Annals of Mathematics and Artificial Intelligence (2019) 87:109–136
https://doi.org/10.1007/s10472-019-09652-8

First-order definable counting-only queries

Jelle Hellings[1,2] · Marc Gyssens[2] · Dirk Van Gucht[3] · Yuqing Wu[4]

Published online: 13 July 2019

Abstract

Many data sources can be represented easily by collections of sets of objects. For several practical queries on such collections of sets of objects, the answer does not depend on the precise composition of these sets, but only on the *number* of sets to which each object belongs. This is the case k=1 for the more general situation where the query answer only depends on the number of sets to which each *collection* of at most k objects belongs. We call such queries k-counting-only. Here, we focus on k-SyCALC, i.e., k-counting-only queries that are first-order definable. As k-SyCALC is semantically defined, however, it is not surprising that it is already undecidable whether a first-order query is in 1-SyCALC. Therefore, we introduce SimpleCALC-k, a syntactically defined (strict) fragment of k-SyCALC. It turns out that many practical queries in k-SyCALC can already be expressed in SimpleCALC-k. We also define the query language GCount-k, which expresses counting-only queries directly by using generalized counting terms, and show that this language is equivalent to SimpleCALC-k. We prove that the k-counting-only queries form a non-collapsing hierarchy: for every k, there exist (k+1)-counting-only queries that are not k-counting-only. This result specializes to both SimpleCALC-k and k-SyCALC. Finally, we establish a strong dichotomy between 1-SyCALC and SimpleCALC-k on the one hand and 2-SyCALC on the other hand by showing that satisfiability, validity, query containment, and query equivalence are decidable for the former two languages, but not for the latter one.

Keywords Bag of sets · Counting-only query · First-order definable query · Satisfiability

Mathematics Subject Classification (2010) 03C07 · 03C80 · 03D15 · 03D55 · 68P15

1 Introduction

Often, (parts of) queries can be viewed as operating on transaction databases [9], bipartite graphs, binary many-to-many relations, or, equivalently, on bag of sets. As an example,

This material is based on work supported by the National Science Foundation under Grant No. NSF 1438990.

✉ Marc Gyssens
marc.gyssens@uhasselt.be

Extended author information available on the last page of the article.

consider the bag-of-sets dataset of Fig. 1, *left*, in which each set represents a course and contains the students taking that course. This bag of sets can alternatively be interpreted as the *bipartite graph*, shown in Fig. 1, *right*.

Many practical queries on bags of sets turn out to be *counting-only*: in order to answer them, it is not necessary to know to which sets each object belongs, but only to *how many* sets each object belongs. As examples, consider the queries 'return students who take at least 2 courses', expressed by

$$\mathsf{Q}_1 = \{\langle x \rangle \mid \mathrm{count}(x) \geq 2\},$$

and 'return pairs of students who take the same number of courses', expressed by

$$\mathsf{Q}_2 = \{\langle x, y \rangle \mid (x \neq y) \wedge \mathrm{count}(x) = \mathrm{count}(y)\}.$$

In the above expressions, "count(·)" counts the number of sets (here, courses) to which the collection of arguments (here, a single student) belongs. Clearly, one need not know which courses each student takes to answer Q_1 or Q_2, but only *how many* courses each student takes. Next, consider the queries 'return pairs of distinct students which take a common course', expressed by

$$\mathsf{Q}_3 = \{\langle x, y \rangle \mid (x \neq y) \wedge \mathrm{count}(x, y) \geq 1\},$$

and 'return pairs of distinct students which take the same courses', expressed by

$$\mathsf{Q}_4 = \{\langle x, y \rangle \mid (x \neq y) \wedge \mathrm{count}(x, y) = \mathrm{count}(x) \wedge \mathrm{count}(x, y) = \mathrm{count}(y)\}.$$

Notice that Q_3 is a basic intersection query and Q_4 is a basic equivalence query. Both can be answered by counting not only *(i)* how many courses each student takes, but also *(ii)* how many courses each *pair* of students share. For $k \geq 0$, we call a query *k-counting-only* if it can be answered by only counting to how many sets each *collection* of at most k objects belongs. Hence, Q_1 and Q_2 are 1-counting-only, while Q_3 and Q_4 are 2-counting-only. Similarly, the Boolean query 'does there exist a course taken by 3 students', expressed by

$$\mathsf{Q}_5 = \{\langle\rangle \mid \exists x \exists y \exists z\ ((x \neq y \wedge x \neq z \wedge y \neq z) \wedge \mathrm{count}(x, y, z) \geq 1)\},$$

is 3-counting-only. In contrast, the Boolean query 'there are at least 3 courses', expressed by

$$\mathsf{Q}_6 = \{\langle\rangle \mid \mathrm{count}() \geq 3\},$$

can already be answered at the scheme level, and is therefore 0-counting-only.

Observe that the counting-only queries Q_3 and Q_4 only differ in the use of the *generalized quantifiers* 'takes some' versus 'takes all and only'. Similar familiar families of counting-only queries can be formulated using other generalized quantifiers such as 'takes only', 'takes all', 'takes no', 'takes at least k', and 'takes all but k'. Such queries are not only of relevance in the study of generalized quantifiers [4, 21, 26], but also play an obvious central role in the *frequent itemset problem* [9]. In essence, bag-of-set-like data models and

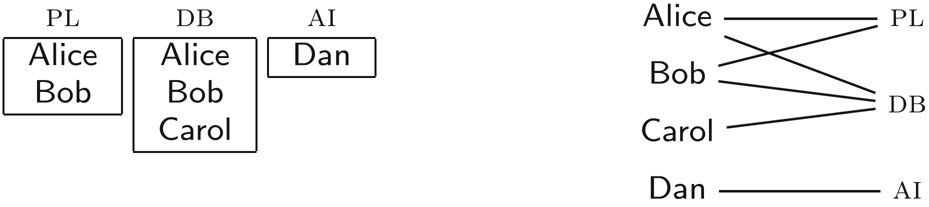


Fig. 1 *Left*, a bag-of-sets dataset. *Right*, the same dataset represented as a bipartite graph

counting-only queries can also be found in the *differential constraints* of Sayrafi et al. [23], *citation analysis and bibliometrics* [6], the *symmetric Boolean functions* of Quine [13, 22], *finite set combinatorics* [2], and the *data spaces* of Fletcher et al. [8], either explicitly or implicitly.

A more formal way to capture the notion of k-counting-only query is that such queries cannot distinguish between bags of sets which share the same up-to-k counting information. Consider, e.g., the bags of sets $\mathbf{S}_1$ and $\mathbf{S}_2$ of Fig. 2. Clearly, $\mathbf{S}_1$ and $\mathbf{S}_2$ agree on all up-to-2 counting information, but disagree on count(Alice, Bob, Carol). Hence, $\mathtt{Q}_1$–$\mathtt{Q}_4$ and $\mathtt{Q}_6$ yield the same result on $\mathbf{S}_1$ and $\mathbf{S}_2$, whereas $\mathtt{Q}_5$, which is 3-counting-only, evaluates to `false` on $\mathbf{S}_1$ and `true` on $\mathbf{S}_2$.

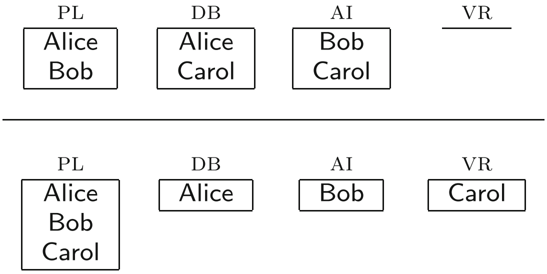
Finally, notice that the concept of counting-only queries applies to more general data models than the bag-of-sets model. Consider, e.g., a database with a student-course relation *SC* and a department-course relation *DC*, with the obvious meaning. On this database, query

$$\mathtt{P} = \{\langle x, y\rangle \mid \text{count}(\{z \mid SC(x, z) \wedge DC(y, z)\}) = \text{count}(\{z \mid SC(x, z)\})\}$$

returns student-department pairs in which the student only takes courses offered by that department. This query, conceptually similar to $\mathtt{Q}_4$ above, certainly has a counting-only flavor.

Motivated by the above, we believe that the class of counting-only queries deserves a broader understanding. Our notion of k-counting-only queries, $k \geq 0$, significantly generalizes the notion of counting-only queries of Gyssens et al. [13], which only captures the case $k = 1$ in this present work.

Since many interesting counting-only queries are first-order definable, including $\mathtt{Q}_1$ and $\mathtt{Q}_3$–$\mathtt{Q}_6$, we study more specifically the class of first-order definable counting-only queries on the bags-of-sets data model. To do so, we use (a variation of) the two-sorted first-order logic SyCALC of Gyssens et al [13]. In this logic, we have object variables, set name variables, and a set-membership relation relating objects and set names. Within SyCALC, we can express simple count terms such as "count(x) $\geq$ 2" and "count(x, y) $\geq$ 1", of $\mathtt{Q}_1$ and $\mathtt{Q}_3$, in a straightforward manner. We also *semantically* define the class k-SyCALC of all first-order definable k-counting-only queries. As this class is *semantically* defined, it is not surprising that it is already undecidable whether a SyCALC query is in k-SyCALC. As an alternative to k-SyCALC, we define the query language GCount-k, the language that expresses counting-only queries by using so-called *generalized counting* terms, which are k-counting-only. We show that GCount-k queries are all in SyCALC. From the relation between GCount and SyCALC, we derive a *syntactic restriction* of k-SyCALC, SimpleCALC. Our main results are as follows:



$$\begin{aligned} \text{count}() &= 4;\\ \text{count}(\mathsf{A}) &= 2;\\ \text{count}(\mathsf{B}) &= 2;\\ \text{count}(\mathsf{C}) &= 2;\\ \text{count}(\mathsf{A}, \mathsf{B}) &= 1;\\ \text{count}(\mathsf{A}, \mathsf{C}) &= 1;\\ \text{count}(\mathsf{B}, \mathsf{C}) &= 1. \end{aligned}$$

Fig. 2 *Left*, Bags of sets $\mathbf{S}_1$ (top) and $\mathbf{S}_2$ (bottom), both assigning students to four courses. *Right*, The count-information shared between $\mathbf{S}_1$ and $\mathbf{S}_2$

1. We semantically define the class of k-counting-only queries, and show that they include many practically relevant first-order-definable queries. We also show that not all counting-only queries are first-order-definable.
2. We formalize GCount-k, $k \geq 0$, as the class of queries that can be written using so-called generalized counting terms. We show that GCount-k captures many practical queries in k-SyCALC. This is in particular the case for those that can be written using simple "count($\cdot$)" terms in a way that shall be made precise. Examples of such queries are Q_1 and Q_3–Q_6.
3. We define the class SimpleCALC-k, $k \geq 0$, as a syntactic fragment of k-SyCALC. We show that the classes SimpleCALC-k and GCount-k are equivalent.
4. We establish that the k-counting-only queries form a non-collapsing hierarchy: for every k, $k \geq 0$, there are (k+1)-counting-only queries that are not k-counting-only. This result specializes to k-SyCALC, SimpleCALC-k, and GCount-k.
5. We show that 1-SyCALC, SimpleCALC-k, and GCount-k, $k \geq 0$, have the finite model property and use that to prove that satisfiability (and, hence, validity, query containment, and query equivalence) is decidable for these classes. We also establish that satisfiability is NEXPTIME-complete for SimpleCALC-k and GCount-k, and is in EXPTIME for 1-SyCALC. The decidability of 1-SyCALC and SimpleCALC-k, $k \geq 0$, sets them apart from many other fragments of first-order logic.

   In particular, this result identifies a large "well-behaved" fragment of first-order logic in which many practical queries can be expressed and which is distinct from the usual classes of "well-behaved" first-order queries such as the conjunctive queries, the monadic first-order logic, and the two-variable fragments of first-order logic [1, 3, 10, 11, 17].
6. In contrast to the above, satisfiability for 2-SyCALC is shown to be undecidable. Hence, we establish a strong dichotomy between 1-SyCALC and SimpleCALC-k, $k \geq 0$, on the one hand and 2-SyCALC on the other hand.

This is a revised and extended version of Hellings et al. [15]. Not only did we add full proofs, but we also managed to simplify proofs significantly, and we introduced GCount-k as an alternative syntactic language for first-order definable counting-only queries. We were also able to show that SimpleCALC-k and GCount-k are equivalent (item 2 in the above). Finally, we settled the complexity of satisfiability for SimpleCALC and 1-SyCALC.

2 Bags of sets and counting-only queries

Let $\mathcal{D}$ and $\mathcal{N}$ be two disjoint infinitely enumerable domains of objects and names. We represent finite bags of finite sets by structures, as follows:

Definition 2.1 A *structure* $\mathbf{S}$ is a pair $\mathbf{S} = (\mathbf{N}, \gamma)$, with $\mathbf{N} \subset \mathcal{N}$ a finite set of *set names* and $\gamma \subset \mathcal{D} \times \mathbf{N}$ a finite *set-membership* relation. For $\textsc{n} \in \mathbf{N}$,

$$\text{objects}(\textsc{n}; \mathbf{S}) = \{\mathsf{o} \mid (\mathsf{o}, \textsc{n}) \in \gamma\}$$

is the set of objects that are a *member* of the set named $\textsc{n}$. We write $\text{adom}(\mathbf{S}) = \bigcup_{\textsc{n} \in \mathcal{N}} \text{objects}(\textsc{n}; \mathbf{S})$ for the *active domain* of $\mathbf{S}$. If $\mathcal{A} \subseteq \mathcal{D}$, then $\mathbf{S}|_{\mathcal{A}}$ denotes the structure $(\mathbf{N}, \gamma \cap (\mathcal{A} \times \mathbf{N}))$.

Structures explicitly define the set $\mathbf{N}$ of set names they use, whereas objects are only defined via the set-membership function γ. In this way, $\mathbf{N}$ allows the representation of empty sets.

Example 2.2 The bag-of-sets dataset of Fig. 1 is represented by the structure $\mathbf{S}_1 = (\mathbf{N}, \gamma)$ with

$$\mathbf{N} = \{\textsc{pl}, \textsc{db}, \textsc{ai}\};$$
$$\gamma = \{(\mathsf{Alice}, \textsc{pl}), (\mathsf{Bob}, \textsc{pl}), (\mathsf{Alice}, \textsc{db}), (\mathsf{Bob}, \textsc{db}), (\mathsf{Carol}, \textsc{db}), (\mathsf{Dan}, \textsc{ai})\}.$$

If we were to add course VR to $\mathbf{N}$ without changing γ, this would mean that VR is offered but no student takes it.

A *query* Q maps a structure to a relation of fixed arity over objects. We write $[\![\mathsf{Q}]\!]_{\mathbf{S}}$ to denote the *evaluation* of Q on structure $\mathbf{S}$. If the arity of Q is 0, then Q is *Boolean*, where `false` and `true` are respectively represented by $\emptyset$ and $\{\langle\rangle\}$, the only two relations of arity 0.

A formal definition of counting-only queries requires a formal definition of when two structures convey the same counting information:

Definition 2.3 Let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure and $\mathrm{I} \subset \mathcal{D}$ a finite set of objects, often referred to as an *itemset*. The *cover* of I in $\mathbf{S}$ is defined by

$$\mathrm{cover}(\mathrm{I}; \mathbf{S}) = \{\textsc{n} \mid (\textsc{n} \in \mathbf{N}) \wedge (\mathrm{I} \subseteq \mathrm{objects}(\textsc{n}; \mathbf{S}))\}.$$

The *support* of I in $\mathbf{S}$ is defined by $[\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}} = |\mathrm{cover}(\mathrm{I}; \mathbf{S})|$. Let $k \geq 0$. Structures $\mathbf{S}_1$ and $\mathbf{S}_2$ are *exactly-k-counting-equivalent* if $[\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}_1} = [\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}_2}$ for every itemset I with $|\mathrm{I}| = k$. Structures $\mathbf{S}_1$ and $\mathbf{S}_2$ are *k-counting-equivalent* if they are exactly-j-counting-equivalent for all j, $0 \leq j \leq k$.

We observe that $[\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}}$ is the semantical interpretation of a count($\cdot$)-term with respect to set of objects I and structure $\mathbf{S}$.

Structures are exactly-0-counting-equivalent if they have the same number of set names. Hence, for all k, $k \geq 0$, k-counting-equivalent structures have the same number of set names.

Example 2.4 Consider the structures $\mathbf{S}_1$ and $\mathbf{S}_2$ in Fig. 2. Both have four set names representing courses. In both $\mathbf{S}_1$ and $\mathbf{S}_2$, each student takes two courses, and each pair of distinct students shares one common course. Since the itemset {Alice, Bob, Carol} has no cover in $\mathbf{S}_1$, but is covered by PL in $\mathbf{S}_2$, we conclude that $\mathbf{S}_1$ and $\mathbf{S}_2$ are 2-counting-equivalent, but not 3-counting-equivalent.

We are now ready to define k-counting-only queries:

Definition 2.5 Let $k \geq 0$. A query q is *k-counting-only* if, for every pair of k-counting-equivalent structures $\mathbf{S}_1$ and $\mathbf{S}_2$, we have $[\![q]\!]_{\mathbf{S}_1} = [\![q]\!]_{\mathbf{S}_2}$. A query is *counting-only* if there exists k, $k \geq 0$, for which it is k-counting-only.[1]

Notice that k-counting-only queries are also j-counting-only for all j, $0 \leq j \leq k$.

Example 2.6 As mentioned in the Introduction, Q_1 and Q_2 are 1-counting-only, Q_3 and Q_4 are 2-counting-only, Q_5 is 3-counting-only, and Q_6 is 0-counting-only. Query Q_5 is not 2-counting-only, since, on the 2-counting-equivalent structures $\mathbf{S}_1$ and $\mathbf{S}_2$ in Fig. 2, it returns

[1] We use the term *counting-only* in a broader sense than Gyssens et al. [13], who designated by it only the first-order definable queries that are 1-counting-only.

different results. Notice that Q_2 involves pairs of objects despite being 1-counting-only. To illustrate that this generalizes, consider

$$\begin{aligned}\mathsf{Q}_7 = \{\langle\rangle \mid \exists x \exists y_1 \exists y_2\ &(x \neq y_1) \wedge (x \neq y_2) \wedge (y_1 \neq y_2)\\ &\wedge \mathrm{count}(y_1) = \mathrm{count}(x, y_1) \wedge \mathrm{count}(y_2) = \mathrm{count}(x, y_2)\\ &\wedge \mathrm{count}(x) = \mathrm{count}(x, y_1) + \mathrm{count}(x, y_2) - \mathrm{count}(x, y_1, y_2)\}.\end{aligned}$$

On the student-courses examples, Q_7 returns `true` if there is a student who takes exactly the courses taken by a pair of distinct other students combined. Clearly, it is 3-counting-only. However, Q_7 is also 2-counting-only, as it is equivalent to

$$\begin{aligned}\mathsf{Q}_7' = \{\langle\rangle \mid \exists x \exists y_1 \exists y_2\ &(x \neq y_1) \wedge (x \neq y_2) \wedge (y_1 \neq y_2)\\ &\wedge \mathrm{count}(y_1) = \mathrm{count}(x, y_1) \wedge \mathrm{count}(y_2) = \mathrm{count}(x, y_2)\\ &\wedge \mathrm{count}(x) = \mathrm{count}(x, y_1) + \mathrm{count}(x, y_2) - \mathrm{count}(y_1, y_2)\}.\end{aligned}$$

So, some 2-counting-only queries can be used to reason on more than two objects.

3 A query language that uses counting information

We observe that k-counting information can be used to express the existence of any set-membership relation between at most k objects. To do so, we use the notion of *generalized support*, borrowed from Calders et al. [7].

Definition 3.1 The *generalized cover* of itemsets I and E in structure $\mathbf{S} = (\mathbf{N}, \gamma)$ is defined by

$$\mathrm{gcover}(\mathrm{I}; \mathrm{E}; \mathbf{S}) = \{\textsc{n} \mid (\textsc{n} \in \mathbf{N}) \wedge (\mathrm{I} \subseteq \mathrm{objects}(\textsc{n}; \mathbf{S})) \wedge (\mathrm{objects}(\textsc{n}; \mathbf{S}) \cap \mathrm{E} = \emptyset)\}$$

and the *generalized support* of I and E in **S** is defined by $[\![\mathrm{gsupport}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}} = |\mathrm{gcover}(\mathrm{I}; \mathrm{E}; \mathbf{S})|$.

Observe that $[\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}} = [\![\mathrm{gsupport}(\mathrm{I}; \emptyset)]\!]_{\mathbf{S}}$, justifying the name "generalized support". Additionally, $\mathrm{I} \cap \mathrm{E} \neq \emptyset$ implies both $\mathrm{gcover}(\mathrm{I}; \mathrm{E}; \mathbf{S}) = \emptyset$ and $[\![\mathrm{gsupport}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}} = 0$.

Example 3.2 Let **S** be the dataset presented in Fig. 1. We have

$$\begin{aligned}\mathrm{gcover}(\textsf{Alice}; \emptyset; \mathbf{S}) &= \{\textsc{pl}, \textsc{db}\};\\ \mathrm{gcover}(\textsf{Alice}; \textsf{Carol}; \mathbf{S}) &= \{\textsc{pl}\}.\end{aligned}$$

Hence,

$$\begin{aligned}[\![\mathrm{gsupport}(\textsf{Alice}; \emptyset)]\!]_{\mathbf{S}} &= [\![\mathrm{support}(\textsf{Alice})]\!]_{\mathbf{S}} = 2;\\ [\![\mathrm{gsupport}(\textsf{Alice}; \textsf{Carol})]\!]_{\mathbf{S}} &= 1.\end{aligned}$$

Using the inclusion-exclusion principle [7], which allows reasoning about generalized-support terms in terms of normal support terms, we can show that generalized-support terms $[\![\mathrm{gsupport}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}}$ are fully expressible using $|\mathrm{I} \cup \mathrm{E}|$-support terms only:

Proposition 3.3 *Let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be* k*-counting-equivalent structures and let* I, E *be itemsets with* $|\mathrm{I} \cup \mathrm{E}| \leq k$. *We have* $[\![\mathrm{gsupport}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}_1} = [\![\mathrm{gsupport}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}_2}$.

The following example illustrates the inclusion-exclusion principle and how it yields Proposition 3.3:

Example 3.4 Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be structures that are 4-counting-equivalent. In the following, we shall show that

$$\llbracket \text{gsupport}(\textsf{Alice}, \textsf{Bob}; \textsf{Carol}, \textsf{Dan}) \rrbracket_{\mathbf{S}_1} = \llbracket \text{gsupport}(\textsf{Alice}, \textsf{Bob}; \textsf{Carol}, \textsf{Dan}) \rrbracket_{\mathbf{S}_2}.$$

We write $\textsf{A}, \textsf{B}, \textsf{C}, \textsf{D}$ as shorthands for $\textsf{Alice}$, $\textsf{Bob}$, $\textsf{Carol}$, and $\textsf{Dan}$, respectively.

Let $i \in \{1, 2\}$. Consider the set $\text{cover}(\textsf{A}, \textsf{B}; \mathbf{S}_i)$. This set consists of all set names in $\mathbf{S}_i$ that are related to both $\textsf{Alice}$ and $\textsf{Bob}$. We can partition the set names in $\text{cover}(\textsf{A}, \textsf{B}; \mathbf{S}_i)$ based on whether they contain $\textsf{Carol}$ and/or $\textsf{Dan}$. Doing so yields the following:

$$\begin{aligned}\text{cover}(\textsf{A}, \textsf{B}; \mathbf{S}_i) = {} & \text{cover}(\textsf{A}, \textsf{B}, \textsf{C}, \textsf{D}; \mathbf{S}_i) \cup \text{gcover}(\textsf{A}, \textsf{B}, \textsf{C}; \textsf{D}; \mathbf{S}_i) \\ & \cup \text{gcover}(\textsf{A}, \textsf{B}, \textsf{D}; \textsf{C}; \mathbf{S}_i) \cup \text{gcover}(\textsf{A}, \textsf{B}; \textsf{C}, \textsf{D}; \mathbf{S}_i).\end{aligned}$$

From the above, we derive the following:

$$\begin{aligned}\llbracket \text{gsupport}(\textsf{A}, \textsf{B}; \textsf{C}, \textsf{D}) \rrbracket_{\mathbf{S}_i} = {} & \llbracket \text{support}(\textsf{A}, \textsf{B}) \rrbracket_{\mathbf{S}_i} - \llbracket \text{support}(\textsf{A}, \textsf{B}, \textsf{C}, \textsf{D}) \rrbracket_{\mathbf{S}_i} \\ & - \llbracket \text{gsupport}(\textsf{A}, \textsf{B}, \textsf{C}; \textsf{D}) \rrbracket_{\mathbf{S}_i} - \llbracket \text{gsupport}(\textsf{A}, \textsf{B}, \textsf{D}; \textsf{C}) \rrbracket_{\mathbf{S}_i}.\end{aligned}$$

Likewise, we observe the following relationships

$$\begin{aligned}\llbracket \text{gsupport}(\textsf{A}, \textsf{B}, \textsf{C}; \textsf{D}) \rrbracket_{\mathbf{S}_i} &= \llbracket \text{support}(\textsf{A}, \textsf{B}, \textsf{C}) \rrbracket_{\mathbf{S}_i} - \llbracket \text{support}(\textsf{A}, \textsf{B}, \textsf{C}, \textsf{D}) \rrbracket_{\mathbf{S}_i} \\ \llbracket \text{gsupport}(\textsf{A}, \textsf{B}, \textsf{D}; \textsf{C}) \rrbracket_{\mathbf{S}_i} &= \llbracket \text{support}(\textsf{A}, \textsf{B}, \textsf{D}) \rrbracket_{\mathbf{S}_i} - \llbracket \text{support}(\textsf{A}, \textsf{B}, \textsf{C}, \textsf{D}) \rrbracket_{\mathbf{S}_i}\end{aligned}$$

Combining the above, we derive

$$\begin{aligned}\llbracket \text{gsupport}(\textsf{A}, \textsf{B}; \textsf{C}, \textsf{D}) \rrbracket_{\mathbf{S}_i} = {} & \llbracket \text{support}(\textsf{A}, \textsf{B}) \rrbracket_{\mathbf{S}_i} - \llbracket \text{support}(\textsf{A}, \textsf{B}, \textsf{C}) \rrbracket_{\mathbf{S}_i} \\ & - \llbracket \text{support}(\textsf{A}, \textsf{B}, \textsf{D}) \rrbracket_{\mathbf{S}_i} + \llbracket \text{support}(\textsf{A}, \textsf{B}, \textsf{C}, \textsf{D}) \rrbracket_{\mathbf{S}_i}.\end{aligned}$$

Hence, we are able to express $\llbracket \text{gsupport}(\textsf{A}, \textsf{B}; \textsf{C}, \textsf{D}) \rrbracket_{\mathbf{S}_i}$ using only $\llbracket \text{support}(\text{I}) \rrbracket_{\mathbf{S}_i}$-terms, $|\text{I}| \leq 4$. As $\mathbf{S}_1$ and $\mathbf{S}_2$ are 4-counting-equivalent, they must agree on all these terms, from which we finally can conclude $\llbracket \text{gsupport}(\textsf{Alice}, \textsf{Bob}; \textsf{Carol}, \textsf{Dan}) \rrbracket_{\mathbf{S}_1} = \llbracket \text{gsupport}(\textsf{Alice}, \textsf{Bob}; \textsf{Carol}, \textsf{Dan}) \rrbracket_{\mathbf{S}_2}$.

Akin to the relationship between the support and count$(\cdot)$-terms, we can introduce gcount$(\cdot)$-terms and relate them to the generalized support. We introduce basic gcount$(\cdot)$ terms of the form $\text{gcount}(\mathcal{X}; \mathcal{Y}) \sim c$, with $\mathcal{X}$ and $\mathcal{Y}$ sets of object variables, "$\sim$" a comparison, and c a constant. These gcount$(\cdot)$-terms often simplify the expression of counting-only queries:[2]

Example 3.5 The queries Q_1, Q_3, Q_5, and Q_6 can be expressed with basic gcount$(\cdot)$ terms. Query Q_2 cannot be rewritten with basic gcount$(\cdot)$ terms, because it is not first-order definable [18] (see also Proposition 7.3). Query Q_4 is equivalent to $\mathsf{Q}_4' = \{\langle x, y \rangle \mid (x \neq y) \wedge \text{gcount}(x; y) = 0 \wedge \text{gcount}(y; x) = 0\}$. Finally, Q_7 and Q_7' are equivalent to

$$\begin{aligned}\mathsf{Q}_7'' = \{\langle\rangle \mid {} & \exists x \exists y_1 \exists y_2\, (x \neq y_1) \wedge (x \neq y_2) \wedge (y_1 \neq y_2) \\ & \wedge \text{gcount}(x; y_1, y_2) = 0 \wedge \text{gcount}(y_1; x) = 0 \wedge \text{gcount}(y_2; x) = 0\}.\end{aligned}$$

Next, we formalize the counting-only query language used in Example 3.5:

[2] These terms play a central role in the normal form of 1-counting-only first-order definable queries of Gyssens et al. [13]: $\mathbf{gteq}(\text{o}, c)$ corresponds to $\llbracket \text{gsupport}(\text{o}; \emptyset) \rrbracket_{\mathbf{S}} \geq c$ and $\mathbf{cogteq}(\text{o}, c)$ to $\llbracket \text{gsupport}(\emptyset; \text{o}) \rrbracket_{\mathbf{S}} \geq |\mathbf{N}| - c$.

Definition 3.6 GCount formulae are defined by the grammar

$$e := \texttt{true} \mid \text{gcount}(\mathcal{X}; \mathcal{Y}) \geq c \mid x = y \mid e \vee e \mid \neg e \mid \exists x\ e,$$

in which the lowercase variables x and y represent objects and the uppercase variables $\mathcal{X}$ and $\mathcal{Y}$ denote sets of objects (which we usually write as a sequence of lowercase variables). We use the usual logical shorthand notations. We also use the following shorthand notations

$$\begin{aligned}
\text{gcount}(\mathcal{X}; \mathcal{Y}) < c &\equiv \neg\, \text{gcount}(\mathcal{X}; \mathcal{Y}) \geq c;\\
\text{gcount}(\mathcal{X}; \mathcal{Y}) \leq c &\equiv \neg(\text{gcount}(\mathcal{X}; \mathcal{Y}) \geq c + 1);\\
\text{gcount}(\mathcal{X}; \mathcal{Y}) = c &\equiv \text{gcount}(\mathcal{X}; \mathcal{Y}) \geq c \wedge \text{gcount}(\mathcal{X}; \mathcal{Y}) \leq c;\\
\text{gcount}(\mathcal{X}; \mathcal{Y}) > c &\equiv \neg\, \text{gcount}(\mathcal{X}; \mathcal{Y}) \leq c.
\end{aligned}$$

As to the semantics of a GCount formula e, let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure and $\nu_{\mathcal{D}}$ a mapping from object variables to objects in $\mathcal{D}$. We define the relationship $(\mathbf{S}, \nu_{\mathcal{D}}) \vDash e$, with all free variables of e in the domain of $\nu_{\mathcal{D}}$, as follows:

$$\begin{aligned}
&(\mathbf{S}, \nu_{\mathcal{D}}) \vDash \texttt{true};\\
&(\mathbf{S}, \nu_{\mathcal{D}}) \vDash \text{gcount}(\mathcal{X}; \mathcal{Y}) \geq c && \text{if } [\![\text{gsupport}(\nu_{\mathcal{D}}(\mathcal{X}); \nu_{\mathcal{D}}(\mathcal{Y}))]\!]_{\mathbf{S}} \geq c;\\
&(\mathbf{S}, \nu_{\mathcal{D}}) \vDash x = y && \text{if } \nu_{\mathcal{D}}(x) = \nu_{\mathcal{D}}(y);\\
&(\mathbf{S}, \nu_{\mathcal{D}}) \vDash e_1 \vee e_2 && \text{if } (\mathbf{S}, \nu_{\mathcal{D}}) \vDash e_1 \text{ or } (\mathbf{S}, \nu_{\mathcal{D}}) \vDash e_2;\\
&(\mathbf{S}, \nu_{\mathcal{D}}) \vDash \neg e && \text{if } (\mathbf{S}, \nu_{\mathcal{D}}) \nvDash e\\
&(\mathbf{S}, \nu_{\mathcal{D}}) \vDash \exists x\ e && \text{if there exists } \mathsf{o} \in \mathcal{D} \text{ with } (\mathbf{S}, \nu_{\mathcal{D}}[x \mapsto \mathsf{o}]) \vDash e.
\end{aligned}$$

In the above, $M[\alpha \mapsto \beta]$, for any mapping M and values α and β, denotes M modified by mapping α to β.

Let e be a GCount formula with free object variables $x_1, \ldots, x_m$ and let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure. We define the *evaluation* of e on $\mathbf{S}$ by

$$[\![e]\!]_{\mathbf{S}} = \{\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \mid (\mathbf{S}, \{x_1 \mapsto \mathsf{o}_1, \ldots, x_m \mapsto \mathsf{o}_m\}) \vDash e\}.$$

We refer to GCount formulae in which terms $\text{gcount}(\mathcal{X}; \mathcal{Y}) \geq c$ are restricted to have $|\mathcal{X} \cup \mathcal{Y}| \leq k$ as the class GCount-k. We refer to GCount formulae in which object quantification is not used as BasicGCount. We refer to BasicGCount formulae in which at most k free object variables are present as BasicGCount-k. We refer to the class of BasicGCount-k formulae in which terms $\text{gcount}(\mathcal{X}; \mathcal{Y}) \geq c$ are restricted to have $c \leq d$ as the class BasicGCount[d]-k.

In Section 5, we shall formally prove that all queries in GCount are counting-only.

4 A first-order logic for bag-of-sets structures

In the previous section, we defined the query language GCount, which is already powerful enough to express many practical counting-only queries. Next, we will study the relationship between, on the one hand, counting-only queries and GCount, and, on the other hand, first-order definable queries. To do so, we use a two-sorted variant of first-order logic denoted SyCALC, based on the work of Gyssens et al. [13].[3]

[3] Gyssens et al. [13] disallow object comparisons ($x = y$ in the grammar).

Definition 4.1 *Partial* SyCALC *formulae* are defined by the grammar

$$e := \texttt{true} \mid \Gamma(x, X) \mid x = y \mid X = Y \mid e \vee e \mid \neg e \mid \exists x\ e \mid \exists X\ e,$$

in which the lowercase variables x and y represent objects, the uppercase variables X and Y denote set names, and Γ is interpreted as the set-membership relation. We also allow the usual shorthands.

As to the semantics of a partial SyCALC formula e, let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure, $\nu_{\mathcal{D}}$ a mapping from object variables to objects in $\mathcal{D}$, and $\nu_{\mathbf{N}}$ a mapping from set name variables to set names in $\mathbf{N}$. We define the relationship $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e$, with all free variables of e in the union of the domains of $\nu_{\mathcal{D}}$ and $\nu_{\mathbf{N}}$, as follows:

$$\begin{array}{ll}
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \texttt{true}; & \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \Gamma(x, X) & \text{if } (\nu_{\mathcal{D}}(x), \nu_{\mathbf{N}}(X)) \in \gamma; \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash x = y & \text{if } \nu_{\mathcal{D}}(x) = \nu_{\mathcal{D}}(y); \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash X = Y & \text{if } \nu_{\mathbf{N}}(X) = \nu_{\mathbf{N}}(Y); \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e_1 \vee e_2 & \text{if } (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e_1 \text{ or } (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e_2; \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \neg e & \text{if } (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \nvDash e; \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \exists x\ e & \text{if there exists } \mathsf{o} \in \mathcal{D} \text{ with } (\mathbf{S}, \nu_{\mathcal{D}}[x \mapsto \mathsf{o}], \nu_{\mathbf{N}}) \vDash e; \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \exists X\ e & \text{if there exists } \mathrm{N} \in \mathbf{N} \text{ with } (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}[X \mapsto \mathrm{N}]) \vDash e.
\end{array}$$

Let e be a partial SyCALC formula with free object variables $x_1, \ldots, x_m$ and free set name variables $X_1, \ldots, X_n$, and let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure. We define the *evaluation* of e on $\mathbf{S}$ by $[\![e]\!]_{\mathbf{S}} = \{\langle \mathsf{o}_1, \ldots, \mathsf{o}_m, \mathrm{N}_1, \ldots, \mathrm{N}_n\rangle \mid (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e\}$ in which $\nu_{\mathcal{D}} = \{x_1 \mapsto \mathsf{o}_1, \ldots, x_m \mapsto \mathsf{o}_m\}$ and $\nu_{\mathbf{N}} = \{X_1 \mapsto \mathrm{N}_1, \ldots, X_n \mapsto \mathrm{N}_n\}$. A *SyCALC query* is a partial SyCALC formula without free set name variables.[4]

Example 4.2 Queries Q_1 and Q_3–Q_7 are all expressible in SyCALC:

$$\begin{aligned}
\mathsf{Q}_1 &= \{\langle x\rangle \mid \exists X_1 \exists X_2\ ((X_1 \neq X_2) \wedge \Gamma(x, X_1) \wedge \Gamma(x, X_2))\}; \\
\mathsf{Q}_3 &= \{\langle x, y\rangle \mid (x \neq y) \wedge \exists X\ (\Gamma(x, X) \wedge \Gamma(y, X))\}; \\
\mathsf{Q}_4 &= \{\langle x, y\rangle \mid (x \neq y) \wedge \forall X\ (\Gamma(x, X) \iff \Gamma(y, X))\}; \\
\mathsf{Q}_5 &= \{\langle\rangle \mid \exists X \exists x \exists y \exists z\ ((x \neq y) \wedge (x \neq z) \wedge (y \neq z) \\
&\qquad \wedge \Gamma(x, X) \wedge \Gamma(y, X) \wedge \Gamma(z, X))\}; \\
\mathsf{Q}_6 &= \{\langle\rangle \mid \exists X_1 \exists X_2 \exists X_3\ ((X_1 \neq X_2) \wedge (X_1 \neq X_3) \wedge (X_2 \neq X_3))\}; \\
\mathsf{Q}_7 &= \{\langle\rangle \mid \exists x \exists y_1 \exists y_2\ ((x \neq y_1) \wedge (x \neq y_2) \wedge (y_1 \neq y_2) \\
&\qquad \wedge (\forall X\ (\Gamma(x, X) \iff (\Gamma(y_1, X) \vee \Gamma(y_2, X)))))\}.
\end{aligned}$$

For $k \geq 0$, k-SyCALC denotes the k-counting-only SyCALC queries. Not all counting-only queries are in SyCALC, however. An example is the 1-counting-only query Q_2 [18]

[4] We also write a SyCALC query e as $\{\langle x_1, \ldots, x_m\rangle \mid e\}$ to show the free object variables and their order explicitly.

(see also Proposition 7.3). Also, not all SyCALC queries are counting-only. To show this, we exhibit a SyCALC query Q and, for every k, $k \geq 1$, a pair of k-counting-equivalent structures $\mathbf{S}_{1,k}$ and $\mathbf{S}_{2,k}$ such that Q can distinguish $\mathbf{S}_{1,k}$ and $\mathbf{S}_{2,k}$. To do so, we generalize the ideas underlying Example 2.4:

Proposition 4.3 *Let $\mathcal{A}$ be a finite nonempty itemset, let $\mathbf{S}_{1,\mathcal{A}}$ be the structure representing the bags of sets $\{\mathrm{T} \mid (\mathrm{T} \subseteq \mathcal{A}) \wedge (\mathrm{even}(|\mathcal{A}-\mathrm{T}|))\}$, and let $\mathbf{S}_{2,\mathcal{A}}$ be the structure representing the bags of sets $\{\mathrm{T} \mid (\mathrm{T} \subseteq \mathcal{A}) \wedge (\mathrm{odd}(|\mathcal{A}-\mathrm{T}|))\}$. We have the following:*

1. *$\mathbf{S}_{1,\mathcal{A}}$ is $(|\mathcal{A}|-1)$-counting-equivalent to $\mathbf{S}_{2,\mathcal{A}}$.*
2. *$\mathbf{S}_{1,\mathcal{A}}$ is not exactly-$|\mathcal{A}|$-counting-equivalent to $\mathbf{S}_{2,\mathcal{A}}$.*
3. *Only one of the structures has a set name to which no objects are related.*

Proof Statement 2 follows from the observation that only the itemset $\mathcal{A}$ has $|\mathcal{A}|$ objects, and only $\mathbf{S}_1$ has a set name that covers this itemset. Statement 3 follows from the observation that $\emptyset$ is represented only in $\mathbf{S}_1$—if $\mathrm{even}(|\mathcal{A}|)$—or only in $\mathbf{S}_2$—if $\mathrm{odd}(|\mathcal{A}|)$. We now turn to Statement 1. Let $k = |\mathcal{A}|$ and $\mathrm{I} \subsetneq \mathcal{A}$ an itemset with $|\mathrm{I}| = m$. We must prove that $[\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}_1} = [\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}_2}$. Consider any itemset T with $\mathrm{I} \subseteq \mathrm{T} \subseteq \mathcal{A}$. Let $|\mathrm{T}| = n$. As T contains the objects of I, T is fully determined by choosing $n-m$ objects among the $k-m$ objects in $\mathcal{A}-\mathrm{I}$. Hence, there are exactly $\binom{k-m}{n-m}$ of such sets T. Thus, using some well-known combinatorial techniques (see, e.g., [2]),

$$\begin{aligned}[\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}_1} &= \sum_{\substack{m \leq n \leq k,\\ \mathrm{even}(k-n)}} \binom{k-m}{n-m} = \sum_{\substack{0 \leq j \leq k-m,\\ \mathrm{even}(k-m-j)}} \binom{k-m}{j} = 2^{k-m-1}\\ &= \sum_{\substack{0 \leq j \leq k-m,\\ \mathrm{odd}(k-m-j)}} \binom{k-m}{j} = \sum_{\substack{m \leq n \leq k,\\ \mathrm{odd}(k-n)}} \binom{k-m}{n-m} = [\![\mathrm{support}(\mathrm{I})]\!]_{\mathbf{S}_2},\end{aligned}$$

completing the proof. □

Using Proposition 4.3, we now prove the following:

Proposition 4.4 *Not all Boolean SyCALC queries are counting-only.*

Proof For all k, $k \geq 1$, let $\mathcal{A}_k \subset \mathcal{D}$ be a set of objects with $|\mathcal{A}_k| = k+1$, and let $\mathbf{S}_{1,\mathcal{A}_k}$ and $\mathbf{S}_{2,\mathcal{A}_k}$ be as in Proposition 4.3. We see that the Boolean SyCALC query

$$\mathsf{Q}_8 = \{\langle\rangle \mid \exists X \forall x\ (\exists Y\ (\Gamma(x, Y)) \implies \Gamma(x, X))\}$$

cannot be counting-only, since $[\![\mathsf{Q}_8]\!]_{\mathbf{S}_{1,\mathcal{A}_k}} = \mathtt{true}$ and $[\![\mathsf{Q}_8]\!]_{\mathbf{S}_{2,\mathcal{A}_k}} = \mathtt{false}$. □

Even though not all counting-only queries are first-order definable, Example 4.2 suggests that many counting-only queries are in SyCALC. Actually, all GCount queries are already in SyCALC:

Proposition 4.5 *Let e be a GCount formula. There exists a query Q in SyCALC such that, for all structures $\mathbf{S}$, $[\![e]\!]_{\mathbf{S}} = [\![\mathsf{Q}]\!]_{\mathbf{S}}$.*

Proof We only need to show how to translate terms $\text{gcount}(\mathcal{X};\mathcal{Y}) \geq c$. We have

$$\begin{aligned}\text{gcount}(\mathcal{X};\mathcal{Y}) \geq c \equiv \exists Z_1 \ldots \exists Z_c \Big(& \textstyle\bigwedge_{1\leq i<j\leq c} (Z_i \neq Z_j) \\ &\wedge \textstyle\bigwedge_{x\in\mathcal{X}} \big(\Gamma(x,Z_1) \wedge \cdots \wedge \Gamma(x,Z_c)\big) \\ &\wedge \textstyle\bigwedge_{y\in\mathcal{Y}} \big(\neg\Gamma(y,Z_1) \wedge \cdots \wedge \neg\Gamma(y,Z_c)\big)\Big).\end{aligned}$$ □

5 QuineCALC and SimpleCALC

In Section 3 we introduced GCount, a simple counting-only query language, and in Section 4 we introduced SyCALC, the first-order definable queries. We also studied the counting-only SyCALC queries, a *semantic* fragment of SyCALC. Finally, in Proposition 4.5, we showed that GCount is in SyCALC. By straightforward inspection of the proof of Proposition 4.5, we observe that GCount is not only in SyCALC, but also that every GCount query can be written as a SyCALC query of a very specific form: no object quantification occurs within the scope of a set name quantifier. This observation inspires us to define the following *syntactic*[5] query languages:

Definition 5.1 QuineCALC[6] consist of all SyCALC queries that do not use object quantification. SimpleCALC consists of all queries that are built from QuineCALC queries using disjunction, negation, and object quantification.

Notice that in SyCALC object and set name quantification can be used without any restrictions, whereas in SimpleCALC no object quantifier can be used within the scope of a set name quantifier.

We refer to the class of QuineCALC queries in which at most k free object variables are present as QuineCALC-k. We refer to the class of SimpleCALC queries in which every QuineCALC query used is in QuineCALC-k as SimpleCALC-k. We refer to the class of QuineCALC-k queries with quantifier depth at-most d as the class QuineCALC[d]-k.

A closer look at the proof of Proposition 4.5 immediately yields the following refinement of this proposition:

Proposition 5.2

1. *Let e be a BasicGCount[d]-k formula. There exists a query Q in QuineCALC[d]-k such that, for all structures* $\mathbf{S}$, $[\![e]\!]_{\mathbf{S}} = [\![\mathsf{Q}]\!]_{\mathbf{S}}$.
2. *Let e be a GCount-k formula. There exists a query Q in SimpleCALC-k such that, for all structures* $\mathbf{S}$, $[\![e]\!]_{\mathbf{S}} = [\![\mathsf{Q}]\!]_{\mathbf{S}}$.

[5] In Corollary 5.7, we show that these syntactic languages can only express counting-only queries. We need such syntactically defined languages for this purpose to ensure that we can easily decide whether an arbitrary SyCALC satisfies the relevant syntactic restrictions. In contrast, it follows readily from a result by Gyssens et al. [12] that it is already undecidable whether an arbitrary SyCALC query belongs to the semantically defined language 1-SyCALC.

[6] Gyssens et al. [13] introduced the single-object-variable fragment of QuineCALC as a first-order query language that provides a conservative extension of the symmetric Boolean functions of Quine [22], hence the name.

Next, we show that all SimpleCALC-k queries are k-counting-only. To do so, we first take a look at QuineCALC-k. From the semantics of QuineCALC-k queries, the following immediately follows.

Lemma 5.3 *Let* $\mathbf{S} = (\mathbf{N}, \gamma)$ *be a structure,* Q *be a QuineCALC-k query with l free object variables,* $l \leq k$*, and let* $\mathsf{o}_1, \ldots, \mathsf{o}_l \in \mathcal{D}$ *be l objects, not necessarily distinct. We have* $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}}$ *if and only if* $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}}$.

Next, we prove the following property.

Proposition 5.4 *Let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be k-counting-equivalent structures,* $k \geq 0$*, with* $|\mathrm{adom}(\mathbf{S}_1)| = |\mathrm{adom}(\mathbf{S}_2)| \leq k$*. The structures* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are isomorphic.*

Proof Let $\mathbf{S}_1 = (\mathbf{N}_1, \gamma_1)$ and $\mathbf{S}_2 = (\mathbf{N}_2, \gamma_2)$. To show that $\mathbf{S}_1$ and $\mathbf{S}_2$ are isomorphic, we use the identity on $\mathcal{D}$ and construct a bijection $b : \mathbf{N}_1 \rightarrow \mathbf{N}_2$ that maps each set name $\mathrm{N} \in \mathbf{N}_1$ to a set name $b(\mathrm{N}) \in \mathbf{N}_2$ with $\mathrm{objects}(\mathrm{N}; \mathbf{S}_1) = \mathrm{objects}(b(\mathrm{N}); \mathbf{S}_2)$. When $k = 0$, $|\mathrm{adom}(\mathbf{S}_1)| = |\mathrm{adom}(\mathbf{S}_2)| = 0$ and we must have $\mathrm{adom}(\mathbf{S}_1) = \mathrm{adom}(\mathbf{S}_2)$. Otherwise, As $\mathbf{S}_1$ and $\mathbf{S}_2$ are k-counting-equivalent structures, we must also have $\mathrm{adom}(\mathbf{S}_1) = \mathrm{adom}(\mathbf{S}_2)$. We denote the set $\mathrm{adom}(\mathbf{S}_1) = \mathrm{adom}(\mathbf{S}_2)$ by $\mathcal{A}$. Let $\mathrm{I} \subset \mathcal{A}$ and $\mathrm{E} = \mathcal{A} - \mathrm{I}$, $C_1 = \mathrm{gcover}(\mathrm{I}; \mathrm{E}; \mathbf{S}_1)$, and $C_2 = \mathrm{gcover}(\mathrm{I}; \mathrm{E}; \mathbf{S}_2)$. By Proposition 3.3, we have $|C_1| = |C_2|$. Hence, we can construct a bijection $b_{\mathrm{I}} : C_1 \rightarrow C_2$. By definition, we have, for all $\mathrm{I}' \subseteq \mathcal{A}$ and $\mathrm{E}' = \mathcal{A} - \mathrm{I}'$ with $\mathrm{I}' \neq \mathrm{I}$, that $\mathrm{gcover}(\mathrm{I}'; \mathrm{E}'; \mathbf{S}_1)$ is disjoint from C_1 and $\mathrm{gcover}(\mathrm{I}'; \mathrm{E}'; \mathbf{S}_2)$ is disjoint from C_2. Hence, we can construct $b = \bigcup_{\mathrm{I} \subseteq \mathcal{A}} b_{\mathrm{I}}$, and it is straightforward to show that b satisfies the claimed properties. □

We are now ready to prove that QuineCALC-k queries are k-counting-only.

Proposition 5.5 *Let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be k-counting-equivalent structures,* $k \geq 0$*, and let* Q *be a QuineCALC-k query. We have* $[\![\mathsf{Q}]\!]_{\mathbf{S}_1} = [\![\mathsf{Q}]\!]_{\mathbf{S}_2}$.

Proof Let Q be a query with free object variables $x_1, \ldots, x_l$, $l \leq k$. Let $\mathsf{o}_1, \ldots, \mathsf{o}_l \in \mathcal{D}$ be l objects, not necessarily distinct. By Lemma 5.3, $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_1|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}}$. By Proposition 5.4, $\mathbf{S}_1|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}$ and $\mathbf{S}_2|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}$ are isomorphic. Hence, $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_2|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}}$. Finally, by Lemma 5.3, $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_2}$. □

Proposition 5.5 implies that QuineCALC-k queries are k-counting-only. Next, we extend this to SimpleCALC-k.

To do so, we use the following closure properties for counting-only SyCALC queries under Boolean connectives and quantifiers. Remember that the counting-only SyCALC queries are a *semantic* and not a syntactic fragment of the SyCALC queries; hence, this result cannot be deduced straightforwardly from the fact that SyCALC is by definition closed under Boolean connectives and quantifiers.

Proposition 5.6 *Let* $k \geq 0$*. k-SyCALC is closed under disjunction, negation, and object quantification: if* e_1 *and* e_2 *are k-SyCALC queries, then* $e_1 \vee e_2$*,* $\neg e_1$*, and* $\exists x\ e_1$ *are also k-SyCALC queries.*

Proof Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be k-counting-equivalent structures. We assume e_1 and e_2 are k-counting only, hence, we have $[\![e_1]\!]_{\mathbf{S}_1} = [\![e_1]\!]_{\mathbf{S}_2}$ and $[\![e_2]\!]_{\mathbf{S}_1} = [\![e_2]\!]_{\mathbf{S}_2}$. Without loss of generality, we can assume that e_1 and e_2 both have free object variables $x_1, \ldots, x_m$. We have the following three cases:

1. We have $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_1 \vee e_2]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_1]\!]_{\mathbf{S}_1}$ or $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_2]\!]_{\mathbf{S}_1}$. As e_1 and e_2 are k-counting only, we have $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_i]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_i]\!]_{\mathbf{S}_2}$, $i \in \{1, 2\}$. Hence, we conclude $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_1 \vee e_2]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_1 \vee e_2]\!]_{\mathbf{S}_2}$.
2. We have $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![\neg e_1]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \notin [\![e_1]\!]_{\mathbf{S}_1}$. As e_1 is k-counting only, we have $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \notin [\![e_1]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \notin [\![e_1]\!]_{\mathbf{S}_2}$. Hence, we conclude $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![\neg e_1]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![\neg e_1]\!]_{\mathbf{S}_2}$.
3. Without loss of generality, we assume that $x = x_m$. We have $\langle \mathsf{o}_1, \ldots, \mathsf{o}_{m-1} \rangle \in [\![\exists x_m\ e_1]\!]_{\mathbf{S}_1}$ if and only if there exists an $\mathsf{o}_m \in \mathcal{D}$ such that $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_1]\!]_{\mathbf{S}_1}$. As e_1 is k-counting only, we have $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_1]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![e_1]\!]_{\mathbf{S}_2}$. Hence, we conclude $\langle \mathsf{o}_1, \ldots, \mathsf{o}_{m-1} \rangle \in [\![\exists x_m\ e_1]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_m \rangle \in [\![\exists x_m\ e_1]\!]_{\mathbf{S}_2}$. □

Proposition 5.2, Proposition 5.5, and Proposition 5.6 immediately yield

Corollary 5.7 *Let $k \geq 0$. All* BasicGCount-*k*, GCount-*k*, QuineCALC-*k*, *and* SimpleCALC-*k* *queries are in k-*SyCALC.

6 From SimpleCALC to GCount

In Section 3, we introduced GCount, a counting-only query language based on the notion of generalized support. In Sections 4 and 5, we showed that GCount is entirely first-order definable. Here, we show that every SimpleCALC query is expressible in GCount. Moreover, this result specializes to SimpleCALC-k and GCount-k, $k \geq 0$. We prove equivalence of SimpleCALC and GCount by proving equivalence of QuineCALC and BasicGCount. First, we observe that the counting-only queries in QuineCALC are only first-order. Hence, based on the structure of these queries, there are clear limitations on the amount of counting such a query can do. Based on this observation, we introduce *partial-counting-equivalence*, a weaker notion than counting-equivalence, and show that partial-counting-equivalence better matches the limited expressive power of QuineCALC (Lemma 6.3). Then, we use partial-counting-equivalence to relate arbitrary structures to *minimal* structures that use only a small set of set names and have a small set-membership relation (Proposition 6.6). Then, we show that BasicGCount can encode the query outcome of a given QuineCALC query on a given minimal structure (Lemma 6.6). Finally, we bootstrap this result to arbitrary minimal structures and, by Proposition 6.6, to arbitrary structures (Theorem 6.8).

Definition 6.1 Let $k, d \geq 0$. Structures $\mathbf{S}_1 = (\mathbf{N}_1, \gamma_1)$ and $\mathbf{S}_2 = (\mathbf{N}_2, \gamma_2)$ are k, d-*partial-counting-equivalent* if, for every pair of itemsets I and E with $|\mathrm{I} \cup \mathrm{E}| \leq k$, either

1. $[\![\mathrm{gsupport(I; E)}]\!]_{\mathbf{S}_1} = [\![\mathrm{gsupport(I; E)}]\!]_{\mathbf{S}_2} < d$; or
2. $d \leq [\![\mathrm{gsupport(I; E)}]\!]_{\mathbf{S}_1} \leq |\mathbf{N}_1| - d$ and $d \leq [\![\mathrm{gsupport(I; E)}]\!]_{\mathbf{S}_2} \leq |\mathbf{N}_2| - d$; or
3. $|\mathbf{N}_1| - [\![\mathrm{gsupport(I; E)}]\!]_{\mathbf{S}_1} = |\mathbf{N}_2| - [\![\mathrm{gsupport(I; E)}]\!]_{\mathbf{S}_2} < d$.

We claim that no query in QuineCALC[d]-k can distinguish k, d-partial-counting-equivalent structures. Before we prove this, we first prove the following, more general, result for SyCALC queries:

Lemma 6.2 *Let $k, d \geq 0$, let* Q *be a SyCALC query with quantifier depth d, and let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be k, d-partial-counting-equivalent structures with* $|\text{adom}(\mathbf{S}_1)| = |\text{adom}(\mathbf{S}_2)| \leq k$. *Then* $[\![\mathsf{Q}]\!]_{\mathbf{S}_1} = [\![\mathsf{Q}]\!]_{\mathbf{S}_2}$.

Proof We consider the Ehrenfeucht-Fraïssé game on structures $\mathbf{S}_1$ and $\mathbf{S}_2$ in which the spoiler can play up to d set names and any number of objects. When the Spoiler plays an object, the Duplicator responds with the same object in the other structure. When the Spoiler plays a set name N not yet played in $\mathbf{S}_i$, $i \in \{1, 2\}$, the Duplicator responds with a set name T not yet played in the other structure, $\mathbf{S}_{3-i}$, such that objects(N; $\mathbf{S}_1$) = objects(T; $\mathbf{S}_{3-i}$). Due to the structures being k, d-partial-counting-equivalent and $|\text{adom}(\mathbf{S}_1)| = |\text{adom}(\mathbf{S}_2)| \leq k$, such a set name T always exist. As a consequence, the Duplicator always has a winning strategy. Hence, no SyCALC query with quantifier depth d can distinguish between $\mathbf{S}_1$ and $\mathbf{S}_2$. It follows that $[\![\mathsf{Q}]\!]_{\mathbf{S}_1} = [\![\mathsf{Q}]\!]_{\mathbf{S}_2}$. □

Next, we specialize the above for QuineCALC queries:

Lemma 6.3 *Let $k, d \geq 0$, let* Q *be a QuineCALC[d]-k query, and let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be k, d-partial-counting-equivalent. Then* $[\![\mathsf{Q}]\!]_{\mathbf{S}_1} = [\![\mathsf{Q}]\!]_{\mathbf{S}_2}$.

Proof Let Q be a query with free object variables $x_1, \ldots, x_l$, $l \leq k$. Let $\mathsf{o}_1, \ldots, \mathsf{o}_l \in \mathcal{D}$ be l objects, not necessarily distinct. By Lemma 5.3, $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_1}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_1|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}}$ and $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_2}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_2|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}}$. Hence, we only need to show $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_1|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}}$ if and only if $\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_2|_{\{\mathsf{o}_1,\ldots,\mathsf{o}_l\}}}$, which follows from Lemma 6.2. □

Combining Proposition 5.2 and Lemma 6.3 yields

Corollary 6.4 *Let $k, d \geq 0$, let e be a BasicGCount[d]-k formula, and let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be k, d-partial-counting-equivalent. Then* $[\![e]\!]_{\mathbf{S}_1} = [\![e]\!]_{\mathbf{S}_2}$.

We remind the reader that k-counting-equivalent structures have the same number of set names. Since k, d-partial-counting-equivalence is a weaker notion, we can use it to relate structures that do not have the same number of set names.

Definition 6.5 Let $k, d \geq 0$ and let $\mathcal{A} \subset \mathcal{D}$ with $|\mathcal{A}| \leq k$. A structure $\mathbf{S} = (\mathbf{N}, \gamma)$ with $\text{adom}(\mathbf{S}) = \mathcal{A}$ is *k, d-minimal* if, for every $\mathrm{I} \subseteq \mathcal{A}$, $[\![\text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I})]\!]_{\mathbf{S}'} \leq d$.

We have the following:

Proposition 6.6 *Let $k, d \geq 0$ and let* $\mathbf{S}$ *be a structure with* $\mathcal{A} = \text{adom}(\mathbf{S})$ *and* $|\mathcal{A}| = k$. *There exists a k, d-minimal structure* $\mathbf{S}'$ *with* $\text{adom}(\mathbf{S}') = \mathcal{A}$ *such that* $\mathbf{S}$ *and* $\mathbf{S}'$ *are k, d-partial-counting-equivalent.*

Proof Let $\mathbf{S} = (\mathbf{N}, \gamma)$. We construct $\mathbf{S}' = (\mathbf{N}', \gamma')$ by encoding

$$c = \min(d, \llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}})$$

copies of each itemset $\mathrm{I} \subseteq \mathcal{A}$, this by introducing c set names for itemset I and relating each of these set names only to the objects in I. By construction, we have $\llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}'} \leq d$. It remains to prove that $\mathbf{S}$ and $\mathbf{S}'$ are k, d-partial-counting-equivalent.

From the construction used, it follows that, for every itemset $\mathrm{I} \subseteq \mathcal{A}$ with $\llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}} < d$, we have $\llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}} = \llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}'}$. Hence, these itemsets I satisfy Definition 6.1.1. Next, we consider itemsets $\mathrm{I} \subseteq \mathcal{A}$ with $\llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}} \geq d$. We distinguish the following two cases:

1. Exactly one itemset $\mathrm{I} \subseteq \mathcal{A}$ with $\llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}} \geq d$ exists. By construction, we must have $|\mathbf{N}| - \llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}} = |\mathbf{N}'| - \llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}'}$. Let $c = |\mathbf{N}| - \llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}}$. If $c \geq d$, then itemset I satisfies Definition 6.1.2. Else, if $c < d$, then itemset I satisfies Definition 6.1.3.
2. Several itemsets I with $\llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}} \geq d$ exist. In this case, $|\mathbf{N}| \geq 2d$ and, by construction, also $|\mathbf{N}'| \geq 2d$. Hence, we must have $d \leq \llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}} \leq |\mathbf{N}| - d$ and $d \leq \llbracket \text{gsupport}(\mathrm{I}; \mathcal{A} - \mathrm{I}) \rrbracket_{\mathbf{S}'} \leq |\mathbf{N}'| - d$. We conclude that these itemsets I satisfy Definition 6.1.2. □

The outcome of $\mathsf{QuineCALC}[d]$-k queries on given k, d-minimal structures can easily be encoded by $\mathsf{BasicGCount}[d]$-k formulae, which we show next.

Lemma 6.7 *Let $k, d \geq 0$, let Q be a $\mathsf{QuineCALC}[d]$-k with k free object variables, and let $\mathbf{S}$ be a k, d-minimal structure. There exists a formula e in $\mathsf{BasicGCount}[d]$-k such that $\llbracket e \rrbracket_{\mathbf{S}} = \llbracket \mathsf{Q} \rrbracket_{\mathbf{S}}$ and, for all structures $\mathbf{S}'$, $\llbracket e \rrbracket_{\mathbf{S}'} \subseteq \llbracket \mathsf{Q} \rrbracket_{\mathbf{S}'}$.*

Proof Let $x_1, \ldots, x_k$ be the free object variables in Q. Choose a mapping $t : \{x_1, \ldots, x_k\} \to \text{adom}(\mathbf{S})$. Let $\mathcal{X} \subseteq \{x_1, \ldots, x_k\}$, let $\mathrm{D} = \{t(x_1), \ldots, t(x_k)\}$, let $\mathrm{I}_{\mathcal{X}} = \{t(x) \mid x \in \mathcal{X}\}$, and let $c_{\mathcal{X}} = \llbracket \text{gsupport}(\mathrm{I}_{\mathcal{X}}; \mathrm{D} - \mathrm{I}_{\mathcal{X}}) \rrbracket_{\mathbf{S}}$. Finally, let

$$\begin{aligned} \varphi_t &= \textstyle\bigwedge_{1 \leq i < j \leq k} x_i \approx_{i,j} x_j; \\ \psi_t &= \textstyle\bigwedge_{\mathcal{X} \subseteq \{x_1, \ldots, x_k\}} \text{gcount}(\mathcal{X}; \{x_1, \ldots, x_k\} - \mathcal{X}) \sim_{\mathcal{X}} c_{\mathcal{X}}, \end{aligned}$$

in which "$\approx_{i,j}$" is "$=$" if $t(x_i) = t(x_j)$ and is "$\neq$" otherwise, and "$\sim_{\mathcal{X}}$" is "$=$" if $c_{\mathcal{X}} < d$ and "$\geq$" otherwise. We define $e_t \equiv \texttt{false}$ if $\langle t(x_1), \ldots, t(x_k) \rangle \notin \llbracket \mathsf{Q} \rrbracket_{\mathbf{S}}$ and we define $e_t \equiv \varphi_t \wedge \psi_t$ otherwise. By construction, we have $\langle t(x_1), \ldots, t(x_k) \rangle \in \llbracket e_t \rrbracket_{\mathbf{S}}$ if and only if $\langle t(x_1), \ldots, t(x_k) \rangle \in \llbracket \mathsf{Q} \rrbracket_{\mathbf{S}}$.

Next, we show that $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle \in \llbracket e_t \rrbracket_{\mathbf{S}'}$ implies $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle \in \llbracket \mathsf{Q} \rrbracket_{\mathbf{S}'}$. Assume we have $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle \in \llbracket e_t \rrbracket_{\mathbf{S}'}$. By Lemma 5.3 and Proposition 5.2, we have $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle \in \llbracket e_t \rrbracket_{\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}}}$. The subformula φ_t enforces that, for all $1 \leq i \leq j \leq k$, $\mathsf{v}_i = \mathsf{v}_j$ if and only if $t(x_i) = t(x_j)$. Hence, $|\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}| = |\mathrm{D}|$. Let $b = \{\mathsf{v}_i \mapsto t(x_i) \mid 1 \leq i \leq k\}$ be a bijection from $\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}$ to D, and let $b(\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}})$ be the structure obtained from $\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}}$ by replacing each $\mathsf{v} \in \text{adom}(\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}})$ by $b(\mathsf{v})$. By construction, we have $\langle t(x_1), \ldots, t(x_k) \rangle \in \llbracket e_t \rrbracket_{b(\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}})}$. Due to the conditions enforced by e_t, $b(\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}})$ and $\mathbf{S}|_{\mathrm{D}}$ are k, d-partial-counting-equivalent. Hence, by Corollary 6.4, $\langle t(x_1), \ldots, t(x_k) \rangle \in \llbracket e_t \rrbracket_{\mathbf{S}|_{\mathrm{D}}}$. By Lemma 5.3, we have $\langle t(x_1), \ldots, t(x_k) \rangle \in \llbracket e_t \rrbracket_{\mathbf{S}}$, which implies $\langle t(x_1), \ldots, t(x_k) \rangle \in \llbracket \mathsf{Q} \rrbracket_{\mathbf{S}}$. By Lemma 5.3 and Lemma 6.3, we have $\langle t(x_1), \ldots, t(x_k) \rangle \in \llbracket \mathsf{Q} \rrbracket_{b(\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}})}$. By construction of $b(\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}})$, we have $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle \in \llbracket \mathsf{Q} \rrbracket_{\mathbf{S}'|_{\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}}}$. Finally, by Lemma 5.3, we conclude $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle \in \llbracket \mathsf{Q} \rrbracket_{\mathbf{S}'}$.

To conclude the proof, we construct

$$e = \bigvee\nolimits_{\text{mappings } t\,:\,\{x_1,\dots,x_k\}\to\text{adom}(\mathbf{S})} e_t.$$

It follows immediately that $[\![e]\!]_{\mathbf{S}} = [\![\mathsf{Q}]\!]_{\mathbf{S}}$ and, for all structures $\mathbf{S}'$, $[\![e]\!]_{\mathbf{S}'} \subseteq [\![\mathsf{Q}]\!]_{\mathbf{S}'}$. □

Finally, we generalize Lemma 6.7 to arbitrary structures.

Theorem 6.8 *Let $k, d \geq 0$ and let Q be a QuineCALC[d]-k query. There exists a formula e in BasicGCount[d]-k such that, for all structures $\mathbf{S}'$, $[\![\mathsf{Q}]\!]_{\mathbf{S}'} = [\![e]\!]_{\mathbf{S}'}$.*

Proof For $k = 0$, we refer to Proposition 7.2. Thus assume $k > 0$. Without loss of generality, we may assume that Q is not in QuineCALC[d]-(k-1) and, hence, has k free object variables. Let $\mathcal{A} = \{\mathsf{o}_1,\dots,\mathsf{o}_k\} \subset \mathcal{D}$ be a set of k distinct objects. Let $X(\mathcal{A})$ be the set of k, d-minimal structures one can construct using only objects in $\mathcal{A}$. We observe that $X(\mathcal{A})$ is finite (up to isomorphisms). Let

$$e = \bigvee\nolimits_{\mathbf{S}\in X(\mathcal{A})} e_{\mathbf{S}},$$

in which $e_{\mathbf{S}}$ is the BasicGCount[d]-k formula for query Q and structure $\mathbf{S}$ obtained from Lemma 6.7. Next, we prove that, for all structures $\mathbf{S}'$, $[\![\mathsf{Q}]\!]_{\mathbf{S}'} = [\![e]\!]_{\mathbf{S}'}$. Let $\mathbf{S}'$ be a structure. By Lemma 5.3, $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}'}$ if and only if $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}'|_{\{\mathsf{v}_1,\dots,\mathsf{v}_k\}}}$. By Proposition 6.6, there exists a k, d-minimal structure $\mathbf{S}'_{k,d}$ that is k, d-counting-equivalent to $\mathbf{S}'|_{\{\mathsf{v}_1,\dots,\mathsf{v}_k\}}$. Hence, by Lemma 6.3, $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}'|_{\{\mathsf{v}_1,\dots,\mathsf{v}_k\}}}$ if and only if $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}'_{k,d}}$. Let $\mathbf{S}_{\mathcal{A}} \in X(\mathcal{A})$ be a k, d-minimal structure over $\mathcal{A}$ that is isomorphic to $\mathbf{S}'_{k,d}$ and chose a bijection $b : \text{adom}(\mathbf{S}'_{k,d}) \to \text{adom}(\mathbf{S}_{\mathcal{A}})$ showing that $\mathbf{S}_{\mathcal{A}}$ and $\mathbf{S}'_{k,d}$ are isomorphic. By construction, $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}'_{k,d}}$ if and only if $\langle b(\mathsf{v}_1),\dots,b(\mathsf{v}_k)\rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_{\mathcal{A}}}$. By Lemma 6.7, $\langle b(\mathsf{v}_1),\dots,b(\mathsf{v}_k)\rangle \in [\![\mathsf{Q}]\!]_{\mathbf{S}_{\mathcal{A}}}$ if and only if $\langle b(\mathsf{v}_1),\dots,b(\mathsf{v}_k)\rangle \in [\![e]\!]_{\mathbf{S}_{\mathcal{A}}}$. By construction, $\langle b(\mathsf{v}_1),\dots,b(\mathsf{v}_k)\rangle \in [\![e]\!]_{\mathbf{S}_{\mathcal{A}}}$ if and only if $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![e]\!]_{\mathbf{S}'_{k,d}}$. By Proposition 6.6, $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![e]\!]_{\mathbf{S}'_{k,d}}$ if and only if $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![e]\!]_{\mathbf{S}'|_{\{\mathsf{v}_1,\dots,\mathsf{v}_k\}}}$. Finally, by Lemma 5.3, $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![e]\!]_{\mathbf{S}'|_{\{\mathsf{v}_1,\dots,\mathsf{v}_k\}}}$ if and only if $\langle \mathsf{v}_1,\dots,\mathsf{v}_k\rangle \in [\![e]\!]_{\mathbf{S}'}$, and we conclude $[\![\mathsf{Q}]\!]_{\mathbf{S}'} = [\![e]\!]_{\mathbf{S}'}$. □

As a direct consequence of Proposition 5.2 and Theorem 6.8, we have

Corollary 6.9

1. *The class BasicGCount[d]-k is equivalent to the class QuineCALC[d]-k.*
2. *The class GCount-k is equivalent to the class SimpleCALC-k.*

7 Counting-only hierarchies

We now present four hierarchies of counting-only queries, for $k \geq 0$: k-counting-only queries, k-SyCALC, QuineCALC-k (or, equivalently, BasicGCount-k), and SimpleCALC-k (or, equivalently, GCount-k). We show that all four hierarchies are non-collapsing:

Theorem 7.1 *Let $k \geq 0$.*

1. *Every k-counting-only query is also (k+1)-counting-only.*
2. *There is a QuineCALC-(k+1) query which is not k-counting-only.*
3. *There is a Boolean SimpleCALC-(k+1) query which is not k-counting-only.*

Proof Statement 1 follows immediately from the definition. For Statements 2 and 3, let $\mathbf{S}_{1,\mathcal{A}}$ and $\mathbf{S}_{2,\mathcal{A}}$ be the structures of Proposition 4.3 with $|\mathcal{A}| = k + 1$. These structures are k-counting-equivalent, but not $(k+1)$-counting-equivalent. For Statement 2, we consider

$$\mathsf{Q} = \exists X \left(\bigwedge_{1 \le i \le k+1} \Gamma(x_i, X)\right),$$

which is a $(k+1)$-counting-only QuineCALC-$(k+1)$ query by Corollary 5.7. Let t be a $(k+1)$-tuple containing each value of $\mathcal{A}$ once. Then, $t \in [\![\mathsf{Q}]\!]_{\mathbf{S}_1}$, but $t \notin [\![\mathsf{Q}]\!]_{\mathbf{S}_2}$. Hence, Q is not k-counting-only. For Statement 3, we construct from Q the Boolean SimpleCALC-$(k+1)$ query

$$\mathsf{Q}' = \exists x_1 \dots x_{k+1} \left(\left(\bigwedge_{1 \le j < j' \le k+1} (x_j \ne x_{j'})\right) \wedge \mathsf{Q}(x_1, \dots, x_{k+1})\right).$$

Then, $[\![\mathsf{Q}']\!]_{\mathbf{S}_1} = \mathtt{true}$ and $[\![\mathsf{Q}']\!]_{\mathbf{S}_2} = \mathtt{false}$. Hence, Q' is not k-counting-only. □

Statement 3 of Theorem 7.1 can be interpreted as the Boolean version of Statement 2. Since QuineCALC-k and SimpleCALC-k queries are also k-SyCALC queries as well as k-counting-only queries, Theorem 7.1 extends to all four hierarchies.

We now proceed by comparing the fragments *mutually*. The 0-counting-only fragments have straightforward relationships:

Proposition 7.2 *The languages 0-SyCALC, SimpleCALC-0, and QuineCALC-0 all express exactly the same set of queries.*

Proof (*sketch*) A 0-counting-only query can only distinguish structures based on the number of set names, independent of the presence or absence of any objects. Distinguishing structures based on the number of set names can already be done using BasicGCount-0. □

We have already argued that the 1-counting-only query Q_2 is not first-order definable [18]. Also the 0-counting-only query

$$\mathsf{Q}_9 = \{\langle\rangle \mid \text{count() is } \textit{even}\}$$

is not first-order definable. Consequently, we have:

Proposition 7.3 *There is a Boolean 0-counting-only query not expressible in SyCALC.*

By Proposition 7.3 and Statement 1 of Theorem 7.1, Q_9 also witnesses that, for all k, $k \ge 0$, there is a Boolean k-counting-only query not expressible in k-SyCALC.

Since QuineCALC queries do not allow object quantification, all Boolean QuineCALC queries are in QuineCALC-0. Hence, no Boolean query that is k-counting-only, $k \ge 1$, but not $(k$-$1)$-counting-only is expressible in QuineCALC-k. Hence, it only remains to establish a separation between k-SyCALC and SimpleCALC-k.

We first deal with the special case $k = 1$.

Proposition 7.4 *There is a Boolean 1-SyCALC query not expressible in SimpleCALC-1.*

Proof The Boolean 1-SyCALC query

$$\mathsf{Q}_{10} = \{\langle\rangle \mid \exists x \exists y\ ((x \ne y) \wedge \exists X \exists Y\ (\Gamma(x, X) \wedge \Gamma(y, Y)))\},$$

which queries for structures with an active domain of at least two objects, is 1-counting-only but not expressible in SimpleCALC-1, as SimpleCALC-1 is syntactically unable to relate object variables. □

To establish the separation between k-SyCALC and SimpleCALC-k, $k \geq 2$, we exhibit a 2-SyCALC query, which is not 1-counting-only, that is not expressible in SimpleCALC. Thereto, let

$$\texttt{set-ids} = \forall X \exists x\ (\Gamma(x, X) \wedge \neg\exists Y\ ((X \neq Y) \wedge \Gamma(x, Y)))$$

be the Boolean query specifying that each set in a bag of sets has a distinct identifying object. We first prove that `set-ids` is in 2-SyCALC, but not in 1-SyCALC, despite it using only a single object variable.

Proposition 7.5 *Query* `set-ids` *is 2-counting-only, but not 1-counting-only.*

Proof Let $\mathsf{o}_1, \mathsf{o}_2 \in \mathcal{D}$ and $\mathrm{N}_1, \mathrm{N}_2 \in \mathcal{N}$. Let $\mathbf{S}_1 = (\{\mathrm{N}_1, \mathrm{N}_2\}, \{(\mathsf{o}_1, \mathrm{N}_1), (\mathsf{o}_2, \mathrm{N}_2)\})$ and $\mathbf{S}_2 = (\{\mathrm{N}_1, \mathrm{N}_2\}, \{(\mathsf{o}_1, \mathrm{N}_1), (\mathsf{o}_2, \mathrm{N}_1)\})$. Since $\mathbf{S}_1$ and $\mathbf{S}_2$ are 1-counting-equivalent, while $[\![\texttt{set-ids}]\!]_{\mathbf{S}_1} = \texttt{true}$ and $[\![\texttt{set-ids}]\!]_{\mathbf{S}_2} = \texttt{false}$, `set-ids` is not 1-counting-only. Next, to prove that `set-ids` is 2-counting-only, consider any structure $\mathbf{S} = (\mathbf{N}, \gamma)$ with $|\mathbf{N}| = n$. We have $[\![\texttt{set-ids}]\!]_{\mathbf{S}} = \texttt{true}$ if and only if there exist $\mathsf{o}_1, \ldots, \mathsf{o}_n \in \mathrm{adom}(\mathbf{S})$ such that, for all i, $1 \leq i \leq n$, $[\![\mathrm{support}(\mathsf{o}_i)]\!]_{\mathbf{S}} = 1$ and, for all i, j, $1 \leq i < j \leq n$, $[\![\mathrm{support}(\mathsf{o}_i, \mathsf{o}_j)]\!]_{\mathbf{S}} = 0$. By Proposition 3.3, `set-ids` is 2-counting-only. □

Observe that `set-ids` can only evaluate to `true` on a structure if the size of its active domain is lower-bounded by the number of set names in the structure. Consequently, by Lemma 5.3, we can already rule out that `set-ids` is expressible in QuineCALC. To prove that `set-ids` is not expressible in SimpleCALC, we generalize Lemma 5.3 to SimpleCALC. The actual result in presented in Proposition 7.8 and proved using Lemma 7.7. In order to state Lemma 7.7, we need the following notion of object quantification:

Definition 7.6 Let e be a SimpleCALC query. We denote the *object variable count* of e by $\mathrm{vars}(e)$, which we define as

$$\mathrm{vars}(e) = \begin{cases} k & \text{if } e \text{ is a QuineCALC-}k \text{ query;} \\ \mathrm{vars}(e') & \text{if } e \equiv \neg e' \text{ or } e \equiv \exists x\ e'; \\ \mathrm{vars}(e_1) + \mathrm{vars}(e_2) & \text{if } e \equiv e_1 \vee e_2. \end{cases}$$

Lemma 7.7 *Let e be a SimpleCALC query with k free object variables, $\mathbf{S} = (\mathbf{N}, \gamma)$ a structure, and $\nu_{\mathcal{D}}$ a mapping from free object variables in e to an itemset* $\mathrm{I} \subset \mathcal{D}$ *with* $|\mathrm{I}| \leq k$. *There exists an itemset* V *with* $\mathrm{I} \subseteq \mathrm{V}$ *and* $|\mathrm{V}| \leq \mathrm{vars}(e)$ *such that, for every itemset* W, $\mathrm{V} \subseteq \mathrm{W} \subseteq \mathcal{D}$, *we have* $(\mathbf{S}, \nu_{\mathcal{D}}, \emptyset) \models e$ *if and only if* $(\mathbf{S}|_{\mathrm{W}}, \nu_{\mathcal{D}}, \emptyset) \models e$.

Proof The proof is by induction on the structure of SimpleCALC queries. The base cases are QuineCALC queries φ with k free object variables, for which we choose $\mathrm{V} = \mathrm{I}$. By Lemma 5.3, the statement of this lemma holds for e.

Assume that, for every query φ of size at most i, the statement of this lemma holds. Let e be a query of size $i + 1$. We distinguish the following inductive cases:

1. $e \equiv e_1 \vee e_2$. We have $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$ if and only if $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e_1$ or $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e_2$. We apply the induction hypothesis on e_1 and e_2 to obtain itemsets V_1 and V_2, respectively, that satisfy the statement of this lemma. Let $\mathrm{V} = \mathrm{V}_1 \cup \mathrm{V}_2$. Observe that $|\mathrm{V}| \leq |\mathrm{V}_1| + |\mathrm{V}_2| \leq \mathrm{vars}(e_1) + \mathrm{vars}(e_2) = \mathrm{vars}(e)$. As $\mathrm{V}_1, \mathrm{V}_2 \subseteq \mathrm{V}$, we conclude that, for every itemset W, $\mathrm{V} \subseteq \mathrm{W} \subseteq \mathcal{D}$, $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e_1$ if and only if $(\mathbf{S}|_{\mathrm{W}}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models$

e_1, $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e_2$ if and only if $(\mathbf{S}|_{\mathrm{W}}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e_2$, and, by the semantics of $\vee$, $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$ if and only if $(\mathbf{S}|_{\mathrm{W}}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$.

2. $e \equiv \neg e'$. We have $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$ if and only if not $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e'$. We apply the induction hypothesis on e' to obtain itemset V' that satisfies the statement of this lemma. Let $\mathrm{V} = \mathrm{V}'$. Observe that $|\mathrm{V}| = |\mathrm{V}'| \leq \mathrm{vars}(e') = \mathrm{vars}(e)$. As $\mathrm{V}' \subseteq \mathrm{V}$, we conclude that, for every set of objects W, $\mathrm{V} \subseteq \mathrm{W} \subseteq \mathcal{D}$, $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e'$ if and only if $(\mathbf{S}|_{\mathrm{W}}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e'$ and, by the semantics of $\neg$, $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$ if and only if $(\mathbf{S}|_{\mathrm{W}}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$.
3. $e \equiv \exists x\; e'$. We have $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$ if and only if there exists an object $\mathsf{o} \in \mathcal{D}$ such that $(\mathbf{S}, \nu_{\mathcal{D}}[x \mapsto \mathsf{o}], \nu_{\mathbf{N}}) \models e'$. We apply the induction hypothesis on e' to obtain itemset V' that satisfies the statement of this lemma. Let $\mathrm{V} = \mathrm{V}'$. Observe that $|\mathrm{V}| = |\mathrm{V}'| \leq \mathrm{vars}(e') = \mathrm{vars}(e)$ and $\mathsf{o} \in \mathrm{V}$. As $\mathrm{V}' \subseteq \mathrm{V}$, we conclude that, for every set of objects W, $\mathrm{V} \subseteq \mathrm{W} \subset \mathcal{D}$, $(\mathbf{S}, \nu_{\mathcal{D}}[x \mapsto \mathsf{o}], \nu_{\mathbf{N}}) \models e'$ if and only if $(\mathbf{S}|_{\mathrm{W}}, \nu_{\mathcal{D}}[x \mapsto \mathsf{o}], \nu_{\mathbf{N}}) \models e'$ and, by the semantics of $\exists x$, $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$ if and only if $(\mathbf{S}|_{\mathrm{W}}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \models e$. □

We can now prove that `set-ids` is not expressible in SimpleCALC:

Proposition 7.8 *The query* `set-ids` *is not expressible in SimpleCALC.*

Proof Assume there exists a (Boolean) SimpleCALC query e such that, for every structure $\mathbf{S}$, $[\![e]\!]_{\mathbf{S}} = [\![\texttt{set-ids}]\!]_{\mathbf{S}}$. Let $n = \mathrm{vars}(e) + 1$, $\{\mathsf{o}_0, \ldots, \mathsf{o}_{n+1}\}$ an itemset, and $\mathbf{N} = \{\mathrm{N}_0, \ldots, \mathrm{N}_{n+1}\} \subset \mathcal{N}$. Let $\mathbf{S}_{n+1} = (\mathbf{N}, \{(\mathsf{o}_i, \mathrm{N}_i) \mid 0 \leq i \leq n+1\})$. We observe that all objects have the same behavior. Hence, with respect to Lemma 7.7 any itemset W with $|\mathrm{W}| = \mathrm{vars}(e)$ must suffice. Let $\mathrm{W} = \{\mathsf{o}_1, \ldots, \mathsf{o}_n\}$ and let $\mathbf{S}_n = \mathbf{S}_{n+1}|_{\mathrm{W}}$. By construction, $[\![e]\!]_{\mathbf{S}_{n+1}} \neq \emptyset$ and $[\![e]\!]_{\mathbf{S}_n} = \emptyset$. By Lemma 7.7, however, $[\![e]\!]_{\mathbf{S}_{n+1}} = \emptyset$ if and only if $[\![e]\!]_{\mathbf{S}_n} = \emptyset$, a contradiction. We conclude that `set-ids` is not expressible in SimpleCALC. □

Corollary 7.9 *There is a Boolean 2-SyCALC query not expressible in SimpleCALC.*

8 Dichotomy for satisfiability-related decision problems

We study the decidability of satisfiability, validity, query containment, and query equivalence for the query languages we introduced. Using Proposition 5.6, we can derive the following:

Lemma 8.1 *Let L be k-SyCALC, SimpleCALC-k, or GCount-k, and let p_1 and p_2 be two decision problems chosen from satisfiability, validity, query containment, and query equivalence. Then p_1 is decidable for L if and only if p_2 is decidable for L.*

Because of Lemma 8.1, we only study the satisfiability problem in more detail.

8.1 Satisfiability of SimpleCALC and GCount is decidable

We first observe that SimpleCALC, or, equivalently, GCount, has the *finite model property*: a query is satisfiable if and only if it is satisfiable in a structure of which the size (in terms of the number of set names and active domain objects) is uniformly bounded in terms of the size of the query. Indeed, Lemma 7.7 gives an upper-bound on the required number of active domain objects. Moreover, if the size of the active domain is upper-bounded, Proposition 6.6

can be used to also upper-bound the number of set names. From this finite model property, it follows immediately that satisfiability is decidable for SimpleCALC and GCount.

Upon closer inspection, Lemma 7.7 and Proposition 6.6 guarantee that the size of this finite model is uniformly upper-bounded by an exponential function of the query size, allowing us to state a stronger result:

Lemma 8.2 *Satisfiability is in NEXPTIME for SimpleCALC and GCount.*

Proof It suffices to consider SimpleCALC. So, let e be a SimpleCALC query with size $|e|$. Observe that both $\text{vars}(e)$ and the set name quantifier depth of e are upper-bounded by $|e|$. We answer satisfiability by trying to find a structure $\mathbf{S} = (\mathbf{N}, \gamma)$ such that $[\![e]\!]_{\mathbf{S}} \neq \emptyset$. From Lemma 7.7, we derive that $|\text{adom}(\mathbf{S})| \leq |e|$. From Proposition 6.6, we derive that $|\mathbf{N}| \leq |e| \cdot 2^{|e|}$. Hence, $|\gamma| \leq |e|^2 \cdot 2^{|e|} \leq 2^{2|e|}$. We solve satisfiability of SimpleCALC by non-deterministically choosing structure $\mathbf{S}$ and then verifying whether $[\![e]\!]_{\mathbf{S}} \neq \emptyset$. The verification step can be done via standard first-order query evaluation, for which the cost is upper-bounded by $\mathcal{O}(|e| \cdot |\mathbf{S}|^{|e|})$ [18, Proposition 6.6], in which $|\mathbf{S}|$ is the size of the structure. As $|\mathbf{S}| = \mathcal{O}(2^{2|e|})$, the cost of the verification step is upper-bounded by $\mathcal{O}(2^{|e|^2+|e|})$, completing the proof. □

Using a reduction involving monadic first-order logic (over structures with only unary relations), for which satisfiability is NEXPTIME-complete [3, 17], we can also prove that this is also a lower-bound on the complexity of the satisfiability problem:

Lemma 8.3 *Satisfiability is NEXPTIME-hard for SimpleCALC-2 and GCount-2.*

Proof We only need to show that satisfiability is NEXPTIME-hard for GCount-2. Thereto, let $S = (\mathcal{M}; X_1, \ldots, X_n)$ be a first-order structure over domain $\mathcal{M}$ with unary predicates $X_1, \ldots, X_n$ and φ a first-order logic formula over S without free variables. We encode the first-order structure S into a bag-of-sets structure. To do so, we represent the unary predicates $X_1, \ldots, X_n$ by set names $\mathrm{N}_1, \ldots, \mathrm{N}_n$. To avoid set name quantification, we associate to each set name N_i a unique identifying object o_i, $1 \leq i \leq n$. We represent the domain elements of $\mathcal{M}$ by objects distinct from $\mathsf{o}_1, \ldots, \mathsf{o}_n$, and represent predicate membership tests of the form $X_j(y)$ by $\text{count}(y, \mathsf{o}_j) = 1$ terms. In summary, we encode S by a structure $\mathbf{S} = (\mathbf{N}, \gamma)$ with $\mathbf{N} = \{\mathrm{N}_1, \ldots, \mathrm{N}_n\}$ and $\gamma = \{(\mathsf{o}_1, \mathrm{N}_1), \ldots, (\mathsf{o}_n, \mathrm{N}_n)\} \cup \{(\mathsf{m}, \mathrm{N}_i) \mid m \in \mathcal{M} \wedge X_i(m)\}$, in which m is the object representing m. We now translate φ to the query Q given by

$$\mathsf{Q}_\varphi \equiv (\text{count}() = n) \wedge \exists y_1 \ldots \exists y_n \Big(\tau(\varphi) \wedge \big(\bigwedge_{1 \leq i \leq n} \text{count}(y_i) = 1\big) \wedge \big(\bigwedge_{1 \leq i < j \leq n} \text{count}(y_i, y_j) = 0\big)\Big),$$

in which $\tau(\varphi)$ is the translation of φ obtained by replacing all subformulae $\exists y\ \varphi'$ by $\exists y\ (\bigwedge_{1 \leq i \leq n} (y \neq y_i) \wedge \tau(\varphi'(y))$ and all terms of the form $X_i(b)$ by $\text{count}(b, y_i) = 1$. We remind the reader that $\text{gcount}(\cdot)$-terms can also express $\text{count}(\cdot)$-terms in a straightforward manner. By construction, this query is a GCount-2 query. It follows straightforwardly that e is satisfiable if and only if the monadic first-order logic formula φ is satisfiable. □

Combinining Lemmas 8.2 and 8.3 immediately yields the following:

Theorem 8.4 *For $k \geq 2$, satisfiability is NEXPTIME-complete for SimpleCALC-k and GCount-k.*

8.2 Satisfiability of 1-SyCALC is decidable

By Proposition 7.2, the decidability of the satisfiability problem for 0-SyCALC follows from the decidability of the satisfiability problem for SimpleCALC-0. This does not extend to 1-SyCALC, unfortunately, but we can still prove that the satisfiability problem for 1-SyCALC is decidable. Again, we show that the finite model property holds. First, we put an upper-bound on the number of set names.

Proposition 8.5 *Let $d \geq 0$, and let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure. There exists a structure $\mathbf{S}' = (\mathbf{N}', \gamma')$ with $|\mathbf{N}'| \leq 2d$ such that $\mathbf{S}$ and $\mathbf{S}'$ are 1, d-partial-counting-equivalent structures.*

Proof If $|\mathbf{N}| \leq 2d$, we put $\mathbf{S}' = \mathbf{S}$, and Proposition 8.5 trivially holds. Otherwise, let $\mathbf{N}' = \{\mathrm{N}_1, \ldots, \mathrm{N}_{2d}\}$ and

$$\begin{aligned}\gamma' = {} & \{(\mathsf{o}, \mathrm{N}_i) \mid ([\![\mathrm{support}(\mathsf{o})]\!]_{\mathbf{S}} < d) \wedge (1 \leq i \leq [\![\mathrm{support}(\mathsf{o})]\!]_{\mathbf{S}})\} \cup \\ & \{(\mathsf{o}, \mathrm{N}_i) \mid (d \leq ([\![\mathrm{support}(\mathsf{o})]\!]_{\mathbf{S}}) \leq |\mathbf{N}| - d) \wedge (1 \leq i \leq d)\} \cup \\ & \{(\mathsf{o}, \mathrm{N}_i) \mid (|\mathbf{N}| - d < [\![\mathrm{support}(\mathsf{o})]\!]_{\mathbf{S}}) \wedge (1 \leq i < (2d - (|\mathbf{N}| - [\![\mathrm{support}(\mathsf{o})]\!]_{\mathbf{S}}))\}.\end{aligned}$$

Using that, for $o \in \mathcal{D}$ and $\mathbf{S}'' = (\mathbf{N}'', \gamma'')$ any structure, $[\![\mathrm{gsupport}(\mathsf{o}; \emptyset)]\!]_{\mathbf{S}''} = [\![\mathrm{support}(\mathsf{o})]\!]_{\mathbf{S}''}$ and $[\![\mathrm{gsupport}(\emptyset; \mathsf{o})]\!]_{\mathbf{S}''} = |\mathbf{N}''| - [\![\mathrm{support}(\mathsf{o})]\!]_{\mathbf{S}''}$, it follows straightforwardly that $\mathbf{S}$ and $\mathbf{S}'$ are 1, d-partial-counting-equivalent structures. □

Next, we put an upper bound on the number of objects.

Proposition 8.6 *Let Q be a 1-SyCALC query with set name quantifier depth d and object quantifier depth r, and let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure. Then, $[\![\mathsf{Q}]\!]_{\mathbf{S}} \neq \emptyset$ if and only if there exists a structure $\mathbf{S}' = (\mathbf{N}', \gamma')$ with $|\mathbf{N}'| \leq 2d$, $|\mathrm{adom}(\mathbf{S}')| \leq 2rd$, and $[\![\mathsf{Q}]\!]_{\mathbf{S}'} \neq \emptyset$.*

Proof By Proposition 8.5, we may assume without loss of generality that $|\mathbf{N}| \leq 2d$. Let $\mathbf{N}' = \{\mathrm{N}_1, \ldots, \mathrm{N}_{|\mathbf{N}|}\}$ and $\mathrm{I}_i = \{\mathsf{o} \mid [\![\mathrm{support}(\mathsf{o})]\!]_{\mathbf{S}} = i\}$, $1 \leq i \leq |\mathbf{N}|$. Since $\mathbf{S}$ and $\mathbf{S}'' = (\mathbf{N}', \gamma'')$ where $\gamma'' = \{(\mathsf{o}, \mathrm{N}_j) \mid (1 \leq j \leq i \leq |\mathbf{N}|) \wedge (\mathsf{o} \in \mathrm{I}_i)\}$ are 1-counting-equivalent, $[\![\mathsf{Q}]\!]_{\mathbf{S}''} = [\![\mathsf{Q}]\!]_{\mathbf{S}}$. Choose $\mathrm{P}_i \subseteq \mathrm{I}_i$ such that $|\mathrm{P}_i| = \min(|\mathrm{I}_i|, r)$, $1 \leq i \leq |\mathbf{N}|$, and let $\mathbf{S}' = (\mathbf{N}', \gamma')$ where $\gamma' = \{(\mathsf{o}, \mathrm{N}_j) \mid (1 \leq j \leq i \leq |\mathbf{N}|) \wedge (\mathsf{o} \in \mathrm{P}_i)\}$.

To show that Q cannot distinguish between structures $\mathbf{S}'$ and $\mathbf{S}''$, we consider the Ehrenfeucht-Fraïssé game in which the spoiler can play up to r objects and we allow the Spoiler to play an arbitrary amount of set names. When the Spoiler plays a set name, the Duplicator simply responds with the same set name (in the other structure). When the Spoiler plays an object o not yet played in one of the structures, the Duplicator responds with an object o' not yet played in the other structure such that the objects o and o' have the same count. The above construction of γ' out of γ'' preserves the behavior of objects with the same count and keeps r objects per count. Hence, such an object o' always exists. We conclude that the Duplicator always has a winning strategy. Hence, no SyCALC query with set name quantifier depth d and object quantifier depth r can distinguish between $\mathbf{S}'$ and $\mathbf{S}''$.

Hence, we have $[\![\mathsf{Q}]\!]_{\mathbf{S}'} \neq \emptyset$ if and only if $[\![\mathsf{Q}]\!]_{\mathbf{S}''} \neq \emptyset$. As Q is in 1-SyCALC and $\mathbf{S}$ and $\mathbf{S}''$ are 1-counting-equivalent, we conclude $[\![\mathsf{Q}]\!]_{\mathbf{S}'} \neq \emptyset$ if and only if $[\![\mathsf{Q}]\!]_{\mathbf{S}} \neq \emptyset$. □

Propositions 8.5 and 8.6 combined prove that 1-SyCALC has the finite model property and that the size of these finite models is uniformly upper-bounded by a polynomial function of the query size. Hence,

Proposition 8.7 *The satisfiability problem is decidable for 1-SyCALC queries.*

From Proposition 8.6, we can straightforwardly deduce an algorithm that decides whether a given 1-SyCALC query is satisfiable, which would give us an upper-bound to the complexity of this problem:

Proposition 8.8 *The satisfiability problem is in EXPTIME for 1-SyCALC queries.*

Proof Observe that SimpleCALC-1 and GCount-1 queries are 1-SyCALC queries. Hence, we only need to show that satisfiability for 1-SyCALC queries is in EXPTIME. Let e be a 1-SyCALC query with size $|e|$. Observe that both the set name quantifier depth and the object quantifier depth are upper-bounded by $|e|$. We answer satisfiability by trying to find a structure $\mathbf{S} = (\mathbf{N}, \gamma)$ such that $[\![e]\!]_{\mathbf{S}} \neq \emptyset$. Due to Proposition 8.6, we have $|\mathbf{N}| \leq 2|e|$, $|\text{adom}(\mathbf{S})| \leq 2|e|^2$, and $|\gamma| \leq 4|e|^3$. Hence, the size of $\mathbf{S}$ is upper-bounded by $|\mathbf{S}| = \mathcal{O}(|e|^3)$. These structures have polynomial size and can be effectively enumerated. To check whether any of these structures $\mathbf{S}$ satisfies $[\![e]\!]_{\mathbf{S}} = \emptyset$, we resort to first-order query evaluation, which is in PSPACE [18, Theorem 6.16], completing our proof. □

8.3 Satisfiability of SyCALC is undecidable

To prove undecidability of satisfiability for 2-SyCALC, we reduce satisfiability of standard first-order logic queries on undirected unlabeled graphs without self-loops, a well-known undecidable problem,[7] to satisfiability of the strict fragment of 2-SyCALC that does not allow object comparisons (of the form $x = y$).

An *undirected unlabeled graph without self-loops*, or *graph*, for short, is a pair $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ in which $\mathbf{V}$ is a set of nodes and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is an antireflexive and symmetric edge relation. On such graphs we consider standard first-order logic formulae of the form

$$e := x_1 = x_2 \mid \mathbf{E}(x_1, x_2) \mid e \vee e \mid \neg e \mid \exists x\ e,$$

in which x_1, x_2, and x are node variables. We write $[\![e]\!]_{\mathbf{G}}$ to denote the evaluation of e on $\mathbf{G}$.

We define the encoding of $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ as the structure $\text{enc}(\mathbf{G}) = (\mathbf{N}, \gamma)$ where $\mathbf{N} = \mathbf{V}$ and $\gamma = \{(\{x_1, x_2\}, x_1), (\{x_1, x_2\}, x_2) \mid (x_1, x_2) \in \mathbf{E}\} \cup \{(\{x\}, x) \mid x \in \mathbf{V}\}$. The active domain consists of node-pair sets, representing the edges of $\mathbf{G}$, and singleton node sets, serving as distinctive identifying objects. Each node pair set has a support of 2, identifying the end-points of the edge represented. The structure $\text{enc}(\mathbf{G})$ always satisfies the following Boolean SyCALC query:

$$\texttt{enc-graph} = \texttt{set-ids} \wedge (\forall x\ \text{count}(x) \leq 2).$$

[7] We have no direct reference, but if we use a straightforward encoding from binary relations to undirected unlabeled graphs without self-loops, we can rely on Trakhtenbrot's Theorem [18, Theorem 9.2].

If ν converts node variables in a first-order logic formula on graphs φ to set name variables, then the corresponding translation $\tau(\varphi)_\nu$ into a SyCALC query is defined as follows:

$$\begin{aligned}
\tau(x_1 = x_2)_\nu &\equiv \nu(x_1) = \nu(x_2);\\
\tau(\mathbf{E}(x_1, x_2))_\nu &\equiv (\nu(x_1) \neq \nu(x_2)) \wedge \exists x \; (\Gamma(x, \nu(x_1)) \wedge \Gamma(x, \nu(x_2)));\\
\tau(e_1 \vee e_2)_\nu &\equiv \tau(e_1)_\nu \vee \tau(e_2)_\nu;\\
\tau(\neg e)_\nu &\equiv \neg\tau(e)_\nu;\\
\tau(\exists x \; e)_\nu &\equiv \exists X \; \tau(e)_{\nu[x\mapsto X]},
\end{aligned}$$

with X a fresh set name variable. We define the encoding of a *Boolean* first-order logic formula on graphs φ in SyCALC as $\mathrm{enc}(\varphi) = \texttt{enc-graph} \wedge \tau(\varphi)_\emptyset$. Obviously,

Lemma 8.9 *Let* $\mathbf{G}$ *be a graph and let* φ *be a Boolean first-order logic formula on graphs. Then,* $\llbracket \varphi \rrbracket_{\mathbf{G}} = \llbracket \mathrm{enc}(\varphi) \rrbracket_{\mathrm{enc}(\mathbf{G})}$.

Next, we prove that, for any first-order Boolean logic formula φ on graphs, $\mathrm{enc}(\varphi)$ is a Boolean 2-SyCALC query. We do so by proving that 2-counting-equivalent structures satisfying the Boolean 2-SyCALC query set-ids must be isomorphic.

Lemma 8.10 *Let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be 2-counting-equivalent structures. If* $\llbracket \texttt{set-ids} \rrbracket_{\mathbf{S}_1} = \texttt{true}$, *then* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are isomorphic.*

Proof Let $\mathbf{S}_1 = (\mathbf{N}_1, \gamma_1)$, $\mathbf{S}_2 = (\mathbf{N}_2, \gamma_2)$, and $\mathbf{N}_1 = \{\mathrm{N}_1, \ldots, \mathrm{N}_k\}$. As $\mathbf{S}_1$ and $\mathbf{S}_2$ are 2-counting-equivalent, we have $|\mathbf{N}_1| = |\mathbf{N}_2|$ and $k = |\mathbf{N}_2|$. Since $\llbracket \texttt{set-ids} \rrbracket_{\mathbf{S}_1} = \texttt{true}$, there exists distinct objects $\mathsf{o}_1, \ldots, \mathsf{o}_k$ with, for all $1 \leq i \leq k$, $(\mathsf{o}_i, \mathrm{N}_i) \in \gamma_i$ and $\llbracket \mathrm{support}(\mathsf{o}_i) \rrbracket_{\mathbf{S}_1} = \llbracket \mathrm{support}(\mathsf{o}_i) \rrbracket_{\mathbf{S}_2} = 1$. Let $b : \mathbf{N} \to \mathbf{N}$ be the bijection $b = \{\mathrm{N}_i \mapsto \mathrm{N} \mid (1 \leq i \leq k) \wedge ((\mathsf{o}_i, \mathrm{N}) \in \gamma_2)\}$. Next, we show that $\mathrm{objects}(\mathrm{N}_i; \mathbf{S}_1) = \mathrm{objects}(b(\mathrm{N}_i); \mathbf{S}_2)$, $1 \leq i \leq k$, showing that the structures $\mathbf{S}_1$ and $\mathbf{S}_2$ are isomorphic. Let $\mathsf{o} \in \mathcal{D}$ be an object with $\mathsf{o} \in \mathrm{objects}(\mathrm{N}_i; \mathbf{S}_1)$. We have $\mathsf{o} \in \mathrm{objects}(\mathrm{N}_i; \mathbf{S}_1)$ if and only if $\llbracket \mathrm{support}(\mathsf{o}, \mathsf{o}_i) \rrbracket_{\mathbf{S}_1} = 1$. Due to 2-counting-equivalence, we have $\llbracket \mathrm{support}(\mathsf{o}, \mathsf{o}_i) \rrbracket_{\mathbf{S}_1} = \llbracket \mathrm{support}(\mathsf{o}, \mathsf{o}_i) \rrbracket_{\mathbf{S}_2} = 1$. Finally, by construction of b, we have $\llbracket \mathrm{support}(\mathsf{o}, \mathsf{o}_i) \rrbracket_{\mathbf{S}_2} = 1$ if and only if $\mathsf{o} \in \mathrm{objects}(b(\mathrm{N}_i); \mathbf{S}_2)$. We conclude $\mathsf{o} \in \mathrm{objects}(\mathrm{N}_i; \mathbf{S}_1)$ if and only if $\mathsf{o} \in \mathrm{objects}(b(\mathrm{N}_i); \mathbf{S}_2)$. □

Now, we can conclude directly

Corollary 8.11 *If* φ *is a Boolean first-order logic formula on graphs, then* $\mathrm{enc}(\varphi)$ *is a 2-SyCALC query.*

Now, let $\mathbf{S}$ be a structure for which $\llbracket \mathrm{enc}(\varphi) \rrbracket_{\mathbf{S}} \neq \emptyset$, with φ a Boolean first-order logic formula. For the last step in our reduction, we must find a graph $\mathbf{G}_{\mathbf{S}}$ such that $\llbracket \varphi \rrbracket_{\mathbf{G}_{\mathbf{S}}} \neq \emptyset$. Ideally, we would like that, up to isomorphism, $\mathrm{enc}(\mathbf{G}_{\mathbf{S}}) = \mathbf{S}$, but that can unfortunately not be guaranteed: a pair of sets might share several objects (which would indicate multiple edges between the nodes represented by these sets). Fortunately, given $\llbracket \mathrm{enc}(\varphi) \rrbracket_{\mathbf{S}} \neq \emptyset$, we can always derive from $\mathbf{S}$ a graph $\mathbf{G}$ and the corresponding structure $\mathrm{enc}(\mathbf{G})$ for which $\llbracket \varphi \rrbracket_{\mathbf{G}} \neq \emptyset$. We detail how next.

Definition 8.12 Let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure such that $\llbracket \texttt{enc-graph} \rrbracket_{\mathbf{S}} = \texttt{true}$. We say that $\mathbf{S}$ is a *graph encoding* if

1. for every set name $\mathrm{N} \in \mathbf{N}$, there exists exactly one object $\mathsf{o} \in \mathcal{D}$ such that $(\mathsf{o}, \mathrm{N}) \in \gamma$ and $[\![\text{support}(\mathsf{o})]\!]_{\mathbf{S}} = 1$;
2. for every pair of distinct set names $\mathrm{N}_1, \mathrm{N}_2 \in \mathbf{N}$, there exists at most one object $\mathsf{o} \in \mathcal{D}$ such that $(\mathsf{o}, \mathrm{N}_1), (\mathsf{o}, \mathrm{N}_2) \in \gamma$.

Proposition 8.13 *Let* $\mathbf{S}$ *be a structure such that* $[\![\texttt{enc-graph}]\!]_{\mathbf{S}} = \texttt{true}$.

1. *If* $\mathbf{S}$ *is a graph encoding, then there exists a graph* $\mathbf{G}$ *such that* $\text{enc}(\mathbf{G})$ *and* $\mathbf{S}$ *are isomorphic.*
2. *If* $\mathbf{S}$ *is not a graph encoding, then there exists a graph encoding* $\mathbf{S}'$ *such that, for every first-order query* φ, $[\![\text{enc}(\varphi)]\!]_{\mathbf{S}} = [\![\text{enc}(\varphi)]\!]_{\mathbf{S}'}$.

Proof Let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure such that $[\![\texttt{enc-graph}]\!]_{\mathbf{S}} = \texttt{true}$. For Statement 1, we construct the graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with $\mathbf{V} = \mathbf{N}$ and

$$\mathbf{E} = \{(\mathrm{N}_1, \mathrm{N}_2) \mid (\mathrm{N}_1 \neq \mathrm{N}_2) \wedge (\exists \mathsf{o}\ ((\mathsf{o}, \mathrm{N}_1) \in \gamma) \wedge ((\mathsf{o}, \mathrm{N}_2) \in \gamma))\}.$$

We observe that $\mathbf{S}$ and $\text{enc}(\mathbf{G})$ have the same set names. Let $f : \text{adom}(\mathbf{S}) \to \text{adom}(\text{enc}(\mathbf{G}))$ be the bijection

$$f = \{\mathsf{o} \mapsto \mathsf{o}' \mid \text{cover}(\mathsf{o}; \mathbf{S}) = \text{cover}(\mathsf{o}'; \text{enc}(\mathbf{G}))\},$$

which is uniquely defined. It is straightforward to see that f shows that $\mathbf{S}$ and $\text{enc}(\mathbf{G})$ are isomorphic.

For Statement 2, we assume that the structure $\mathbf{S}$ is not a graph encoding. In this case, there must exist distinct objects $\mathsf{p}, \mathsf{q} \in \mathcal{D}$ such that $\mathsf{p} \neq \mathsf{q}$ and $\text{cover}(\mathsf{p}; \mathbf{S}) = \text{cover}(\mathsf{q}; \mathbf{S})$. As the query $\text{enc}(\varphi)$ does not use object comparisons, the objects p and q cannot be distinguished by $\text{enc}(\varphi)$. Hence, we can reduce structure $\mathbf{S}$ to a structure $\mathbf{S}''$ by removing either p or q, and still have $[\![\text{enc}(\varphi)]\!]_{\mathbf{S}} = [\![\text{enc}(\varphi)]\!]_{\mathbf{S}''}$. We repeatedly use this step to reduce $\mathbf{S}$ to a graph encoding $\mathbf{S}'$. □

We combine Lemma 8.9, Corollary 8.11, and Proposition 8.13 to conclude:

Lemma 8.14 *Let* φ *be a Boolean first-order logic formula on graphs. If there exists a structure* $\mathbf{S}$ *satisfying* $\text{enc}(\varphi)$*, then we can construct from* $\mathbf{S}$ *a graph satisfying* φ.

Using Lemmas 8.9 and 8.14, we conclude the following:

Theorem 8.15 *The satisfiability problem is undecidable for 2-SyCALC queries.*

9 Connections with related approaches

Before concluding, we want to reflect on the choices we made with regard to the data model used in the present paper.

As we pointed out in the Introduction, the type of data that are at the basis of this study can be represented in different ways: as a bipartite graph, a bag of sets, or a two-sorted binary relation linking set names to set elements. We chose the last representation, because it is the closest to the relational model. This allows us to use the many tools developed for the latter to the fullest in our model.

The representation we chose also begs the question how our model is related with the nested relational model [25], in which set names can be linked directly with the associated set of objects. Since our model corresponds to only a very special case of the nested relational model, one may wonder why we did not consider to represent more general forms of nesting into our model. The reason for this is that the present authors studied *symmetric queries* in earlier work [12]. Symmetric queries are by definition queries on a collection of sets the order of which is irrelevant for the result. Results about the genericity of first-order logic [5, 19] and the equivalence of the relational algebra and the nested relational algebra for queries with unnested inputs and outputs [20] yield that SyCALC queries can only express *symmetric queries*, and, therefore, this language was also an ideal tool to study these queries. In the present authors' earlier work [12], it became clear very quickly that queries that could be solved merely by counting the number of elements in certain sets (as opposed to the nature and interrelationships of these elements) formed an important subclass of the symmetric queries, also from a practical perspective. In the present work, the focus was therefore shifted from symmetric queries to counting-only queries. It remains interesting, however, to generalize the concept of counting-only query to a more general context than the one considered here, and this is definitely a topic for future research.

Although generalized quantifiers are not the subject of this paper, several counting-only queries can alternatively be expressed using generalized quantifiers [4, 21, 26], as already mentioned in the Introduction. Some of these quantifiers, such as 'takes all', 'takes some', and 'takes no' have a set-based flavor, while others, such as 'takes at least k' or 'takes all but k' have more of a counting flavor. Moreover, set-based generalized quantifiers can often be translated into counting-based quantifiers, and vice-versa.

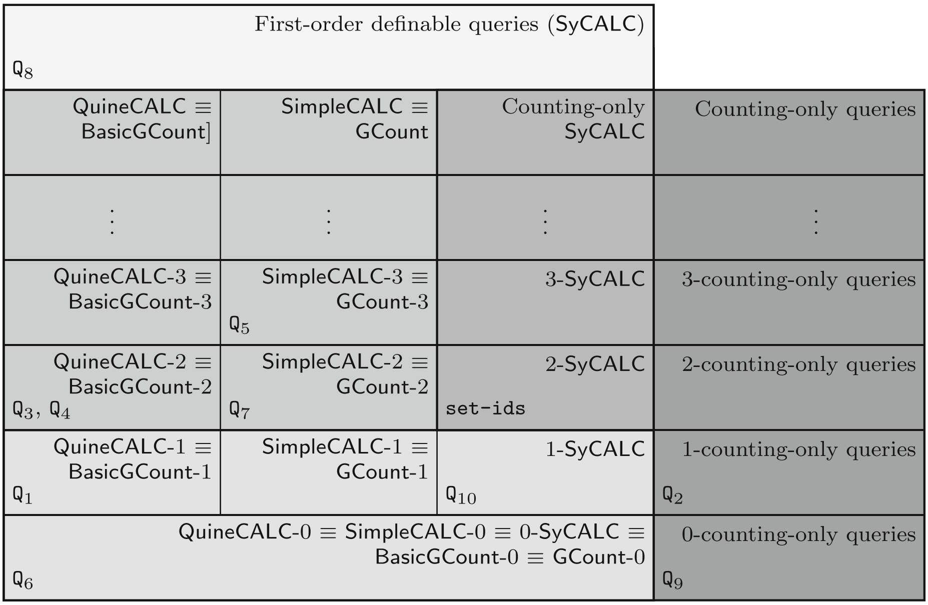
This is actually not coincidential, as pointed out by Westerstål [28]. Westerstål calls a generalized quantifier *monadic* if it involves subsets of the universe and *isomorphism-closed* if it yields the same truth value for isomorphic structures (in a data query context, this property is better known as *genericity* [27]). Notice that the generalized quantifiers considered above (and in particular the set-based ones) are both monadic and isomorphism-closed. Westerstål points out that monadic isomorphism-closed quantifiers deal only with sizes of set rather than the sets themselves. In particular, emptiness is equivalent to zero-cardinality, and this simple fact allows for the conversion of set relationships to cardinality constraints, and vice-versa.

10 Conclusion and discussion

In this paper, we studied so-called counting-only queries on bag-of-sets data, which can be answered by only counting the occurrence of itemsets of objects. In particular, we identified and studied the syntactic counting-only fragments QuineCALC and SimpleCALC of first-order logic. We showed that SimpleCALC is equivalent to GCount, the language that expresses counting-only queries by using generalized counting terms. These query languages can express many practically relevant queries other than the usual classes of "well-behaved" first-order queries—such as the conjunctive queries, the monadic first-order logic, and the two-variable fragments of first-order logic—while, at the same time, still being simple enough for satisfiability, validity, query containment, and query equivalence to be decidable. We have summarized our findings in Fig. 3.

We have identified several directions for future research:

1. In this paper, we have studied the formal aspect of counting-only first-order queries, but we have not yet studied practical issues such as query evaluation. Since the queries we



First-order definable queries (SyCALC) Q_8			
QuineCALC ≡ BasicGCount]	SimpleCALC ≡ GCount	Counting-only SyCALC	Counting-only queries
⋮	⋮	⋮	⋮
QuineCALC-3 ≡ BasicGCount-3	SimpleCALC-3 ≡ GCount-3 Q_5	3-SyCALC	3-counting-only queries
QuineCALC-2 ≡ BasicGCount-2 Q_3, Q_4	SimpleCALC-2 ≡ GCount-2 Q_7	2-SyCALC set-ids	2-counting-only queries
QuineCALC-1 ≡ BasicGCount-1 Q_1	SimpleCALC-1 ≡ GCount-1	1-SyCALC Q_{10}	1-counting-only queries Q_2
QuineCALC-0 ≡ SimpleCALC-0 ≡ 0-SyCALC ≡ BasicGCount-0 ≡ GCount-0 Q_6			0-counting-only queries Q_9

Fig. 3 Main relationships between the query languages considered. The counting-only languages are highlighted in dark-grey, and the first-order definable languages in light gray. A language to the left and/or below another language, is less expressive than the latter. Separate boxes also indicate strict separation in expressive power. The light gray area at the bottom indicates the first-order definable counting-only queries for which satisfiability is in EXPTIME, the medium-light gray area on the left indicates the first-order definable counting-only queries for which satisfiability is NEXPTIME-complete, and the medium-dark gray area in the middle indicates the first-order definable counting-only queries for which satisfiability is undecidable. The example queries Q_1–Q_6 (Introduction), Q_7 (Example 2.6), Q_8 (proof of Proposition 4.4), Q_9 (proof of Proposition 7.3), and Q_{10} (proof of Proposition 7.4) are added to the smallest language in which they can be expressed

study are all first-order queries, we can, off course, borrow standard techniques from first-order logic for their evaluation. One may wonder, however, if some of the more restricted classes considered in this paper allow for more efficient query evaluation, for example by using specialized counting-only index structures.

As an example, consider GCount. Queries in GCount provide a direct connection to an underlying frequent itemset problem, which can be exploited to further optimize query evaluation. A good example of such a technique is the FP-tree used by the FP-Growth Algorithm [7, 14]. The FP-tree can be used as an index to quickly find candidate sets of up-to-k-objects that have a minimum count, and prune away all other sets of up-to-k-objects without any counting. Due to these implementation optimization opportunities and the prevalence of counting-only queries, we believe that the evaluation of these GCount queries and their relationship to frequent itemset mining deserves a deeper understanding.

2. In the Introduction, we have already mentioned that the bag-of-sets data model and the notion of counting-only query can easily be generalized, e.g., to a model with relations between more than two disjoint domains. Therefore, it is only natural to wonder if the concepts we developed generalize to a richer data model without giving up on the well-behaved nature of SimpleCALC or GCount.

3. In the Introduction, we also mentioned connections between set theory and the bags-of-sets data model. For example, certain comparisons of counts can be linked to set-theoretic comparisons. We believe that these connections deserve further investigation.
4. From a more theoretical perspective, there are several open problems for further investigation. For example, we do not yet know if it is decidable whether a given (k+1)-counting-only query is also k-counting-only. Also the precise complexity of satisfiability and the related decision problems for 1-SyCALC remain open. Crucial in pinpointing an exact upper-bound on the complexity of these decision problems is finding the exact upper-bound on the complexity of model checking, which might be better than the worst-case upper-bound for model checking general first-order logic.
5. In Section 7 we introduced the `set-ids` query, a query in 2-SyCALC not expressible in SimpleCALC. This query asks whether each set in a structure is uniquely identified by some object. It remains open to identify queries in k-SyCALC, $k > 2$, that are not (k-1)-counting-only and not expressible in SimpleCALC. A positive result on this would strengthen our understanding of the hierarchy of counting-only queries as studied in Section 7, as it would provide a hierarchy of counting-only queries in SyCALC not expressible in SimpleCALC.
6. Counting is only one type of measure that can be used to define practical queries on bag-of-sets data, and we have seen that taking counting into account leads to naturally definable and well-behaved query languages. Many other practical types of measure exist [24], hence it is only natural to ask if these measures can be captured in an encompassing framework that leads, for each measure, to naturally definable and well-behaved query languages.
7. As we have shown in this paper, not all counting-only queries are first-order definable. To express some of these queries, one might consider augmenting first-order logic with non-first-order definable counting-based quantifiers and predicates such as the predicate *even* [16]. We believe that it is worthwhile to study whether one can construct such query languages while, at the same time, retain the well-behaved nature of SimpleCALC and GCount.

Acknowledgments This is a revised and extended version of the conference paper ‘First-order definable counting-only queries’, presented at the 10th International Symposium on Foundations of Information and Knowledge Systems, Budapest, Hungary (FoIKS 2018) [15].

References

1. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases: The Logical Level, 1st edn. Addison-Wesley Longman Publishing Co. Inc, Reading (1995)
2. Anderson, I.: Combinatorics of Finite Sets. Dover Publications, New York (2011)
3. Bachmair, L., Ganzinger, H., Waldmann, U.: Set constraints are the monadic class. In: Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science, pp. 75–83 (1993)
4. Badia, A., Van Gucht, D., Gyssens, M.: Querying with Generalized Quantifiers, pp. 235–258. Springer, Boston (1995)
5. Bancilhon, F.: On the completeness of query languages for relational data bases. In: Winkowski, J. (ed.) Mathematical Foundations of Computer Science 1978, Proceedings, 7th Symposium, Zakopane, Poland, September 4-8, 1978, Lecture Notes in Computer Science, vol. 64, pp. 112–123. Springer (1978). https://doi.org/10.1007/3-540-08921-7_60
6. Bayer, A.E., Smart, J.C., McLaughlin, G.W.: Mapping intellectual structure of a scientific subfield through author cocitations. J. Amer. Soc. Inf. Sci. **41**(6), 444–452 (1990)
7. Calders, T., Goethals, B.: Non-derivable itemset mining. Data Min. Knowl. Disc. **14**(1), 171–206 (2007)
8. Fletcher, G.H.L., Van Den Bussche, J., Van Gucht, D., Vansummeren, S.: Towards a theory of search queries. ACM Trans. Database Syst. **35**(4), 28:1–28:33 (2010)

9. Goethals, B.: Survey on Frequent Pattern Mining. Technical report, University of Helsinki (2003)
10. Grädel, E., Otto, M.: On logics with two variables. Theor. Comput. Sci. **224**(1–2), 73–113 (1999)
11. Grohe, M.: Finite variable logics in descriptive complexity theory. Bull. Symb. Log. **4**, 345–398 (1998)
12. Gyssens, M., Hellings, J., Paredaens, J., Van Gucht, D., Wijsen, J., Wu, Y.: Calculi for symmetric queries. J. Comput. Syst. Sci. **105**, 54–86 (2019)
13. Gyssens, M., Paredaens, J., Van Gucht, D., Wijsen, J., Wu, Y.: An approach towards the study of symmetric queries. Proc. VLDB Endow. **7**(1), 25–36 (2013)
14. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min. Knowl. Disc. **8**(1), 53–87 (2004)
15. Hellings, J., Gyssens, M., Van Gucht, D., Wu, Y.: First-order definable counting-only queries. In: Foundations of Information and Knowledge Systems, pp. 225–243. Springer International Publishing (2018)
16. Kuske, D., Schweikardt, N.: First-order logic with counting. In: 2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 1–12 (2017)
17. Lewis, H.R.: Complexity results for classes of quantificational formulas. J. Comput. Syst. Sci. **21**(3), 317–353 (1980)
18. Libkin, L.: Elements of Finite Model Theory. Springer, Berlin (2004)
19. Paredaens, J.: On the expressive power of the relational algebra. Inf. Process. Lett. **7**(2), 107–111 (1978). https://doi.org/10.1016/0020-0190(78)90055-8
20. Paredaens, J., Van Gucht, D.: Converting nested algebra expressions into flat algebra expressions. ACM Trans. Database Syst. **17**(1), 65–93 (1992). https://doi.org/10.1145/128765.128768
21. Peters, S., Westerståhl, D.: Quantifiers in Language and Logic. Oxford University Press, London (2008). https://doi.org/10.1093/acprof:oso/9780199291267.001.0001
22. Quine, W.V.: Selected Logic Papers. Harvard University Press, Cambridge (1995)
23. Sayrafi, B., Van Gucht, D.: Differential constraints. In: Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 348–357. ACM (2005)
24. Sayrafi, B., Van Gucht, D., Gyssens, M.: Measures in Databases and Data Mining. Tech. Rep. TR602, Indiana University. https://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR602 (2004)
25. Thomas, S.J., Fischer, P.C.: Nested relational structures. Adv. Comput. Res. **3**, 269–307 (1986)
26. Väänänen, J.: Generalized Quantifiers, an Introduction. In: Generalized Quantifiers and Computation: 9th European Summer School in Logic, Language, and Information, pp. 1–17. Springer, Berlin (1999)
27. Vianu, V., Van Gucht, D.: Computationally complete relational query languages. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems. 1st edn., pp. 1–7. Springer, New York (2017)
28. Westerståhl, D.: Generalized quantifiers. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy, winter 2016 edn. Metaphysics Research Lab, Stanford University (2016)

Affiliations

Jelle Hellings[1,2] · Marc Gyssens[2] · Dirk Van Gucht[3] · Yuqing Wu[4]

Jelle Hellings
jhellings@ucdavis.edu

Dirk Van Gucht
vgucht@cs.indiana.edu

Yuqing Wu
melanie.wu@pomona.edu

[1] Exploratory Systems Lab, Department of Computer Science, University of California,
Davis, CA 95616-8562, USA

[2] Faculty of Sciences, Hasselt University, Martelarenlaan 42, 3500, Hasselt, Belgium

[3] School of Informatics, Computing, and Engineering, Indiana University,
919 E 10th St, Bloomington, IN 47408, USA

[4] Pomona College, 185 E 6th St., Claremont, CA 91711, USA