In particular, it defines *solution* and *problem* symmetries: a solution (problem) symmetry is a permutation of the set of pairs $\langle variable, value \rangle$ which preserves the set of solutions (constraints). Exchangeability is thus a form of solution symmetry: it maps solutions to solutions (and non-solutions to non-solutions). The probabilistic inference literature uses also the terms *indistinguishable* or *interchangeable* (Milch et al. 2008; Taghipour et al. 2012) to refer to exchangeable variables. However, we use these terms to refer to different concepts in combinatorics and CSPs: we already used "indistinguishable" to refer to unlabelled copies of objects following the mathematical terminology presented in Stanley (2012) and formalized in Section 3.2, while in Section 4.2 we use "interchangeable" to refer to a particular form of symmetry in counting CSPs.

We now informally discuss some fundamental principles of lifted reasoning that are shared by both lifted inference frameworks based on variable elimination (Kisynski and Poole 2009; Poole 2003), and those based on knowledge compilation (Van den Broeck, Taghipour, et al. 2011).

Multiplication Multiplication is the operation that exploits the independence of two subproblems (parametric factors in lifted variable elimination or clauses in lifted knowledge compilation) to lift the count of their combinations. For example, we exploited this principle to compute the combinations of the choices for the workers to assign to the mixing cement task and the laying bricks task in **P4**: the problem of choosing 7 workers out of 14 for mixing cement is independent of choosing 5 workers out of 7 for laying bricks, therefore the respective number of choices can be multiplied to derive the solution of the problem.

Splitting and Shattering Splitting is the operation of dividing a problem (parametric factor or clause) into a set of subproblems equivalent to the original one such that they are pairwise independent, and thus the multiplication principle can be applied. Shattering is the repetition of the splitting operation until no longer applicable. For instance in **P1** to count the possible rows described we can split the problem between the subproblem of counting the different choices for placing an object

in second position and the problem of counting the permutations for the other three positions.

Propositionalization Propositionalization, also called grounding, is a principle that acknowledges that lifted reasoning is not always possible and in such cases traditional reasoning on the propositional level is necessary. Propositionalization however can create new opportunities for the application of lifted principles. For instance in **P2** we have to reason explicitly on three different cases, namely the problem where the size of the group with the three green objects is s with $s \in \{3,4,5\}$, and thus the group contains $n = s-3$ non-green objects. Any solution for a fixed s can now be lifted by choosing n of the non-green objects to add to this group, and then distributing the remaining ones over the other two groups. The number of possible choices is given respectively by the binomial coefficient and the Stirling numbers of the second kind. We now present our application of these concepts to the setting of #CSPs.

4.2 Applying Lifted Reasoning Principles to #CSPs

We adapt and develop in the context of #CSPs the definitions and concepts that lifted probabilistic inference proposed to reason about groups of individuals. We begin with defining the difference between exchangeability, interchangeability and indistinguishability:

Definition 15. *A tuple of variables $V = \langle v_1, \ldots, v_n \rangle$ is* exchangeable *($\equiv^V$) w.r.t. a* CSP *$\langle V, D, C \rangle$ if for all satisfying assignments $\langle d_1, \ldots, d_n \rangle$ and all permutations π of $\{1, \ldots, n\}$, $\langle d_{\pi(1)}, \ldots, d_{\pi(n)} \rangle$ is a satisfying assignment as well.*

Exchangeability thus applies to variables, while interchangeability and indistinguishability apply to values. We say that values are *interchangeable* when they correspond to a *value symmetry* (Rossi et al. 2006, Chapter 10): two values a and b are symmetric values if each solution containing the value a can be mapped to a solution containing the value b and vice versa. Finally, *indistinguishable* values are interchangeable values that do not correspond to new solutions, thus are considered identical (unlabelled) copies of each other:

Definition 16. *Given a #CSP $\langle V, D, C \rangle$ and two interchangeable values a and b, a and b are* indistinguishable *if each solution obtained by mapping a to b, and vice versa, is* not *counted as a different solution.*

Example 4.1. Given a #CSP $\langle V, D, C \rangle$ where $V = \langle v_1, v_2 \rangle$ and $D_1 = D_2 = \{d_1, d_2, d_3\}$, C={}, if d_2 and d_3 are indistinguishable then of the 9 possible

assignments only 4 are counted because the following assignments are symmetric w.r.t. the indistinguishability of d_2 and d_3: $(\langle d_1,d_2\rangle,\langle d_1,d_3\rangle)$, $(\langle d_2,d_1\rangle,\langle d_3,d_1\rangle)$, $(\langle d_2,d_3\rangle,\langle d_3,d_2\rangle,\langle d_2,d_2\rangle,\langle d_3,d_3\rangle)$. These 8 assignments count only for 3 distinguishable solutions, along with the fourth $\langle d_1,d_1\rangle$.

In the rest of the chapter we will exploit these kinds of symmetries to count solutions of #CSPs. In particular, when variables are exchangeable we know that all the permutations of a satisfying assignment are solutions as well:

Property 1. *Given a* CSP $\langle V,D,C\rangle$ *such that V is exchangeable, if $\langle d_1,\ldots,d_n\rangle$ is a satisfying assignment then there are $n!$ corresponding exchangeable satisfying assignments.*

Variables v_1, v_2 are exchangeable ($v_1 \equiv v_2$) when their domains are equal and all constraints affect both variables such that exchanging their values does not change any constraint's satisfiability. If v_1 and v_2 are such that $D_1 = D_2$ and all constraints are not affected by different permutations of their assignments, then for each valid assignment where $v_1 = d_1$ and $v_2 = d_2$ the same assignment where $v_1 = d_2$ and $v_2 = d_1$ is also valid. This is because $d_2 \in D_1, d_1 \in D_2$ from $D_1 = D_2$, and under the aforementioned constraints if $\langle d_1,d_2\rangle$ is valid then $\langle d_2,d_1\rangle$ is valid as well. We denote the domain of a set of exchangeable variables V as $dom(V)$.

In CSPs symmetries are redundancies to be avoided. Hence, the model is usually reformulated to exclude symmetric parts of the search space or expanded with symmetry-breaking constraints. In contrast, in #CSPs the symmetric solutions also have to be counted, therefore lifted reasoning explicitly tries to identify partitions of the problem where symmetries can be recognized and exploited for fast counting. We now analyze how this goal can be achieved in the context of #CSPs.

We define the multiplication operation in the context of #CSPs as follows: the multiplication rule counts the combinations of two sets of partial assignments for two subproblems, under the precondition that variables are disjoint.

Definition 17. *Given two* CSP*s $\langle V_1,D_1,C_1\rangle$, $\langle V_2,D_2,C_2\rangle$, such that $V_1 \cap V_2 = \emptyset$, the model count of the union of the two problems is the product of the model counts of the two problems: $MC(V_1 \cup V_2, D_1 \cup D_2, C_1 \cup C_2) = MC(V_1,D_1,C_1) \cdot MC(V_2,D_2,C_2)$ (multiplication rule).*

A *split* of a problem generates two subproblems where the multiplication rule is applicable. We also consider the case where the count from the multiplication rule is adjusted by means of a constant.

Definition 18. *Given $P = \langle V_1 \cup V_2, D, C\rangle, V_1 \cap V_2 = \emptyset$, a* split *for $MC(P)$ is a pair of* CSP*s $(P_1,P_2), P_i = \langle V_i,D_i,C_i\rangle$, such that $MC(P) = c \cdot MC(P_1) \cdot MC(P_2)$, $c \in \mathbb{N}$.*

The role of the constant c is to account for exchangeable or interchangeable choices that may be lifted when defining P_1 and P_2, for example as we do in the lifted operators presented in Section 4.

Example 4.2. A CSP $\langle V,D,C\rangle$, $V = \langle v_1,v_2,v_3\rangle$, $D_1 = D_2 = D_3 = \langle 1,2,3\rangle$, $C = \{v_1 < 3, v_2 \neq v_3\}$, can be split into $(P_1 = \langle\{v_1\},\{D_1\},\{v_1 < 3\}\rangle, P_2 = \langle\{v_2,v_3\},\{D_2,D_3\},\{v_2 \neq v_3\}\rangle$: $MC(V,D,C) = MC(P_1)\cdot MC(P_2) = 2\cdot 6 = 12$. Note that v_2 and v_3 are exchangeable and $c = 1$. As in probabilistic lifted inference, the repeated application of the splitting operation leads to a *shattering*:

Definition 19. *Given a* CSP $P = \langle V,D,C\rangle$, *a* shattering *for* $MC(P)$ *is a set of problems* $\{P_1,\ldots,P_n\}$, $P_i = \langle V_i,D_i,C_i\rangle$, *such that* $MC(V,D,C) = c\cdot\prod_{i=1}^{n} MC(P_i), c \in \mathbb{N}$.

The multiplication rule holds only if the subproblems P_i are independent, i.e. a satisfying assignment for variables V_i on P_i is a partial assignment for the initial problem $\langle V,D,C\rangle$ independent of the others. That is, C does not contain any non-trivial constraint relating the value of a variable v_1 to the value of a variable v_2 with $v_1 \in P_i$, $v_2 \in P_j$ and $i \neq j$. To decompose further the problem we introduce the notions of constraint split and shattering. They allow us to express the count of a problem where a constraint binds two or more variables as the combination of the counts of a set of problems where the constraint is no longer present. The solutions of the original problem are obtained by uniting the solutions of the problems defined by the split or shattering.

Definition 20. *Given two tuples of disjoint variables* V_1,V_2 *and two (partial) assignments* $f_1(V_1) = \langle d_1,\ldots,d_m\rangle, f_2(V_2) = \langle d_{m+1},\ldots,d_n\rangle$, *the* union *of the two assignments* $f_1 + f_2$ *is the tuple:* $\langle d_1,\ldots,d_m,d_{m+1},\ldots,d_n\rangle$. *The union of two sets* M_1,M_2 *of partial assignments is* $M_1 + M_2 = \bigcup_{f_1\in M_1}\bigcup_{f_2\in M_2}\{f_1(V_1) + f_2(V_2)\}$.

A constraint split defines a pair of problems such that the union of their solutions is a solution for a problem with the constraint.

Definition 21. *Given a* CSP $\langle V_1 \cup V_2, D_1 \cup D_2, C\rangle, V_1 \cap V_2 = \emptyset$, *a* constraint split *is a pair* $\langle C_1,C_2\rangle$ *such that for each satisfying assignment* f_1 *and* f_2 *for, respectively,* $\langle V_1,D_1,C_1\rangle$ *and* $\langle V_2,D_2,C_2\rangle$:

$$f_1(V_1) \vdash C_1 \wedge f_2(V_2) \vdash C_2 \implies f_1(V_1) + f_2(V_2) \vdash C$$

Example 4.3. Consider Example 4.2: P_2 cannot be further split since the constraint $v_2 \neq v_3$ relates the choices for the two variables. A constraint split for $\{v_2 \neq v_3\}$ is $(\{v_2 = 1\},\{v_3 \neq 1\})$: any union of satisfying assignments for the subproblems $\langle\{v_2\},\{D_2\},\{v_2 = 1\}\rangle$ and $\langle\{v_3\},\{D_3\},\{v_3 \neq 1\}\rangle$ satisfies P_2 as well.

Similarly to problem splits, we generalize constraint splits to constraint shatterings:

Definition 22. *Given a* CSP $\langle V,D,C\rangle$, $V = V_1 \cup V_2 \cup \cdots \cup V_n, V_i \cap V_j = \emptyset, i,j \in \{1,\ldots,n\}, i \neq j$, $D = D_1 \cup D_2 \cup \cdots \cup D_n$, *a* constraint shattering *is a set of problems* $\langle V_i,D_i,C_i\rangle$ *such that for each satisfying assignment* f_i*:*

$$f_1(V_1) \vdash C_1 \wedge f_2(V_2) \vdash C_2 \wedge \cdots \wedge f_n(V_n) \vdash C_n \implies f_1(V_1) + f_2(V_2) + \cdots + f_n(V_n) \vdash C$$

While constraint splits and shatterings preserve satisfiability, they do not preserve the model count, as Example 4.3 shows: the set of satisfying assignments for the split is a subset of the set of solutions for P_2. We then consider sets of constraint splits (shatterings) such that the union of the solutions corresponding to each split (shattering) is the set of satisfying assignments of the original problem. We call such a set of constraint splits (shatterings) a *constraint partition*.

Definition 23. *Given a* CSP $P = \langle V,D,C\rangle$, $V = V_1 \cup V_2 \cup \cdots \cup V_n, V_i \cap V_j = \emptyset, i,j \in \{1,\ldots,n\}, i \neq j$, $D = D_1 \cup D_2 \cup \cdots \cup D_n$, *a* constraint partition $\mathbb{C}(P)$ *is a set of constraint shatterings* $\{\boldsymbol{C}^1 = \langle C_1^1, C_2^1, \ldots, C_n^1\rangle, \ldots, \boldsymbol{C}^m = \langle C_1^m, C_2^m, \ldots, C_n^m\rangle\}$ *such that:*

$$M(V,D,C) = \bigcup_{\boldsymbol{C}^i = \langle C_1^i, C_2^i, \ldots, C_n^i\rangle \in \mathbb{C}(P)} M(V_1,D_1,C_1^i) + M(V_2,D_2,C_2^i) + \ldots + M(V_n,D_n,C_n^i)$$

Constraint partitions can be exploited in counting by summing the counts of the corresponding subproblem. However, this is sound only when no solution is double-counted. This is guaranteed when *non-overlapping* splits (shatterings) are considered:

Definition 24. *Two constraint splits* $\langle C_1^1, C_2^1\rangle$ *and* $\langle C_1^2, C_2^2\rangle$ *for a problem* $\langle V_1 \cup V_2, D, C\rangle$, $V_1 \cap V_2 = \emptyset$, overlap *if there exist two satisfying assignment* f_1 *and* f_2 *such that* $f_1(V_1) \vdash C_1^1, f_1(V_2) \vdash C_2^1, f_2(V_1) \vdash C_1^2, f_2(V_2) \vdash C_2^2$ *and* $f_1(V_1) + f_1(V_2) = f_2(V_1) + f_2(V_2)$*. Otherwise, the two splits are* non-overlapping.

We generalize the notion of non-overlapping splits to constraint shatterings by considering more than two (disjoint) sets of variables and constraints.

A *problem partition* is a set of problem shatterings corresponding to non overlapping constraint shatterings that partition the solution space of the problem into multiple subproblems:

Definition 25. *Given a* CSP $P = \langle V,D,C\rangle$, $V = V_1 \cup V_2 \cup \cdots \cup V_n, V_i \cap V_j = \emptyset, i,j \in \{1,\ldots,n\}, i \neq j$, $D = D_1 \cup D_2 \cup \cdots \cup D_n$, *and a* constraint partition $\mathbb{C}(P)$*, a problem partition* $\mathbb{P}(V,D,C)$ *is the set* $\{\{\boldsymbol{P}_1^i = \langle V_1,D_1,C_1^i\rangle, \ldots, \boldsymbol{P}_n^i = \langle V_n,D_n,C_n^i\rangle\} \mid \boldsymbol{C}^i = \langle C_1^i, C_2^i, \ldots, C_n^i\rangle \in \mathbb{C}(P)\}$.

We can thus express the count of the solutions of a CSP $\langle V,D,C\rangle$ as the sum over all the problem shatterings in the partition (because the respective constraint shatterings are non-overlapping), and, by definition, on each problem shattering we can apply the multiplication rule (because each $\langle V_i,D_i,C_i\rangle$ is pairwise independent):

Property 2. *Given a* CSP $\langle V,D,C\rangle$, $V = V_1 \cup V_2 \cup \cdots \cup V_n, V_i \cap V_j = \emptyset, i,j \in \{1,\ldots,n\}, i \neq j$, $D = D_1 \cup D_2 \cup \cdots \cup D_n$, *and a problem partition* $\mathbb{P}(V,D,C)$ *:*

$$MC(V,D,C) = \sum_{\{P_1,P_2,\ldots,P_n\} \in \mathbb{P}(V,D,C)} \prod_{i=1}^{n} MC(V_i,D_i,C_i)$$

Example 4.4. Consider Example 4.3: we can partition the problem on the splits for constraint $v_2 \neq v_3$: $\{(\langle\{v_2\},\{D_2\},\{v_2 = i\}\rangle, \langle\{v_3\},\{D_3\},\{v_3 \neq i\}\rangle) \mid i \in D_2\}$. Each split i has a count of 2, therefore summing the subproblems from the shattering equals 6. This operation has analogies with propositionalization since a constraint shattering can be obtained by propositionalizing one of the variables in the constraint, as in Example 4.4. When further shattering of the two split problems is required, this operation has also analogies with summing out, a (lifted) variable elimination principle where nested summations are factored w.r.t. independent subproblems.

4.3 CoSo

We now describe how these principles are implemented in CoSo[1], a solver for the full range of problems expressible in CoLa (Section 3.2). CoSo implements our novel general-purpose lifted approach for counting constraint satisfaction problems (Section 4.2) in a solver for the class of problems identified by CoLa.

While the Twelvefold-way provides the counting rules to directly solve certain combinatorics problems (Table 3.2), our framework is more general than it. One of our key contributions presented in this section is a set of efficient counting strategies based on the lifted reasoning principles for CSPs (Section 4.2) to reduce the problems outside the Twelvefold-way to liftable base cases.

Constraints generally prevent the application of counting rules, therefore we rely on splitting and shattering techniques from Section 4.2 to divide a problem into (simpler) subproblems. The subproblems are solved either by an applicable counting rule or by further decomposition into smaller subproblems, in a divide-and-conquer fashion. Figure 4.1 provides a high-level description of how CoSo partitions problems into liftable subproblems. The left side of the diagram represents the steps where exchangeability is exploited to lift the count of the solutions of the problem. The right side corresponds to the divide-and-conquer approach: we collect the counts of the subproblems obtained by splitting and shattering, and from them we derive the count of the valid configurations.

[1] Python implementation: https://github.com/PietroTotis/CoSo

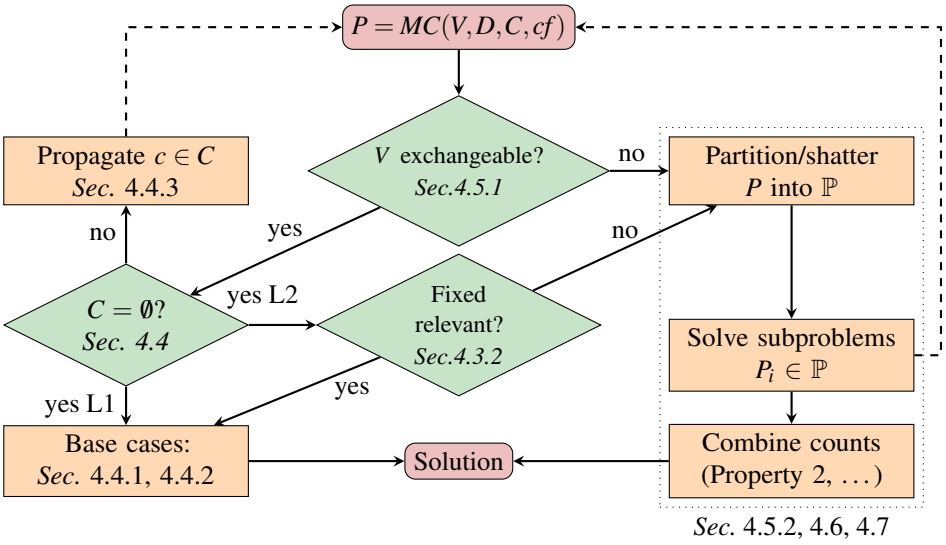


Figure 4.1: Solver reasoning schema. L1= level 1 configurations, L2 = level 2 configurations. Dashed arrows denote recursive calls to the solver.

With the term "lifted" we denote the application of the lifted reasoning principles presented in Section 4.1.2, and not a domain-liftable algorithm running in time polynomial in the size of the domain (Niepert and Van den Broeck 2014). This because in our setting the domain is a multiset and the number and size of each indistinguishable subset affects the complexity of reasoning. Its formal analysis is thus an interesting direction for future work on generalizing the definition of domain-liftable algorithms on multisets.

This section is organized as follows: first, we present the preliminary steps that lead to identifying the case in which the instance falls, and the corresponding operator (Section 4.3.1). We then present these cases starting from the base ones (Section 4.4), followed by the propagation of counting constraints under exchangeability (Section 4.4.3). We then move to the right-hand side of the diagram in Figure 4.1 and consider the dual case when variables are not exchangeable, defining the corresponding partition of the problem (Section 4.5.2). We then extend this operator to the level 1 configurations where the entities are not repeated (Section 4.6), and finally consider the level 2 partition problems when the base cases are not applicable (Section 4.7).

4.3.1 Manipulating Domains, Multisets and Constraints

We now describe how variables, domains, and constraints in $MC(V,D,C,cf)$ are instantiated in CoSo, along with the general grouping techniques applied at any stage. The first step is to consider the size constraint specifying the sizes of valid configuration: let I be the interval of natural numbers defined by the constraint. For each valid $i \in I$, we solve a different #CSP where the number of variables is i. The solution of the problem is thus $\sum_{i \in I} MC(\langle v_1, \ldots, v_i \rangle, D, C, cf)$.

To define the domains, the first step is to define the universe by considering the objects declarations: each declaration by enumeration is directly converted into the corresponding multiset, while the objects belonging to the properties declared by size are inferred as follows. For each statement of the form `#`φ `= n` we convert φ into disjunctive normal form, to identify the unions and intersection of properties that characterize the entities in the set. We assume that all properties are disjoint unless otherwise specified (e.g. with a statement `#`φ_1 `&` φ_2 `= n`). We then build a directed acyclic graph (N,A) on nodes $(\varphi,n) \in N$ corresponding to the declared sets: φ is a set-formula and $n \in \mathbb{N} \cup \{\xi\}$ is a size, where ξ stands for an unknown size. The arcs A denote the subset relation: $(\varphi_i, \varphi_j) \in A$ if φ_j is a subset of φ_i. From this graph, the sizes of the nodes are inferred as follows: first, given a parent (φ, ξ), its size is the sum of its children: we replace ξ with $n_\varphi = \sum_{((\varphi,\xi),(\varphi_i,n_i)) \in E} n_i$, if each child has a known size. Second, given a parent $(\varphi,n), n \neq \xi$, we can infer the size of a child (φ_i, ξ) from the sizes of the other children: we replace ξ with $n_i = n - \sum_{((\varphi,n),(\varphi_j,n_j)) \in A, (\varphi_j,n_j) \neq (\varphi_i,\xi)} n_j$. This is applicable only if a single child has unknown size. We reject incomplete or inconsistent programs, that is, when the inference criteria are not met or there are inconsistent declaration of sizes, i.e. the size of a parent is different form the sum of the sizes of the children.

Once the universe is defined, each domain is instantiated according to the configuration level. Domains are represented implicitly by set formulas in problems of level 1. In level 2, they are represented by a set of (level 2) constraints to exploit the fact that each part belongs to some refinement of the power set of the universe. Then, positional constraints (i, φ) are propagated immediately, by instantiating the corresponding domain D_i to the set φ.

Example 4.5. Consider **P1** and Example 3.6: we want to count the number of 4-permutations where the second object is green and there are 2 squares, corresponding to the CoLa statements:

```
row in [shapes]; #row = 4; row[2] in green;
#row & squares = 2;
```

The corresponding #CSP is $MC(V,D,C,cf)$ where: $V = \langle v_1, v_2, v_3, v_4 \rangle$, $D = \langle \text{shapes}, \text{green}, \text{shapes}, \text{shapes} \rangle$, $C = \{(\text{squares}, \geq, 2)\}$, $cf = [|\cdot|]$.

Example 4.6. Consider **P2** and Example 3.6: we want to count the number of 3-partitions where exactly one of the parts has three green objects, corresponding to the CoLa statements:

```
parts in {{shapes}}; #parts = 3;
#{#part & green = 3} = 1;
```

The corresponding #CSP is $MC(V,D,C,cf)$ where: $V = \langle v_1, v_2, v_3 \rangle$, $D = \langle \{\}, \{\}, \{\} \rangle$, $C = \{((green, =, 3), =, 1)\}$, $cf = \{\{\cdot\}\}$. Note that in level 2 problems the empty domains represent the absence of additional constraints on the actual domain, the power set of the universe.

Multisets group objects according to their indistinguishability (properties). However, we can often identify combinations of properties that are more specific than the counting task requires.

Example 4.7. Consider Example 4.5: the constraints are defined w.r.t. two properties: *green* and *squares*. This means that red and blue squares are interchangeable w.r.t. these properties since they are (both) squares and not green. The following sections show how to exploit this interchangeability to lift the counts for this problem. Therefore, we define relevant properties, relevant parts, and histograms. They allow us to reason on a higher lever of granularity than multisets by grouping not only indistinguishable entities but also the interchangeable ones.

4.3.2 Relevant Properties, Relevant Parts, Histograms

We define relevant properties to formally characterize the groups of entities which are interchangeable w.r.t a given configuration.

Definition 26. *A* relevant property *P is a property such that:*

- *there is a pair of different entities $e_i, e_j \in P$ such that e_i and e_j are indistinguishable.*
- *a counting constraint counts entities from (a subset of) P.*
- *a positional constraint restricts the domain of a variable w.r.t. P.*

A *relevant part* groups together a set of interchangeable entities w.r.t. relevant properties. By capturing only the properties important for solving the given problem, they enable lifting reasoning from the entity level to sets of interchangeable entities. We determine the partitioning of the universe where only relevant properties divide entities from one part to the other as follows:

Definition 27. *Let $\mathscr{P} = \{P_1, \ldots, P_n\}$ be the set of relevant properties, the* relevant parts *of $\mathscr{P}$ are defined recursively, with a distinction in the base case between level 1 and 2. In level 1 configurations $relevant(\mathscr{P}) = \{P_1\}$ if $n = 1$. In level 2 configurations $relevant(\mathscr{P}) = \{P_1, \overline{P_1}\}$ if $n = 1$. In both cases if $n > 1$: $relevant(\{P_1, \ldots, P_{n-1}\} \cup \{P_n\}) = \{P_n \cap R \,|\, R \in relevant(\{P_1, \ldots, P_{n-1}\})\} \cup \{\overline{P_n} \cap R \,|\, R \in relevant(\{P_1, \ldots, P_{n-1}\})\}$.*

The two base cases differ because in level 1 defining the number of entities from P_1 in the configuration also determines the number of $\overline{P_1}$ as a difference with the universe. On the contrary, in level 2 the number of P_1 in a subset does not determine how many entities form $\overline{P_1}$ belong to the subset.

Finally, we associate to relevant partitions a *size*, which we will use in the definitions of the constraint shatterings (Section 4.5).

Definition 28. *A* histogram *for a set of entities E and a set of relevant parts $\mathscr{R}$ is a set $h = \{(n_i, R_i) \,|\, R_i \subseteq E, n_i \in \mathbb{N}\}$ such that $\bigcup_{(n_i,R_i)\in h} R_i = E$ and $0 \leq n_i \leq |R_i|$.*

We denote by $hst(E)$ the set of histograms corresponding to all possible combinations of n_i.

Example 4.8. In Example 4.7 the relevant properties are "*green*", from indistinguishability and "*squares*" from the counting constraints. We want to reason over the cases where the green variable is a square, or vice versa, when the unconstrained variables are a triangle: (green $\cap$ $\overline{\text{squares}}$ and green $\cap$ squares), hence the histograms that summarize the relevant information are:

$$hst(\text{univ}) = \{\{(n_1, \text{green} \cap \overline{\text{squares}}), (n_2, \text{green} \cap \text{squares})\} \,|\, n_1 \in \{0, \ldots, 3\}, n_2 \in \{0\}\}$$

We will use such histograms to define the subproblems that should be considered to properly decompose the counts over the free variables and the one constrained to be "green".

Relevant partitions are fundamental to avoid unnecessary reasoning over a larger number of cases which could otherwise be lifted, without loss of relevant information. In fact, $|relevant(\mathscr{P})| = 2^{n-1}$ in level 1 and $|relevant(\mathscr{P})| = 2^n$ in level 2, therefore reasoning on the coarsest partition can reduce the number of subproblems considered by an exponential factor.

Example 4.9. In Example 4.8 it is not relevant to make distinctions between blue or red objects, since such information does not change the satisfiability of the constraint over green objects. We thus consider only $relevant(\{\text{green}, \text{squares}\})$: a histogram counting the number of green (resp. non-green) objects, $n = n_1 + n_2$ (resp. $|U| - n$,

where U is the universe), accounts for all the different combinations of blue and red objects without considering each case for a subproblem corresponding to the different combinations of non-green objects summing up to $|U| - n$.

4.4 Base Cases

In this section we analyze the case where variables are exchangeable, and thus no partitioning of the problem is required. When there are no counting constraints, then for configurations of level 1 it is possible to count the number of solutions by applying one or more counting rules. The same applies to level 2 configurations but only when the number of entities from the *relevant* partitions is known for each part. When there are counting constraints, we propagate one of the counting constraints with the same operator for both cases.

4.4.1 Level 1: Base Cases

Level 1 configurations correspond to the four top-left cases in Table 3.2, which provide the counting rules for the problems where all entities are distinguishable. In CoLa, however, the universe is a multiset, therefore, we generalize the "all distinguishable" base cases with the corresponding base cases for multisets. In this case, all variables V are exchangeable, hence they share the same domain $dom(V) = (S, f)$.

Sequences Sequences are solved by the formula

$$|dom(V)|^n \tag{4.1}$$

since each of the n variables is an independent choice of any element of $dom(V)$.

Permutations The counting rule for permutations over a multiset $dom(V) = (S, f)$ is the well-known expression $\frac{|dom(V)|!}{f(s_1)!\cdot f(s_2)!\cdot \ldots \cdot f(s_k)!}, s_i \in S, k = |S|$. This formula however counts the permutations of length $|dom(V)|$: to consider the cases where $n \leq |dom(V)|$, we have to consider the multisubsets $N \subseteq dom(V)$ s.t. $|N| = n$, then apply the counting rule returning the number of permutations of M, therefore the number of permutations of $dom(V)$ of length n is:

$$\sum_{M=(T,g)\subseteq dom(V), |M|=n} \frac{|M|!}{f(t_1)!\cdot f(t_2)!\cdot \ldots \cdot f(t_j)!}, t_i \in T, j = |T| \tag{4.2}$$

Subsets The same reasoning applies to subsets: when counting the subsets of size n of $dom(V) = (S, f)$, we can use the binomial coefficient to count the subset of distinguishable entities, but we have to account for the indistinguishable copies in $dom(V)$. Note that this is different from counting multisubsets of S: the number of indistinguishable copies appearing in the subset is limited by how many copies are in $dom(V)$, while in counting multisubsets the number of repetitions of an object in the subset is not bound. Let $S = \{s_1, \ldots, s_k\}, k = |S|$ and let $a_i = f(s_i)$, Ferraris et al. (2015) define a counting rule for this case:

$$\sum_{L \in \mathscr{P}(I_k)} (-1)^{|L|} \binom{n + k - 1 - |L| - \sum_{i \in L} a_i}{k - 1} \tag{4.3}$$

Where $\mathscr{P}(I_k)$ is the power set of $I_k = \{1, 2, \ldots, k\}$.

Multisets Because in counting multisubsets the number of repetitions of an object in the subset is not bound, we reduce the count of multisubsets of size n of a multiset $dom(V)$ to the number of multisubsets of S:

$$\binom{|S| + n - 1}{n} \tag{4.4}$$

since the number of identical copies in $dom(V)$ has no influence on the number of identical copies in the subset.

4.4.2 Level 2: Base Cases

The counting rules from the Twelvefold-way are applicable when there are no additional constraints on the configuration and all objects are either all distinguishable or all indistinguishable.

In this section we define a novel counting rule for level 2 problems over multisets of entities. By considering any type of multiset, we go beyond the rules in the Twelvefold-way, which only contains rules for multisets where the entities are either all distinguishable or all indistinguishable.

Compositions Let $P = MC(V, D, C, cf)$, $V = \langle v_1, \ldots, v_n \rangle$: since relevant parts partition the universe, each part v_i can be described as the union of subsets of each relevant part R_j. In this base case we assume that the number of entities from each R_j is known for each variable: let $\langle h_1, \ldots h_n \rangle$ be the histograms describing such quantities for each part. Relevant parts are by definition disjoint, therefore the choices for a selection from R_j are independent of those from R_k if $j \neq k$. The number of interchangeable

subsets for v_i is thus the product of the number x^i_j of interchangeable subsets of each relevant part R_j. Since relevant parts distinguish between interchangeable and indistinguishable entities, this number is $x^i_j \geq 1$ for the former and $x^i_j = 1$ for the latter. For interchangeable entities x^i_j is the binomial coefficient between the available entities a^i_j and the number n^i_j to be selected. The former is given by how many entities of R_j have already been used: $a^i_j = \sum_{k<i} n^k_j$. Here the order of the variables matters, hence we are counting compositions. The latter is defined by the histogram for v_i: $(n^i_j, R_j) \in h_i$. Therefore:

$$x^i_j = \begin{cases} \binom{a^i_j}{n^i_j} & \text{if entities in } R_j \text{ are interchangeable} \\ 1 & \text{if entities in } R_j \text{ are indistinguishable} \end{cases} \tag{4.5}$$

Finally, the total number of compositions is the product of the choices for each subset, which is the product of the choices for each relevant part:

$$\mathit{compositions}(V) = \prod_{i \in \{1,\ldots,|V|\}} \prod_{(n^i_j, R_j) \in h_i} x^i_j \tag{4.6}$$

Partitions Let $V/\equiv$ be the partition of V into equivalence classes induced by the exchangeability relation, i.e. $V/\equiv$ is $\{[v_1], \ldots, [v_k]\}$ where $[v] = \{v_i \in V \mid v_i \equiv v\}$. The number of partitions is obtained by dividing the number of compositions by the number of indistinguishable permutations of each class of exchangeable variables, that is: $|[v_1]|! \cdot \ldots \cdot |[v_k]|!$ for each $[v_i] \in V/\equiv$. Therefore, the counting rule for partitions with fixed sizes of each relevant part is:

$$\mathit{partitions}(V) = \frac{\mathit{compositions}(V)}{\prod_{[v_i] \in V/\equiv} |[v_i]|}. \tag{4.7}$$

Example 4.10. Consider Example 4.6 and a partition of green entities in three subsets with the following histograms: $\{\{(3, \mathit{green}), (0, \overline{\mathit{green}})\}, \{(0, \mathit{green}), (2, \overline{\mathit{green}})\}, \{(0, \mathit{green}), (2, \overline{\mathit{green}})\}\}$. Green triangles are indistinguishable, therefore no exchangeable solutions w.r.t. green triangles is counted as different, but the count of exchangeable solutions w.r.t. non-green is lifted by Equation 4.7 as: $\binom{4}{2} \cdot \binom{4-2}{2} = 6$. In this case part 2 and 3 are exchangeable, since the number of green and non-green entities is the same, hence the count of partitions is $\frac{6}{2!} = 3$, that is $\{tr_g, tr_g, tr_g\}$ combined with one of: $\{\{sq_r, sq_b\}, \{tr_r, tr_b\}\}$, $\{\{sq_r, tr_b\}, \{tr_r, sq_b\}\}$, $\{\{sq_r, tr_r\}, \{tr_b, sq_b\}\}$ (cf. P_4 in Example 4.15).

4.4.3 Level 1 and 2: Propagating Counting Constraints

Counting constraints are propagated without distinctions between level 1 and 2. When variables are exchangeable and $C \neq \emptyset$ one of the counting constraints $c \in C$ is propagated. Otherwise, the problem is partitioned according to the shattering of one of the counting constraints (Section 4.5.2). Let c be a counting constraint $(\varphi, =, s)$. We focus on equality as the other operators derive from it: let $S = \{s_1, \ldots, s_k\}$ be the set of admissible sizes defined by the constraint. Then, the solution of the problem is $\sum_{s \in S} MC(V, D, C \cup \{(\varphi, =, s)\}, cf)$. To satisfy c we narrow the domain of s variables in V to entities (or parts) in φ. Here exchangeability is exploited: under distinguishability of positions (sequences, permutations, and compositions) there are $e = \binom{n}{s}$ exchangeable choices of variables to propagate c (as usual, $n = |V|$), 1 otherwise.

Algorithm 1 COUNT$_{\equiv}$

Precondition: $P = \langle V, D, C, cf \rangle, \equiv^V, C = \{(\varphi, =, s)\}$
  Operator: COUNT$_{\equiv}(P)$
    let $e = \binom{n}{s}$ **if** $cf \in \{[\cdot], [\{\cdot\}]\}$ **else** 1
    $D_{sat} = \{D_i = dom(V) \cap \varphi \mid i \in \{1, \ldots, s\}\}$
    $D_{unsat} = \{D_i = dom(V) \cap \overline{\varphi} \mid i \in \{s+1, \ldots, n\}\}$
    $P_1 = \langle \{v_1, \ldots, v_s\}, D_{sat}, \{\}, cf \rangle$
    $P_2 = \langle \{v_{s+1}, \ldots, v_n\}, D_{unsat}, \{\}, cf \rangle$
    return $\langle P_1, P_2 \rangle$
Postcondition: $\langle P_1, P_2 \rangle$ is a split for P: $MC(P) = e \cdot MC(P_1) \cdot MC(P_2)$ (*Definition 17*)

Proof. We prove that $\langle P_1, P_2 \rangle$ is a split for P, therefore that the multiplication rule counts e exchangeable solutions of P. The multiplication rule counts solutions of P because each assignment f_1 for P_1 has exactly s entities (parts) belonging to φ, and similarly, an assignment f_2 for P_2 has no entities (parts) belonging to φ. Therefore, each union $f_1 + f_2$ has exactly s entities (parts) in φ. Any assignment $f_1 + f_2$ is thus a satisfying assignment for P. If the exchangeable variables are distinguishable, then for each satisfying assignment $f_1 + f_2$ there are $n!$ exchangeable assignments. $MC(P_1)$ accounts for $s!$ exchangeable assignments within P_1 and $MC(P_2)$ accounts for $(n-s)!$ exchangeable assignments for P_2, therefore there are $\frac{n!}{s!(s-n)!} = \binom{n}{s} = e$ distinguishable exchangeable assignments of the split. □

Configuration	Fixed relevant	Exchangeable	Non-exchangeable
sequence multisubset	n.a.	Base case (Section 4.4.1, Equations 4.1 and 4.4)	
permutation subset		Base case (Sec. 4.4.1, Eq. 4.2 and 4.3)	NOREP_BASE$_{\not\equiv}$ (Section 4.6)
partition composition	$\|\mathscr{R}\| = 0$	Base case (Section 4.4.2, Equations 4.6 and 4.7)	
	$\|\mathscr{R}\| = 1$	PARTS$_{\equiv}$ (Section 4.7.2)	PARTS$_{\not\equiv}$ (Section 4.7.3)
	$\|\mathscr{R}\| > 1$	PART (Section 4.7.4)	

Table 4.2 : Operator cases without counting constraints. (n.a.= not applicable)

4.5 Level 1 and 2: Exchangeability and Shattering Counting Constraints

If none of the base cases are applicable, then variables are not exchangeable, and we resort to splitting and partitioning. Therefore, we have to define variables, domains and constraints for the splits and parts. In Section 4.5.1 we describe how the non-exchangeable variables are divided between the two subproblems of a split. Because the corresponding domains and constraints are specific to the type of problem, we consider particular combinations of exchangeability and constraints, summarized in Tables 4.2 and 4.3. In Section 4.5.2 we begin with the partitioning of counting constraints, which applies to all problems but those with a non-repetition constraint (permutations and subsets).

Configuration	Exchangeable	Non-exchangeable
sequences, multisets partitions, compositions	COUNT$_{\equiv}$ (Section 4.4.3)	COUNT$_{\not\equiv}$ (Section 4.5.2)
permutations subsets		NOREP$_{\not\equiv}$ (Section 4.6)

Table 4.3 : Operator cases with counting constraints.

4.5.1 Exchangeability

When the precondition of exchangeability is not met ($\not\equiv^V$), then we shatter the problem $P = MC(V,D,C,cf)$ into splits $\langle P_1^i, P_2^i\rangle$ such that P_1^i is on exchangeable variables, thus leading to a liftable subproblem, while P_2^i is further shattered if necessary. We define the variables of the two splits of P as follows. Consider the equivalence classes induced by the exchangeability relation $\equiv$ over V: $\equiv$ induces a partition $V/\equiv$ into sets (classes) of exchangeable variables, i.e. $[v] = \{v_i \in V \mid v_i \equiv v\}$. Each class $[v]$ is a set of exchangeable variables and when V is not exchangeable $|V/\equiv| > 1$. To define $\langle P_1^i, P_2^i\rangle$ we consider a class $[v] \in V/\equiv$, and the remaining variables $\hat{V} = \{w \,|\, w \in [v_i], [v_i] \in V/\equiv \setminus \{[v]\}\}$. Then $P_1^i = ([v], D^i_{[v]}, C^i_{[v]}, cf)$ and $P_2^i = MC(\hat{V}, D^i_{\hat{V}}, C^i_{\hat{V}}, cf)$ with problem P_1^i defined on exchangeable variables. The domains $D^i_{[v]}$ and $D^i_{\hat{V}}$ and constraints $C^i_{[v]}$ and $C^i_{\hat{V}}$ depend on the specific splitting operator.

Example 4.11. In Example 4.5 domain $D = \langle \text{univ}, \text{green}, \text{univ}, \text{univ}\rangle$ hence $[v_1] = \langle v_1, v_3, v_4\rangle$ and $[v_2] = \{v_2\}$. If the second class $[v_2]$ is chosen for the split, then $\hat{V} = \{v_1, v_3, v_4\}$. Note that in this example there are only two classes, hence $\hat{V}$ is also exchangeable, but when $|V/\equiv| > 2$ this is not the case.

4.5.2 Level 1 and 2: Shattering Counting Constraints

Again, we make no distinction between the two levels when reasoning on counting constraints, but we distinguish between level 1 configuration with or without repetition (Table 4.3). When variables are not exchangeable and $C \neq \emptyset$, we partition the problem to account for the different choices of variables to propagate the counting constraints. In this section we shatter counting constraints alone, in Section 4.6.2 we define the shattering of both size and non-repetition constraints.

We shatter a counting constraint $c = (F, =, s)$ into the possible numerical contribution from the split class $[v]$ and the other variables in $\hat{V}$, defined as in the previous section. The shattering of multiple counting constraints is obtained by recursively combining the individual constraint shattering with the splits of the other constraints. Let $P = \langle V, D, C, cf\rangle$ and let c be a counting constraint: we denote with $P + c$ the problem $\langle V, D, C \cup \{c\}, cf\rangle$.

Algorithm 2 $\text{COUNT}_{\not\equiv}$

Precondition: $P = MC(V,D,C,cf) : \not\equiv^V, C = \{(\varphi,=,s)\} \cup C', cf \notin \{[|\cdot|],\{|\cdot|\}\}$
Operator: $\text{COUNT}_{\not\equiv}(P)$
$\mathbb{P} = \{\}$
if $C' = \emptyset$ **then**
for i **in** $\{0,\ldots,s\}$ **do**
$P_1^i = \langle [v], D_{[v]}, \{(\varphi,=,i)\}, cf\rangle$
$P_2^i = \langle \hat{V}, D_{\hat{V}}, \{(\varphi,=,s-i)\}, cf\rangle$
$\mathbb{P} = \mathbb{P} \cup \{\langle P_1^i, P_2^i\rangle\}$
else
$\mathbb{P}' = \text{COUNT}_{\not\equiv}(V,D,C',cf)$
for $\langle P_1^j, P_2^j\rangle \in \mathbb{P}'$ **do**
for $i \in \{0,\ldots,s\}$ **do**
$P_1^i = P_1^j + \{(\varphi,=,i)\}$
$P_2^i = P_2^j + \{(\varphi,=,s-i)\}$
$\mathbb{P} = \mathbb{P} \cup \{\langle P_1^i, P_2^i\rangle\}$
return $\mathbb{P}$
Postcondition: $\mathbb{P}$ is a partition for P: $MC(P) = \sum_{\langle P_1^i, P_2^i\rangle \in \mathbb{P}} MC(P_1^i) \cdot MC(P_2^i)$ *(Property 2)*

Proof. By induction on $|C|$. We prove that $\mathbb{P}$ is a partition, hence that each pair in $\mathbb{P}$ is a split, that does not overlap with the others, and that all solutions of $\mathbb{P}$ are counted.

Base case: $|C| = 1$. If $|C| = 1$ then $C' = \emptyset$ and the pairs $\langle P_1^i, P_2^i\rangle$ are defined accordingly. Each $\langle P_1^i, P_2^i\rangle$ is a split for P since $[v] \cap \hat{V} = \emptyset$ and each satisfying assignment f_1 (resp. f_2) for P_1 (resp. P_2) has exactly i (resp. $s-i$) entities (parts) from φ, therefore a satisfying assignment $f_1 + f_2$ for P has $i + s - i = s$ entities (parts) from φ. The fact that we consider a different i for each pair ensures that the solutions do not overlap, and by considering each $i \in \{0,\ldots,s\}$ we cover all the possible cases summing up to s, hence $\mathbb{P}$ is a partition for $\langle V,D,C,cf\rangle$.

Inductive case: $|C| > 1$. Let $C = \{(\varphi,=,s)\} \cup C'$ with $C' \neq \emptyset$, let $\mathbb{P}'$ be a partition $\langle V,D,C',cf\rangle$ and let $\langle P_1^j, P_2^j\rangle$ be a split in $\mathbb{P}'$. Let f_1 and f_2 be respectively a solution for P_1^j and P_2^j: $f_1 + f_2$ is a satisfying assignment $\langle V,D,C',f\rangle$. The same arguments from the base case apply: $\langle P_1^j + (\varphi,=,i), P_2^j + (\varphi,=,s-i)\rangle$ is a split for P: $[v] \cap \hat{V} = \emptyset$ and the new counting constraints ensure that each union of solutions of the two splits satisfies $\{(\varphi,=,s)\}$. The different values of i define non-overlapping splits and cover all the possible cases to partition s entities from φ between the two subproblems, hence $\mathbb{P}$ is a partition for $\langle V,D,C,cf\rangle$. □

4.6 Level 1: Shattering Non-repetition

The distinction between configurations with or without repetition is specific to level 1 configurations. Like for counting constraints, we consider the two cases: exchangeable and non-exchangeable variables. If variables are exchangeable, then we can apply a base case (Section 4.4.1). In this section we describe the case where variables are not exchangeable. First, we define the case without additional counting constraints, $|C| = 0$ (Section 4.6.1), then we generalize it to $|C| > 0$ (Section 4.6.2).

4.6.1 Level 1: Shattering Repetition Without Counting Constraints

The non-repetition constraint in $P = \langle V, D, C, cf \rangle$ is split into two problems P_1 and P_2 where the respective variables $[v]$ and $\hat{V}$ are defined as in Section 4.5.1. Let $D_{[v]}$ and $D_{\hat{V}}$ be the corresponding domains in D. We obtain a split by fixing a number k of entities in each relevant property for variables $[v]$, such that the problem on $\hat{V}$ is solved knowing that k fewer different choices are available. We thus exploit interchangeability of entities to avoid reasoning on each possible entity choice in a solution for P_1. If variables are not exchangeable then there are some positional constraints on the variables that determine different domains, therefore the relevant properties, in addition to the groups of indistinguishable entities, are the equivalence classes of the exchangeability relation over $\hat{V}$: $\mathscr{P} = \{W | [W] \in \hat{V} / \equiv\}$. We define histograms for $[v]$, as usual, as the set of the combinations of the possible cardinalities of the relevant partitions $relevant(\mathscr{P})$. Then we solve P_1 by constraining each of its satisfying assignments f_1 to yield the given histogram, and we solve P_2 knowing for each relevant property (and partition) how many different entities are already in the partial assignment f_1 for P. To do so, we denote with $D - h$ the set of domains obtained by removing from each $D_i \in D$ a number of entities r from each relevant partition R such that $(r, R) \in h$.

Algorithm 3 NOREP_BASE$_{\not\equiv}$

Precondition: $P = MC(V, D, C, cf) : \not\equiv^V, |C| = 0$
Operator: NOREP_BASE$_{\not\equiv}(P)$
$\mathbb{P} = \{\}$
for h **in** $hst(dom([v]))$ **do**
$C_1^h = \{(R, =, r) \mid (r, R) \in h\}$
$P_1^h = \langle [v], D_{[v]}, C_1^h, f \rangle$
$P_2^h = \langle \hat{V}, D_{\hat{V}} - h, \{\}, f \rangle$
$\mathbb{P} = \mathbb{P} \cup \{\langle P_1^h, P_2^h \rangle\}$
return $\mathbb{P}$
Postcondition: $\mathbb{P}$ is a partition for P: $MC(P) = \sum_{\langle P_1^h, P_2^h \rangle \in \mathbb{P}} MC(P_1^h) \cdot MC(P_2^h)$ *(Prop. 2)*

Proof. We prove that each $\langle P_1^h, P_2^h \rangle$ is a (non-overlapping) split of P and that the sum of the counts for each split is the count for P. By definition the histogram is a partition of the relevant properties and the respective cardinalities are a different (valid) combination for each $h \in hst(dom([v]))$. This guarantees that the pairs are non-overlapping and cover all possible satisfiable cases.

To prove that each $\langle P_1^h, P_2^h \rangle$ is a split, we note that a satisfying assignment f_2 for P_2 excludes a *specific* selection of entities according to a histogram h, but there is no guarantee that any solution f_1 for P_1 is such that $f_1([v]) \cap f_2(\hat{V}) = \emptyset$ and thus that $f_1 + f_2$ is a solution for P. Therefore, we prove that each union of the two satisfying assignments is a solution for P up to a renaming of the entities.

Let H be the set of entities removed from the domains $D_{\hat{V}}$. If f_1 maps variables to exactly the entities in H then $f_1 + f_2$ is trivially a satisfying assignment for P, since f_2 does not map any variable to H. If this is not the case then there is at least an entity e in a relevant partition R such that $e \in R \backslash H$ and $e \in f_1([v])$. We examine the individual case which can be generalized to multiple entities. If $e \notin f_2(\hat{V})$ then $f_1 + f_2$ is a satisfying assignment for P. If $e \in f_2(\hat{V})$ then $f_1 + f_2$ is not a satisfying assignment for P, however this entails the existence of an entity $e' \in R \cap H$ which does not appear in $f_1([v])$. This because $|R \cap [v]| = |R \cap H| = r$ and e is in $f_1([v])$, but not in H, otherwise $e \notin f_2(\hat{V})$. Therefore, by renaming e with e' in $f_2(\hat{V})$, which are *interchangeable* entities since they belong to the same relevant partition R, $f_1([v]) + f_2(\hat{V})$ corresponds to a satisfying assignment. □

4.6.2 Level 1: Shattering Repetition With Counting Constraints

We generalize the operators $\text{NOREP_BASE}_{\not\equiv}(P)$ and $\text{COUNT}_{\not\equiv}(P)$ to shatter problems where both counting constraints and the non-repetition constraint are present. We combine the two approaches presented in the previous sections into a general operator for non-repetition where histograms define the shattering of counting constraints as well as the non-repetition constraints. Let $[v]$ and $\hat{V}$ be the usual split of the non-exchangeable variables of $P = MC(V, D, C, cf)$. The relevant properties $\mathscr{P}$ in this case are the union of the relevant properties for counting constraints and non-repetition: $\forall c \in C, c$ is $(\varphi, =, s) : \varphi \in \mathscr{P}$ and $\forall [w] \in \hat{V} / \equiv : dom([w]) \in \mathscr{P}$, which also contain the information about indistinguishability. From the histograms on $\mathscr{P}$ we define the constraints and domains for the split problems P_2^h similarly to the previous operators: complementary counting constraints for the two split problems to satisfy the counting constraints over P and domain exclusion on variables in $\hat{V}$ to count satisfying assignments w.r.t. non-repetition.

Algorithm 4 NOREP$_{\not\equiv}$

Precondition: $P = MC(V,D,C,cf) : \not\equiv^V$
Operator: NOREP$_{\not\equiv}(P)$
$\mathbb{P} = \{\}$
for h **in** $hst(\mathscr{P})$ **do**
$C_1^h = \{(R,=,r) \,|\, (r,R) \in h\}$
$C_2^h : \{(\varphi,=,s-i) \,|\, c : (\varphi,=,s), c \in C, i = \sum_{(r,R)\in h, R\subseteq\varphi} r\}$
$P_1^h = \langle [v], D_{[v]}, C_1^h, cf\rangle$
$P_2^h = \langle \hat{V}, D_{\hat{V}} - h, C_2^h, cf\rangle$
$\mathbb{P} = \mathbb{P} \cup \{\langle P_1^h, P_2^h\rangle\}$
return $\mathbb{P}$
Postcondition: $\mathbb{P}$ is a partition for P: $MC(P) = \sum_{\langle P_1^h, P_2^h\rangle \in \mathbb{P}} MC(P_1^h) \cdot MC(P_2^h)$ *(Property 2)*

Proof. The arguments for this proof are similar to those previously presented in the proofs for the two distinct constraint shatterings: constraints C_2^h ensure that for each counting constraint $(\varphi,=,s)$ the number of entities in φ in the union of the satisfying assignments for P_1^h and P_2^h is s. At the same time the domains $D_{\hat{V}} - h$ ensure the correctness of the count of the assignments for P that satisfy the non-repetition constraint. Histograms guarantee non-overlapping splits accounting for all solutions of P. □

In Section 4.8 we give a detailed description of how Example 4.11 is partitioned and solved by applying the operators presented in this section.

4.7 Level 2: Shattering Parts

Level 2 problems require us to distinguish two levels of properties in a problem $P = MC(V,D,C,cf)$: global properties of the parts, i.e. counting constraints C (level 2 properties), and local properties of each individual subset (level 1 properties), i.e. the set of constraints defining D_i for each part i. We propagate the former as described in Section 4.5.2, therefore in this section we assume that $C = \emptyset$ and focus on the local constraints, that is, counting the valid combinations of domains resulting from this propagation.

Example 4.12. Consider Example 4.6: after the propagation of the counting constraint the #CSP is: $V = \langle v_1, v_2, v_3\rangle$, $D = \langle\{(green,=,3)\},\{(green,\neq,3)\},\{(green,\neq,3)\}\rangle$, $C = \emptyset$, $cf = \{\{\cdot\}\}$. Clearly, level 1 constraints prevent the application of the counting rules (Table 3.2) for problems of level 2.

Next, we generalize the approach in Section 4.4.2 for shattering a level 2 problem into n subset problems to the case where the number of entities from each relevant part is not known for each subset. Intuitively, we fix the content of each subset by considering one relevant part at a time. We begin with fixing the number of entities from a relevant part in all subsets, and recursively add on top of this distribution the possible distributions for the remaining relevant parts. In doing so, we have to account for the satisfiability of the constraints and exchangeability.

Example 4.13. In Example 4.6 propagating the number of green objects in the parts results in Example 4.12. While the number of green entities in the three parts is fixed by the counting constraint to be $\langle 3,0,0\rangle$, the number of non-green entities can vary, e.g. $\langle 0,2,2\rangle$, $\langle 1,1,2\rangle, \ldots$ Of the possible distributions of the 4 entities into the 3 parts $\langle 4,0,0\rangle$ and $\langle 3,1,0\rangle$ (and the exchangeable $\langle 3,0,1\rangle$) are not valid distributions because at least one part would be empty.

In Section 4.7.1 we give a high-level description of the function *distribute*, which ensures that the configuration constraints remain satisfiable at each step. In the rest of the section we present the operators that account for exchangeability in the definition of the distributions of each relevant part. We distinguish three cases: (1) the case where variables are exchangeable and a count of a relevant property can be propagated (Section 4.7.2); (2) the case where variables are not exchangeable (Section 4.7.3); and (3) the general case with multiple relevant properties, solved by means of recursion (Section 4.7.4).

4.7.1 Distributing Entities

The goal of the operators presented in this section is to define a histogram for each subset of the partition or composition. These histograms must satisfy three kinds of constraints: the two defined by the configuration type *cf* plus the additional constraints, namely:

1. parts partition the universe, hence all entities must belong to some part;
2. all parts are non-empty;
3. the individual counting constraints restricting the domains of the parts.

We distribute one relevant part at a time, which means that the entities at each step are either indistinguishable or interchangeable, therefore we can do lifted reasoning over the number of the entities rather than their exact identity. We thus divide the entities by considering *integer distributions*. An integer distribution of r over k parts is an integer partition where summands can be zero. In fact, a subset can have zero entities from a

relevant part as long as no constraint is violated. The satisfiability of the constraints is ensured by a function *distribute*. The function $distribute(P,R,r,k)$ takes as input the problem P, the number of entities r from the set R, and the number of parts k. It returns a set I of tuples of integers of the form $\langle i_1,\ldots,i_k\rangle$ describing an integer distribution of r in k parts, such that no constraint in P is violated.

Integer distributions ensure that all entities of a relevant part are assigned to some subset (constraint 1). Moreover, we use bounds consistency techniques to consider only integer distributions of $|R|$ that satisfy constraints (2) and (3). For instance, if the valid subsets for v_j have 0 entities from all relevant partitions but R, then $i_j > 0$ (constraint 1). Example 4.13 shows the distributions of entities from the universe excluded by $distribute(P,\overline{green},4,3)$. We also exclude the distributions where i_j does not satisfy a local constraint about R in D_j (constraint 3).

We rely on the function *distribute* in both cases where variables are exchangeable (Section 4.7.2) and they are not exchangeable (Section 4.7.3), with the difference that under exchangeability we choose one ordering for the distribution $\langle i_1,\ldots,i_k\rangle$ and account for the exchangeable ones in a lifted manner. On the contrary, when variables are not exchangeable we consider a distribution over exchangeability classes rather than individual variables, therefore different orderings in the integer distribution define different problems. For this reason in this case we have to consider explicitly the different permutations in propagating the number of entities of a relevant partition.

4.7.2 One Non-fixed Relevant Part: Exchangeable Variables

When variables are exchangeable (case 1) we compute the set of valid integer distributions and for each of them consider a new subproblem. In each suproblem corresponding to a distribution $\mathbf{i} = \langle i_1,\ldots,i_k\rangle$, the number of entities from R is fixed to be i_j for each v_j in V by adding to D_j a counting constraint $(R,=,i_j)$. Variables are exchangeable, therefore we can lift the count of exchangeable assignments. Similarly to the propagation of counting constraints, we pick an order of $\{i_1,\ldots,i_k\}$ for the tuple of variables and account with a constant for the exchangeable ways of propagating the corresponding counting constraints to the variables. We denote such constant with $e(cf,\mathbf{i})$. If the configuration is a partition, exchangeable assignments are indistinguishable, hence $e(\{\{\cdot\}\},\mathbf{i}) = 1$ for any $\mathbf{i}$. Otherwise, the number of exchangeable propagations of $\mathbf{i}$ is $e([\{\cdot\}],\mathbf{i}\}) = \frac{|\mathbf{i}|}{n_1!\cdot\ldots\cdot n_j!}$ where $n_1,\ldots,n_j$ count the occurrences of each of the j different integers in $\mathbf{i}$.

Algorithm 5 PARTS$_{\equiv}$

Precondition: $P = \langle V, D, C, cf \rangle : \equiv^V, C = \emptyset, \mathscr{R} = \{(r, R)\}$
Operator: PARTS$_{\equiv}(P, \mathscr{R})$
$\mathbb{P} = \{\}$
for $\mathbf{i} = \langle i_1, \ldots, i_{|V|} \rangle$ **in** *distribute*$(P, R, |V|)$ **do**
$D' = \{D_j \cup \{(R, =, i_j)\} \mid D_j \in D\}$
$\mathbb{P} = \mathbb{P} \cup \{(e(cf, \mathbf{i}), \langle V', D', C, cf \rangle)\}$
return $\mathbb{P}$
Postcondition: $\mathbb{P}$ is a partition for P: $MC(P) = \sum_{(c, \langle V', D', C, cf \rangle) \in \mathbb{P}} c \cdot MC(V', D', C, cf)$ *(Property 2)*

Proof. We prove that $\mathbb{P}$ is a partition of P, hence that $\mathbb{P}$ defines non-overlapping shatterings that cover the solution space of P. If P is unsatisfiable, i.e. $MC(V, D, C, cf) = 0$, there are two cases. Either *distribute*$(P, R, |V|) = \emptyset$, hence $\mathbb{P} = \emptyset$ and $MC(V', D, C, cf) = 0$, or regardless of how we partition the entities in R, the problem remains unsatisfiable, hence $MC(V', D', C, cf) = 0$ for all V', D'. We now consider the case where P is satisfiable, hence we prove that $\mathbb{P}$ defines subproblems corresponding to non-overlapping constraints, from which the model count of P can be derived. If P is satisfiable then there exists at least a solution f_1 where the entities of R are partitioned across V. Let $\mathbf{i} = \langle i_1, \ldots, i_{|V|} \rangle$ be the distribution of entities from R in each of the $|V|$ parts in such solution. Since this is a valid distribution of R w.r.t. V, then it belongs to *distribute*$(P, R, |V|)$. $\mathbf{i}$ corresponds to a subproblem of P where each part v_j is constrained to contain exactly i_j entities in R. f_1 is a solution for such subproblem and since variables are exchangeable, f_1 is one of the $c = e(cf, \{i_1, \ldots, i_{|V|}\})$ different exchangeable assignments of each $i_k \in \{i_1, \ldots, i_{|V|}\}$ to some part. A set of constraints corresponding to a distribution $\mathbf{i}' \neq \mathbf{i}$ does not overlap with those corresponding to $\mathbf{i}$: if $\mathbf{i}' \neq \mathbf{i}$ then there is either some $i'_k \in \mathbf{i}'$ such that $i'_k \notin \mathbf{i}$ or such that $i'_k \in \mathbf{i}$ and the number of occurrences is different between $\mathbf{i}'$ and $\mathbf{i}$. Therefore, no assignment for V can satisfy both sets of constraints at the same time, hence the constraints corresponding to *distribute*$(P, R, r, |V|)$ are non-overlapping. Since *distribute*$(P, R, r, |V|)$ contains all valid distributions of R, we can conclude that $\mathbb{P}$ partitions P.

□

4.7.3 One Non-fixed Relevant Part: Non-exchangeable Variables

If variables are not exchangeable (case 2), we distribute r entities from a relevant property R into the exchangeable classes first, then propagate the constraints in each class. The first step is done by *distribute*(P, R, r, k), which produces a distribution of

r, $\mathbf{i} = \langle i_1, \ldots, i_k \rangle$,over the k exchangeability classes. This means that the j^{th} class corresponds to a new problem of distributing i_j entities from R over its exchangeable variables (parts), that is, case 1. Each distribution within a class is independent of the others, hence they can be united in a solution for the original problem. Given two sets $A = \{(e_1^A, P_1^A), \ldots, (e_n^A, P_n^A)\}$ and $B = \{(e_1^B, P_1^B), \ldots, (e_m^B, P_m^B)\}$ we denote with $A \times B$ the union of the two sets of problems in a cartesian-product form: $\{(e_1^A \cdot e_1^B, P_1^A \cup P_1^B), \ldots, (e_1^A \cdot e_m^B, P_1^A \cup P_m^B), \ldots, (e_n^A \cdot e_m^B, P_n^A \cup P_m^B)\}$ where $P_i^X = \langle v_i^X, D_i^X, C, cf \rangle$ and $P_i^A \cup P_j^B = \langle v_i^A \cup v_j^B, D_i^A \cup D_j^B, C, cf \rangle$. We also denote with $\times\{A_1, A_2, \ldots, A_n\}$ the operation $A_1 \times A_2 \cdots \times A_n$. The operator corresponding to case 2 thus distributes entities across the exchangeability classes first, then within each class according to case 1, and finally combines the case 1 distributions of each class with those from the other classes.

Algorithm 6 PARTS$_{\not\equiv}$

Precondition: $P = \langle V, D, C, cf \rangle : \not\equiv^V, C = \emptyset, \mathscr{R} = \{(r, R)\}$
  Operator: PARTS$_{\not\equiv}(P, \mathscr{R})$
    let $V/\equiv$ **be** $\{V_1, \ldots, V_k\}$
    $\mathbb{P} = \{\}$
    for $\mathbf{i} = \langle i_1, \ldots, i_k \rangle$ **in** *distribute*(P, R, k) **do**
      $join = \times\{\text{PARTS}_{\equiv}(\langle V_j, D_{V_j}, C, f \rangle, \{(i_j, R)\}) \,|\, V_j \in V/\equiv\}$
      $\mathbb{P} = \mathbb{P} \cup join$
    return $\mathbb{P}$
Postcondition: $\mathbb{P}$ is a partition for P: $MC(P) = \sum_{(c, \langle V', D', C, cf \rangle) \in \mathbb{P}} c \cdot MC(V', D', C, cf)$ *(Property 2)*

Proof. As in the exchangeable case, we prove that $\mathbb{P}$ is a partition of the solutions of P. We distribute the entities of R across the exchangeability classes: classes are not exchangeable hence different orders in the integer partition define different subproblems. For a given integer distribution, each exchangeability class corresponds to a set of valid distributions of the given number of entities as in PARTS$_{\equiv}(P, \mathscr{R})$ hence they do not overlap. Therefore, a solution for one class can be united with one solution of each class: classes are distinguishable hence each subproblem is different from each other. The result is a satisfiable subproblem of P since the sum of the entities in R is equal to $|R|$ and no constraint in D is violated.

□

4.7.4 Many Non-fixed Relevant Parts

Finally, when more than one relevant property requires to be propagated (case 3), we propagate one relevant property R and recursively consider the different subproblems corresponding to each valid distribution of R across the parts.

Algorithm 7 PART

Precondition: $P = \langle V, D, C, cf \rangle : C = \emptyset, \mathscr{R} = \{(r, R)\} \cup \mathscr{R}'$
  Operator: PART$(P, \mathscr{R})$
    if $|\mathscr{R}'| > 0$ **then**
      $\mathbb{P}' =$ PART$(P, \mathscr{R}')$
      for $(c, \langle V', D', C, cf \rangle)$ **in** $\mathbb{P}'$ **do**
        if $|V'/\equiv| = 1$ **then**
          $\mathbb{P} =$ PARTS$_{\equiv}(\langle V', D', C, cf \rangle, \{(r, R)\})$
        else
          $\mathbb{P} =$ PARTS$_{\not\equiv}(\langle V', D', C, cf \rangle, \{(r, R)\})$
    else
      if $|V/\equiv| = 1$ **then**
        $\mathbb{P} =$ PARTS$_{\equiv}(P, \mathscr{R})$
      else
        $\mathbb{P} =$ PARTS$_{\not\equiv}(P, \mathscr{R})$
    return $\mathbb{P}$
Postcondition: $\mathbb{P}$ is a partition for P: $MC(P) = \sum_{(c, \langle V', D', C, cf \rangle) \in \mathbb{P}} c \cdot MC(V', D', C, cf)$ *(Property 2)*

In Section 4.8 we complete Examples 4.12 and 4.13 with a detailed example of the application of the operators for level 2 configurations.

4.8 Detailed Examples

We give a step-by-step description of how the lifted reasoning techniques implemented in the operators in CoSo solve combinatorics math problems of level 1 (with a non-repetition constraint) and level 2, with counting and positional constraints. For each lifted step we describe the corresponding ground configurations that are counted.

Example 4.14.

We complete Example 4.11: the goal is to count the number of 4-permutations where the second object is green and there are 2 squares. Let $univ = squares \cup triangles = \{$■, ■, ▲, ▲, ▲, ▲, ▲$\}$. As described in Example 4.11 the result of the propagation of the positional constraint is the #CSP $P = MC(V, D, C, cf)$, where $V = \langle v_1, v_2, v_3, v_4 \rangle$, $D = \langle \text{univ}, \text{green}, \text{univ}, \text{univ} \rangle$, $C = \{(\text{squares}, \geq, 2)\}$, $cf = [|\cdot|]$. We represent the corresponding configuration as $\langle \bigstar, ▲, \bigstar, \bigstar \rangle$, where $\bigstar$ denotes an undecided shape with undecided colour.

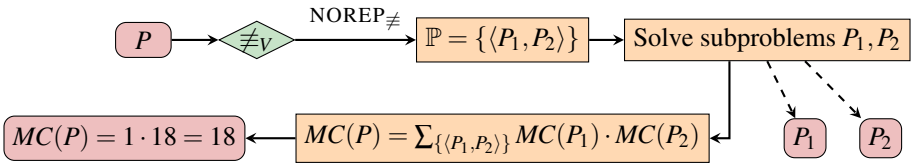


Figure 4.2: Solver execution flow for **P1**

First, we notice that variables are not exchangeable due to the positional constraint, hence we have to partition the problem. We pick $[v_2]$ as the split class for $\langle \text{univ}, \text{green}, \text{univ}, \text{univ} \rangle$, thus $P_1 = \langle V_1, D_1, C_1^h, cf \rangle$ with $V_1 = \langle v_2 \rangle, D_1 = \langle green \rangle$ and $P_2 = \langle V_2, D_2, C_2^h, cf \rangle$ with $V_2 = \langle v_1, v_3, v_4 \rangle, D_2 = \langle univ, univ, univ \rangle$. Then, following NOREP$_{\not\equiv}$, we consider the different histograms counting the relevant properties in a solution for P_1 to solve P_2 independently.

In Example 4.8 we define the relevant parts *green* and *squares*, along with the histograms in Example 4.11. Of the possible histograms for P_1, only $h = \{(1, green \cap \overline{squares}), (0, squares \cap green)\}$ is feasible. Therefore, we consider just one split where $C_1^h = \{(green \cap \overline{squares}, =, 1), (squares \cap green, =, 0)\}$ and $C_2^h = \{(squares, =, 2)\}$. The domains of P_2 are then updated ($D_{\hat{V}} - h$) such that one $green \cap \overline{squares}$ entity is removed from the universe ($univ^* = \{\blacksquare, \blacksquare, \blacktriangle, \blacktriangle, \blacktriangle, \blacktriangle\}$, hence $\overline{squares} = \{\blacktriangle, \blacktriangle, \blacktriangle, \blacktriangle\}$ in P_2).

P_1 First we solve P_1: the variables ($\langle v_2 \rangle$) are exchangeable, therefore the counting constraints can be propagated (COUNT$_{\equiv}$): the domain of v_2 remains unchanged in the resulting problem P_1^*. C_1^h is now empty and a counting rule can be applied, returning 1 as the number of solutions for P_1^* and since $e = 1$ this is the solution for P_1 as well.

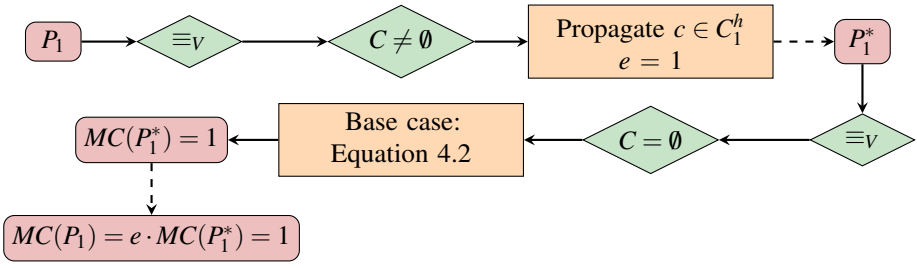


Figure 4.3: Solver execution flow for the first subproblem of **P1**

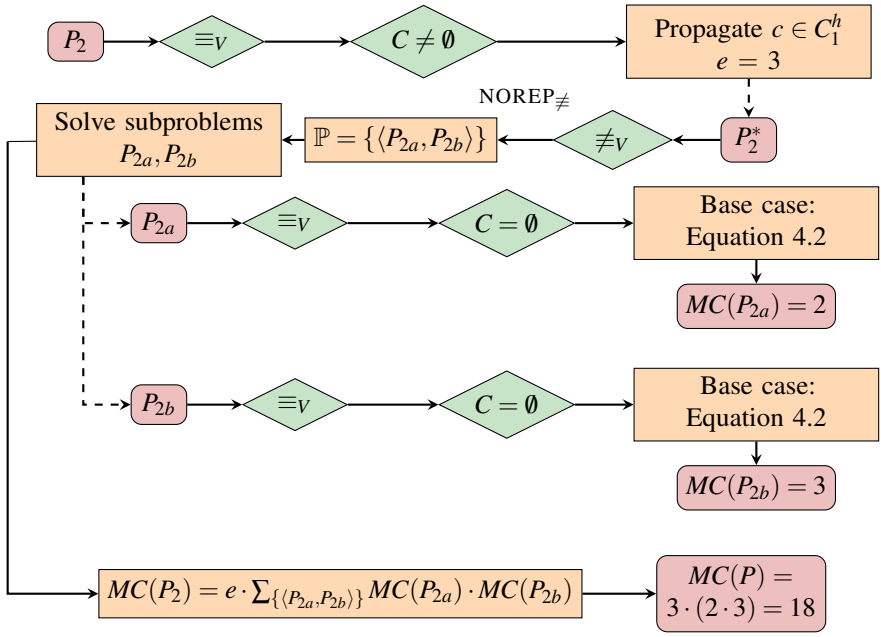


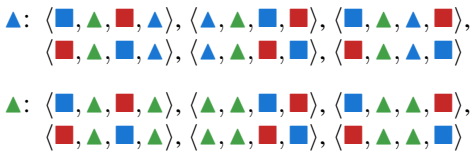
Figure 4.4: Solver execution flow for the second subproblem of **P1**

P_2 In P_2 variables are exchangeable and there is a counting constraint: COUNT$_{\equiv}$ propagates the constraint and the resulting domains (in P_2^*) are $\langle squares, squares, \overline{squares}\rangle$ with 3 exchangeable choices. This step corresponds to the (non-lifted) configurations: $\langle$■,▲,■,▲$\rangle$, $\langle$▲,▲,■,■$\rangle$, $\langle$■,▲,▲,■$\rangle$. Now variables are not exchangeable hence we split (skipping trivial steps) P_2^* into $P_{2a} = \langle\langle v_1, v_3\rangle, \langle squares, squares\rangle, \{\}, cf\rangle$ and $P_{2b} = \langle\langle v_4\rangle, \langle\overline{squares}\rangle, \{\}, cf\rangle$. Both now are base cases: $MC(P_{2a}) = 2! = 2$. This means lifting the count of:

- ■■: $\langle$■,▲,■,▲$\rangle$, $\langle$▲,▲,■,■$\rangle$, $\langle$■,▲,▲,■$\rangle$
- ■■: $\langle$■,▲,■,▲$\rangle$, $\langle$▲,▲,■,■$\rangle$, $\langle$■,▲,▲,■$\rangle$

$MC(P_{2b}) = 3$ because the two green triangles in $\overline{squares}$ are indistinguishable and the different interchangeable choices are only between a red/blue/green triangle. P_{2b} lifts the count of:

▲: $\langle$■,▲,■,▲$\rangle$, $\langle$▲,▲,■,■$\rangle$, $\langle$■,▲,▲,■$\rangle$,
$\langle$■,▲,■,▲$\rangle$, $\langle$▲,▲,■,■$\rangle$, $\langle$■,▲,▲,■$\rangle$



Therefore the solution for P_2 is the product of the 3 exchangeable choices for propagating the number of squares: $MC(P_2) = 3 \cdot (2 \cdot 3) = 18$. The solution of the problem is thus:

$$MC(P) = MC(P_1) \cdot MC(P_2) = 1 \cdot 18 = 18.$$

Example 4.15.

We complete Examples 4.12 and 4.13: the goal is to count the number of partitions in three non-empty subsets where green triangles all belong to the same part. The propagation of the counting constraint picks one of the parts to be the one containing the triangles. Since variables are exchangeable but the parts are indistinguishable, there are no interchangeable solutions to count. After the counting constraint propagation the domains of the three variables not all sizes of relevant parts ($\{green, \overline{green}\}$) are known in each partition (Example 4.13): $V = \langle v_1, v_2, v_3 \rangle$, $D = \langle \{(green, =, 3)\}, \{(green, \neq, 3)\}, \{(green, \neq, 3)\} \rangle$, $C = \emptyset$, $cf = \{\{\cdot\}\}$. Variables $\langle v_1, v_2, v_3 \rangle$ are not exchangeable, i.e. $[v_1] = \{v_1\}, [v_2] = \{v_2, v_3\}$.

We thus apply the operator $\text{PARTS}_{\not\equiv}$: the only satisfiable composition of *green* is $[3,0]$ (because the counting constraint already determined a part for all green entities), let P'' denote the corresponding partition. We thus propagate the distribution to the two exchangeability classes with $\text{PARTS}_{\equiv}$: the domain in $[v_1]$ is unchanged, the domains of $[v_2]$ from $\langle \{(green, \neq, 3)\}, \{(green, \neq, 3)\} \rangle$ become $\langle \{(green, =, 0)\}, \{(green, =, 0)\} \rangle$. We thus recursively fixed the sizes of the first relevant part and now propagate the second. This completes the second recursive call on the two relevant parts: with the (singleton) partition obtained we consider the first recursive call on the relevant part $\overline{green}$.

The following step is to consider the single subproblem obtained from the propagation of the number of *green* entities. The exchangeability classes are unchanged, hence we consider again $\text{PARTS}_{\not\equiv}$: the valid distributions for $\overline{green}$ over the two classes are $\{[0,4], [1,3], [2,2]\}$ because $[4,0]$ and $[3,1]$ would leave one of the two parts empty (thus *distribute* does not return them). We then consider a different partition of the problem for the three valid distributions and propagate each within the exchangeability classes ($\text{PARTS}_{\equiv}$). For each valid $[i, j]$ in the corresponding problem the domain for v_1 is simply updated with $(\overline{green}, =, i)$. For $\langle v_2, v_3 \rangle$ we consider again the valid distributions of j entities over 2 parts.

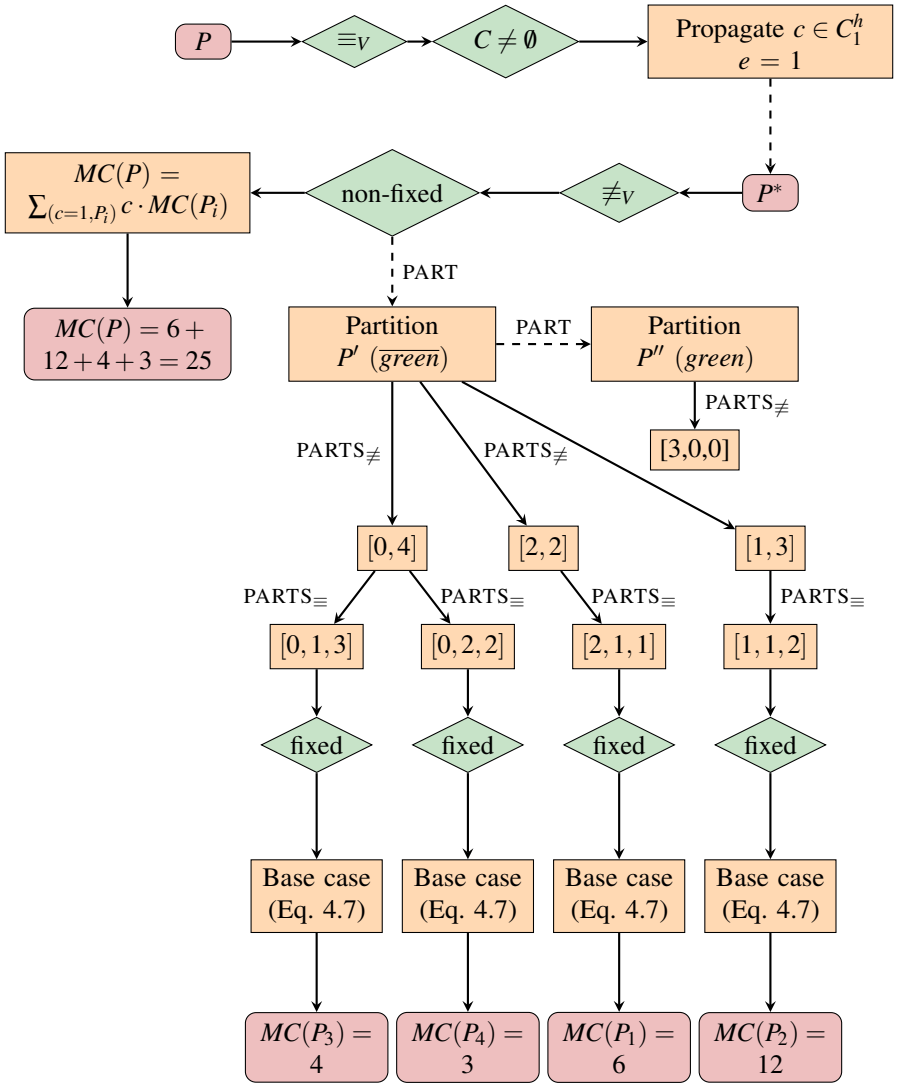


Figure 4.5: Solver execution flow for **P2**

For $j = 2$ the valid distribution is $[1,1]$ ([2,0] violates non-emptiness), similarly, $j = 3$ corresponds to $[1,2]$, while $j = 4$ can be distributed either as $[2,2]$ or $[1,3]$.

Once the distribution is propagated as counting constraints we obtain the following problem partition:

P_1. $\langle\{(green,=,3),(\overline{green},=,2)\},\{(green,=,0),(\overline{green},=,1)\},\{(green,=,0),(\overline{green},=,1)\}\rangle$
Corresponding to: $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\bigstar,\bigstar\},\{\bigstar\},\{\bigstar\}\}$

P_2. $\langle\{(green,=,3),(\overline{green},=,1)\},\{(green,=,0),(\overline{green},=,2)\},\{(green,=,0),(\overline{green},=,1)\}\rangle$
Corresponding to: $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\bigstar\},\{\bigstar\},\{\bigstar,\bigstar\}\}$

P_3. $\langle\{(green,=,3),(\overline{green},=,0)\},\{(green,=,0),(\overline{green},=,3)\},\{(green,=,0),(\overline{green},=,1)\}\rangle$
Corresponding to: $\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\bigstar\},\{\bigstar,\bigstar,\bigstar\}\}$

P_4. $\langle\{(green,=,3),(\overline{green},=,0)\},\{(green,=,0),(\overline{green},=,2)\},\{(green,=,0),(\overline{green},=,2)\}\rangle$
Corresponding to: $\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\bigstar,\bigstar\},\{\bigstar,\bigstar\}\}$

The relevant parts are now fixed for each subset, therefore we can apply the base cases for a lifted count (note that in the first and last problem partitions two subsets are exchangeable):

P_1. $\frac{\binom{4}{2}\cdot\binom{2}{1}}{1\cdot 2} = 6$, Corresponding to:

$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacksquare\},\{\blacksquare\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare,\blacktriangle\},\{\blacksquare\},\{\blacktriangle\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare\},\{\blacktriangle\},\{\blacksquare\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare,\blacksquare\},\{\blacktriangle\},\{\blacktriangle\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare,\blacktriangle\},\{\blacksquare\},\{\blacktriangle\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare\},\{\blacktriangle\},\{\blacksquare\}\}$.

P_2. $\frac{\binom{4}{2}\cdot\binom{3}{2}}{1\cdot 1\cdot 1} = 12$ Corresponding to:

$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacktriangle\},\{\blacksquare,\blacksquare\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacksquare\},\{\blacksquare,\blacktriangle\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacksquare\},\{\blacktriangle,\blacksquare\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare\},\{\blacktriangle\},\{\blacktriangle,\blacksquare\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare\},\{\blacktriangle\},\{\blacksquare,\blacktriangle\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare\},\{\blacksquare\},\{\blacktriangle,\blacktriangle\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacksquare\},\{\blacktriangle,\blacksquare\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacktriangle\},\{\blacksquare,\blacksquare\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacksquare\},\{\blacksquare,\blacktriangle\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare\},\{\blacktriangle\},\{\blacktriangle,\blacksquare\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare\},\{\blacktriangle\},\{\blacksquare,\blacktriangle\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\blacksquare\},\{\blacksquare\},\{\blacktriangle,\blacktriangle\}\}$.

P_3. $\frac{\binom{4}{0}\cdot\binom{4}{3}}{1\cdot 1\cdot 1} = 4$ Corresponding to:

$\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacktriangle\},\{\blacktriangle,\blacksquare,\blacksquare\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacktriangle\},\{\blacktriangle,\blacksquare,\blacksquare\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacksquare\},\{\blacktriangle,\blacksquare,\blacktriangle\}\}$, $\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacksquare\},\{\blacktriangle,\blacksquare,\blacktriangle\}\}$.

P_4. $\frac{\binom{4}{0}\cdot\binom{4}{2}}{1\cdot 2} = 3$ (Example 4.10) Corresponding to:

$\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacktriangle,\blacktriangle\},\{\blacksquare,\blacksquare\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacktriangle,\blacksquare\},\{\blacktriangle,\blacksquare\}\}$,
$\{\{\blacktriangle,\blacktriangle,\blacktriangle\},\{\blacktriangle,\blacksquare\},\{\blacktriangle,\blacksquare\}\}$.

Then, the solution of the problem is the sum over the problem partitions nr. 1,2,3 and 4, hence 25.

4.9 Reasoning: Solver Analysis

In this section we evaluate empirically the contributions of this chapter on a dataset of combinatorics math problems (Dries et al. 2017) and on a set of synthetic benchmarks. We designed a language, CoLa, to express a wide range of combinatorial problems and a solver, CoSo, to efficiently compute their solutions; with the experiments we want to assess:

Q1) how many of these problems can CoSo solve,

Q2) what type of problems is hard for CoSo,

Q3) how CoSo compares to existing methods.

To answer the questions about reasoning we consider two experimental setups for benchmarking: 1) the dataset from Dries et al. (2017) to test CoSo on questions designed to be approachable by humans, and 2) a set of randomly generated CoLa problems which includes more computationally challenging problems. We compare CoSo (**Q3**) with three modelling languages and the corresponding reasoning systems:

- **ASP, Clingo**. Answer Set Programming (Gebser, Kaminski, Kaufmann, and Schaub 2012) is a prominent logic programming framework based on first order logic with counting constraints. Clingo (Gebser, Kaminski, Kaufmann, Ostrowski, et al. 2016) is one of the widely adopted solvers for ASP programs (version 5.4.0).
- **CNF, sharpSAT**. sharpSAT (Thurley 2006) is one of the most prominent propositional counters for satisfiability, which takes as input a propositional logic formula in conjunctive normal form (CNF). The sharpSAT version adopted is 12.08.
- **ESSENCE, Conjure**. ESSENCE (cf. Section 3.1) is translated by Conjure (Frisch, Jefferson, et al. 2005) to the input format of the solver Minion (Gent et al. 2006), a CSP solver. We use ESSENCE 1.3 along with Conjure 2.3.0.

We test CoSo on the benchmarks on a machine equipped with a 4 cores/4 threads CPU and 32 Gb of RAM. Both Clingo and Conjure are called in enumeration mode to count the number of solutions of the problem. We set a timeout of 300 seconds on each problem.

4.9.1 Real-world Problems

Problem decomposition CoSo solves correctly all models (**Q1**): in Figure 4.6 we analyze the running time with respect to the number of subproblems considered by the solver. A subproblem is counted each time the solver is called recursively, therefore when the start node diagram in Figure 4.1 is traversed. Figure 4.6 shows that compared to synthetic benchmarks, real world problems require considering fewer subproblems, and thus can be reduced to the lifted reasoning principles in fewer steps. Of the problems that are not decomposed (1 subproblem, 135 instances) 30 (22

The slowest problem of the real-world dataset, which CoSo decomposes in 87 subproblems, requires 8.49 seconds to be solved:

P11. From three Russians, four Americans, and two Spaniards, how many selections of people can be made, taking at least one of each kind?

P11 belongs to a class of problems that highlights some opportunities for improving the reasoning techniques (**Q2**). In fact, the most complex instances do not specify a single size for the configuration: in Section 4 we explain that we solve a separate #CSP for each valid size. In practice, for some problems it may be possible to exploit

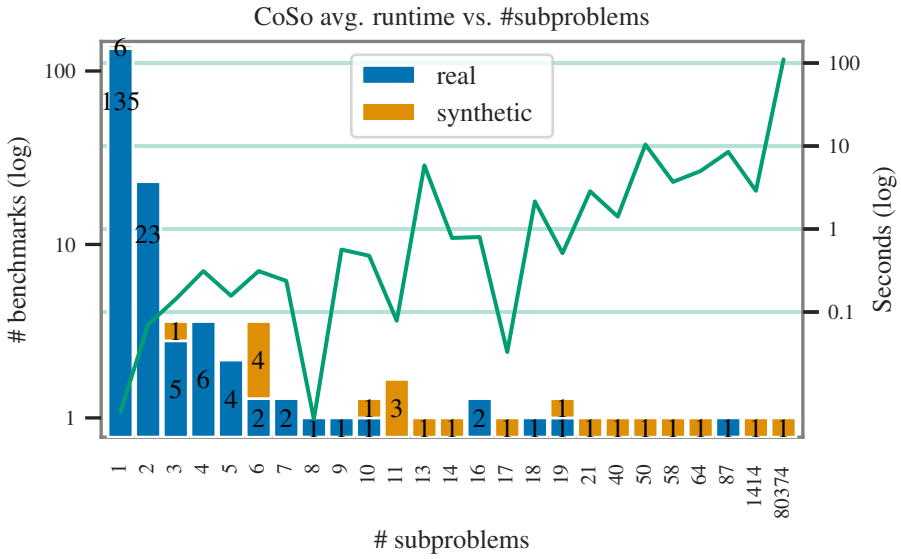


Figure 4.6: CoSo solving time (log scale) tends to increase with the number of subproblems considered. Real-world dataset and synthetic benchmarks combined.

Framework	# unsolved	avg. time (solved)
CoLa-CoSo	0	0.18*s*
ASP-Clingo	52	5.70*s*
CNF-sharpSAT	75	7.44*s*
ESSENCE-Conjure	32	34.99*s*

Table 4.4 : On the 185 problems that can be encoded in CoLa, CoSo outperforms other frameworks in the number of solved problems within the timeout and average runtime.

the work done on the smaller instances to speed up the solving of the bigger ones. Similarly, hard instances contain counting constraints of the kind "at least *n*", which are decomposed, one by one, into many equality constraints corresponding to each valid value (Section 4). Here instead humans are able to satisfy the three constraints at once, by recognizing that they are independent because the sets of Russians, Americans and Spaniards are disjoint and by "building" the valid subsets already including one person of each type.

We automatically translated the CoLa models to ASP and Essence to compare the propositional solvers with CoSo on the real-world examples. The number of problems that reach the 300 seconds timeout for each solver is presented in Table 4.4, along with the average running time on the remaining problem for which the solution could be computed. Even on problems designed to be solvable by humans, the frameworks based on propositional reasoning struggle to compute the answer within the time limit on many instances.

Growing domains Given a combinatorics math problem, we expect that increasing the number of entities involved does not affect the performances of CoSo because of its lifted reasoning techniques. On the other hand we expect a degradation of the performances of propositional methods because of the exponentially larger number of combinations at the propositional level. To verify this, we consider our three real-world running examples, **P3**, **P4** and **P5**, and test the solvers on problems of increasing size as follows. For **P3**, starting from the original problem with 12 TVs of which 3 are defective (Problem 0), we add at each step i one defective TV and one working TV for $i \in \{1,\dots,10\}$, thus reaching 32 TVs of which 13 are defective in Problem 10. For **P4** we add a worker at each step i for $i \in \{1,\dots,10\}$ and increase by one the size of partition i mod 3, thus reaching in Problem 10 24 workers partitioned in groups of sizes $\{11,8,5\}$. For **P5** we maintain the number of objects constant (twice the original: 2 Bs, 6 As, 4 Ns) and increase the size of a word, starting from 2 and increasing it up to 12 over the 10 Problems.

The results, summarized in Figure 4.7, confirm our expectations. All problems are solved by CoSo in less than a second with constant time with respect to the increase

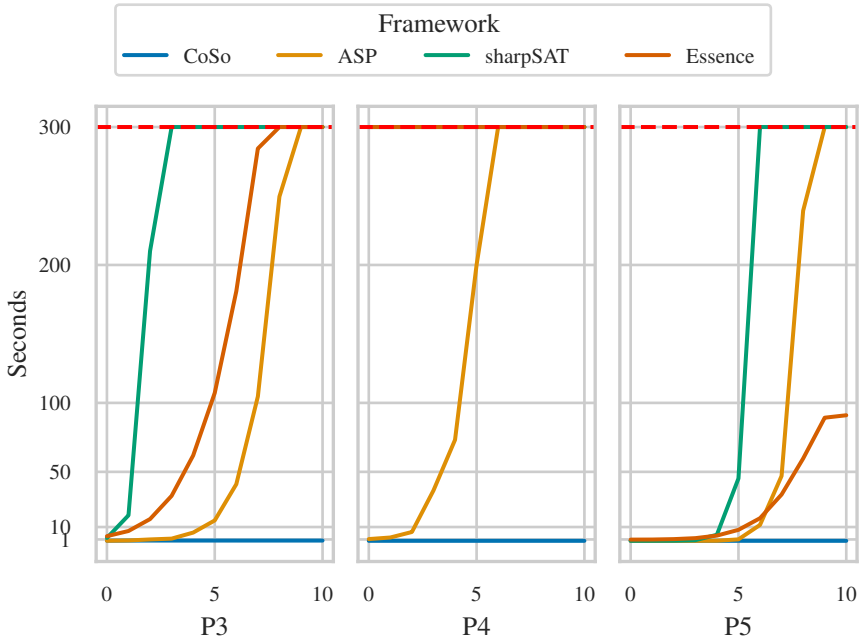


Figure 4.7: The solving time of propositional methods increases exponentially with the number of objects or the size of the configuration.

of the number of entities. On the contrary, the running time of propositional methods increases exponentially in the size of the universe or, in the case of **P5**, the size of the configuration. We note that for **P5** the number of solutions between Problem 9 and Problem 10 does not change, hence in this case there is no exponential increase in the running time of ESSENCE-Conjure between the two. Here the couple ESSENCE-Conjure scales better than ASP-Clingo or CNF-sharpSAT, while in **P4** ASP-Clingo is the only combination that does not time out already from the original problem formulation (Problem 0).

4.9.2 Synthetic Benchmarks

In order to test CoSo and propositional methods on computationally harder benchmarks we generate a set of random CoLa problems as follows. We consider all types of configuration except partitions as we argued (Section 3.1) that not all frameworks support them. For each configuration type we consider a size of the universe of u objects with $u \in \{10, 15, 20\}$, and a configuration size of reference $s \in \{5, 10, 15\}$. We

generate a random multiset by adding a new object and a random number of copies until the target size u is reached. We complete the universe by selecting a random number of subsets representing their properties. We randomly choose a comparison symbol in $\{=, \neq, \geq, \leq, >, <\}$ to set the size of the configuration w.r.t. s. A random number of positional constraints (if applicable to the configuration type) and counting constraints are generated similarly, by choosing a random number in $\{0, \ldots, s\}$ for the position or the required count, and a random subset to require at the position or to be counted. Comparison symbols in counting constraints are random as well.

Of the 30 synthetic benchmarks in three cases none of the solvers finished within the timeout. We report one of them (permutation problem) in the plots as it was possible to compute the solution and the number of subproblems by running CoSo for a reasonable amount of time beyond the time limit. The other two are composition problems, which in general are difficult for CoSo (**Q2**) when the multiset presents

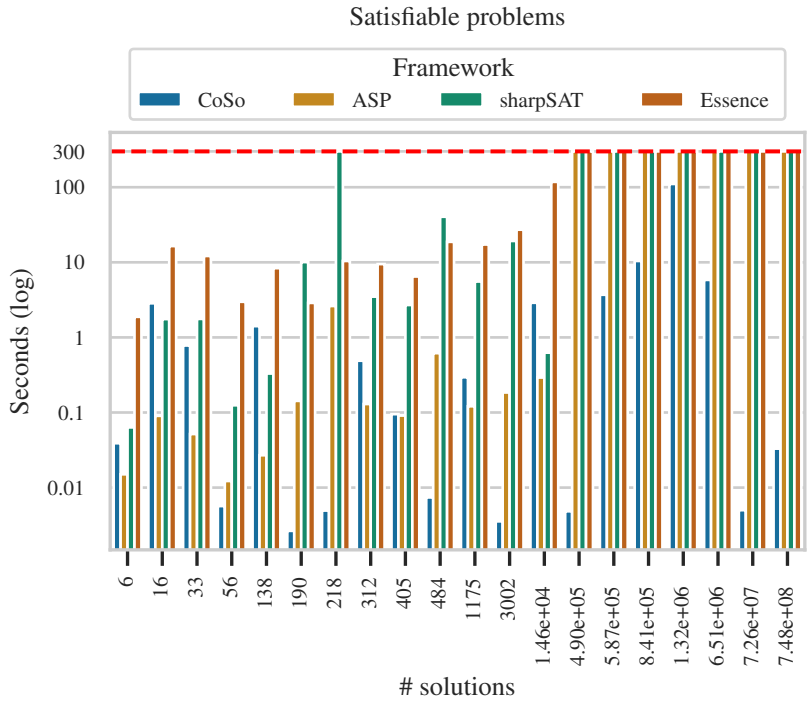


Figure 4.8: Solving times (log scale) on synthetic benchmarks: CoSo outperforms propositional frameworks in most cases, in particular with the increase of the number of solutions.

many entities representing few copies: the many entities usually require to consider a large number of cases for distributing entities into the parts and the few copies of each do not offer an advantage in terms of reasoning over set sizes. This situation thus results in a solving procedure similar to enumeration. The results of the benchmarks are summarized in three plots.

Figure 4.6 relates the running time of CoSo to the number of subproblem considered. We combine the two datasets of problems into the figure: the majority of the problems from the real-world dataset contribute to the instances that can be solved by considering a small amount of splits, while the synthetic benchmarks populate the bars on the right side of the plot. The plot shows a mild correlation between the number of subproblems in which the instance is partitioned and the total time required to compute the solution.

Figure 4.8 contains the running time for the satisfiable problems, ordered on the x axis by the number of solutions. Here we observe how the performances of the

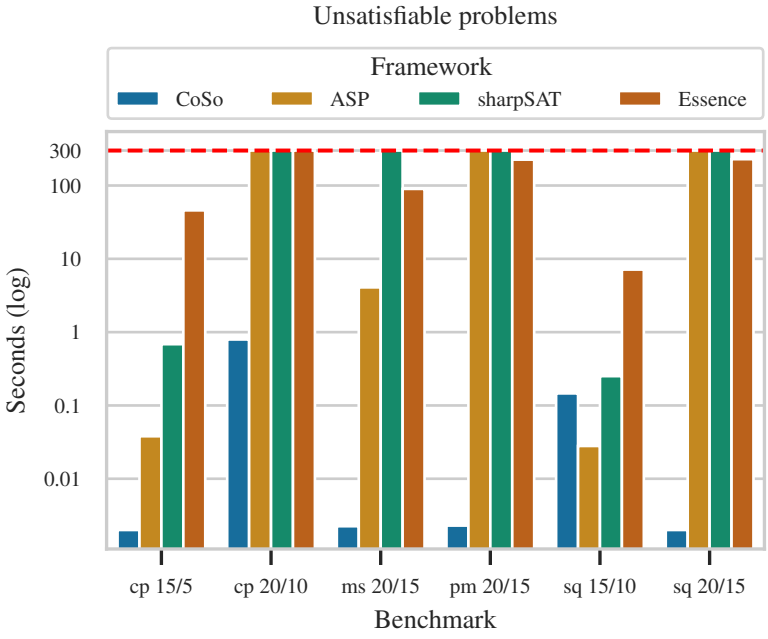


Figure 4.9: On unsatisfiable configurations CoSo outperforms propositional frameworks on most of the instances (log scale). For ASP and sharpSAT most of these instances have the largest groundings of the dataset. Legend format: *configuration type universe size/configuration size of reference*. *ms=multiset, pm=permutation, sq=sequence, cp=composition*.

propositional methods degrades quickly with the increase of the number of solutions to be enumerated, while CoSo's runtime is unrelated to it.

In Figure 4.9 we report the running time for the unsatisfiable problems. These examples confirm the trend of CoSo being faster than propositional methods. At the same time these experiments show that the efficiency of propositional methods is influenced not only by the number of solutions to be enumerated, but also in the case of Clingo and sharpSAT by the size of the ground program, which needs to be computed before constraints are analyzed. In particular, the problems where ASP and sharpSAT time out are ordered configurations, hence requiring considering all possible permutations of values, of size ≥ 10 and universe with cardinality ≥ 15.

4.10 Conclusion

In this chapter we addressed our research question **Q2** by developing lifted reasoning techniques for counting CSPs, and by implementing them in a solver, CoSo, for problems modelled with CoLa. We compared the solving performances of CoSo with propositional frameworks based on first order logic or constraint programming, evaluating them on a real-world dataset of combinatorics math problems and sythetic benchmarks. The experiments show that CoSo outperforms the traditional counting techniques and scales effectively when the number of objects increases yielding a larger number of solutions.

Part II

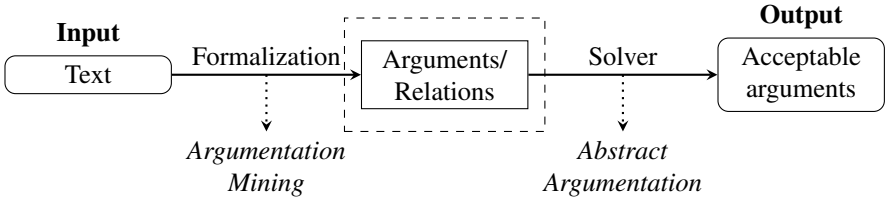
Argumentation under Uncertainty

Chapter 5

Probabilistic Argumentation Problems: Modelling

This chapter is largely based on the paper submitted for publication:

P. Totis, A. Kimmig, and L. De Raedt (2023). "SMProbLog: Stable Model Semantics in ProbLog and its Applications in Argumentation". In: *Theory and Practice of Logic Programming*



Designing systems that are able to argue and persuade is a very relevant and challenging task in artificial intelligence. In real-world scenarios the information available is often incomplete or questionable, therefore the ability to take into account uncertainty is fundamental in such situations. Probabilistic logic programming (PLP) frameworks and languages, such as PRISM (Sato and Kameya 2008), ICL (Poole 2008), ProbLog (De Raedt et al. 2007) or LPAD/CP-logic (Vennekens et al. 2009), are designed to provide powerful general-purpose tools for modelling and reasoning about structured, uncertain

domains. On the other hand, abstract argumentation frameworks (Dung 1995) aim specifically at describing argumentation processes and reasoning about the set of acceptable arguments. For instance, consider again Example 1.3, an argumentative microtext adapted from (Stede, Afantenos, et al. 2016):

Example 5.1. Yes, it's annoying and cumbersome to separate your rubbish properly all the time (a_1), but small gestures become natural by daily repetition (a_2). Three different bin bags stink away in the kitchen and have to be sorted into different wheelie bins (a_3). But still Germany produces way too much rubbish (a_4) and too many resources are lost when what actually should be separated and recycled is burnt (a_5). We Berliners should take the chance and become pioneers in waste separation! (a_6)

An argumentation problem defines arguments, e.g. $a_1, a_2, \ldots$ and their relations, for example a_1 attacks a_2, a_3 supports $a_1 \ldots$ from which a set of acceptable arguments is derived according to a given semantics.

When uncertainty is part of the argumentation problem, we want to reason about it in order to answer queries like "What is the likelihood of accepting argument a_6?". Inferring the probability of one variable taking a particular value is a typical PLP task. PLP frameworks offer a wide range of tools and algorithms for probabilistic inference and learning on probabilistic logic programs. Probabilistic argumentation systems, on the other hand, propose different combinations of argumentation frameworks, probability interpretations and reasoning systems, tailored to manipulating probabilities in argumentation. When considering PLP and argumentation systems, the question is natural: can PLP effectively model argumentation processes and reason over its intrinsic uncertainty?

This question has been only partially investigated so far: of the two main interpretations of probabilities in argument graphs (Hunter 2013), namely the *constellations* approach and the *epistemic* approach, only the former has been studied in the context of PLP (Mantadelis and Bistarelli 2020). This chapter fills this gap and shows how PLP can be used to reason about a probabilistic argumentation graph with an epistemic view of its probabilities. Table 5.1 summarizes the differences in the approaches.

In the constellations approach, probabilities describe a probability distribution over argument (sub)graphs. Such a distribution is used to define the probability of an

Framework	Epistemic	PLP	EM Learning
MetaProblog (Mantadelis and Bistarelli 2020)	✗	✓	✗
Epistemic graphs (Hunter, Polberg, et al. 2020)	✓	✗	✗
SMProbLog	✓	✓	✓

Table 5.1 : Argumentation frameworks overview.

argument being in the set of accepted arguments according to the argumentation semantics of choice. On the contrary, in the epistemic approach probabilities are a direct measure of the belief of an agent in an argument, and therefore of its acceptability. Previous work on probabilistic argument graphs under the epistemic approach focuses on modelling a family of probability distributions that are compatible with the graph and given constraints (Hunter, Polberg, et al. 2020).

Instead of considering a probabilistic argument graph as a model for a family of probability distributions over fixed beliefs that have to satisfy constraints, we regard a probabilistic argument graph as a model for a single joint probability distribution over arguments. We view nodes as random variables associated to the arguments, whose prior is the epistemic arbitrary assignment of probabilities to arguments. We then interpret their relations as the conditional (in)dependencies influencing the posterior joint probability (belief) distribution of the random variables. Given a set of independent beliefs regarding arguments and relations, modelled by means of PLP, we thus define the joint probability (belief) of the arguments by means of probabilistic inference.

The joint distribution defines *marginal and conditional* beliefs that are coherent with the given priors and argumentative (logic) structure. The advantage of this method is that the joint belief distribution allows us to answer a broad range of questions that are of interest in the argumentation setting. For example: the marginal probability of (belief in) an argument ("What is the belief in a_1 considering the influence of the other arguments?"), relating the beliefs of two arguments ("How does my belief in a_1 change if a_5 is accepted?"), or perform typical PLP tasks such as learning the beliefs of an agent given a set of observations of accepted arguments.

5.1 Modelling Probabilistic Argumentation with PLP

Let us now answer the question as to whether PLP can effectively model argumentation processes and reason over their intrinsic uncertainty. Our answer takes the form of encoding argumentation problems and their uncertainty in probabilistic logic programs. By encoding arguments and uncertainty using PLP, we embed argumentation in probabilistic approaches to automated reasoning under uncertainty. More specifically, probabilistic logic programs represent the joint probability distribution. Modelling the joint probability distribution is central to the success of probabilistic graphical models (Koller and Friedman 2011), which are the most popular tools for learning and reasoning about uncertainty in AI. Naively specifying joint probability distribution however requires a number of parameters exponential in the number of random variables (Charniak 1991; Pearl 1989). The success of probabilistic graphical models and Bayesian Networks (Pearl 1989) is based on the use of a graphical structure that

represents a set of conditional independence assumptions, which allow to compactly encode the joint probability distributions in a factorized form. In Bayesian Networks, for instance, the key assumption is that each variable is conditionally independent of its non-descendants, given its parents. Two random variables X and Y are said to be *conditionally independent given* Z, a third variable, if in a joint probability distribution P including the three variables $P(X|Y,Z) = P(X|Z)$ whenever $P(Y,Z) > 0$, that is, Y does not provide extra information X, if we already know Z. In argumentation, conditional probabilities such as $P(X|Z)$ can be used to update the belief in X in the light of the argument Z, while conditional independencies can be used to reason about the (ir)relevance of a certain argument Y in the context of $P(X|Z)$.

By approaching probabilistic argumentation problems from a PLP perspective, we apply the principles in modelling uncertainty in AI formulated by Pearl (1989):

> We will also stress that probability theory is unique in its ability to process context-sensitive beliefs, and what makes the processing computationally feasible is that the information needed for specifying context dependencies can be represented by graphs and manipulated by local propagation.

This suggests an interpretation of the argument graph as a probabilistic graphical model, that is, a graph modelling the conditional independencies between random variables (arguments). However, Bayesian Networks are directed acyclic graphs, and with argument graphs we are concerned with more complex relations involving cycles and negations (attacks). This motivates the need for a richer framework, PLP, and semantics (Chapter 6) to derive the joint probability distribution. The argument graph thus expresses how conditional independencies and context dependencies shape a joint belief distribution across the arguments. We thus follow the epistemic interpretation of probabilities where probabilities measure the degree of belief in the arguments. As Hunter (2013) remarks, the emphasis in the epistemic approach is to find a probability distribution that is rational with respect to the argument graph. With PLP, the derivation of the joint probability distribution from the argument graph mirrors the process of evaluating personal beliefs described by Nilsson (2014):

> If we were to examine the relationships among all of our beliefs carefully, an impossible task in practice but one that is interesting to think about, we would see that some of them should make others more credible and some less. They would even compete among themselves with conflicting influences. We can imagine all of the beliefs in our large network of beliefs "fighting it out" to agree finally on the strength of each belief in the network. When they do finally agree, we say that the beliefs *cohere*.

We begin by defining the type of models that describe a probabilistic argumentation problem, that is, we encode the problem in a probabilistic logic program. Following the epistemic interpretation of probabilities we map arguments to logical atoms whose probability (belief) can be queried in the traditional PLP style, i.e. querying marginal and conditional probabilities (beliefs) given evidence. Therefore, we do not define an additional layer of logic rules specifying argumentation semantics over a (probabilistic) graph, as in the constellations approach. Mapping a probabilistic argument graph to a probabilistic logic program means defining its semantics, because the PLP semantics of choice for the program in turn defines the graph's semantics. For this reason, we first discuss the informal semantics of the mapping, and then characterize its formal semantics by means of PLP (Chapter 6).

Mapping. Given a probabilistic argument graph $G = (A, R^-, P_A, P_{R^-})$, we associate each argument $a \in A$ to a random variable $arg(a)$ representing the belief in a. This belief is described by means of the two probability functions: P_A and P_{R^-}. We interpret $P_A(a)$ as the prior belief for a, that is, a bias independent of the other arguments, similar to a base score in a QBAF. The independence assumption of the prior beliefs in the arguments does not mean that arguments are assumed to be independent, but only the degree to which the agent is biased towards them. The *posterior* belief in an argument is determined by a combination of the bias and the attacks received. We interpret an attack $P_{R^-}(a,b)$ as the belief that accepting a causes the rejection of b. Since we measure beliefs by means of random variables, the natural interpretation of attacks is that the belief in the target is (negatively) conditioned by the belief in the attacker, therefore believing in a causes a decrease in the belief of b. Therefore, the belief in an argument depends on the belief in the attacking arguments: a strong belief in any of the attackers can cause the agent to not believe the attacked argument. In Chapter 2 we showed how in PLP such causal relation is expressed by means of the *inhibition effect*, therefore we model attacks with negation in the heads.

Definition 29. *Given a probabilistic argument graph $G = (A, R^-, P_A, P_{R^-})$ the corresponding probabilistic logic program $\mathscr{L}$ is a program where:*

- *for each $a \in A$ and $P_A(a) = p$:*
  - $p :: bias(a). \in \mathscr{L}$
  - $arg(a) \leftarrow bias(a). \in \mathscr{L}$
- *for each $(a,b) \in R^-$ and $P_{R^-}((a,b)) = p$:*
  - $p :: \neg arg(b) \leftarrow arg(a). \in \mathscr{L}$

Example 5.2. Consider the example in Figure 2.3: the corresponding program is:

```
0.6::bias(a1).  0.7::bias(a2).
0.4::bias(a3).  0.2::bias(a4).
arg(a1) ← bias(a1).  arg(a2) ← bias(a2).
arg(a3) ← bias(a3).  arg(a4) ← bias(a4).
0.9::¬arg(a1) ← arg(a3).  0.8::¬arg(a2) ← arg(a4).
0.4::¬arg(a1) ← arg(a2).  0.7::¬arg(a2) ← arg(a1).
```

A rule $\neg arg(b) \leftarrow arg(a)$ therefore says that believing argument a causes not believing in b. In Chapter 2 we showed how the negation in the head is interpreted in PLP by rewriting the rule as: $arg_{neg}(b) \leftarrow arg(a).\ arg(b) \leftarrow arg_{pos}(b), \sim arg_{neg}(b).$ expresses "believing a causes believing in a counterargument for b and b cannot be believed when a counterargument is believed". The probability $p = P_{R^-}((a,b))$ measures the belief in the effectiveness of the attack, that is, how likely it is to conclude that argument b is rejected given that a is believed.

Example 5.3. The rules in Example 5.2 are internally rewritten as follows.

```
0.6::bias(a1).    0.7::bias(a2).
0.4::bias(a3).    0.2::bias(a4).
arg_pos(a1) ← bias(a1).  arg_pos(a2) ← bias(a2).
arg_pos(a3) ← bias(a3).  arg_pos(a4) ← bias(a4).
0.9::f1.  0.4::f2
0.8::f3.  0.7::f4
arg(a1) ← arg_pos(a1),∼arg_neg(a1).  arg_neg(a1) ← arg(a3), f1.
arg(a2) ← arg_pos(a2),∼arg_neg(a2).  arg_neg(a1) ← arg(a2), f2.
arg(a3) ← arg_pos(a3),∼arg_neg(a3).  arg_neg(a2) ← arg(a4), f3.
arg(a4) ← arg_pos(a4),∼arg_neg(a4).  arg_neg(a2) ← arg(a1), f4.
```

We started by considering basic probabilistic argument graphs, but PLP can encode much more complex relations over random variables, hence we exploit the flexibility and expressivity of PLP to effortlessly extend this modelling technique to incorporate many relevant aspects of argumentation in a framework capable of encoding:

- Quantitative evaluations of relations.
- Support relations.
- Attack (and supports) from sets of arguments.
- Distinctions between proponents of arguments.

We showed in Example 5.2 how annotated rules express a fine-grained quantification of how strong the causal relation between the arguments (random variables) is.

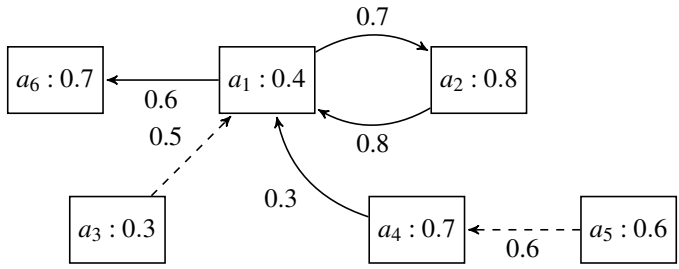


Figure 5.1: Example 5.4 representation. Dashed (resp. solid) edges represent supports (resp. attacks) R^+ (resp. R^-). Nodes (resp. edges) are labelled with the corresponding bias (belief).

Support. Support relations are symmetric to attacks: we consider the (bipolar) extension of a probabilistic argument graph with a relation support R^+ ($R^+ \subseteq A \times A, R^+ \cap R = \emptyset$) and the corresponding probability $P_{R^+} : R^+ \to [0,1]$. Believing in a supporter a causes believing in b with probability p: this corresponds to an increase of the belief in the supported argument b. This relation is encoded as a simple inference rule:

Definition 30. *Given a bipolar probabilistic argument graph* $G = (A, R^-, R^+, P_A, P_{R^-}, P_{R^+})$ *the corresponding probabilistic logic program* $\mathscr{L}$ *is a program where:*

- *for each* $a \in A$ *and* $P_A(a) = p$*:*
  - $p :: bias(a). \in \mathscr{L}$
  - $arg(a) \leftarrow bias(a). \in \mathscr{L}$
- *for each* $(a,b) \in R^-$ *and* $P_{R^-}((a,b)) = p$*:*
  - $p :: \neg arg(b) \leftarrow arg(a). \in \mathscr{L}$
- *for each* $(a,b) \in R^+$ *and* $P_{R^+}((a,b)) = p$*:*
  - $p :: arg(b) \leftarrow arg(a). \in \mathscr{L}$

We have thus introduced all the elements required to describe Example 5.1:

Example 5.4. Consider Example 5.1: we model $G = (A, R^-, R^+, P_{R^-}, P_{R^+})$ (see Figure 5.1) with arguments $a_i \in A$, attacks $R = \{(a_1,a_6),(a_4,a_1)\},(a_1,a_2),(a_2,a_1)\}$, and supports $S = \{(a_5,a_4),(a_3,a_1)\}$. We also define the following probability functions to derive an example of a probabilistic argumentation problem: $P_A(a_1) = 0.4$,

$P_A(a_2) = 0.8$, $P_A(a_3) = 0.3$, $P_A(a_4) = 0.7$, $P_A(a_5) = 0.6$, $P_A(a_6) = 0.7$, $P_{R^-}(a_1,a_6) = 0.6$, $P_{R^-}(a_2,a_1) = 0.8$, $P_{R^-}(a_1,a_2) = 0.7$, $P_{R^-}(a_4,a_1) = 0.3$, $P_{R^+}(a_5,a_4) = 0.6$, $P_{R^+}(a_3,a_1) = 0.5$. Which we can thus encode as:

Example 5.5. Example 5.4 is encoded in the following probabilistic logic program:

```
0.4::bias(a1). 0.8::bias(a2).
0.3::bias(a3). 0.7::bias(a4).
0.6::bias(a5). 0.7::bias(a6).
arg(A) ← bias(A).
0.6::¬arg(a6) ← arg(a1). 0.6::arg(a4) ← arg(a5).
0.3::¬arg(a1) ← arg(a4). 0.5::arg(a_1) ← arg(a3).
0.8::¬arg(a1) ← arg(a2). 0.7::¬arg(a2) ← arg(a1).
```

Sets. In this context, it is immediate to generalize relations between pairs of arguments to relations involving multiple arguments. This is the case of argument systems (Nielsen and Parsons 2006), where the attack between two arguments is generalized to an attack relation from a set of arguments towards a single argument (set-attacks). We can model this in PLP by conjoining attackers (resp. supporters) in the body of rules.

Example 5.6. $\neg arg(a_1) \leftarrow arg(a_4), arg(a_5)$. is a causal relation effective only when all the arguments in the set of attackers (body) are believed. Also in this case we can easily combine different argumentative features with each other, for example:

Example 5.7. Examples of gradual joint relations in the setting of Example 5.1:

- gradual set attacks: $0.6 :: \neg arg(a_1) \leftarrow arg(a_4), arg(a_5)$.
- gradual set supports: $0.3 :: arg(a_3) \leftarrow arg(a_4), arg(a_5)$.

PLP tools. Note that in Example 5.5 we use first-order predicates to compactly define the relation between arguments and their bias. This is a modelling advantage offered by PLP, whose flexibility can also be exploited to model less standard scenarios. For instance, in the following example we distinguish between different proponents of the arguments and define the bias in terms of a measure of the trust in them:

Example 5.8. Consider Example 5.5 and the distinction between arguments coming from a proponent ($\{a_2, a_4, a_5, a_6\}$) and those introduced by an opponent ($\{a_1, a_3\}$). We can model bias based on the source of the argument as follows:

```
0.4::proponent.  0.7::opponent.
arg(a1) ← opponent.  arg(a2) ← proponent.
arg(a3) ← opponent.  arg(a4) ← proponent.
arg(a5) ← proponent.  arg(a6) ← proponent.
```

First order logic allows for example expressing compactly that the more agents propose the same argument, the higher its bias is. Assume we express proponents and trust as $p_1 :: prop(1). \cdots p_n :: prop(n).$ and $proposes(i,a)$ denotes that argument a is backed by proponent i, then a rule $bias(A) \leftarrow proposes(P,A), prop(P).$ defines a contribution to the bias of each argument from all its proponents, denoted by each belief p_i.

We thus showed how PLP offers an expressive language that is capable to encode complex relations between arguments and their beliefs. Ideally, we would like to be able to perform the inference and learning tasks over such programs, and apply the traditional PLP algorithms and tools to models of argumentation problems. This means answering marginal probability queries of the kind: "what is the inferred belief in an argument given its bias and the relationships with the other arguments?" or answering conditional queries by reasoning over evidence, i.e. "how does the belief in an argument change if the validity or falsehood of a set of arguments is determined?". The natural question arising at this point is whether the syntax is paired with semantics and reasoning systems that support this type of models. Unfortunately, the answer is negative: the assumptions that PLP semantics impose on the input program are too restrictive for a broad class of argumentation problems. In particular, the presence of cycles through negation in a probabilistic program $\mathscr{L} = F \cup R$ usually results in a violation of the requirement of the existence of a single two-valued well-founded model for all the possible (deterministic) logic programs $\omega \cup R$, $\omega \subseteq F$. This situation is often present in the case of reciprocal (or, in general, cyclic) attacks in argument graphs. Example 5.9 shows how arguments attacking each other results in a cyclic relation where negation is involved:

Example 5.9. In Example 5.5 rules $0.8 :: \neg arg(a_1) \leftarrow arg(a_2).$ and $0.7 :: \neg arg(a_2) \leftarrow arg(a_1).$ correspond to the following rewriting of the negated heads:

```
0.7::f.  0.8::g.
arg_neg(a2) ← f, arg(a1).
arg_neg(a1) ← g, arg(a2).
arg(a1) ← arg_pos(a1), ∼arg_neg(a1).
arg(a2) ← arg_pos(a2), ∼arg_neg(a2).
```

The negation in the head thus leads to a cyclic dependency, we use $\xrightarrow{\sim}$ to denote a dependency through negation, $\rightarrow$ otherwise:

$$arg(a_1) \xrightarrow{\sim} arg_{neg}(a_1) \rightarrow arg(a_2) \xrightarrow{\sim} arg_{neg}(a_2) \rightarrow arg(a_1)$$

which causes the program to not satisfy the requirement of a unique two-valued well-founded model in the possible worlds where f, g, and the probabilistic facts corresponding to the supports of the two arguments are included.

In the Chapter 6 we discuss why such programs are not captured by traditional PLP semantics, and we propose a new semantics that, on the contrary, allows us to reason over this particular class of programs.

5.2 Related Work

Our approach is novel as previous work considers different semantics and reasoning techniques for probabilistic argument graphs. Table 5.2 summarizes these differences: the choice between interpreting probabilities as in the constellation approach or as in the epistemic approach, approaching the problem from PLP modelling a single joint probability distribution as in Bayesian networks, whether it can reason about marginal and conditional queries about the probability of arguments, and the implementation in a framework where the probabilities of the graph can be learned. Moreover, our framework is the only one providing an implementation that can learn from observations (of accepted arguments).

Framework	Epistemic	PLP	Conditionals & Marginals	Implementation
QBAF (Baroni et al. 2019)	✗	✗	✗	✗
Prob. argument graphs (Li et al. 2011)	✗	✗	✗	✓
MetaProblog (Mantadelis and Bistarelli 2020)	✗	✓	✗	✓
Epistemic graphs (Hunter, Polberg, et al. 2020)	✓	✗	✗	✓
L-Credal (Rocha and Cozman 2022)	✓	✓	$\sim$	✗
SMProbLog	✓	✓	✓	✓

Table 5.2 : Argumentation frameworks comparison. $\sim$ denotes partial support

Epistemic. We follow the epistemic interpretation of probabilities (Hunter 2013; Hunter, Polberg, et al. 2020) as a direct measure of the belief of an agent in arguments. Section 6.2 already discussed the main differences with previous work. We use fine-grained (gradual) evaluations of relations and base scores for arguments in a bipolar setting similarly to QBAFs (Baroni et al. 2019). Similarly to our framework, in a QBAF a set of arguments contains both an attack and support relation, and defines a base score τ, which corresponds in our case to the prior probability (belief) P_A, as Baroni et al. (2019) describes it: "... the nature and meaning of the base score [...] corresponds to an assessment of arguments which precedes the consideration of the relations of attack and support with other arguments". In our case the joint probability distribution is the result of this consideration. QBAF is a restricted setting compared to our framework since we also define a score (following the terminology of Baroni et al. (2019)) for the relations R^+ and R^-, besides considering set-attacks. Most importantly, we do this with probabilistic semantics.

PLP. At the same time, we propose a mapping from probabilistic argument graphs to probabilistic logic programs, which provides a novel semantics for probabilistic argument graphs. We consider a single probability distribution in a Bayesian style rather than families of distributions. Previous work in fact focused on reasoning about the properties of families of probability distributions that are consistent with the argument graph and additional constraints (*epistemic graphs*) (Hunter, Polberg, et al. 2020).

Despite following the epistemic approach, our framework has similarities with the constellations approach. The definition of the distribution over subgraphs in the constellations approach is similar to the distribution over subprograms $F \cup R$ of distribution semantics. In fact, the distribution modelled by the probabilistic logic program is determined by the probability that an argument is included in a possible world, similar to the inclusion or exclusion of nodes and edges in the subgraphs considered in the constellations approach. However, in order to determine whether an argument is true or not, we rely on the stable model semantics for logic programs. In the constellations approach, on the other hand, the admissible arguments are defined by the classical extension-based semantics (Dung 1995). Stable models are equivalent to stable extensions for basic arguments graphs (Dung 1995). We consider however more sophisticated relations between arguments that cannot be encoded in a traditional (probabilistic) abstract argumentation framework, e.g. support (Amgoud, Cayrol, et al. 2004) or set-attacks (Nielsen and Parsons 2006). For this reason, our approach does not fit any of the previous categorizations about the connections between logic (programming) and argumentation (Besnard, Cayrol, et al. 2020).

Modelling (deterministic) argumentation problems by means of logic programming dates back to the foundational work of Dung (1995). Dung (1995) proposes a method to define meta-interpreters for argument systems encoding the extension-based semantics

as logic rules. A change in argumentation semantics thus requires to encode in the meta-interpreter a new reasoning technique. This is the approach taken in Mantadelis and Bistarelli (2020), while we directly compute the stable models semantics for logic programs on a mapping from argument graphs to probabilistic logic programs.

Wu et al. (2009) propose a mapping for Dung's abstract argumentation frameworks to logic programs, therefore they do not address the probabilistic settings as well as other extensions to the original framework considered in this chapter. Wu et al. (2009) focus on complete labellings and propose a translation where given an argument graph (A,R), given a set of attacks of the form $(b_i,a) \in (A,R)$, $i \in \{1,\ldots,n\}$ a logic rule of the form $a \leftarrow \sim b_1,\ldots,\sim b_n$. is added to the logic program translation. This is the same principle followed in our encoding, where attacks (b_i,a) with $P_R((b_i,a)) = p_i$ are translated to rules of the form $p_i :: \neg a \leftarrow b_i$. In fact, the rewriting of negation in the head produces rules $a \leftarrow a_{pos}, \sim a_{neg}$ plus $p_i :: a_{neg} \leftarrow b_i$. Therefore, in both cases all attackers b_i must be false in order to consider a acceptable. However, there is a difference between negation in the head (and the corresponding translation) and negation in the body. Attacks directly encoded with negation in the body do not exclude that the two choices can be true at the same time because of external justification, but exclude that both are false. Vice versa, an encoding with negation in the head excludes that both choices are true at the same time, but not that both can be false (e.g. in Example 6.8).

Example 5.10. Consider the difference between a program with a loop with negation in the body (A) and one with negation in the head (B).

Program A:

```
0.6::too_expensive_X.
0.7::too_noisy_Y.
stay_at_Y ← too_expensive_X.
stay_at_X ← too_noisy_Y.
¬stay_at_X ← stay_at_Y.
¬stay_at_Y ← stay_at_X.
```

Program B:

```
0.6::too_expensive_X.
0.7::too_noisy_Y.
stay_at_Y ← too_expensive_X.
stay_at_X ← too_noisy_Y.
stay_at_X ←∼stay_at_Y.
stay_at_Y ←∼stay_at_X.
```

Program A is the program of Example 6.8. Program B is a similar encoding with negation in the body. In this case the possible worlds ω_2 and ω_3 have the same models of A. However, for $\omega_1 = \{\}$ where program A has an empty model, program B has the models $\{\{stay_at_Y\},\{stay_at_X\}\}$. While program B forces a choice when the counterarguments are not accepted, program A allows no choice. At the same time, program B allows on ω_4 to infer a stay at both hotels from the counterarguments for each, that is, the model $\{too_expensive_X, too_noisy_Y, stay_at_X, stay_at_Y\}$. Table 5.3 compares the models for each total choice.

Past work on the connections with logic programming beyond basic abstract argumentation frameworks are limited to the deterministic setting, except for the work by Rocha and Cozman (2022). They adopt the translation by Wu et al.

Total choice	Model A	Model B
$\omega_1 = \{\}$	$\{\}$	$\{\{stay_at_Y\}, \{stay_at_X\}\}$
$\omega_2 = \{too_expensive_X\}$	$\{\omega_2 \cup \{stay_at_Y\}\}$	$\{\omega_2 \cup \{stay_at_Y\}\}$
$\omega_3 = \{too_noisy_Y\}$	$\{\omega_3 \cup \{stay_at_X\}\}$	$\{\omega_3 \cup \{stay_at_X\}\}$
$\omega_4 = \{too_expensive_X, too_noisy_Y\}$	$\{\omega_4 \cup \{stay_at_Y\}, \omega_4 \cup \{stay_at_X\}\}$	$\{\omega_4 \cup \{stay_at_Y, stay_at_X\}\}$

Table 5.3 : Possible models comparison of Example 5.10

(2009) to encode Assumption Based Argumentation Frameworks (Bondarenko et al. 1993) in a probabilistic logic program. Caminada et al. (2015) explore equivalence relations for more expressive deterministic argumentation frameworks, by framing logic programming semantics in terms of argumentation semantics. Alfano et al. (2020) consider the opposite approach and follow in a *deterministic* setting a similar approach to ours for *probabilistic* argumentation. In fact, a simple but general logical framework is shown to be able to capture, in a systematic and succinct way, the different features of several argumentation frameworks under different argumentation semantics. The authors remark how the flexibility of a logic programming approach encourages the study of more extensions and can be used for better understanding the semantics of extended argumentation frameworks.

Marginals and conditionals. With our methodology it is possible to apply general PLP reasoning techniques to probabilistic argument graphs, such as marginal probability computation (Example 6.12), conditioning over evidence (Example 6.13), and parameter learning (Section 6.3.2). This results in a modular, expressive, extensible framework reflecting the dynamic nature of beliefs in an argumentation process. On the contrary, argumentation systems like epistemic graphs require (ad-hoc) reasoning algorithms that do not follow the traditional probabilistic reasoning techniques. In the framework by Rocha and Cozman (2022) conditioning on atoms that appear negated in a body is deferred to future work.

Implementation. In Section 6.3 we present a system that computes SMProbLog semantics, while in the past the focus has been on developing new argumentation semantics rather than tools to compute them. Moreover, we provide an implementation of the EM learning algorithm for probabilistic parameters whose application to probabilistic argumentation problems is novel.

5.3 Conclusion

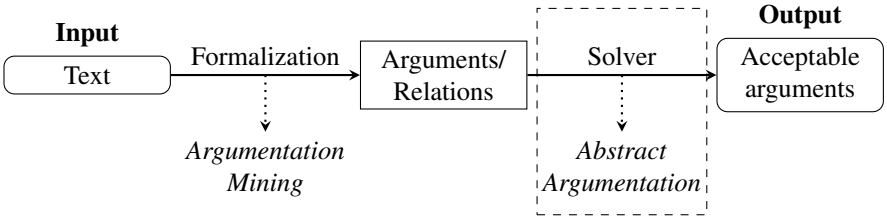
In this chapter we addressed our research question **Q3** by interpreting a probabilistic argument graph as a probabilistic graphical model for arguments' beliefs, encoded by means of PLP. We exploited typical constructs from PLP languages, such as the negation in the head and annotated rules, to provide a compact and intuitive representation of the epistemic effects of beliefs and the respective arguments. This technique provides a declarative framework for argumentation problems which is flexible and adaptable, since it can easily be extended to model new types of relations and argumentative features.

Chapter 6

Probabilistic Argumentation Models: Reasoning

This chapter is largely based on the paper submitted for publication:

P. Totis, A. Kimmig, and L. De Raedt (2023). "SMProbLog: Stable Model Semantics in ProbLog and its Applications in Argumentation". In: *Theory and Practice of Logic Programming*



In this chapter we propose a new PLP framework to reason over probabilistic argumentation problems, since we argue that existing PLP systems are limited in what problems they can model and reason about. In fact, traditional PLP frameworks do not allow modelling cyclic relations involving negations. This is a pattern often found in argument graphs, where reciprocal attacks are common. For example, encoding this pattern, i.e. accepting a_1 inhibits my belief in a_2 and vice versa, corresponds to the logic rules $\neg a_1 \leftarrow a_2.\ \neg a_2 \leftarrow a_1.$ A model containing such rules

Framework	Model semantics	Distribution semantics
PLP frameworks (ProbLog, PRISM, ...)	Least or 2-valued Well-founded model	✓
Probabilistic ASP (P-Log, LP^{MLN})	Stable model	✗
SMProbLog	Stable model	✓

Table 6.1 : PLP frameworks overview.

would not be valid in traditional PLP frameworks, because they assume that each (deterministic) logic program induced by the probabilistic model has exactly one two-valued well-founded model (Gelder et al. 1991). Programs containing cyclic dependencies involving negation usually do not satisfy this assumption. Probabilistic Answer Set Programming frameworks, e.g. Baral et al. (2009) or Lee and Wang (2016), do not make this assumption, however the definition of the probability distribution is not based on distribution semantics (Sato 1995), hence there are differences that we will analyze throughout this chapter. Table 6.1 summarizes the main characteristics of PLP frameworks.

6.1 Reasoning over smProbLog Programs

In this section we present a novel PLP semantics to reason over the programs obtained from the mapping of probabilistic argumentation frameworks described in Section 5.1. In fact, traditional PLP frameworks cannot reason over these programs because they may define cyclic dependencies through negation (Example 5.9). In PLP, in particular ProbLog (Fierens et al. 2011), the requirement of a single two-valued well-founded model is always satisfied when the program does not define cyclic relations through negation (a sufficient but not necessary condition). In fact, in the case of well-founded semantics for normal logic programs, cycles through negation can result in a unique model where some atoms are labelled with a third value "undefined" (Gelder et al. 1991). At the same time, in stable model semantics, which is two-valued, cycles through negation can lead to one, many, or zero stable models (Gelfond and Lifschitz 1988).

Example 6.1. Gelder et al. (1991) shows that the program $a \leftarrow \sim b. b \leftarrow \sim a.$ has no two-valued well-founded model, hence the three-valued well-founded model considers both a and b as *undefined*. On the other hand, the program has two stable models, namely $\{a, \neg b\}$ and $\{\neg a, b\}$.

For this reason in Answer Set Programming (ASP) non-deterministic choices between atoms can be expressed by recursive dependencies through negation. The answer sets of a program are possible sets of beliefs that a rational agent may hold on the basis of the information expressed by the logic rules (Gelfond and Lifschitz 1991). For this reason, we will consider stable model semantics in order to handle non-probabilistic choices introduced by the logic component.

The case with multiple stable models corresponds to the expression of non-deterministic aspects of the problem modelled by the logic rules (Saccà and Zaniolo 1990).

Example 6.2. Consider Example 2.2: if we think that Diane will find Hotel X too expensive with probability 0.6 and that Hotel Y will be noisy with probability 0.7, we can write the following program, with the equivalent rewriting on the right:

Program:

```
0.6::too_expensive_X.
0.7::too_noisy_Y.
stay_at_Y ← too_expensive_X.
stay_at_X ← too_noisy_Y.
¬stay_at_Y ← stay_at_X.
¬stay_at_X ← stay_at_Y.
```

Internal rewriting:

```
0.6::too_expensive_X.
0.7::too_noisy_Y.
stay_at_Y_pos ← too_expensive_X.
stay_at_X_pos ← too_noisy_Y.
stay_at_Y_neg ← stay_at_X.
stay_at_X_neg ← stay_at_Y.
stay_at_X ← stay_at_X_pos,
            ∼stay_at_X_neg.
stay_at_Y ← stay_at_Y_pos,
            ∼stay_at_Y_neg
```

Intuitively, if the counterarguments for both hotels are considered, i.e. the possible world where *too_expensive_X* and *too_noisy_Y* are included, then we are left with a choice between hotel X and hotel Y. In fact, while the first two rules of the program suggest that Diane should stay at both hotels, the last two rules state that this is logically not possible and only one of the two should be chosen. Therefore, in this possible world logic defines a non-determisitic choice that was not modelled by probabilistic facts.

This case is not compatible with PLP frameworks because the underlying assumption is that uncertainty is fully captured by each total choice and therefore that a user exactly knows the causes and effects of all relevant non-deterministic events that might happen, e.g. "believing in argument a determines a choice between arguments b and c". However, we showed in Section 5.1 (for instance Example 5.2 and 5.5) that this is not always the case and obtaining such complete information over the choices may require inference itself. For instance, in argumentation some choices are induced by the relations between arguments (Example 5.9) and may be required only if some particular conditions apply, e.g. not accepting an argument, which may depend in turn on other probabilistic aspects of the problem. Stable model semantics provides a

solution to handling non-probabilistic choices introduced by the logic component of a program. At the same time, the absence of a stable model signals that a given selection of probabilistic facts F (total choice) is inconsistent with the knowledge encoded in the rules R of the program.

A program with zero stable models reflects the case where it is not possible to define an interpretation of the atoms that satisfies all logic rules, therefore a program that has at least one stable model is called "consistent" (Gelfond 2008).

Example 6.3. Consider a program modelling the famous barber paradox: in a village the barber shaves all men, and those only, who do not shave themselves, does the barber shave himself?

```
0.5::barber(bob).   0.5::villager(bob).
shaves(X,Y) ← barber(X), villager(Y), ∼shaves(Y,Y).
```

In the possible world where *bob* is both a villager and the barber (with probability 0.25), we have an inconsistency: there is neither a two-valued well-founded model nor a stable model. Therefore, we will say that the program is inconsistent with probability 0.25 and the atoms for the inconsistent possible worlds are interpreted with a third value "inconsistent". This requires us to point out a relevant consideration about the semantics of total choices.

In the past total choices have been defined in two different ways, as an *interpretation* or a *selection* of facts. They are equivalent under the assumption of a one-to-one correspondence between a total choice and the least model or the two-valued well-founded model, but this is not the case when considering a more general semantics for models. In the original definition by Sato (1995) a total choice is an *interpretation* (an assignment of truth values) of the set of probabilistic facts, whose probability distribution is extended to a distribution over least models. Fierens et al. (2011), on the other hand, define in ProbLog an atomic choice $(p : f)$ as the choice of *selecting* f for being included with probability p or discarded with probability $1 - p$. A total choice for a ProbLog program $\mathscr{L} = F \cup R$ (probabilistic facts F plus rules R) is thus any subset $\omega \subseteq F$ of the facts, which defines a logic program $\omega \cup R$. These two definitions are equivalent under two-valued well-founded semantics because facts are always interpreted as true. This holds also when considering programs with at least one stable model. However, if there is no stable model, it is not possible to consider a total choice an interpretation of the (probabilistic) facts.

Example 6.4. In Example 6.3 defining the total choice as an interpretation assigning true to both probabilistic facts does not generalize to a normal logic program without (two-valued) models, because such interpretation is no longer a subset of any (two-valued) model. The total choice $\{barber(bob), villager(bob)\}$ cannot be regarded as a

partial interpretation where both are true because there is no model that extends such interpretation. On the contrary, considering the total choice as a selection of facts does not make any assumption on the eventual interpretation. $\{barber(bob), villager(bob)\}$ can be thus regarded as an inconsistent choice for the given program and the logical atoms can be interpreted as inconsistent instead of *true*.

Therefore, in this thesis we adopt the definition of total choices as subsets of (probabilistic) facts, which thus makes no assumption about their interpretation in the corresponding model(s). More specifically, our semantics uses the following principles to deal with the novel setting.

Zero models. Inconsistent programs correspond to a loss of probability mass in the interpretations of probabilistic facts, because such probability mass can no longer be associated to a two-valued model. For instance:

Example 6.5. An alarm goes off when a burglary or earthquake happen. The owner however thinks that the alarm was not properly installed and should not go off because of a valid reason. If the alarm does not activate then the owner is right. We can model this situation with the following (probabilistic) normal logic program:

```
0.5::burglary. 0.5::earthquake.
alarm← burglary.
alarm← earthquake.
df← alarm, ∼defective.
right←∼alarm.
```

There are four total choices: $\omega_1 = \{burglary, earthquake\}$, $\omega_2 = \{burglary\}$, $\omega_3 = \{earthquake\}$, $\omega_4 = \{\}$. The logic part of the program is inconsistent when *alarm* is true, therefore when either *burglary* or *earthquake* are true. This means that for three total choices, ω_1, ω_2, and ω_3, the corresponding possible world has neither a two-valued well-founded model nor stable models. In these cases the program is inconsistent with probability $P(\omega_1) + P(\omega_2) + P(\omega_3) = 0.75$ and the only total choice corresponding to a consistent possible world is ω_4 which corresponds to the stable model $\{right\}$.

We thus assign the probability mass associated to inconsistencies to an "inconsistent" state, whose probability thus quantifies how likely it is to consider an inconsistent possible world among those defined by the program. We define this in terms of a model where all atoms are labelled with a third value "inconsistent". This solution has the advantage of preserving the probability mass of total choices by assigning it to the model representing an inconsistent possible world. The semantics is thus informative as to the degree of inconsistency of the logical part of the program. Note that this

is different from three-valued well-founded semantics, where the atoms involved in the cycles responsible for multiple stable models are labelled with the third value ("undefined") as well, and not all atoms are interpreted undefined at the same time.

Example 6.6. When arguing about getting vaccinated or not there are different media we can choose to believe in: virologists, newspapers and Facebook. Virologists endorse vaccines and their safety, newspapers argue that there is some degree of risk involved, Facebook posts reject the acceptability of such degree of risk. We can model this reasoning pattern and the level of trust in the different media with the following program:

```
0.9::trust_virologists. 0.7::trust_newspapers.
0.3::trust_facebook.
safe ← trust_virologists, ∼dangerous.
reasonable_risk ← trust_newspapers, ∼safe.
dangerous ← trust_facebook, ∼reasonable_risk.
```

The three rules determine a cyclic dependency (of odd length) with negation involved between *safe*, *reasonable_risk* and *dangerous* and therefore trusting all media is an inconsistency, because there is no possible truth assignment to *safe*, *reasonable_risk*, and *dangerous*. We will then say that this program is inconsistent with probability $0.9 \cdot 0.7 \cdot 0.3 = 0.189$. The remaining probability mass of $(1 - 0.189) = 0.811$ is divided over the other possible words where the total choice of probabilistic facts is a consistent choice according to the logic of the program. These choices thus induce a two-valued assignment to the atoms of the program.

At least one model. In order to give a probabilistic interpretation of logical choices, we follow the principle of maximum entropy. The maximum-entropy principle is a widely adopted principle in Bayesian reasoning and Statistical Relational Learning. It states that the probability distribution which best represents the current state of knowledge about a system is the one with the largest entropy (Jaynes 1988). In this setting this principle translates to considering a uniform distribution for the (multiple) stable models of a total choice.

Example 6.7. Consider the following (probabilistic) normal logic program:

```
0.5::burglary.  0.5::earthquake.
alarm← burglary.
df← earthquake.
alarm← ∼df.
df← ∼alarm.
```

There are four total choices: $\omega_1 = \{burglary, earthquake\}$, $\omega_2 = \{burglary\}$, $\omega_3 = \{earthquake\}$, $\omega_4 = \{\}$. While the first three correspond to one stable model, i.e. $MOD(\omega_1) = \{\{burglary, earthquake, alarm, df\}\}$, $MOD(\omega_2) = \{\{burglary, alarm\}\}$, $MOD(\omega_3) = \{\{earthquake, df\}\}$, ω_4 has two stable models: $MOD(\omega_4) = \{\{alarm\}, \{df\}\}$. Since this choice is not related to beliefs but rather to logical consistency, we assume that all stable models are equally probable for the given total choice ω_4. In this case we will choose *alarm* half of the times and *df* the other half. Since $P(\omega_4) = 0.25$ the probability of the model $\{alarm\}$ is equal to the probability of the model $\{df\}$, namely 0.125.

Example 6.8.

The program of Example 6.2 has four consistent possible worlds, one of which with two stable models:

Possible world's choices	$P(\omega)$	Model M_ω	$\hat{P}(M_\omega)$
$\omega_1 = \{\}$	0.12	$M_{\omega_1} = \omega_1$	0.12
$\omega_2 = \{too_expensive_X\}$	0.18	$M_{\omega_2} = \omega_2 \cup \{stay_at_Y\}$	0.18
$\omega_3 = \{too_noisy_Y\}$	0.28	$M_{\omega_3} = \omega_3 \cup \{stay_at_X\}$	**0.28**
$\omega_4 = \{too_expensive_X,$ $too_noisy_Y\}$	0.42	$M^1_{\omega_4} = \omega_4 \cup \{stay_at_Y\}$	0.21
		$M^2_{\omega_4} = \omega_4 \cup \{stay_at_X\}$	**0.21**

Consider ω_4: when both probabilistic facts *too_expensive_X* and *too_noisy_Y* are conjoined with the rules, there is no probabilistic justification for choosing one hotel over the other, but rules impose a choice. In this case, we consider the probability of *stay_at_X* and *stay_at_Y* to be equivalent, because in a possible world where both counterarguments are true, a choice of a hotel is still necessary. The probabilistic part of the program which is quantified by the user is fixed in the total choice ω_4, therefore in determining the models of ω_4 there is no probabilistic justification for assigning one model a probability higher than another. In the definition of the probability distribution of the models $\hat{P}$, we thus divide equally the probability of the possible world determined by ω_4 between its two models: $\hat{P}(M^1_{\omega_4}) = \hat{P}(M^2_{\omega_4}) = \frac{0.42}{2} = 0.21$. The probability of an atom a remains the sum of the probability of the models where a is true, for example the probability of *stay_at_X* is $\hat{P}(M_{\omega_3}) + \hat{P}(M^2_{\omega_4}) = 0.28 + 0.21 = 0.49$.

6.1.1 Formal Semantics

We define the models for each total choice and the corresponding probability in order to extend a probability distribution over total choices to a probability distribution over models. Each total choice ω corresponds to: (1) a probability $P(\omega)$ as defined by distribution semantics and (2) a set $SM(\omega)$ of stable models, over which the probability gets distributed if it is non-empty. For this distribution to be well-defined, we only consider programs with finite Herbrand base, and thus a finite number of interpretations.

Definition 31. *A valid* SM*ProbLog program is a probabilistic normal logic program without function symbols.*

Definition 32. *Given a valid* SM*ProbLog program $\mathscr{L}$, the probability $P(\omega)$ of a total choice ω is:*

$$P(\omega) = \prod_{(f:p)\in\mathscr{L}, f\in\omega} p \cdot \prod_{(f:p)\in\mathscr{L}, f\notin\omega} 1-p$$

that is, the product of the probabilities of the facts for being included/excluded from ω.

Example 6.9. The probability of the total choices in Examples 6.5 and 6.7 is $P(\omega_1) = P(\omega_2) = P(\omega_3) = P(\omega_4) = 0.5 \cdot 0.5 = 0.25$

Definition 33. *Given a probabilistic normal logic program $\mathscr{L} = F \cup R$ and a corresponding total choice $\omega \subseteq F$, $SM(\omega)$ is the set of stable models of the (deterministic) program $\omega \cup R$.*

We distinguish two cases: $SM(\omega) = \emptyset$ and $|SM(\omega)| > 0$, and we define accordingly a set of models for a possible world.

A model is a three-valued interpretation (T,F) where T is the set of true atoms and F is the set of false atoms. The remaining atoms are interpreted as being part of an inconsistent possible world.

Definition 34. *Given a probabilistic logic program $\mathscr{L}$ and let $HB(\mathscr{L})$ be its Herbrand base. For each total choice $\omega \in \Omega_{\mathscr{L}}$, the corresponding set of models $MOD(\mathscr{L},\omega)$ is:*

$$MOD(\mathscr{L},\omega) = \begin{cases} \{(\emptyset,\emptyset)\} & \text{if } |SM(\omega)| = 0 \\ \{(M_\omega, HB(\mathscr{L}) \setminus M_\omega) \mid M_\omega \in SM(\omega)\} & \text{if } |SM(\omega)| > 0 \end{cases}$$

Each total choice thus has at least one model: either the "inconsistent" model or one or more stable models. We extend the probability of the total choices to a distribution over the corresponding models by applying the maximum-entropy principle.

The probability of a model $M_\omega \in MOD(\mathscr{L}, \omega)$ is the probability of the corresponding total choice ω normalized w.r.t. the number of the models for that possible world:

Definition 35. *Given a probabilistic normal logic program $\mathscr{L}$ and a total choice ω, $\forall M_\omega \in MOD(\mathscr{L}, \omega)$:*

$$\hat{P}(M_\omega) = \frac{P(\omega)}{|MOD(\mathscr{L}, \omega)|}$$

We thus derive a distribution over models $\hat{P}$ from the probability distribution over total choices P. We can now define the probability of the program of being inconsistent and of a query of an atom. The degree of inconsistency of the program $\mathscr{L}$, $\mathbb{P}(\mathscr{L} \vdash \bot)$ is the sum of the probabilities of interpreting the atoms as inconsistent in each possible world:

$$\mathbb{P}(\mathscr{L} \vdash \bot) = \sum_{\omega \in \Omega_\mathscr{L}, MOD(\mathscr{L}, \omega) = \{(\emptyset, \emptyset)\}} \hat{P}((\emptyset, \emptyset))$$

As for the consistent possible worlds, we can query the program for the probability of an atom being interpreted as true (or false):

Definition 36. *Given a probabilistic normal logic program $\mathscr{L}$, the probability of success of querying a is the probability of interpreting a true in a consistent possible world:*

$$\mathbb{P}(a) = \sum_{a \in M_\omega, (M_\omega, HB(\mathscr{L}) \setminus M_\omega) \in MOD(\mathscr{L}, \omega), \omega \in \Omega_\mathscr{L}} \hat{P}(M_\omega)$$

Note that we can derive the probability of an atom being false from the probability of the inconsistent possible worlds and the probability of success:

$$\mathbb{P}(\neg a) = 1 - \mathbb{P}(\mathscr{L} \vdash \bot) - \mathbb{P}(a)$$

because the probability mass of the non-inconsistent models is divided over the two-valued stable models, therefore if an atom is in a consistent possible world, then it is interpreted either true or false.

Example 6.10. We complete Example 6.5: the probability of the program being inconsistent is $P(\omega_1) + P(\omega_2) + P(\omega_3) = 0.75$. The only consistent possible world is ω_4, whose only model is $\{e\}$: $\hat{P}(right) = 0.25$, therefore $\mathbb{P}(right) = 0.25$ and $\mathbb{P}(\neg right) = 0$, vice versa, $\mathbb{P}(burglary) = \mathbb{P}(earthquake) = \mathbb{P}(alarm) = \mathbb{P}(df) = 0$ and $\mathbb{P}(\neg a) = \mathbb{P}(\neg b) = \mathbb{P}(\neg c) = \mathbb{P}(\neg d) = 0.25$.

Example 6.11. We complete Example 6.7: models $\{burglary, earthquake, alarm, df\}$, $\{burglary, alarm\}$, $\{earthquake, df\}$ are the unique model for, respectively, $\omega_1, \omega_2, \omega_3$ hence $\hat{P}(\{burglary, earthquake, alarm, df\}) = \hat{P}(\{burglary, alarm\}) = \hat{P}(\{earthquake, df\}) = P(\omega_1) = P(\omega_2) = P(\omega_3) = 0.25$. The likelihood of ω_4 is

uniformly distributed over the two stable models $\{alarm\}$ and $\{df\}$, whose probability is thus $\frac{P(\omega_4)}{|MOD(\mathscr{L},\omega_4)|} = \frac{0.25}{2} = 0.125$. Therefore, $\mathbb{P}(burglary) = \mathbb{P}(earthquake) = 0.5$, $\mathbb{P}(alarm) = \mathbb{P}(df) = 0.625$.

6.2 The Joint Distribution of Beliefs

By defining the semantics of SMProbLog programs we also define the semantics of the mapping from probabilistic argumentation problems to PLP (Section 5.1). Under these semantics a probabilistic argument graph thus represents a *joint probability distribution* of the arguments' beliefs by means of the corresponding probabilistic logic program. As we argued in Section 5.1, the joint probability distribution represents the marginal and conditional strengths of the beliefs in the arguments from their biases and relations. That is, we are concerned with determining the probabilities of logically related propositions.

Example 6.12. In Example 5.4 the marginal probabilities of each argument in the inferred joint probability distribution are: $\mathbb{P}(arg(a_1)) = 0.29$, $\mathbb{P}(arg(a_2)) = 0.63$, $\mathbb{P}(arg(a_3)) = 0.3$, $\mathbb{P}(arg(a_4)) = 0.81$, $\mathbb{P}(arg(a_5)) = 0.6$, $\mathbb{P}(arg(a_6)) = 0.58$.

These marginal distributions are thus coherent with the initial biases and relations in the sense of Nilsson (2014). Polberg et al. (2017) define several properties for epistemic argument graphs, among which coherency, to evaluate arbitrary assignments of values in $[0,1]$ under the principle that a probability higher than 0.5 defines an argument as accepted. In the epistemic approach literature, acceptance is thus a function of the arbitrary probability values with respect to an arbitrary threshold. Marginal probabilities, representing the agent's beliefs, are thus directly assigned to arguments, and they are evaluated with respect to local properties. Hunter, Polberg, et al. (2018) generalize this approach with *epistemic graphs*, where, instead of considering a single assignment, a set of belief assignments is associated to the argument graph such that they satisfy a set of constraints. This does not narrow down the belief distribution to a single joint probability distribution, but admits a set of possible assignments compatible with the constraints.

Therefore, past work evaluates arbitrary belief assignments with respect to an arbitrary threshold: considering a belief *absolute* value of 0.5 the discriminant between accepted and rejected arguments. This poses a well-known question in the PLP literature, "Where do the numbers come from?" (Charniak 1991), in regard of evaluating subjective beliefs with a fixed value of reference. The main difference with our approach is that in our framework acceptance is a function of logic and its (joint) probability is the associated (joint) belief measure. The joint belief (probability) distribution

represents how the prior beliefs (biases) interact with each other (arguments' logical relations) to define the *unique* joint belief distribution that is coherent with the given argumentative structure (argument graph). The joint belief distribution defines how the *relative* strength of the arguments changes when their logical relations are considered, regardless of an acceptance threshold. This expresses two principles of modelling uncertainty in AI. First, as Finetti (1931) remarks in his discussion on subjective beliefs, rather than evaluating the absolute degree of beliefs, an agent is coherent when the relative differences between such degrees do not contradict probability axioms. Second, we apply the approach of probabilistic graphical models to probability and reasoning: "Probability is not really about numbers; it is about the structure of reasoning" (Glenn Shafer, cited in Pearl (1989)).

This method allows us to learn probabilities (beliefs) from data (Sections 6.3.2 and 6.4) and thus ground their nature in the observation of accepted or rejected arguments. Moreover, to study the properties of the joint probability distribution of arguments we can rely on the traditional inference tasks of PLP frameworks. Interestingly, Nilsson (2014) remarks that considering the consequences and the explanations for beliefs is fundamental for critical thinking, and we can do precisely this by means of two typical PLP tasks: conditional queries and most probable explanation queries.

Conditionals. We can perform conditional reasoning on the joint probability distribution by querying the probability of atoms given some evidence. We can thus analyze the consequences of beliefs by answering conditional queries such as: "What are the consequences of accepting a_1?". Note that a_6 is *conditionally independent* of the other arguments *given* a_1, because when a_1 is known to be accepted (or rejected) the belief in a_6 depends only on the corresponding attack and bias.

Example 6.13. By querying the model from Example 5.9 with the addition of the evidence $arg(a_1) = true$, usually simply denoted with $arg(a_1)$, we infer the following conditional probabilities: $\mathbb{P}(arg(a_1)|arg(a_1)) = 1$, $\mathbb{P}(arg(a_2)|arg(a_1)) = 0.08$, $\mathbb{P}(arg(a_3)|arg(a_1)) = 0.36$, $\mathbb{P}(arg(a_4)|arg(a_1)) = 0.75$, $\mathbb{P}(arg(a_5)|arg(a_1)) = 0.58$, $\mathbb{P}(arg(a_6)|arg(a_1)) = 0.28$. We can also condition the distribution on the relations, for instance by adding the evidence that accepting $arg(a_1)$ always excludes $arg(a_6)$. In this case, if we make explicit the belief in the attack with a probabilistic fact *att* we obtain the same distribution except for $\mathbb{P}(arg(a_6)|arg(a_1), att) = 0$.

The conditional independence assumption materializes also when we introduce a belief update to an argument which is conditionally independent of another, given the evidence for an argument that represents the connection of the two beliefs.

Example 6.14. Consider again Example 6.13 with evidence $arg(a_1) = true$. If we add the evidence $arg(a_5) = false$, let $ev = (arg(a_1) = true \wedge arg(a_5) = false)$,

then we obtain the following marginals: $\mathbb{P}(arg(a_1)|ev) = 1$, $\mathbb{P}(arg(a_2)|ev) = 0.08$, $\mathbb{P}(arg(a_3)|ev) = 0.36$, $\mathbb{P}(arg(a_4)|ev) = 0.62$, $\mathbb{P}(arg(a_5)|ev) = 0$, $\mathbb{P}(arg(a_6)|ev) = 0.28$. The only beliefs affected by the new information about a_5 are those for a_5 and a_4, because a_3 is independent of a_5, a_1 is known, and a_6 and a_2 are *conditionally independent given* a_1.

Conditional independence can also be exploited in argumentation to select arguments that provide new information with respect to an argument or claim. If these arguments' uncertainty can be eliminated by means of experiments or observations, the conditional independence assumption allows us to select only those strictly necessary to influence the belief in the argument of interest.

Example 6.15. Bayesian statistical methods are suited for designing medical clinical trials to find an optimal design which minimizes the empirical results required and at the same time maximizes the information provided regarding the quality of the trial (Berry 2006). Given an argumentative graph with respect to some treatment hypothesis, if some experimental results are unknown then they are associated with a probability (degree of belief). The uncertainty can be removed by collecting empirical evidence, but this comes at a cost of time and resources. In this case, the conditional independence of the claim with respect to some experiments given other empirical results can be exploited to select only those experiments that truly affect the belief in the claim.

In SMProbLog it is also possible to condition the theory to be consistent, by providing the evidence that an atom is not interpreted as inconsistent. This allows us to reason about how beliefs change if they are conditioned to be consistent with the logical structure.

Example 6.16. Consider Example 6.3: if we add the evidence $villager(bob) = \neg inconsistent$ then we consider only the consistent possible worlds. Therefore, the probability of each possible world is normalized w.r.t. the probability mass of the consistent total choices. Hence, the probability that bob is both a villager and a barber becomes 0, because there are no models for consistent worlds where this is true. Each consistent possible world then increases its probability from 0.25 to $\frac{1}{3}$. For instance $\mathbb{P}(villager(bob)|villager(bob) = \neg inconsistent) = \frac{1}{3}$.

MPE. *Most probable explanation* (MPE) is a typical PLP task that allows us to address the other point of critical thinking, reasoning on explanations for beliefs. The MPE task is to compute the most likely possible world where some given evidence holds.

Example 6.17. If we compute a MPE query for $arg(a_1) = true$, we are asking which set of beliefs compatible with accepting a_1 is the most probable. This means finding the most likely set of probabilistic facts representing a choice of biases and relations with the consequence of accepting a_1. In the MPE world the biases believed to support the corresponding argument are $\{bias(a_1).bias(a_2).bias(a_4).bias(a_5).bias(a_6).\}$. All relations are believed to hold, except for the attack of a_4 to a_1. a_1 is thus independent of a_4 and a_5 and its acceptance is supported by its bias and the non-deterministic choice with a_2 (following the evidence). These choices, that is, the MPE, have probability 0.0093. Their model is $\{arg(a_1), \neg arg(a_2), \neg arg(a_3), arg(a_4), arg(a_5), \neg arg(a_6)\}$ which is thus compatible with the evidence of accepting $arg(a_1)$. The two arguments attacked by a_1, a_2 and a_6, are rejected, along with a_3 because of the choice of not considering a bias towards it.

6.2.1 Epistemic Properties

The joint probability distribution describes the subjective assignment of beliefs to arguments coherent with all possible belief choices of prior bias and relations, regardless of arbitrary thresholds. This means that the joint beliefs are globally coherent with the argumentative structure and therefore the marginal beliefs are also locally coherent with the parents and children's beliefs. The properties for epistemic probabilistic argumentation by Polberg et al. (2017) can thus be used to evaluate the bias of the agent, which is indeed an arbitrary assignment of values, but in evaluating the joint probability distribution, having probability higher than 0.5 has no special meaning. We clarify this by considering some of these properties and show that the marginal belief of the arguments defined by the joint distribution is compatible with the given argumentative structure even if they may not satisfy such properties.

Coherency. Polberg et al. (2017) define coherency in epistemic probabilistic argumentation as the property that for all attacks (a,b) believed with probability > 0.5, $P_A(a) \leq 1 - P_A(b)$. In the joint probability distribution the marginal probability of arguments $\mathbb{P}(arg(_))$ is coherent not just locally, considering pairwise marginals, but globally, where the structure of all relations and biases is taken into account.

Example 6.18. Consider the case where two arguments a and b are such that a attacks b with probability 0.6 and both arguments have bias (prior) 1. a is not attacked so $P_A(a) = \mathbb{P}(arg(a)) = 1$. We can say that the initial assignment, the priors, is not coherent in the sense of Polberg et al. (2017) because $P_A(a) = 1 \not\leq 1 - 1$. However, the joint probability distribution defines posterior marginals that are coherent in the sense of Nilsson (2014): $\mathbb{P}(arg(b)) = 0.4$ because the uncertainty about the attack leaves room for possible worlds where the attack is not believed to hold (probability

$1-0.6=0.4$) and thus it is possible to accept both a and b. Here $1=\mathbb{P}(arg(a)) \not\leq 1-\mathbb{P}(arg(b))=0.6$, but $\mathbb{P}(arg(a))$ should not be less or equal than 0.6 because it is independent of b, while accepting b is influenced by accepting a and its attack, and thus its prior bias, not the attacker, is inhibited to 0.6.

Rational. A rational (resp. strict) assignment resembles the conflict-free principle that if an attacker a is accepted than the target b should not: $P_R((a,b)) > 0.5$ and $P_A(a) > 0.5$ implies $P_A(b) \leq 0.5$ (resp. $P_A(b) < 0.5$). In our framework, when choosing a set of prior beliefs (probabilistic facts) the conflict-freeness of the accepted (true) arguments is always guaranteed by the logic rules. This does not exclude the possibility that both a and b in the joint probability distribution are believed with probability > 0.5. In fact, when an argument has high bias, e.g. we think it is almost certain, only a combination of strong attackers with strong relations can lower its inferred probability under 0.5.

Example 6.19. For example, a prior probability of 0.6 of believing in a and 0.6 in the attack, yields an inhibition by a factor of 0.36, which is not guaranteed to lower the marginal probability of b under 0.5. In 36% of the possible worlds b is rejected because of the presence of a and the attack, but if $P_A(b)=1$ then $\mathbb{P}(arg(b))=0.64$, together with $\mathbb{P}(arg(a))=0.6$ and $P_R((a,b))=0.6$. $\mathbb{P}(arg(b)) \leq 0.5$ only when $\mathbb{P}(a) \cdot P_R((a,b)) \geq 0.5$ (in the case of inhibition from a single attack). This does not mean that the joint probability distribution is not rational in the sense that it defines a set of accepted arguments not conflict-free. The joint probability distribution reflects the relative strength of the arguments and simply defines a (coherent) probability ≥ 0.5 to be in a possible world where either a or b are accepted, but there is no possible world where the attack is believed and both are accepted.

Protective. A (restricted) protective assignment is such that $P_R((a,b)) \geq 0.5$ (resp. $P_R((a,b)) > 0.5$) and $P_A(b) > 0.5$ implies $P_A(a) \leq 0.5$. Similarly to the rational property, in epistemic probabilistic argumentation this is the principle of rejecting the attacker if the target is accepted. Again, this principle in the joint belief distribution is not enforced by means of the probability (belief) values, but by means of the logical structure of the attack relation.

Example 6.20. Our previous example also describes a belief distribution that is not protective: while $\mathbb{P}(b)=0.64$ and $P_R((a,b))=0.6$, $\mathbb{P}(a)$ remains equal to its original prior $P_A(a)=0.6$, thus $\mathbb{P}(arg(a)) > 0.5$. This does not mean that both arguments are accepted *because* their probability is higher than 0.5, but it means that a joint belief induced by the argumentative structure coherent with the prior bias assigns a subjective marginal belief $\mathbb{P}(arg(a))$ and $\mathbb{P}(arg(b))$ higher than 0.5 to the arguments.

The difference in strength is still coherent with the respective biases and belief in the attack relation and probability axioms.

Similar arguments can be constructed for the other properties from Polberg et al. (2017): past epistemic argumentation approaches consider the probabilities of the argument graph as the *marginal* beliefs of the agent and check if those are (locally) rational, coherent, ... Our approach uses this arbitrary assignment as the *prior* set of beliefs, and the argumentative structure, to infer a joint belief distribution that is compatible with the initial arbitrary assignment according to the probability postulates and the logical relations of the arguments. The joint belief (probability) distribution thus represents how the prior beliefs (biases) interact with each other to define *marginal and conditional* beliefs that are coherent with the given priors and argumentative (logic) structure.

6.3 Inference and Learning

In this section we describe the inference and learning algorithms for SMProbLog semantics. We consider the inference and learning algorithms implemented in ProbLog2 and adapt them to the semantics defined in Section 6.1. This allows us to study the advantages and shortcomings of traditional PLP inference and learning techniques when a more general semantics is concerned, and indirectly examine their application to argumentation problems.

6.3.1 Inference Task

The state-of-the-art inference technique for probabilistic logic programs is reducing the inference task to a weighted model counting problem (cfr. Chapter 2), therefore we consider its applicability in the context of our semantics. In ProbLog2 inference is implemented by means of a transformation of the ground program to Conjunctive Normal Form (CNF), which is the input for a knowledge compiler (under well-founded semantics) that returns the logical circuit (e.g. a d-DNNF) such that on the corresponding arithmetic circuit the WMC task can be solved in polynomial time (Figure 6.1). In the rest of the section we describe the adaptations required to this inference technique required for SMProbLog's semantics.

The normalization of the weight of the models entails considering a variant of the *WMC* problem where models are weighted with the corresponding normalization constant. We denote such constant with $\hat{w}(M_\omega)$, where M_ω is a model for a total choice ω, hence $\hat{w}(M_\omega) = \frac{1}{|MOD(\mathscr{L},\omega)|}$. The *WMC* problem ($WMC_{\mathscr{L}}(\varphi) =$

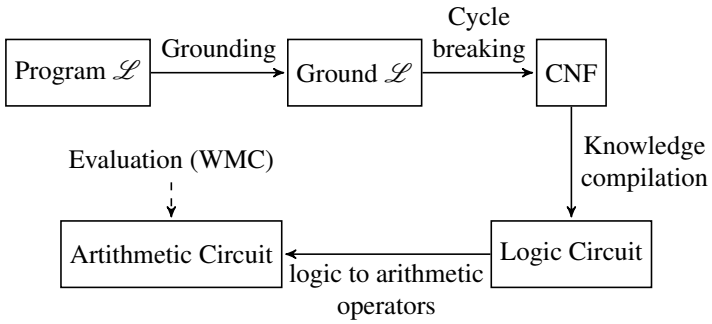


Figure 6.1: ProbLog2 inference schema

$\sum_{M\in MOD(\mathscr{L}),M\models\varphi}\prod_{l\in M} w(l))$ thus becomes:

$$\widehat{WMC}_{\mathscr{L}}(\varphi) = \sum_{M\in MOD(\mathscr{L}),M\models\varphi} \hat{w}(M)\cdot\prod_{l\in M} w(l)$$

This formulation of the problem requires us to determine the weights $\hat{w}(M_\omega)$, which, contrary to the weights of the literals, are not known from $\mathscr{L}$. In order to determine the weighting function $\hat{w}$ we thus need to solve an additional counting problem, namely the task of counting how many models agree on a subset of atoms in $\mathscr{L}$, that is, the probabilistic facts. We now describe the reduction pipeline in SMProbLog to a $\widehat{WMC}$ problem. The reduction can be broadly divided into four components, represented in Figure 6.2: (1) grounding the program $\mathscr{L}$, (2) compiling the logic theory $\mathscr{L}$ into a d-DNNF, (3) deriving a circuit for retrieving the number of models corresponding to a

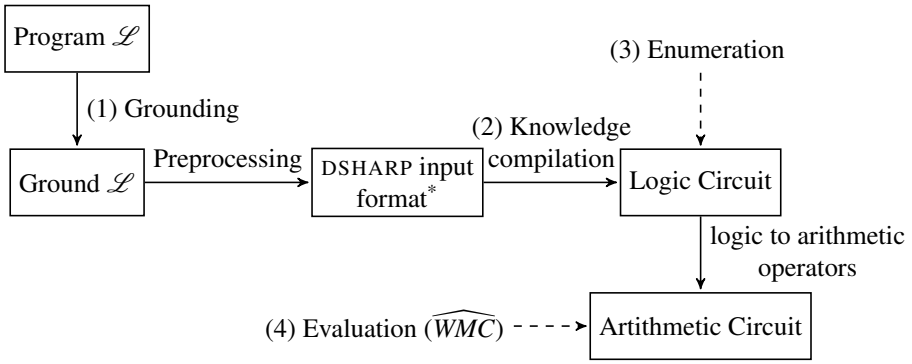


Figure 6.2: SMProbLog inference schema. (*) denotes a different version of DSHARP from ProbLog2, specific for stable model counting.

total choice and an arithmetic circuit for the $WMC_{\mathscr{L}}$ problem (4) the evaluation of the two circuits to define the solution of the $\widehat{WMC}_{\mathscr{L}}$ problem (Algorithm 8).

1) Grounding. In SMProbLog we adopt a standard bottom-up grounding technique. In ProbLog2, the detection of cycles through negations during the grounding step is used as a sufficient condition to reject invalid programs. On the contrary, we use the absence of cycles through negations as a condition to reduce to ProbLog's evaluation algorithm, where SMProbLog's semantics is equivalent, as discussed in Section 6.5. In fact, when this is the case, we skip the enumeration step (Step 3) because $\hat{w}(M) = 1$ for all stable (and well-founded) models M. Therefore, the $\widehat{WMC}_{\mathscr{L}}$ problem becomes equivalent to a $WMC_{\mathscr{L}}$ problem.

2) Compilation. The goal of the compilation step is to transform the ground propositional theory in a representation that allows us to answer efficiently queries about the probability of atoms. Knowledge compilation produces a logical circuit describing the possible worlds' models of the input propositional theory. Therefore, this step regards exclusively the logical part of the program, while the following inference steps concern the computation of probabilities. As in Problog2, the knowledge compiler is a black box with respect to the probabilistic inference system. Therefore, in SMProbLog we are only concerned with transforming the propositional ground theory to the suitable input format. The input format and the implementation details of the knowledge compiler in SMProbLog can be found in Aziz et al. (2015), which extends the DSHARP compiler (Muise et al. 2012) with stable model semantics. In ProbLog2 the input format is a theory in Conjunctive Normal Form (CNF), Aziz et al. (2015) replace it with an encoding in CNF format of the rules, variables and the corresponding strongly connected components. The output is a logical circuit, a d-DNNF, describing the *stable* models of the input propositional theory. The circuit therefore expresses only the models corresponding to the consistent total choices. This representation, with a standard transformation in an arithmetic circuit, allows efficient stable model counting, that is, solving the *WMC* problem. Figure 6.3 shows the output of the knowledge compilers from the representations of Example 6.5 and 6.7. Contrary to ProbLog2, however, the transformation of the logical circuit into an arithmetic circuit is not sufficient to solve the $\widehat{WMC}$ problem. This because the arithmetic circuit allows us to efficiently compute the solution to a *WMC* problem, but the normalization constants $\hat{w}$ are still unknown. Unless the weighting is determined by the absence of cycles through negations, as defined in the previous paragraph, the enumeration step is required to associate each model to the corresponding normalization constant.

3) Enumeration. The purpose of the enumeration step is to count the number of stable models corresponding to each (consistent) total choice, which defines $\hat{w}$.

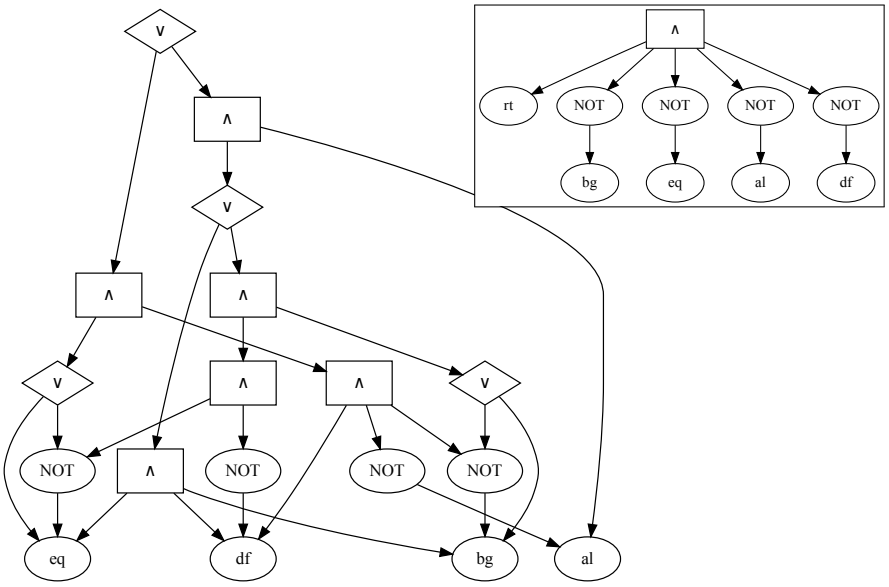


Figure 6.3: Compilation step of Example 6.5 (top right) and Example 6.7 (left). Models for both theories are compactly encoded as d-DNNF logical circuits. $bg = burglary$, $eq = earthquake$, $al = alarm$, $df = df$, $rt = right$.

Unfortunately, this operation cannot be performed efficiently on the representation obtained from the knowledge compiler, because models corresponding to the same total choice may be represented by different nodes and subtrees. Therefore, on this kind of representation we are forced to traverse the circuit in order to retrieve the list of models and the corresponding total choices.

We thus derive from the d-DNNF a circuit where each leaf for a literal l is replaced by a set of (partial) models $\{\{l\}\}$, disjunctions are replaced by the union of the children and conjunctions correspond to the Cartesian product of the children (Figure 6.4). Traversing bottom-up such circuit returns the list of models, from which we build a map $\# : \Omega_{\mathscr{L}} \to \mathbb{N}$ from total choices to the corresponding number of models, i.e. $\#(\omega) = |MOD(\mathscr{L}, \omega)|$. The reciprocal thus defines the normalization constants $\hat{w}$. For instance, in Example 6.7 we obtain $\#(\omega_1) = \#(\omega_2) = \#(\omega_3) = 1$, $\#(\omega_4) = 2$, hence $\hat{w}(\{burglary, alarm, earthquake, right\}) = \hat{w}(\{burglary, alarm\}) = \hat{w}(\{earthquake, defective\}) = 1$, and $\hat{w}(\{alarm\}) = \hat{w}(\{df\}) = \frac{1}{2}$. Note that the total choices that do not appear in the enumeration of models are those that have zero models and thus are inconsistent. Because probabilistic facts are disjoint from the rules' heads, for each model described by the circuit the truth value of the literals corresponding to probabilistic facts defines the corresponding total choice (they can be true iff chosen by

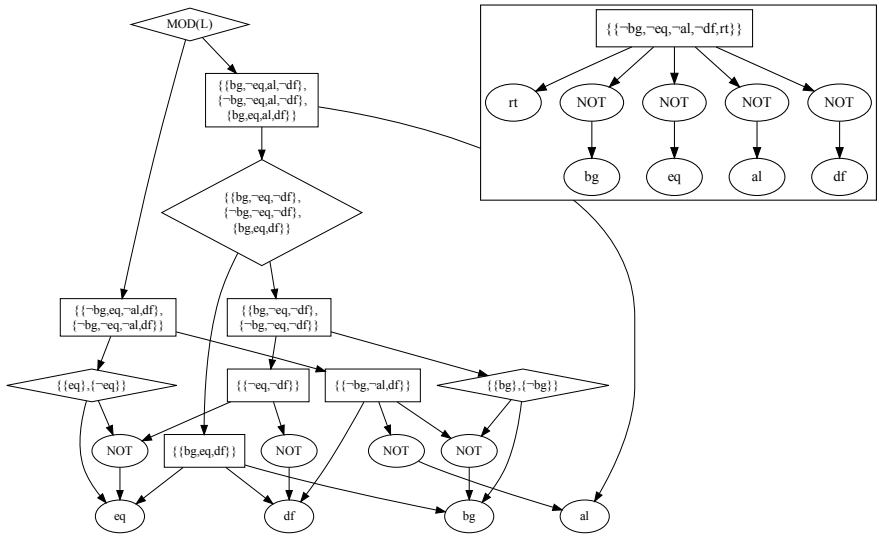


Figure 6.4: Enumeration step of Example 6.5 (top right) and Example 6.7 models (left). Each node contains the corresponding (partial) models. Diamonds are union nodes, squares are Cartesian products. M is the complete list of the 5 models of Example 6.7.

ω).

The circuit traversal is a computationally expensive step since it entails enumerating all possible models. Contrary to sums (logical ORs) and products (logical ANDs), the Cartesian product is not a linear-time operation. Therefore, the computational cost of the internal nodes raises with the length of the lists of partial models corresponding to the children. For this reason, the study of novel knowledge compilation techniques that produce a representation where the normalization constants can be obtained from linear time operations is an interesting direction for improvement.

This is a novel problem because SMProbLog semantics applies a different normalization constant per total choice (2^n with n facts), while in probabilistic ASP the normalization constant is the same for all models, that is, the weight of all possible worlds. This can be obtained with a single evaluation of the weight of the root of the circuit. Therefore, obtaining the normalization constants from a circuit where node-level operations are computable in linear time would remove this bottleneck, which is introduced because we consider a fundamentally different inference task.

4) Evaluation. As we previously mentioned, to evaluate $P(q|E=e)$, with $\varphi = q \wedge E = e$, on a program $\mathscr{L}$ we solve the problem $\widehat{WMC}_{\mathscr{L}}(\varphi) =$

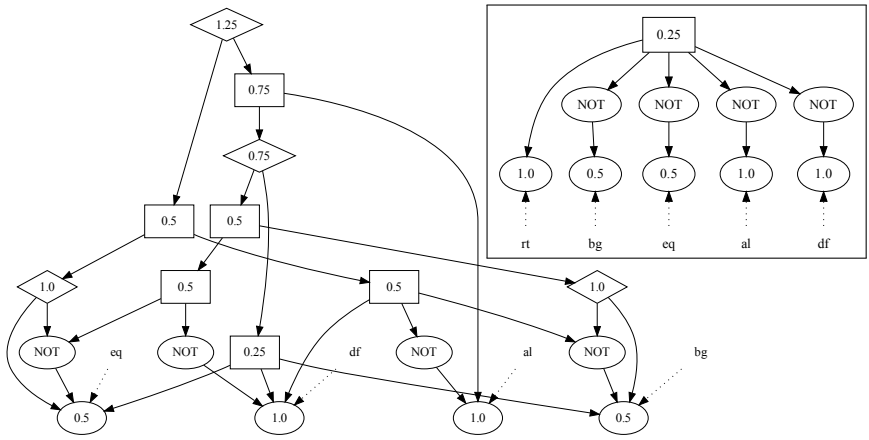


Figure 6.5: *WMC* of Example 6.5 (top right) and Example 6.7 (left) . Each node contains the corresponding weight, logical variables do not influence the weights. Diamonds are sum nodes, squares are product nodes.

$\sum_{M \in MOD(\mathscr{L}), M \models \varphi} \hat{w}(M) \prod_{l \in M} w(l)$. The weights of the leaves in the arithmetic circuit are instantiated according to the query and the given evidence (Chapter 2). Fierens et al. (2011) shows that the arithmetic circuit derived from the d-DNNF computes $WMC_{\mathscr{L}}(\varphi) = \sum_{M \in MOD(\mathscr{L}), M \models \varphi} \prod_{l \in M} w(l)$. Under a one-to-one correspondence between models and total choices, given $\omega \in \Omega_{\mathscr{L}}$ and its model M_ω, $WMC(M_\omega) = WMC(\omega)$. Therefore, if the enumeration step is skipped, i.e. $\hat{w}(M_\omega) = 1$ for all stable (and well-founded) models M_ω, then $WMC(M_\omega) = WMC(\omega) = \widehat{WMC}(M_\omega)$.

On the contrary, if there are multiple models for some total choices, then $WMC(M_\omega) = WMC(\omega) \neq \widehat{WMC}(M_\omega)$, because $WMC(M_\omega)$ overcounts the normalized weights of the models corresponding to the same total choice. Figure 6.5 shows the weight calculation of the root node of the circuit: the root corresponds to all consistent total choices (e.g. Figure 6.4). Example 6.7 shows that the original algorithm overcounts the weight when multiple stable models correspond to the same total choice. This because, their weight is not normalized (cfr. Example 6.23 for the explanation of the 1.25 weight). This raises the question as to how the evaluation algorithm has to account for the normalization constants for SMProbLog semantics.

As Figure 6.4 shows, multiple models can correspond to the same node, therefore it is not possible to associate a normalization constant to a single node. Moreover, the aggregation of the weights in one value per node, as in Figure 6.5, is no longer possible, because two partial models (e.g. $\{\{b\}, \{\neg b\}\}$) may be conjoined (multiplied) with partial models (weights) that require different normalization constants. Once the weights are aggregated it is not possible to normalize the different components

of the weight with different constants. We now describe two evaluation strategies to normalize weights in the evaluation step.

The first is essentially a repetition of the enumeration step: we traverse again the circuit bottom-up and along with each set of atoms we carry the corresponding weight: once a total choice is defined, the weights of each partial model are multiplied and normalized w.r.t. the corresponding number of models obtained from the enumeration step. The drawback of this strategy is that the complexity of the enumeration step is repeated at each query.

The second method is to exploit the efficiency of the *WMC* task on the circuit and compute first the unnormalized weight of the root and then correct it (Algorithm 8). For each total choice ω s.t. $n = \#(\omega) > 1$ and $\omega \models \varphi$, the unnormalized evaluation of the circuit overcounts the weight of its models, because they are not multiplied by $\hat{w}(M_\omega) = \frac{1}{n}$. Remember that the unnormalized weight of a model is the probability of the corresponding total choice: $WMC(M_\omega) = WMC(\omega) = P(\omega)$, therefore the unnormalized weight is such that

$$WMC(M_\omega) = \hat{w}(M_\omega) \cdot WMC(\omega) + (1 - \hat{w}(M_\omega)) \cdot WMC(\omega)$$

Since $\hat{w}(M_\omega) = \frac{1}{n}$ and $WMC(\omega) = P(\omega)$, each weight the of models $M_\omega \in MOD(\mathscr{L}, \omega)$ is overcounted by

$$(1 - \hat{w}(M_\omega)) \cdot WMC(\omega) = \frac{P(\omega) \cdot (n-1)}{n}$$

Removing such weight from the unnormalized one, gives the probability defined by SMProbLog's semantics:

$$\widehat{WMC}(M_\omega) = WMC(M_\omega) - \frac{P(\omega) \cdot (n-1)}{n} = P(\omega) - \frac{P(\omega) \cdot (n-1)}{n} = \frac{P(\omega)}{n} = \hat{P}(M_\omega)$$

The overhead of this method per query lies in the iteration over the total choices with multiple models to retrieve and apply the different normalization constants. On the other hand, there is no additional cost in the operations performed at a node level while traversing the circuit, in contrast to the Cartesian product operations in the first method. Algorithm 8 describes the correction of the weight for any formula φ, where only the models and total choices compatible with φ contribute to $WMC(\varphi)$ with unnormalized weights. In Section 6.4 we show experimentally that exploiting the efficiency of the *WMC* task results in the evaluation step being significantly faster than the enumeration step, whose complexity reflects on the first method.

Since the circuit describes only the stable models for consistent total choices, in the case of inconsistent programs we derive the probability of an inconsistency by removing from 1 the weight (probability) of the consistent total choices, as described in Section 6.1.

Algorithm 8 Evaluation step schema

Precondition: $\varphi = q \wedge E = e$
Postcondition: $P(q \mid E = e)$
  $w \leftarrow WMC(\varphi)$
  for ω **in** $\Omega_{\mathscr{L}}$ **s.t.** $n = \#(\omega) > 1, \omega \models E = e$ **do**
    for $M_\omega \in MOD(\mathscr{L}, \omega)$ **s.t.** $M_\omega \models \varphi$ **do**
      $w = w - \frac{P(\omega)\cdot(n-1)}{n}$
  return w

6.3.2 Learning Task

In this section we show that the expectation-maximization learning algorithm implemented in ProbLog2 (Gutmann et al. 2011) is correct also for consistent SMProbLog programs. With inconsistent programs, in fact, learning is not possible in a traditional EM setting, because we cannot associate a selection of probabilistic facts to an interpretation of inconsistent atoms. If one of the atoms is inconsistent then all other atoms are inconsistent. Therefore, we can estimate $\mathbb{P}(\mathscr{L} \models \bot)$ by counting the proportion of inconsistent observations, but we cannot learn how that estimate is distributed across the total choices causing an inconsistency. This information is essential, in this learning setting, to estimate the probability that an atom is chosen. (Cussens 2001) extends the EM framework in *stochastic logic programs* (Muggleton et al. 1996) to account for failed derivation paths. In future work, investigating the applicability of his *failure adjusted maximization* algorithm to inconsistent probabilistic logic programs might offer a solution to overcome the limits of a traditional EM framework on inconsistent SMProbLog programs. On the other hand, we show that in the case of consistent SMProbLog programs we can learn the probabilities of the program in both total and partial observability.

Total observability. The correctness under total observability is guaranteed by the fact that probabilisitic facts are independent. In fact, in total observability each interpretation $I_m \in \mathbb{I}$ observes the truth value of each atom and probabilistic fact of $\mathscr{L}$. This case reduces to counting the number of true occurrences of each probabilistic fact in the interpretations $\mathbb{I}$. We clarify this by analyzing Equation 6.1 from Gutmann et al. (2011) which is also implemented in SMProbLog. Let $\hat{p_n}$ be the estimate for $p_n :: f_n$. Let $\theta^m_{n,j}$ be the j-th possible grounding substitution for f_n in the interpretation I_m. Interpretations I_j thus represent a possible combination of ground substitutions for the probabilistic facts. Let J^m_n be the number of such substitutions, and $Z_n = \sum_{m=1}^{M} J^m_n$

is the total number of ground instances of f_n in all training examples:

$$\hat{p_n} = \frac{1}{Z_n} \sum_{m=1}^{M} \sum_{j=1}^{J_n^m} \delta_{n,j}^m \qquad \text{where} \qquad \delta_{n,j}^m = \begin{cases} 1 \text{ if } f_n \theta_{n,j}^m \in I_m \\ 0 \text{ else} \end{cases} \tag{6.1}$$

The outermost sum ranges over all observed interpretations and the innermost sum counts each different observation of a ground instance of f_n. Each number of ground substitutions observed is then divided by the total number of ground substitutions. The estimate $\hat{p_n}$ is thus defined as the proportion of ground substitutions observed over all interpretations observed. This is correct for SMProbLog programs because the observation of a probabilistic fact remains independent of the rest of the program for each interpretation.

Partial observability. The correctness under partial observability is guaranteed by the fact that the parameter updates rely on probabilistic inference, which is performed under SMProbLog's semantics and thus returns correct probability estimates. In the partially observable case, where each interpretation I_m observes the truth value of a subset of $\mathscr{L}$, the parameter update iteration relies on probabilistic inference to update the likelihood of a fact given I_m. At each iteration k the parameters from the previous iteration $k-1$ are used in $\mathscr{L}$ to compute the conditional expectation of the parameter given the interpretations, until convergence. The number of observations $\delta_{n,j}^m$ in (6.1) is replaced by the conditional expectation under the current model $\mathbb{E}[\delta_{n,j}^m | I_m]$:

$$\hat{p_n} = \frac{1}{Z_n} \sum_{m=1}^{M} \sum_{j=1}^{J_n^m} \mathbb{E}[\delta_{n,j}^m | I_m] \tag{6.2}$$

To compute the conditional expectation values we query the model for $\mathbb{P}(f_n)$ at each step under the current parameters estimate. This means that the normalization w.r.t. multiple stable models is incorporated in the conditional probabilities computed by the inference task, therefore also the parameter estimate is normalized w.r.t. the different stable models that may correspond to a given partial observation.

6.4 Experiments

The goal of our experiments is to establish the feasibility of our approach, and to answer the following questions:

Q1) What is the computational cost of the different steps in the pipeline for SMProbLog semantics on the inference task?

Q2) Given a set of observations of accepted arguments generated from a known ProbLog program, can we learn the degrees of belief of the agent that modelled the original program?

Q3) How does the inference computational cost reflect on learning?

Q1) Inference. We consider a variation of a typical PLP example where a set of people has a certain probability of having asthma or being stressed, and stress leads with some probability to smoking:

```
asthma(X)← person(X).
0.3::stress(X)← person(X).
0.4::smokes(X)← stress(X).
```

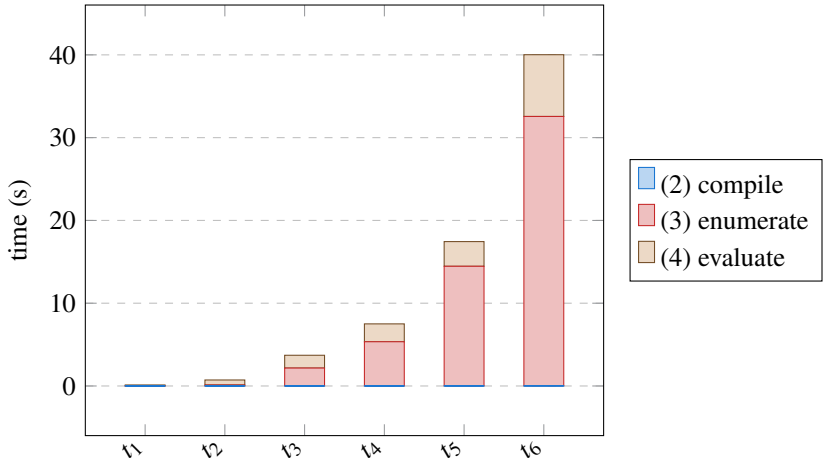
People are related by an influence relation: if a person smokes and influences to some extent another one, then the other person will smoke, and if someone smokes there is a probability to suffer from asthma. If someone suffers from asthma then the person does not smoke (an example of a cycle with negation).

```
smokes(X)← influences(Y,X), smokes(Y).
0.4::asthma(X)← smokes(X).
¬smokes(X)← asthma(X).
```

With this last rule we add a cycle through negation to a classical standard benchmark used in the Statistical Relational Learning community. This benchmark is often used to study the scaling capabilities of frameworks by growing the size of the domain of people. Let R be the aforementioned rules, we consider examples with an increasing number of people and relationships:

- $t_1 = R \cup \{person(1).\ person(2).\ 0.3 :: influences(1,2).\ 0.6 :: influences(2,1).\}$
- $t_2 = t_1 \cup \{person(3).\}$
- $t_3 = t_2 \cup \{person(4).\}$
- $t_4 = t_3 \cup \{0.2 :: influences(2,3).\}$
- $t_5 = t_4 \cup \{0.7 :: influences(3,4).\}$
- $t_6 = t_5 \cup \{0.9 :: influences(4,1).\}$

Figure 6.6 shows the running time of SMProbLog on the different benchmarks for the queries $query(smokes(X))$. and $query(asthma(X))$. The cost of grounding (step 1) is negligible, hence we omit it from the plot. This experiment answers question Q1: as



Benchmark	t_1	t_2	t_3	t_4	t_5	t_6
# prob. facts	10	14	18	19	20	21
# nodes circuit	156	192	228	462	750	1465

Figure 6.6: Inference time on benchmarks.

expected, the enumeration step dominates the running time of SMProbLog and it is directly related to the size of the circuit. A larger circuit is likely to present a higher number of Cartesian products in the enumeration step, hence a higher computational cost is expected. This experiment confirms empirically the expected exponential cost of computing weights for the SMProbLog semantics on a standard d-DNNF, in contrast with the polynomial time required for the traditional *WMC* task. At the same time, it provides an example of a problem where the optimization of the evaluation step presented in Section 6.3.1 prevents the repetition of the cost of the enumeration step. The cost of the evaluation step is in fact consistently lower than the enumeration step. Finally, we remark that the approach based on knowledge compilation allows us to answer an increasing number of (ground) queries: 4 for t_1, 6 for t_2 and 8 for the remaining benchmarks. This affects the evaluation time, but the grounding, compilation and enumeration steps are executed once, regardless of the number of queries.

Q2) Learning. We answer question 2 by considering a dataset of 283 argument graphs (Stede, Afantenos, et al. 2016) and by deriving from each annotation of the dataset a bipolar argument graph. We attach to arguments and relations random probabilities to reflect an agent's belief in each, as we did for the running example, Example 5.2. The graphs contain from 4 to 11 nodes (average 5.3) with an average of 1.4 attacks and 3 supports per graph.

Given this dataset of probabilistic argument graphs, we test learning as follows. For each argumentation graph modelled in a program $\mathscr{L}$ we obtain the model to be learned $\mathscr{L}^*$ by replacing each probability with a (randomly initialized) learnable parameter $t(_)$, similarly to Example 2.8. The programs $\mathscr{L}^*$ contain on average 10 learnable parameters divided between probabilistic facts (arguments' biases) and probabilistic rules (relations' belief). We use $\mathscr{L}$ to sample n observations of the argumentation graph by means of the sampling tool from ProbLog2. Then, we use $\mathscr{L}^*$ to learn the original probabilities from the samples. We evaluate the accuracy of the learned parameter by considering the Mean Absolute Error (MAE) of the learned probabilities in $\mathscr{L}^*$ compared to the original ones in $\mathscr{L}$. We run the learning test with increasing size of the samples, i.e. $n \in \{50, 100, 150, 200, 250, 300\}$, and with an upper bound of 100 EM iterations.

Note that we are in the case of partial observability, as the parameters are attached to the predicates *bias*, arg_{pos} and arg_{neg}, but we observe only the outcome in the form of the predicates *arg*. We evaluate the quality of the learned programs with the mean absolute error (MAE), summarized in Figure 6.7. From the experiments we observe that the MAE decreases with the increase of the sample size, that is, the more observations from the original distribution are provided, the higher is the accuracy in learning the original probabilities. At the same time when at least 100 samples are provided, the MAE drops below 10%. Therefore, we can can answer positively to our question about learnability of beliefs in argument graphs with SMProbLog.

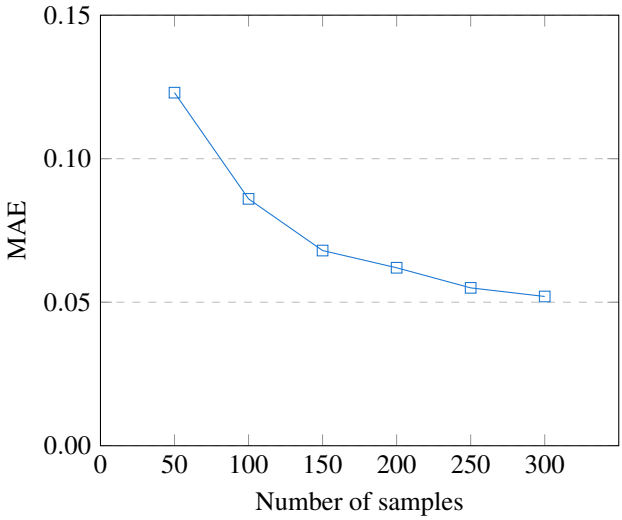


Figure 6.7: Mean absolute error by number of samples.

Q3) Learning vs. inference. Under the same experimental setup of *Q2*, we now analyze the time required to learn the parameters with respect to the cost of inference, which is part of the EM learning algorithm (Equation 6.2). There are two dimensions that influence the time required for learning with respect to inference, which we represent in Figure 6.8. The first is the cost of updating the probabilities for an increasing number of observations. In Figure 6.8 we show the running time (left scale) of EM learning on increasing sample sizes in $100, 150, 200, 250, 300$. The second is the intrinsic cost of inference with respect to the complexity of the program. We measure the complexity of the program by measuring the size of the compiled circuit, represented in Figure 6.8 with a dashed line (right scale).

The experiment shows that the second dimension is the main factor in determining the time required for learning, which follows the same trend of inference shown in Figure 6.6. Since we replace all probabilistic facts with learnable parameters, the x-axis of the two figures are comparable. The dashed line represents the average size of the circuit for a set of programs with a given number of learnable parameters (x-axis). As expected from Equation 6.2, the experiment shows a strong correlation between the inference cost on an increasingly large circuit and the learning time. In particular, we observe that the average learning time with 17 parameters decreases from 15, because in the dataset the average size of the circuits with 17 parameters is also smaller. The first dimension of the experiment shows that on smaller models the difference between the sample sizes (coloured lines) is negligible, while on more complex models the

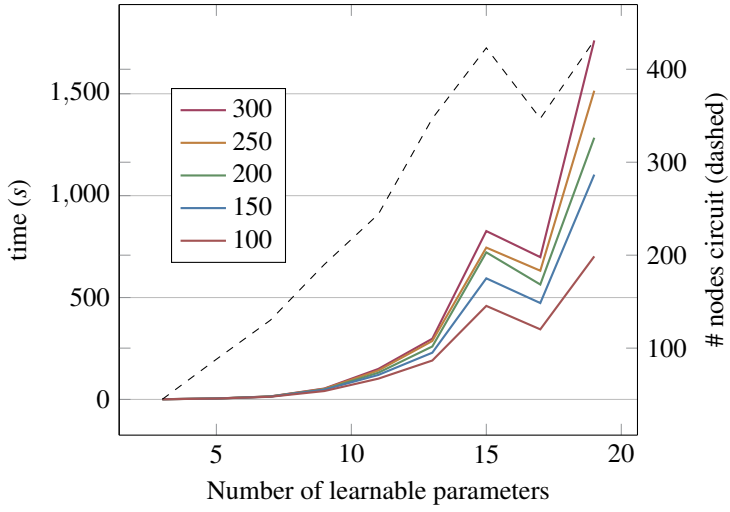
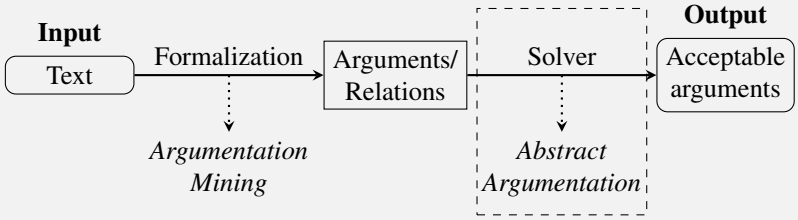


Figure 6.8: Mean running time by number of parameters and size of circuit (dashed line) on increasing number of observations (coloured lines).

differences spread out. This is plausible because on more complex models the cost of inference is higher and thus the repetition of a larger cost over more examples becomes more evident. However, the impact of a higher number of observation on the total time required for learning is clearly inferior to a bigger (more complex) logical circuit.

Intermezzo 2: Efficient Knowledge Compilation Beyond Weighted Model Counting



Algebraic Prolog (Kimmig, Van den Broeck, et al. 2011), algebraic model counting (AMC) (Kimmig, Van den Broeck, et al. 2017a) and algebraic answer set counting (Eiter et al. 2021) define general frameworks based on semirings to express and solve quantitative *first level* problems, which compute *one* aggregate over all models, e.g., counting the number of models (WMC), or summing or maximizing values associated with them, such as probabilities or utilities. Kimmig, Van den Broeck, et al. (2017a) showed that we can solve first level problems by compiling the logic program into a *tractable circuit representation*, on which evaluating an AMC task is in polynomial time if the semiring operations have constant cost.

However, many interesting tasks require two kinds of aggregation, and thus are *second level* problems that go beyond AMC. SMProbLog programs belong to this category because SMProbLog's semantics introduce a second aggregation operation, that is, normalization, along with summing. Other examples include Maximum A Posteriori (MAP) inference in probabilistic programs, which involves maximizing over some variables while summing over others, and optimization tasks in decision-theoretic or constrained probabilistic programming languages such as DTProbLog (Derkinderen and De Raedt 2020; Van den Broeck et al. 2010) and SC-ProbLog (Latour et al. 2017).

Example 6.21. Consider the following ProbLog program:

```
0.4::a.
0.6::b.
c:-a.
d:-b.
e ← ~f.
```

```
f ← ∼e.
```

Here SMProbLog semantics are required to reason over the multiple stable models induced by the cycle over `e` and `f`. In this case the probability of each possible world has to be normalized by the corresponding count of stable models, 2. To compute the probability of an atom we thus solve a second level problem: we sum (first operation) the normalized (second operation) probabilities of the possible worlds where the atom is true (Definition 36).

As in first order problems, MAP, DTProbLog and SMProbLog inference are 2AMC can be expressed in a semiring-based unifying framework for second level quantitative problems, second level algebraic model counting (2AMC).

Definition 37. *A* 2AMC *instance is a tuple* $A = (\mathscr{T}, \mathbf{X}_I, \mathbf{X}_O, \alpha_I, \alpha_O, \mathscr{S}_I, \mathscr{S}_O, t)$*, where*

- $\mathscr{T}$ *is a propositional theory,*
- $(\mathbf{X}_I, \mathbf{X}_O)$ *is a partition of the variables in* $\mathscr{T}$*,*
- $\mathscr{S}_j = (S_j, \oplus^j, \otimes^j, e_{\oplus^j}, e_{\otimes^j})$ *for* $j \in \{I, O\}$ *is a commutative semiring,*
- $\alpha_j : lit(\mathbf{X}_j) \to S_j$ *for* $j \in \{I, O\}$ *is a labeling function for literals, and*
- $t : S_I \to S_O$ *is a weight transformation function that respects* $t(e_{\oplus^I}) = e_{\oplus^O}$*.*

The value *of A, denoted by* $\mathrm{2AMC}(A)$*, is defined as*

$$\mathrm{2AMC}(A) = \bigoplus^{O}_{\mathbf{x}_O \in int(\mathbf{X}_O)} \bigotimes^{O}_{x \in \mathbf{x}_O} \alpha_O(x) \otimes^O t\left(\bigoplus^{I}_{\mathbf{x}_I \in mod(\mathscr{T}|_{\mathbf{x}_O})} \bigotimes^{I}_{y \in \mathbf{x}_I} \alpha_I(y)\right).$$

O corresponds to the *outer* semiring that correspond to the first type of aggregation, while I corresponds to the second. t converts the values from the inner semiring to the outer semiring.

Example 6.22. Computing SMProbLog semantics requires (1) counting the number of models for a given total choice and those that satisfy the query , and (2) summing the normalized probabilities. Thus, we have $\mathbf{X}_O = F$ and $\mathbf{X}_I = HB(\mathscr{T}) \setminus F$, . The transformation function is given by $t((n_1, n_2)) = \frac{n_1}{n_2}$. The outer problem then corresponds to usual probabilistic inference, i.e., $\mathscr{S}_O = ([0,1], +, \cdot, 0, 1)$, and α_O assigns p and $1-p$ to the positive and negative literals for probabilistic facts $p :: f$, respectively, and 1 to all other literals.

While second level problems stay hard on general tractable circuit representations, as we showed in Section 6.4, DTProbLog and SC-ProbLog

are known to be polynomial time on **X**-*constrained* SDDs. The key idea is to ensure that all variables of the *outer* aggregation appear before those of the *inner* one in the circuit so that we can perform both aggregations sequentially. This, however, comes at a high cost: circuits respecting this constraint may be exponentially larger than non-constrained ones, resulting in significantly slower inference. Additionally, certain optimization techniques used in knowledge compilation, such as unit propagation, may cause constraint violations. 2AMC is thus a hard problem, even harder than #SAT or AMC in general, as it imposes significant constraints on variable orders in knowledge compilation.

However, theoretical results show that these constraints can be weakened by exploiting definedness of variables. A variable Y is defined by a set of variables $\mathbf{X}$ and a propositional theory $\mathscr{T}$ if the value of Y is functionally determined by the assignment to $\mathbf{X}$ in every satisfying assignment to $\mathscr{T}$. E.g., a is defined by b in the theory $\{a \leftrightarrow b\}$. Informally, if a variable participating in the inner aggregation becomes defined by the variables of the outer aggregation at any point during compilation, we can move that variable to the outer aggregation. This can allow for exponentially smaller circuits and consequently faster evaluation, and additionally justifies the use of unit propagation.

In practice, this allow us to (i) introduce a strategy to construct variable orders for compilation into $\mathbf{X}/\mathbf{D}$-first sd-DNNFs with a-priori guarantees on complexity, and (ii) to use unit propagation to decide literals earlier than specified by the variable order during compilation. This is done by considering *tree decompositions* of the primal graph of the theory $\mathscr{T}$ in CNF, that is, the graph where the nodes are variables and an edges connect variables appearing in the same clause. We adapt the strategies for generating good variable orders statically from tree decompositions to the constrained setting where we need to ensure that the tree decomposition satisfies an additional property that allows compilation to consider independently the non-defined inner variables and the outer variables.

Experimental evaluation shows that (ii) generally improves the performance and (i) can boost it when many variables are defined. Furthermore, we see that compilation usually performs much better than an enumeration based approach to solve 2AMC and that drastically smaller circuits can be possible. Last but not least, the extensions of aspmc and SMProbLog are competitive with PITA and ProbLog, the state of the art solvers for MAP, MEU and SMProbLog inference for logic programs, and even exhibit improved performance on MEU and SMProbLog.

This intermezzo is based on the previously published paper:

R. Kiesel, P. Totis, and A. Kimmig (2022). "Efficient Knowledge Compilation Beyond Weighted Model Counting". In: *Theory Pract. Log. Program.* 22.4, pp. 505–522. DOI: 10.1017/S147106842200014X. URL: https://doi.org/10.1017/S147106842200014X

6.5 Related Work

Both three-valued well-founded model semantics and stable model semantics have been considered to extend traditional PLP framework's semantics: Hadjichristodoulou and Warren (2012) presents an extension of PRISM based on three-valued well-founded models. When considering general normal logic programs, well-founded semantics (Gelder et al. 1991) is a three-valued semantics, that is, logical atoms can be true, false or undefined. A three-valued interpretation is a pair $I = (T, F)$ where T and F are disjoint subsets of the Herbrand base $HB(\mathscr{L})$ of the program $\mathscr{L}$. The (ground) atoms in T (resp. F) are interpreted as *true* (resp. *false*) in I. The remaining atoms in $HB(\mathscr{L})\setminus(T \cup F)$ are said to be *undefined*. A three-valued model is thus a three-valued interpretation which satisfies all the rules of the program. Our approach is different because with stable model semantics we assign a truth value to the non-deterministic choices that well-founded semantics would label as undefined. At the same time we use the third value to denote the absence of stable models.

On the other hand, the application of stable model semantics in PLP is represented by probabilistic answer set programming (ASP) (Cozman and Mauà 2017, 2020). Table 6.2 summarizes the differences between the frameworks, which emerge on the definition of the probability distribution and the definition of model of a possible world.

Relation with probabilistic ASP. The key difference between SMProbLog and probabilistic ASP frameworks is that our semantics is based on the distribution semantics, that is, we define a probability distribution over models from a probability distribution over total choices. On the contrary, probabilistic ASP languages such as P-Log (Baral et al. 2009) and LP^{MLN} Lee and Wang (2016) use the probability labels to directly define a (globally normalized) probability distribution over the models of a (derived) program. This difference allows us to preserve the marginal probability of independent facts in the joint distribution in the presence of multiple stable models, and to measure the probability of an inconsistency in a program. In fact,

Framework	Distribution semantics	Model semantics	Logic values	Models per total choice
ProbLog2 (Fierens et al. 2011)	✓	Well-founded	2	1
WF-PRISM (Hadjichristodoulou and Warren 2012)	✓	Well-founded	3	1
P-Log (Baral et al. 2009)	✗	Stable models	2	n.a.
LP^{MLN} (Lee and Wang 2016)	✗	Stable models	2	n.a.
SMProbLog	✓	Stable models	2^*	≥ 1

Table 6.2 : PLP frameworks comparison. n.a. = not applicable. 2^*= the logic semantics is two-valued stable models, but we introduce a third value for total choices with 0 stable models.

following the probabilistic ASP approach can give counterintuitive results as we show in Example 6.23.

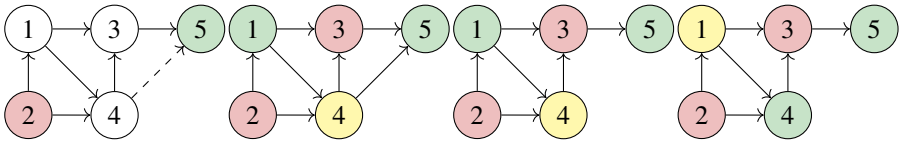
Example 6.23. In Example 6.7, if we associate to each stable model M_ω the probability of the corresponding total choice $P(\omega)$, the sum of the probabilities of the stable models is $5 \cdot 0.25 = 1.25$. A normalization w.r.t. all answer sets, as P-Log does, leads to a probability distribution over stable models $\hat{P}'(M_\omega) = 0.25/1.25 = 0.2$ for all M_ω. Then the probability of a query *burglary* is $\mathbb{P}(burglary) = \hat{P}'(M_{\omega_1}) + \hat{P}'(M_{\omega_2}) = 0.4$. This means that the marginal probability of *burglary* in the joint distribution is lower than the prior 0.5 despite no epistemic influence is defined on *burglary*. In our approach, the marginal probability of that fact being true under the final distribution equals the fact's label, and is thus directly interpretable. Similarly, when some possible worlds are inconsistent, the corresponding probability mass is not extended to a model, but rather a global normalization redistributes uniformly such mass to the valid models. For this reason, in Example 6.5 a global normalization would assign all the probability mass to ω_4, i.e. $\hat{P}'(\omega_4) = \frac{0.25}{0.25} = 1$. This because $\{right\}$ is the only possible stable model hence its weight, 0.25, is also the sum of all possible stable models. Therefore, all atoms but *right* would be interpreted false and *right* would be interpreted true with probability 1, thus making the presence of logical inconsistencies unclear.

Example 6.24. In Example 6.2 we have a similar situation: by weighing each stable model with the probability labels of the literals in the model we have that the sum of the weights of the models is 1.42 and that $\hat{P}'(M_{\omega_1}) = \frac{0.12}{1.42} = 0.085$, $\hat{P}'(M_{\omega_2}) = \frac{0.18}{1.42} =$

0.1268, $\hat{P}'(M_{\omega_3}) = \frac{0.28}{1.42} = 0.1972$, $\hat{P}'(M^1_{\omega_4}) = \hat{P}'(M^2_{\omega_4}) = \frac{0.42}{1.42} = 0,2958$. Therefore, the probabilities of *too_expensive_X* and *too_noisy_Y* are respectively $\hat{P}'(M_{\omega_2}) + \hat{P}'(M^1_{\omega_4}) + \hat{P}'(M^2_{\omega_4}) = 0.7183$ and $\hat{P}'(M_{\omega_3}) + \hat{P}'(M^1_{\omega_4}) + \hat{P}'(M^2_{\omega_4}) = 0.7887$. Therefore, the probability of probabilistic facts in the inferred distribution increases w.r.t. the original declared value. This happens with a globally normalized definition of the probability of the models and multiple stable models per total choice. In this thesis we focus on computing exact probabilities as Baral et al. (2009) and Lee and Wang (2016) rather than defining an interval of probabilities for each atom as in *credal semantics* (Lukasiewicz 1998). Cozman and Mauà (2020) remarks the difference between a credal semantics approach and a distribution semantics approach like ours. They emphasize that in distribution semantics a probabilistic fact $p :: f$. the fact f is imposed with probability p and discarded with probability $1-p$, while their approach distinguishes between taking f with probability p and $\neg f$ with probability $1-p$. Moreover, credal semantics are defined only for consistent logic programs. Recent work (Rocha and Cozman 2022) extends credal semantics to L-Credal semantics to handle inconsistencies. We illustrate the differences by means of the graph colouring example from Cozman and Mauà (2016), other comparisons are also present in (Rocha and Cozman 2022).

Example 6.25.

```
coloredBy(V,red)← ∼coloredBy(V, yellow),
    ∼coloredBy(V, green), vertex(V).
coloredBy(V,yellow) ← ∼coloredBy(V,red),
    ∼coloredBy(V, green), vertex(V).
coloredBy(V, green) ← ∼coloredBy(V,red),
    ∼coloredBy(V, yellow), vertex(V).
noClash← ∼noClash, edge(V,U),
    coloredBy(V,C), coloredBy(U,C).
edge(1,4).  edge(2,1).  edge(2,4).  edge(3,5).
edge(4,3).  edge(1,3).  0.5::edge(4,5).
coloredBy(2, red).  coloredBy(5, green).
```



(i) Graph with 0.5 :: $edge(4,5)$ (ii) One coloring with $edge(4,5)$. (iii) First coloring without $edge(4,5)$. (iv) Second coloring without $edge(4,5)$.

Figure 6.9: Possible colorings of Example 6.25

.

Credal semantics here define probability intervals based on the stable models of the two cases where $edge(4,5)$ is present (one model) or not (two models). The possible colorings are represented in Figure 6.9. Credal semantics define a lower ($\underline{\mathbb{P}}$) and upper ($\overline{\mathbb{P}}$) probability for each node: $\underline{\mathbb{P}}(coloredBy(1,yellow)) = 0$, $\overline{\mathbb{P}}(coloredBy(1,yellow)) = \frac{1}{2}$, $\underline{\mathbb{P}}(coloredBy(4,yellow)) = \frac{1}{2}$, $\overline{\mathbb{P}}(coloredBy(4,yellow)) = 1$, and $\underline{\mathbb{P}}(coloredBy(3,red)) = \overline{\mathbb{P}}(coloredBy(3,red)) = 1$. On the contrary with our semantics we compute exact values: $\mathbb{P}(coloredBy(1,yellow)) = 0.25$, $\mathbb{P}(coloredBy(4,yellow)) = 0.75$, $\mathbb{P}(coloredBy(3,red)) = 1$. These values derive from the probability of 0.5 of the coloring with $edge(4,5)$ and the probability of $\frac{0.5}{2}$ of each of the two possible coloring without $edge(4,5)$.

Relations with traditional PLP. Our approach generalizes traditional PLP frameworks on programs without function symbols: when a program defines total choices corresponding to exactly one two-valued well-founded model, then the semantics agree on both models and probability. As for the model, if a normal logic program has a total two-valued well-founded model, then the model is the unique stable model (Gelder et al. 1991). Thus, in this case $|M(\omega)| = 1$ for each total choice ω, from which it follows that $\hat{P}(M_\omega) = P(\omega)$ for the single stable model $M_\omega \in MOD(\mathscr{L},\omega)$ ($MOD(\mathscr{L},\omega) = \{M_\omega\}$). This means that the probability of the model is the probability of the corresponding total choice as in (probabilistic) two-valued well-founded semantics. The one-to-one correspondence between total choices and stable models also guarantees that $\mathbb{P}((\emptyset,\emptyset)) = 0$.

For this reason, our pipeline is also a generalization of ProbLog2's pipeline. ProbLog2 (Fierens et al. 2011) reduces the probabilistic inference task to a *weighted model counting problem* (Cadoli and Donini 1997) in three steps:

1. Compute the relevant grounding, that is, ground only the part of the program necessary to answer the query.
2. Convert the ground rules into an equivalent boolean formula (CNF).
3. Compile the boolean formula into an arithmetic circuit to efficiently compute the weighted model count of the formula.

The grounding procedure in ProbLog2 computes the *relevant* ground program w.r.t. the given queries Q and evidence E. The relevant ground program is obtained by applying SLD resolution to prove all atoms in $Q \cup E$. However, in SMProbLog an atom can be derived from a logical choice whose atoms are not directly justified by some (probabilistic) fact, therefore in this case meaningful parts of the program would not be included in the relevant ground program. This is why we use a standard bottom-up

technique. Aziz et al. (2015) apply stable model counting techniques in ProbLog (ASPProbLog). In particular, they replace the DSHARP compiler (Muise et al. 2012) used to implement step 3 with a stable model knowledge compiler (an extension of DSHARP itself). Step 2 is also modified, since the CNF conversion is skipped and a simple representation of the ground rules is provided to the stable model knowledge compiler. However, ASPProbLog does not change the semantics of ProbLog, and invalid programs according to the original semantics are still rejected. Moreover, ASPProbLog does not implement the syntactical features of ProbLog2 on which our approach to argumentation is based, such as negation in the heads or annotated rules.

Distribution semantics has been proven by Sato (1995) to be well-defined for definite programs with function symbols, and later Riguzzi (2015) proves this for the case of normal programs with two-valued well-founded models. Studying the semantics of SMProbLog programs with function symbols is thus an interesting direction for future work.

6.6 Conclusion

In this chapter we addressed our research question **Q4** by presenting a novel PLP semantics and a PLP system based on it, SMProbLog, to reason on programs where logical rules can induce non-deterministic choices. In fact, traditional PLP frameworks are based on the assumption that non-determinism in the domain is fully captured by probabilistic facts, and once a total choice is fixed, logic rules determine exactly one interpretation of the logical atoms. This is not the case for programs defining cyclic relations through negation, such as the programs for argumentation problems presented in Chapter 5. SMProbLog can do inference and learning on these problems, however inference is a harder task because it is an instance of a 2AMC problem. Knowledge compilation techniques constraining the variable order allow us to still solve the task in time polynomial in the size of the circuit, but the additional constraints can lead to exponentially larger circuits. However, the exponential growth of the circuit can be avoided by exploiting the definedness when selecting the variable order in the circuit.

Chapter 7

Conclusion

In this section we summarize the key contributions of this thesis and discuss the directions for future work.

Summary and Results

The key contributions of this thesis introduce novel modelling and reasoning techniques for automated reasoning over combinatorics math problems and argumentation problems. The desirable properties of an automated reasoning system in a two-step approach are:

1. to rely on a formal language that is close to the expressivity of natural language,
2. to rely on efficient reasoning techniques that minimize the number of reasoning steps required to compute the answer to the query.

The contributions narrow the gap between the state of the art and the desirable properties of an automated reasoning system by answering the following research questions:

Q1) What are the key characteristics of a modelling language for encoding combinatorics math problems?

Q2) How can models for combinatorics math problems be efficiently solved?

Q3) Can probabilistic logic programming model probabilistic argumentation problems?

Q4) Can probabilistic logic programming systems efficiently solve probabilistic argumentation models?

The key results presented in this thesis answering our research questions are the following.

Combinatorics Math Word Problems

Past work is not sufficient to provide tools to model effectively combinatorics math problems and efficiently compute the solution. The application of an end-to-end neural approach to probability math problems (Intermezzo 1) discouraged pursuing the same line of thought for combinatorics problems, because they present many similarities with probability problems in terms of objects, logic, and configuration complexity. Here emerge the typical limits of neural networks in terms of generalizability, interpretability and logical reasoning. Designing a two-step approach with the existing automated reasoning tools presented relevant limitations in both modelling and reasoning, which we addressed with the following contributions.

Q1) Chapter 3 answers **Q1** by identifying three key primitives for modelling combinatorics math problems: 1) multisets 2) configurations and 3) constraints. Multisets should be first-class citizens of the modelling language, i.e. used as arguments and assigned to variables, because combinatorics math problems are often concerned with objects indistinguishable from each other. Dedicated statements for specifying configurations should also be present, because an indirect formulation of how objects are arranged may be convoluted and may preclude efficient inference. For the same reason, constraints should be expressed at a high level of abstraction, because this maintains the formal language closer to natural language and at the same time facilitates the identification of useful symmetries for reasoning. We applied these principles in the design of CoLa, a language to model combinatorics math problems which can represent 88% of a real-world combinatorics math word problems dataset.

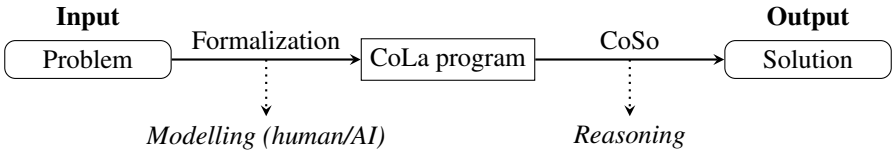


Figure 7.1: CoLa and CoSo to solve combinatorial problems.

Q2) Chapter 4 answers **Q2** by 1) applying to counting CSPs the lifted reasoning principles developed in the context of probabilistic inference, and by 2) describing their implementation in CoSo, a novel solver for combinatorics math problems encoded in CoLa. The existing propositional methods are based on grounding or enumeration, which is a highly inefficient strategy for combinatorics math problems, while lifted reasoning is closer to how humans typically solve these problems, by reasoning on groups of objects at once and by splitting the main problem into subproblems. The experiments show that this method offers significantly more efficient solving performances than the existing propositional tools, both on real-world problems and more challenging synthetic benchmarks.

Probabilistic Argumentation Problems

Past work on probabilistic argumentation problems focused on designing semantics and reasoning tools specific to argumentation. Ad-hoc argumentation tools present little flexibility and modularity as the introduction of new argumentation aspects in the framework requires new semantics and reasoning techniques. On the other hand, PLP systems offer general-purpose languages to model logical reasoning under uncertainty in AI, paired with inference and learning algorithms rooted in the principle of modelling a joint probability distribution of random variables. Investigating whether it is possible to apply these algorithms to probabilistic argumentation problems required us to study how to model argumentation problems by means of PLP and whether this was sufficient to apply the inference and learning tools of PLP to argumentation.

Q3) Chapter 5 answers **Q3** by proposing a novel mapping from probabilistic argumentation problems to PLP programs. The key novelty is the use of negation in the head and the corresponding inhibition effect to model attacks to beliefs. The

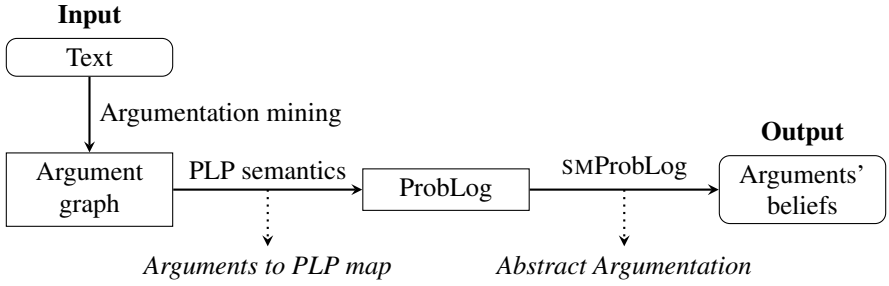


Figure 7.2: Declarative Programming approach to beliefs in argumentation.

advantage of this approach compared to existing ad-hoc probabilistic argumentation frameworks is the declarative nature of the encoding. In fact, the resulting programs are modular and generalizable, in the sense that introducing new argumentation concepts in the framework does not require to redefine the semantics of the language. At the same time, this mapping connects probabilistic argumentation problems to the traditional methods developed to model beliefs and uncertainty in AI.

Q4) Chapter 6 answers **Q4** by presenting a novel PLP framework, SMProbLog, which implements probabilistic inference and learning tools for the type of programs obtained from the proposed mapping of probabilistic argumentation problems. In fact, existing traditional PLP systems make assumptions on the type of input programs too restrictive to reason over any program from the mapping. SMProbLog relaxes one of these assumptions to reason over the programs not supported by traditional semantics. We show that conventional inference and learning techniques can be adapted for this class of programs, but at an efficiency cost. The second Intermezzo explains how part of this efficiency can be recovered by framing inference in SMProbLog as a 2AMC problem and exploiting the corresponding reasoning techniques. Constraining the variable order in knowledge compilation allows us to solve 2AMC problems in polynomial time as AMC

Perspectives and Future Work

Solving math word problems and argumentation problems embraces different fields of AI: natural language understanding, knowledge representation and reasoning, problem-solving. These are considered to be AI-complete problems (Shapiro 2003), that is, problems that are as difficult as the entire AI problem. It is thus unsurprising that, despite the progress presented in this thesis, there are still many open problems and gaps to be filled in the realization of AI systems for solving any math word problem or argumentation problem.

We decomposed the problem of intelligent behaviour in this setting into two steps. The fist step, mapping natural language to a formal language, links natural language understanding with knowledge representation and reasoning. The second step, computing the solution, links knowledge representation and reasoning with problem-solving. Therefore, intelligent behaviour is achieved by combining different cognitive skills for the AI-complete problems at hand. We identified such skills as mapping, modelling, and reasoning. These are the core challenges that future AI systems need to address to claim a satisfactory resolution of automated reasoning problems in natural language.

Mapping. Closing the distance between natural and formal language with more expressive knowledge representation formats is a common challenge for future automated reasoning systems based on a two-step approach.
Moreover, future work on combinatorics math problems should focus on the interpretation of boolean operators and the quantities associated to properties and sets, because these were the critical errors on probability math word problems that can affect the combinatorics setting as well.
The better performances deriving from the use of pre-trained language models in the two-step approach suggests that future improvements in such models will contribute to a higher accuracy in mapping the text in natural language to the correct formalization.
An interesting direction for future work on argumentation problems is to map natural language directly to the corresponding probabilistic logic program, instead of relying on argument graphs as an intermediate knowledge representation format.

Modelling. Future work on combinatorics math problems and argumentation problems should improve expressivity and extensiblity of the formal language without compromising the ability to reason about it.
CoLa is not easily extensible at the moment because there are no primitives to build arbitrary configurations: identifying these primitives and extending CoLa with the corresponding statements is an interesting direction for future work. Two further lines for future work improving expressivity are 1) the expansion of the constraint language towards the traditional constraint modelling languages and 2) the addition of statements to model multiple configurations at the same time. In this case, the core challenge is to match the knowledge representation format with suitable lifted reasoning techniques.
As regards argumentation, future work should go in the direction of exploiting the extensibility of probabilistic logic programs, to expand the mapping to argumentation frameworks more complex than basic (probabilistic) argument graphs, for example structured argumentation (Besnard, Garcìa, et al. 2014).

Reasoning. The core reasoning challenge that future work has to address is optimizing the efficiency of inference. Efficiency is related to connecting different levels of reasoning and the ability of switching between them by choosing at any time the most lifted one.
For CoSo this means integrating the lifted techniques with traditional propositional methods to reason over non-liftable problems. For instance, the solver for probability math word problems from Dries et al. (2017) relies on ProbLog for lower-level probabilistic reasoning. An integration of the two systems where the solver for combinatorics problems is used for high-level combinatorial reasoning in probability problems is also an interesting direction for future work.
As for argumentation, we showed how expanding the expressivity of the language increases the intrinsic complexity of reasoning, therefore any fundamental progress in

the efficiency of probabilistic inference would represent a significant contribution to the efficiency of reasoning about argumentation problems under uncertainty. SMProbLog introduces new semantics that can be computed by means of knowledge compilation, therefore an interesting direction for future work is to define novel probabilistic argumentation frameworks based on alternative model (that is, acceptability) semantics, a common operation in argumentation. Then, a further compelling question would be whether the PLP inference tools and algorithms can be adapted to compute other types of model semantics. A positive answer to these questions would allow us to use SMProbLog as a general-purpose framework for proabilistic inference under distribution semantics, with the flexibility of changing the desired (deterministic) model semantics.

As for a one-step approach, specializing language models on math word problems to generate explainable solutions is an interesting direction for future work. From the examples presented in Appendix C it is clear that an unspecialized model can already produce interesting reasoning steps, however by repeating the same prompt it is also clear that the output is highly random. In fact, not only the solution, but also the explanation of the reasoning steps, can be completely different from one question to the other for the same prompt. This suggests that current language models cannot systematically reason over the semantics of the description, and injecting logical reasoning capabilities in such models will be one of the main AI challenges for the next years.

Appendix A

CoLa grammar

In this chapter we report the formal grammar for the language CoLa.

```
S -> program
program
    -> statement
    -> statement program
statement
    -> declare_set SEMI
    -> arrangement SEMI
    -> pos_constraint SEMI
    -> count_constraint SEMI
obj_list
    -> LABEL
    -> LABEL COMMA obj_list
comp -> EQUALS
     -> LT
     -> GT
     -> GT EQUALS
     -> LT EQUALS
     -> DIFF EQUALS
base_set
    -> LABEL
    -> LABEL LSPAR NUM RSPAR
    -> UNIVERSE
set_atom
    -> base_set
```

```
    -> LRPAR set RRPAR
set_not
    -> NOT set_atom
    -> set_atom
set_inter
    -> set_not
    -> set_not INTER set_atom
set_union
    -> set_inter
    -> set UNION set_inter
set -> set_union
    -> PART
declare_set
    -> PROP LABEL EQUALS LPAR obj_list RPAR
    -> UNIVERSE LABEL EQUALS LPAR obj_list RPAR
    -> PROP UNIVERSE EQUALS LPAR obj_list RPAR
    -> LAB PROP LABEL EQUALS LPAR obj_list RPAR
    -> PROP LABEL
    -> LAB PROP LABEL
arrangement
    -> LABEL IN LPAR REPEAT set RPAR
    -> LABEL IN LPAR set RPAR
    -> LABEL IN LSPAR REPEAT set RSPAR
    -> LABEL IN LSPAR set RSPAR
    -> LABEL IN LPAR LPAR set RPAR RPAR
    -> LABEL IN LSPAR LPAR set RPAR RSPAR
pos_constraint
    -> LABEL LSPAR NUM RSPAR EQUALS set
    -> LABEL LSPAR NUM RSPAR IN set
    -> LABEL LSPAR NUM RSPAR EQUALS LPAR obj_list RPAR
count_constraint
    -> COUNT set comp NUM
    -> COUNT LRPAR count_constraint RRPAR comp NUM


NUM -> [0-9]+
LABEL -> [a-z][a-zA-Z\-\_0-9]*|[0-9]+\-[0-9\-]+
COMMA -> ","
COUNT -> "#"
DIFF -> "!"
EQUALS -> "="
GT -> ">"
```

```
LT -> "<"
INTER -> "&"
UNION -> "+"
NOT -> "¬"
LPAR -> "{"
RPAR -> "}"
LRPAR -> "("
RRPAR -> ")"
LSPAR -> "["
RSPAR -> "]"
SEMI  -> ";"
IN -> "in"
LAB -> "labelled"
PART -> "part"
PROP -> "property"
REPEAT -> "repeated"
UNIVERSE -> "universe"
```

Appendix B

Automatic Translation of CoLa to ASP and Essence

This appendix describes the automatic translation from the CoLa encoding to the ASP and ESSENCE formats that are adopted in the experiments, along with some examples of the translations. CoSo can output each format when invoked with the corresponding option. The CNF (sharpSAT) encodings are obtained by first generating the ASP encoding and then translating it to CNF with existing tools[1]. The CNF encodings consist of several thousand lines, therefore we do not report them here. The translation is divided in two parts: the translation of the universe (and properties) and the translation of the configuration with the corresponding constraints.

ASP In the translation to ASP, if the universe is a set, then properties are predicates with a domain where each object is a different number. For example, the domain of 12 TVs of which 3 are defective in **P3** is translated to `universe(1..12).`, `defective(1..3).` If the universe is a multiset, for example the letters BANANA in **P5**, then each object is paired with the number of indistinguishable copies, e.g. `letter("A", 3). letter("N", 2). letter("B", 1).` Along with the declared universe, the same mapping is applied on each property used in the constraints.

Example B.1. The ASP encoding of the set of objects in **P3** automatically generated from CoLa is as follows:

```
% Multisets and universe specification.
tvs(1..3).
```

[1] https://research.ics.aalto.fi/software/asp/lp2sat/

```
tvs(4..12).
universe(X) :- tvs(X).
defective(1..3).
universe(X) :- defective(X).
sf_0(1..3).
universe(X) :- sf_0(X).
```

The ASP encoding of the set of objects in **P5** automatically generated from CoLa is as follows:

```
% Multisets and universe specification.
bs_0("B", 1).
bs(X) :- bs_0(X, _).
universe(X) :- bs(X).
as_0("A",3).
as(X) :- as_0(X, _).
universe(X) :- as(X).
ns_0("N",2).
ns(X) :- ns_0(X, _).
universe(X) :- ns(X).
```

Level 1 configurations of size n are defined by means of a predicate of arity n where each variable is a slot of the configuration with the corresponding domain. A choice rule selects exactly one valid configuration per stable model, such that the number of stable models is the number of valid configurations (solutions). For subsets and multisets symmetry breaking constraints (respectively $<$ and $\leq$) are added to remove exchangeable solutions.

Example B.2.

The ASP encoding of the configuration in **P3** automatically generated from CoLa is as follows:

```
% Configuration declaration
subset_guess_5(A,B,C,D,E) :-
    universe(A), universe(B), universe(C),
    universe(D), universe(E),
    A<B, B<C, C<D, D<E.
% Choose one subset for each answer set
1{subset_5(A,B,C,D,E):subset_guess_5(A,B,C,D,E)}1.
% Auxiliary predicate for aggregates
used_5(X,0) :- subset_5(X, _, _, _, _).
used_5(X,1) :- subset_5(_, X, _, _, _).
```

```
used_5(X,2) :- subset_5(_, _, X, _, _).
used_5(X,3) :- subset_5(_, _, _, X, _).
used_5(X,4) :- subset_5(_, _, _, _, X).
```

If the universe is a multiset or the configuration does not contain repetitions, constraints with counting predicates are added to ensure that no configuration contains more indistinguishable copies of an objects than allowed.

Example B.3. The ASP encoding of the configuration in **P5** automatically generated from CoLa is as follows:

```
% Configuration declaration
perm_guess_6(A,B,C,D,E,F) :-
    universe(A), universe(B), universe(C),
    universe(D), universe(E), universe(F).
% Choose one permutation for each answer set
1{perm_6(A,B,C,D,E,F):perm_guess_6(A,B,C,D,E,F)}1.
% Auxiliary predicate for aggregates
used_6(X,0) :- perm_6(X, _, _, _, _, _).
used_6(X,1) :- perm_6(_, X, _, _, _, _).
used_6(X,2) :- perm_6(_, _, X, _, _, _).
used_6(X,3) :- perm_6(_, _, _, X, _, _).
used_6(X,4) :- perm_6(_, _, _, _, X, _).
used_6(X,5) :- perm_6(_, _, _, _, _, X).
% Do not use more copies than the available
:- bs_0(S,SN), C = #count{N:used_6(S,N)}, C>SN.
:- as_0(S,SN), C = #count{N:used_6(S,N)}, C>SN.
:- ns_0(S,SN), C = #count{N:used_6(S,N)}, C>SN.
```

Compositions are modelled with a constant identifying each set and a predicate
`1{put(E,N,P): int(N), N<=EN} 1 :- property(E, EN), part(P).`
chooses for each entity `E` how many of the available indistinguishable copies to put in each part. Constraints ensure that each part is non-empty:
`:- part(P), #count{E,N:put(E,N,P), N>0}==0.`
and all entities are distributed:
`:- property(E,EN), #sum{N,P:put(E,N,P),part(P)}!=EN.`
Finally, positional constraints are imposed in level 1 on the domain of the variables representing the objects, and in level 2 on the facts denoted by `put/3`.

Example B.4. If we wanted to constrain position 2 in **P5** to be an “a”, the declaration of the permutation would look like:

```
perm_guess_6(A,B,C,D,E,F) :-
    universe(A), as(B), universe(C),
    universe(D), universe(E), universe(F).
```

The encoding of counting constraints is similar to the constraints for the number of indistinguishable copies. The variable object in the `used` predicate is counted if it belongs to the predicate denoting the set formula in the constraint.

Example B.5. The ASP encoding of the constraint in **P3** automatically generated from CoLa is as follows:

```
% Exclude invalid values
:- C = #count{N:used_5(S,N),sf_0(S)}, C=0.
:- C = #count{N:used_5(S,N),sf_0(S)}, C=1.
```

Essence In the translation to ESSENCE the universe and properties are mapped to an enumerated type. For example on **P5**, this would entail `letting the universe be new type enum {b, a, n}`. If the universe is a multiset, then the definition of a function counting the number of indistinguishable copies is added, e.g. `letting f be function(b->1, a->3, n->2)`.

Example B.6.

The ESSENCE encoding of the universe in **P3** automatically generated from CoLa is as follows:

```
letting universe be new type enum
    { td_0, td_1, td_2,
      tnd_0, tnd_1, tnd_2, tnd_3, tnd_4,
      tnd_5, tnd_6, tnd_7, tnd_8 }
letting f_univ be function(
    td_0 --> 1, td_1 --> 1, td_2 --> 1,
    tnd_0 --> 1, tnd_1 --> 1, tnd_2 --> 1,
    tnd_3 --> 1, tnd_4 --> 1, tnd_5 --> 1,
    tnd_6 --> 1, tnd_7 --> 1, tnd_8 --> 1)
letting defective be { td_0, td_1, td_2 }
letting df_0 be { td_0, td_1, td_2 }
```

The ESSENCE encoding of the universe for **P5** automatically generated from CoLa is as follows:

```
% Multisets declaration: entities and
```

```
% corresponding copies
letting universe be new type enum { a, n, b }
letting f_univ be function(a --> 3, n --> 2, b --> 1)
letting bs be { b }
letting as be { a }
letting ns be { n }
```

As we remarked in Section 3.1, currently it is not possible to declare multisets to be the domain of a configuration, e.g. `letting S be mset(0,1,1,1)`, `find s : sequence (size l) of S`. Level 1 configurations of size n are defined by means of a sequence (ordered) or multiset (unordered) of length n where each variable is a slot of the configuration with the corresponding domain. If the configuration forbids repetitions, then an upper bound is added on the number of copies of the objects. This set to be less or equal to those available, such as:

```
forAll e: universe. sum([1 | i: int(1..l),
configuration(i)=e])<=f(e)
```

Example B.7. The ESSENCE encoding of the configuration in **P3** automatically generated from CoLa is as follows:

```
% Configuration declaration
letting vals_0 be { 2,3,4,5 }
letting l_5 be 5
find conf_5 : mset (size l_5) of tvs
```

The ESSENCE encoding of the configuration in **P5** automatically generated from CoLa is as follows:

```
% Configuration declaration
letting l_6 be 6
find conf_6 : sequence (size l_6) of universe
such that
    % no repetition
    forAll e: universe.
        sum([1|i: int(1..l_6),conf_6(i)=e])<=f_univ(e)
```

Compositions are modelled with a matrix indexed by the universe and the parts, such that for each pair (e, p) the number in the matrix specifies how many copies of entity e belong to part p. Also in this case additional constraints impose the non-emptiness of all parts:

```
forAll p:myparts. sum([put[e,p] | e:universe]) > 0
```

and the requirement that all objects are distributed over the parts:
`forAll e: universe. sum([put[e,p] | p:myparts]) = f(e).`
Positional and counting constraints are applied similarly. For the former, a constraint restricts the domain of the variable in the given position, e.g. `conf(i) in property`. For the latter, the frequency of the objects belonging to the constraint property is checked, e.g. `sum([freq(conf, i) | i: universe, i in property]) in valid_values`.

Example B.8. The ESSENCE encoding of the constraint in **P3** automatically generated from CoLa is as follows:

```
find conf_5 : mset (size l_5) of tvs
%Configuration constraints
such that
    % no repetition
    forAll e: tvs.
        forAll e: universe. freq(conf_5,e) <= f_tvs(e)
    % Counting constraints
    /\ sum([freq(conf_5,i)|i:tvs,i in df_0]) in vals_0
```

Problem 1

We present as an additional representative example of the translations to ASP and ESSENCE the encodings for **P1**.

Example B.9. The ASP encoding of **P1** automatically generated from CoLa is as follows:

```
shapes_0("5",3).                    % shapes
shapes(X) :- shapes_0(X, _).
shapes_1("1", 1).
shapes(X) :- shapes_1(X, _).
shapes_2("2", 1).
shapes(X) :- shapes_2(X, _).
shapes_3("3", 1).
shapes(X) :- shapes_3(X, _).
shapes_4("4", 1).
shapes(X) :- shapes_4(X, _).
universe(X) :- shapes(X).
red_0("1", 1).                      % red
red(X) :- red_0(X, _).
```

```
red_1("3", 1).
red(X) :- red_1(X, _).
universe(X) :- red(X).
blue_0("2", 1).                  % blue
blue(X) :- blue_0(X, _).
blue_1("4", 1).
blue(X) :- blue_1(X, _).
universe(X) :- blue(X).
green_0("5",3).                  % green
green(X) :- green_0(X, _).
universe(X) :- green(X).
triangle_0("5",3).               % triangles
triangle(X) :- triangle_0(X, _).
triangle_1("3", 1).
triangle(X) :- triangle_1(X, _).
triangle_2("4", 1).
triangle(X) :- triangle_2(X, _).
universe(X) :- triangle(X).
squares_0("1", 1).               % squares
squares(X) :- squares_0(X, _).
squares_1("2", 1).
squares(X) :- squares_1(X, _).
universe(X) :- squares(X).
df_0_0("1", 1).     % two squares set formula
df_0(X) :- df_0_0(X, _).
df_0_1("2", 1).
df_0(X) :- df_0_1(X, _).
universe(X) :- df_0(X).
% each variable corresponds to an object
% of the permutation
perm_guess_4(A,B,C,D) :-
    universe(A), pf_1(B), universe(C), universe(D).
pf_1_0("5",3).                   % second green object
pf_1(X) :- pf_1_0(X, _).
universe(X) :- pf_1(X).
% each stable model is a different solution
1{permutation_4(A,B,C,D):perm_guess_4(A,B,C,D)}1.
% multiset constraints: do not use more copies
% than the available
used_4(X,0) :- permutation_4(X, _, _, _).
used_4(X,1) :- permutation_4(_, X, _, _).
used_4(X,2) :- permutation_4(_, _, X, _).
```

```
used_4(X,3) :- permutation_4(_, _, _, X).
:- shapes_0(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- shapes_1(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- shapes_2(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- shapes_3(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- shapes_4(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- red_0(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- red_1(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- blue_0(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- blue_1(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- green_0(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- triangle_0(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- triangle_1(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- triangle_2(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- squares_0(S,SN), C = #count{N:used_4(S,N)}, C>SN.
:- squares_1(S,SN), C = #count{N:used_4(S,N)}, C>SN.
% counting constraints: used squares are exactly 2
:- C = #count{N:used_4(S,N),df_0(S)}, C=0.
:- C = #count{N:used_4(S,N),df_0(S)}, C=1.
:- C = #count{N:used_4(S,N),df_0(S)}, C=3.
:- C = #count{N:used_4(S,N),df_0(S)}, C=4.
```

Example B.10. The ESSENCE encoding of **P1** automatically generated from CoLa is as follows:

```
% Multisets declaration: entities and
% corresponding copies
letting universe be new type enum {e_5,e_1,e_2,e_3,e_4}
letting f_univ be function(
    e_5-->3, e_1-->1, e_2-->1, e_3--> 1, e_4-->1
)
letting red be { e_1, e_3 }
letting blue be { e_2, e_4 }
letting green be { e_5 }
letting triangle be { e_5, e_3, e_4 }
letting squares be { e_1, e_2 }
letting pf_1_0 be { e_5 }
letting df_0 be { e_1, e_2 }
letting l_4 be 4
letting vals_0 be { 2 }
% configuration declaration
find conf_4 : sequence (size l_4) of shapes
```

```
such that
  forAll e: shapes.
        % do not use more copies than available
    sum([1|i: int(1..l_4), conf_4(i)=e])<=f_shapes(e)
    % positional constraint
  /\ conf_4(2) in pf_1_0
    % counting constraint
  /\sum([1|i:int(1..l_4),conf_4(i) in df_0]) in vals_0
```

Appendix C

Asking ChatGPT to solve combinatorics math word problems

In this appendix we report the answers of ChatGPT (OpenAI 2022), a recent conversational language model, to some of the combinatorics math word problems considered in this thesis. We report two different answers for each prompt to remark how the reasoning steps and solutions proposed highly vary from one answer to the other.

Example C.1. We ask ChatGPT to solve **P3**:

Prompt 1: A shipment of 12 different TVs contains 3 defective ones. In how many ways can a hotel purchase 5 of these TVs and receive at least 2 of the defective ones?
ChatGPT: *It is not possible to answer this question without additional information. In order to solve this problem, it is necessary to know the number of defective TVs in the shipment of 12 TVs, as well as the number of TVs that the hotel is purchasing. Additionally, it is necessary to know whether the TVs are distinguishable or indistinguishable, as this will affect the number of possible combinations. Without this information, it is not possible to determine the number of ways in which the hotel can purchase 5 TVs and receive at least 2 defective ones.*
Analysis: Here an answer is not given on the basis of missing information. It is not correct that the number of defective and purchased TVs is not given, however it is correct that it is necessary to know whether TVs are distinguishable or not and that this is not specified. Here TVs are assumed to be distinguishable, but this is an example of background knowledge that is not explicitly given.

Prompt 2: A shipment of 12 distinguihshable TVs contains 3 defective ones. In how many ways can a hotel purchase 5 of these TVs and receive at least 2 of the defective ones?
ChatGPT 2: *In this situation, there are 3 defective TVs and 9 non-defective TVs in the shipment of 12 TVs. The hotel is purchasing 5 TVs, so they must choose at least 2 of the defective TVs and at most 3 of the non-defective TVS. There are 3 ways to choose 2 of the defective TVs:*

- *Choose 2 of the defective TVs and 3 of the non-defective TVs.*
- *Choose 2 of the defective TVs and 2 of the non-defective TVs.*
- *Choose 2 of the defective TVs and 1 of the non-defective TVs.*

For each of these 3 ways there are $\binom{9}{3}$ *ways to choose the non-defective TVs, and* $\binom{9}{2} = 36$ *ways to choose the non-defective TVs. Therefore, there are a total of* $3 \cdot 84 + 3 \cdot 36 = 372$ *ways for the hotel to purchase 5 TVs and receive at least 2 of the defective ones.*
Analysis: The result is incorrect due to the errors in decomposing the problem into subproblems. First, only the first case forms a purchase of five TVs. Second, the proposed count is not the same for each of the three cases.

Example C.2. We ask ChatGPT to solve **P4**:

Prompt: Fourteen construction workers are to be assigned to three different tasks. Seven workers are needed for mixing cement, five for laying bricks, and two for carrying the bricks to the brick layers. In how many different ways can the workers be assigned to these tasks?
ChatGPT: *If the workers are to be assigned to the tasks in distinct groups, then the number of ways to assign the workers is equal to the number of ways to partition a set of 14 objects into three non-empty subsets, which is equal to the number of ways to partition a set of 11 objects into three non-empty subsets. This is because the first subset of workers will consist of 7 workers, and the remaining workers will be partitioned into two non-empty subsets.*

To count the number of ways to partition a set of 11 objects into three non-empty subsets, we can use the following equation:

$$\binom{11}{3} = \frac{11!}{3!8!} = 165$$

Therefore, there are 165 ways to assign the 14 construction workers to the three tasks.
Analysis: Assigning 14 workers to three groups is not equal to assigning 11 workers, moreover, the constraints about the size of each group are ignored.

ChatGPT 2: *To solve this problem, we can use the concept of combinations. Since we have 14 workers and need to assign them to 3 tasks, with 7 workers needed for the first task, 5 for the second, and 2 for the third, the number of ways in which the workers can be assigned to the tasks is equal to the number of combinations of 14 items taken 7 at a time, 5 at a time, and 2 at a time. This can be calculated using the following formula:*

14!/(7! * 5! * 2!) = 14 * 13 * 12 * 11 * 10 * 9 * 8 / (7 * 6 * 5 * 4 * 3 * 2 * 1 * 5 * 4 * 3 * 2 * 1 * 2 * 1) = 3003.

Therefore, there are 3003 different ways in which the workers can be assigned to the three tasks.
Analysis: Here the sizes of the groups are used but the counting rule applied is incorrect: the proposed solution is the count of permutations where the workers of each group are indistinguishable.

Example C.3. We ask ChatGPT to solve **P5**:

Prompt: In how many distinguishable ways can the letters in B A N A N A be written?
ChatGPT: *There are 6 letters in the word BANANA, and two of them are A's. This means that the word BANANA has 3 distinguishable letters: B, N, and A.*

We can arrange these 3 distinguishable letters in 6! *ways, but we must divide by the number of ways that the A's can be arranged amongst themselves, which is 2!, and the number of ways that the N's can be arranged amongst themselves, which is also 2!. This gives us the following:*

$$\frac{6!}{2!2!} = \frac{720}{4} = 180$$

Therefore, there are 180 distinguishable ways to arrange the letters in BANANA.
Analysis: The solution is almost correct, but the count of the number of A's is 2 instead of 3, therefore despite the correct reasoning and counting rule the result is wrong.

ChatGPT 2: *We use the Principle of Inclusion-Exclusion. There are* 6! *ways to arrange the six letters of the word BANANA. However, since there are two As, two Ns, and one B, we have overcounted the arrangements in which the two As are together, the two Ns are together, or the B is by itself. There are* $(5!)^2$ *ways to arrange the letters so that the two As are together and the two Ns are together (regardless of the order in which the As and Ns appear), and there are* 5! *ways to arrange the letters so that the B is by itself.*

However, we have undercounted the arrangements in which the two As are together, the two Ns are together, and the B is by itself. There are $(4!)^2$ *such arrangements, so our final answer is*

$$6! - 2 \cdot (5!)^2 + (4!)^2 = \boxed{735}.$$

Analysis: The reasoning in this case is faulty as the included-excluded cases are wrong. In the motivation if this choice it is not clear what the meaning of two letters being together is. Moreover, the number of counted A's is two again.

Example C.4. We ask ChatGPT to solve **P1**:

Prompt: A kit of toy shapes contains five triangles and two squares. One triangle and one square are red. Another triangle and the other square are blue and the remaining triangles are green. In how many different rows of four objects can the shapes be arranged if the two squares are included and the second object is green?

ChatGPT: *There are two possible arrangements for the red and blue shapes: either the red triangle is followed by the blue square, or the blue triangle is followed by the red square. In either case, the green triangles can be arranged in* $5!$ *ways. Therefore, there are a total of* $2 \times 5! = 120$ *ways to arrange the red, blue, and green shapes so that the second object is green.*|

To arrange the four objects in a row, we can first arrange the two squares in 2! ways, and then choose which of the two possible arrangements of the red, blue, and green shapes to use. This gives us a total of $2! \times 2 = 4$ *ways to arrange the four objects in a row.*

Therefore, there are a total of $120 \times 4 = 480$ *ways to arrange the shapes so that the two squares are included and the second object is green.*

Analysis: There are several incorrect reasoning steps: first, the possible orders between red and blue shapes is not 2. Second, the number of arrangements for the triangles assumes that they are all distinguishable and ignores the size of the row. Third, the constraints are ignored.

ChatGPT 2: *There are three ways to choose the first object in the row: a red triangle, a blue triangle, or a green triangle. Once the first object is chosen, there are two ways to choose the second object in the row: a red square or a blue square. If the first object is green, there are two ways to choose the second object in the row: a red square or a blue square. Thus, the total number of rows that can be formed is* $3 \cdot 2 \cdot 2 = \boxed{12}$.

Analysis: The constraints are again ignored, moreover the possible choices for each position are incorrect: for the first choice the squares are ignored, while for the second object the choice is incorrectly limited to squares. This step is then reiterated and accounted in the solution but the remaining positions are not considered.

Example C.5. We ask ChatGPT to solve **P2**:

Prompt: Given the same set of shapes, in how many ways can the objects be divided into three (non-empty) groups such that the green objects all belong to the same group?
Answer: *We can divide the shapes into three groups in 3! ways. However, within each group, the red and blue shapes can be arranged in 2! ways, and the green triangles can be arranged in 5! ways. Therefore, the total number of ways to divide the shapes into three groups such that the green objects all belong to the same group is*

$$3! \times 2! \times 5! = 720$$

Therefore, there are 720 ways to divide the shapes into three non-empty groups such that the green objects all belong to the same group.
Analysis: The first counting rule is wrong both in the type and the number used. It is correct to account for the distinguishability of red and blue shapes but the number of green triangles is then wrong. In any case, it is incorrect to multiply these arrangements because the ordered cases should be excluded from the count rather than included.

ChatGPT 2: *There are three ways to choose which group contains the three green triangles: the first group, the second group, or the third group. In each of these cases, the remaining two groups each contain two objects, and the objects in each group can be chosen in* $\binom{4}{2} = 6$ *ways. Hence, the total number of ways to divide the objects into three groups is* $3 \cdot 6 = \boxed{18}$.
Analysis: The first step is correct, however only one of the four possible distributions of the remaining objects is accounted for.

Bibliography

Akgün, Ö., A. M. Frisch, I. P. Gent, C. Jefferson, I. Miguel, and P. Nightingale (2022). "Conjure: Automatic Generation of Constraint Models from Problem Specifications". In: *Artif. Intell.* 310, p. 103751. DOI: 10.1016/j.artint.2022.103751. URL: https://doi.org/10.1016/j.artint.2022.103751.

Alfano, G., S. Greco, F. Parisi, and I. Trubitsyna (2020). "On the Semantics of Abstract Argumentation Frameworks: A Logic Programming Approach". In: *Theory and Practice of Logic Programming* 20.5, pp. 703–718. DOI: 10.1017/S1471068420000253. URL: https://doi.org/10.1017/S1471068420000253.

Amgoud, L., J. Ben-Naim, D. Doder, and S. Vesic (2017). "Acceptability Semantics for Weighted Argumentation Frameworks". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Ed. by C. Sierra. ijcai.org, pp. 56–62. ISBN: 978-0-9992411-0-3. DOI: 10.24963/ijcai.2017/9. URL: https://doi.org/10.24963/ijcai.2017/9.

Amgoud, L., C. Cayrol, and M.-C. Lagasquie-Schiex (2004). "On the bipolarity in argumentation frameworks". In: *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, Canada, June 6-8, 2004, Proceedings*. Ed. by J. P. Delgrande and T. Schaub, pp. 1–9. ISBN: 92-990021-0-X. URL: http://www.pims.math.ca/science/2004/NMR/papers/paper01.pdf.

Amini, A., S. Gabriel, S. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi (2019). "MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Association for Computational Linguistics, pp. 2357–2367. ISBN: 978-1-950737-13-0. DOI: 10.18653/v1/n19-1245. URL: https://doi.org/10.18653/v1/n19-1245.

Aziz, R. A., G. Chu, C. J. Muise, and P. J. Stuckey (2015). "Stable Model Counting and Its Application in Probabilistic Logic Programming". In: *Proceedings of the Twenty-*

Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. Ed. by B. Bonet and S. Koenig. AAAI Press, pp. 3468–3474. ISBN: 978-1-57735-698-1. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9709.

Baral, C., M. Gelfond, and J. N. Rushton (2009). "Probabilistic reasoning with answer sets". In: *Theory and Practice of Logic Programming* 9.1, pp. 57–144. DOI: 10.1017/S1471068408003645. URL: https://doi.org/10.1017/S1471068408003645.

Baroni, P., A. Rago, and F. Toni (2019). "From fine-grained properties to broad principles for gradual argumentation: A principled spectrum". In: *International Journal of Approximate Reasoning* 105, pp. 252–286. DOI: 10.1016/j.ijar.2018.11.019. URL: https://doi.org/10.1016/j.ijar.2018.11.019.

Berry, D. A. (2006). "Bayesian clinical trials". In: *Nature reviews Drug discovery* 5.1, pp. 27–36.

Besnard, P., C. Cayrol, and M.-C. Lagasquie-Schiex (2020). "Logical theories and abstract argumentation: A survey of existing works". In: *Argument Comput.* 11.1-2, pp. 41–102. DOI: 10.3233/AAC-190476. URL: https://doi.org/10.3233/AAC-190476.

Besnard, P., A. J. Garcìa, A. Hunter, S. Modgil, H. Prakken, G. R. Simari, and F. Toni (2014). "Introduction to structured argumentation". In: *Argument & Computation* 5.1, pp. 1–4. DOI: 10.1080/19462166.2013.869764. URL: https://doi.org/10.1080/19462166.2013.869764.

Bondarenko, A., F. Toni, and R. A. Kowalski (1993). "An Assumption-Based Framework for Non-Monotonic Reasoning". In: *Logic Programming and Non-monotonic Reasoning, Proceedings of the Second International Workshop, Lisbon, Portugal, June 1993*. Ed. by L. M. Pereira and A. Nerode. MIT Press, pp. 171–189. ISBN: 0-262-66083-0.

Brewka, G., J. Dix, and K. Konolige (1997). *Nonmonotonic Reasoning: An Overview*. Vol. 73. CSLI Lecture Notes. CSLI.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Bulatov, A. A. (2013). "The complexity of the counting constraint satisfaction problem". In: *J. ACM* 60.5, 34:1–34:41. DOI: 10.1145/2528400. URL: https://doi.org/10.1145/2528400.

Cadoli, M. and F. M. Donini (1997). “A Survey on Knowledge Compilation”. In: *AI Commun.* 10.3-4, pp. 137–150. URL: http://content.iospress.com/articles/ai-communications/aic133.

Caminada, M., S. Sà, J. F. L. Alcántara, and W. Dvoràk (2015). “On the equivalence between logic programming semantics and argumentation semantics”. In: *International Journal of Approximate Reasoning* 58, pp. 87–111. DOI: 10.1016/j.ijar.2014.12.004. URL: https://doi.org/10.1016/j.ijar.2014.12.004.

Cayrol, C. and M.-C. Lagasquie-Schiex (2005). “Graduality in Argumentation”. In: *Journal of Artificial Intelligence Research* 23, pp. 245–297. DOI: 10.1613/jair.1411. URL: https://doi.org/10.1613/jair.1411.

Charniak, E. (1991). “Bayesian Networks without Tears”. In: *AI Mag.* 12.4, pp. 50–63. DOI: 10.1609/aimag.v12i4.918. URL: https://doi.org/10.1609/aimag.v12i4.918.

Chesani, F., P. Mello, and M. Milano (2017). “Solving Mathematical Puzzles: A Challenging Competition for AI”. In: *AI Mag.* 38.3, pp. 83–96. DOI: 10.1609/aimag.v38i3.2736. URL: https://doi.org/10.1609/aimag.v38i3.2736.

Choi, A., D. Kisa, and A. Darwiche (2013). “Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams”. In: *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 12th European Conference, ECSQARU 2013, Utrecht, The Netherlands, July 8-10, 2013. Proceedings*. Ed. by L. C. van der Gaag. Vol. 7958. Lecture Notes in Computer Science. Springer, pp. 121–132. ISBN: 978-3-642-39090-6. DOI: 10.1007/978-3-642-39091-3_11. URL: https://doi.org/10.1007/978-3-642-39091-3%5C_11.

Clark, K. L. (1977). “Negation as Failure”. In: *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d’études et de recherches de Toulouse, France, 1977*. Ed. by H. Gallaire and J. Minker. Advances in Data Base Theory. New York: Plemum Press, pp. 293–322. ISBN: 0-306-40060-X. DOI: 10.1007/978-1-4684-3384-5_11. URL: https://doi.org/10.1007/978-1-4684-3384-5%5C_11.

Cooper, G. F. (1990). “The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks”. In: *Artif. Intell.* 42.2-3, pp. 393–405. DOI: 10.1016/0004-3702(90)90060-D. URL: https://doi.org/10.1016/0004-3702(90)90060-D.

Cozman, F. G. and D. D. Mauà (2016). “The Structure and Complexity of Credal Semantics”. In: *Proceedings of the 3rd International Workshop on Probabilistic Logic Programming co-located with 26th International Conference on Inductive Logic Programming (ILP 2016), London, UK, September 3, 2016*. Ed. by A. Hommersom and S. A. Abdallah. Vol. 1661. CEUR Workshop Proceedings. CEUR-WS.org, pp. 3–14. URL: http://ceur-ws.org/Vol-1661/paper-01.pdf.

Cozman, F. G. and D. D. Mauà (2017). “On the Semantics and Complexity of Probabilistic Logic Programs”. In: *Journal of Artificial Intelligence Research* 60,

pp. 221–262. DOI: 10.1613/jair.5482. URL: https://doi.org/10.1613/jair.5482.

– (2020). "The joy of Probabilistic Answer Set Programming: Semantics, complexity, expressivity, inference". In: *International Journal of Approximate Reasoning* 125, pp. 218–239. DOI: 10.1016/j.ijar.2020.07.004. URL: https://doi.org/10.1016/j.ijar.2020.07.004.

Cussens, J. (2001). "Parameter Estimation in Stochastic Logic Programs". In: *Mach. Learn.* 44.3, pp. 245–271. DOI: 10.1023/A:1010924021315. URL: https://doi.org/10.1023/A:1010924021315.

Darwiche, A. (2004). "New Advances in Compiling CNF into Decomposable Negation Normal Form". In: *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*. Ed. by R. L. de Màntaras and L. Saitta. IOS Press, pp. 328–332. ISBN: 1-58603-452-9.

De Raedt, L., K. Kersting, S. Natarajan, and D. Poole (2016). *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. DOI: 10.2200/S00692ED1V01Y201601AIM032. URL: https://doi.org/10.2200/S00692ED1V01Y201601AIM032.

De Raedt, L., A. Kimmig, and H. Toivonen (2007). "ProbLog: A Probabilistic Prolog and Its Application in Link Discovery". In: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. Ed. by M. M. Veloso, pp. 2462–2467. URL: http://ijcai.org/Proceedings/07/Papers/396.pdf.

Derkinderen, V. and L. De Raedt (2020). "Algebraic Circuits for Decision Theoretic Inference and Learning". In: *ECAI*. Vol. 325, pp. 2569–2576.

Dries, A., A. Kimmig, J. Davis, V. Belle, and L. De Raedt (2017). "Solving Probability Problems in Natural Language". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Ed. by C. Sierra. ijcai.org, pp. 3981–3987. ISBN: 978-0-9992411-0-3. DOI: 10.24963/ijcai.2017/556. URL: https://doi.org/10.24963/ijcai.2017/556.

Dung, P. M. (1995). "On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games". In: *Artificial Intelligence* 77.2, pp. 321–358. DOI: 10.1016/0004-3702(94)00041-X. URL: https://doi.org/10.1016/0004-3702(94)00041-X.

Eiter, T., M. Hecher, and R. Kiesel (2021). "Treewidth-Aware Cycle Breaking for Algebraic Answer Set Counting". In: *KR*, pp. 269–279. DOI: 10.24963/kr.2021/26. URL: https://doi.org/10.24963/kr.2021/26.

Emden, M. H. van and R. A. Kowalski (1976). "The Semantics of Predicate Logic as a Programming Language". In: *J. ACM* 23.4, pp. 733–742. DOI: 10.1145/321978.321991. URL: https://doi.org/10.1145/321978.321991.

Ferraris, S., A. Mendelson, G. Ballesio, and T. Vercauteren (2015). *Counting sub-multisets of fixed cardinality*. DOI: 10.48550/ARXIV.1511.06142. URL: https://arxiv.org/abs/1511.06142.

Fierens, D., G. V. den Broeck, I. Thon, B. Gutmann, and L. De Raedt (2011). "Inference in Probabilistic Logic Programs using Weighted CNF's". In: *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*. Ed. by F. G. Cozman and A. Pfeffer. AUAI Press, pp. 211–220. ISBN: 978-0-9749039-7-2. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article%5C_id=2240%5C&proceeding%5C_id=27.

Finetti, B. de (1931). "Sul significato soggettivo della probabilità." In: *Fundamenta Mathematicae* 17, pp. 298–329.

Frisch, A. M., W. Harvey, C. Jefferson, B. M. Hernàndez, and I. Miguel (2008). "Essence : A constraint language for specifying combinatorial problems". In: *Constraints An Int. J.* 13.3, pp. 268–306. DOI: 10.1007/s10601-008-9047-y. URL: https://doi.org/10.1007/s10601-008-9047-y.

Frisch, A. M., C. Jefferson, B. M. Hernàndez, and I. Miguel (2005). "The Rules of Constraint Modelling". In: *IJCAI 2005, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*. Ed. by L. P. Kaelbling and A. Saffiotti. Professional Book Center, pp. 109–116. ISBN: 0938075934. URL: http://ijcai.org/Proceedings/05/Papers/1667.pdf.

Gebser, M., R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and P. Wanko (2016). "Theory Solving Made Easy with Clingo 5". In: *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*. Ed. by M. Carro, A. King, N. Saeedloei, and M. D. Vos. Vol. 52. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2:1–2:15. ISBN: 978-3-95977-007-1. DOI: 10.4230/OASIcs.ICLP.2016.2. URL: https://doi.org/10.4230/OASIcs.ICLP.2016.2.

Gebser, M., R. Kaminski, B. Kaufmann, and T. Schaub (2012). *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gelder, A. V., K. A. Ross, and J. S. Schlipf (1991). "The Well-Founded Semantics for General Logic Programs". In: *J. ACM* 38.3, pp. 620–650. DOI: 10.1145/116825.116838. URL: https://doi.org/10.1145/116825.116838.

Gelfond, M. (2008). "Answer Sets". In: *Handbook of Knowledge Representation*. Ed. by F. van Harmelen, V. Lifschitz, and B. W. Porter. Vol. 3. Foundations of Artificial Intelligence. Elsevier, pp. 285–316. ISBN: 978-0-444-52211-5. DOI: 10.1016/S1574-6526(07)03007-6. URL: https://doi.org/10.1016/S1574-6526(07)03007-6.

Gelfond, M. and V. Lifschitz (1988). "The Stable Model Semantics for Logic Programming". In: *Logic Programming, Proceedings of the Fifth International*

Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes). Ed. by R. A. Kowalski and K. A. Bowen. MIT Press, pp. 1070–1080. ISBN: 0-262-61056-6.

– (1991). "Classical Negation in Logic Programs and Disjunctive Databases". In: *New Generation Computing* 9, pp. 365–385. URL: http://www.cs.utexas.edu/users/ai-lab?gel91b.

Gent, I. P., C. Jefferson, and I. Miguel (2006). "Minion: A Fast Scalable Constraint Solver". In: *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*. Ed. by G. Brewka, S. Coradeschi, A. Perini, and P. Traverso. Vol. 141. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 98–102. ISBN: 1-58603-642-4. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=1654.

Gottlob, G., N. Leone, and F. Scarcello (2000). "A comparison of structural CSP decomposition methods". In: *Artif. Intell.* 124.2, pp. 243–282. DOI: 10.1016/S0004-3702(00)00078-3. URL: https://doi.org/10.1016/S0004-3702(00)00078-3.

Gradel, E., M. Otto, and E. Rosen (1997). "Two-variable logic with counting is decidable". In: *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, pp. 306–317. DOI: 10.1109/LICS.1997.614957.

Greco, G. and F. Scarcello (2010). "On The Power of Tree Projections: Structural Tractability of Enumerating CSP Solutions". In: *CoRR* abs/1005.1567. arXiv: 1005.1567. URL: http://arxiv.org/abs/1005.1567.

Gutmann, B., I. Thon, and L. De Raedt (2011). "Learning the Parameters of Probabilistic Logic Programs from Interpretations". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I*. Ed. by D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis. Vol. 6911. Lecture Notes in Computer Science. Springer, pp. 581–596. ISBN: 978-3-642-23779-9. DOI: 10.1007/978-3-642-23780-5_47. URL: https://doi.org/10.1007/978-3-642-23780-5%5C_47.

Hadjichristodoulou, S. and D. S. Warren (2012). "Probabilistic Logic Programming with Well-Founded Negation". In: *42nd IEEE International Symposium on Multiple-Valued Logic, ISMVL 2012, Victoria, BC, Canada, May 14-16, 2012*. Ed. by D. M. Miller and V. C. Gaudet. IEEE Computer Society, pp. 232–237. ISBN: 978-1-4673-0908-0. DOI: 10.1109/ISMVL.2012.26. URL: https://doi.org/10.1109/ISMVL.2012.26.

Hosseini, M. J., H. Hajishirzi, O. Etzioni, and N. Kushman (2014). "Learning to Solve Arithmetic Word Problems with Verb Categorization". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by A. Moschitti, B. Pang, and W. Daelemans. ACL, pp. 523–533.

ISBN: 978-1-937284-96-1. DOI: 10.3115/v1/d14-1058. URL: https://doi.org/10.3115/v1/d14-1058.

Huang, D., S. Shi, C.-Y. Lin, J. Yin, and W.-Y. Ma (Aug. 2016). "How well do Computers Solve Math Word Problems? Large-Scale Dataset Construction and Evaluation". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 887–896. DOI: 10.18653/v1/P16-1084. URL: https://www.aclweb.org/anthology/P16-1084.

Hunter, A. (2013). "A probabilistic approach to modelling uncertain logical arguments". In: *International Journal of Approximate Reasoning* 54.1, pp. 47–81. DOI: 10.1016/j.ijar.2012.08.003. URL: https://doi.org/10.1016/j.ijar.2012.08.003.

Hunter, A., S. Polberg, and M. Thimm (2018). "Epistemic Graphs for Representing and Reasoning with Positive and Negative Influences of Arguments". In: *CoRR* abs/1802.07489. arXiv: 1802.07489. URL: http://arxiv.org/abs/1802.07489.

– (2020). "Epistemic graphs for representing and reasoning with positive and negative influences of arguments". In: *Artificial Intelligence* 281, p. 103236. DOI: 10.1016/j.artint.2020.103236. URL: https://doi.org/10.1016/j.artint.2020.103236.

Jaynes, E. T. (1988). "The Relation of Bayesian and Maximum Entropy Methods". In: *Maximum-Entropy and Bayesian Methods in Science and Engineering: Foundations*. Ed. by G. J. Erickson and C. R. Smith. Dordrecht: Springer Netherlands, pp. 25–29. ISBN: 978-94-009-3049-0. DOI: 10.1007/978-94-009-3049-0_2. URL: https://doi.org/10.1007/978-94-009-3049-0_2.

Kazemi, S. M., A. Kimmig, G. Van den Broeck, and D. Poole (2016). "New Liftable Classes for First-Order Probabilistic Inference". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, pp. 3117–3125. URL: http://papers.nips.cc/paper/6603-new-liftable-classes-for-first-order-probabilistic-inference.

Kiesel, R., P. Totis, and A. Kimmig (2022). "Efficient Knowledge Compilation Beyond Weighted Model Counting". In: *Theory Pract. Log. Program.* 22.4, pp. 505–522. DOI: 10.1017/S147106842200014X. URL: https://doi.org/10.1017/S147106842200014X.

Kimmig, A., G. Van den Broeck, and L. De Raedt (2011). "An algebraic Prolog for reasoning about possible worlds". In: *AAAI*.

– (2017a). "Algebraic model counting". In: *JAL* 22, pp. 46–62. DOI: 10.1016/j.jal.2016.11.031. URL: https://doi.org/10.1016/j.jal.2016.11.031.

– (2017b). “Algebraic model counting”. In: *J. Appl. Log.* 22, pp. 46–62. DOI: 10.1016/j.jal.2016.11.031. URL: https://doi.org/10.1016/j.jal.2016.11.031.

Kisynski, J. and D. Poole (2009). “Constraint Processing in Lifted Probabilistic Inference”. In: *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*. Ed. by J. A. Bilmes and A. Y. Ng. AUAI Press, pp. 293–302. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article%5C_id=1591%5C&proceeding%5C_id=25.

Koller, D. and N. Friedman (2011). “*Probabilistic Graphical Models: Principles and Techniques* by Daphne Koller and Nir Friedman, MIT Press, 1231 pp., ISBN 0-262-01319-3”. In: *Knowl. Eng. Rev.* 26.2, pp. 237–238. DOI: 10.1017/S0269888910000275. URL: https://doi.org/10.1017/S0269888910000275.

Koncel-Kedziorski, R., H. Hajishirzi, A. Sabharwal, O. Etzioni, and S. D. Ang (2015). “Parsing Algebraic Word Problems into Equations”. In: *Transactions of the Association for Computational Linguistics* 3, pp. 585–597. DOI: 10.1162/tacl_a_00160. URL: https://www.aclweb.org/anthology/Q15-1042.

Kowalski, R. and S. W. Smoliar (1982). “Logic for problem solving”. In: *ACM SIGSOFT Softw. Eng. Notes* 7.2, pp. 61–62. DOI: 10.1145/1005937.1005947. URL: https://doi.org/10.1145/1005937.1005947.

Kowalski, R. A. and F. Toni (1996). “Abstract Argumentation”. In: *Artif. Intell. Law* 4.3-4, pp. 275–296. DOI: 10.1007/BF00118494. URL: https://doi.org/10.1007/BF00118494.

Kushman, N., Y. Artzi, L. Zettlemoyer, and R. Barzilay (June 2014). “Learning to Automatically Solve Algebra Word Problems”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 271–281. DOI: 10.3115/v1/P14-1026. URL: https://www.aclweb.org/anthology/P14-1026.

Kuzelka, O. (2021). “Weighted First-Order Model Counting in the Two-Variable Fragment With Counting Quantifiers”. In: *J. Artif. Intell. Res.* 70, pp. 1281–1307. DOI: 10.1613/jair.1.12320. URL: https://doi.org/10.1613/jair.1.12320.

Latour, A. L. D., B. Babaki, A. Dries, A. Kimmig, G. Van den Broeck, and S. Nijssen (2017). “Combining Stochastic Constraint Optimization and Probabilistic Programming - From Knowledge Compilation to Constraint Solving”. In: *CP*, pp. 495–511. DOI: 10.1007/978-3-319-66158-2_32. URL: https://doi.org/10.1007/978-3-319-66158-2%5C_32.

Lee, J. and Y. Wang (2016). “Weighted Rules under the Stable Model Semantics”. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29,*

2016. Ed. by C. Baral, J. P. Delgrande, and F. Wolter. AAAI Press, pp. 145–154. ISBN: 978-1-57735-755-1. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12901.

Legg, S. and M. Hutter (2006). "A Collection of Definitions of Intelligence". In: *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms - Proceedings of the AGI Workshop 2006 [May 20-21, 2006, Washington DC, USA]*. Ed. by B. Goertzel and P. Wang. Vol. 157. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 17–24. ISBN: 978-1-58603-758-1. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=5774.

Li, H., N. Oren, and T. J. Norman (2011). "Probabilistic Argumentation Frameworks". In: *Theorie and Applications of Formal Argumentation - First International Workshop, TAFA 2011. Barcelona, Spain, July 16-17, 2011, Revised Selected Papers*. Ed. by S. Modgil, N. Oren, and F. Toni. Vol. 7132. Lecture Notes in Computer Science. Springer, pp. 1–16. ISBN: 978-3-642-29183-8. DOI: 10.1007/978-3-642-29184-5_1. URL: https://doi.org/10.1007/978-3-642-29184-5%5C_1.

Ling, W., D. Yogatama, C. Dyer, and P. Blunsom (July 2017). "Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 158–167. DOI: 10.18653/v1/P17-1015. URL: https://www.aclweb.org/anthology/P17-1015.

Lloyd, J. W. (1994). "Practical Advtanages of Declarative Programming." In: *GULP-PRODE (1)*, pp. 18–30.

– (1987). *Foundations of Logic Programming, 2nd Edition*. Springer. ISBN: 3-540-18199-7. DOI: 10.1007/978-3-642-83189-8. URL: https://doi.org/10.1007/978-3-642-83189-8.

Lukasiewicz, T. (1998). "Probabilistic Logic Programming". In: *13th European Conference on Artificial Intelligence, Brighton, UK, August 23-28 1998, Proceedings.* Ed. by H. Prade. John Wiley and Sons, pp. 388–392. ISBN: 9780471984313.

Mandal, S., A. A. Sekh, and S. K. Naskar (2020). "Solving arithmetic word problems: A deep learning based approach". In: *J. Intell. Fuzzy Syst.* 39.2, pp. 2521–2531. DOI: 10.3233/JIFS-179911. URL: https://doi.org/10.3233/JIFS-179911.

Mantadelis, T. and S. Bistarelli (2020). "Probabilistic abstract argumentation frameworks, a possible world view". In: *International Journal of Approximate Reasoning* 119, pp. 204–219. DOI: 10.1016/j.ijar.2019.12.006. URL: https://doi.org/10.1016/j.ijar.2019.12.006.

Marcus, G. (2018). "Deep Learning: A Critical Appraisal". In: *CoRR* abs/1801.00631. arXiv: 1801.00631. URL: http://arxiv.org/abs/1801.00631.

Marriott, K., N. Nethercote, R. Rafeh, P. J. Stuckey, M. G. de la Banda, and M. Wallace (2008). "The Design of the Zinc Modelling Language". In: *Constraints An Int. J.*

13.3, pp. 229–267. DOI: 10.1007/s10601-008-9041-4. URL: https://doi.org/10.1007/s10601-008-9041-4.

Meert, W. and J. Vennekens (2014). "Inhibited Effects in CP-Logic". In: *Probabilistic Graphical Models - 7th European Workshop, PGM 2014, Utrecht, The Netherlands, September 17-19, 2014. Proceedings*. Ed. by L. C. van der Gaag and A. J. Feelders. Vol. 8754. Lecture Notes in Computer Science. Springer, pp. 350–365. ISBN: 978-3-319-11432-3. DOI: 10.1007/978-3-319-11433-0_23. URL: https://doi.org/10.1007/978-3-319-11433-0%5C_23.

Miao, S.-y., C.-C. Liang, and K.-Y. Su (July 2020). "A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 975–984. DOI: 10.18653/v1/2020.acl-main.92. URL: https://www.aclweb.org/anthology/2020.acl-main.92.

Milch, B., L. S. Zettlemoyer, K. Kersting, M. Haimes, and L. P. Kaelbling (2008). "Lifted Probabilistic Inference with Counting Formulas". In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. Ed. by D. Fox and C. P. Gomes. AAAI Press, pp. 1062–1068. ISBN: 978-1-57735-368-3. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-168.php.

Mitra, A. and C. Baral (2016). "Learning To Use Formulas To Solve Simple Arithmetic Problems". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics. ISBN: 978-1-945626-00-5. DOI: 10.18653/v1/p16-1202. URL: https://doi.org/10.18653/v1/p16-1202.

Muggleton, S. et al. (1996). "Stochastic logic programs". In: *Advances in inductive logic programming* 32, pp. 254–264.

Muise, C. J., S. A. McIlraith, J. C. Beck, and E. I. Hsu (2012). "Dsharp: Fast d-DNNF Compilation with sharpSAT". In: *Advances in Artificial Intelligence - 25th Canadian Conference on Artificial Intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28-30, 2012. Proceedings*. Ed. by L. Kosseim and D. Inkpen. Vol. 7310. Lecture Notes in Computer Science. Springer, pp. 356–361. ISBN: 978-3-642-30352-4. DOI: 10.1007/978-3-642-30353-1_36. URL: https://doi.org/10.1007/978-3-642-30353-1%5C_36.

Nethercote, N., P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack (2007). "MiniZinc: Towards a Standard CP Modelling Language". In: *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*. Ed. by C. Bessiere. Vol. 4741. Lecture Notes in Computer Science. Springer, pp. 529–543. ISBN: 978-3-540-74969-1. DOI: 10.1007/978-3-540-74970-7_38. URL: https://doi.org/10.1007/978-3-540-74970-7%5C_38.

Nielsen, S. H. and S. Parsons (2006). "A Generalization of Dung's Abstract Framework for Argumentation: Arguing with Sets of Attacking Arguments". In: *Argumentation in Multi-Agent Systems, Third International Workshop, ArgMAS 2006, Hakodate, Japan, May 8, 2006, Revised Selected and Invited Papers*. Ed. by N. Maudet, S. Parsons, and I. Rahwan. Vol. 4766. Lecture Notes in Computer Science. Springer, pp. 54–73. ISBN: 978-3-540-75525-8. DOI: 10.1007/978-3-540-75526-5_4. URL: https://doi.org/10.1007/978-3-540-75526-5%5C_4.

Niepert, M. and G. Van den Broeck (2014). "Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Quèbec City, Quèbec, Canada*. Ed. by C. E. Brodley and P. Stone. AAAI Press, pp. 2467–2475. ISBN: 978-1-57735-661-5. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8615.

Nilsson, N. (2014). *Understanding Beliefs*. The MIT Press Essential Knowledge series. MIT Press. ISBN: 9780262526432. URL: https://books.google.be/books?id=Z%5C_PvAwAAQBAJ.

Nunes, T. (2012). "Logical Reasoning and Learning". In: *Encyclopedia of the Sciences of Learning*. Ed. by N. M. Seel. Boston, MA: Springer US, pp. 2066–2069. ISBN: 978-1-4419-1428-6. DOI: 10.1007/978-1-4419-1428-6_790. URL: https://doi.org/10.1007/978-1-4419-1428-6_790.

OpenAI (2022). *ChatGPT: Optimizing Language Models for Dialogue*. https://openai.com/blog/chatgpt/. [Online; accessed 04 December 2022].

Pak, I. (2018). *Complexity problems in enumerative combinatorics*. DOI: 10.48550/ARXIV.1803.06636. URL: https://arxiv.org/abs/1803.06636.

Patel, A., S. Bhattamishra, and N. Goyal (2021). "Are NLP Models really able to Solve Simple Math Word Problems?" In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*. Ed. by K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tür, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou. Association for Computational Linguistics, pp. 2080–2094. ISBN: 978-1-954085-46-6. DOI: 10.18653/v1/2021.naacl-main.168. URL: https://doi.org/10.18653/v1/2021.naacl-main.168.

Pearl, J. (1989). *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann.

– (2006). "Two journeys into human reasoning". In: *Los Angeles: UCLA Cognitive systems Laboratory*.

Polberg, S., A. Hunter, and M. Thimm (2017). "Belief in Attacks in Epistemic Probabilistic Argumentation". In: *Scalable Uncertainty Management - 11th International Conference, SUM 2017, Granada, Spain, October 4-6, 2017, Proceedings*. Ed. by S. Moral, O. Pivert, D. Sànchez, and N. Marìn. Vol. 10564. Lecture Notes in Computer Science. Springer, pp. 223–236. ISBN: 978-3-319-67581-

7. DOI: 10.1007/978-3-319-67582-4_16. URL: https://doi.org/10.1007/978-3-319-67582-4%5C_16.

Poole, D. (2003). "First-order probabilistic inference". In: *IJCAI 2003, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. Ed. by G. Gottlob and T. Walsh. Morgan Kaufmann, pp. 985–991. URL: http://ijcai.org/Proceedings/03/Papers/142.pdf.

– (2008). "The Independent Choice Logic and Beyond". In: *Probabilistic Inductive Logic Programming - Theory and Applications*. Ed. by L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton. Vol. 4911. Lecture Notes in Computer Science. Springer, pp. 222–243. ISBN: 978-3-540-78651-1. DOI: 10.1007/978-3-540-78652-8_8. URL: https://doi.org/10.1007/978-3-540-78652-8%5C_8.

Richardson, M. and P. Domingos (2006). "Markov logic networks". In: *Machine learning* 62.1-2, pp. 107–136.

Riguzzi, F. (2015). "The Distribution Semantics is Well-Defined for All Normal Programs". In: *Proceedings of the 2nd International Workshop on Probabilistic Logic Programming co-located with 31st International Conference on Logic Programming (ICLP 2015), Cork, Ireland, August 31st, 2015*. Ed. by F. Riguzzi and J. Vennekens. Vol. 1413. CEUR Workshop Proceedings. CEUR-WS.org, pp. 69–84. URL: http://ceur-ws.org/Vol-1413/paper-06.pdf.

Rocha, V. H. N. and F. G. Cozman (2022). "A Credal Least Undefined Stable Semantics for Probabilistic Logic Programs and Probabilistic Argumentation". In: *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022*. Ed. by G. Kern-Isberner, G. Lakemeyer, and T. Meyer. ISBN: 978-1-956792-01-0. DOI: 10.24963/kr.2022. URL: https://proceedings.kr.org/2022/31/.

Rossi, F., P. van Beek, and T. Walsh, eds. (2006). *Handbook of Constraint Programming*. Vol. 2. Foundations of Artificial Intelligence. Elsevier. ISBN: 978-0-444-52726-4. URL: http://www.sciencedirect.com/science/bookseries/15746526/2.

Roy, S. and D. Roth (2018). "Mapping to Declarative Knowledge for Word Problem Solving". In: *Trans. Assoc. Comput. Linguistics* 6, pp. 159–172. URL: https://transacl.org/ojs/index.php/tacl/article/view/1319.

Saccà, D. and C. Zaniolo (1990). "Stable Models and Non-Determinism in Logic Programs with Negation". In: *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*. Ed. by D. J. Rosenkrantz and Y. Sagiv. ACM Press, pp. 205–217. ISBN: 0-89791-352-3. DOI: 10.1145/298514.298572. URL: https://doi.org/10.1145/298514.298572.

Salvo Braz, R. de, E. Amir, and D. Roth (2005). "Lifted First-Order Probabilistic Inference". In: *IJCAI 2005, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*. Ed. by L. P. Kaelbling and A. Saffiotti. Professional Book Center,

pp. 1319–1325. ISBN: 0938075934. URL: http://ijcai.org/Proceedings/05/Papers/1548.pdf.

Sato, T. (1995). "A Statistical Learning Method for Logic Programs with Distribution Semantics". In: *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*. Ed. by L. Sterling. MIT Press, pp. 715–729. ISBN: 0-262-69177-9.

Sato, T. and Y. Kameya (2008). "New Advances in Logic-Based Probabilistic Modeling by PRISM". In: *Probabilistic Inductive Logic Programming - Theory and Applications*. Ed. by L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton. Vol. 4911. Lecture Notes in Computer Science. Springer, pp. 118–155. ISBN: 978-3-540-78651-1. DOI: 10.1007/978-3-540-78652-8_5. URL: https://doi.org/10.1007/978-3-540-78652-8%5C_5.

Saxton, D., E. Grefenstette, F. Hill, and P. Kohli (2019). "Analysing Mathematical Reasoning Abilities of Neural Models". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=H1gR5iR5FX.

Schmidt, G. and T. Ströhlein (1993). "Homogeneous Relations". In: *Relations and Graphs: Discrete Mathematics for Computer Scientists*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 5–27. ISBN: 978-3-642-77968-8. DOI: 10.1007/978-3-642-77968-8_2. URL: https://doi.org/10.1007/978-3-642-77968-8_2.

Shapiro, S. C. (2003). "Artificial Intelligence (AI)". In: *Encyclopedia of Computer Science*. GBR: John Wiley and Sons Ltd., pp. 89–93. ISBN: 0470864125.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529, pp. 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Stanley, R. (2012). *Enumerative combinatorics*. New York: Cambridge University Press. ISBN: 9781107602625.

Stede, M., S. D. Afantenos, A. Peldszus, N. Asher, and J. Perret (2016). "Parallel Discourse Annotations on a Corpus of Short Texts". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016*. Ed. by N. Calzolari, K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, and S. Piperidis. European Language Resources Association (ELRA). URL: http://www.lrec-conf.org/proceedings/lrec2016/summaries/477.html.

Stede, M. and J. Schneider (2018). *Argumentation Mining*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers. DOI: 10.2200/S00883ED1V01Y201811HLT040. URL: https://doi.org/10.2200/S00883ED1V01Y201811HLT040.

Suster, S., P. Fivez, P. Totis, A. Kimmig, J. Davis, L. De Raedt, and W. Daelemans (2021). "Mapping probability word problems to executable representations". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Ed. by M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih. Association for Computational Linguistics, pp. 3627–3640. DOI: 10.18653/v1/2021.emnlp-main.294. URL: https://doi.org/10.18653/v1/2021.emnlp-main.294.

Taghipour, N., D. Fierens, J. Davis, and H. Blockeel (2012). "Lifted Variable Elimination with Arbitrary Constraints". In: *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, Spain, April 21-23, 2012*. Ed. by N. D. Lawrence and M. A. Girolami. Vol. 22. JMLR Proceedings. JMLR.org, pp. 1194–1202. URL: http://proceedings.mlr.press/v22/taghipour12.html.

Thurley, M. (2006). "sharpSAT - Counting Models with Advanced Component Caching and Implicit BCP". In: *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*. Ed. by A. Biere and C. P. Gomes. Vol. 4121. Lecture Notes in Computer Science. Springer, pp. 424–429. ISBN: 3-540-37206-7. DOI: 10.1007/11814948_38. URL: https://doi.org/10.1007/11814948%5C_38.

Totis, P., J. Davis, L. D. Raedt, and A. Kimmig (2023). "Lifted Reasoning for Combinatorial Counting". In: *J. Artif. Intell. Res.* 76, pp. 1–58. DOI: 10.1613/jair.1.14062. URL: https://doi.org/10.1613/jair.1.14062.

Totis, P., A. Kimmig, and L. De Raedt (2023). "SMProbLog: Stable Model Semantics in ProbLog and its Applications in Argumentation". In: *Theory and Practice of Logic Programming*.

Upadhyay, S. and M.-W. Chang (2015). *Draw: A challenging and diverse algebra word problem set*. Tech. rep. Number MSR-TR-2015-78, Oct. 2015.[Online].

Valiant, L. G. (1979). "The Complexity of Computing the Permanent". In: *Theor. Comput. Sci.* 8, pp. 189–201. DOI: 10.1016/0304-3975(79)90044-6. URL: https://doi.org/10.1016/0304-3975(79)90044-6.

Van den Broeck, G. (2015). "Towards High-Level Probabilistic Reasoning with Lifted Inference". In: *2015 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 22-25, 2015*. AAAI Press. URL: http://www.aaai.org/ocs/index.php/SSS/SSS15/paper/view/10243.

Van den Broeck, G., K. Kersting, and S. Natarajan (Aug. 2021). *An Introduction to Lifted Probabilistic Inference*. The MIT Press. ISBN: 9780262365598. DOI: 10.7551/mitpress/10548.001.0001. URL: https://doi.org/10.7551/mitpress/10548.001.0001.

Van den Broeck, G., W. Meert, and A. Darwiche (2014). "Skolemization for Weighted First-Order Model Counting". In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014,*

Vienna, Austria, July 20-24, 2014. Ed. by C. Baral, G. D. Giacomo, and T. Eiter. AAAI Press. ISBN: 978-1-57735-657-8. URL: http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8012.

Van den Broeck, G., N. Taghipour, W. Meert, J. Davis, and L. De Raedt (2011). "Lifted Probabilistic Inference by First-Order Knowledge Compilation". In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. Ed. by T. Walsh. IJCAI/AAAI, pp. 2178–2185. ISBN: 978-1-57735-516-8. DOI: 10.5591/978-1-57735-516-8/IJCAI11-363. URL: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-363.

Van den Broeck, G., I. Thon, M. van Otterlo, and L. De Raedt (2010). "DTProbLog: A Decision-Theoretic Probabilistic Prolog". In: *AAAI*. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1695.

Vennekens, J. (2013). "Negation in the Head of CP-logic Rules". In: *CoRR* abs/1312.6156. arXiv: 1312.6156. URL: http://arxiv.org/abs/1312.6156.

Vennekens, J., M. Denecker, and M. Bruynooghe (2009). "CP-logic: A language of causal probabilistic events and its relation to logic programming". In: *TPLP* 9.3, pp. 245–308. DOI: 10.1017/S1471068409003767. URL: https://doi.org/10.1017/S1471068409003767.

Wang, Y., X. Liu, and S. Shi (2017). "Deep Neural Solver for Math Word Problems". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Ed. by M. Palmer, R. Hwa, and S. Riedel. Association for Computational Linguistics, pp. 845–854. ISBN: 978-1-945626-83-8. DOI: 10.18653/v1/d17-1088. URL: https://doi.org/10.18653/v1/d17-1088.

Wu, Y., M. Caminada, and D. M. Gabbay (2009). "Complete Extensions in Argumentation Coincide with 3-Valued Stable Models in Logic Programming". In: *Stud Logica* 93.2-3, pp. 383–403. DOI: 10.1007/s11225-009-9210-5. URL: https://doi.org/10.1007/s11225-009-9210-5.

Zhang, D., L. Wang, L. Zhang, B. T. Dai, and H. T. Shen (2020). "The Gap of Semantic Parsing: A Survey on Automatic Math Word Problem Solvers". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 42.9, pp. 2287–2305. DOI: 10.1109/TPAMI.2019.2914054. URL: https://doi.org/10.1109/TPAMI.2019.2914054.



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DTAI
Celestijnenlaan 200A box 2402
B-3001 Leuven
pietro.totis@kuleuven.be