Journal of Artificial Intelligence Research 76 (2023) 1-58 Submitted 07/2022; published 01/2023

Lifted Reasoning for Combinatorial Counting

Pietro Totis PIETRO.TOTIS@KULEUVEN.BE
Jesse Davis JESSE.DAVIS@KULEUVEN.BE
Luc De Raedt LUC.DERAEDT@KULEUVEN.BE
Angelika Kimmig ANGELIKA.KIMMIG@KULEUVEN.BE
Department of Computer Science, KU Leuven
Celestijnenlaan 200A, 3001 Heverlee, Belgium

Abstract

Combinatorics math problems are often used as a benchmark to test human cognitive and logical problem-solving skills. These problems are concerned with counting the number of solutions that exist in a specific scenario that is sketched in natural language. Humans are adept at solving such problems as they can identify commonly occurring structures in the questions for which a closed-form formula exists for computing the answer. These formulas exploit the exchangeability of objects and symmetries to avoid a brute-force enumeration of all possible solutions. Unfortunately, current AI approaches are still unable to solve combinatorial problems in this way. This paper aims to fill this gap by developing novel AI techniques for representing and solving such problems. It makes the following five contributions. First, we identify a class of combinatorics math problems which traditional lifted counting techniques fail to model or solve efficiently. Second, we propose a novel declarative language for this class of problems. Third, we propose novel lifted solving algorithms bridging probabilistic inference techniques and constraint programming. Fourth, we implement them in a lifted solver that solves efficiently the class of problems under investigation. Finally, we evaluate our contributions on a real-world combinatorics math problems dataset and synthetic benchmarks.

1. Introduction

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 (Mitra & Baral, 2016; Roy & Roth, 2018; Dries et al., 2017). Among math problems, combinatorics math problems have received less attention in the problem-solving AI literature despite posing interesting 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, Wang, Zhang, Dai, & Shen, 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 lan-

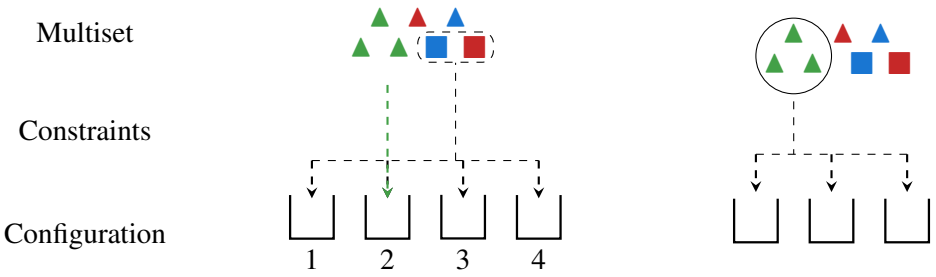


Figure 1: Visualization of **P1** (left) and **P2** (right).

guage down to defining a solving procedure. Dries et al. (2017) and Suster, Fivez, Totis, Kimmig, Davis, De Raedt, and Daelemans (2021) introduced a two-step approach for probability problems, in which text is first mapped to a model expressed in a modelling language, that is then solved in a second step (Figure 2). They showed 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. The key contribution of this paper is that we, for the first time, introduce a modelling language and accompanying solver to directly support combinatorics math problems.

While combinatorial counting problems can sometimes be encoded in particular modelling languages (such as logic and Constraint Satisfaction Problems) and even solved, the encodings are, as we shall show in Section 2, 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

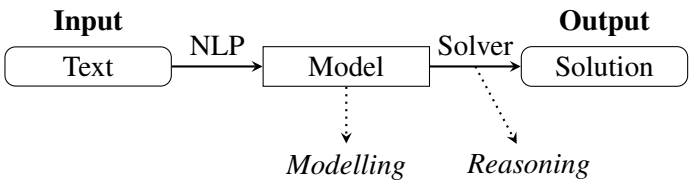


Figure 2: Declarative approach to math word problems.

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 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.

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 specify desired properties of the configuration (e.g. "the second object is green" in **P1**). Table 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 and Constraint Satisfaction Problems (CSPs) do not directly support the modelling of multi-sets and certain types of configurations, (cf. Section 2).

Reasoning, on the other hand, requires the identification of independent subproblems that allow for exploiting the well-known counting formulas from combinatorics as illustrated in Table 1. This is akin to lifted probabilistic inference (Van den Broeck, 2015; Poole, 2003). 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 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 2.

Combinatorics math problems 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 does not come at a 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 (Poole, 2003) 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 Section 2 we discuss the details of these limitations and we address them in the remainder of this paper. In particular, our contributions are as follows:

1. In Section 3 we define lifted counting techniques to reason over general counting constrained satisfaction problems (#CSPs).
2. In Section 4 we identify a class of combinatorics math problems which traditional lifted counting techniques fail to model or solve efficiently, and
3. we address the modelling gap by defining a dedicated modelling language, CoLa.
4. In Section 5 we present a lifted solver, CoSo, for CoLa problems. CoSo implements the lifted reasoning techniques presented in Section 3 for the combinatorics math problems formalized in Section 4.
5. In Section 6 we compare different propositional approaches to our lifted solver, showing how lifted inference outperforms propositional techniques on both a real-world dataset (Dries et al., 2017) and synthetic benchmarks.

We find that we can represent and correctly solve 88% of the real-world problems using CoLa and CoSo. Moreover, we found that 32% of the questions cannot be solved by directly applying a counting rule. In these cases CoSo divides the problem into independent subproblems, solves each one, and then combines those solutions to arrive at the final answer. The comparison of CoSo with propositional methods on a set of real and synthetic benchmarks shows how a lifted approach can bring significant speed-ups in counting the number of valid configurations. At the same time with CoLa we show that the level of abstraction offered by the modelling language influences the opportunities for lifted reasoning and counting.

2. Related Work and Motivation

In this section we discuss the limitations of traditional declarative frameworks concerning modelling and reasoning over combinatorics math problems. We consider Constraint Satisfaction Problems (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.

2.1 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.

Modelling 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 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.

Counting 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 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 (Gottlob, Leone, & Scarcello, 2000; Bulatov, 2013) rather than designing a solver that can efficiently answer a counting question, as we do in this paper.

2.2 Lifted Probabilistic Inference

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. This prevents a direct support for multisets and unordered configurations, as we show in the following paragraphs.

Modelling 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 sets, 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 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*

$$tvs(tv_1) \wedge \cdots \wedge tvs(tv_5) \wedge tv_1 < tv_2 \wedge tv_2 < tv_3 \wedge \cdots \wedge tv_4 < tv_5 \tag{1}$$

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 1 thus corresponds to the number of subsets of 5 TVs.

Moreover, unique identifiers prevent the encoding of partitions altogether.

Example 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 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. stating that any two of the purchased TVs is defective, instead of specifying that the first first and the second are defective, or the first and the third,... However, without dedicated language constructs the combinations of specific assignments have to be explicitly modelled, thus preventing reasoning at the lifted level.

Example 3. *Consider again **P3** and let $df/1$ be a predicate identifying defective TVs. Following Example 1 we can enforce the number of defective TVs by conjoining the logic formula (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:

$$
\begin{aligned}
&df(tv_1) \wedge df(tv_2) \wedge \neg df(tv_3) \wedge \neg df(tv_4) \wedge \neg df(tv_5)) \vee \\
&df(tv_1) \wedge \neg df(tv_2) \wedge df(tv_3) \neg df(tv_4) \wedge \neg df(tv_5)) \vee \\
&\cdots \vee \\
&\neg df(tv_1) \wedge \neg df(tv_2) \wedge \neg df(tv_3) \wedge df(tv_4) \wedge df(tv_5)
\end{aligned}
$$

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, & Schaub, 2012), Prolog (van Emden & 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 1) and domains (Example 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 et al. (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, such as counting, over variable configurations still resorts to explicitly grounding out the different combinations and reasoning over individual objects. Consider for instance the following example:

Example 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).
tvs(X) :- defective(X).
working(tv4).  working(tv5).  working(tv6).
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 leads 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)}`.

	Sets/ Multisets	Configurations	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 2: 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.

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 the such logic rules.

Counting 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 (Poole, 2003), 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 (de Salvo Braz, Amir, & Roth, 2005; Taghipour, Fierens, Davis, & Blockeel, 2012) and knowledge compilation (Van den Broeck, Taghipour, Meert, Davis, & De Raedt, 2011). Lifting can lead to polynomial complexity results for some classes of problems (Van den Broeck, Meert, & Darwiche, 2014; Kazemi, Kimmig, Van den Broeck, & Poole, 2016) 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, & De Raedt, 2017) generalizes probabilistic inference, #SAT, and Weighted Model Counting and shows how knowledge compilation solves any Algebraic Model Counting 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 2.

Weighted *First-Order* Model Counting (WFOMC), implemented in Forclift (Van den Broeck 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.

WFOMC with counting quantifiers (Kuzelka, 2021) considers the two-variable fragment of first-order logic (FO^2) with counting quantifiers (Gradel, Otto, & Rosen, 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. 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.

3. Lifted Reasoning over Counting Problems

Probabilistic lifted reasoning techniques are designed for first-order logic representations and not CSPs. CSPs on the other hand are a more suitable formal representation than first-order logic, but they rely on enumeration for counting. For this reason, in this section we introduce the missing intersection of the two approaches: general lifted reasoning techniques for #CSPs based on the principles of probabilistic lifted reasoning. The section is organized as follows: first, we present the necessary background and notation for #CSPs and probabilistic lifted reasoning (Section 3.1) and then propose novel lifted reasoning techniques for #CSPs (Section 3.2).

3.1 Background

We divide the background information between #CSPs and lifted probabilistic inference techniques.

3.1.1 #CSPs

A counting Constraint Satisfaction Problem, #CSP, is the task of counting the number of solutions of a CSP.

Definition 1. *(Rossi, van Beek, & Walsh, 2006) A* Constraint Satisfaction Problem *(*CSP*) P is a triple $P = \langle V, D, C \rangle$ where $V = \langle v_1, \ldots, v_n \rangle$ is a 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.*

Note that a CSP treats variables as a *tuple*, which means that an ordering is implicit in the problem statement. This is undesirable in our setting because problems require reasoning over unordered configurations: this point will be revisited in Section 4. A solution of a CSP is an assignment that satisfies the given constraints. Let D_W denote the subset of D corresponding to variables W.

Definition 2. *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.*

Definition 3. *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: $f(\langle v_1, \ldots, v_m \rangle) = \langle d_1, \ldots, d_m \rangle$, $d_i \in D_i, D_i \in D_W$.*

f is a *partial* assignment when $W \subset V$, or *total* when $W = V$. A satisfying assignment is an assignment that satisfies all the constraints in C.

Definition 4. *An assignment* f satisfies *a constraint* $c = (W, R_W)$, $f(W) \vdash c$, *when* $f(W) \in R_W$.

The task in a #CSP is to count the number of *satisfying assignments* (solutions). 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 a counting problem is specified as the problem of finding the number of assignments to a set of logic variables that satisfy a given (first-order) logic formula (the model).

3.1.2 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 5. *(Niepert & 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* (Taghipour et al., 2012; Milch et al., 2008) 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 4, while in Section 3.2 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 & Poole, 2009; Poole, 2003), and those based on knowledge compilation (Van den Broeck 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.

3.2 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 6. *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 7. *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 5. *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 paper 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 8. *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 9. *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 5.

Example 6. *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 10. *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 11. *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 12. *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 7. *Consider Example 6: 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 13. *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 7 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 14. *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 15. *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 16. *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 8. *Consider Example 7: 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 8. 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. Modelling Combinatorics Math Problems

The challenge of designing a framework for combinatorial counting based on lifted reasoning lies in recognizing when and where the principles presented in Section 3 are applicable. 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 1 and 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, Section 5). The goal of CoLa is thus to match the scope of the lifted solver presented in this paper, 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 6.1). 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 6.2).

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 4.1);
2. the configurations corresponding to the special cases where counting rules are applicable (Section 4.2);
3. the constraint language for the problems at hand (Section 4.3).

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 5).

4.1 Multisets

The first characterizing component is the finite set of atomic objects we reason about. As we argued in Section 2, 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.

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). 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 9. *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 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 2) 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.

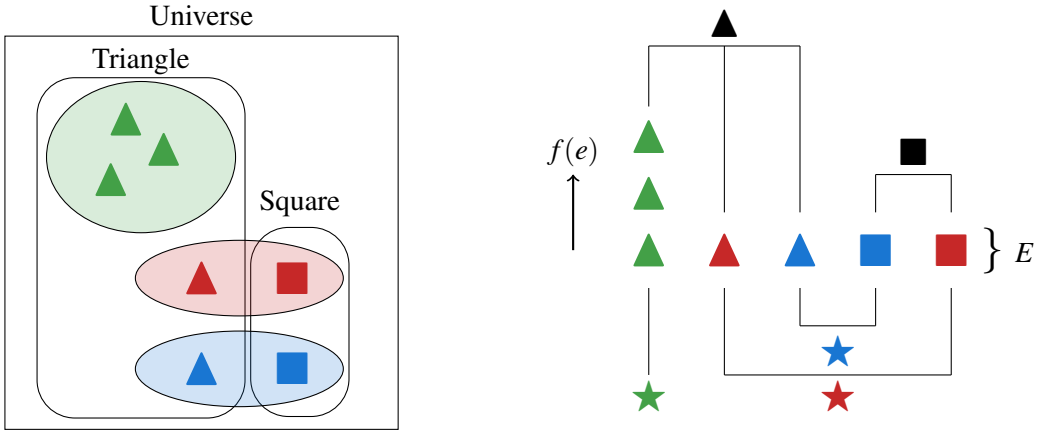


Figure 3: On the left, the objects in Example 1 grouped by their properties. On the right, the corresponding multiset. Stars represent the colour property, black items represent the shape property.

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 4.2.

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 10. *The universe of **P1**, following Example 9, 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 11. *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 4.3) 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 12. *The universe of **P1**, following Example 9, is declared with the CoLa statements:*

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

Example 13. *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;
```

4.2 Configurations

The second characterizing component of combinatorics math problems is the *configuration* in which objects are arranged. In Sections 1 and 2 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, 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 4.3). The second dimension is concerned with the indistinguishability of objects, where we have all objects distinguishable ($\neq$, a 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. Figure 4 represents the difference between the two types.

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 14. *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:

$g : X \to Y$	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: 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.

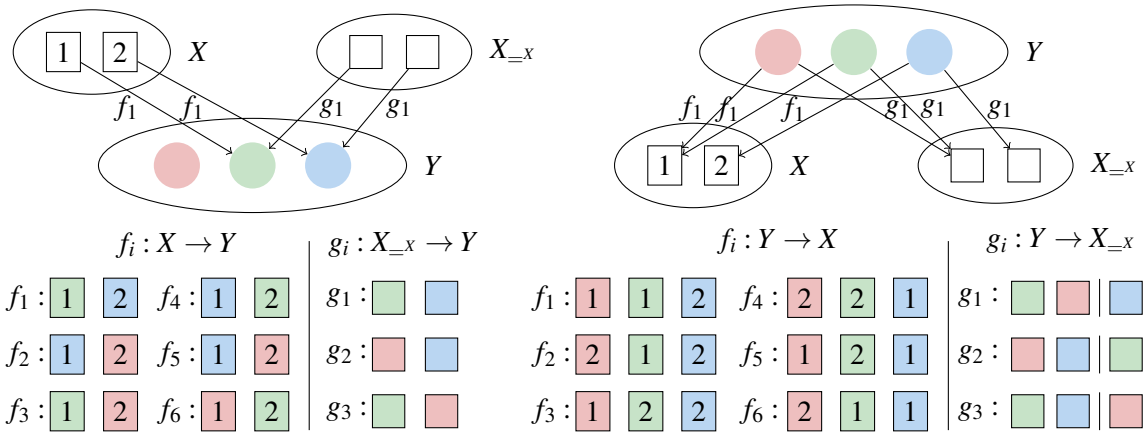


Figure 4: Level 1 configurations map slots to objects Y (left). Level 2 configurations map objects Y to slots (right)

Definition 17. *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* $\{\cdot\}$
- ***Level.*** *Levels define how entities are grouped together: $g : C \to E$ (resp. $g : E \to C$) is a level 1 (level 2) configuration. Figure 4 visualizes 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 4).*
- ***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 4).*

Table 4 summarizes the configurations we will consider throughout the paper. 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 4

Example 15. *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*

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

Table 4: Configuration types and the corresponding CoLa syntax for a set-formula φ

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 10.*
*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 5). 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 4.3) from those intrinsic in the configuration.

As we argued in Section 2, 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 16. ***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 18. *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.*

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 6).

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.

4.3 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 19. *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 17. *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 20. *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 18. *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 19. *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 17:*

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

Example 20. ***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; #tvs = 12;
property defective; #(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 21. *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 21. *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:*

```
row[2] = green;
```

In a configuration "composition" $((green,=,3),1)$ *states that the first part in the order contains exactly three green objects.*

Example 22. ***P6** with the additional positional constraint:*

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

In Table 5 we summarize all CoLa statements and their semantics presented in this section.

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}]`	$cf = \{\{\cdot\}\}/[\{\cdot\}]$, $D_i = \mathcal{P}(U)$	Define level 2 config., its label and set
`cf in [repeated U]` or `cf in [U]`	$cf = [\lvert\lvert\cdot\rvert]/[\lvert\cdot\rvert]$, $D_i = U$	Define ordered level 1 config., label, set
`cf in {repeated U}` or `cf in {U}`	$cf = \{\lvert\lvert\cdot\}/\{\lvert\cdot\}$, $D_i = U$	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 `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 5: Summary of CoLa syntax and corresponding semantics. U denotes the universe corresponding to the given objects and properties.

5. Lifted Reasoning over Combinatorics Math Problems

This section describes CoSo[2], a solver for the full range of problems expressible in CoLa (Section 4). CoSo thus implements our novel general-purpose lifted approach for counting constraint satisfaction problems (Section 3) 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), 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 3) 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 3 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 5 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.

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 5.1). We then present these cases starting from the base ones (Section 5.2), followed by the propagation of counting constraints under exchangeability (Section 5.2.3). We then move to the right-hand side of the diagram in Figure 5 and consider the dual case when variables are not exchangeable, defining the corresponding partition of the problem (Section 5.3.2). We then extend this operator to the level 1 configurations where the entities are not repeated (Section 5.4), and finally consider the level 2 partition problems when the base cases are not applicable (Section 5.5).

2. Python implementation: https://github.com/PietroTotis/CoSo

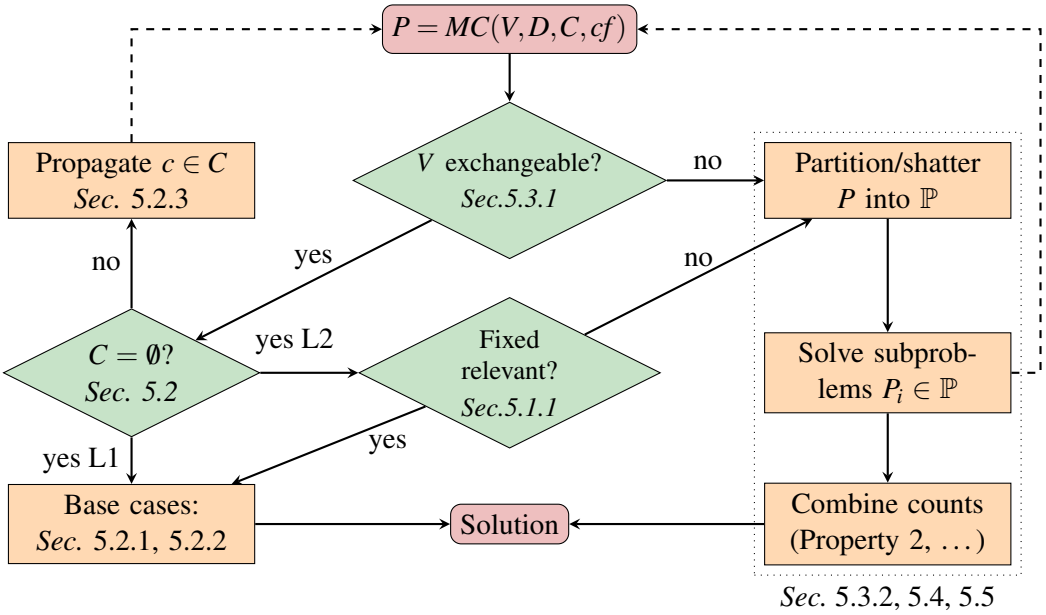


Figure 5: Solver reasoning schema. L1= level 1 configurations, L2 = level 2 configurations. Dashed arrows denote recursive calls to the solver.

5.1 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 rep-

resented 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 23. *Consider **P1** and Example 10: 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:
$V = \langle v_1, v_2, v_3, v_4 \rangle$, $D = \langle shapes, green, shapes, shapes \rangle$, $C = \{(squares, \geq, 2)\}$, $cf = [|\cdot|]$.

Example 24. *Consider **P2** and Example 10: 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 25. *Consider Example 23: 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.

5.1.1 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 22. *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 23. *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 5.3).

Definition 24. *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 26. *In Example 25 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: ($green \cap \overline{squares}$ and $green \cap squares$), hence the histograms that summarize the relevant information are:*

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

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 27. *In Example 26 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 $relevant(\{green, squares\})$: 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 $|U| - n$.*

5.2 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.

5.2.1 LEVEL 1: BASE CASES

Level 1 configurations correspond to the four top-left cases in Table 3, 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

$$|dom(V)|^n \tag{2}$$

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:

$$\sum_{M=(T,g)\subseteq dom(V),|M|=n} \frac{|M|!}{f(t_1)!\cdot f(t_2)!\cdot\ldots\cdot f(t_j)!}, t_i \in T, j = |T| \tag{3}$$

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, Mendelson, Ballesio, and Vercauteren (2015) define a counting rule for this case:

$$\sum_{L\in\mathscr{P}(I_k)} (-1)^{|L|} \binom{n+k-1-|L|-\sum_{i\in L} a_i}{k-1} \tag{4}$$

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:

$$\binom{|S|+n-1}{n} \tag{5}$$

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

5.2.2 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:

$$x^i_j = \begin{cases} \binom{a^i_j}{n^i_j} & \text{if entities in } R_j \text{ are interchangeable} \\ 1 & \text{if entities in } R_j \text{ are indistinguishable} \end{cases} \tag{6}$$

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:

$$compositions(V) = \prod_{i\in\{1,\ldots,|V|\}} \prod_{(n^i_j,R_j)\in h_i} x^i_j \tag{7}$$

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:

$$partitions(V) = \frac{compositions(V)}{\prod_{[v_i]\in V/\equiv} |[v_i]|}. \tag{8}$$

Example 28. *Consider Example 24 and a partition of green entities in three subsets with the following histograms:* $\{\{(3,green),(0,\overline{green})\},\{(0,green),(2,\overline{green})\},\ \{(0,green),(2,\overline{green})\}\}$. *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 8 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 33).*

5.2.3 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 5.3.2). 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
$D_{sat}=\{D_i=dom(V)\cap\varphi\,|\,i\in\{1,\ldots,s\}\}$
$D_{unsat}=\{D_i=dom(V)\cap\overline{\varphi}\,|\,i\in\{s+1,\ldots,n\}\}$
$P_1=\langle\{v_1,\ldots,v_s\},D_{sat},\{\},cf\rangle$
$P_2=\langle\{v_{s+1},\ldots,v_n\},D_{unsat},\{\},cf\rangle$ **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 8*)

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. □

5.3 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

Configuration	Fixed relevant	Exchangeable	Non-exchangeable
sequence multisubset	n.a.	Base case (Section 5.2.1, Equations 2 and 5)	
permutation subset		Base case (Sec. 5.2.1, Eq. 3 and 4)	NOREP_BASE$_{\not\equiv}$ (Section 5.4)
partition composition	$\|\mathscr{R}\|=0$	Base case (Section 5.2.2, Equations 7 and 8)	
	$\|\mathscr{R}\|=1$	PARTS$_{\equiv}$ (Section 5.5.2)	PARTS$_{\not\equiv}$ (Section 5.5.3)
	$\|\mathscr{R}\|>1$	PART (Section 5.5.4)	

Table 6: Operator cases without counting constraints. (n.a.= not applicable)

Configuration	Exchangeable	Non-exchangeable
sequences, multisets partitions, compositions	COUNT$_{\equiv}$ (Section 5.2.3)	COUNT$_{\not\equiv}$ (Section 5.3.2)
permutations subsets		NOREP$_{\not\equiv}$ (Section 5.4)

Table 7: Operator cases with counting constraints.

and parts. In Section 5.3.1 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 6 and 7. In Section 5.3.2 we begin with the partitioning of counting constraints, which applies to all problems but those with a non-repetition constraint (permutations and subsets).

5.3.1 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 $\hat{V} = \{w \mid w \in [v_i], [v_i] \in V/\equiv \setminus \{[v]\}\}$. 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 29. *In Example 23 domain $D = \langle univ, green, univ, univ \rangle$ 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.*

5.3.2 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 7). 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 5.4.2 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 $\langle V,D,C,cf\rangle$. □

5.4 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 5.2.1). In this section we describe the case where variables are not exchangeable. First, we define the case without additional counting constraints, $|C| = 0$ (Section 5.4.1), then we generalize it to $|C| > 0$ (Section 5.4.2).

5.4.1 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 5.3.1. 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: $MC(P) = \sum_{\langle P_1^h, P_2^h \rangle \in \mathbb{P}} MC(P_1^h) \cdot MC(P_2^h)$ *(Property 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. □

5.4.2 LEVEL 1: SHATTERING REPETITION WITH COUNTING CONSTRAINTS

We generalize the operators NOREP_BASE$_{\not\equiv}(P)$ and 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 5.6 we give a detailed description of how Example 29 is partitioned and solved by applying the operators presented in this section.

5.5 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 5.3.2, 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 30. *Consider Example 24: 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) for problems of level 2.

Next, we generalize the approach in Section 5.2.2 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 31. *In Example 24 propagating the number of green objects in the parts results in Example 30. 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 5.5.1 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 5.5.2); (2) the case where variables are not exchangeable (Section 5.5.3); and (3) the general case with multiple relevant properties, solved by means of recursion (Section 5.5.4).

5.5.1 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 31 shows the distributions of entities from the universe excluded by $\mathit{distribute}(P,\overline{green},4,3)$. 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 5.5.2) and they are not exchangeable (Section 5.5.3), 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.

5.5.2 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 $e([\{\cdot\}],\mathbf{i}\}) = \frac{|\mathbf{i}|}{n_1!\cdots n_j!}$ 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** $\mathit{distribute}(P,R,|V|)$ **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 $\mathit{distribute}(P,R,|V|) = \emptyset$, 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 $\mathit{distribute}(P, R, |V|)$. $\mathbf{i}$ 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 $\mathit{distribute}(P, R, r, |V|)$ are non-overlapping. Since $\mathit{distribute}(P, R, r, |V|)$ contains all valid distributions of R, we can conclude that $\mathbb{P}$ partitions P. □

5.5.3 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 $\mathit{distribute}(P, R, r, k)$, 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: $\{(e_1^A \cdot e_1^B, P_1^A \cup P_1^B), \ldots, (e_1^A \cdot e_m^B, P_1^A \cup P_m^B), \ldots, (e_n^A \cdot e_n^B, P_n^A \cup P_m^B)\}$ 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** $\mathit{distribute}(P, R, k)$ **do**
      $join = \times\{\text{PARTS}_{\equiv}(\langle V_j, D_{V_j}, C, f \rangle, \{(i_j, R)\}) \mid V_j \in V/\equiv\}$
      $\mathbb{P} = \mathbb{P} \cup join$
    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. □

5.5.4 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 5.6 we complete Examples 30 and 31 with a detailed description of the application of the operators for level 2 configurations.

5.6 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 32. *We complete Example 29: the goal is to count the number of 4-permutations where the second object is green and there are 2 squares. Let univ = squares ∪ triangles = {■, ■, ▲, ▲, ▲, ▲, ▲}. As described in Example 29 the result of the propagation of the positional constraint is the #CSP $P = MC(V, D, C, cf)$, where $V = \langle v_1, v_2, v_3, v_4 \rangle$, $D = \langle univ, green, univ, univ \rangle$, $C = \{(squares, \geq, 2)\}$, $cf = [|\cdot|]$. We represent the corresponding configuration as $\langle ★, ▲, ★, ★ \rangle$, where ★ denotes an undecided shape with undecided colour.*

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 $\langle univ, green, univ, univ \rangle$, 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 $V_2 = \langle v_1, v_3, v_4 \rangle, D_2 =$

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

Operator: $\text{COUNT}_{\equiv}(P)$ =
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
$MC(P) = e \cdot MC(P_1) \cdot MC(P_2)$

Figure 6: Operators required for solving Example 32

$\langle univ, univ, univ\rangle$. *Then, following* $\text{NOREP}_{\not\equiv}$*, we consider the different histograms counting the relevant properties in a solution for* P_1 *to solve* P_2 *independently.*

In Example 26 we define the relevant parts green *and* squares*, along with the histograms in Example 29. 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* $(univ^* = \{■, ■, ▲, ▲, ▲, ▲\}$*, hence* $\overline{squares} = \{▲, ▲, ▲, ▲\}$ *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 (*$\text{COUNT}_{\equiv}$*): 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.*

P_2 *In* P_2 *variables are exchangeable and there is a counting constraint:* $\text{COUNT}_{\equiv}$ *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:*

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

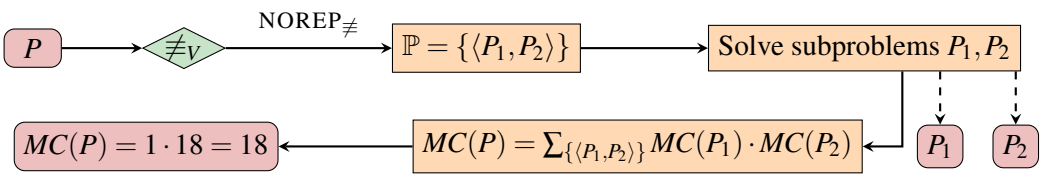


Figure 7: Solver execution flow for **P1**

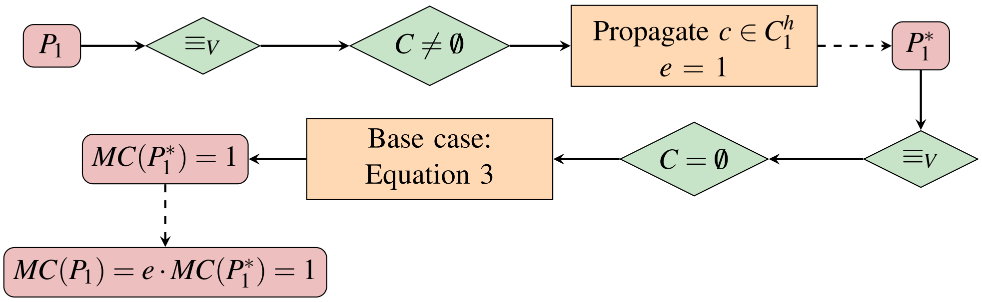
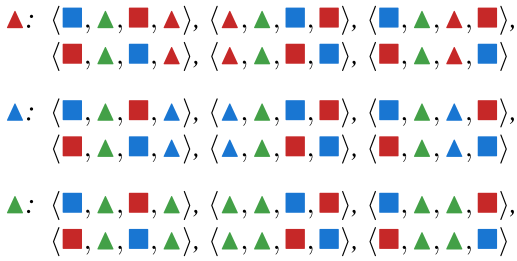


Figure 8: Solver execution flow for the first subproblem of **P1**

$MC(P_{2b}) = 3$ 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:



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:

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

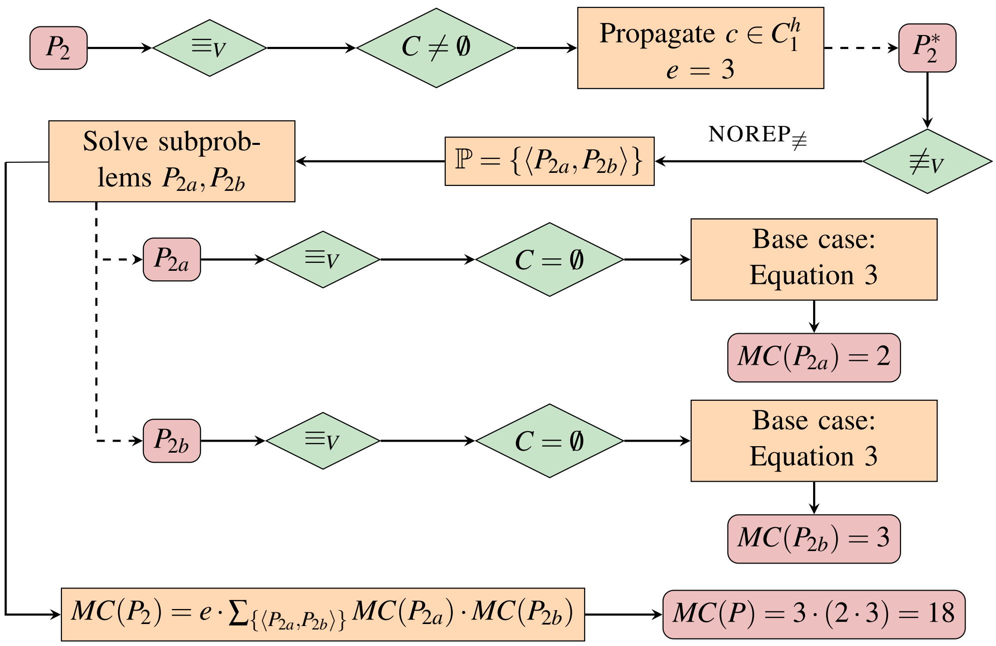


Figure 9: Solver execution flow for the second subproblem of **P1**

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

Operator: $\text{PARTS}_{\not\equiv}(P,\mathscr{R})$ =
let $V/\equiv$ **be** $\{V_1,\ldots,V_k\}$:
$\mathbb{P} = \{\}$
for i $= \langle i_1,\ldots,i_k\rangle$ **in** *distribute*(P,R,k):
  $j = \times\{\text{PARTS}_{\equiv}(\langle V_j, D_{V_j}, C, f\rangle, \{(i_j,R)\}) | V_j \in V/\equiv\}$
  $\mathbb{P} = \mathbb{P} \cup j$
return $\mathbb{P}$
Postcondition:
  $MC(P) = \sum_{(c,P')\in\mathbb{P}} c \cdot MC(P')$

Figure 10: Operators required for solving Example 33

Example 33. *We complete Examples 30 and 31: 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 31): $V = \langle v_1,v_2,v_3\rangle$, $D = \langle\{(green,=,3)\},\{(green,\neq,3)\},\{(green,\neq,3)\}\rangle$, $C = \emptyset$, $cf = \{\{\cdot\}\}$. 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. 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: $\{\{\blacktriangle,\blacktriangle,\blacktriangle,\bigstar\},\{\bigstar\},\{\bigstar,\bigstar\}\}$

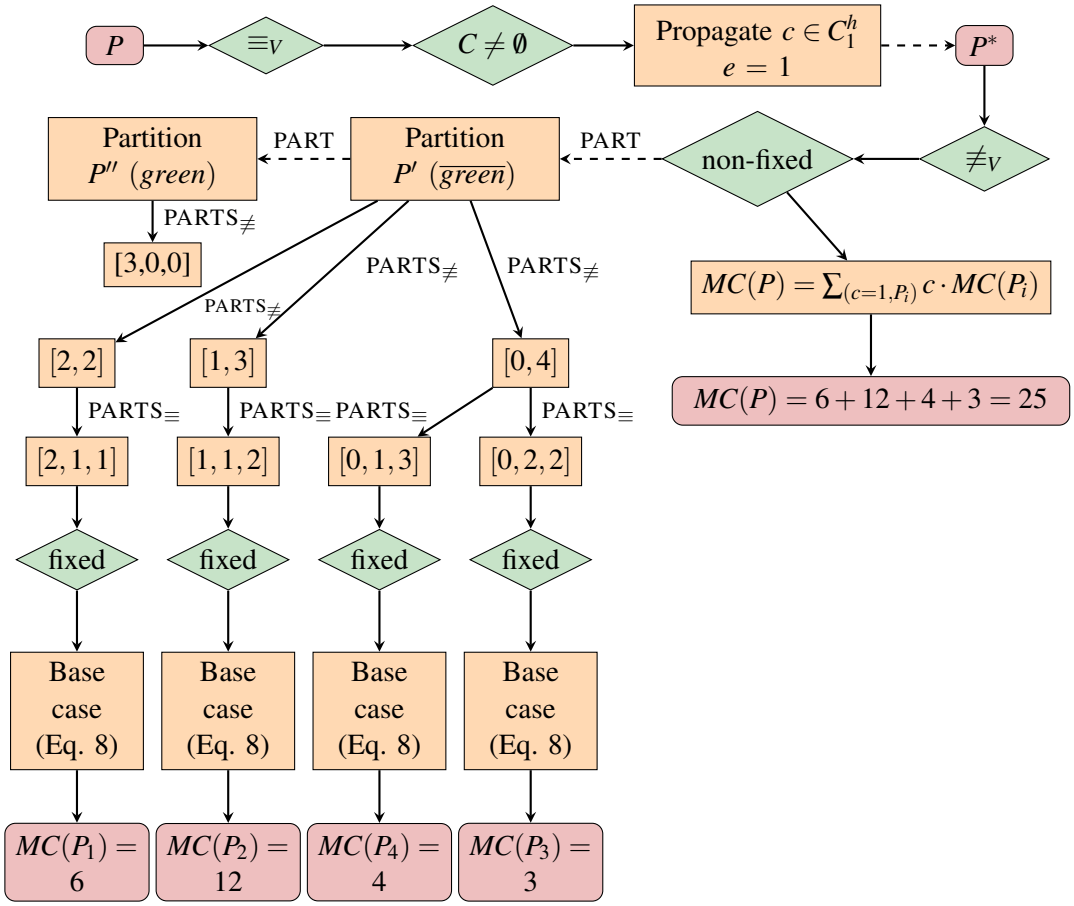


Figure 11: Solver execution flow for **P2**

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

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

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:*

$\{\{▲,▲,▲,▲,▲\},\{■\},\{■\}\}$, $\{\{▲,▲,▲,■,▲\},\{■\},\{▲\}\}$,
$\{\{▲,▲,▲,▲,■\},\{▲\},\{■\}\}$, $\{\{▲,▲,▲,■,■\},\{▲\},\{▲\}\}$,
$\{\{▲,▲,▲,■,▲\},\{■\},\{▲\}\}$, $\{\{▲,▲,▲,▲,■\},\{▲\},\{■\}\}$.

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

$\{\{▲,▲,▲,▲\},\{▲\},\{■,■\}\}$, $\{\{▲,▲,▲,▲\},\{■\},\{■,▲\}\}$,
$\{\{▲,▲,▲,▲\},\{■\},\{▲,■\}\}$, $\{\{▲,▲,▲,■\},\{▲\},\{▲,■\}\}$,

{{▲,▲,▲,■},{▲},{■,▲}}, {{▲,▲,▲,■},{■},{▲,▲}},
{{▲,▲,▲,▲},{■},{▲,■}}, {{▲,▲,▲,▲},{▲},{■,■}},
{{▲,▲,▲,▲},{■},{■,▲}}, {{▲,▲,▲,■},{▲},{▲,■}},
{{▲,▲,▲,■},{▲},{▲,■}}, {{▲,▲,▲,■},{■},{▲,▲}}.

P_3. $\frac{\binom{4}{0}\cdot\binom{4}{3}}{1\cdot1\cdot1} = 4$ *Corresponding to:*

{{▲,▲,▲},{▲},{▲,■,■}}, {{▲,▲,▲},{▲},{▲,■,■}},
{{▲,▲,▲},{■},{▲,■,▲}}, {{▲,▲,▲},{■},{▲,■,▲}}.

P_4. $\frac{\binom{4}{0}\cdot\binom{4}{2}}{1\cdot2} = 3$ *(Example 28) Corresponding to:*

{{▲,▲,▲},{▲,▲},{■,■}},
{{▲,▲,▲},{▲,■},{▲,■}},
{{▲,▲,▲},{▲,■},{▲,■}}.

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

6. Language and Solver Analysis

In this section we evaluate empirically the contributions of the paper 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) whether CoLa can encode real-world combinatorics math problems,

Q2) how many of these problems can CoSo solve,

Q3) what type of problems is hard for CoSo,

Q4) how CoSo compares to existing methods.

6.1 Modelling: Language Analysis (Q1)

To answer the first question, we encoded in CoLa 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 and 106 (51%) 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. These could be added in one of two ways. 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 & 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 a 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 splitting rules, both individually and in combination with other constraints. In fact, with the operators presented in Section 5, we showed that splitting typically requires shattering of constraints and different combinations must be taken into account. For example, the non-repetition constraint and counting constraints are split at the same time and the corresponding operator ($\text{NOREP}_{\neq}$) defines how this can be done by means of histograms. In general, existing constraint modelling languages provide a reference point as to which type of constraints are most useful and interesting language-wise.

6.2 Reasoning: Solver Analysis (Q2, Q3, Q4)

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 (**Q4**) with three modelling languages and the corresponding reasoning systems:

- **ASP, Clingo**. Answer Set Programming (Gebser et al., 2012) is a prominent logic programming framework based on first order logic with counting constraints. Clingo (Gebser, Kaminski, Kaufmann, Ostrowski, Schaub, & Wanko, 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 2) is translated by Conjure (Akgün et al., 2022) to the input format of the solver Minion (Gent, Jefferson, & Miguel, 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.

6.2.1 REAL-WORLD PROBLEMS (Q2, Q3, Q4)

Problem decomposition CoSo solves correctly all models (**Q2**): in Figure 12 we analyze the running time with respect to the number of subproblems considered by the solver. A subproblem is

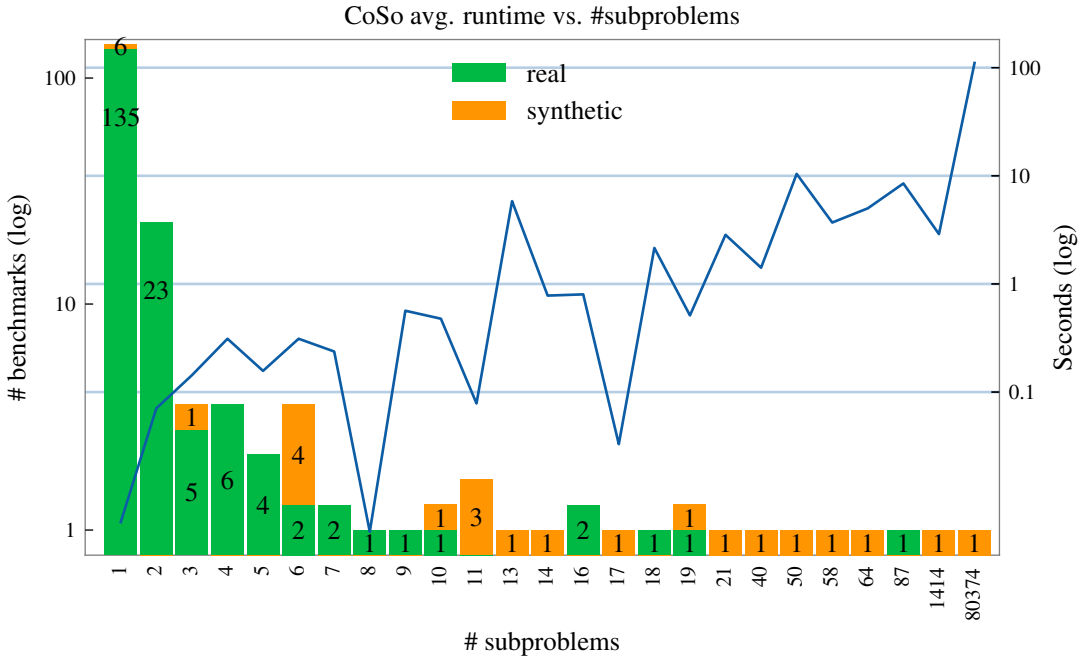


Figure 12: CoSo solving time (log scale) tends to increase with the number of subproblems considered. Real-world dataset and synthetic benchmarks combined.

counted each time the solver is called recursively, therefore when the start node diagram in Figure 5 is traversed. Figure 12 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%) cannot be solved by the Twelvefold-way, because either a multiset (19) or a constraint (11, propagated without splitting) are needed. These 11 problems together with the 48 (26%) that require a decomposition into subproblems, make a total of 59 (32%) problems which cannot be solved by directly applying a counting rule.

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 (**Q3**). In fact, the most complex instances do not specify a single size for the configuration: in Section 5 we explain that we solve a separate #CSP for each valid size. In practice, for some problems it may be possible to exploit 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 5). 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.

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 8: Comparison of the solvers on the 185 problems that can be encoded in CoLa. CoSo outperforms on the real-world dataset the other frameworks by solving all problems within the timeout and with a lower average running time.

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 8, 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

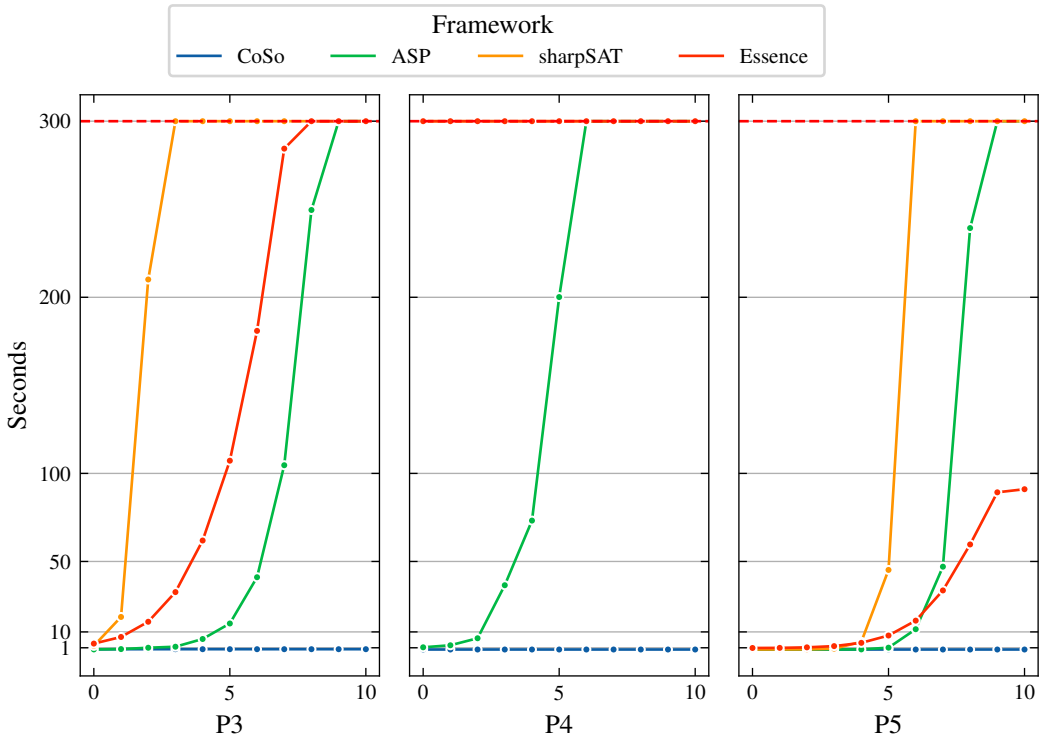


Figure 13: The solving time of propositional methods increases exponentially with the number of objects or the size of the configuration.

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 on the x scale), 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 13, confirm our expectations. All problems are solved by CoSo in less than a second with constant time with respect to the increase 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 on the x scale).

6.2.2 SYNTHETIC BENCHMARKS (Q3, Q4)

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 2) 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 $\{0,\dots,s\}$ 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 (**Q3**) when the multiset presents 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 12 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

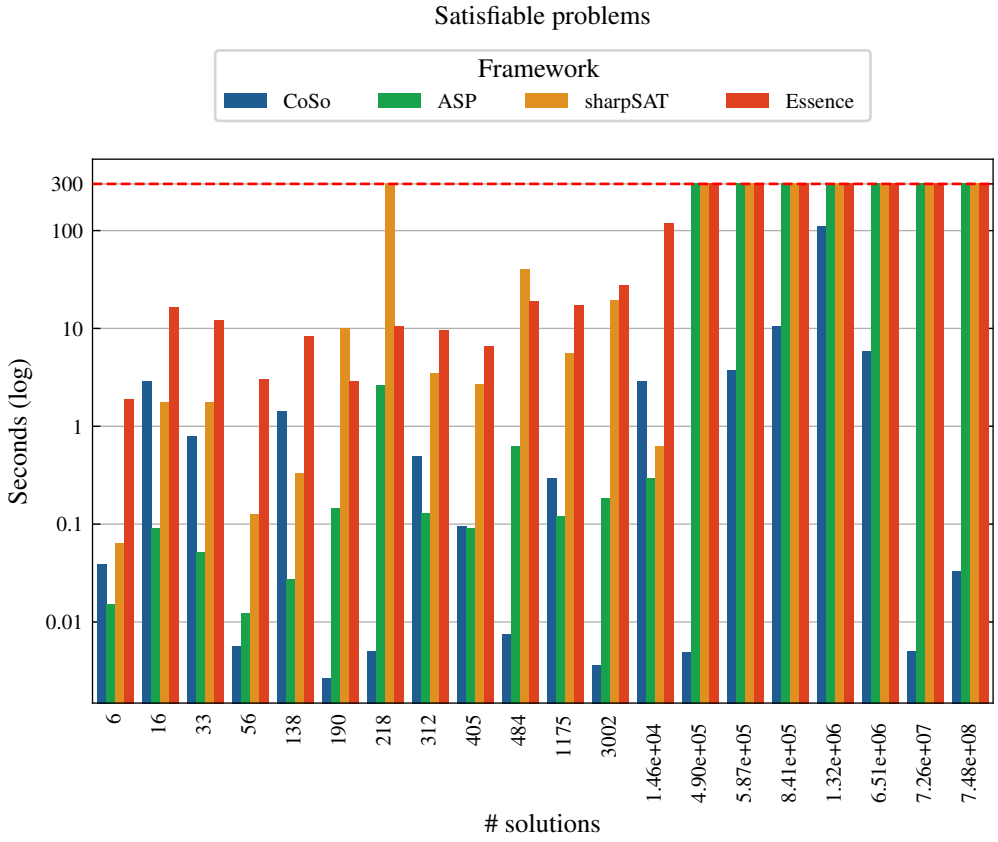


Figure 14: Solving times (log scale) on synthetic benchmarks: CoSo outperforms propositional frameworks in most cases, in particular with the increase of the number of solutions.

correlation between the number of subproblems in which the instance is partitioned and the total time required to compute the solution.

Figure 14 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 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 15 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.

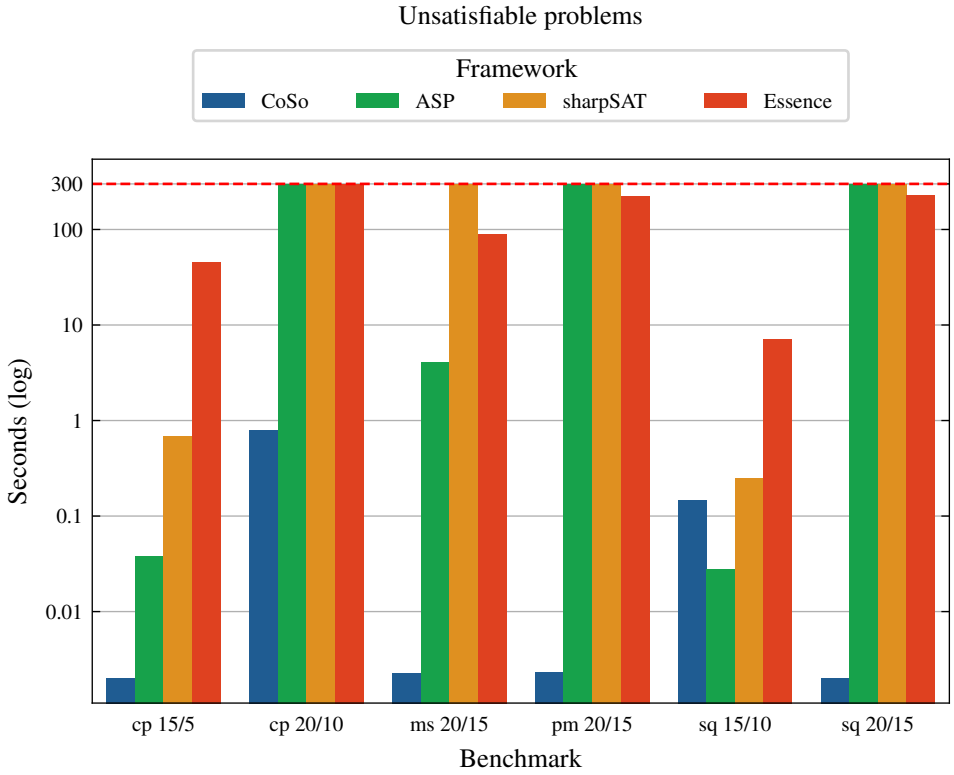


Figure 15: On unsatisfiable configurations CoSo outperforms propositional frameworks on most of the instances (log scale). For ASP and sharpSAT most of these instances have the largest groundings of the dataset. Legend format: *configuration type universe size/configuration size of reference*. *ms=multiset, pm=permutation, sq=sequence, cp=composition.*

7. Conclusion

We presented a framework capable of modelling and efficiently solving a wide range of real-world combinatorics math problems. Contrary to traditional declarative frameworks, our framework has direct support for the fundamental primitives required to model such problems and combinatorial counting. The framework is composed by (1) a novel modelling language for combinatorics math problems, CoLa, and (2) a solver for CoLa problems, CoSo. CoLa expands a well-known characterization of the basic combinatorial problems, the *Twelvefold-way*, with a richer constraint language and a more general multiset specification. CoSo implements novel lifted reasoning techniques for counting constraint satisfaction problems. They represent an important step in bridging lifted probabilistic inference techniques based on first-order logic with general #CSPs. We considered a class of combinatorics math problems which the existing declarative frameworks do not directly support, failing at either modelling adequately the problems or performing efficiently the counting task. An experimental evaluation of propositional methods shows these limitations and the benefit of a lifted approach. This motivates the relevance of our work on CoLa and CoSo, which are capable of effectively modelling and solving the large majority of a real-world dataset of combinatorics math problems. The analysis of the cases where this is not possible suggests interesting directions for

future work, not only in terms of expanding the expressivity of CoLa, but also in terms of the corresponding lifted reasoning techniques.

Acknowledgments

We thank Timothy Van Bremen for the valuable feedback and discussions, and the anonymous reviewers for the helpful comments. This work was supported by FWO [project N. G066818N].

Appendix A

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[3]. 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 34. *The ASP encoding of the set of objects in **P3** automatically generated from CoLa is as follows:*

```
% Multisets and universe specification.
tvs(1..3).
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).
```

3. https://research.ics.aalto.fi/software/asp/lp2sat/

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 35. *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 36. *The ASP encoding of the configuration in **P5** automatically generated from CoLa is as follows:*

```
% Configuration declaration
permutation_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{permutation_6(A,B,C,D,E,F):permutation_guess_6(A,B,C,D,E,F)}1.
% Auxiliary predicate for aggregates
used_6(X,0) :- permutation_6(X, _, _, _, _, _).
used_6(X,1) :- permutation_6(_, X, _, _, _, _).
used_6(X,2) :- permutation_6(_, _, X, _, _, _).
used_6(X,3) :- permutation_6(_, _, _, X, _, _).
used_6(X,4) :- permutation_6(_, _, _, _, X, _).
used_6(X,5) :- permutation_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 37. *If we wanted to constrain position 2 in **P5** to be an "a", the declaration of the permutation would look like:*

```
permutation_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 38. *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, 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 39. *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_universe 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_universe 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 2, 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 40. *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_universe(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 41. *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 42. *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
permutation_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):permutation_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 43. *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_universe 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
```

References

Akgün, Ö., Frisch, A. M., Gent, I. P., Jefferson, C., Miguel, I., & Nightingale, P. (2022). Conjure: Automatic generation of constraint models from problem specifications. *Artif. Intell.*, *310*, 103751.

Bulatov, A. A. (2013). The complexity of the counting constraint satisfaction problem. *J. ACM*, *60*(5), 34:1–34:41.

Chesani, F., Mello, P., & Milano, M. (2017). Solving mathematical puzzles: A challenging competition for AI. *AI Mag.*, *38*(3), 83–96.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artif. Intell.*, *42*(2-3), 393–405.

de Salvo Braz, R., Amir, E., & Roth, D. (2005). Lifted first-order probabilistic inference. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *IJCAI 2005, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pp. 1319–1325. Professional Book Center.

Dries, A., Kimmig, A., Davis, J., Belle, V., & De Raedt, L. (2017). Solving probability problems in natural language. In Sierra, C. (Ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pp. 3981–3987. ijcai.org.

Ferraris, S., Mendelson, A., Ballesio, G., & Vercauteren, T. (2015). Counting sub-multisets of fixed cardinality. *CoRR*, *abs/1511.06142*.

Frisch, A. M., Harvey, W., Jefferson, C., Hernández, B. M., & Miguel, I. (2008). Essence : A constraint language for specifying combinatorial problems. *Constraints An Int. J.*, *13*(3), 268–306.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Wanko, P. (2016). Theory solving made easy with clingo 5. In Carro, M., King, A., Saeedloei, N., & Vos, M. D. (Eds.), *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*, Vol. 52 of *OASIcs*, pp. 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gent, I. P., Jefferson, C., & Miguel, I. (2006). Minion: A fast scalable constraint solver. In Brewka, G., Coradeschi, S., Perini, A., & Traverso, P. (Eds.), *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*, Vol. 141 of *Frontiers in Artificial Intelligence and Applications*, pp. 98–102. IOS Press.

Gottlob, G., Leone, N., & Scarcello, F. (2000). A comparison of structural CSP decomposition methods. *Artif. Intell.*, *124*(2), 243–282.

Gradel, E., Otto, M., & Rosen, E. (1997). Two-variable logic with counting is decidable. In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, pp. 306–317.

Greco, G., & Scarcello, F. (2010). On the power of tree projections: Structural tractability of enumerating CSP solutions. *CoRR*, *abs/1005.1567*.

Kazemi, S. M., Kimmig, A., Van den Broeck, G., & Poole, D. (2016). New liftable classes for first-order probabilistic inference. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I.,

& Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3117–3125.

Kimmig, A., Van den Broeck, G., & De Raedt, L. (2017). Algebraic model counting. *J. Appl. Log.*, *22*, 46–62.

Kisynski, J., & Poole, D. (2009). Constraint processing in lifted probabilistic inference. In Bilmes, J. A., & Ng, A. Y. (Eds.), *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pp. 293–302. AUAI Press.

Kuzelka, O. (2021). Weighted first-order model counting in the two-variable fragment with counting quantifiers. *J. Artif. Intell. Res.*, *70*, 1281–1307.

Marriott, K., Nethercote, N., Rafeh, R., Stuckey, P. J., de la Banda, M. G., & Wallace, M. (2008). The design of the zinc modelling language. *Constraints An Int. J.*, *13*(3), 229–267.

Milch, B., Zettlemoyer, L. S., Kersting, K., Haimes, M., & Kaelbling, L. P. (2008). Lifted probabilistic inference with counting formulas. In Fox, D., & Gomes, C. P. (Eds.), *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pp. 1062–1068. AAAI Press.

Mitra, A., & Baral, C. (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.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). Minizinc: Towards a standard CP modelling language. In Bessiere, C. (Ed.), *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, Vol. 4741 of *Lecture Notes in Computer Science*, pp. 529–543. Springer.

Niepert, M., & Van den Broeck, G. (2014). Tractability through exchangeability: A new perspective on efficient probabilistic inference. In Brodley, C. E., & Stone, P. (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pp. 2467–2475. AAAI Press.

Poole, D. (2003). First-order probabilistic inference. In Gottlob, G., & Walsh, T. (Eds.), *IJCAI 2003, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pp. 985–991. Morgan Kaufmann.

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*, Vol. 2 of *Foundations of Artificial Intelligence*. Elsevier.

Roy, S., & Roth, D. (2018). Mapping to declarative knowledge for word problem solving. *Trans. Assoc. Comput. Linguistics*, *6*, 159–172.

Schmidt, G., & Ströhlein, T. (1993). *Homogeneous Relations*, pp. 5–27. Springer Berlin Heidelberg, Berlin, Heidelberg.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis,

D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*, 484–503.

Stanley, R. (2012). *Enumerative combinatorics*. Cambridge University Press, New York.

Suster, S., Fivez, P., Totis, P., Kimmig, A., Davis, J., De Raedt, L., & Daelemans, W. (2021). Mapping probability word problems to executable representations. In Moens, M., Huang, X., Specia, L., & Yih, S. W. (Eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 3627–3640. Association for Computational Linguistics.

Taghipour, N., Fierens, D., Davis, J., & Blockeel, H. (2012). Lifted variable elimination with arbitrary constraints. In Lawrence, N. D., & Girolami, M. A. (Eds.), *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, Spain, April 21-23, 2012*, Vol. 22 of *JMLR Proceedings*, pp. 1194–1202. JMLR.org.

Thurley, M. (2006). sharpsat - counting models with advanced component caching and implicit BCP. In Biere, A., & Gomes, C. P. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, Vol. 4121 of *Lecture Notes in Computer Science*, pp. 424–429. Springer.

Valiant, L. G. (1979). The complexity of computing the permanent. *Theor. Comput. Sci.*, *8*, 189–201.

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.

Van den Broeck, G., Meert, W., & Darwiche, A. (2014). Skolemization for weighted first-order model counting. In Baral, C., Giacomo, G. D., & Eiter, T. (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press.

Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., & De Raedt, L. (2011). Lifted probabilistic inference by first-order knowledge compilation. In Walsh, T. (Ed.), *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2178–2185. IJCAI/AAAI.

van Emden, M. H., & Kowalski, R. A. (1976). The semantics of predicate logic as a programming language. *J. ACM*, *23*(4), 733–742.

Zhang, D., Wang, L., Zhang, L., Dai, B. T., & Shen, H. T. (2020). The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE Trans. Pattern Anal. Mach. Intell.*, *42*(9), 2287–2305.