Calculi for Symmetric Queries*

Marc Gyssens[a,*], Jelle Hellings[b,a], Jan Paredaens[c], Dirk Van Gucht[d], Jef Wijsen[e], Yuqing Wu[f,1]

[a] *Hasselt University, Martelarenlaan 42, 3500 Hasselt, Belgium*
[b] *Exploratory Systems Lab, Department of Computer Science, University of California, Davis, CA 95616-8562, USA*
[c] *University of Antwerp, Middelheimlaan 1, 2020 Antwerpen, Belgium*
[d] *Indiana University, School of Informatics, Computing and Engineering, 919 E 10th Street, Bloomington, IN 47408, USA*
[e] *University of Mons, Place du Parc 20, 7000 Mons, Belgium*
[f] *Pomona College, 185 E 6th Street, Claremont, CA 91711, USA*

Abstract

Symmetric queries are introduced as queries on a sequence of sets of objects the result of which does not depend on the order of the sets. An appropriate data model is proposed, and two query languages are introduced, QuineCALC and SyCALC. They are correlated with the symmetric Boolean functions of Quine, respectively symmetric relational functions. The former correlation yields an incidence-based normal form for QuineCALC queries. More generally, we propose *counting-only* queries as those SyCALC queries the result of which only depends on incidence information, and characterize them as quantified Boolean combinations of QuineCALC queries. A normal form is proposed for them too. It is shown that, while it is undecidable whether a SyCALC query is counting-only, it is decidable whether a counting-only query is a QuineCALC query. Finally, some classical decidability problems are considered which are shown to be undecidable for SyCALC, but decidable for QuineCALC and counting-only queries.

Keywords: bag of sets data model, symmetric query, two-sorted first-order logic, two-sorted relational calculus, symmetric Boolean function, symmetric relational function, counting-only query, normal form, expressibility, satisfiability

*This is a revised and extended version of the conference paper "An Approach towards the Study of Symmetric Queries" by Marc Gyssens, Jan Paredaens, Dirk Van Gucht, Jef Wijsen, and Yuqing Wu, presented at the 40th International Conference on Very Large Data Bases (VLDB 2014), Hangzhou, China, September 1–5, 2014.

*Corresponding author.

Email addresses: marc.gyssens@uhasselt.be (Marc Gyssens), jhellings@ucdavis.edu (Jelle Hellings), jan.paredaens@uantwerpen.be (Jan Paredaens), vgucht@cs.indiana.edu (Dirk Van Gucht), jef.wijsen@umons.ac.be (Jef Wijsen), melanie.wu@pomona.edu (Yuqing Wu)

[1] Yuqing Wu carried out part of her work during a sabbatical visit to Hasselt University with a Senior Visiting Postdoctoral Fellowship of the Research Foundation Flanders (FWO).

1. Introduction

Many applications, several of which data-intensive, have to deal with sequences of sets of objects, where all objects are of the same type. Here are some classical examples:

- objects are parts, and $S_1, \ldots, S_n$ is a sequence of sets of parts such that S_j is the set of parts supplied by supplier j.
- objects are products, and $S_1, \ldots, S_n$ is a sequence of sets of products such that each S_j is the set of products bought in transaction j.
- objects are students, and $S_1, \ldots, S_n$ is a sequence of sets of students such that each S_j is the set of students taking course j.

Observe that, in all these examples, it is possible that $S_i = S_j$ for $i \neq j$. Indeed, two distinct suppliers may supply exactly the same parts; or two distinct transactions may involve exactly the same products; or two distinct courses may have exactly the same students enrolled in them. Other possible examples include companies and their customers, documents and the words contained therein, or RDF relationships involving pairs of objects [1–3].

In this paper, we study computable queries $\mathbf{q}(S_1, \ldots, S_n)$ that take as input a sequence of sets $S_1, \ldots, S_n$, $n \geq 0$, of objects of some common type, return as output a set of m-tuples of such objects (for some fixed value of $m \geq 0$), and satisfy, for each permutation $i_1, \ldots, i_n$ of $1, \ldots, n$,

$$\mathbf{q}(S_{i_1}, \ldots, S_{i_n}) = \mathbf{q}(S_1, \ldots, S_n).$$

We call such queries *symmetric queries*.

It should be emphasized at this point that, unlike m, the number n should not be considered as fixed, but rather as a *parameter* of the problem under consideration.

Obviously, the class of symmetric queries is a strict subset of the class of all computable queries that operate on sequences of sets. For example, the query returning the first set of the input sequence is clearly not symmetric. Nevertheless, the class of symmetric queries is quite rich. The following example queries, referring to the application with parts and suppliers, illustrate this.

1. Retrieve the parts that are supplied by at least two suppliers.
2. Retrieve the parts that are supplied by all suppliers.
3. Is each supplied part supplied by just one supplier?
4. Retrieve the parts that are supplied by exactly one supplier, provided that there exist parts that are supplied by at least three suppliers.
5. Do all suppliers supply the same parts?

6. Retrieve the pairs of parts that together are supplied by at least two suppliers.

7. Retrieve the pairs of parts supplied by exactly the same suppliers.

The above queries will be used in examples throughout the paper. We shall refer to them as Queries 1–7, respectively.

Wherever numbers of sets are mentioned in Queries 1–7, we chose small values for purposes of exposition. In the context of vast amounts of data, it is to be expected that these numbers will actually be quite large (e.g., variations on Query 6 in the context of the frequent-itemset problem [4]).

Not only symmetric queries, functions, and operators are prevalent in many fields. The same holds for the simple sequence-of-sets data model we use in this paper. Practical applications can be found in cluster computing, data-parallel computation on partitioned data, data analytics, and other Big Data techniques. In these applications, the commutative and associative nature of symmetric operators can be exploited to improve performance, as these operators can be ordered, grouped, combined, and merged arbitrarily. A good example of this is MapReduce, where the overall communication cost of the reduce step can be minimized by first reducing data at each computational node using a so-called combiner function, and only then redistributing the partly-reduced data, grouping them, and applying the final reduce step [5–8]. For this optimization to work, it suffices that the reducer and combiner functions are symmetric. Typically, it is up to the programmers to guarantee that this property is satisfied. The strategy followed in this paper is to propose expressive query languages that guarantee this property implicitly, and thus liberate programmers from having to argue for it explicitly.

MapReduce is often illustrated via the problem of *counting words* in documents and, based on these counts, make further decisions. This setting is closely related to itemset mining in transaction databases [9]. Additionally, decisions based on frequency of objects is also at the basis of many machine learning techniques [10]. We observe that the input to *counting words* is a sequence of documents, each document consisting of a set of words. As word-counts do not depend on the ordering of documents, we disregard the order of the sequence. Hence, the data is simply a bag of sets. These bags of sets have many alternative representations, such as bipartite graphs or binary many-to-many relations. We illustrate this in Figure 1. On the *left* is a sequence of documents, each a sequence of words. When disregarding the order, this sequence is a bag of sets of words and this bag of sets of words can alternatively be interpreted as the *bipartite graph*, shown in Figure 1, *middle*. On the *right*, the frequency of each word is provided, which is independent of the order of the documents and the order of the words within each document.

Symmetric functions are also prevalent in other fields, such as mathematics. As an example, symmetric polynomials play a fundamental role in finding roots of single-variable polynomials and finding solutions to systems of multivariable polynomial equations [11]. The study of these symmetric polynomials has a long history, and even dates back to fundamental results established by

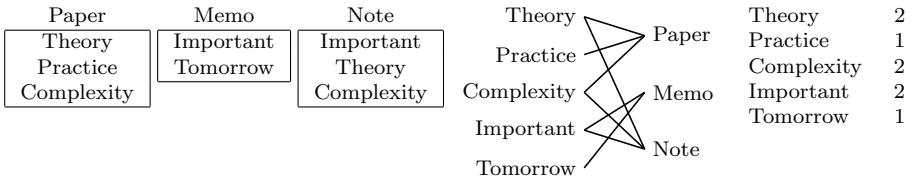


Figure 1: *Left*, a sequence of documents. *Middle*, the same dataset represented as a bipartite graph. *Right*, the frequency of words in this data set.

Isaac Newton [12]. In linear algebra, functions such as those that determine the rank, determinant, and eigenvalues of a square matrix are invariant under permutations of rows or columns [11, 13]. In statistics, most summary data are symmetric functions of the input, such as sum, count, average, median, maximum, minimum, variance, and higher-order moments. There is also a comprehensive literature on symmetric Boolean functions (e.g., [14–16]). In programming, examples of symmetric functions on lists of data include size, membership checking, and sorting. Furthermore, the `HAVING` clause of SQL reasons about incidence information, as in, e.g.,

```
SELECT   product, SUM(price * quantity)
FROM     Purchase
WHERE    date > ‘‘9/1’’
GROUP BY product
HAVING   SUM(quantity) > 30
```

It is therefore surprising that symmetric queries have hardly been studied in the context of database systems, even though our examples above show that symmetric queries are quite prevalent. We should note that certain special examples of symmetric queries have been considered in the context of nested relations and complex-object databases. For example, the “unnest” operator in the nested relational model [17] is an operator that, when applied to a set of sets, returns the union of these sets (see also, the “$\bigcup$” operator in NRC [18] and the “set-collapse” operator in the complex-object algebra [19]). Other examples of symmetric queries were introduced by Sarathy et al. [20], using the “$\bigcup$,” “$\bigcap$,” and the “$\bigoplus$” operators. Applied to a set of sets, “$\bigcup$” returns the union of these sets, “$\bigcap$” returns the intersection of these sets, and “$\bigoplus$” returns the set of objects that are members of just one of these sets.

Notice that Queries 1–7 above can be expressed in terms of union, intersection, complement, projection, and Cartesian product. Below, we give the

corresponding expression for each of these seven queries.[2]

$$\mathbf{q}_1(S_1,\ldots,S_n) = \bigcup_{1\le i<j\le n} S_i \cap S_j;$$

$$\mathbf{q}_2(S_1,\ldots,S_n) = \bigcap_{1\le i\le n} S_i;$$

$$\mathbf{q}_3(S_1,\ldots,S_n) = \overline{\pi_{\langle\rangle}\Big(\bigcup_{1\le i<j\le n} S_i \cap S_j\Big)};$$

$$\mathbf{q}_4(S_1,\ldots,S_n) = \Big(\Big(\bigcup_{1\le i\le n} S_i\Big) \cap \overline{\bigcup_{1\le i<j\le n} S_i \cap S_j}\Big) \times \pi_{\langle\rangle}\Big(\bigcup_{1\le i<j<k\le n} S_i \cap S_j \cap S_k\Big);$$

$$\mathbf{q}_5(S_1,\ldots,S_n) = \overline{\pi_{\langle\rangle}\Big(\bigcup_{1\le i<j\le n} S_i \cap \overline{S_j}\Big)};$$

$$\mathbf{q}_6(S_1,\ldots,S_n) = \bigcup_{1\le i<j\le n} (S_i \cap S_j) \times (S_i \cap S_j);$$

$$\mathbf{q}_7(S_1,\ldots,S_n) = \overline{\bigcup_{1\le i\le n} (S_i \times \overline{S_i}) \cup (\overline{S_i} \times S_i)}.$$

Observe that several of the above expressions can be rewritten using set difference instead of complement[3]. The latter is stronger, as $S_1 - S_2 = S_1 \cap \overline{S_2}$.[4]

To our knowledge, the class of symmetric queries that can be expressed using union, intersection, complement, projection, and Cartesian product, has not been studied. Initiating such a study is the purpose of the present paper.

For this study, we start from the work of Quine [16], who studied so-called symmetric Boolean functions which have as argument a sequence of sets of objects of a given length and return a set of objects defined in terms of the input sets using only union, intersection, and complement. Quine obtained the remarkable result that such a symmetric Boolean function can be entirely characterized in terms of the *incidence* of each object in the domain, i.e., the number of sets in which this object occurs. Concretely, given a sequence $S_1,\ldots,S_n$ of sets of objects as argument for the function, there is some subset N of $\{0,\ldots,n\}$ such that, for each object in the domain, this object is in the result of the function applied to $S_1,\ldots,S_n$ if and only if the number of sets among $S_1,\ldots,S_n$

[2] If S is a set, then $\pi_{\langle\rangle}(S) = \{\langle\rangle\}$ if $S \neq \emptyset$, and $\pi_{\langle\rangle}(S) = \emptyset$ if $S = \emptyset$. These are the only null-ary sets. We view "$\{\langle\rangle\}$" as a representation of **true** and "$\emptyset$" as a representation of **false**. In this way, Boolean queries can easily be expressed. Also notice that $T \times \{\langle\rangle\} = T$ and $T \times \emptyset = \emptyset$.

[3] With respect to some appropriately chosen domain.

[4] For domain-independent queries, complement and difference can be used interchangeably; we chose, however, not to impose additional semantic and/or syntactic restrictions which could obfuscate the focus of this work.

to which the object belongs to is in N. Moreover, this property *characterizes* symmetry of Boolean functions.

Returning to our example symmetric queries above, notice that Queries 1 and 2 have been expressed as symmetric Boolean queries in the sense of Quine.[5] For these queries, the set N in Quine's characterization result is $\{2, \ldots, n\}$, respectively $\{n\}$. Notice that this characterization allows for an efficient evaluation of these queries, as the relevant incidence information can be retrieved efficiently. All other queries are not expressed as symmetric Boolean functions in the sense of Quine, as the corresponding expressions involve projection and/or Cartesian product. Notice, however, that the expressions for Queries 3, 4, and 5 contain subexpressions representing symmetric Boolean functions in the sense of Quine. We may therefore hope that Quine's characterization can still be of use to evaluate also such queries efficiently. In sharp contrast with these three queries, the expressions for Queries 6 and 7 do not contain subexpressions representing symmetric Boolean functions in the sense of Quine. This should not be too surprising if we look at the semantics of these symmetric queries. For example, if we look at Query 6, "Retrieve the pairs of parts that together are supplied by at least two suppliers," or Query 7, "Retrieve the pairs of parts supplied by exactly the same suppliers," knowing the number of suppliers for each part is not very helpful for answering them. Not only these example queries, but also the *word counting* problem illustrated in Figure 1, which is at the basis of decision-based systems, underline the relationship between symmetric queries and counting.

In order to study the issues raised above more closely, we first want to get rid of the explicit occurrence of n in the model considered so far, which is undesirable from a database perspective. To see this, consider again parts and suppliers. The interesting setting is of course a dynamic one where new suppliers start up their business all the time and old ones go out of business. Unfortunately, the number of suppliers n is "hard-wired" in the expressions given above for our example queries. Changing n will yield another expression. Thus, to overcome this limitation, we need a data model for representing sequences of sets of arbitrary length. In such a model, we must moreover be able to define query languages for specifying symmetric queries without making explicit reference to the length of the represented sequence of sets.

Concretely, we propose to model an arbitrary sequence of sets by a set σ of set names, one for each entry in the sequence, and a binary membership relation γ. In this representation, a set name S in σ represents the set of all objects o for which $\langle o, S\rangle \in \gamma$. Notice that we need the set σ because some sets in the sequence under consideration may be empty and hence will not occur in γ. In the representation we propose, we of course lose the order of the sets in the sequence, but this is irrelevant in our setting as all queries under consideration are symmetric anyway.

In this paper, we propose as a query language a two-sorted first-order logic

[5] Technically, one for each value of the parameter n.

over a binary predicate Γ representing the set membership relation of our data model, called SyCALC (from "Symmetric Calculus"). As mentioned, SyCALC has two sorts of variables: one ranges over set names and one over objects. The language is designed in such a way that the only comparisons allowed are between set variables. Of course, we will ensure that only symmetric queries can be expressed in SyCALC. As an illustration, Query 6 is expressed in SyCALC by $\{(x, y) \mid \exists X \exists Y\ \Gamma(x, X) \wedge \Gamma(x, Y) \wedge \Gamma(y, X) \wedge \Gamma(y, Y) \wedge (X \neq Y)\}$. Our considerations above lead naturally to the following research questions:

1. Is there a syntactically definable fragment of SyCALC that is a conservative extension of the symmetric Boolean functions in the sense of Quine?

2. If so, let us call this fragment QuineCALC. Can the characterization result of Quine for symmetric Boolean functions using incidence information be lifted to a characterization of QuineCALC?

3. It is possible to extend the symmetric Boolean functions in the sense of Quine to what we call *symmetric relational functions* by also allowing projection and Cartesian product besides union, intersection, and complement. Is SyCALC a conservative extension of the symmetric relational functions?

4. Are there unary symmetric queries that are expressible in SyCALC but not in QuineCALC which can nevertheless be characterized in terms of incidence information?

5. Are there also non-unary symmetric queries expressible in SyCALC which can be characterized in terms of incidence information?

6. We shall call SyCALC queries that can be expressed in terms of incidence information *counting-only*. Are there SyCALC queries that are not counting-only?

7. Is there a syntactically definable fragment of SyCALC that expresses precisely the counting-only SyCALC queries?

8. Is it decidable whether a counting-only SyCALC query is a QuineCALC query? Is it decidable whether a SyCALC query is counting-only?

9. Finally, we may consider decidability problems such as satisfiability, containment, equivalence, validity, or emptiness. Are these problems decidable for SyCALC queries? Or for counting-only SyCALC queries? Or for QuineCALC queries?

In this paper, we show that the answer to Research Questions 1–7 is "yes." As for Research Question 8, it is decidable whether a counting-only SyCALC query is a QuineCALC query, but it is *not* decidable whether a SyCALC query is counting-only. As for Research Question 9, finally, we show that the problems considered are decidable for counting-only SyCALC queries and QuineCALC queries, but *not* for general SyCALC queries.

This paper is organized as follows. In Section 2, we elaborate some more on related work, both from the present authors and other authors. In Section 3, we present our data model. We introduce symmetric queries over our data model as well as functions on finite sequences of sets of a given length, and correlate both. In Section 4, we introduce QuineCALC, and establish a correspondence between QuineCALC queries and symmetric Boolean functions. We also characterize QuineCALC queries in terms of incidence information of the objects they return. In Section 5, we introduce SyCALC, and establish a correspondence between SyCALC queries and symmetric relational functions. We also introduce counting-only SyCALC queries, which we characterize as quantified Boolean combinations of QuineCALC queries. In Section 6, we show that, while it is undecidable whether a SyCALC query is counting-only, it is decidable whether a counting-only SyCALC query is equivalent to a QuineCALC query. We also show that the problems mentioned in Research Question 9 are decidable for counting-only SyCALC queries and QuineCALC queries, but not for general SyCALC queries. Finally, in Section 7, we formulate some conclusions, and discuss direction for future research.

2. Related work

This is a revised and extended version of Gyssens et al. [21]. Not only did we add full proofs, but we also added additional results with respect to decision problems (Research Questions 8 and 9). More specifically, we answered several questions that were stated as open problems in Gyssens et al. [21]. First, we provide results on whether it is decidable if a SyCALC query is a QuineCALC query. We also provide results on the behavior of SyCALC and QuineCALC queries with respect to traditional decision problems such as satisfiability, containment, equivalence, validity, or emptiness.

Additionally, the work presented in this paper inspired us to further study the concept of "counting-only" in more depth as stated in the Conclusions and Future Work section, under "Extensions of the concept 'counting-only'." As it turns out, the counting-only queries we study in this paper are only one fragment of a much larger class of counting-based queries that are well-behaved and intuitive to understand. Hellings et al. [22] study these counting-based queries and address the questions raised in this paper.

This work is inspired on the one hand by the work on symmetric Boolean functions [14–16]), and on the other hand by the occurrence in practice of several counting-based queries—including the common statistical queries—which by nature are all symmetric. Ample examples of operators and formula expressing such queries can be found in the literature, including, e.g., [5–9, 17–20]. Despite there being numerous examples of symmetric queries, both in the literature and in practice, we believe, as mentioned in the Introduction, that this is to the best of our knowledge the first systematic study of symmetric queries.

3. Preliminaries

As explained in the Introduction, we work with two sorts, objects and sets of these objects. We assume the existence of an infinitely enumerable domain $\mathcal{D}$ of objects and an infinitely enumerable domain $\mathcal{S}$ of names of sets of objects. For clarity of exposition, we shall distinguish between sets and set names in this section by denoting the former with (possibly subscripted) capital letters, such as $S_1, S_2, S_3, \ldots$, and the latter with (possibly subscripted) accented capital letters, such as $S'_1, S'_2, S'_3, \ldots$ We shall always implicitly assume that each object under consideration is in $\mathcal{D}$, and each set name under consideration is in $\mathcal{S}$.

For our data model, we consider *structures* $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, where σ is a finite subset of $\mathcal{S}$, expliciting the set names under consideration in the structure, and γ is a finite subset of $\mathcal{D} \times \sigma$, providing set membership information. Hence, for all S' in σ, S' is the name of the set $\{o \mid o \in \mathcal{D} \;\&\; \langle o, S' \rangle \in \gamma\}$. Notice that this set may be empty: all set names in σ not occurring in the set membership relation γ represent the empty set.

For each o in $\mathcal{D}$, we define the *incidence* of o in γ as $\mathsf{inc}(o, \gamma) = |\{S' \in \sigma \mid \langle o, S' \rangle \in \gamma\}|$, i.e., the number of sets under consideration to which o belongs.[6] Similarly we define the *co-incidence* of o in γ as $\mathsf{coinc}(o, \gamma) = |\{S' \in \sigma \mid \langle o, S' \rangle \notin \gamma\}|$, i.e., the number of sets under consideration to which o does not belong. Clearly, $\mathsf{coinc}(o, \gamma) = |\sigma| - \mathsf{inc}(o, \gamma)$.

In the work of Quine [16], symmetric Boolean functions operate on a finite sequence of sets. We now explain formally how such a sequence can be encoded in our model. Thereto, let $S_1, \ldots, S_n$ be a sequence of sets, and let $S'_1, \ldots, S'_n$ be a sequence of pairwise different set names. Then, the *encoding* of $S_1, \ldots, S_n$ given set names $S'_1, \ldots, S'_n$, denoted by $\mathsf{enc}(S_1, \ldots, S_n; S'_1, \ldots, S'_n)$, is the structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, where $\sigma = \{S'_1, \ldots, S'_n\}$ and γ is defined by

$$\gamma = \{\langle o, S'_i \rangle \mid 1 \leq i \leq n \;\&\; o \in S_i\}.$$

Notice that, whenever $i_1, \ldots, i_n$ is a permutation of $1, \ldots, n$, then

$$\mathsf{enc}(S_1, \ldots, S_n; S'_1, \ldots, S'_n) = \mathsf{enc}(S_{i_1}, \ldots, S_{i_n}; S'_{i_1}, \ldots, S'_{i_n}).$$

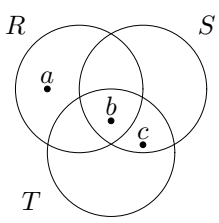
Conversely, if $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ is a structure with $\sigma = \{S'_1, \ldots, S'_n\}$, and $S_1, \ldots, S_n$ are the sets represented by $S'_1, \ldots, S'_n$, respectively, then $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ equals $\mathsf{enc}(S_1, \ldots, S_n; S'_1, \ldots, S'_n)$. This converse encoding is not unique, of course, as $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ also encodes every permutation of $S_1, \ldots, S_n$, as shown above.

We notice that the encoding of a sequence of sets omits any ordering information. Hence, all permutations of a sequence will yield isomorphic structures (that possibly differ in set names). This is on purpose, the main focus of this work are *symmetric queries* on sequences of sets, queries which do not rely on ordering information in the sequence.

[6] Observe that this number does not depend on $\mathcal{D}$ or $\mathcal{S}$, justifying the notation.

If we denote by $\mathsf{inc}(o, S_1, \ldots, S_n)$ the incidence of o in the sequence of sets $S_1, \ldots, S_n$, i.e., the number of sets in this sequence to which o belongs, then, clearly, $\mathsf{inc}(o, S_1, \ldots, S_n) = \mathsf{inc}(o, \gamma)$ in the encoding.

Example 1. Consider the sets R, S, T, and U visualized by the Venn diagram in Figure 2, *left*. (Elements of $\mathcal{D}$ not in R, S, or T are not shown.) Furthermore, we assume that U is empty. The sequence R, S, T, U (or any of the 24 permutations thereof) is encoded by the structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, where $\sigma = \{R', S', T', U'\}$ and the binary membership relation γ is shown in Figure 2, *right*.



γ

Object	Set
a	R'
b	R'
b	S'
b	T'
c	S'
c	T'

Figure 2: Encoding of a finite sequence of sets by a binary membership relation.

In this example, we have $\mathsf{inc}(a, R, S, T, U) = \mathsf{inc}(a, \gamma) = 1$, $\mathsf{inc}(b, R, S, T, U) = \mathsf{inc}(b, \gamma) = 3$, and $\mathsf{inc}(c, R, S, T, U) = \mathsf{inc}(c, \gamma) = 2$.

As explained in the Introduction, we consider (symmetric) queries at two levels: a restricted "static" level, in which we consider as input sequences of sets of a given length, and a "dynamic" level, in which this restriction is removed by encoding the sequence of sets into a structure as defined above.

Inspired by the terminology of Quine [16], we shall speak of *functions* on sequences of sets at the "static" level. Such a function f, taking as arguments a sequence of n sets, for some fixed $n \geq 0$, and returning m-tuples of objects of these sets, for some fixed $m \geq 0$, is called *symmetric* if, for all sequences of sets $S_1, \ldots, S_n$ and for all permutations $i_1, \ldots, i_n$ of $1, \ldots, n$, $f(S_{i_1}, \ldots, S_{i_n}) = f(S_1, \ldots, S_n)$.

At the "dynamic" level, we shall speak of *queries*. A query $\mathbf{q}$ takes as input a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ and maps it to a subset of $\mathcal{D}^m$ for some fixed $m \geq 0$. We say that $\mathbf{q}$ is *symmetric* if, for all permutations π of $\mathcal{S}$ and for all structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, $\mathbf{q}((\mathcal{D}, \mathcal{S}, \pi(\sigma), \pi(\gamma)) = \mathbf{q}((\mathcal{D}, \mathcal{S}, \sigma, \gamma))$, where $\pi(\sigma) = \{\pi(S') \mid S' \in \sigma\}$ and $\pi(\gamma) = \{\langle o, \pi(S')\rangle \mid \langle o, S'\rangle \in \gamma\}$. This condition formalizes the intuition that symmetric queries only look at the content of the sets and not at their names.

If $\mathbf{q}$ is symmetric, then, for all sequences of sets $S_1, \ldots, S_n$, for all sequences of pairwise different set names $T'_1, \ldots, T'_n$ and $U'_1, \ldots, U'_n$, and for all permutations $i_1, \ldots, i_n$ of $1, \ldots, n$,

$$\mathbf{q}(\mathsf{enc}(S_1, \ldots, S_n; T'_1, \ldots, T'_n)) = \mathbf{q}(\mathsf{enc}(S_{i_1}, \ldots, S_{i_n}; U'_1, \ldots, U'_n)),$$

matching the notion of symmetric functions at the static level.

The "static" level and the "dynamic" level are of course closely interconnected.

For a fixed value of $n \geq 0$, we can associate with each symmetric query $\mathbf{q}$ a function $f_{\mathbf{q},n}$ on sequences of n sets $S_1, \ldots, S_n$ defined by

$$f_{\mathbf{q},n}(S_1, \ldots, S_n) := \mathbf{q}(\mathsf{enc}(S_1, \ldots, S_n; S'_1, \ldots, S'_n)),$$

where $S'_1, \ldots, S'_n$ is an arbitrary sequence of pairwise different set names.[7] The above property guarantees that $f_{\mathbf{q},n}$ is both well-defined and symmetric. Since n is a parameter in this construction, we actually obtain a *family* of symmetric functions, one for each value of n.

Conversely, consider a family $F = \{f_n \mid n \geq 0\}$ of symmetric functions such that f_n, $n \geq 0$, operates on sequences of n sets and returns output of arity independent of n. Then, we can associate with F a query $\mathbf{q}_F$ operating on structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ as follows:

$$\mathbf{q}_F(\mathcal{D}, \mathcal{S}, \sigma, \gamma) := f_n(S_1, \ldots, S_n),$$

where n is the cardinality of σ and $S_1, \ldots, S_n$ is the sequence of sets (in some order) represented by the set names in σ. The well-definedness of $\mathbf{q}_F$ relies on the symmetry of $f_0, f_1, f_2, \ldots$. Clearly, $f_{\mathbf{q}_F,n} = f_n$.

Notice that the mathematical construction detailed above corresponds to a definite reality. Indeed, in all examples of symmetric functions on sequences of sets $S_1, \ldots, S_n$ presented in the Introduction, the number n is in fact a *parameter*. Hence, it is indeed fair to say that, in all the cases, we have been dealing with a *family* of symmetric functions, one for each value of n, rather than with just one symmetric function for some fixed value of n.

Remark 2. As we have seen above, the particular names that are chosen to represent sets in a structure are immaterial in the context of symmetric queries. To simplify notation, we shall therefore no longer make an explicit distinction in what follows between the sets that are encoded and the corresponding set names, and use (possibly subscripted) capital letters such as $S_1, S_2, S_2, \ldots$ for both. In the same vein, we shall henceforth no longer refer explicitly to the particular set names used in an encoding of a sequence of sets.

In this paper, we shall establish interconnections between particular classes of symmetric queries and particular classes of symmetric functions on sequences of sets. We must point out, though, that our main focus is the study of symmetric queries.

4. QuineCALC

We now define a first-order language, called QuineCALC, of which we show that it is a conservative extension of the symmetric Boolean functions in the

[7] Since the choice of the set names $S'_1, \ldots, S'_n$ is arbitrary, we shall henceforth abbreviate $\mathsf{enc}(S_1, \ldots, S_n; S'_1, \ldots, S'_n)$ to $\mathsf{enc}(S_1, \ldots, S_n)$, by slight abuse of notation. See also Remark 2.

sense of Quine. Later, in Section 5, we will extend QuineCALC to SyCALC, the language which is at the core of this study.

4.1. Language definition

QuineCALC is a restricted first-order logic with a single binary relation name Γ representing set membership, i.e., $\Gamma(x, X)$ means that object x belongs to the set named X.

The alphabet contains two sorts of variables: *lowercase* variables $x, y, z, \ldots$ and *uppercase* variables $X, Y, Z, \ldots$, possibly subscripted. Lowercase variables denote objects and uppercase variables denote set names. The alphabet contains no constant symbols.

QuineCALC *formulae* are defined by the following syntax rule:

$$\varphi := \Gamma(x, X) \mid X = Y \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi_1 \mid \exists X\, \varphi_1.$$

We also allow the usual abbrevations, such as $X \neq Y$, $\varphi_1 \wedge \varphi_2$, and $\forall X\, \varphi$. Observe that the (in)equality predicate and existential quantification operate on uppercase variables only. Since the language has no quantification over lowercase variables, all occurrences of lowercase variables in a QuineCALC formula must be free.

A QuineCALC *query* $\{x \mid \varphi(x)\}$ is defined by a QuineCALC formula with exactly one lowercase variable x and without free occurrences of uppercase variables.

Given a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, a QuineCALC query is evaluated in the usual way, where lowercase (object) variables range over $\mathcal{D}$ and uppercase (set name) variables range over σ. Observe that equality or inequality of uppercase variables refers to the equality or inequality of the set *names* to which they are evaluated, and not the contents of the corresponding sets! The binary relation symbol Γ is interpreted as the membership relation γ. Observe that QuineCALC queries are symmetric by their definition: set names are always quantified and the language does not allow referencing specific set names via constants.

For $o \in \mathcal{D}$, we denote by $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o)$ that $\varphi(x)$ evaluates to **true** in the structure under consideration if x is substituted by o. For $n \geq 0$, we say that two QuineCALC queries $\{x \mid \varphi_1(x)\}$ and $\{x \mid \varphi_2(x)\}$ are *n-equivalent* if, for all structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ with $|\sigma| = n$, and for all objects $o \in \mathcal{D}$, $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi_1(o)$ if and only if $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi_2(o)$. Two QuineCALC queries are *equivalent* if they are n-equivalent for all $n \geq 0$.

Example 3. The QuineCALC query $\{x \mid \exists X \exists Y\, (\Gamma(x, X) \wedge \Gamma(x, Y) \wedge (X \neq Y))\}$ expresses Query 1 and the QuineCALC query $\{x \mid \neg\exists X\, \neg\Gamma(x, X)\}$ expresses Query 2 in the Introduction.

In the following example, we present QuineCALC queries which will be used throughout this paper.

Example 4. For evey natural number $i \geq 0$, the query that upon input the structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ returns the objects that belong to at least i sets of σ

according to the membership information in γ is expressed by the QuineCALC query

$$\{x \mid \exists X_1 \cdots \exists X_i \, (\bigwedge_{1 \leq j < k \leq i} X_j \neq X_k) \wedge (\bigwedge_{1 \leq j \leq i} \Gamma(x, X_j))\}.$$

We shall denote the QuineCALC formula in this query by $\mathsf{gteq}(x, i)$. The query that returns the objects that belong to *exactly* i sets of σ is then expressed by the QuineCALC query $\{x \mid \mathsf{gteq}(x, i) \wedge \neg \mathsf{gteq}(x, i+1)\}$. We shall denote the QuineCALC formula in this query by $\mathsf{eq}(x, i)$. We shall also consider the query that returns the objects that do *not* belong to at least i sets of σ (or, equivalently, the objects that belong to at most $|\sigma| - i$ sets of σ), which is expressed by the QuineCALC query

$$\{x \mid \exists X_1 \cdots \exists X_i \, (\bigwedge_{1 \leq j < k \leq i} X_j \neq X_k) \wedge (\bigwedge_{1 \leq j \leq i} \neg\Gamma(x, X_j))\}.$$

We shall denote the QuineCALC formula in this query by $\mathsf{cogteq}(x, i)$. The query that returns the objects that do *not* belong to *exactly* i sets of σ (or, equivalently, the objects that belong to *exactly* $|\sigma| - i$ sets of σ) is then expressed by the QuineCALC query $\{x \mid \mathsf{cogteq}(x, i) \wedge \neg \mathsf{cogteq}(x, i+1)\}$. We shall denote the QuineCALC formula in this query by $\mathsf{coeq}(x, i)$.

4.2. QuineCALC and symmetric Boolean functions

Obviously, the class of sets that can be specified by QuineCALC queries given a particular structure as input is closed under union, intersection, and complement. We will take this observation one step further, and show that QuineCALC is a conservative extension of the symmetric Boolean functions in the sense of Quine, thereby solving Research Question 1. Thereto, we introduce the following terminology.

Definition 5. Let $n \geq 0$, and let f be a symmetric function operating on sequences of n sets of objects and returning sets of these objects, and let $\mathbf{q} := \{x \mid \varphi(x)\}$ be a QuineCALC query. We say that $\mathbf{q}$ is *n-equivalent* to f, denoted $\mathbf{q} \equiv_n f$, if, for all sequences of n sets $S_1, \ldots, S_n$ and for all objects $o \in \mathcal{D}$, we have that o is in $f(S_1, \ldots, S_n)$ if and only if $\mathsf{enc}(S_1, \ldots, S_n) \models \varphi(o)$.

Intuitively, $\mathbf{q} \equiv_n f$ says that $\mathbf{q}$ and f return the same values on inputs consisting of sequences of n sets, provided this input is appropriately encoded for applying QuineCALC queries.

We now formally define *Boolean functions* and *symmetric Boolean functions* in the sense of Quine.

Definition 6. Let $n \geq 0$. A (symmetric) function operating on sequences of n sets of objects $S_1, \ldots, S_n$ is called *Boolean* if the output is again a set of objects, and this set can be described as a Boolean combination of $S_1, \ldots, S_n$ (using union, intersection, and complement).

The following two theorems link QuineCALC queries with symmetric Boolean functions, one for each direction.

Theorem 7. *For every QuineCALC query* $\mathbf{q}$*, and for every integer* $n \geq 0$*, there exists a symmetric Boolean function* $f_{\mathbf{q},n}$ *operating on sequences of* n *sets such that* $\mathbf{q} \equiv_n f_{\mathbf{q},n}$*.*

Proof. Let $\mathbf{q} := \{x \mid \varphi(x)\}$ be a QuineCALC query and let $n \geq 0$. The operator $\mathsf{qe}(\cdot)$ eliminates existential quantifiers from QuineCALC queries, and is defined as follows, where $1 \leq i, j \leq n$:

$$\begin{aligned}
\mathsf{qe}(\Gamma(x, S_i)) &= \Gamma(x, S_i);\\
\mathsf{qe}(S_i = S_j) &= \begin{cases} \mathbf{true} & \text{if } i = j,\\ \mathbf{false} & \text{if } i \neq j; \end{cases}\\
\mathsf{qe}(\varphi_1 \vee \varphi_2) &= \mathsf{qe}(\varphi_1) \vee \mathsf{qe}(\varphi_2);\\
\mathsf{qe}(\neg\varphi_1) &= \neg\mathsf{qe}(\varphi_1);\\
\mathsf{qe}(\exists X\, \varphi_1) &= \bigvee_{1 \leq i \leq n} \mathsf{qe}(\varphi_1[X \to S_i]).
\end{aligned}$$

In the last line above, $\varphi_1[X \to S_i]$ denotes the expression obtained from φ_1 by replacing each free occurrence of X with S_i and empty disjunctions are interpreted as "**false**".

We next compute $\mathsf{fun}(\mathsf{qe}(\varphi))$ as follows, where $1 \leq i \leq n$:

$$\begin{aligned}
\mathsf{fun}(\mathbf{true}) &= \mathcal{D};\\
\mathsf{fun}(\mathbf{false}) &= \emptyset;\\
\mathsf{fun}(\Gamma(x, S_i)) &= S_i;\\
\mathsf{fun}(\varphi_1 \vee \varphi_2) &= \mathsf{fun}(\varphi_1) \cup \mathsf{fun}(\varphi_2);\\
\mathsf{fun}(\neg\varphi_1) &= \overline{\mathsf{fun}(\varphi_1)}.
\end{aligned}$$

Above, we assume that "$\mathcal{D}$" and "$\emptyset$" are abbreviations of "$\bigcap_{1 \leq i \leq n} S_i \cup \overline{S_i}$" and "$\bigcup_{1 \leq i \leq n} S_i \cap \overline{S_i}$," respectively, symmetric expressions which always return the intended value, even in the limit case $n = 0$.

It is now straightforward that the expression $\mathsf{fun}(\mathsf{qe}(\varphi))$ defines a symmetric Boolean function $f_{\mathbf{q},n}(S_1, \ldots, S_n)$ on sequences of n sets for which $\mathbf{q} \equiv_n f_{\mathbf{q},n}$. □

Observe that the last rule for the computation of $\mathsf{qe}(\cdot)$ reveals in which way n occurs as a parameter in $f_{\mathbf{q},n}$.

Example 8. Consider the QuineCALC queries in Example 3, expressing Queries 1 and 2. Choose $n = 3$. Then the symmetric Boolean functions on sequences of three sets S_1, S_2, S_3 that are 3-equivalent to these QuineCALC queries are, after some straightforward simplifications, defined by the expressions $(S_1 \cap S_2) \cup (S_1 \cap S_3) \cup (S_2 \cap S_3)$ and $S_1 \cap S_2 \cap S_3$, respectively.

Conversely, Theorem 10 below explains how to translate symmetric Boolean functions on sequences of n sets into QuineCALC queries. The proof of Theorem 10 relies on the following property, due to Quine [16, p. 178] (slightly adapated to our notations and terminology):

Lemma 9 (Quine [16])**.** *For a Boolean function f on sequences of $n \geq 0$ sets of objects, the following statements are equivalent:*

1. *f is symmetric;*
2. *there exists a finite set N of natural numbers such that, for all sequences of sets $S_1, \ldots, S_n$ and all objects o, $o \in f(S_1, \ldots, S_n)$ if and only if $\mathsf{inc}(o, S_1, \ldots, S_n) \in N$.*

Theorem 10. *For every integer $n \geq 0$ and for every symmetric Boolean function f_n on sequences of n sets of objects, there exists a QuineCALC query $\mathbf{q}_{f_n}$ such that $\mathbf{q}_{f_n} \equiv_n f_n$.*

Proof. Let N be the set of natural numbers characterizing the symmetric Boolean function f_n in the statement of Theorem 10 in the sense of Lemma 9. Consider the QuineCALC query $\mathbf{q}_{f_n} := \{x \mid \varphi(x)\}$ where $\varphi(x)$ is **false** if $N = \emptyset$ and

$$\bigvee_{i \in N} \mathsf{eq}(x, i)$$

otherwise. It is straightforward that $\mathbf{q}_{f_n} \equiv_n f_n$. □

Example 11. We revisit Example 8.

First consider the symmetric Boolean function $f_3(S_1, S_2, S_3) = (S_1 \cap S_2) \cup (S_1 \cap S_3) \cup (S_2 \cap S_3)$. For this function, the characterizing set N according to Lemma 9 equals $\{2, 3\}$. Hence, it follows from Theorem 10 that $\mathbf{q}_{f_3} \equiv_3 f_3$, where

$$\mathbf{q}_{f_3} := \{x \mid \mathsf{eq}(x, 2) \vee \mathsf{eq}(x, 3)\}.$$

The QuineCALC query in Example 3 (from which f_3 was derived in Example 8) can be rewritten as $\{x \mid \mathsf{gteq}(x, 2)\}$. The latter QuineCALC query is 3-equivalent to $\mathbf{q}_{f_3}$, and, hence, they are both 3-equivalent to f_3. Notice, however, that both QuineCALC queries are *not* equivalent: they are not even 4-equivalent.

For the other symmetric Boolean function in Example 8, $g_3(S_1, S_2, S_3) = S_1 \cap S_2 \cap S_3$, we have that $N = \{3\}$. Hence, $\mathbf{q}_{g_3} \equiv_3 g_3$, with

$$\mathbf{q}_{g_3} := \{x \mid \mathsf{eq}(x, 3)\}.$$

The QuineCALC query in Example 3 from which g_3 was derived in Example 8 can be rewritten as $\{x \mid \mathsf{coeq}(x, 0)\}$. The latter QuineCALC query is 3-equivalent to $\mathbf{q}_{g_3}$, and, hence, they are both 3-equivalent to g_3. Notice, however, that both QuineCALC queries are *not* equivalent: they are not even 4-equivalent.

Theorems 7 and 10 together settle Research Question 1: QuineCALC (which will turn out to be a syntactically definable fragment of SyCALC in Section 5) is a conservative extension of the fixed-arity symmetric Boolean functions.

From Theorem 7 and Lemma 9, we can immediately derive the following corollary.

Corollary 12. *Let $\{x \mid \varphi(x)\}$ be a QuineCALC query and let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ be a structure. Let $o_1, o_2 \in \mathcal{D}$ such that $\mathsf{inc}(o_1, \gamma) = \mathsf{inc}(o_2, \gamma)$. Then $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_1)$ if and only if $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_2)$.*

4.3. QuineCALC and counting

In Section 4.2, we already established a correspondence between QuineCALC queries and incidence information, provided we only consider structures where n, the number of set names under consideration, is fixed. How does this incidence information for different values of n relate to each other? We provide an answer to that question in Theorem 13, below.

Theorem 13. *Let* $\mathbf{q} := \{x \mid \varphi(x)\}$ *be a QuineCALC query for which* $\varphi(x)$ *has quantifier depth* $q \geq 0$*. Then, there exists a QuineCALC query* $\mathbf{q}_{\text{inc}} = \{x \mid \psi(x)\}$ *where* ψ *is a disjunction of subformulae of the form* $\mathsf{eq}(x, i)$ *(*$0 \leq i < q$*), subformulae of the form* $\mathsf{coeq}(x, j)$ *(*$0 \leq j < q$*), and at most one subformula of the form* $\mathsf{gteq}(x, q) \wedge \mathsf{cogteq}(x, q)$*, such that, for all* $n \geq 2q-1$*,* $\mathbf{q}$ *is* n*-equivalent to* $\mathbf{q}_{\text{inc}}$*.*

To put Theorem 13 into perspective, recall that Quine's result states, that, for every symmetric Boolean function f on n sets, there exists $N \subseteq \{0, 1, 2, \ldots, n\}$ such that f is equivalent to the following query: *"return precisely those objects whose incidence belongs to* N*."* Analogously, Theorem 13 states that, for every QuineCALC query $\mathbf{q} = \{x \mid \varphi(x)\}$ with quantifier depth q, there exist two sets $N_1, N_2 \subseteq \{\{0\}, \{1\}, \{2\}, \ldots, \{q-1\}, \{n \in \mathbb{N} \mid n \geq q\}\}$ such that on structures with at least $2q-1$ sets, $\mathbf{q}$ is equivalent to the following query: *"return precisely those objects whose incidence belongs to* $\bigcup N_1$ *or whose co-incidence belongs to* $\bigcup N_2$*."* Two remarks are in place:

- If $\bigcup N_1$ contains some number that is greater than or equal to q, then $\bigcup N_1$ contains all numbers that are greater than or equal to q, and likewise for $\bigcup N_2$. The reason is that a QuineCALC query with quantifier depth q can verify whether the number of sets an object x belongs to (or does not belong to) is equal to $0, 1, 2, \ldots, q-1$, or strictly greater than $q-1$. Intuitively, a QuineCALC query with quantifier depth q can count up to, but not beyond $q-1$.

- In a structure with at least $2q-1$ sets, the incidence and the co-incidence of an object cannot both be smaller than or equal to $q-1$. Therefore, if the incidence of an object is smaller than or equal to $q-1$, then its co-incidence must necessarily be greater than or equal to q. Symmetrically, if the co-incidence of an object is smaller than or equal to $q-1$, then its incidence must necessarily be greater than or equal to q.

One way of proving Theorem 13 is by using Ehrenfeucht-Fraïssé games. Here, we choose for a more constructive approach. Lemma 14, below, generalizes Theorem 13 to arbitrary *sub*formulae of QuineCALC formulae. This generalization allows for a proof by structural induction. The details of this proof reveal how we must transform a QuineCALC formule bottom up starting from its constituent atoms until the entire formula is in the form required by Theorem 13.

Lemma 14. *Let $\{x \mid \varphi(x)\}$ be a* QuineCALC *query, with quantifier depth q. When restricted to structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ with $n = |\sigma| \geq 2q-1$, all subformulas $\varrho(x, X_1, \ldots, X_r)$ of φ, $0 \leq r \leq q$, can be rewritten as*[8]

$$\bigvee_{1 \leq i \leq m} \varrho_i(x, X_1, \ldots, X_r) \tag{1}$$

with

- *$m \geq 0$ and,*
- *for $i = 1, \ldots, m$, ϱ_i equals*

$$\psi_i \wedge c_i \wedge {}^{s_{i1}}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_{ir}}\Gamma(x, X_r),$$

where,[9]

  - *ψ_i is $\bigwedge_{1 \leq j < k \leq r} X_j \; \theta_{ijk} \; X_k$ with θ_{ijk} either "$=$" or "$\neq$";*
  - *c_i is of the form $\mathsf{eq}(x, n_i)$, with $n_i < q$, or of the form $\mathsf{coeq}(x, n_i)$, with $n_i < q$, or of the form $\mathsf{gteq}(x, q) \wedge \mathsf{cogteq}(x, q)$; and*
  - *for $j = 1, \ldots, r$, s_{ij} is either "$+$" or "$-$", where ${}^{+}\Gamma(x, X_j)$ must be interpreted as "$\Gamma(x, X_j)$" and ${}^{-}\Gamma(x, X_j)$ must be interpreted as "$\neg\Gamma(x, X_j)$".*

Proof. The proof goes by structural induction.

Base cases: The building blocks of QuineCALC formulae are expressions of the form $\Gamma(x, X)$ and expressions of the form $X = Y$.

1. An expression of the form $\Gamma(x, X)$ can be rewritten as

$$\mathbf{true} \wedge \mathbf{true} \wedge {}^{+}\Gamma(x, X).$$

   The first "**true**" can be seen as an empty conjunction (cf. Footnote 9). The second "**true**" can be rewritten as

$$\mathsf{eq}(x, 0) \vee \ldots \vee \mathsf{eq}(x, q-1) \vee (\mathsf{gteq}(x, q) \wedge \mathsf{cogteq}(x, q)) \vee \mathsf{coeq}(x, q-1) \vee \ldots \vee \mathsf{coeq}(x, 0).$$

   Using distributivity, we can rewrite the above expression for $\Gamma(x, X)$ as a disjunction of subformulae of the required form.

2. An expression of the form $X = Y$, where X and Y are different set name variables, can be rewritten as

$$\big((X = Y) \wedge \mathbf{true} \wedge {}^{+}\Gamma(x, X) \wedge {}^{+}\Gamma(x, Y)\big) \vee \big((X = Y) \wedge \mathbf{true} \wedge {}^{-}\Gamma(x, X) \wedge {}^{-}\Gamma(x, Y)\big).$$

[8] Empty disjunctions are always interpreted as "**false**."
[9] Empty conjunctions are always interpreted as "**true**."

An expression of the form $X = X$, which always evaluates to **true**, can be rewritten as

$$\big(\mathbf{true} \wedge \mathbf{true} \wedge {}^{+}\Gamma(x, X)\big) \vee$$
$$\big(\mathbf{true} \wedge \mathbf{true} \wedge {}^{-}\Gamma(x, X)\big).$$

The occurrences of "**true**" can be dealt with as in Case 1 to obtain subformulae of the required form.

Padding: Before proceeding with the induction step, we explain a technique, to which we shall henceforth refer to as *padding*. Let $\varrho(x, X_1, \ldots, X_r)$ be a subformula of the QuineCALC formula $\varphi(x)$ satisfying the Lemma, and let Y be a set name variable not occurring in ϱ which is quantified at a higher level in $\varphi(x)$. We now show how $\varrho(x, X_1, \ldots, X_r)$ can be rewritten to a subformula $\varrho'(x, X_1, \ldots, X_r, Y)$, also satifsying the Lemma, without changing the semantics of $\varphi(x)$.

Since $\varrho(x, X_1, \ldots, X_r)$ satisfies the Lemma, it can be rewritten as

$$\bigvee_{1 \leq i \leq m} \varrho_i(x, X_1, \ldots, X_r)$$

with, for $i = 1, \ldots, m$,

$$\varrho_i = \psi_i \wedge c_i \wedge {}^{s_{i1}}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_{ir}}\Gamma(x, X_r),$$

as explained in the statement of the Lemma. Now, let α be

$$\bigvee_{1 \leq j \leq r} (X_i = Y) \vee (X_i \neq Y),$$

which always evaluates to **true**. We take $\varrho'(x, X_1, \ldots, X_r, Y)$ to be the subformula

$$\bigvee_{1 \leq i \leq m} \varrho'_i(x, X_1, \ldots, X_r, Y)$$

where, for $i = 1, \ldots, m$, ϱ'_i equals

$$\psi_i \wedge \alpha \wedge c_i \wedge {}^{s_{i1}}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_{ir}}\Gamma(x, X_r) \wedge ({}^{+}\Gamma(x, Y) \vee {}^{-}\Gamma(x, Y)).$$

Clearly, for every valid assignment to the variables $x, X_1, \ldots, X_r$, the terms $\varrho_i(x, X_1, \ldots, X_r)$ and $\varrho'_i(x, X_1, \ldots, X_r, Y)$ evaluate to the same truth value, irrespective of the set name assigned to Y. By applying distributivity, the above expression can be cast in the desired form.

We are now ready to proceed with the induction step of our structural induction proof, and consider all the constructs that may occur in a QuineCALC formula.

Disjunction: Without loss of generality, we may assume that the subformula to be rewritten is of the form

$$\varrho^1(x, X_1, \ldots, X_p, Y_{p+1}, \ldots, Y_{r_1}) \vee \varrho^2(x, X_1, \ldots, X_p, Z_{p+1}, \ldots, Z_{r_2})$$

with $X_1, \ldots, X_p$ precisely the set name variables common to both disjuncts. By the induction hypothesis, both disjuncts satisfy the Lemma. Using padding repeatedly, we can rewrite the first disjunct to

$$\varrho^{1'}(x, X_1, \ldots, X_p, Y_{p+1}, \ldots, Y_{r_1}, Z_{p+1}, \ldots, Z_{r_2})$$

and the second disjunct to

$$\varrho^{2'}(x, X_1, \ldots, X_p, Y_{p+1}, \ldots, Y_{r_1}, Z_{p+1}, \ldots, Z_{r_2}),$$

both also satisfying the Lemma. Since the rewritten disjuncts now run over the same set of variables, their disjunction obviously also satisfies the Lemma.

Negation: Consider a subformula of the form $\neg\varrho(x, X_1, \ldots, X_r)$ for which $\varrho(x, X_1, \ldots, X_r)$ satisfies the Lemma, i.e., can be rewritten as in Expression (1). Hence, $\neg\varrho(x, X_1, \ldots, X_r)$ is equivalent to a conjunction of the form

$$\bigwedge_{1 \leq i \leq m} \neg\varrho_i(x, X_1, \ldots, X_r),$$

where, for $i = 1, \ldots, m$, ϱ_i is of the form

$$\psi_i \wedge c_i \wedge {}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r),$$

satisfying the conditions of the Lemma.

We first show that the induction step in this case follows provided we can prove that

> $\neg\varrho_i(x, X_1, \ldots, X_r)$ can be rewritten as a disjunction of the form
>
> $$\bigvee_{1 \leq l \leq t_i} \varrho_{il_i}(x, X_1, \ldots, X_r), \tag{2}$$
>
> where, for $i = 1, \ldots, m$ and $l_i = 1, \ldots, t_i$, ϱ_{il_i} is of the form
>
> $$\psi_{il_i} \wedge c_{il_i} \wedge {}^{s_{il_1}1}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{il_ir}}\Gamma(x, X_r),$$
>
> as in the Lemma.

Indeed, (2) implies that $\neg\varrho_i(x, X_1, \ldots, X_r)$ is equivalent to a conjunction of disjunctions of the form

$$\bigwedge_{1 \leq i \leq m} \Big(\big(\bigvee_{1 \leq l_i \leq t_i} \varrho_{il_i}(x, X_1, \ldots, X_r) \big) \Big), \tag{3}$$

which, by distributivity, is equivalent to the disjunction of conjunctions

$$\bigvee_{(l_1, \ldots, l_m) \in CP} \big(\varrho_{1l_1}(x, X_1, \ldots, X_r) \wedge \ldots \wedge \varrho_{ml_m}(x, X_1, \ldots, X_r) \big),$$

where CP is the Cartesian product $\{1, \ldots, t_1\} \times \cdots \times \{1, \ldots, t_m\}$. We now focus on each of the disjuncts $\varrho_{1l_1}(x, X_1, \ldots, X_r) \wedge \ldots \wedge \varrho_{ml_m}(x, X_1, \ldots, X_r)$ separately. Such a disjunct is unsatisfiable—and may then be omitted from the disjunction—unless

1. $\psi_{1l_1}, \ldots, \psi_{ml_m}$ are mutually equivalent;
2. $c_{1l_1}, \ldots, c_{ml_m}$ are mutually equivalent; and
3. for all $v = 1, \ldots, r$, $s_{1l_1v} = \cdots = s_{ml_mv}$.

in which case the disjunct is equivalent to each of its conjuncts—and hence can be replaced by any of it. We may thus conclude that Disjunction (3) can be rewritten in the form required by the Lemma.

Hence, it only remains to prove Claim (2). Clearly, $\neg\varrho_i(x, X_1, \ldots, X_r)$ is equivalent to

$$\begin{array}{l}
(\psi_i \wedge c_i \wedge \neg({}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r))) \vee \\
(\psi_i \wedge \neg c_i \wedge ({}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r))) \vee \\
(\psi_i \wedge \neg c_i \wedge \neg({}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r))) \vee \\
(\neg\psi_i \wedge c_i \wedge ({}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r))) \vee \\
(\neg\psi_i \wedge c_i \wedge \neg({}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r))) \vee \\
(\neg\psi_i \wedge \neg c_i \wedge ({}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r))) \vee \\
(\neg\psi_i \wedge \neg c_i \wedge \neg({}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r))).
\end{array} \tag{4}$$

In the rewriting above, there are only three different negated subexpressions: (i) $\neg({}^{s_{i1}}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_{ir}}\Gamma(x, X_r))$, (ii) $\neg c_i$, and (iii) $\neg\psi_i$. We show that each of these three negated subexpressions can be rewritten as a disjunction of subformulae of the appropriate form. By applying distributivity, it then readily follows that the entire Expression (4) can be rewritten in the appropriate form.

We first rewrite $\neg({}^{s_{i1}}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_{ir}}\Gamma(x, X_r))$, as follows:

$$\begin{array}{l}
\Big({}^{-s_{i1}}\Gamma(x, X_1) \wedge \big({}^{+}\Gamma(x, X_2) \vee {}^{-}\Gamma(x, X_2)\big) \wedge \big({}^{+}\Gamma(x, X_3) \vee {}^{-}\Gamma(x, X_3)\big) \wedge \ldots \\
\qquad \ldots \wedge \big({}^{+}\Gamma(x, X_r) \vee {}^{-}\Gamma(x, X_r)\big)\Big) \vee \\
\Big(\big({}^{+}\Gamma(x, X_1) \vee {}^{-}\Gamma(x, X_1)\big) \wedge {}^{-s_{i2}}\Gamma(x, X_2) \wedge \big({}^{+}\Gamma(x, X_3) \vee {}^{-}\Gamma(x, X_3)\big) \wedge \ldots \\
\qquad \ldots \wedge \big({}^{+}\Gamma(x, X_r) \vee {}^{-}\Gamma(x, X_r)\big)\Big) \vee \\
\qquad\qquad\qquad\qquad \vdots \\
\Big(\big({}^{+}\Gamma(x, X_1) \vee {}^{-}\Gamma(x, X_1)\big) \wedge \big({}^{+}\Gamma(x, X_2) \vee {}^{-}\Gamma(x, X_2)\big) \wedge \ldots \\
\qquad \ldots \wedge \big({}^{+}\Gamma(x, X_{r-1}) \vee {}^{-}\Gamma(x, X_{r-1})\big) \wedge {}^{-s_{ir}}\Gamma(x, X_r)\Big),
\end{array}$$

where $-s_{ij}$, $1 \leq j \leq r$, stands for the sign opposite to s_{ij}.

We next rewrite $\neg c_i$. If c_i is $\mathsf{eq}(x, n_i)$, with $n_i < q$, then $\neg c_i$ is equivalent to

$$\mathsf{eq}(x, 0) \vee \ldots \vee \mathsf{eq}(x, n_i - 1) \vee (\mathsf{gteq}(x, n_i + 1) \wedge \mathsf{cogteq}(x, 0)).$$

Clearly, the last condition can be written as a disjunction of conditions of the types allowed in the statement of this Lemma. A similar reasoning can be made

if c_i is $\mathsf{coeq}(x, n_i)$, with $n_i < q$. Finally, if c_i is $\mathsf{gteq}(x, q) \wedge \mathsf{cogteq}(x, q)$, then $\neg c_i$ is equivalent to

$$\mathsf{eq}(x, 0) \vee \ldots \vee \mathsf{eq}(x, q-1) \vee \mathsf{coeq}(x, q-1) \vee \ldots \vee \mathsf{coeq}(x, 0).$$

We conclude this argument by observing that $\neg\psi_i$ is equivalent to the disjunction of all other expressions of the same form on the same set of variables.

Quantification: Consider a subformula $\exists X_p\, \varrho(x, X_1, \ldots, X_r)$, $1 \leq p \leq r$, for which $\varrho(x, X_1, \ldots, X_r)$ satisfies the Lemma, i.e., can be rewritten as in Expression (1). Since we can distribute the existential quantification over disjunction, we may assume without loss of generality that $\varrho(x, X_1, \ldots, X_r)$ is of the form $\psi \wedge c \wedge {}^{s_1}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_r}\Gamma(x, X_r)$. Since c does not contain set name variables, $\exists X_p\, \varrho(x, X_1, \ldots, X_r)$ can be rewritten as

$$c \wedge \exists X_p\, (\psi \wedge {}^{s_1}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_r}\Gamma(x, X_r)). \tag{5}$$

In order to be able to proceed, we first introduce a few concepts. Let $V = \{X_1, \ldots, X_r\}$. The set V is the disjoint union of V^+ and V^-, were V^+ consists of the set name variables occuring in a positive Γ-conjunct and V^- consists of the set name variables occuring in a negative Γ-conjunct of Expression (5). Furthermore, let G be the complete undirected graph on V, where the edge between variables X_i and X_j, $1 \leq i < j \leq r$, is labeled with either "$=$" or "$\neq$", depending on whether $X_i = X_j$ or $X_i \neq X_j$ is the corresponding conjunct of ψ. Finally, we define G^+ and G^- as the subgraphs of G induced by V^+ and V^-, respectively. Observe that G^+ and G^- are complete, since G is.

If G contains an edge between a variable of V^+ and a variable of V^- labeled "$=$", then Expression (5) is unsatisfiable, and can be omitted from the disjunction of which it is part. Thus, without loss of generality, we assume that all edges in G connecting a variable of V^+ to a variable of V^- are labeled "$\neq$".

Also, if G is not colorable (with r colors), then, again, Expression (5) is unsatisfiable, and can be omitted from the disjunction of which it is part. Thus, without loss of generality, assume that G is colorable (with r colors). Since G is a complete graph, all colorings of G are actually isomorphic. The colorings of G induce colorings of G^+ (respectively G^-), and, by the same argument, these are also isomorphic. So, let s^+ and s^- be the exact numbers of colors needed to color G^+ and G^-, respectively (and, hence, $s^+ + s^-$ is the exact number of colors needed to color G). Since $s^+ + s^- \leq r \leq q$, it follows that both $s^+ \leq q$ and $s^- \leq q$.

Now, let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ be a structure with $n = |\sigma| \geq 2q - 1$. We evaluate $\varphi(x)$ over this structure. For convenience we shall abbreviate the subformula in Expression (5) to the right of the existential quantifier as $\varrho'(x, X_1, \ldots, X_r)$. (Hence, Expression (5) can be written as $c \wedge \exists X_p\, \varrho'(x, X_1, \ldots, X_r)$.) Let $o \in \mathcal{D}$.

We first claim that, if there exist set names $S_1, \ldots, S_r$ in σ (not necessarily all different) such that[10] $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varrho'(o, S_1, \ldots, S_r)$, then $s^+ \leq \mathsf{inc}(o, \gamma) \leq$

[10] Slightly extending a previously introduced notation in the straightforward way.

$n - s^-$.[11] To see this, notice that it follows from $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varrho'(o, S_1, \ldots, S_r)$ that G can be colored by assigning the "color" S_k to X_k, $1 \leq k \leq r$. The number of different set names assigned to variables in V^+ is precisely s^+ and the number of different set names assigned to variables in V^- is precisely s^-. Hence, $s^+ \leq \mathsf{inc}(o, \gamma) \leq n - s^-$.

Now, let $\varrho''(x, X_1, \ldots, X_{p-1}, X_{p+1}, \ldots, X_r)$ be the formula obtained from $\varrho'(x, X_1, \ldots, X_r)$ by omitting all conjuncts containing X_p. We claim that, for all objects o in $\mathcal{D}$ and for all set names $S_1, \ldots, S_{p-1}, S_{p+1}, \ldots, S_r$ in σ (not necessarily all different), $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models (\exists X_p\, \varrho')(o, S_1, \ldots, S_{p-1}, S_{p+1}, \ldots, S_r)$ if and only if $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varrho''(o, S_1, \ldots, S_{p-1}, S_{p+1}, \ldots, S_r) \wedge s^+ \leq \mathsf{inc}(o, \gamma) \leq n - s^-$. We start with the "only if." If $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models (\exists X_p\, \varrho')(o, S_1, \ldots, S_{p-1}, S_{p+1}, \ldots, S_r)$, then, by definition, there exists a set name S_p in σ such that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varrho'(o, S_1, \ldots, S_r)$. As shown above, it follows that $s^+ \leq \mathsf{inc}(o, \gamma) \leq n - s^-$. By construction, it also follows that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varrho''(o, S_1, \ldots, S_{p-1}, S_{p+1}, \ldots, S_r)$. We now turn to the "if." Thereto, we need to distinguish two cases: $X_p \in V^+$ and $X_p \in V^-$. As both cases are completely symmetric, we assume without loss of generality that $X_p \in V^+$. Hence, in Expression (5), s_p equals "+." Let G' be the subgraph of G generated by $V - \{X_p\}$, and let G'^+ and G'^- be the subgraphs of G' generated by $V^+ - \{X_p\}$ and $V^- - \{X_p\}$, respectively. Notice that G'^+ is also the subgraph of G^+ generated by $V^+ - \{X_p\}$, and that G'^- equals G^-. Since $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varrho''(o, S_1, \ldots, S_{p-1}, S_{p+1}, \ldots, S_r)$, we can color G' by assigning S_k to X_k, $1 \leq l \leq r$, $k \neq p$. We must again distinguish two cases:

1. *There exists $X_i \in V^+$, $i \neq p$, such that the edge between X_i and X_p in G^+ is labeled "$=$".* Let S_p denote the same set name as S_i. Then, $S_1, \ldots, S_k$ is a coloring of G. (To conclude this, we rely on the colorability of G and the fact that all colorings of a node-generated subgraph of the complete graph G are isomorphic). Moreover, the assumption implies $\gamma(o, S_i)$, and hence also $\gamma(o, S_p)$

2. *For all $X_k \in V^+$, $k \neq j$, the edge between X_k and X_j in G^+ is labeled "$\neq$".* Since G^+ requires s^+ colors to color, G'^+ requires only $s^+ - 1$ colors to color. Hence, there are only $s^+ - 1$ different set names among $S, \ldots, S_{p-1}, S_{p+1}, \ldots, S_r$ associated to variables in V^+. Since $\mathsf{inc}(o, \gamma) \geq s^+$, however, there exists a set name different from all those used to color G'^+, say S_p, such that $\gamma(o, S_p)$. If we associate S_p to X_p, we obtain a coloring for G^+, and hence also one for G.

From the assumption, and the fact that the assignment of S_k to X_k, $1 \leq k \leq r$, is a coloring of G satisfying $\gamma(o, S_p)$, it readily follows that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varrho'(o, S_1, \ldots, S_r)$, and, hence, also that

$$(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models (\exists X_p\, \varrho')(o, S_1, \ldots, S_{p-1}, S_{p+1}, \ldots, S_r).$$

[11] The two statements are actually equivalent, but the reverse implication is not relevant to this proof.

We may thus replace $\exists X_p\, \varphi(x, X_1, \ldots, X_r)$ by

$$c \wedge \mathsf{gteq}(x, s^+) \wedge \mathsf{cogteq}(x, s^-) \wedge \varrho''(X_1, \ldots, X_{p-1}, X_{p+1}, \ldots, X_r).$$

Since both $s^+ \leq q$ and $s^- \leq q$, it follows that $c \wedge \mathsf{gteq}(x, s^+) \wedge \mathsf{cogteq}(x, s^-)$ is either unsatisfiable, in which case the subformula can be omitted from the disjunction of which it is part, or equivalent to c. It now suffices to observe that the subformula $c \wedge \varrho''(X_1, \ldots, X_{p-1}, X_{p+1}, \ldots, X_r)$ is of the required form. □

Of course, every QuineCALC formula $\varphi(x)$ can be considered as a subformula of itself. If we apply Lemma 14 to $\varphi(x)$ as subformula of itself, we observe that each ϱ_i term can be simplified to c_i since there are no free uppercase variables and, hence, the terms ψ_i and ${}^{s_{i1}}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_{ir}}\Gamma(x, X_r)$ are both equivalent to **true** and can be omitted. This yields precisely Theorem 13.

What Lemma 14 and Theorem 13 add to what we already know from Quine's results is that we can not only express a QuineCALC query in terms of incidence information for structures with a given size n of σ, but also that we can do this uniformly so from a certain minimal value of n onward, defined as one less than twice the quantifier depth. The following example shows that, in general, this bound is tight.

Example 15. Consider the QuineCALC query

$$\{x \mid \neg(\exists X \exists Y \exists Z\, (X \neq Y) \wedge (Y \neq Z) \wedge (Z \neq X) \wedge \Gamma(x, X) \wedge \Gamma(x, Y) \wedge \Gamma(x, Z)) \wedge \neg(\exists X \exists Y \exists Z\, (X \neq Y) \wedge (Y \neq Z) \wedge (Z \neq X) \wedge \neg\Gamma(x, X) \wedge \neg\Gamma(x, Y) \wedge \neg\Gamma(x, Z))\}.$$

In words, this query returns an object if and only if both the number of sets in which this object occurs and the number of sets in which this object does not occur is at most 2. The quantifier depth q of the above formula is 3, and hence $2q - 1 = 5$. Theorem 13 therefore pertains to all values of n greater than or equal to 5.

Indeed, if we only consider structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ with $n = |\sigma| \geq 5$, the output of the above query is obviously empty (i.e, the query formula is equivalent to an empty disjunction, which we interpret as **false**). For $n = 4$, however, the query is equivalent to $\mathsf{eq}(x, 2)$; for $n = 3$, the query is equivalent to $\mathsf{eq}(x, 1) \vee \mathsf{eq}(x, 2)$; for $n \leq 2$, the query is equivalent to $\mathsf{eq}(x, 0) \vee \ldots \vee \mathsf{eq}(x, n)$, which evaluates to **true**.

The underlying reason for the lowerbound $2q - 1$ for n lies in the fact that $\mathsf{eq}(x, n_i) \wedge \mathsf{coeq}(x, n_j)$, with $0 \leq n_i, n_j \leq q - 1$, is only guaranteed to evaluate to **false** if $n \geq 2q - 1$. For smaller values of n, it may be that $n_i = n - n_j$, for example, if $q = 3$, $n = 4$, and $n_i = n_j = 2$ (cf. Example 15 above).

In the proof of Lemma 14, we had to rely $\mathsf{eq}(x, n_i) \wedge \mathsf{coeq}(x, n_j)$ evaluating to **false**, for example in the induction step for negation where we had to consider conjunctions of subformulae $\varrho_i(x, X_1, \ldots, X_r)$ of the form

$$\psi_i \wedge c_i \wedge {}^{s_{i1}}\Gamma(x, X_1) \wedge \ldots \wedge {}^{s_{ir}}\Gamma(x, X_r).$$

Depending on the precise values of n_i and n_j in the conjunctions $\mathsf{eq}(x, n_i) \wedge \mathsf{coeq}(x, n_j)$ that must be considered, however, it may sometimes be possible to decrease the lowerbound of $2q-1$. In the extreme case where no such conjuction occurs, there is actually no lowerbound. This is, e.g., the case for the query $\{x \mid \neg\exists X\, \neg\Gamma(x, X)\}$, expressing Query 1 in the Introduction, which in general returns the objects that are in all sets under consideration. Obviously, this query is equivalent to $\{x \mid \mathsf{coeq}(x, 0)\}$.

Since there are only a finite number of values of n to which Theorem 13 does not apply, we can deal with these values separately using Quine's results, yielding the following Corollary.

Corollary 16. *Let* $\mathbf{q} := \{x \mid \varphi(x)\}$ *be a QuineCALC query for which* $\varphi(x)$ *has quantifier depth* $q \geq 0$. *Then,* $\mathbf{q}$ *is equivalent to the QuineCALC query* $\mathbf{q}' := \{x \mid \varphi'(x)\}$, *where* $\varphi'(x)$ *has the form*

$$\Big(\bigvee_{n=0}^{2q-2} (\mathsf{Eq}(n) \wedge \psi_n(x))\Big) \vee (\mathsf{Gteq}(2q-1) \wedge \psi(x)),$$

where

- $\mathsf{Gteq}(r)$ *stands for* $\exists X_1 \ldots \exists X_r \bigwedge_{1 \leq i < j \leq r} X_i \neq X_j$;
- $\mathsf{Eq}(r)$ *stands for* $\mathsf{Gteq}(r) \wedge \neg\mathsf{Gteq}(r+1)$;
- $\psi_n(x)$ *is a disjunction of subformulae of the form* $\mathsf{eq}(x, i)$ *(*$0 \leq i \leq n$*); and*
- $\psi(x)$ *is a disjunction of subformulae of the form* $\mathsf{eq}(x, i)$ *(*$0 \leq i < q$*), subformulae of the form* $\mathsf{coeq}(x, j)$ *(*$0 \leq j < q$*), and at most one subformula of the form* $\mathsf{gteq}(x, q) \wedge \mathsf{cogteq}(x, q)$.

Example 17. Consider again the QuineCALC query of Example 15. By Corollary 16, and after applying some straightforward simplifications, this query is equivalent to

$$\mathsf{Eq}(0) \vee \mathsf{Eq}(1) \vee \mathsf{Eq}(2) \vee (\mathsf{Eq}(3) \wedge (\mathsf{eq}(x, 1) \vee \mathsf{eq}(x, 2))) \vee (\mathsf{Eq}(4) \wedge \mathsf{eq}(x, 2)).$$

We can see the incidence-defined form for a QuineCALC query provided by Corollary 16 as a *normal form* for QuineCALC queries. Notice, however, that this normalization leads to quantifier depth that is alsmost double of the original one, as the quantifier depth of $\mathsf{Eq}(2q-2)$ equals $2q-1$. This, however, is the price one has to pay for normalization. The situation can be compared with putting a first-order formula in prenex normal form. The standard algorithm to achieve this increase the quantifier rank to the number of quantifiers in the original formula. Moreover it can easily be shown that it is in general impossible to replace a first-order formula by an equivalent prenex normal form formula with the same quantifier rank.

As the characterization result of Corollary 16 lifts the characterization result of Quine for symmetric Boolean functions using incidence information to QuineCALC queries, we have answered Research Question 2 in the affirmative.

5. SyCALC

As announced in the opening paragraph of Section 4, we will now extend QuineCALC to SyCALC, the language which is at the core of this study.

5.1. Language definition

QuineCALC is a generalization of symmetric n-ary Boolean functions whose arguments and values are sets, and that are specifiable exclusively by means of union, intersection, and complement. We now add projection and Cartesian product to this list of operators. In our logic framework, this corresponds to extending QuineCALC by allowing multiple lowercase variables in formulas over which quantification is allowed. More precisely, SyCALC *formulae* are defined by the following syntax rule:

$$\varphi := \Gamma(x, X) \mid X = Y \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi_1 \mid \exists x\ \varphi_1 \mid \exists X\ \varphi_1.$$

We also allow the usual abbrevations, such as $X \neq Y$ and $\varphi_1 \wedge \varphi_2$. A SyCALC *query* has the form $\{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$, where $\varphi(x_1, \ldots, x_m)$ is a SyCALC formula without free occurrences of uppercase variable and where $x_1, \ldots, x_m$ are the only free lowercase variables. A SyCALC formula is called *closed* if no variable occurs free in it. A SyCALC query defined by a closed SyCALC formula represents a query with Boolean output or a "yes-no query," where "$\{\langle\rangle\}$" is interpreted as **true** and "$\emptyset$" is interpreted as **false**. We usually refer to such SyCALC queries as Boolean queries.

The semantics of SyCALC is analogous to the semantics of QuineCALC. As a consequence, also SyCALC queries are symmetric. For a sequence of objects $o_1, \ldots, o_m \in \mathcal{D}$, we denote by $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_1, \ldots, o_m)$ that $\varphi(x_1, \ldots, x_m)$ evaluates to **true** in the structure under consideration if x_i is substituted by o_i, $1 \leq i \leq m$.[12] For $n \geq 0$, we say that two SyCALC queries $\{\langle x_1, \ldots, x_m\rangle \mid \varphi_1(x_1, \ldots, x_m)\}$ and $\{\langle x_1, \ldots, x_m\rangle \mid \varphi_2(x_1, \ldots, x_m)\}$ are *n-equivalent* if, for all structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ with $|\sigma| = n$, and for all sequences of objects $o_1, \ldots, o_m$, $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi_1(o_1, \ldots, o_m)$ if and only if $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi_2(o_1, \ldots, o_m)$. Two SyCALC queries are *equivalent* if they are n-equivalent for all $n \geq 0$.

Example 18. The SyCALC queries

(3) $\{\langle\rangle \mid \neg\exists x \exists X \exists Y\ (\Gamma(x, X) \wedge \Gamma(x, Y) \wedge (X \neq Y))\}$;

(4) $\{\langle x\rangle \mid \exists X\ (\Gamma(x, X) \wedge \neg\exists Y\ (\Gamma(x, Y) \wedge (X \neq Y))) \wedge \exists y \exists X \exists Y \exists Z\ (\Gamma(y, X) \wedge \Gamma(y, Y) \wedge \Gamma(y, Z) \wedge (X \neq Y) \wedge (Y \neq Z) \wedge (Z \neq X))\}$;

(5) $\{\langle\rangle \mid \neg\exists x \exists X \exists Y\ (\Gamma(x, X) \wedge \neg\Gamma(x, Y))\}$;

(6) $\{\langle x, y\rangle \mid \exists X \exists Y\ (\Gamma(x, X) \wedge \Gamma(y, X) \wedge \Gamma(x, Y) \wedge \Gamma(y, Y) \wedge (X \neq Y))\}$;

(7) $\{\langle x, y\rangle \mid (\exists X\ \Gamma(x, X)) \wedge (\exists X\ \Gamma(y, X)) \wedge$

[12] Recall that lowercase (object) variables range over $\mathcal{D}$, whereas uppercase (set name) variables range over σ.

$$(\neg\exists X\ (\Gamma(x,X) \wedge \neg\Gamma(y,X))) \wedge (\neg\exists X\ (\neg\Gamma(x,X) \wedge \Gamma(y,X)))\}$$

respectively express Queries 3–7 in the Introduction.

Example 19. Let $r \geq 0$. The expressions $\mathsf{Gteq}(r)$ and $\mathsf{Eq}(r)$ described in the statement of Corollary 16 are closed SyCALC formulae. The corresponding queries $\{\langle\rangle \mid \mathsf{Gteq}(r)\}$ and $\{\langle\rangle \mid \mathsf{Eq}(r)\}$ are Boolean SyCALC queries that, upon input a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, return whether $n = |\sigma| \geq r$, respectively whether $n = |\sigma| = r$.

Unsurprisingly, the language SyCALC is more expressive than the language QuineCALC, even if we restrict ourselves to SyCALC queries returning unary output. We give an example of such a SyCALC query that is not expressible in QuineCALC.

Example 20. Consider the SyCALC query in Example 18 equivalent to Query 4 in the Introduction. Let $o_1, o_2 \in \mathcal{D}$, $S_1, S_2, S_3 \in \mathcal{S}$, and $\sigma = \{S_1, S_2, S_3\}$, and let $\gamma_1 = \{\langle o_1, S_1\rangle, \langle o_2, S_1\rangle, \langle o_2, S_2\rangle, \langle o_2, S_3\rangle\}$, and $\gamma_2 = \{\langle o_1, S_1\rangle\}$. Although $\mathsf{inc}(o_1, \gamma_1) = \mathsf{inc}(o_1, \gamma_2) = 1$, o_1 is returned upon input the structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1)$, but not upon input the structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2)$, in violation of Corollary 16. Hence, this query is *not* equivalent to a QuineCALC query.

5.2. SyCALC and symmetric relational functions

In order to solve Research Question 3, we extend Theorems 7 and 10 from QuineCALC to SyCALC.

First, we extend Quine's notion of "(symmetric) Boolean function" to accommodate the presence of projection and Cartesian product. Thereto, we must allow the output to be relations of any arity over the objects in $\mathcal{D}$. To emphasize the distinction, we shall refer to such functions as *(symmetric) relational functions*.

Definition 21. Let $n, m \geq 0$. A (symmetric) function operating on sequences of n sets of objects $S_1, \ldots, S_n$ is called *relational* if the output is an m-ary relation on these objects, and this relation can be described as a combination of $S_1, \ldots, S_n$ using intersection, union, complement, projection, and Cartesian product.[13]

We also extend the notion of equivalence of a QuineCALC query and a symmetric function returning sets of objects to the equivalence of a general SyCALC query and a symmetric function returning a relation on these objects.

Definition 22. Let $n, m \geq 0$, and let f be a symmetric function operating on sequences of n sets of objects and returning m-ary relations on these objects, and let $\mathbf{q} := \{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ be a SyCALC query. We say that $\mathbf{q}$ is *n-equivalent* to f, denoted $\mathbf{q} \equiv_n f$, if, for all sequences of n sets $S_1, \ldots, S_n$ and for all sequences of m objects $o_1, \ldots, o_m$, we have that $\langle o_1, \ldots, o_m\rangle \in f(S_1, \ldots, S_n)$ if and only if $\mathsf{enc}(S_1, \ldots, S_n) \models \varphi(o_1, \ldots, o_m)$.

[13] Note that union and intersection are only applied to operands with the same arity.

We can now generalize Theorem 7.

Theorem 23. *For every SyCALC query* $\mathbf{q}$*, and for every natural number* $n \geq 0$*, there exists a symmetric relational function* $f_{\mathbf{q},n}(S_1, \ldots, S_n)$ *such that* $\mathbf{q} \equiv_n f_{\mathbf{q},n}$*.*

Proof. Let $\mathbf{q} := \{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ be a SyCALC query and $n \geq 0$. The proof goes along the same lines as the proof of Theorem 7. In the context of SyCALC, the function $\mathsf{qe}(\cdot)$ to eliminate quantification over uppercase variable must be extended by adding the rule

$$\mathsf{qe}(\exists x\ \varphi_1) = \exists x\ \mathsf{qe}(\varphi_1).$$

to take into account quantification over lowercase variables.

Defining the function $\mathsf{fun}(\cdot)$ that translates $\mathsf{qe}(\varphi)$ into a symmetric relational function requires some more care. From the proof of Theorem 7, we retain the rules

$$\begin{aligned}\mathsf{fun}(\mathbf{true}) &= \mathcal{D};\\ \mathsf{fun}(\mathbf{false}) &= \emptyset;\\ \mathsf{fun}(\Gamma(x, S_i)) &= S_i.\end{aligned}$$

In the other rules below, $\mathsf{fun}(\varphi_1(x_1, \ldots, x_r))$ always defines a subset of $\mathcal{D}^r$:

$$\begin{aligned}\mathsf{fun}(\exists x_{r+1}\ \varphi_1(x_1, \ldots, x_r, x_{r+1})) &= \pi_{1,\ldots,r}(\mathsf{fun}(\varphi_1(x_1, \ldots, x_r, x_{r+1})));\\ \mathsf{fun}(\varphi_1(x_{\tau(1)}, \ldots, x_{\tau(r)})) &= \pi_{\tau(1),\ldots,\tau(r)}(\mathsf{fun}(\varphi_1(x_1, \ldots, x_r)));\\ \mathsf{fun}(\varphi_1(x_1, \ldots, x_{r_1}) \vee \varphi_2(x_{r_2+1}, \ldots, x_r)) &=\\ (\mathsf{fun}(\varphi_1(x_1, \ldots, x_{r_1})) \times \mathcal{D}^{r-r_1}) &\cup (\mathcal{D}^{r_2} \times \mathsf{fun}(\varphi_2(x_{r_2+1}, \ldots, x_r)));\\ \mathsf{fun}(\neg\varphi_1(x_1, \ldots, x_r)) &= \mathcal{D}^r - \mathsf{fun}(\varphi_1(x_1, \ldots, x_r)).\end{aligned}$$

In the second rule, τ is a permutation of $\{1, \ldots, r\}$. We use this rule to reorder the variables whenever needed to apply the rules before. Notice that, in the last rule, $\mathcal{D}^r - \mathsf{fun}(\varphi_1(x_1, \ldots, x_r))$ is the complement of $\mathsf{fun}(\varphi_1(x_1, \ldots, x_r))$, which we shall often denote more compactly as $\overline{\mathsf{fun}(\varphi_1(x_1, \ldots, x_r))}$. It is now straightforward that the expression $\mathsf{fun}(\mathsf{qe}(\varphi(x_1, \ldots, x_m)))$ defines a symmetric relational function $f_{\mathbf{q},n}$ on sequences of n sets that returns m-ary relations for which $\mathbf{q} \equiv_n f_{\mathbf{q},n}$. □

Example 24. Consider the SyCALC queries in Example 18, expressing Queries 3–7. Choose $n = 3$. Then the symmetric relational functions on sequences of three sets S_1, S_2, S_3 that are 3-equivalent to these SyCALC queries are, after some

straightforward simplications,

(3) $\overline{\pi_{\langle\rangle}((S_1 \cap S_2) \cup (S_2 \cap S_3) \cup (S_3 \cap S_1))}$;

(4) $\big((S_1 \cap \overline{S_2 \cup S_3}) \cup (S_2 \cap \overline{S_3 \cup S_1}) \cup (S_3 \cap \overline{S_1 \cup S_2})\big) \times \pi_{\langle\rangle}(S_1 \cap S_2 \cap S_3)$;

(5) $\overline{\pi_{\langle\rangle}\big((S_1 \cap \overline{S_2}) \cup (S_2 \cap \overline{S_3}) \cup (S_3 \cap \overline{S_1})\big)}$;

(6) $\big((S_1 \times S_1) \cap (S_2 \times S_2)\big) \cup \big((S_2 \times S_2) \cap (S_3 \times S_3)\big) \cup \big((S_3 \times S_3) \cap (S_1 \times S_1)\big)$;

(7) $\big((S_1 \cup S_2 \cup S_3) \times (S_1 \cup S_2 \cup S_3)\big) \cap$
$\overline{(S_1 \times \overline{S_1}) \cup (S_2 \times \overline{S_2}) \cup (S_3 \times \overline{S_3})} \cap \overline{(\overline{S_1} \times S_1) \cup (\overline{S_2} \times S_2) \cup (\overline{S_3} \times S_3)}$,

respectively.

We now turn to the generalization of Theorem 10 to SyCALC queries.

Theorem 25. *For all natural numbers $n, m \geq 0$ and for every symmetric relational function $f_n(S_1, \ldots, S_n)$ on sequences of n sets that return m-ary relations over $\mathcal{D}$, there exists a SyCALC query $\mathbf{q}_{f_n} := \{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ such that $\mathbf{q}_{f_n} \equiv_n f_n$.*

Proof. By assumption, the symmetric relational function f_n in the statement of Theorem 25 can be described by some expression $E(S_1, \ldots, S_n)$ that only uses $S_1, \ldots, S_n$, intersection, union, complement, projection, and Cartesian product. Hence, E can be translated to a relational calculus expression $\{(x_1, \ldots, x_m) \mid C(x_1, \ldots, x_m)\}$. Now let $C'(x_1, \ldots, x_m, X_1, \ldots, X_n)$, be $C(x_1, \ldots, x_m)$ in which each atomic subexpression of the form "$x_i \in S_j$" is substituted by "$\Gamma(x_i, X_j)$." Finally, define $\varphi(x_1, \ldots, x_m)$ as

$$\exists X_1 \cdots \exists X_n \ \big(C'(x_1, \ldots, x_m, X_1, \ldots, X_n) \wedge \bigwedge_{1 \leq i < j \leq n} X_i \neq X_j\big).$$

Then, the expression $\mathsf{qe}(\varphi)$ computed in the proof of Theorem 23 yields

$$\bigwedge_{\tau \in \mathrm{Perm}\{1,\ldots,n\}} C'(x_1, \ldots, x_m, S_{\tau(1)}, \ldots, S_{\tau(n)}).$$

In the computation of $\mathsf{fun}(\mathsf{qe}(\varphi))$ in the proof of Theorem 23, $\Gamma(x_i, S_j)$ is translated into S_j. Hence, we may conclude that the expression $\mathsf{fun}(\mathsf{qe}(\varphi))$ is the standard translation of

$$\{(x_1, \ldots, x_m) \mid \bigwedge_{\tau \in \mathrm{Perm}\{1,\ldots,n\}} C(x_{\tau(1)}, \ldots, x_{\tau(n)})\}.$$

into the relational algebra (with complement instead of difference), which, by construction, is equivalent to the expression $\bigcup_{\tau \in \mathrm{Perm}\{1,\ldots,n\}} E(S_{\tau(1)}, \ldots, S_{\tau(n)})$, describing the relational function $\bigcup_{\tau \in \mathrm{Perm}\{1,\ldots,n\}} f_n(S_{\tau(1)}, \ldots, S_{\tau(n)})$. Since f_n is a symmetric relational function, all terms in this union are equal, and hence equal to $f_n(S_1, \ldots, S_n)$. Theorem 25 now follows readily. □

Example 26. We revisit Example 24. As a first example, consider the symmetric relational function $f_3(S_1, S_2, S_3) = \overline{\pi_{\langle\rangle}((S_1 \cap S_2) \cup (S_1 \cap S_3) \cup (S_2 \cap S_3))}$. We apply the construction in the proof of Theorem 25 to this relational function. First, we rewrite the given function to a relational calculus expression $\{\langle\rangle \mid C()\}$ with $C() =$

$$\neg\exists x\ ((x \in S_1 \wedge x \in S_2) \vee (x \in S_1 \wedge x \in S_3) \vee (x \in S_2 \wedge x \in S_3)).$$

Next, we construct $C'(X_1, X_2, X_3)$ by replacing $x \in S_j$, $1 \leq j \leq 3$, by $\Gamma(x, X_j)$, resulting in $C'(X_1, X_2, X_3) =$

$$\neg\exists x\ ((\Gamma(x, X_1) \wedge \Gamma(x, X_2)) \vee (\Gamma(x, X_1) \wedge \Gamma(x, X_3)) \vee (\Gamma(x, X_2) \wedge \Gamma(x, X_3))).$$

Finally, we obtain the SyCALC query

$$\{\langle\rangle \mid \exists X_1 \exists X_2 \exists X_3\ ((X_1 \neq X_2) \wedge (X_1 \neq X_3) \wedge (X_2 \neq X_3) \wedge C'(x, X_1, X_2, X_3))\},$$

which, on structures with $n \geq 3$, can be simplified to the SyCALC query in Example 18 expressing Query 3. So, both queries are 3-equivalent, and hence also 3-equivalent to f_3.

As a second example, consider $g_3(S_1, S_2, S_3) = ((S_1 \times S_1) \cap (S_2 \times S_2)) \cup ((S_2 \times S_2) \cap (S_3 \times S_3)) \cup ((S_3 \times S_3) \cap (S_1 \times S_1))$. If we apply the construction in the proof of Theorem 25 to this relational function, we obtain the SyCALC query

$$\begin{aligned}\{\langle x, y\rangle \mid \exists X \exists Y \exists Z\ \Big(&X \neq Y \wedge Y \neq Z \wedge Z \neq X \wedge \big(\\ &(\Gamma(x, X) \wedge \Gamma(y, X) \wedge \Gamma(x, Y) \wedge \Gamma(y, Y)) \vee\\ &(\Gamma(x, Y) \wedge \Gamma(y, Y) \wedge \Gamma(x, Z) \wedge \Gamma(y, Z)) \vee\\ &(\Gamma(x, Z) \wedge \Gamma(y, Z) \wedge \Gamma(x, X) \wedge \Gamma(y, X))\big)\Big)\},\end{aligned}$$

which, on structures with $n \geq 3$, can be simplified to the SyCALC query in Example 18 expressing Query 6. So, both queries are 3-equivalent, and hence also 3-equivalent to f_3.

Theorems 23 and 25 together settle Research Question 3.

5.3. SyCALC queries that only count

Let us call two structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1)$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2)$ *incidence-equivalent* if, for each object $o \in \mathcal{D}$, $\mathsf{inc}(o, \gamma_1) = \mathsf{inc}(o, \gamma_2)$. By Corollary 16, QuineCALC queries can, alternatively, be expressed in terms of counting-only terms such as $\mathsf{eq}(x, i)$. As such, QuineCALC queries cannot distinguish between incidence-equivalent structures. This is no longer true for SyCALC queries, however.

Example 27. Consider the SyCALC query in Example 18 equivalent to Query 7 in the Introduction. Let $o_1, o_2 \in \mathcal{D}$, $S_1, S_2, S_3 \in \mathcal{S}$, and $\sigma = \{S_1, S_2, S_3\}$, and let

$$\begin{aligned}\gamma_1 &= \{\langle o_1, S_1\rangle, \langle o_1, S_2\rangle, \langle o_2, S_1\rangle, \langle o_2, S_2\rangle\} \text{ and}\\ \gamma_2 &= \{\langle o_1, S_1\rangle, \langle o_1, S_3\rangle, \langle o_2, S_2\rangle, \langle o_2, S_3\rangle\}.\end{aligned}$$

Although $\mathsf{inc}(o_1, \gamma_1) = \mathsf{inc}(o_1, \gamma_2) = 2$ and $\mathsf{inc}(o_2, \gamma_1) = \mathsf{inc}(o_2, \gamma_2) = 2$, (o_1, o_2) is returned upon input the structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1)$, but not upon input the structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2)$.

Therefore, it makes sense to call SyCALC queries that cannot distinguish between incidence-equivalent structures counting-only.

Definition 28. Let $\mathbf{q} := \{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ be a SyCALC query. We say that $\mathbf{q}$ is a *counting-only* query if, for all incidence-equivalent structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1)$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2)$, we have, for all objects $o_1, \ldots, o_m \in \mathcal{D}$, that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1) \models \varphi(o_1, \ldots, o_m)$ if and only if $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2) \models \varphi(o_1, \ldots, o_m)$.

By Corollary 16, all QuineCALC queries are counting-only. There are, however, many counting-only SyCALC queries that are *not* equivalent to a QuineCALC query.

Example 29. Consider the SyCALC queries in Example 18.

The SyCALC query expressing Query 3 in the Introduction returns **true** on structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ precisely if, for all $o \in \mathcal{D}$, $\mathsf{inc}(o, \gamma) \leq 1$. Hence, it is counting-only. As it does not return unary output, it can of course not be equivalent to a QuineCALC query.

Given a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, the SyCALC query expressing Query 4 returns all objects $o \in \mathcal{D}$ with $\mathsf{inc}(o, \gamma) = 1$ provided there exists $o' \in \mathcal{D}$ with $\mathsf{inc}(o', \gamma) \geq 3$. Hence, it is counting-only. Even though it returns unary output, it is not equivalent to a QuineCALC query, as shown in Example 20.

Given a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, the SyCALC query expressing Query 5 returns **true** if, for all objects $o \in \mathcal{D}$, $\mathsf{inc}(o, \gamma) = n$, with $n = |\sigma|$. Hence, it is counting-only. As it does not return unary output, it can of course not be equivalent to a QuineCALC query.

Next consider the SyCALC query expressing Query 6. Let $o_1, o_2, o_3 \in \mathcal{D}$, $S_1, S_2, S_3 \in \mathcal{S}$, and $\sigma = \{S_1, S_2, S_3\}$, and let

$$\gamma_1 = \{\langle o_1, S_1\rangle, \langle o_1, S_2\rangle, \langle o_2, S_1\rangle, \langle o_2, S_2\rangle, \langle o_3, S_3\rangle\} \text{ and}$$
$$\gamma_2 = \{\langle o_1, S_1\rangle, \langle o_1, S_2\rangle, \langle o_2, S_1\rangle, \langle o_2, S_3\rangle, \langle o_3, S_2\rangle\}.$$

While we have that $\mathsf{inc}(o_1, \gamma_1) = \mathsf{inc}(o_1, \gamma_2) = 2$, $\mathsf{inc}(o_2, \gamma_1) = \mathsf{inc}(o_2, \gamma_2) = 2$, and $\mathsf{inc}(o_3, \gamma_1) = \mathsf{inc}(o_3, \gamma_2) = 1$, the query returns (o_1, o_2) upon input $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1)$, but does *not* return (o_1, o_2) upon input $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2)$. Hence the query is *not* counting-only, and, therefore, not equivalent to a QuineCALC query.

Finally, the SyCALC query expressing Query 7 is not counting-only either, as shown in Example 27, and, therefore, also not equivalent to a QuineCALC query.

In Figure 3, we summarize the above classification of queries. Observe that not all symmetric queries are also SyCALC queries, and not all queries are necessarily symmetric. An example of a symmetric query not in SyCALC is $\mathbf{q}_A$, *"return the maximum number of objects in a set encoded by the structure,"* and an example of a non-symmetric query is $\mathbf{q}_B$, *"return the objects in the set encoded by set name S in the structure."*

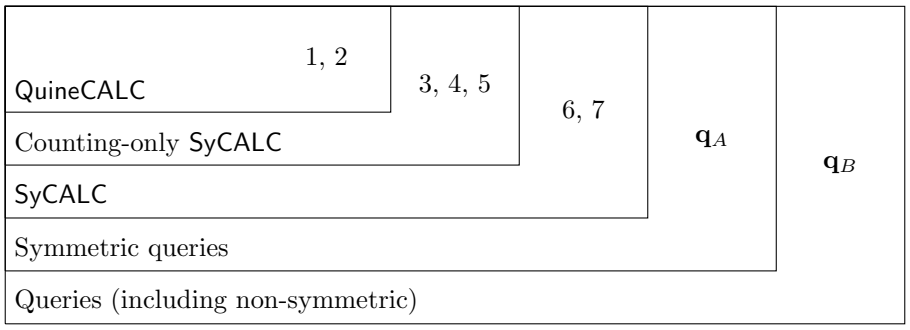


Figure 3: Summary of the hierarchal structure of classes of symmetric queries. For each class, we provide example queries which are not in the immediate subclass. Query $\mathbf{q}_A$ is a symmetric query not in SyCALC, and query $\mathbf{q}_B$ is a non-symmetric query. Both are introduced in Example 29. The other example queries are referred to by the number that was assigned to them in the Introduction.

With Example 29, Research Questions 4, 5, and 6 have been answered in the affirmative.

Definition 28 is in our opinion a very compelling, intuitive semantic definition of counting-only SyCALC queries, but, unfortunately, it does not teach us much about the nature of counting-only SyCALC queries. Therefore, we state a characterization of counting-only SyCALC queries in the same vein as in Corollary 16 for QuineCALC queries.

Theorem 30. *Let* $\mathbf{q} := \{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ *be a counting-only SyCALC query for which* $\varphi(x_1, \ldots, x_m)$ *has quantifier depth* $q_{\mathcal{S}} \geq 0$ *in the uppercase (set name) variables. Then,* $\mathbf{q}$ *is equivalent to a SyCALC query* $\mathbf{q}' := \{\langle x_1, \ldots, x_m\rangle \mid \varphi'(x_1, \ldots, x_m)\}$*, in which* $\varphi'(x_1, \ldots, x_m)$ *has the form*

$$\Big(\bigvee_{n=0}^{2q_{\mathcal{S}}-2} (\mathsf{Eq}(n) \wedge \psi_n(x_1, \ldots, x_m))\Big) \vee (\mathsf{Gteq}(2q_{\mathcal{S}} - 1) \wedge \psi(x_1, \ldots, x_m)),$$

where

- *for* $n = 0, \ldots, 2q_{\mathcal{S}} - 2$, $\psi_n(x_1, \ldots, x_m)$ *is a disjunction of formulae of the form* $\vartheta_0 \wedge \ldots \wedge \vartheta_n \wedge \alpha_1(x_1) \wedge \ldots \wedge \alpha_m(x_m)$*, with*[14]
  - *for* $i = 1, \ldots, n$, ϑ_i *is* $\exists x\ \mathsf{eq}(x, i)$ *or* $\neg\exists x\ \mathsf{eq}(x, i)$*;*
  - *for* $\ell = 1, \ldots, m$, $\alpha_\ell(x_\ell)$ *is of the form* $\mathsf{eq}(x_\ell, k_\ell)$*, with* $0 \leq k_\ell \leq n$*; and*
- $\psi(x_1, \ldots, x_m)$ *is a disjunction of formulae of the form*

$$\vartheta_1 \wedge \ldots \wedge \vartheta_{q_{\mathcal{S}}-1} \wedge \vartheta \wedge \vartheta^{q_{\mathcal{S}}-1} \wedge \ldots \wedge \vartheta^0 \wedge \alpha_1(x_1) \wedge \ldots \wedge \alpha_m(x_m),$$

[14] Observe that, in this construction, ψ_0 can always be simplified to either **true** or **false**.

with

- *for* $i = 1, \ldots, q_{\mathcal{S}} - 1$, ϑ_i *is either* $\exists x\ \mathsf{eq}(x, i)$ *or* $\neg\exists x\ \mathsf{eq}(x, i)$*;*
- ϑ *is* $\exists x\ (\mathsf{gteq}(x, q_{\mathcal{S}}) \wedge \mathsf{cogteq}(x, q_{\mathcal{S}}))$ *or* $\neg\exists x\ (\mathsf{gteq}(x, q_{\mathcal{S}}) \wedge \mathsf{cogteq}(x, q_{\mathcal{S}}))$*;*
- *for* $j = q_{\mathcal{S}} - 1, \ldots, 0$, ϑ^j *is either* $\exists x\ \mathsf{coeq}(x, j)$ *or* $\neg\exists x\ \mathsf{coeq}(x, j)$*;*
- *for* $\ell = 1, \ldots, m$, $\alpha_\ell(x_\ell)$ *is either of the form* $\mathsf{eq}(x_\ell, k_\ell)$*, with* $0 \leq k_\ell < q_{\mathcal{S}}$*; or of the form* $\mathsf{coeq}(x_\ell, k_\ell)$*, with* $0 \leq k_\ell < q_{\mathcal{S}}$*; or of the form* $\mathsf{gteq}(x_\ell, q_{\mathcal{S}}) \wedge \mathsf{cogteq}(x_\ell, q_{\mathcal{S}})$.

The formula $\varphi'(\vec{x})$ in Theorem 30 above is a disjunction of $2q_{\mathcal{S}}$ disjuncts. Each structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ will satisfy exactly one of these disjuncts, depending on whether the size of σ is equal to $0, 1, 2, \ldots, 2q_{\mathcal{S}} - 2$, or is greater than or equal to $2q_{\mathcal{S}} - 1$.

Since φ has quantifier depth $q_{\mathcal{S}}$ in the uppercase variables, φ can be expressed by a formula that contains at most $q_{\mathcal{S}}$ distinct uppercase variables. Given an object o, such formula can test whether the number of objects o belongs to (or does not belong to) is equal to $0, 1, \ldots, q_{\mathcal{S}} - 1$, or is greater than or equal to $q_{\mathcal{S}}$. Intuitively, a formula with only $q_{\mathcal{S}}$ distinct uppercase variables can "count" up to $q_{\mathcal{S}} - 1$, but not beyond. Significantly, if σ contains at least $2q_{\mathcal{S}} - 1$ sets, the incidence and co-incidence of o cannot both be in $\{0, 1, \ldots, q_{\mathcal{S}} - 1\}$; therefore, if φ can count one of these numbers, it cannot count the other. This is the reason why the number $2q_{\mathcal{S}} - 1$ is a threshold in φ'.

As was the case in Corollary 16 for QuineCALC queries, the quantifier depth of φ' in Theorem 30 is $2q - 1$, i.e., alsmost double the quantifier depth of φ.

Theorem 30 will also serve to derive the final result of this Section (Corollary 39), which states that every counting-only SyCALC query is equivalent to a quantified Boolean combination of QuineCALC queries.

As already mentioned, Theorem 30 relies in essence on the limited ability of SyCALC queries to count and distinguish objects. To be able to prove Theorem 30, we first need to show that two structures who are similar with respect to their incidence and co-incidence information as far as counting up to $q_{\mathcal{S}}$ is concerned cannot be distinguished by a counting-only SyCALC query with quantifier depth $q_{\mathcal{S}}$ in the uppercase (set name) variables provided theses structures involve at least $2q_{\mathcal{S}} - 1$ set names.[15]

We achieve this result step-by-step in an series of five lemmas.

1. First, we show (Lemma 31) that two structures in which each object either

   - has the same coincidence in both structures, if this coincidence is at most $q_{\mathcal{S}} - 1$; or else
   - has the same incidence in both structures

   cannot be distinguished by a counting-only SyCALC query with quantifier depth $q_{\mathcal{S}}$ in the uppercase (set name) variables. This lemma is at the basis of comparing structures involving different sets of set names.

[15] See also the explanation of Theorem 30.

2. Next, we show (Lemma 32 and Corollary 33) that two structures over the *same* set of set names in which each object either

   - has the same incidence in both structures, if this incidence is at most $q_{\mathcal{S}} - 1$; or else
   - has the same coincidence in both structures, if this coincidence is at most $q_{\mathcal{S}} - 1$; or else
   - has an incidence and a coincidence of at least $q_{\mathcal{S}}$ in both structures

   cannot be distinguished by a counting-only SyCALC query with quantifier depth $q_{\mathcal{S}}$ in the uppercase (set name) variables.

3. Lemma 34 combines Lemma 31 and Corollary 33 and states essentially the same as Corollary 33, but with the condition removed that the sets of set names in both structures must be the same.

4. In our penultimate lemma (Lemma 35), we remove the condition that the two structures must actively involve the same objects. We show that two structures, which involved the same number of sets names, and in which each natural number is either the incidence of an object in both structures of not the incidence of an object in either structure cannot be distinguished by a counting-only SyCALC query.

5. Finally, Lemma 36 combines Lemmas 34 and 35 and states essentially the same as Lemma 34, but with the condition removed that the objects actively involved in both structures must be the same.

We start with Lemma 31 below, which compares structures involving different numbers of set names.

Lemma 31. *Let* $\{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ *be a counting-only SyCALC query, where* φ *has quantifier depth* $q_{\mathcal{D}}$ *in the lowercase variables and* $q_{\mathcal{S}}$ *in the uppercase variables. Let* $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1)$ *and* $(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2)$ *be structures with* $|\sigma_1| \geq 2q_{\mathcal{S}} - 1$ *and* $|\sigma_2| \geq 2q_{\mathcal{S}} - 1$ *such that, for all* $o \in \mathcal{D}$*,*

- *if* $\mathsf{coinc}(o, \gamma_1) \geq q_{\mathcal{S}}$*, then* $\mathsf{inc}(o, \gamma_2) = \mathsf{inc}(o, \gamma_1)$*; or*
- *else, if* $\mathsf{coinc}(o, \gamma_1) \leq q_{\mathcal{S}} - 1$*, then* $\mathsf{coinc}(o, \gamma_2) = \mathsf{coinc}(o, \gamma_1)$*.*

Then, for all $o_1, \ldots, o_m \in \mathcal{D}$*,* $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1) \models \varphi(o_1, \ldots, o_m)$ *if and only if* $(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2) \models \varphi(o_1, \ldots, o_m)$*.*

Proof. Let $n_1 = |\sigma_1|$ and $n_2 = |\sigma_2|$. Notice that Lemma 31 holds trivially if $n_1 = n_2$. Thus assume $n_1 \neq n_2$. By symmetry, we may assume that $n_2 > n_1$. Now, it suffices to prove Lemma 31 for the special case where $n_2 = n_1 + 1$, as the general case follows from repeatedly applying Lemma 31 for this special case.

To simplify the exposition, we call objects $o \in D$ for which $\mathsf{coinc}(o, \gamma_1) \geq q_{\mathcal{S}}$ *objects of the first category* and objects $o \in D$ for which $\mathsf{coinc}(o, \gamma_1) \leq q_{\mathcal{S}} - 1$ *objects of the second category*. Now, let $\sigma_1' = \{S_1, \ldots, S_{n_1}\} \subset \mathcal{S}$ and $\sigma_2' = \{T_1, \ldots, T_{n_1+1}\} \subset \mathcal{S}$ with $\sigma_1' \cap \sigma_2' = \emptyset$. We construct γ_1' and γ_2', which are initially empty, as follows:

1. for each object o of the first category, add the pairs $\langle o, S_1\rangle, \ldots, \langle o, S_i\rangle$ to γ_1' and add the pairs $\langle o, T_1\rangle, \ldots, \langle o, T_i\rangle$ to γ_3', where $i = \mathsf{inc}(o, \gamma_1)$;

2. for each object o of the second category, add the pairs $\langle o, S_{j+1}\rangle, \ldots, \langle o, S_{n_1}\rangle$ to γ_1' and add the pairs $\langle o, T_{j+1}\rangle, \ldots, \langle o, T_{n_1}\rangle, \langle o, T_{n_1+1}\rangle$ to γ_3', where $j = \mathsf{coinc}(o, \gamma_1)$.

By construction, we have that, for each object o in the first category, $\mathsf{inc}(o, \gamma_1) = \mathsf{inc}(o, \gamma_1') = \mathsf{inc}(o, \gamma_2') = \mathsf{inc}(o, \gamma_2)$, and, for each object o in the second category, $\mathsf{coinc}(o, \gamma_1) = \mathsf{coinc}(o, \gamma_1') = \mathsf{coinc}(o, \gamma_2') = \mathsf{coinc}(o, \gamma_2)$. Since $|\sigma_1| = |\sigma_1'|$, $|\sigma_2| = |\sigma_2'|$, and the SyCALC query under consideration is counting-only, it follows that, for all $o_1, \ldots, o_m \in \mathcal{D}$,

$$(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1) \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma_1', \gamma_1') \models \varphi(o_1, \ldots, o_m);$$
$$(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2) \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma_2', \gamma_2') \models \varphi(o_1, \ldots, o_m).$$

Now, consider the Ehrenfeucht-Fraïssé pebble game of $q_\mathcal{D} + q_\mathcal{S}$ rounds ($q_\mathcal{D}$ of which involve selecting objects in $\mathcal{D}$ and $q_\mathcal{S}$ of which involve selecting set names) on structures $(\mathcal{D}, \mathcal{S}, \sigma_1', \gamma_1')$ and $(\mathcal{D}, \mathcal{S}, \sigma_2', \gamma_2')$. To show that the Dupplicator has a winning strategy for this game, we make the following observations:

1. For all objects $o \in \mathcal{D}$, and for all $k = 1, \ldots, n_1$, $\langle o, S_k\rangle \in \gamma_1'$ if and only if $\langle o, T_k\rangle \in \gamma_2'$. This follows immediately from the construction of γ_1' and γ_2'.

2. For all objects $o \in \mathcal{D}$, and for all $k = n_1 - q_\mathcal{S} + 1, \ldots, n_1$, $\langle o, S_k\rangle \in \gamma_1'$ if and only if o is of the second category. To see the "only if," it suffices to observe that if o is of the first category, then $\mathsf{coinc}(o, \gamma_1) \geq q_\mathcal{S}$, and, hence, by construction, none of $S_{n_1-q_\mathcal{S}+1}, \ldots, S_{n_1}$ can be associated with o in γ_1'. To see the "if," let o be an object of the second category, and let $\mathsf{coinc}(o, \gamma_1) = j$. Then, by construction, for all $k = j+1, \ldots, n_1$, $\langle o, S_k\rangle \in \gamma_1'$. Since $j \leq q_\mathcal{S} - 1$, it follows in particular that, for all $k = q_\mathcal{S}, \ldots, n_1$, $\langle o, S_k\rangle \in \gamma_1'$. It now suffices to observe that $n_1 \geq 2q_\mathcal{S} - 1$ implies that $q_\mathcal{S} \leq n_1 - q_\mathcal{S} + 1$.

3. Similarly, we have that, for all objects $o \in \mathcal{D}$, and for all $k = n_1 - q_\mathcal{S} + 1, \ldots, n_1 + 1$, $\langle o, T_k\rangle \in \gamma_2'$ if and only if o is of the second category.

Properties 2 and 3 above imply that the set names $S_{q_\mathcal{S}+1}, \ldots, S_{n_1}$ in γ_1' and $T_{q_\mathcal{S}+1}, \ldots, T_{n_1}, T_{n_1+1}$ in γ_2' all correspond with the set of all objects of the second category.

We now exhibit a winning strategy for the Duplicator:

- if the Spoiler chooses an object in $\mathcal{D}$ in one structure, the Duplicator chooses the same object in the other structure;

- if the Spoiler chooses S_i, $1 \leq i \leq n_1 - q_\mathcal{S}$, in γ_1', then the Duplicator chooses T_i in γ_2';

- if the Spoiler chooses T_i, $1 \leq i \leq n_1 - q_\mathcal{S}$, in γ_2', then the Duplicator chooses S_i in γ_1';

- if the Spoiler chooses S_i, $n_1 - q_{\mathcal{S}} + 1 \leq i \leq n_1$, in γ_1' not selected before, then the Duplicator chooses one of $T_{n_1-q_{\mathcal{S}}+1}, \ldots, T_{n_1+1}$ in γ_2' not selected before;[16] and
- if the Spoiler chooses T_i, $n_1 - q_{\mathcal{S}} + 1 \leq i \leq n_1 + 1$, in γ_2' not selected before, then the Duplicator chooses one of $S_{n_1-q_{\mathcal{S}}+1}, \ldots, S_{n_1}$ in γ_1' not selected before.[17]

By Properties 1–3, set names selected in the same round are associated with precisely the same objects in γ_1' and γ_2', respectively. Hence, the above is indeed a winning strategy for the Duplicator. Using a straightforward generalization of the classical result for Ehrenfeucht-Fraïssé pebble games to two-sorted logics, it now follows that, for all $o_1, \ldots, o_m \in \mathcal{D}$,

$$(\mathcal{D}, \mathcal{S}, \sigma_1', \gamma_1') \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma_2', \gamma_2') \models \varphi(o_1, \ldots, o_m),$$

which concludes the proof. □

We now show that two structures over the *same* set of set names in which each object either

- has the same incidence in both structures, if this incidence is at most $q_{\mathcal{S}} - 1$; or else
- has the same coincidence in both structures, if this coincidence is at most $q_{\mathcal{S}} - 1$; or else
- has an incidence and a coincidence of at least $q_{\mathcal{S}}$ in both structures

cannot be distinguished by a counting-only SyCALC query with quantifier depth $q_{\mathcal{S}}$ in the uppercase (set name) variables. Lemma 32 below exhibits a special case of this result, a repeated application of which will lead to the actual result (Corollary 33).

Lemma 32. *Let $\{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ be a counting-only SyCALC query, where φ has quantifier depth $q_{\mathcal{D}}$ in the lowercase variables and $q_{\mathcal{S}}$ in the uppercase variables. Let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1)$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2)$ be structures such that, for some $o \in \mathcal{D}$,*

1. $\mathsf{inc}(o, \gamma_1) \geq q_{\mathcal{S}}$ *and* $\mathsf{coinc}(o, \gamma_1) \geq q_{\mathcal{S}}$*; and*
2. $\mathsf{inc}(o, \gamma_2) \geq q_{\mathcal{S}}$ *and* $\mathsf{coinc}(o, \gamma_2) \geq q_{\mathcal{S}}$.

Assume furthermore that, for all $o' \in \mathcal{D} \setminus \{o\}$, $\mathsf{inc}(o', \gamma_1) = \mathsf{inc}(o', \gamma_2)$. Then, for all $o_1, \ldots, o_m \in \mathcal{D}$, $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1) \models \varphi(o_1, \ldots, o_m)$ if and only if $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2) \models \varphi(o_1, \ldots, o_m)$.

[16] Since there are more than $q_{\mathcal{S}}$ set names available to choose from, this is always possible.
[17] Since there are $q_{\mathcal{S}}$ set names available to choose from, this is always possible.

Proof. Let $n = |\sigma|$. The object o in the statement of Lemma 32 can only exist if $n \geq 2q_{\mathcal{S}}$. If $n = 2q_{\mathcal{S}}$, then, necessarily, $\mathsf{inc}(o, \gamma_1) = \mathsf{inc}(o, \gamma_2) = q_{\mathcal{S}}$ and Lemma 32 holds trivially. Therefore, we assume in this proof that $n > 2q_{\mathcal{S}}$.

Also, it suffices to consider the case where, in one of the structures, o occurs in exactly $q_{\mathcal{S}}$ sets, as the general case follows from two applications of Lemma 32 in this special case.

Thus, assume that $\mathsf{inc}(o, \gamma_1) = q_{\mathcal{S}}$. Let $\mathsf{inc}(o, \gamma_2) = k$. Without loss of generality, we may assume that $q_{\mathcal{S}} < k \leq n - q_{\mathcal{S}}$.[18] Let $\sigma = \{S_1, \ldots, S_n\}$, let $T_1, \ldots, T_{q_{\mathcal{S}}}, U_1, \ldots, U_k$ be pairwise different set names not in σ, and let $\sigma' = \{S_1, \ldots, S_n, T_1, \ldots, T_{q_{\mathcal{S}}}, U_1, \ldots, U_k\}$. We construct γ_1' and γ_2' by adding pairs to $\gamma_1 \setminus \{\langle o, S_1\rangle, \ldots, \langle o, S_n\rangle\}$, as follows:

1. for each object $o' \in \mathcal{D} \setminus \{o\}$ with $\mathsf{coinc}(o', \gamma_1) \leq q_{\mathcal{S}} - 1$, add the pairs $\langle o', T_1\rangle, \ldots, \langle o', T_{q_{\mathcal{S}}}\rangle, \langle o', U_1\rangle, \ldots, \langle o', U_k\rangle$ to both γ_1' and γ_2'.
2. add the pairs $\langle o, T_1\rangle, \ldots, \langle o, T_{q_{\mathcal{S}}}\rangle$ to γ_1';
3. add the pairs $\langle o, U_1\rangle, \ldots, \langle o, U_k\rangle$ to γ_2'.

Then, the structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1)$ and $(\mathcal{D}, \mathcal{S}, \sigma', \gamma_1')$, respectively $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2)$ and $(\mathcal{D}, \mathcal{S}, \sigma', \gamma_2')$, satisfy the conditions of Lemma 31[19], and hence

$$(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1) \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma', \gamma_1') \models \varphi(o_1, \ldots, o_m);$$
$$(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2) \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma', \gamma_2') \models \varphi(o_1, \ldots, o_m).$$

Now, consider the Ehrenfeucht-Fraïssé pebble game of $q_{\mathcal{D}} + q_{\mathcal{S}}$ rounds ($q_{\mathcal{D}}$ of which involve selecting objects in $\mathcal{D}$ and $q_{\mathcal{S}}$ of which involve selecting set names) on structures $(\mathcal{D}, \mathcal{S}, \sigma_1', \gamma_1')$ and $(\mathcal{D}, \mathcal{S}, \sigma_2', \gamma_2')$. We exhibit a winning strategy for the Duplicator:

- if the Spoiler chooses an object in $\mathcal{D}$ in one structure, the Duplicator chooses the same object in the other structure;
- if the Spoiler chooses S_i, $1 \leq i \leq n$, in one structure, then the Duplicator chooses the same set name in the other structure;
- if the Spoiler chooses T_i, $1 \leq i \leq q_{\mathcal{S}}$, in γ_1' not selected before, then the Duplicator chooses one of $U_1, \ldots, U_k$ in γ_2' not selected before;[20]
- if the Spoiler chooses T_i, $1 \leq i \leq q_{\mathcal{S}}$, in γ_2' not selected before, then the Duplicator chooses one of $U_1, \ldots, U_k$ in γ_1' not selected before.[20]
- if the Spoiler chooses U_i, $1 \leq i \leq k$ in γ_1' not selected before, then the Duplicator chooses one of $T_1, \ldots, T_{q_{\mathcal{S}}}$ in γ_2' not selected before;[21]

[18] Observe again that the case where $k = q_{\mathcal{S}}$ holds trivially.
[19] In particular, this is the case for the condition $|\sigma| \geq 2q_{\mathcal{S}} - 1$.
[20] Since there are more than $q_{\mathcal{S}}$ set names available to choose from, this is always possible.
[21] Since there are $q_{\mathcal{S}}$ set names available to choose from, this is always possible.

- if the Spoiler chooses U_i, $1 \leq i \leq k$, in γ_2' not selected before, then the Duplicator chooses one of $T_1, \ldots, T_{q_\mathcal{S}}$ in γ_1' not selected before.[21]

Since set names selected in the same round are associated with precisely the same objects in γ_1' and γ_2', respectively, the above is indeed a winning strategy for the Duplicatator. As in the proof of Lemma 31, it now follows that, for all $o_1, \ldots, o_m \in \mathcal{D}$,

$$(\mathcal{D}, \mathcal{S}, \sigma', \gamma_1') \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma', \gamma_2') \models \varphi(o_1, \ldots, o_m),$$

which concludes the proof. □

The desired result of the second step of our step-by-step approach follows from a repeated application of Lemma 32, and is stated and proved below, as Corollary 33.

Corollary 33. *Let $\{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ be a counting-only SyCALC query, where φ has quantifier depth $q_\mathcal{D}$ in the lowercase variables and $q_\mathcal{S}$ in the uppercase variables. Let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1)$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2)$ be structures with $|\sigma| \geq 2q_\mathcal{S} - 1$ such that, for all $o \in \mathcal{D}$,*

- *either* $\mathsf{inc}(o, \gamma_1) = \mathsf{inc}(o, \gamma_2) \leq q_\mathcal{S} - 1$*;*
- *or* $\mathsf{coinc}(o, \gamma_1) = \mathsf{coinc}(o, \gamma_2) \leq q_\mathcal{S} - 1$*;*
- *or* $\mathsf{inc}(o, \gamma_1) \geq q_\mathcal{S}$*,* $\mathsf{coinc}(o, \gamma_1) \geq q_\mathcal{S}$*,* $\mathsf{inc}(o, \gamma_2) \geq q_\mathcal{S}$*, and* $\mathsf{coinc}(o, \gamma_2) \geq q_\mathcal{S}$*.*

Then, for all $o_1, \ldots, o_m \in \mathcal{D}$, $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1) \models \varphi(o_1, \ldots, o_m)$ if and only if $(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2) \models \varphi(o_1, \ldots, o_m)$.

Proof. Let $\{o^1, \ldots, o^k\}$ be the set of all objects such that, for $j = 1, \ldots, k$, $\mathsf{inc}(o^j, \gamma_1) \geq q_\mathcal{S}$ and $\mathsf{coinc}(o^j, \gamma_1) \geq q_\mathcal{S}$.[22] By the statement of Corollary 33, these are also all objects such that, for $j = 1, \ldots, k$, $\mathsf{inc}(o^j, \gamma_2) \geq q_\mathcal{S}$ and $\mathsf{coinc}(o^j, \gamma_2) \geq q_\mathcal{S}$. Notice that, for $o \in \mathcal{D} \setminus \{o^1, \ldots, o^k\}$, $\mathsf{inc}(o, \gamma_1') = \mathsf{inc}(o, \gamma_2')$. Now define $\gamma^0 := \gamma_1$, and, for $j = 1, \ldots, k$,

$$\gamma^j := (\gamma^{j-1} \setminus \{\langle o^j, S_i\rangle \mid \langle o^j, S_i\rangle \in \gamma_1\}) \cup \{\langle o^j, S_i\rangle \mid \langle o^j, S_i\rangle \in \gamma_2\}.$$

Clearly, $\gamma^k = \gamma_2$. By Lemma 32, we have that, for all $j = 1, \ldots, k$,

$$(\mathcal{D}, \mathcal{S}, \sigma, \gamma^{j-1}) \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma, \gamma^j) \models \varphi(o_1, \ldots, o_m).$$

Hence,

$$(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1) \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma, \gamma_2) \models \varphi(o_1, \ldots, o_m).$$ □

Using Lemma 31, we bootstrap Corollary 33 to the more general case where both structures do not necesarilly involve the same set names:

[22] Notice that this set may be empty if $|\sigma| = 2q_\mathcal{S} - 1$.

Lemma 34. *Let* $\{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ *be a counting-only* SyCALC *query, where* φ *has quantifier depth* $q_{\mathcal{D}}$ *in the lowercase variables and* $q_{\mathcal{S}}$ *in the uppercase variables. Let* $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1)$ *and* $(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2)$ *be structures with* $|\sigma_1| \geq 2q_{\mathcal{S}} - 1$ *and* $|\sigma_2| \geq 2q_{\mathcal{S}} - 1$ *such that, for all* $o \in \mathcal{D}$*,*

- *either* $\mathsf{inc}(o, \gamma_1) = \mathsf{inc}(o, \gamma_2) \leq q_{\mathcal{S}} - 1$*;*
- *or* $\mathsf{coinc}(o, \gamma_1) = \mathsf{coinc}(o, \gamma_2) \leq q_{\mathcal{S}} - 1$*;*
- *or* $\mathsf{inc}(o, \gamma_1) \geq q_{\mathcal{S}}$*,* $\mathsf{coinc}(o, \gamma_1) \geq q_{\mathcal{S}}$*,* $\mathsf{inc}(o, \gamma_2) \geq q_{\mathcal{S}}$*, and* $\mathsf{coinc}(o, \gamma_2) \geq q_{\mathcal{S}}$*.*

Then, for all $o_1, \ldots, o_m \in \mathcal{D}$*,* $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1) \models \varphi(o_1, \ldots, o_m)$ *if and only if* $(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2) \models \varphi(o_1, \ldots, o_m)$*.*

Proof. Without loss of generality, we may assume that $\sigma_1 \cap \sigma_2 = \emptyset$. Let $\sigma = \sigma_1 \cup \sigma_2$, and let

$$\gamma_1' = \gamma_1 \cup \{\langle o, T\rangle \mid \mathsf{coinc}(o, \gamma_1) \leq q_{\mathcal{S}} - 1 \;\&\; T \in \sigma_2\};$$
$$\gamma_2' = \gamma_2 \cup \{\langle o, S\rangle \mid \mathsf{coinc}(o, \gamma_2) \leq q_{\mathcal{S}} - 1 \;\&\; S \in \sigma_1\}.$$

Then, the structures $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1)$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1')$, respectively $(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2)$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2')$, satisfy the conditions of Lemma 31, and hence

$$(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1) \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma, \gamma_1') \models \varphi(o_1, \ldots, o_m);$$
$$(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2) \models \varphi(o_1, \ldots, o_m) \iff (\mathcal{D}, \mathcal{S}, \sigma, \gamma_2') \models \varphi(o_1, \ldots, o_m).$$

Lemma 34 now follows from the observation that the structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1')$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2')$ satisfy the conditions of Corollary 33. □

Before we can prove Theorem 30, we need to generalize Lemma 34 to the situation where the structures under consideration do not necessarily involve the same objects (Lemma 36). The key to this generalization is Lemmas 35, below. As both Lemmas 35 and 36 will be used to prove Theorem 30, Lemma 35 is stated slightly more general than strictly required to prove Lemma 36.

Lemma 35. *Let* $\{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ *be a counting-only* SyCALC *query. Let* $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1)$ *and* $(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2)$ *be structures with* $|\sigma_1| = |\sigma_2|$ *such that, for all* $o_1 \in \mathcal{D}$*, there exists* $o_2 \in \mathcal{D}$ *with* $\mathsf{inc}(o_2, \gamma_2) = \mathsf{inc}(o_1, \gamma_1)$*, and vice versa. Let* $o_{11}, \ldots, o_{1m}, o_{21}, \ldots, o_{2m} \in \mathcal{D}$ *be such that, for* $i = 1, \ldots, m$*,* $\mathsf{inc}(o_{1i}, \gamma_1) = \mathsf{inc}(o_{2i}, \gamma_2)$*. Then,* $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1) \models \varphi(o_{11}, \ldots, o_{1m})$ *if and only if* $(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2) \models \varphi(o_{21}, \ldots, o_{2m})$*.*

Proof. Let $\sigma_1 = \{S_1, \ldots, S_n\}$. We construct γ_1' from γ_1 as follows. Initially, $\gamma_1' = \emptyset$. Then, for all $o \in \mathcal{D}$, if $\mathsf{inc}(o, \gamma_1) = i$, add $\langle o, S_1\rangle, \ldots, \langle o, S_i\rangle$ to γ_1'. We construct γ_2' from γ_2 in a similar way. Since the SyCALC query under consideration is counting-only, we have that

$$(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1) \models \varphi(o_{11}, \ldots, o_{1m}) \iff (\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1') \models \varphi(o_{11}, \ldots, o_{1m});$$
$$(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2) \models \varphi(o_{21}, \ldots, o_{2m}) \iff (\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2') \models \varphi(o_{21}, \ldots, o_{2m}).$$

By assumption, $\{\mathsf{inc}(o,\gamma_1) \mid o \in \mathcal{D}\} = \{\mathsf{inc}(o,\gamma_2) \mid o \in \mathcal{D}\}$. Let us denote this set as $\{i_1,\ldots,i_k\}$. Now, for $j = 1,\ldots,k$, choose o_j^1 and o_j^2 in $\mathcal{D}$ such that $\mathsf{inc}(o_j^1,\gamma_1) = \mathsf{inc}(o_j^2,\gamma_2) = i_j$. We construct γ_1'' from γ_1 as follows. Initially, $\gamma_1'' = \emptyset$. Then, for all $j = 1,\ldots,k$, add $\langle o_j^1, S_1\rangle,\ldots,\langle o_j^1, S_{i_j}\rangle$ to γ_1''. We construct γ_2'' from γ_2 in a similar way. Let $h_1 : \mathcal{D} \to \mathcal{D}$ be the mapping sending an object o to the unique object o_j^1, $1 \leq j \leq k$, for which $\mathsf{inc}(o,\gamma_1) = i_j$. Similarly, let $h_2 : \mathcal{D} \to \mathcal{D}$ be the mapping sending an object o to the unique object o_j^2, $1 \leq j \leq k$, for which $\mathsf{inc}(o,\gamma_2) = i_j$. Observe that for each object o in $\mathcal{D}$, o and $h_1(o)$ are associated with exactly the same set names in γ_1', and o and $h_2(o)$ are associated with exactly the same set names in γ_2'. Since, furthermore, lowercase (object) variables are never compared in SyCALC, it follows that[23]

$$(\mathcal{D},\mathcal{S},\sigma_1,\gamma_1') \models \varphi(o_{11},\ldots,o_{1m}) \iff (\mathcal{D},\mathcal{S},\sigma_1,\gamma_1'') \models \varphi(h_1(o_{11}),\ldots,h_1(o_{1m}));$$
$$(\mathcal{D},\mathcal{S},\sigma_2,\gamma_2') \models \varphi(o_{21},\ldots,o_{2m}) \iff (\mathcal{D},\mathcal{S},\sigma_2,\gamma_2'') \models \varphi(h_2(o_{21}),\ldots,h_2(o_{2m})).$$

Now, let $h : \mathcal{D} \to \mathcal{D}$ be any bijective mapping sending o_i^1 to o_i^2, $1 \leq i \leq k$. Clearly, h defines an isomorphism between $(\mathcal{D},\mathcal{S},\sigma_1,\gamma_1'')$ and $(\mathcal{D},\mathcal{S},\sigma_2,\gamma_2'')$. Also, by construction, we have that $h(h_1(o_{11})) = h_2(o_{21}),\ \ldots,\ h(h_1(o_{1m})) = h_2(o_{2m})$. Hence, by genericity [23], $(\mathcal{D},\mathcal{S},\sigma_2,\gamma_1'') \models \varphi(h_1(o_{11}),\ldots,h_1(o_{1m}))$ if and only if $(\mathcal{D},\mathcal{S},\sigma_2,\gamma_2'') \models \varphi(h_2(o_{21}),\ldots,h_2(o_{2m}))$, which concludes the proof. □

Lemma 36. *Let $\{\langle x_1,\ldots,x_m\rangle \mid \varphi(x_1,\ldots,x_m)\}$ be a counting-only SyCALC query, where φ has quantifier depth $q_{\mathcal{D}}$ in the lowercase variables and $q_{\mathcal{S}}$ in the uppercase variables. Let $(\mathcal{D},\mathcal{S},\sigma_1,\gamma_1)$ and $(\mathcal{D},\mathcal{S},\sigma_2,\gamma_2)$ be structures with $|\sigma_1| \geq 2q_{\mathcal{S}} - 1$ and $|\sigma_2| \geq 2q_{\mathcal{S}} - 1$ such that, for all $o_1 \in \mathcal{D}$,*

- *if $\mathsf{inc}(o_1,\gamma_1) \leq q_{\mathcal{S}} - 1$, there exists $o_2 \in \mathcal{D}$ with $\mathsf{inc}(o_2,\gamma_2) = \mathsf{inc}(o_1,\gamma_1)$, and vice-versa;*
- *if $\mathsf{coinc}(o_1,\gamma_1) \leq q_{\mathcal{S}}-1$, there exists $o_2 \in \mathcal{D}$ with $\mathsf{coinc}(o_2,\gamma_2) = \mathsf{coinc}(o_1,\gamma_1)$, and vice-versa; and*
- *if $\mathsf{inc}(o_1,\gamma_1) \geq q_{\mathcal{S}}$ and $\mathsf{coinc}(o_1,\gamma_1) \geq q_{\mathcal{S}}$, there exists $o_2 \in \mathcal{D}$ with $\mathsf{inc}(o_2,\gamma_2) \geq q_{\mathcal{S}}$ and $\mathsf{coinc}(o_2,\gamma_2) \geq q_{\mathcal{S}}$, and vice-versa.*

Furthermore, let $o_{1i},\ldots,o_{1m},o_{21},\ldots,o_{2m} \in \mathcal{D}$ such that, for $i = 1,\ldots,m$,

- *either $\mathsf{inc}(o_{1i},\gamma_1) = \mathsf{inc}(o_{2i},\gamma_2) \leq q_{\mathcal{S}} - 1$;*
- *or $\mathsf{coinc}(o_{1i},\gamma_1) = \mathsf{coinc}(o_{2i},\gamma_2) \leq q_{\mathcal{S}} - 1$;*
- *or $\mathsf{inc}(o_{1i},\gamma_1) \geq q_{\mathcal{S}}$, $\mathsf{coinc}(o_{1i},\gamma_1) \geq q_{\mathcal{S}}$, $\mathsf{inc}(o_{2i},\gamma_2) \geq q_{\mathcal{S}}$, and $\mathsf{coinc}(o_{2i},\gamma_2) \geq q_{\mathcal{S}}$.*

Then, $(\mathcal{D},\mathcal{S},\sigma_1,\gamma_1) \models \varphi(o_{11},\ldots,o_{1m})$ if and only if $(\mathcal{D},\mathcal{S},\sigma_2,\gamma_2) \models \varphi(o_{21},\ldots,o_{2m})$.

[23] This intuition can be corroborated by a straightforward structural induction argument.

Proof. Let $n = 2q_{\mathcal{S}}$, and let $\sigma = \{S_1, \ldots, S_n\}$. Using γ_1, we construct γ_1', as follows. Initially, γ_1' is empty. Let $o \in \mathcal{D}$. If $1 \leq i = \mathsf{inc}(o, \gamma_1) \leq q_{\mathcal{S}} - 1$, then add $\langle o, S_1\rangle, \ldots, \langle o, S_i\rangle$ to γ_1'. Otherwise, if $j = \mathsf{coinc}(o, \gamma_1) \leq q_{\mathcal{S}} - 1$, then add $\langle o, S_1\rangle, \ldots, \langle o, S_{n-j}\rangle$ to γ_1'. Otherwise, i.e., if $\mathsf{inc}(o, \gamma_1) \geq q_{\mathcal{S}}$ and $\mathsf{coinc}(o, \gamma_1) \geq q_{\mathcal{S}}$, add $\langle o, S_1\rangle, \ldots, \langle o, S_{q_{\mathcal{S}}}\rangle$ to γ_1'.[24] Using γ_2, we construct γ_2' in a similar way. The structures $(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1)$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1')$ satisfy the conditions of Lemma 34, and hence

$$(\mathcal{D}, \mathcal{S}, \sigma_1, \gamma_1) \models \varphi(o_{11}, \ldots, o_{1m}) \iff (\mathcal{D}, \mathcal{S}, \sigma, \gamma_1') \models \varphi(o_{11}, \ldots, o_{1m});$$
$$(\mathcal{D}, \mathcal{S}, \sigma_2, \gamma_2) \models \varphi(o_{21}, \ldots, o_{2m}) \iff (\mathcal{D}, \mathcal{S}, \sigma, \gamma_2') \models \varphi(o_{21}, \ldots, o_{2m}).$$

Lemma 36 now follows from the observation that the structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_1')$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_2')$ satisfy the conditions of Lemma 35. □

Using Lemmas 35 and 36. we can now prove Theorem 30, stated on p. 31.

of Theorem 30. Let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ be a structure and let $o_1, \ldots, o_m \in \mathcal{D}$ such that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_1, \ldots, o_m)$. We now construct a SyCALC formula $\varphi_{\sigma,\gamma,\vec{o}}$ describing the incidence information contained herein, where $\vec{o}$ denotes the sequence $o_1, \ldots, o_m$. Thereto, we distinguish two cases.

1. $n = |\sigma| < 2q_{\mathcal{S}} - 1$. Then, let $\varphi_{\sigma,\gamma,\vec{o}}$ be the formula $\mathsf{Eq}(n) \wedge \psi_{\sigma,\gamma,\vec{o}}$, where $\psi_{\sigma,\gamma,\vec{o}}$ is a conjunction of the following formulae:
   - for $i = 1, \ldots, n$, $\exists x\ \mathsf{eq}(x, i)$ if there exists $o \in \mathcal{D}$ with $\mathsf{inc}(o, \gamma) = i$, and $\neg\exists x\ \mathsf{eq}(x, i)$ otherwise; and
   - for $j = 1, \ldots, m$, $\mathsf{eq}(x_j, \mathsf{inc}(o_j, \gamma))$.
2. $n = |\sigma| \geq 2q_{\mathcal{S}} - 1$. Let $\varphi_{\sigma,\gamma,\vec{o}}$ be the formula $\mathsf{Gteq}(2q_{\mathcal{S}} - 1) \wedge \psi_{\sigma,\gamma,\vec{o}}$, where $\psi_{\sigma,\gamma,\vec{o}}$ is a conjunction of the following formulae:
   - for $i = 1, \ldots, q_{\mathcal{S}} - 1$, $\exists x\ \mathsf{eq}(x, i)$ if there exists $o \in \mathcal{D}$ with $\mathsf{inc}(o, \gamma) = i$, and $\neg\exists x\ \mathsf{eq}(x, i)$ otherwise;
   - $\exists x\ (\mathsf{gteq}(x, q_{\mathcal{S}}) \wedge \mathsf{cogteq}(x, q_{\mathcal{S}}))$ if there exists $o \in \mathcal{D}$ with $q_{\mathcal{S}} \leq \mathsf{inc}(o, \gamma) \leq n - q_{\mathcal{S}}$, and $\neg\exists x\ (\mathsf{gteq}(x, q_{\mathcal{S}}) \wedge \mathsf{cogteq}(x, q_{\mathcal{S}}))$ otherwise;
   - for $j = q_{\mathcal{S}} - 1, \ldots, 0$, $\exists x\ \mathsf{coeq}(x, j)$ if there exists $o \in \mathcal{D}$ with $\mathsf{inc}(o, \gamma) = n - j$, and $\neg\exists x\ \mathsf{coeq}(x, j)$ otherwise;
   - for $\ell = 1, \ldots, m$, $\alpha_\ell(x_\ell)$, which equals

$$\begin{cases} \mathsf{eq}(x_\ell, \mathsf{inc}(o_\ell, \gamma)) & \text{if } \mathsf{inc}(o_\ell, \gamma) < q_{\mathcal{S}}; \\ \mathsf{coeq}(x_\ell, \mathsf{inc}(o_\ell, \gamma)) & \text{if } \mathsf{inc}(o_\ell, \gamma) > n - q_{\mathcal{S}}; \\ \mathsf{gteq}(x_\ell, q_{\mathcal{S}}) \wedge \mathsf{cogteq}(x_\ell, q_{\mathcal{S}}) & \text{otherwise.} \end{cases}$$

[24] Notice that $\mathsf{inc}(o, \gamma_1') = q_{\mathcal{S}}$ and $\mathsf{coinc}(o, \gamma_1') = q_{\mathcal{S}}$.

Now, let $\mathbf{q}' := \{\langle x_1, \ldots, x_m\rangle \mid \varphi'(x_1, \ldots, x_m)\}$ with φ' equal to

$$\bigvee_{\substack{\sigma,\gamma,\vec{o} \text{ with} \\ (\mathcal{D},\mathcal{S},\sigma,\gamma)\models\varphi(\vec{o})}} \varphi_{\sigma,\gamma,\vec{o}}(x_1, \ldots, x_m).$$

From the onset, it appears that φ' may be an infinite disjunction, as the size of σ is in principle unbounded and $\vec{o}$ is drawn from the infinite enumerable domain $\mathcal{D}$. Closer inspection of the construction of the formula $\varphi_{\sigma,\gamma,\vec{o}}(x_1, \ldots, x_m)$ reveals, however, that there are only finitely many different such formulae (their number being bounded by a function of $q_{\mathcal{S}}$ only). Hence, by ignoring duplicates, we may perceive the disjunction as finite, and $\mathbf{q}'$ as a well-formed SyCALC formula.

We claim that the original counting-only SyCALC query $\mathbf{q}$ is equivalent to $\mathbf{q}'$. To see this, let $(\mathcal{D}, \mathcal{S}, \sigma', \gamma')$ be a structure and let $o'_1, \ldots, o'_m \in \mathcal{D}$ be such that $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi(o'_1, \ldots, o'_m)$. Hence, $\varphi_{\sigma',\gamma',\vec{o}'}$ is a disjunct of φ', and $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi'(o'_1, \ldots, o'_m)$. Conversely, assume that $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi'(o'_1, \ldots, o'_m)$. Hence, for at least one of the disjuncts $\varphi_{\sigma,\gamma,\vec{o}}$ of φ', $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi_{\sigma,\gamma,\vec{o}}(o'_1, \ldots, o'_m)$. Since $\varphi_{\sigma,\gamma,\vec{o}}$ is a disjunct of φ', we also know that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_1, \ldots, o_m)$. We distinguish two cases:

1. $|\sigma| < 2q_{\mathcal{S}} - 1$. Then, by construction of $\varphi_{\sigma,\gamma,\vec{o}}$ and from $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi_{\sigma,\gamma,\vec{o}}(o'_1, \ldots, o'_m)$, it follows that the conditions of Lemma 35 are met for the structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ and $(\mathcal{D}, \mathcal{S}, \sigma', \gamma')$ and the objects $o_1, \ldots, o_m$, $o'_1, \ldots, o'_m$. Since $\mathbf{q}$ is counting-only, $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_1, \ldots, o_m)$ implies $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi(o'_1, \ldots, o'_m)$.

2. $|\sigma| \geq 2q_{\mathcal{S}} - 1$. We reason precisely as in the previous case, except that we use Lemma 36 instead of Lemma 35. Hence, also in this case, we may conclude that $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi(o'_1, \ldots, o'_m)$.

It now suffices to observe that all the disjuncts of $\varphi'(x_1, \ldots, x_m)$ are of the form described in the statement of this Theorem. □

Remark 37. Superficially, the proof of Theorem 30 seems non-constructive, because of the argument involving the infinite disjunction. A closer look to the construction of the subformulae $\varphi_{\sigma,\gamma,\vec{o}}(x_1, \ldots, x_m)$ of that disjunction reveals that it is possible to consider all different such formulae by only considering structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ with at most $2q_{\mathcal{S}} - 1$ set names in σ and at most $2q_{\mathcal{S}} - 1$ objects in the active domain (to ensure that, for all $N \subseteq \{1, \ldots, |\sigma|\}$, one can construct γ in such a way that $\{\mathsf{inc}(o, \gamma) \mid \exists \in \sigma\ \langle o, S\rangle \in \gamma\} = N$). Notice that, upon isomorphism, these structures $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ and objects $o_1, \ldots, o_m$ can be finitely enumerated. It can also be verified in each instance whether $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_1, \ldots, o_m)$, and hence whether $\varphi_{\sigma,\gamma,\vec{o}}(x_1, \ldots, x_m)$ is part of the disjunction.

Example 38. As shown in Example 29, the SyCALC queries in Example 18 expressing Queries 3–5 are counting-only.

The SyCALC query expressing Query 3 can be rewritten as $\{\langle\rangle \mid \neg\exists x\ \mathsf{gteq}(x, 2)\}$; the SyCALC query expressing Query 4 can be rewritten as $\{x \mid \mathsf{eq}(x, 1) \wedge$

$\exists y\ \mathsf{gteq}(y, 3)\}$; and, finally, the SyCALC query expressing Query 5 can be rewritten as $\{\langle\rangle \mid \neg\exists x\ (\mathsf{gteq}(x, 1) \wedge \mathsf{cogteq}(x, 1))\}$.

The rewritten queries conform to Theorem 30, after applying some straightforward simplifications. In particular, we did not have to distinguish between different sizes of σ. This is not always the case, however, as was already illustrated in Example 17 for QuineCALC queries (which are special cases of counting-only SyCALC queries).

The formulae $\mathsf{Eq}(n)$ or $\mathsf{Gteq}(2q_{\mathcal{S}} - 1)$ in the statement of Theorem 30 are of course not QuineCALC formulae (if only because they do not have a free lowercase variable). However, they can easily be grouped with one of the formulae with which they are conjoined, so that we can derive the following corollary to Theorem 30.

Corollary 39. *Let* $\mathbf{q} := \{\langle x_1, \ldots, x_m\rangle \mid \varphi(x_1, \ldots, x_m)\}$ *be a SyCALC query. Then* $\mathbf{q}$ *is counting-only if and only if* φ *is equivalent to a quantified Boolean combination of QuineCALC query formulae.*

In other words, the query language that includes QuineCALC and that is closed under quantification and Boolean operators is equivalent to the language of the counting-only SyCALC queries.

Theorem 30 and Corollary 39 also provide a positive answer to Research Question 7.

6. Decidability

Here, we consider the decision problems stated in Research Question 8 and 9. We first show that it is decidable if a counting-only SyCALC query is a QuineCALC query (Section 6.1), but that it is undecidable if a SyCALC query is counting-only (Section 6.2). We then show that satisfiability, validity, emptiness, containment, and equivalence are decidable for counting-only SyCALC queries, but undecidable for general SyCALC queries (Section 6.3).

6.1. Is a counting-only SyCALC query a QuineCALC query?

All QuineCALC queries are unary counting-only SyCALC queries, but, as already established in Example 29, not all unary counting-only SyCALC queries are equivalent to a QuineCALC query. Comparing the normal forms exhibited for QuineCALC queries in Corollary 16 respectively for general counting-only SyCALC queries in Theorem 30, we also see why.

Let $\mathbf{q} := \{x \mid \varphi(x)\}$ be a unary query, let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ be a structure, and let $o \in \mathcal{D}$. The normal form for QuineCALC queries in Corollary 16 reveals that, for a fixed size of σ, the truth of $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o)$ in that case only depends on the incidence of o in the structure. For general counting-only SyCALC queries, on the other hand, the normal form in Theorem 30 reveals that the incidence of other objects in the structure may also play a role.

It turns out that the difference between QuineCALC queries and general counting-only SyCALC queries exhibited above actually *characterizes* the distinction between both. Moreover, this difference, appropriately formalized, may actually be used to decide whether a unary counting-only SyCALC query is equivalent to a QuineCALC query, as is shown next.

Theorem 40. *It is decidable whether a counting-only SyCALC query is equivalent to a QuineCALC query.*

Proof. By definition, non-unary queries cannot be equivalent to QuineCALC queries. Thus, consider a unary counting-only SyCALC query $\mathbf{q} := \{x \mid \varphi'(x)\}$ with quantifier depth $q_{\mathcal{S}}$ in the uppercase variables. Let $S_1, \ldots, S_{2q_{\mathcal{S}}-1} \in \mathcal{S}$ be pairwise different set names, and let $o_0, o_1, \ldots, o_{2q_{\mathcal{S}}-1} \in \mathcal{D}$ be pairwise different objects. Now, consider n, $1 \leq n \leq 2q_{\mathcal{S}} - 1$ and $N \subseteq \{0, \ldots, n\}$. Let $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_N)$ be the structure where $\sigma_n = \{S_1, \ldots, S_n\}$ and

$$\gamma_N = \bigcup_{i \in N} \{(o_i, S_1), \ldots, (o_i, S_i)\}.^{25}$$

Define $K_{n,N} = \{k \in N \mid (\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_N) \models \varphi(o_k)\}$. We claim that $\mathbf{q}$ is equivalent to a QuineCALC query if and only if, for all $n = 1, \ldots, 2q_{\mathcal{S}} - 1$, and for all $N_1, N_2 \subseteq \{0, \ldots, n\}$, $K_{n,N_1} \cap N_2 = K_{n,N_2} \cap N_1$.[26] We now prove this claim.

- We start with the "only if". Thus suppose $\mathbf{q}$ is equivalent to a QuineCALC query of the form shown in Corollary 16. We use $\psi(x)$ and $\psi_n(x)$ as defined in Corollary 16. Observe that the quantifier depth q occurring in the expression for φ' need not be equal to $q_{\mathcal{S}}$! By symmetry, it suffices to prove that, for all $n = 1, \ldots, 2q_{\mathcal{S}} - 1$, and for all $N_1, N_2 \subseteq \{0, \ldots, n\}$, $K_{n,N_1} \cap N_2 \subseteq K_{n,N_2} \cap N_1$. So, assume that for some n, N_1, and N_2, $k \in K_{n,N_1} \cap N_2$. In particular, $k \in N_1$, from which we derive that $\mathsf{inc}(o_k, \gamma_{N_1}) = k$, and $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \varphi'(o_k)$. We again distinguish two cases:

  1. $n \leq 2q-2$. Since $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \varphi'(o_k)$, we have $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \psi_n(o_k)$. This is only possible if $\psi_n(x)$ contains the disjunct $\mathsf{eq}(x, k)$. Since $k \in N_2$, we have $\mathsf{inc}(o_k, \gamma_{N_2}) = k$, and hence $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_2}) \models \psi_n(o_k)$, $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_2}) \models \varphi'(o_k)$, and $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_2}) \models \varphi(o_k)$. Hence, $k \in K_{n,N_2}$.
  2. $n \geq 2q-1$. Since $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \varphi'(o_k)$, we have $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \psi(o_k)$. We now distinguish three subcases:
     (a) $k < q$. Then $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \psi(o_k)$ can only hold if $\psi(x)$ contains the disjunct $\mathsf{eq}(x, k)$.
     (b) $q \leq k \leq n - q$. Then $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \psi(o_k)$ can only hold if $\psi(x)$ contains the disjunct $\mathsf{gteq}(x, q) \wedge \mathsf{cogteq}(x, q)$.

[25] Hence, even if $0 \in N$, o_0 never occurs in γ_N.

[26] Intuitively, this condition expresses that the truth of $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_N) \models \varphi(o_k)$ only depends on the incidence of o_k in this structure and not on the incidence of other objects.

(c) $k > n - q$. Then $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \psi(o_k)$ can only hold if $\psi(x)$ contains the disjunct $\mathsf{coeq}(x, n-k)$.

The remainder of the reasoning in all three subcases is now completely analogous to Case 1.

- We now turn to the "if". Since $\mathbf{q}$ is counting-only, we may assume it is equivalent to a query of the form shown in Theorem 30 with $m = 1$. We use $\psi(x)$ and $\psi_n(x)$ as defined in Theorem 30. We show now that the condition that, for all $n = 1, \ldots, 2q_{\mathcal{S}} - 1$, and for all $N_1, N_2 \subseteq \{0, \ldots, n\}$, $K_{n,N_1} \cap N_2 = K_{n,N_2} \cap N_1$ implies that subformula with existentially quantified domain (lowercase) variables can be eliminated from φ', and hence that $\mathbf{q}$ is equivalent to a QuineCALC query.

  1. We first consider the subformula $\psi_n(x_1)$, $1 \leq n \leq 2q_{\mathcal{S}} - 2$.[27] Let $\vartheta_{11} \wedge \ldots \wedge \vartheta_{1n} \wedge \alpha_1(x_1)$ be one of the disjuncts of ψ_n, with $\alpha_1(x_1) := \mathsf{eq}(x_1, k)$, $0 \leq k \leq n$. We distinguish two subcases:
     (a) $k = 0$. Let $N_1 = \{0\} \cup \{i \mid 1 \leq i \leq n \;\&\; \vartheta_{1i} \equiv \exists x\ \mathsf{eq}(x, i)\}$. Clearly, $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \varphi'(o_0)$, and, hence, $0 \in K_{n,N_1}$. Now, let N_2 be any subset of $\{0, 1, \ldots, n\}$ containing 0. Since $0 \in K_{n,N_1} \cap N_2 = K_{n,N_2} \cap N_1$, it follows that $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_2}) \models \varphi'(o_0)$. This is only possible, however, if ψ_n contains a disjunct $\vartheta_{21} \wedge \ldots \wedge \vartheta_{2n} \wedge \alpha_1(x_1)$, where, for $i = 1, \ldots, n$, ϑ_{2i} is $\exists x\ \mathsf{eq}(x, i)$ if $i \in N_2$ and $\neg\exists x\ \mathsf{eq}(x, i)$ otherwise. Hence, all disjuncts of $\psi_n(x_1)$ containing $\alpha_1(x_1)$ together are logically equivalent to $\alpha_1(x_1)$.
     (b) $1 \leq k \leq n$. Without loss of generality, we may assume that ϑ_k is $\exists x\ \mathsf{eq}(x, k)$, otherwise the disjunct is unsatisfiable and can be omitted. Let $N_1 = \{i \mid 1 \leq i \leq n \;\&\; \vartheta_{1i} \equiv \exists x\ \mathsf{eq}(x, i)\}$. Clearly, $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_1}) \models \varphi'(o_k)$, and, hence, $k \in K_{n,N_1}$. Now, let N_2 be any subset of $\{0, 1, \ldots, n\}$ containing k. Since $k \in K_{n,N_1} \cap N_2 = K_{n,N_2} \cap N_1$, it follows that $(\mathcal{D}, \mathcal{S}, \sigma_n, \gamma_{N_2}) \models \varphi'(o_k)$. This is only possible, however, if ψ_n contains a disjunct $\vartheta_{21} \wedge \ldots \wedge \vartheta_{2n} \wedge \alpha_1(x_1)$, where, for $i = 1, \ldots, n$, ϑ_{2i} is $\exists x\ \mathsf{eq}(x, i)$ if $i \in N_2$ and $\neg\exists x\ \mathsf{eq}(x, i)$ otherwise. Hence, all satisfiable disjuncts of $\psi_n(x_1)$ containing $\alpha_1(x_1)$ together are logically equivalent to $(\exists x\ \mathsf{eq}(x, k)) \wedge \alpha_1(x_1)$, which in turn is logically equivalent to $\alpha_1(x_1)$.
  2. We next consider the subformula $\psi(x_1)$. Let

$$\vartheta_1 \wedge \ldots \wedge \vartheta_{q_{\mathcal{S}}-1} \wedge \vartheta \wedge \vartheta^{q_{\mathcal{S}}-1} \wedge \ldots \vartheta^0 \wedge \alpha_1(x_1)$$

     be a subformula of $\psi(x)$. We distinguish three subcases:
     (a) $\alpha_1(x)$ is $\mathsf{eq}(x, k)$, with $0 \leq k < q_{\mathcal{S}}$. Then, choose $n = 2q_{\mathcal{S}} - 1$ and proceed as in Case 1 for $\psi_n(x_1)$ with $1 \leq n \leq 2q_{\mathcal{S}} - 2$.

[27] We need not consider $n = 0$, since, by construction, ψ_0 is free of quantification over domain variables.

(b) $\alpha_1(x)$ is $\mathsf{gteq}(x, q_{\mathcal{S}}) \wedge \mathsf{cogteq}(x, q_{\mathcal{S}})$. Then, choose $n = 2q_{\mathcal{S}} - 1$ and $k = q_{\mathcal{S}}$, and proceed as in Subcase 1b of Case 1 for $\psi_n(x_1)$ with $1 \leq n \leq 2q_{\mathcal{S}} - 2$.

(c) $\alpha_1(x)$ is $\mathsf{coeq}(x, k')$, with $0 \leq k' < q_{\mathcal{S}}$. Then, choose $n = 2q_{\mathcal{S}} - 1$ and $k = n - k'$, and proceed as in Subcase 1b of Case 1 for $\psi_n(x_1)$ with $1 \leq n \leq 2q_{\mathcal{S}} - 2$.

It now suffices to observe that there are only a finite number of sets $K_{n,N}$, $1 \leq n \leq 2q_{\mathcal{S}} - 1$ and $N \subseteq \{0, \ldots, n\}$, that can all be computed. Hence, the condition shown above to be equivalent with "$\mathbf{q}$ being equivalent to a QuineCALC query" can effectively be evaluated. □

Hence, Research Question 8 has a positive answer for QuineCALC queries.

6.2. Is a SyCALC query counting-only?

We show that this problem is undecidable by a reduction of satisfiabilty of a domain-independent Boolean relational calculus query which uses only one, ternary, relation symbol and no constants, which is undecidable [24, Theorem 6.3.1 and Exercise 6.19], to deciding whether a SyCALC query is counting-only.

The reduction consists of two steps. First, in Section 6.2.1, we show how to encode arbitrary ternary relations by binary relations which can be represented by the structures considered in this paper. Then, in Section 6.2.2, we provide the actual reduction.

6.2.1. Encoding ternary relations in binary relations

Let $I = \{t_1, \ldots, t_n\}$ be a set of triples that do not use the pairwise different constants $\sharp\mathbf{0}$, $\$\mathbf{1}$, $\$\mathbf{2}$, $\$\mathbf{3}$, $\$\mathbf{4}$, and, for $1 \leq i \leq n$, the constants $\hat{\imath}$, $\sharp i_1$, $\sharp i_2$, $\sharp i_3$, $\$i_1$, $\$i_2$, and $\$i_3$. We shall refer to these constants as *encoding constants*. We now construct the binary relation containing the pairs $\langle \sharp\mathbf{0}, \$\mathbf{1} \rangle$, $\langle \sharp\mathbf{0}, \$\mathbf{2} \rangle$, $\langle \sharp\mathbf{0}, \$\mathbf{3} \rangle$, $\langle \sharp\mathbf{0}, \$\mathbf{4} \rangle$, and, for every triple $t_i = \langle A_i, B_i, C_i \rangle$, $1 \leq i \leq n$, the tuples:

- $\langle \sharp\mathbf{0}, \$i_1 \rangle$, $\langle \sharp\mathbf{0}, \$i_2 \rangle$, and $\langle \sharp\mathbf{0}, \$i_3 \rangle$;
- $\langle \hat{\imath}, A_i \rangle$;
- $\langle \hat{\imath}, \$i_1 \rangle$, $\langle \sharp i_1, \$i_1 \rangle$, and $\langle \sharp i_1, B_i \rangle$; and
- $\langle \hat{\imath}, \$i_2 \rangle$, $\langle \sharp i_2, \$i_2 \rangle$, $\langle \sharp i_2, \$i_3 \rangle$, $\langle \sharp i_3, \$i_3 \rangle$, and $\langle \sharp i_3, C_i \rangle$.

Example 41. Let $I = \{\langle A_1, B_1, C_1 \rangle, \langle A_2, B_2, C_2 \rangle\}$ be a ternary relation with two triples. If we apply the above construction to I, we obtain the binary relation in Figure 4, *left*. In Figure 4, *right*, we have visualized this relation (except for the parts involving $\sharp\mathbf{0}$).

Let $\Delta = \{\sharp\mathbf{0}\} \cup \big(\bigcup_{1 \leq i \leq n} \{\hat{\imath}, \sharp i_1, \sharp i_2, \sharp i_3\}\big)$ and

$$\sigma = \{\$\mathbf{1}, \$\mathbf{2}, \$\mathbf{3}, \$\mathbf{4}\} \cup \big(\bigcup_{1 \leq i \leq n} \{A_i, B_i, C_i, \$i_1, \$i_2, \$i_3\}\big).$$

	$\langle A_1, B_1, C_1\rangle$	$\langle A_2, B_2, C_2\rangle$
$\langle \sharp\mathbf{0}, \$\mathbf{1}\rangle$	$\langle \sharp\mathbf{0}, \$1_1\rangle$	$\langle \sharp\mathbf{0}, \$2_1\rangle$
$\langle \sharp\mathbf{0}, \$\mathbf{2}\rangle$	$\langle \sharp\mathbf{0}, \$1_2\rangle$	$\langle \sharp\mathbf{0}, \$2_2\rangle$
$\langle \sharp\mathbf{0}, \$\mathbf{3}\rangle$	$\langle \sharp\mathbf{0}, \$1_3\rangle$	$\langle \sharp\mathbf{0}, \$1_3\rangle$
$\langle \sharp\mathbf{0}, \$\mathbf{4}\rangle$	$\langle \hat{1}, A_1\rangle$	$\langle \hat{2}, A_2\rangle$
	$\langle \hat{1}, \$1_1\rangle$	$\langle \hat{2}, \$2_1\rangle$
	$\langle \sharp 1_1, \$1_1\rangle$	$\langle \sharp 2_1, \$2_1\rangle$
	$\langle \sharp 1_1, B_1\rangle$	$\langle \sharp 2_1, B_2\rangle$
	$\langle \hat{1}, \$1_2\rangle$	$\langle \hat{2}, \$2_2\rangle$
	$\langle \sharp 1_2, \$1_2\rangle$	$\langle \sharp 2_2, \$2_2\rangle$
	$\langle \sharp 1_2, \$1_3\rangle$	$\langle \sharp 2_2, \$2_3\rangle$
	$\langle \sharp 1_3, \$1_3\rangle$	$\langle \sharp 2_3, \$2_3\rangle$
	$\langle \sharp 1_3, C_1\rangle$	$\langle \sharp 2_3, C_2\rangle$

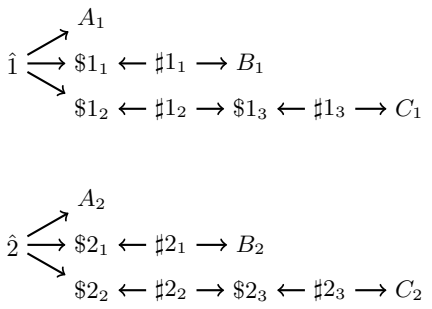


Figure 4: The ternary relation $I = \{\langle A_1, B_1, C_1\rangle, \langle A_2, B_2, C_2\rangle\}$ translated to a binary relation.

Observe that $\Delta \cap \sigma = \emptyset$ and that the constructed binary relation is a subset of $\Delta \times \sigma$. Therefore, we can easily store the binary relation as a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ with $\Delta \subseteq \mathcal{D}$, $\sigma \subseteq \mathcal{S}$, and γ the constructed binary relation. We write $\mathsf{TerToBi}(I)$ to denote this structure (or any isomorphic structure obtained by renaming the encoding constants).[28]

We now exhibit a SyCALC formula $\mathit{Triple}(X, Y, Z)$ such that $\mathsf{TerToBi}(I) \models \mathit{Triple}(A, B, C)$ if and only if $\langle A, B, C\rangle \in I$. In order to do so, we first observe that it is not necessary to know which object and set names represent which type of encoding constant, or which set names represent entries of the ternary relation I. It turns out that we can derive this information from the encoding, as follows.

1. The *zero constant* $\sharp\mathbf{0}$ can be distinguished as the only object in the encoding associated to at least four set names. We express this by $\mathit{Zero}(x) := \mathsf{gteq}(x, 4)$.

2. The *identifier constants* $\hat{\imath}$, $1 \leq i \leq n$, can be distinguished as precisely those objects in the encoding associated to exactly three set names. We express this by $\mathit{Id}(x) := \mathsf{eq}(x, 3)$.

3. The *sharp constants* $\sharp i_1$, $\sharp i_2$, and $\sharp i_3$, $1 \leq i \leq n$, can be distinguished as precisely those objects in the encoding associated to exactly two set names. We express this by $\mathit{Sharp}(x) := \mathsf{eq}(x, 2)$.

4. The *dollar constants* $\$i_1$, $\$i_2$, and $\$i_3$, $1 \leq i \leq n$, can be distinguished as precisely those set names in the encoding to which the zero constant has been associated. We express this by $\mathit{DollarSet}(X) := \exists z\ (\mathit{Zero}(z) \wedge \Gamma(z, X))$.

[28] Genericity considerations [23] allow us to ignore this minor ambiguity.

5. All other set names in the encoding are referred to as *pure constants*, and these are precisely the entries A_i, B_i, and C_i, $1 \leq i \leq n$, of the triples of the original ternary relation I. We express this by $\mathit{PureSet}(X) := \neg \mathit{DollarSet}(X) \wedge \exists z\, \Gamma(z, X)$.

Looking at the encoding of the triple $t_i = \langle A_i, B_i, C_i \rangle$, $1 \leq i \leq n$, we additionally observe the following:

6. The first entry of triple t_i, A_i, is represented by in the binary membership relation of the encoding the single pair $\langle \hat{\imath}, A_i \rangle$. We express this by $\mathit{First}(x, X) := \mathit{Id}(x) \wedge \mathit{PureSet}(X) \wedge \Gamma(x, X)$.

7. The second entry of triple t_i, B_i, is represented in the binary membership relation of the encoding by a so-called *short path* consisting of the three pairs $\langle \hat{\imath}, \$i_1 \rangle$, $\langle \sharp i_1, \$i_1 \rangle$, $\langle \sharp i_1, B_i \rangle$. We express this by

$$\begin{aligned}\mathit{ShortPath}(x, Y_1, z_1, Y) := \mathit{Id}(x) \wedge \mathit{DollarSet}(Y_1) \wedge \mathit{Sharp}(z_1) \wedge \\ \mathit{PureSet}(Y) \wedge \Gamma(x, Y_1) \wedge \Gamma(z_1, Y_1) \wedge \Gamma(z_1, Y).\end{aligned}$$

8. The third entry of triple t_i, C_i, is represented in the binary membership relation of the encoding by a so-called *long path* consisting of the five pairs $\langle \hat{\imath}, \$i_2 \rangle$, $\langle \sharp i_2, \$i_2 \rangle$, $\langle \sharp i_2, \$i_3 \rangle$, $\langle \sharp i_3, \$i_3 \rangle$, $\langle \sharp i_3, C_i \rangle$. We express this by

$$\begin{aligned}\mathit{LongPath}(x, Y_2, z_2, Y_3, z_3, Z) := \mathit{Id}(x) \wedge \mathit{DollarSet}(Y_2) \wedge \mathit{Sharp}(z_2) \wedge \\ \mathit{DollarSet}(Y_3) \wedge \mathit{Sharp}(z_3) \wedge \mathit{PureSet}(Z) \wedge Y_2 \neq Y_3 \wedge \\ \Gamma(x, Y_2) \wedge \Gamma(z_2, Y_2) \wedge \Gamma(z_2, Y_3) \wedge \Gamma(z_3, Y_3) \wedge \Gamma(z_3, Z).\end{aligned}$$

This encoding using short and long paths is graphically visualized in Figure 4, *right*, for the ternary instance considered in Example 41. We now define the following abbreviation:

$$\begin{aligned}\mathit{Triple}(X, Y, Z) := \exists x \exists Y_1 \exists Y_2 \exists Y_3 \exists z_1 \exists z_2 \exists z_3 \\ \mathit{First}(x, X) \wedge \mathit{ShortPath}(x, Y_1, z_1, X) \wedge \mathit{LongPath}(x, Y_2, z_2, Y_3, z_3, Z),\end{aligned}$$

Clearly, $\langle A, B, C \rangle \in I$ if and only if $\mathsf{TerToBi}(I) \models \mathit{Triple}(A, B, C)$.

Of course, the SyCALC formula $\mathit{Triple}(X, Y, Z)$ can also be applied to arbitrary structures. Given such a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, we associate to it the ternary relation $\mathsf{BiToTer}(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ as the set of all triples $\langle A, B, C \rangle$ for which $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \mathit{Triple}(A, B, C)$.

Upon evaluating this formula, the various objects and set names corresponding to encoding constants and pure constants must of course satisfy the constraints expressed by the subformulae in Items 1–8 above. From these, additional constraints can be inferred. To see this, consider a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, and assume that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \mathit{LongPath}(\hat{\imath}, \$i_2, \sharp i_2, \$i_3, \sharp i_3, C_i)$. Taking into account that objects and set names are disjoint, the formulae in Items 1–5 distinguish pure constants from encoding constants, as well as all the various

types of encoding constants we considered. In particular, $\$i_3 \neq C_i$. By definition of long path, we also have $\$i_2 \neq \$i_3$ (Item 8). Since each sharp constant is associated with precisely two set names (Item 3), it follows that $\sharp i_2 \neq \sharp i_3$. For the same reason, $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \not\models \mathit{LongPath}(\hat{\jmath}, \$j_2, \sharp i_3, \$j_3, \sharp i_2, C_j)$, irrespective of $\hat{\jmath}$, $\$j_2$, $\$j_3$, and C_j.

If both $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \mathit{LongPath}(\hat{\imath}, \$i_2, \sharp i_2, \$i_3, \sharp i_3, C_i)$ and $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \mathit{ShortPath}(\hat{\imath}, \$i_1, \sharp i_1, B_i)$, we can use an analogous argument to conclude that $\sharp i_1 \neq \sharp i_2$. However, it is possible that $\sharp i_1 = \sharp i_3$, provided also $\$i_1 = \$i_3$ and $B_i = C_i$.

Example 42. Consider a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ of which the membership relation contains the pairs shown graphically in Figure 5. Then, $\langle A, D, D\rangle \in \mathsf{BiToTer}(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$.

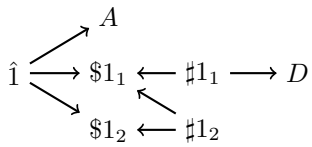


Figure 5: For any structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$. whose membership relation contains the pairs visualized above, $\mathsf{BiToTer}(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ contains $\langle A, D, D\rangle$.

Clearly, for a ternary relation I, we have $\mathsf{BiToTer}(\mathsf{TerToBi}(I)) = I$.

6.2.2. Reduction

Let φ be a domain-independent relational calculus query over a single ternary relation named R and that does not use constants. For convenience, we use uppercase variables for the variables in φ. We recursively translate φ to a SyCALC formula $[\![\varphi]\!]$, with the same free variables, as follows:

$$
\begin{aligned}
[\![R(A, B, C)]\!] &:= \mathit{Triple}(A, B, C);\\
[\![X = Y]\!] &:= X = Y;\\
[\![\neg\varphi]\!] &:= \neg[\![\varphi]\!];\\
[\![\exists X\ \varphi]\!] &:= \exists X\ [\![\varphi]\!];\\
[\![\varphi_1 \wedge \varphi_2]\!] &:= [\![\varphi_1]\!] \wedge [\![\varphi_2]\!].
\end{aligned}
$$

We relate this translation to the encoding of ternary relations by structures in Section 6.2.1, as follows:

Lemma 43. *Let $\varphi(X_1, \ldots, X_k)$ be a domain-independent relational calculus query that uses no constants and only one ternary relation name R. Let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ be a structure and let I be a finite ternary relation over the relation scheme R. We have the following:*

1. *for all pure constants $D_1, \ldots, D_n$ in σ, $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models [\![\varphi]\!](D_1, \ldots, D_k)$ if and only if $\mathsf{BiToTer}(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(D_1, \ldots, D_k)$, where R is considered to be the scheme of this ternary relation;*

2. *for all entries* $D_1, \ldots, D_n$ *in the active domain of* I, $I \models \varphi(D_1, \ldots, D_k)$ *if and only if* $\mathsf{TerToBi}(I) \models [\![\varphi]\!](D_1, \ldots, D_k)$.

Proof. 1. For relation atoms, the first statement follows from the definition of $\mathsf{BiToTer}(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$. For equalities, the first statement follows trivially, because $(X = Y)(D_1, D_2)$ holds if and only if $D_1 = D_2$ independent of the context in which it is evaluated. The remainder of the proof of this statement goes by a straightforward structual induction.

2. The second statement follows from the first by putting $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) = \mathsf{TerToBi}(I)$ and using that $\mathsf{BiToTer}(\mathsf{TerToBi}(I)) = I$. □

We are actually only interested in the Boolean case, which we obtain by putting $k = 0$ in Lemma 43, but we also had to include the case $k > 0$ in order to be able to use structural induction.

We are now able to prove the following:

Theorem 44. *It is undecidable whether a SyCALC query is counting-only.*

Proof. Let ψ be a domain-independent Boolean relational calculus query that uses no constants and only one ternary relation name R. Consider the SyCALC query $\{\langle z_2, z_3\rangle \mid \varphi(z_2, z_3)\}$ with

$$\varphi(z_2, z_3) := [\![\psi]\!] \wedge \exists x \exists Y_1 \exists Y_2 \exists Y_3 \exists z_1 \exists X \exists Y \exists Z \\ \mathit{First}(x, X) \wedge \mathit{ShortPath}(x, Y_1, z_1, X) \wedge \mathit{LongPath}(x, Y_2, z_2, Y_3, z_3, Z).$$

We now show that ψ has a nonempty model if and only if $\{\langle z_2, z_3\rangle \mid \varphi(z_2, z_3)\}$ is not counting-only. The desired result then follows, because the following problem is undecidable: given a domain-independent Boolean relational calculus query that uses no constants and only one ternary relation name, decide whether this has a nonempty model (since the unsatisfiability of such queries [24, Theorem 6.3.1 and Exercise 6.19] can be reduced straightforwardly to that problem).

To see this, first assume that ψ has a nonempty model. Hence, there exists a nonempty ternary relation I such that $I \models \psi$. By Lemma 43, $\mathsf{TerToBi}(I) \models [\![\psi]\!]$. Let $\langle A_i, B_i, C_i\rangle$ be any triple of I. By construction, there exist constants $\imath$, $\$i_1$, $\$i_2$, $\$i_3$, $\sharp i_1$, $\sharp i_2$, and $\sharp i_3$ such that

$$\{\langle \hat{\imath}, A\rangle\} \subseteq \mathsf{TerToBi}(I);$$
$$\{\langle \hat{\imath}, \$i_1\rangle, \langle \sharp i_1, \$i_1\rangle, \langle \sharp i_1, B\rangle\} \subseteq \mathsf{TerToBi}(I);$$
$$\{\langle \hat{\imath}, \$i_2\rangle, \langle \sharp i_2, \$i_2\rangle, \langle \sharp i_2, \$i_3\rangle, \langle \sharp i_3, \$i_3\rangle, \langle \sharp i_3, C\rangle\} \subseteq \mathsf{TerToBi}(I).$$

Hence, $\mathsf{TerToBi}(I) \models \varphi(\sharp i_2, \sharp i_3)$. The last inclusion above expresses that, in particular, $\mathsf{TerToBi}(I) \models \mathit{LongPath}(\hat{\imath}, \$i_2, \sharp i_2, \$i_3, \sharp i_3, C_i)$. From our analysis in Section 6.2.1, we may deduce that $\sharp i_2 \neq \sharp i_3$, and that, as a consequence, $\mathsf{TerToBi}(I) \not\models \mathit{LongPath}(\hat{\jmath}, \$j_2, \sharp i_3, \$j_3, \sharp i_2, C_j)$, irrespective of $\hat{\jmath}$, $\$j_2$, $\$j_3$, and C_j. Hence, $\mathsf{TerToBi}(I) \not\models \varphi(\sharp i_3, \sharp i_2)$. Now, if γ is the membership relation

of $\mathsf{TerToBi}(I)$, then $\mathsf{inc}(\sharp i_2, \gamma) = \mathsf{inc}(\sharp i_3, \gamma) = 2$. If $\{\langle z_2, z_3\rangle \mid \varphi(z_2, z_3)\}$ were counting-only, then, by Lemma 35, $\mathsf{TerToBi}(I) \models \varphi(\sharp i_2, \sharp i_3)$ if and only if $\mathsf{TerToBi}(I) \models \varphi(\sharp i_3, \sharp i_2)$. Hence, we must conclude that $\{\langle z_2, z_3\rangle \mid \varphi(z_2, z_3)\}$ is not counting-only.

Conversely, assume that ψ has no nonempty model. We show that $\{\langle z_2, z_3\rangle \mid \varphi(z_2, z_3)\}$ is unsatisfiable. Assume to the contrary that there exists a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ and objects $o_1, o_2 \in \mathcal{D}$ such that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_1, o_2)$. In particular, $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models [\![\psi]\!]$. By Lemma 43, $\mathsf{BiToTer}(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \psi$, if we assume that the scheme of this ternary relation is R. By construction, $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_1, o_2)$ implies that $\mathsf{BiToTer}(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ contains at least one tuple, implying that $\mathsf{BiToTer}(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ is a nonempty model of ψ, a contradiction. Hence, φ is unsatisfiable, from which it voidly follows that $\{\langle y, z\rangle \mid \varphi(y, z)\}$ is counting-only. □

Hence, Research Question 8 has a negative answer for SyCALC queries.

6.3. Deciding properties of symmetric queries

We first look at the decidability of the following decision properties for counting-only SyCALC queries.

Definition 45. A Boolean SyCALC query φ is *satisfiable* if it is satisfied by some structure and is *valid* if it is satisfied by all structures. A SyCALC query φ is *empty* if, for every structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, $\varphi(\mathcal{D}, \mathcal{S}, \sigma, \gamma) = \emptyset$. A SyCALC query φ is *contained* in a SyCALC query ψ if, for every structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, $\varphi(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \subseteq \psi(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$. SyCALC queries φ and ψ are *equivalent* if, for every structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, $\varphi(\mathcal{D}, \mathcal{S}, \sigma, \gamma) = \psi(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$.

In turns out that all these properties are decidable for counting-only SyCALC queries:

Theorem 46.
1. *Satisfiability is decidable for Boolean counting-only SyCALC queries;*
2. *Validity is decidable for Boolean counting-only SyCALC queries;*
3. *Emptiness is decidable for counting-only SyCALC queries;*
4. *Containment is decidable for counting-only SyCALC queries; and*
5. *Equivalence is decidable for counting-only SyCALC queries.*

Proof. If the quantifier depth of the counting-only SyCALC query (queries) involved in checking one the above properties is at most $q_{\mathcal{S}} \geq 0$, then, by Remark 37 on the constructive nature of the proof of Theorem 30, it suffices to check the property on (upon isomorphism) all structures with up to $2q_{\mathcal{S}} - 1$ set names and up to $2q_{\mathcal{S}} - 1$ active domain objects. □

Since QuineCALC queries are unary counting-only SyCALC queries, query containment and query equivalence is also decidable for them. Because of their unary nature, the definitions of satisfiability and validity in Definition 45 do not literally apply to them, but we can ask a very related question, which we can also answer in the positive:

Corollary 47. *Given a QuineCALC query $\varphi(x)$, it is decidable whether $\exists x\ \varphi(x)$ is satisfiable, respectively valid.*

Proof. If $\varphi(x)$ is a QuineCALC query, then, by Corollary 39, $\exists x\ \varphi(x)$ is a Boolean counting-only SyCALC query. □

So, it is fair to say that we have answered Research Question 9 in the positive for both counting-only SyCALC queries and QuineCALC queries.

Unfortunately, Research Question 9 has a negative answer for general SyCALC queries:

Theorem 48. *Satisfiability, emptiness, validity, containment, and equivalence are undecidable for Boolean SyCALC queries.*[29].

Proof. Using Lemma 43, it is straightforward to reduce satisfiability of a domain-independent Boolean relational calculus query over a single ternary relation and that does not use constants, which is undecidable [24, Theorem 6.3.1 and Exercise 6.19], to satisfiability of a Boolean SyCALC query. This problem can then be reduced straightforwardly to any of the other problems under consideration. □

7. Conclusions and future work

In this paper, we have introduced two query languages, QuineCALC and SyCALC, with the purpose of capturing symmetric queries over sequences of sets of objects. We have defined these languages in such a way that QuineCALC is a syntactic fragment of SyCALC. We have shown that QuineCALC queries correspond to symmetric functions specifiable by means of union, intersection, and complement, i.e., the symmetric Boolean functions of Quine [16], while SyCALC queries also capture projection and Cartesian product.

We have characterized QuineCALC queries in terms of incidence information of the objects involved, which is an important simplification in order to answer these queries. In general, this simplification is no longer possible for SyCALC queries. However, we have been able to characterize the class of SyCALC queries that can be answered using only incidence information as quantified Boolean combinations of QuineCALC queries. Unfortunately, it is undecidable whether a SyCALC query is such a counting-only query, but it is decidable whether a counting-only SyCALC query is equivalent to a QuineCALC query.

Reviewing both our original motivation to study symmetric queries and the theoretical results reported upon in this paper, we may thus conclude that,

[29] Notice that emptiness coincides with satisfiability for Boolean queries.

on the one hand, the class of symmetric queries is interesting to study from a practical, application-oriented, point of view and, on the other hand, that non-trivial foundational questions can be answered about this class. At the same time, however, we realize that our paper is just a first step in the study of symmetric queries, and leaves many problems unaddressed. Below, we list some of these.

1. *Extensions and restrictions.* Several extensions or restrictions of SyCALC are worthwhile to study:

   (a) Observe that in SyCALC we excluded the binary predicate "$x = y$" on domain variables. On the one hand, several results in this paper depend on that (in particular, Theorem 30 and Corollary 39 on counting-only SyCALC queries), but, on the other hand, adding this predicate would permit us to study symmetric queries that can be expressed in terms of the full relational algebra (including equality and inequality selection).

   (b) We could study extensions of SyCALC that incorporate aggregate functions. For example, the query "Find all pairs of students taking the same number of courses" is not expressible in SyCALC, but is clearly an interesting counting-only symmetric query.

   (c) It would also be interesting to characterize the monotonic (or anti-monotonic) fragments of the languages considered in this paper.

2. *Complexity and optimization problems.* In this paper, we did not study the efficiency of evaluating and optimizing symmetric queries. For example, we have algorithms to "normalize" QuineCALC (Corollary 16) and counting-only SyCALC queries (Theorem 30) into queries that only involve incidence predicates. We have not yet analyzed time or space complexity of these algorithms, however. In any case, these normal form algorithms are not effective translations of QuineCALC or SyCALC queries to queries in terms of incidence information, as they can cause a huge blow-up in the size of the query. So, one may ask if there is an effective translation of a SyCALC query to a query in terms of incidence information. What is the worst-case blow-up in the query size of such a translation?

   Another topic for further study is query optimization. For example, the counting-only SyCALC query $\{x \mid \mathsf{gteq}(x, 3) \wedge \neg\exists y \mathsf{gteq}(y, 3)\}$ can be optimized to $\{x \mid \mathbf{false}\}$.

3. *Extensions of the concept "counting-only".* If we consider the query "Retrieve the pairs of words that occur together in at least three documents," we cannot help but feel that it has the flavor of a counting-only SyCALC query, yet we can prove it is not. A strategy to study this query is to extend our notion of incidence information to pairs of objects. For a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, and $o_1, o_2 \in \mathcal{D}$, we can define

$$\mathsf{inc}_2(o_1, o_2, \gamma) = |\{S \mid \langle o_1, S\rangle \in \gamma \ \wedge \ \langle o_2, S\rangle \in \gamma\}|.$$

The above query actually searches for all pairs of objects (o_1, o_2) for which $\mathsf{inc}_2(o_1, o_2, \gamma) \geq 3$. Of course, this notion of 2-incidence can be generalized to k-incidence for any $k \geq 1$. We plan to investigate whether our current results about counting-only SyCALC queries can be extended for a broader notion of "counting-only" based on these more general notions of incidence information.

4. *Precomputation and indexes.* To evaluate efficiently QuineCALC and, more generally, counting-only SyCALC queries, we could precompute the incidence relation and maintain an index on it. For example, we could store and maintain an index that keeps pairs of the form $(i, \{o_1, \ldots, o_n\})$ where $\{o_1, \ldots, o_n\}$ is the set of all objects that occur in at least i sets. This could speed up evaluating symmetric queries that involve incidence predicates.

5. *Simulation.* Since SyCALC queries are first-order, it makes sense to ask how these queries may be simulated in SQL and MapReduce in a "smart" manner. This could well be very challenging, since (1) many interesting symmetric queries are non-monotonic and (2) the data sets involved can be very large.

References

References

[1] C. Gutierrez, C. A. Hurtado, A. O. Mendelzon, J. Pérez, Foundations of semantic web databases, J. Comput. Syst. Sci. 77 (3) (2011) 520–541.

[2] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of SPARQL, ACM Trans. Database Syst. 34 (3) (2009) 16.

[3] World Wide Web Consortium (W3C), RDF current status.
URL `http://www.w3.org/standards/techs/rdf#w3c_all`.

[4] R. Agrawal, T. Imielinski, A. N. Swami, Mining association rules between sets of items in large databases, in: P. Buneman, S. Jajodia (Eds.), Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 25–28, 1993, pp. 207–216.

[5] H.-C. Yang, A. Dasdan, R.-L. Hsiao, D. Stott Parker, Jr., Map-reduce-merge: simplified relational data processing on large clusters, in: C. Y. Chan, B. C. Ooi, A. Zhou (Eds.), Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12–14, 2007, pp. 1029–1040.

[6] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.

[7] M. T. Goodrich, N. Sitchinava, Q. Zhang, Sorting, searching, and simulation in the MapReduce framework, in: T. Asano, S.-I. Nakano, Y. Okamoto, O. Watanabe (Eds.), Proceedings 22nd International Symposium on Algorithms and Computation, ISAAC 2011, Yokohama, Japan, December 5–8, 2011, Vol. 7074 of Lecture Notes in Computer Science, Springer, 2011, pp. 374–383.

[8] R. Lämmel, Google's MapReduce programming model—revisited, Sci. Comput. Program. 70 (1) (2008) 1–30.

[9] B. Goethals, Survey on frequent pattern mining, Tech. Rep., University of Helsinki (2003).

[10] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach: 3rd Edition, Pearson, 2010.

[11] B. L. van der Waerden, Algebra: Volume I, Frederick Ungar Publishing Co., 1970.

[12] D. Mead, Newton's identities, Am. Math. Mon. 99 (8) (1992) 749–751.

[13] S. Lang, Linear Algebra: 3rd Edition, Springer, 1987.

[14] G. Audemard, B. Mazure, L. Sais, Dealing with symmetries in quantified Boolean formulas, in: 7th International Conference on Theory and Applications of Satisfiability Testing, SAT 2004, Vancouver, BC, Canada, Online Proceedings, 2004.

[15] A. Canteaut, M. Videau, Symmetric Boolean functions, IEEE Trans. Inf. Theory 51 (8) (2005) 2791–2811.

[16] W. V. Quine, Selected Logic Papers, Harvard University Press, 1995.

[17] S. J. Thomas, P. C. Fischer, Nested relational structures, Adv. Comput. Res. 3 (1986) 269–307.

[18] L. Wong, Normal forms and conservative properties for query languages over collection types, in: C. Beeri (Ed.), Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1994), Washington, DC, USA, May 25–28, 1993, pp. 26–36.

[19] S. Abiteboul, C. Beeri, The power of languages for the manipulation of complex values, VLDB J. 4 (4) (1995) 727–794.

[20] V. Sarathy, L. Saxton, D. Van Gucht, An object based algebra for parallel query processing and optimization, Tech. Rep. 368, Indiana University Computer Science (1992).

[21] M. Gyssens, J. Paredaens, D. Van Gucht, J. Wijsen, and Y. Wu, An approach towards the study of symmetric queries, Proceedings of the VLDB Endowment 7 (1) (2013) 25–36.

[22] J. Hellings, M. Gyssens, D. Van Gucht, Y. Wu, First-order definable counting-only queries, in: F. Ferrarotti, S. Woltran (Eds.), Foundations of Information and Knowledge Systems–10th International Symposium (FoIKS 2018), Budapest, Hungary, May 14–18, 2018, Proceedings. Lecture Notes in Computer Science 10833, Springer, 2018, pp. 225–243.

[23] A. K. Chandra, D. Harel, Computable queries for relational data bases, J. Comput. Syst. Sci. 21 (2) (1980) 156–178.

[24] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.