An Approach towards the Study of Symmetric Queries

Marc Gyssens
Hasselt University
Transnational Univ. of Limburg
Hasselt, Belgium
marc.gyssens@uhasselt.be

Jan Paredaens
University of Antwerp
Antwerp, Belgium
jan.paredaens@ua.ac.be

Dirk Van Gucht
Indiana University
Bloomington, Indiana, USA
vgucht@cs.indiana.edu

Jef Wijsen
University of Mons
Mons, Belgium
jef.wijsen@umons.ac.be

Yuqing Wu*
Indiana University
Bloomington, Indiana, USA
yuqwu@cs.indiana.edu

ABSTRACT

Many data-intensive applications have to query a database that involves sequences of sets of objects. It is not uncommon that the order of the sets in such a sequence does not affect the result of the query. Such queries are called *symmetric*. In this paper, the authors wish to initiate research on symmetric queries.

Thereto, a data model is proposed in which a binary relation between objects and set names encodes set membership. On this data model, two query languages are introduced, QuineCALC and SyCALC. They are correlated in a manner that is made precise with the symmetric Boolean functions of Quine, respectively symmetric relational functions, on sequences of sets of given length. The latter do not only involve the Boolean operations union, intersection, and complement, but also projection and Cartesian product. Quine's characterization of symmetric Boolean functions in terms of incidence information is generalized to QuineCALC queries. In the process, an incidence-based normal form for QuineCALC queries is proposed.

Inspired by these desirable incidence-related properties of QuineCALC queries, *counting-only* queries are introduced as SyCALC queries for which the result only depends on incidence information. Counting-only queries are then characterized as quantified Boolean combinations of QuineCALC queries, and a normal form is proposed for them as well. Finally, 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.

*Most of this work was carried out during a sabbatical visit of this author to Hasselt University with a senior visiting postdoctoral fellowship of the FWO Research Foundation – Flanders.

Proceedings of the VLDB Endowment, Vol. 7, No. 1

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 more 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, as is the case in the frequent-itemset problem [2].
- 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 [8, 12, 18].

In this article, we study computable queries $\mathbf{q}(S_1, \ldots, S_n)$ taking as input a sequence of sets $S_1, \ldots, S_n$, $n \geq 0$, of objects of some common type, and returning as output a set of m-tuples of such objects for some fixed value of $m \geq 0$, and, which, in addition, satisfy the following condition:

> 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 unary 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 areas listed above, 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 products that occur together in at least two transactions.
7. Retrieve the pairs of students taking the same courses.

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).

As a matter of fact, symmetric queries or functions (not necessarily on sequences of sets) abound in very diverse fields. For instance, symmetric functions are very prevalent in mathematics. As an example, symmetric polynomials play a fundamental role in finding roots of single-variable polynomials and finding solutions to systems of multi-variable polynomial equations[1] [16]. In linear algebra, functions such as those that determine the rank, determinant, and eigenvalues of a square matrix are invariant under permutations of rows, and of columns [10, 16]. There is also a comprehensive literature on symmetric Boolean functions (e.g., [3, 4, 13]). In statistics, most summary data are symmetrical functions of the input, such as sum, count, average, median, maximum, minimum, variance, and higher-order moments. In programming, examples of symmetric functions on lists of data include size, membership checking, and sorting.

With the current strong interest in cluster computing, data-parallel computation on partitioned data, data analytics, etc., one is interested in operators that are commutative and associative, and can therefore be ordered, grouped, combined, and merged arbitrarily. A good example of this is MapReduce, where it is commonly assumed that the reducer and combiner functions are symmetric [5, 6, 7, 9]. Typically, it is up to the programmer to guarantee that this property is satisfied. The strategy followed in this paper is to propose expressive query languages that guarantee this property implicity, and thus liberate the programmer from having to argue for it explicity.

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 as well. 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 [15] is an operator that when applied to a set of sets returns the union of these sets (see also, the "$\bigcup$" operator in NRC [17] and the "set-collapse" operator in the complex-object algebra [1]). Other examples of symmetric queries were introduced by Sarathy et al. [14], 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]

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

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

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

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

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

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

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

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 can start from the work of Quine [13], 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

[1]The study of symmetric polynomials in mathematics has a long history. For example, Isaac Newton already established fundamental results about symmetric polynomials [11].

[2]If S is a set, then $\pi_{\langle\rangle}(S) = \{\langle\rangle\}$ if $S \ne \emptyset$, 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; however, we do not want to impose domain independence at this stage, although we believe it is possible to deal with this issues pretty much in the same way as in the standard relational model. We chose, however, not to impose additional semantic and/or syntactic restrictions which could obfuscate the focus of this work.

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 $S_1, \ldots, S_n$ 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 property 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 products that occur together in at least two transactions," then knowing the number of transactions each product occurs in is not very helpful for answering this query. Similarly, knowing the number of courses each student takes is not very helpful for answering Query 7, "Retrieve the pairs of students taking the same courses."

In order to study the issues raised above more closely, we first want to do away with 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. First of all, the interesting setting is a dynamic one where new suppliers start up a business all the time and old ones go out of business. Second, 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 these limitations, we need a data model for representing sequences of sets of arbitrary length which allows defining query languages over that data model 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 and a binary membership relation γ. An object o belongs to a set of the sequence named S if and only if $S \in \sigma$ and $\langle o, S \rangle \in \gamma$. 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 lose of course 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 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 object variables and set variables cannot be compared. 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 regarding SyCALC.

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 the subclass of SyCALC queries that can be expressed in terms of incidence information the *counting-only queries*. Are there SyCALC queries that are not counting-only queries?
7. Is there a syntactically definable fragment of SyCALC that expresses precisely the counting-only queries?
8. Is it decidable if a SyCALC query is a counting-only query?
9. Is it decidable if a counting-only query is a QuineCALC query?

In this paper, we show that the answer to each of these research questions is "yes," except for Research Question 8, for which the answer is "no."

This paper is organized as follows. In Section 2, 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 3, 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 4, we introduce SyCALC, and establish a correspondence between SyCALC queries and symmetric relational functions. We also introduce counting-only queries, which we characterize as quantified Boolean combinations of QuineCALC queries. We show that, while it is undecidable whether a SyCALC query is

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

counting-only, it is decidable whether a counting-only query is equivalent to a QuineCALC query. Finally, in Section 5, we formulate some conclusions, and discuss directions for future research.

2. 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. From now on, we shall always implicitly assume that every object under consideration is in $\mathcal{D}$, and that every set name is in $\mathcal{S}$. (In particular, the set corresponding to such a set name is assumed to consist of objects in $\mathcal{D}$.) Even though in practical examples we may expect that $\mathcal{D} \cap \mathcal{S} = \emptyset$, there is no need to make that assumption explicitly because we shall never compare objects and set names.

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, and γ is a finite subset of $\mathcal{D} \times \sigma$, providing set membership information. For $o \in \mathcal{D}$, we define the *incidence* of o in γ as $\mathsf{inc}(o, \gamma) = |\{S \mid \langle o, S\rangle \in \gamma\}|$, i.e., the number of named sets to which o belongs.[6]

To capture better the semantics of our data model, we introduce the following definition. Let $S_1, \ldots, S_n$ be a sequence of named sets.[7] Then the *encoding* of $S_1, \ldots, S_n$, denoted $\mathsf{enc}(S_1, \ldots, S_n)$, 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 $\mathsf{enc}(S_{i_1}, \ldots, S_{i_n}) = \mathsf{enc}(S_1, \ldots, S_n)$. Conversely, every structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ is the encoding of a finite sequence of named sets (and of all its permutations). If we denote by $\mathsf{inc}(o, S_1, \ldots, S_n)$ the incidence of o in $S_1, \ldots, S_n$, i.e., the number of sets in the sequence $S_1, \ldots, S_n$ to which o belongs, then, clearly, $\mathsf{inc}(o, S_1, \ldots, S_n) = \mathsf{inc}(o, \gamma)$, justifying the use of that notation also in this context.

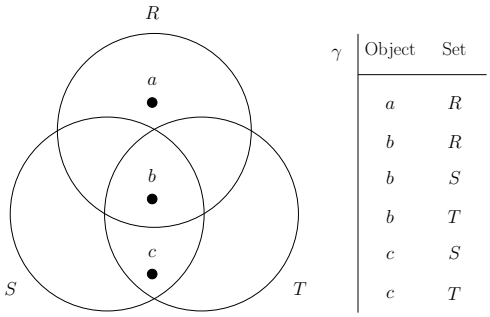
Example 1. Consider set names R, S, T, and U. The sets corresponding to R, S, and T are visualized by the Venn diagram in Figure 1, *left*. (Elements of $\mathcal{D}$ not in R, S, or T are not shown.) Furthermore, we assume that the set corresponding to U is empty. The sequence R, S, T, U (or any of the fifteen other permutations thereof) is represented 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 1, *right*.

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 more restricted, "static" level

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

[7]As we have argued in the Introduction, it is important to emphasize that $S_1, \ldots, S_n$ are the *names* of the sets under consideration, as it may well be that, for $1 \leq i \neq j \leq n$, S_i and S_j represent the same set of objects. For simplicity of notation, however, we shall not distinguish between set names and the sets they represent. Hence, a statement such as $S_2 = \{o_1, o_2, o_5\}$ must be read as "the set named S_2 consists of the objects o_1, o_2, and o_5." Similarly, $o_1 \in S_2$ must be read as "object o_1 belongs to the set named S_2."



γ	Object	Set
	a	R
	b	R
	b	S
	b	T
	c	S
	c	T

Figure 1: Encoding of a finite sequence of named sets by a binary membership relation.

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, 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 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 $m \geq 0$.

Both concepts are of course closely interconnected.

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

$$f_{\mathbf{q},n}(S_1, \ldots, S_n) := \mathbf{q}(\mathsf{enc}(S_1, \ldots, S_n)),$$

for all sequences of n named sets $S_1, \ldots, S_n$. By construction, this function is 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 $F = \{f_0, f_1, f_2, \ldots\}$ of symmetric functions such that f_n, $n \geq 0$, operates on sequences of n sets (all of which produce output of the same arity). Then, we can associate with F a query $\mathbf{q}_F$, as follows. Let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ be a structure, with $\sigma = \{S_1, \ldots, S_n\}$. Associate sets to $S_1, \ldots, S_n$ such that $\mathsf{enc}(S_1, \ldots, S_n) = (\mathcal{D}, \mathcal{S}, \sigma, \gamma)$. Then, $\mathbf{q}_F(\mathcal{D}, \mathcal{S}, \sigma, \gamma) := f_n(S_1, \ldots, S_n)$. The well-definedness of $\mathbf{q}_F$ relies on the symmetry of $f_0, f_1, f_2, \ldots$. Clearly, $f_{\mathbf{q}_F,n} = f_n$. Of course, the query $\mathbf{q}_F$ will not be very meaningful from a practical point of view unless the symmetric functions of the family F are closely related.

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.

To conclude these Preliminaries, we point out that, while we will establish interconnections between particular classes of queries and particular classes of symmetric functions on sequences of sets, the main focus in this study is on queries.

3. QUINECALC

Rather than first defining SyCALC, and then identifying a syntactically definable fragment of it that is a conservative extension of the symmetric Boolean functions in the sense of Quine (cf. Research Question 1), we first define a first-order language, called QuineCALC, of which we show that it is indeed a conservative extension of the symmetric Boolean functions in the sense of Quine. Later, in Section 4, we will extend QuineCALC to SyCALC, the language which is at the core of this study.

3.1 Language definition

QuineCALC is a restricted first-order logic with a single binary relation name Γ for 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. Intuitively, lowercase variables denote objects, and uppercase variables denote sets. The alphabet contains no constant symbols. QuineCALC *formulae* are defined by the following syntax rule:[8]

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

Observe that the (in)equality predicate and existential quantification cannot be applied to lowercase variables.

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.[9]

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. 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 2. The QuineCALC query

$$\big\{x \mid \exists X \exists Y \big(\Gamma(x, X) \wedge \Gamma(x, Y) \wedge X \neq Y\big)\big\}$$

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 3. 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

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

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[10], which is expressed by the QuineCALC query

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

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[11] 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 $\mathsf{coeq}(x, i)$.

3.2 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 1. 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 named sets $S_1, \ldots, S_n$ and for all objects $o \in \mathcal{D}$, we have that $o \in f(S_1, \ldots, S_n)$ 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 2. 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 1. *For every* QuineCALC *query* $\mathbf{q}$*, and for every natural number* $n \geq 0$*, there exists a symmetric Boolean function* $f_{\mathbf{q},n}$ *operating on sequences of* n *sets such that* $\mathbf{q} \equiv_n f_{\mathbf{q},n}$.

[8] For convenience, we allowed some redundance in this rule.

[9] Observe that all occurrences of lowercase variables in a QuineCALC formula must be free, since the language has no quantification over lowercase variables.

[10] Or, equivalently, the objects that belong to at most $n - i$ sets of σ, with $n = |\sigma|$.

[11] Or, equivalently, the objects that belong to *exactly* $n - i$ sets of σ, with $n = |\sigma|$.

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

$$\mathsf{qe}(\Gamma(x, S_i)) = \Gamma(x, S_i);$$

$$\mathsf{qe}(S_i = S_j) = \begin{cases} \mathbf{true} & \text{if } i = j, \\ \mathbf{false} & \text{if } i \neq j; \end{cases}$$

$$\mathsf{qe}(S_i \neq S_j) = \begin{cases} \mathbf{true} & \text{if } i \neq j, \\ \mathbf{false} & \text{if } i = j; \end{cases}$$

$$\mathsf{qe}(\varphi_1 \wedge \varphi_2) = \mathsf{qe}(\varphi_1) \wedge \mathsf{qe}(\varphi_2);$$

$$\mathsf{qe}(\varphi_1 \vee \varphi_2) = \mathsf{qe}(\varphi_1) \vee \mathsf{qe}(\varphi_2);$$

$$\mathsf{qe}(\neg\varphi_1) = \neg\mathsf{qe}(\varphi_1);$$

$$\mathsf{qe}(\exists X \varphi_1) = \begin{cases} \mathbf{false} & \text{if } n = 0, \\ \bigvee_{1 \leq i \leq n} \mathsf{qe}(\varphi_1[X \to S_i]) & \text{if } n > 0. \end{cases}$$

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.

Although not essential for the proof, we implicitly assume that some obvious simplications are made when applying these rules. For instance, if $\mathsf{qe}(\varphi_1) \equiv \mathbf{true}$, then $\mathsf{qe}(\varphi_1 \wedge \varphi_2)$ may simply be replaced by $\mathsf{qe}(\varphi_2)$, $\mathsf{qe}(\varphi_1 \vee \varphi_2)$ may simply be replaced by **true**, and $\mathsf{qe}(\neg\varphi_1)$ may simply be replaced by **false**. Similarly, if X does not occur explicitly as a free variable in φ_1 and $n > 0$, then $\mathsf{qe}(\exists X \varphi_1)$ may simply be replaced by $\mathsf{qe}(\varphi_1)$.

We next compute $\mathsf{fun}(\mathsf{qe}(\varphi))$ as follows[12], where $1 \leq i \leq n$:

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

Also in this stage of the translation, we implicitly assume that some obvious simplifications are made.

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 4. Consider the QuineCALC queries in Example 2, 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, the second theorem explains how to translate symmetric Boolean functions on sequences of n sets into QuineCALC queries.

THEOREM 2. *For every natural number $n \geq 0$ and for every symmetric Boolean function $f_n(S_1, \ldots, S_n)$, there exists a QuineCALC query $\mathbf{q}_{f_n}$ such that $\mathbf{q}_{f_n} \equiv_n f_n$.*

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

PROOF. The proof of Theorem 2 relies on the following theorem by Quine [13, p. 178] (slightly adapated to our notations and terminology).

> *Property 1.* Let $n \geq 0$. For a Boolean function f on sequences of n sets of objects and returning a set of these objects, the following statements are equivalent:
>
> 1. f is symmetric;
> 2. there exists a 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 $\mathsf{inc}(o, S_1, \ldots, S_n) \in N$.

Thus, let N be the set of natural numbers characterizing the symmetric Boolean function f_n in the statement of Theorem 2 in the sense of Property 1. 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 5. We revisit Example 4. 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 Property 1 equals $\{2, 3\}$. Hence, by the proof of Theorem 2, we have that $\mathbf{q}_{f_3} \equiv_3 f_3$, with

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

The QuineCALC query in Example 2 from which f_3 was derived in Example 4 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.

For the other symmetric Boolean function in Example 4, $g_3(S_1, S_2, S_3) = S_1 \cap S_2 \cap S_3$, 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 2 from which g_3 was derived in Example 4 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.

Theorems 1 and 2 together settle Research Question 1: QuineCALC (which will turn out to be a syntactically definable fragment of SyCALC in Section 4) is a conservative extension of the fixed-arity symmetric Boolean functions.

From Theorem 1 and Property 1 in the proof of Theorem 2, we can immediately derive the following corollary.

COROLLARY 1. *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 $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \varphi(o_2)$.*

3.3 QuineCALC and counting

In Section 3.2, 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 the following theorem.

THEOREM 3. *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*[13] *of subformulae of the form* $\mathit{eq}(x,i)$ *(*$0 \leq i < q$*), subformulae of the form* $\mathit{coeq}(x,j)$ *(*$0 \leq j < q$*), and at most one subformula of the form* $\mathit{gteq}(x,i) \wedge \mathit{cogteq}(x,j)$ *(*$0 \leq i,j \leq q$*), such that, for all* $n \geq 2q-1$, $\mathbf{q}$ *is* n*-equivalent to* $\mathbf{q}_{\text{inc}}$.

PROOF. (Sketch.) Subformulae from which QuineCALC formulae are built can be written as $\varphi(x, X_1, \ldots, X_r)$, $r \geq 0$. (Even if x does not occur explicitly, such as in subformulae of the form $X = Y$ or $X \neq Y$, we can still assume that x is an unconstrained implicit variable in this formula.) We can prove by structural induction (details omitted) that, under the assumption in the statement of the Theorem, $\varphi(x, X_1, \ldots, X_r)$ can be rewritten as

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

with

1. $m \geq 0$; and
2. for $i = 1, \ldots, m$,

$$\varphi_i = \varrho_i \wedge c_i \wedge {}^{s_{i_1}}\Gamma(x, X_1) \wedge \cdots \wedge {}^{s_{i_r}}\Gamma(x, X_r).$$

Here,[14]

(a) $\varrho_i = \bigwedge_{1 \leq j < k \leq r} X_j \; \theta_{ijk} \; X_k$, where "$\theta_{ijk}$" is either "$=$" or "$\neq$;"

(b) c_i is either $\mathsf{eq}(x, p_i)$ $(0 \leq p_i < q)$, $\mathsf{coeq}(x, n_i)$ $(0 \leq n_i < q)$, or $\mathsf{gteq}(x, p_i) \wedge \mathsf{cogteq}(x, n_i)$ $(0 \leq p_i, n_i \leq q)$.

(c) for $j = 1, \ldots, r$, "${}^{s_{i_j}}\Gamma(x, X_j)$" is either "$\Gamma(x, X_j)$" or "$\neg\Gamma(x, X_j)$."

The result then follows immediately by considering $\varphi(x)$ as a subformula of itself. □

We already know from Quine's results that we can express a QuineCALC query in terms of incidence information provided we only consider structures for a given size n of σ. What Theorem 3 adds is that this can be done uniformly so from a certain minimal value of n onward defined as one less than twice the quantifier depth. The following example shows that the bound is tight.

Example 6. Consider the QuineCALC query

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

In words, this query returns all objects such that 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 3 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 $\mathsf{eq}(x, 1) \vee \mathsf{eq}(x, 2)$; and for $n \leq 2$, the query is equivalent to **true**, which can be written as $\mathsf{gteq}(x, 0) \wedge \mathsf{cogteq}(x, 0)$.

Where in the proof of Theorem 3 does the lowerbound $2q-1$ for n arise? If, in the construction, it is required to consider the conjunction of a subexpression $\mathsf{eq}(x, p_i)$ $(0 \leq p_i < q)$ and a subexpression $\mathsf{coeq}(x, n_i)$ $(0 \leq n_i < q)$, then this conjunction evaluates to **false** if $n \geq 2q - 1$. If n is smaller, however, it may be that $p_i = n - n_i$, for example, if $q = 3$, $n = 4$, and $p_i = n_i = 2$ (cf. Example 6 above). Depending on the precise values of p_i and n_i in the conjunctions that must be considered, it may sometimes be possible to decrease the lowerbound of $2q - 1$. In the extreme case where no such conjuctions occur, 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 $\{x \mid \mathsf{coeq}(x, 0)\}$.

From Theorem 3 and Quine's results, we can derive the following Corollary.

COROLLARY 2. *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*

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

where

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

Example 7. Consider the QuineCALC query of Example 6. By Corollary 2, and after applying some straightforward simplifications, this query is equivalent to the QuineCALC query expressed by the formula

$$\begin{aligned}&\mathsf{Eq}(0) \vee \mathsf{Eq}(1) \vee \mathsf{Eq}(2) \vee \\ &\quad (\mathsf{Eq}(3) \wedge (\mathsf{eq}(x, 1) \vee \mathsf{eq}(x, 2))) \vee (\mathsf{Eq}(4) \wedge \mathsf{eq}(x, 2)).\end{aligned}$$

As the characterization result of Corollary 2 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.

[13] An empty disjunction is interpreted as "**false**."
[14] An empty conjunction is interpreted as "**true**."

4. SYCALC

As announced in the opening paragraph of Section 3, we will now extend QuineCALC to SyCALC, the language which is at the core of this study.

4.1 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:

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

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 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**.

The semantics of SyCALC is analogous to the semantics of QuineCALC. 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, $1 \leq i \leq m$.[15] 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 8. 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) $\{\langle x, y\rangle \mid (\exists X\Gamma(x, X)) \wedge (\exists X\Gamma(y, X)) \wedge \neg\exists X(\Gamma(x, X) \wedge \neg\Gamma(y, X)) \wedge \neg\exists X(\neg\Gamma(x, X) \wedge \Gamma(y, X))\}$

respectively express Queries 3–7 in the Introduction.

Example 9. Let $r \geq 0$, and let $\mathsf{Gteq}(r)$ and $\mathsf{Eq}(r)$ be the expressions described in the statement of Corollary 2. The queries $\{\langle\rangle \mid \mathsf{Gteq}(r)\}$ and $\{\langle\rangle \mid \mathsf{Eq}(r)\}$ are SyCALC queries that, upon input a structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$, return whether $n = |\sigma| \geq r$, respectively whether $n = |\sigma| = r$.

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

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 10. Consider the SyCALC query in Example 8 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 2. Hence, this query is *not* equivalent to a QuineCALC query.

4.2 SyCALC and symmetric relational functions

In order to solve Research Question 3, we extend Theorems 1 and 2 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 $\mathcal{D}$. Notice that sets of objects in $\mathcal{D}$ can be interpreted as unary relations. To emphasize the distinction, we shall refer to such functions as *(symmetric) relational functions*.

Definition 3. 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.[16]

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 4. 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 named 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 $\mathsf{enc}(S_1, \ldots, S_n) \models \varphi(o_1, \ldots, o_m)$.

We can now generalize Theorem 1.

THEOREM 4. *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 1. 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.

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

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 1, we retain the rules

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

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

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

In the last rule, τ is a permutation of $\{1, \ldots, r\}$. We use this rule to reorder the variables whenever needed to apply the rules before. 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 11. Consider the SyCALC queries in Example 8, 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 2 to SyCALC queries.

THEOREM 5. *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, 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 5 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 consider

$$C'(x_1, \ldots, x_m, X_1, \ldots, X_n),$$

which is $C(x_1, \ldots, x_m)$ in which each atomic subexpression "$x_i \in S_j$" is substituted by "$\Gamma(x_i, X_j)$." Finally, define $\varphi(x_1, \ldots, x_m)$ as

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

Then, the expression $\mathsf{qe}(\varphi)$ computed in the proof of Theorem 4 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 4, $\Gamma(x_i, S_j)$ is translated into S_j. Hence, we may conclude that the expression $\mathsf{fun}(\mathsf{qe}(\varphi))$ is in fact 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 $f_n(S_1, \ldots, S_n)$. It follows that $\varphi \equiv_n f_n$. □

Example 12. We revisit Example 11. As a first example, consider the symmetric relational function $f_3(S_1, S_2, S_3) = \pi_{\langle\rangle}((S_1 \cap S_2) \cup (S_1 \cap S_3) \cup (S_2 \cap S_3))$. If we apply the construction in the proof of Theorem 5 to this relational function, we obtain the SyCALC query

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

which, on structures with $n \geq 3$, can be simplified to the SyCALC query in Example 8 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 5 to this relational function, we obtain the SyCALC query

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

which, on structures with $n \geq 3$, can be simplified to the SyCALC query in Example 8 expressing Query 6. So, both queries are 3-equivalent, and hence also 3-equivalent to f_3.

Theorems 4 and 5 together settle Research Question 3.

4.3 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 2, QuineCALC queries cannot distinguish between incidence-equivalent structures. This is no longer true for SyCALC queries, however.

Example 13. Consider the SyCALC query in Example 8 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

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

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 define counting-only queries as SyCALC queries that cannot distinguish between incidence-equivalent structures.

Definition 5. 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 *counting-only* 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 2, all QuineCALC queries are counting-only. There are, however, many counting-only SyCALC queries that are *not* equivalent to a QuineCALC query.

Example 14. Consider the SyCALC queries in Example 8.

The SyCALC query expressing Query 3 in the Introduction returns **true** on a 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 10.

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.

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 $\gamma_1 = \{\langle o_1, S_1\rangle, \langle o_1, S_2\rangle, \langle o_2, S_1\rangle, \langle o_2, S_2\rangle, \langle o_3, S_3\rangle\}$, and $\gamma_2 = \{\langle o_1, S_1\rangle, \langle o_1, S_2\rangle, \langle o_2, S_1\rangle, \langle o_2, S_3\rangle, \langle o_3, S_2\rangle\}$. 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 $\mathsf{inc}(o_3, \gamma_1) = \mathsf{inc}(o_3, \gamma_2) = 1$, yet 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 it is *not* counting-only.

Finally, the SyCALC query expressing Query 7 is not counting-only either, as shown in Example 13.

With Example 14, Research Questions 4, 5, and 6 have been answered in the affirmative.

Definition 5 is in our opinion a very compelling, intuitive semantic definition of counting-only 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 2 for QuineCALC queries.

THEOREM 6. *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 the* 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*

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

where

- *for* $n = 1, \ldots, 2q_{\mathcal{S}} - 2$, $\psi_n(x_1, \ldots, x_m)$ *is a disjunction of formulae of the form* $\vartheta_1 \wedge \ldots \wedge \vartheta_n \wedge \alpha_1(x_1) \wedge \ldots \wedge \alpha_m(x_m)$*, with*
  - *for* $i = 1, \ldots, n$, ϑ_i *is* $\exists x \mathit{eq}(x, i)$ *or* $\neg\exists x \mathit{eq}(x, i)$*;*
  - *for* $j = 1, \ldots, m$, $\alpha_j(x_j)$ *is of the form* $\mathit{eq}(x_j, k_j)$*, with* $0 \leq k_j \leq n$*; and*
- $\psi(x_1, \ldots, x_m)$ *is a disjunction of formulae of the form*

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

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

PROOF. (Sketch.) Let $(\mathcal{D}, \mathcal{S}, \sigma, \gamma)$ be a structure and $\vec{o} := o_1, \ldots, o_m \in \mathcal{D}$ such that $(\mathcal{D}, \mathcal{S}, \sigma, \gamma) \models \vec{o}$. We construct a SyCALC formula $\varphi_{\sigma,\gamma,\vec{o}}$ describing the incidence information contained herein. 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 formula:
   - 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 formula:
   - 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 $\ell = 1, \ldots, m$, $\alpha_\ell(x_\ell)$ equals

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

Let $(\mathcal{D}, \mathcal{S}, \sigma', \gamma')$ be a structure and $\vec{o}' = o'_1, \ldots, o'_m \in \mathcal{D}$ a sequence of m objects such that $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi_{\sigma,\gamma,\vec{o}}(\vec{o}')$. It can be shown that $(\mathcal{D}, \mathcal{S}, \sigma', \gamma') \models \varphi(\vec{o}')$ (details omitted). Hence, the counting-only query $\mathbf{q}$ is equivalent to $\mathbf{q}' := \{\langle x_1, \ldots, x_m\rangle \mid \varphi'(x_1, \ldots, x_m)\}$ with φ' equal to

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

Inspection of the formulae $\varphi_{\sigma,\gamma,\vec{o}}(x_1, \ldots, x_m)$ reveals that there are only finitely many different ones in this seemingly infinite disjuction, and that they are of the form described in the statement of this Theorem. □

Example 15. As shown in Example 14, the SyCALC queries in Example 8 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 6, 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 7 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 6 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 6.

COROLLARY 3. *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.*

Theorem 6 and Corollary 3 also provide a positive answer to Research Question 7.

Unfortunately, it is undecidable whether a given SyCALC query is counting-only (proof omitted):

THEOREM 7. *It is undecidable whether a* SyCALC *query is counting-only.*

If we know, however, that a SyCALC query is counting-only, we can decide if it is equivalent to a QuineCALC query.

THEOREM 8. *It is decidable whether a counting-only* SyCALC *query is equivalent to a* QuineCALC *query.*

PROOF. (Sketch.) The proof is based on the one hand on Theorem 6, describing a normal form for counting-only SyCALC queries, and on the other hand on Corollary 2, describing a normal form for QuineCALC queries. Thus, let $\mathbf{q} := \{x \mid \varphi(x)\}$ be a counting-only SyCALC query returning unary output, and having quantifier depth $q_{\mathcal{S}}$ in the uppercase variables. For each $n = 1, \ldots, 2q_{\mathcal{S}} - 1$, let $S_1, \ldots, S_n$ be different set names in $\mathcal{S}$, and consider the 2^{n+1} subsets N of $\{0, \ldots, n\}$. For each subset N, let o_i, $i \in N$, be different objects in $\mathcal{D}$, and consider the structure $(\mathcal{D}, \mathcal{S}, \sigma, \gamma_N)$ where $\sigma = \{S_1, \ldots, S_n\}$ and $\gamma_N = \bigcup_{i\in N}\{(o_i, S_1), \ldots, (o_i, S_i)\}$.[17] Define

$$K_{n,N} = \{k \in N \mid (\mathcal{D}, \mathcal{S}, \sigma, \gamma_N) \models \varphi(o_k)\}.$$

Then, $\mathbf{q}$ is equivalent to a QuineCALC query if and only if, for all $n = 1, \ldots, 2q_{\mathcal{S}} - 1$, and for all $N_1, N_2 \subseteq \{0, \ldots, n\}$, $K_{n,N_1} \cap N_2 = K_{n,N_2} \cap N_1$. □

Hence, while Research Question 8 has a negative answer, Research Question 9 has a positive answer.

5. 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 [13], 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 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, on the one hand, the class of symmetrical 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 worth-while 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 dependend on that (in particular, Theorem 6 and Corrollary 3 on counting-only 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).

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

(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 symmetric query.

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

2. *Other decision problems.* We think of satisfiability, equivalence, ... For example, it would be interesting to determine whether the equivalence problem for counting-only queries is decidable. We conjecture that this is the case.

3. *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 and counting-only SyCALC queries into queries that only involve incidence predicates. We have not analyzed the time and/or space complexity of these algorithms, however. Another topic for further study is query optimization. For example, the counting-only query $\{x \mid \mathsf{gteq}(x,3) \wedge \neg\exists y \mathsf{gteq}(y,3)\}$ can be optimized to $\{x \mid \mathbf{false}\}$.

4. *Extensions of the concept of counting-only queries.* 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 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

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

The above query actually searches for all pairs (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 if our current results about counting-only queries can be extended for a broader notion of "counting-only" queries based on these more general notions of incidence information.

5. *Precomputation and indexes.* To evaluate QuineCALC and, more generally, counting-only queries, efficiently, 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.

6. *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.

6. REFERENCES

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

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *SIGMOD Conference*, pages 207–216. ACM, 1993.

[3] G. Audemard, B. Mazure, and L. Sais. Dealing with symmetries in quantified Boolean formulas. In *SAT*, 2004.

[4] A. Canteaut and M. Videau. Symmetric Boolean functions. *IEEE Transactions on Information Theory*, 51(8):2791–2811, 2005.

[5] H. Chih Yang, A. Dasdan, R.-L. Hsiao, and D. S. P. Jr. Map-Reduce-Merge: simplified relational data processing on large clusters. In C. Y. Chan, B. C. Ooi, and A. Zhou, editors, *SIGMOD Conference*, pages 1029–1040. ACM, 2007.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150. USENIX Association, 2004.

[7] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the MapReduce framework. In T. Asano, S.-I. Nakano, Y. Okamoto, and O. Watanabe, editors, *ISAAC*, volume 7074 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2011.

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

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

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

[11] D. Mead. Newton's identities. *The American Mathematical Monthly*, 99(8):749–751, 1992.

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

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

[14] V. Sarathy, L. Saxton, and D. Van Gucht. An object based algebra for parallel query processing and optimization. Technical Report 368, Indiana University Computer Science, 1992.

[15] S. J. Thomas and P. C. Fischer. Nested relational structures. *Advances in Computing Research*, 3:269–307, 1986.

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

[17] L. Wong. Normal forms and conservative properties for query languages over collection types. In C. Beeri, editor, *PODS*, pages 26–36. ACM, 1993.

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