First-Order Definable Counting-Only Queries

Jelle Hellings\textsuperscript{1}  
Marc Gyssens\textsuperscript{1}  
Dirk Van Gucht\textsuperscript{2}  
Yuqing Wu\textsuperscript{3}

\textsuperscript{1} Hasselt University  
\textsuperscript{2} Indiana University  
\textsuperscript{3} Pomona College
Bag-of-sets datasets and queries

- ‘Return students who take at least 2 courses.’
  \[
  \{ \langle x \rangle \mid \text{count}(x) \geq 2 \}
  \]

- ‘Return pairs of students who take the same number of courses.’
  \[
  \{ \langle x, y \rangle \mid (x \neq y) \land \text{count}(x) = \text{count}(y) \}
  \]
Formalization: the bag-of-sets data model

Definition

A \textit{structure} $S$ over domain $\mathcal{D}$ of objects is a pair $S = (N, \gamma)$, with:

- $N$ a finite set of \textit{set names};
- $\gamma \subset \mathcal{D} \times N$ a finite \textit{set-membership} relation.

Let $S = (N, \gamma)$, $n \in N$ a set name, and $A \subset \mathcal{D}$ a finite set of objects:

- The \textit{cover} is defined by
  \[
  \text{cover}(A; S) = \{ n | (n, A) \in \gamma \}. 
  \]
- The \textit{support} is defined by
  \[
  \mathbb{K}\left(\text{count}(A)\right)_S = |\text{cover}(A; S)|. 
  \]
Counting-only queries

- ‘Return pairs of distinct students which take a common course.’
  \[
  \{ \langle x, y \rangle \mid (x \neq y) \land \text{count}(x, y) \geq 1 \}\]

- ‘Return pairs of distinct students which take the same courses.’
  \[
  \{ \langle x, y \rangle \mid (x \neq y) \land \text{count}(x, y) = \text{count}(x) = \text{count}(y) \}\]

<table>
<thead>
<tr>
<th>PL</th>
<th>DB</th>
<th>AI</th>
<th>VR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Alice</td>
<td>Bob</td>
<td>Carol</td>
</tr>
<tr>
<td>Bob</td>
<td>Carol</td>
<td>Path: Alice, Bob, Alice, Bob</td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td>Alice</td>
<td>Path: Alice, Carol, Alice, Carol</td>
<td></td>
</tr>
</tbody>
</table>

- \(\text{count}() = 4\);
- \(\text{count}(A) = 2\);
- \(\text{count}(B) = 2\);
- \(\text{count}(C) = 2\);
- \(\text{count}(A, B) = 1\);
- \(\text{count}(A, C) = 1\);
- \(\text{count}(B, C) = 1\).
Formalization: $k$-counting-only queries

**Definition**

Let $S_1$ and $S_2$ be two structures.

- $S_1$ and $S_2$ are *exactly-$k$-counting-equivalent* if, for every set of objects $A$ with $|A| = k$, 
  \[
  [\text{count}(A)]_{S_1} = [\text{count}(A)]_{S_2}. 
  \]

- $S_1$ and $S_2$ are *$k$-counting-equivalent* if they are exactly-$j$-counting-equivalent for every $j$, $0 \leq j \leq k$.

**Definition**

Let $Q$ be a query.

- $Q$ is *$k$-counting-only* if $[Q]_{S_1} = [Q]_{S_2}$ for every pair of $k$-counting-equivalent structures $S_1$ and $S_2$.

- $Q$ is *counting-only* if it is $k$-counting-only for some $k$. 
Proving that a query is not 2-counting-only

‘Does there exist a course taken by 3 students?’

\[ \{\langle \rangle \mid \exists x \exists y \exists z ((x \neq y \land x \neq z \land y \neq z) \land \text{count}(x, y, z) \geq 1)\} \]

This query is clearly 3-counting-only

Is this query 2-counting-only?
It can distinguish between 2-counting-equivalent structures!
Proving that a query is 2-counting-only

‘Does there exists a student who takes courses that are taken by a pair of other students?’

\{ \langle \rangle \mid \exists x \exists y_1 \exists y_2 \ (x \neq y_1) \land (x \neq y_2) \land (y_1 \neq y_2) \land \\
\text{count}(y_1) = \text{count}(x, y_1) \land \text{count}(y_2) = \text{count}(x, y_2) \land \\
\text{count}(x) = \text{count}(x, y_1) + \text{count}(x, y_2) - \text{count}(y_1, y_2) \}.

This query is clearly 3-counting-only.

Is this query 2-counting-only?

It is equivalent to the 2-counting-only query

\{ \langle \rangle \mid \exists x \exists y_1 \exists y_2 \ (x \neq y_1) \land (x \neq y_2) \land (y_1 \neq y_2) \land \\
\text{count}(y_1) = \text{count}(x, y_1) \land \text{count}(y_2) = \text{count}(x, y_2) \land \\
\text{count}(x) = \text{count}(x, y_1) + \text{count}(x, y_2) - \text{count}(y_1, y_2) \}.
First-order logic and counting-only queries
SyCALC: first-order logic on bag-of-sets

Definition

SyCALC formulae are defined by the grammar

\[ e := \Gamma(x, X) \mid x = y \mid X = Y \mid e \lor e \mid \neg e \mid \exists x \ e \mid \exists X \ e. \]

A SyCALC query is a formula without free set name variables.

Example

- ‘Return students who take at least 2 courses.’

  \[ \{ \langle x \rangle \mid \text{count}(x) \geq 2 \} \quad \rightarrow \quad \{ \langle x \rangle \mid \exists Y \exists Z ((Y \neq Z) \land \Gamma(x, Y) \land \Gamma(x, Z)) \} \]

- ‘Return pairs of distinct students which take a common course.’

  \[ \{ \langle x, y \rangle \mid (x \neq y) \land \text{count}(x, y) \geq 1 \} \quad \rightarrow \quad \{ \langle x, y \rangle \mid (x \neq y) \land \exists C (\Gamma(x, C) \land \Gamma(y, C)) \} \]
**Definition**

$k$-SyCALC, $k \geq 0$, denotes the *k-counting-only SyCALC queries*.

**Proposition**

Let $Q_1$ and $Q_2$ be $k$-SyCALC queries:

- $Q_1 \lor Q_2$ is in $k$-SyCALC.
- $\neg Q_1$ is in $k$-SyCALC.
- $\exists x \ Q_1$ is in $k$-SyCALC.

Hence: $k$-SyCALC is closed under disjunction, negation, and object quantification.
Not all counting-only queries are in SyCALC

- ‘Return pairs of students who take the same number of courses.’
  \[ \{ \langle x, y \rangle \mid (x \neq y) \land \text{count}(x) = \text{count}(y) \} \].
  This query is 1-counting-only, but not first-order definable.

- ‘Are there an even number of courses?’
  \[ \{ \langle \rangle \mid \exists k \ \text{count}() = 2k \} \].
  This query is 0-counting-only, but not first-order definable.
Not all SyCALC queries are counting-only

- ‘Does there exist a course followed by all students?’
  \[ \{ \langle \rangle \mid \exists C \forall x \exists Y (\Gamma(x, Y) \implies \Gamma(x, C)) \} \]

- ‘Does there exist a course followed by no students?’
  \[ \{ \langle \rangle \mid \exists C \forall x (\neg \Gamma(x, C)) \} \]

How to prove this?

Find \( k \)-counting equivalent structures distinguished by these queries (for every \( k \)).
Constructing $k$-counting equivalent structures

Example: $k = 3$.
Constructing $k$-counting equivalent structures

Example: $k = 3.$

$S_1$

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
\end{array}
\]

$T_1$

\[
\begin{array}{c}
A \\
B \\
C \\
\end{array}
\]

$T_2$

\[
\begin{array}{c}
A \\
B \\
D \\
\end{array}
\]

$T_3$

\[
\begin{array}{c}
A \\
C \\
D \\
\end{array}
\]

$T_4$

\[
\begin{array}{c}
B \\
C \\
D \\
\end{array}
\]

Structures are 3-counting equivalent, but not 4-counting equivalent.
Constructing $k$-counting equivalent structures

Example: $k = 3$.

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

Structures are 3-counting equivalent, but not 4-counting equivalent.
Constructing $k$-counting equivalent structures

Example: $k = 3$.

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

Structures are 3-counting equivalent, but not 4-counting equivalent.
Constructing $k$-counting equivalent structures

Example: $k = 3$.

<table>
<thead>
<tr>
<th></th>
<th>S_1</th>
<th>S_2</th>
<th>S_3</th>
<th>S_4</th>
<th>S_5</th>
<th>S_6</th>
<th>S_7</th>
<th>S_8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Structures are 3-counting equivalent, but not 4-counting equivalent.
Constructing $k$-counting equivalent structures

Example: $k = 3$.

$\begin{array}{cccccc}
S_1 & S_2 & S_3 & S_4 & S_5 & S_6 \\
B & B & C & D & C & D \\
C & \_ & \_ & \_ & \_ & \_ \\
D & \_ & \_ & \_ & \_ & \_ \\
\end{array}$

$\begin{array}{cccccc}
T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \\
B & B & C & C & B & C \\
C & D & D & D & C & D \\
\_ & \_ & \_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ & \_ & \_ \\
\end{array}$

Structures are $3$-counting equivalent, but not $4$-counting equivalent.
A counting-only fragment of SyCALC
A powerful observation

Many queries can be expressed using simple \textit{generalized count} terms.

Example

‘Does there exists a student who takes courses that are taken by a pair of other students?’

\[
\langle \emptyset \rangle \mid \exists x \exists y_1 \exists y_2 \ (x \neq y_1) \land (x \neq y_2) \land (y_1 \neq y_2) \land \\
\text{count}(y_1) = \text{count}(x, y_1) \land \text{count}(y_2) = \text{count}(x, y_2) \land \\
\text{count}(x) = \text{count}(x, y_1) + \text{count}(x, y_2) - \text{count}(x, y_1, y_2) \}
\]

This query is equivalent to

\[
\langle \emptyset \rangle \mid \exists x \exists y_1 \exists y_2 \ (x \neq y_1) \land (x \neq y_2) \land (y_1 \neq y_2) \land \\
g\text{count}(x; y_1, y_2) = 0 \land g\text{count}(y_1; x) = 0 \land g\text{count}(y_2; x) = 0\}
\]
QuineCALC and SimpleCALC

Terms \( \text{count}(S) \geq c \) or \( g\text{count}(X; Y) \geq c \) are simple to express:

\[
g\text{count}(X; Y) \geq c = \exists Z_1 \ldots \exists Z_c \\
\left( \bigwedge_{1 \leq i < j \leq c} (Z_i \neq Z_j) \land \bigwedge_{x \in X} (\Gamma(x, Z_1) \land \cdots \land \Gamma(x, Z_c)) \land \bigwedge_{y \in Y} (\neg \Gamma(y, Z_1) \land \cdots \land \neg \Gamma(y, Z_c)) \right).
\]
QuineCALC and SimpleCALC

Terms \( \text{count}(S) \geq c \) or \( \text{gcount}(X; Y) \geq c \) are simple to express:

\[
gcount(X; Y) \geq c = \exists Z_1 \ldots \exists Z_c \left( \bigwedge_{1 \leq i < j \leq c} (Z_i \neq Z_j) \land \bigwedge_{x \in X} (\Gamma(x, Z_1) \land \cdots \land \Gamma(x, Z_c)) \land \bigwedge_{y \in Y} (\neg \Gamma(y, Z_1) \land \cdots \land \neg \Gamma(y, Z_c)) \right).
\]

Definition

- QuineCALC-\( k \) consist of all SyCALC queries that do not use object quantification and with at most \( k \) free object variables.
- SimpleCALC-\( k \) consists of all queries that are built from QuineCALC-\( k \) queries using disjunction, negation, and object quantification.
Every SimpleCALC-\(k\) query is in \(k\)-SyCALC

Proposition

*Every SimpleCALC-\(k\) query is \(k\)-counting-only.*

- SimpleCALC-\(k\) are built from QuineCALC-\(k\) queries using disjunction, negation, and object quantification.
- Closure results of \(k\)-SyCALC apply to SimpleCALC-\(k\).

Hence: prove that QuineCALC-\(k\) queries are \(k\)-counting-only!
Every QuineCALC-\(k\) query is in \(k\)-SyCALC

**Definition**

Let \(S = (N, \gamma)\) be a structure. If \(A\) is a set of objects, then \(S|_A\) denotes the structure \((N, \gamma \cap (A \times N))\).

**Lemma**

Let \(S = (N, \gamma)\) be a structure and \(Q(x_1, \ldots, x_k)\) be a QuineCALC-\(k\) query. We have

\[
\langle o_1, \ldots, o_k \rangle \in [Q]_S \iff \langle o_1, \ldots, o_k \rangle \in [Q]_{S|\{o_1, \ldots, o_k\}}.
\]
Every QuineCALC-\(k\) query is in \(k\)-SyCALC

**Definition**
Let \(S = (\mathbb{N}, \gamma)\) be a structure. If \(A\) is a set of objects, then \(S\rvert_A\) denotes the structure \((\mathbb{N}, \gamma \cap (A \times \mathbb{N}))\).

**Lemma**
Let \(S = (\mathbb{N}, \gamma)\) be a structure and \(Q(x_1, \ldots, x_k)\) be a QuineCALC-\(k\) query. We have

\[
\langle o_1, \ldots, o_k \rangle \in \llbracket Q \rrbracket_S \iff \langle o_1, \ldots, o_k \rangle \in \llbracket Q \rrbracket_{S\rvert\{o_1, \ldots, o_k\}}.
\]

**Lemma**
Let \(S_1\) and \(S_2\) be two \(k\)-counting-equivalent structures. If \(A\) is a set of objects with \(|A| \leq k\), then \(S_1\rvert_A\) is isomorphic to \(S_2\rvert_A\).
A counting-only hierarchy

Theorem

Let \( k \geq 0 \). There are

1. QuineCALC-(\(k+1\)) queries and
2. Boolean SimpleCALC-(\(k+1\)) queries

that are not \(k\)-counting-only.

Proof.

1. ‘Return \(k + 1\) objects that occur together.’

\[
\{ \langle x_1, \ldots, x_{k+1} \rangle \mid \text{count}(x_1, \ldots, x_{k+1}) \geq 1 \} = \\
\{ \langle x_1, \ldots, x_{k+1} \rangle \mid \exists X \left( \bigwedge_{1 \leq i \leq k+1} \Gamma(x_i, X) \right) \}
\]

2. ‘Does there exist a set with \(k + 1\) objects?’

\[
\exists x_1 \ldots x_{k+1} \left( \left( \bigwedge_{1 \leq i < j \leq k+1} (x_i \neq x_j) \right) \land \text{count}(x_1, \ldots, x_{k+1}) \geq 1 \right).
\]
Proposition

Let $Q(x_1, \ldots, x_m)$ be a SimpleCALC-$k$ query. There exists an $n$ such that

$$\langle o_1, \ldots, o_k \rangle \in [Q]_S \iff \langle o_1, \ldots, o_k \rangle \in [Q]_{S|A}$$

for some $A$ with $|A| \leq n$ and $\{o_1, \ldots, o_k\} \subseteq A$.

Example

‘Return students that take courses with another student.’

$$Q = \{ \langle x \rangle \mid \exists y \exists C \ (\Gamma(x, C) \land \Gamma(y, C)) \}$$
Not all 2-SyCALC queries are in SimpleCALC

‘Is each course followed by a unique student?’

\[ \text{set-ids} = \{\langle\rangle \mid \forall C \exists x (\Gamma(x, C) \land \neg \exists Y ((X \neq Y) \land \Gamma(x, Y)))\} \]

Proposition

Query set-ids is 2-counting-only, but not 1-counting-only.
Not all 2-SyCALC queries are in SimpleCALC

‘Is each course followed by a unique student?’

\[ \text{set-ids} = \{ \langle \rangle \mid \forall C \exists x (\Gamma(x, C) \land \neg \exists Y ((X \neq Y) \land \Gamma(x, Y))) \} \]

Proposition

Query \text{set-ids} \text{ is 2-counting-only, but not 1-counting-only.}

Proof.

Assume we have \( n \) set names. The query \text{set-ids} \ is equivalent to

\[
\{ \langle \rangle \mid \exists x_1 \ldots \exists x_n \left( \bigwedge_{1 \leq i \leq n} \text{count}(x_j) = 1 \right) \land \left( \bigwedge_{1 \leq i < j \leq n} \text{gcount}(x_i; x_j) = 0 \right) \}.
\]
Not all 2-SyCALC queries are in SimpleCALC

‘Is each course followed by a unique student?’

\[
\text{set-ids} = \{\langle \rangle \mid \forall C \exists x (\Gamma(x, C) \land \neg \exists Y ((X \neq Y) \land \Gamma(x, Y)))\}
\]

Proposition

Query \text{set-ids} is 2-counting-only, but not 1-counting-only.

Proof.

Assume we have \( n \) set names. The query \text{set-ids} is equivalent to

\[
\{\langle \rangle \mid \exists x_1 \ldots \exists x_n (\bigwedge_{1 \leq i \leq n} \text{count}(x_j) = 1) \land (\bigwedge_{1 \leq i < j \leq n} \text{gcount}(x_i; x_j) = 0)\}.
\]

Not 1-counting only: 

\[
\begin{array}{c|c}
S_1 & S_2 \\
\hline
A & & \\
B & & \\
\end{array}
\quad
\begin{array}{c|c}
T_1 & T_2 \\
\hline
A & B \\
\end{array}
\]
Decision problems and counting-only queries
Classical decision problems

- Satisfiability.
- Validity.
- Query containment.
- Query equivalence.
Classical decision problems

- Satisfiability.
- Validity.
- Query containment.
- Query equivalence.
Satisfiability and first-order logic

Satisfiability of first-order logic is *not decidable*.

Satisfiability is decidable for FO when *restricted* to

- unary predicates (monadic first-order logic);
- two variables (FO²);
- formulae of the form
  - $\exists \cdots \exists \forall \exists \cdots \exists$ (the Ackermann class);
  - $\exists \cdots \exists \forall \forall \exists \cdots \exists$ (the Gödel class);
  - $\exists \cdots \exists \forall \cdots \forall$ (the Schönfinkel-Bernays class).

What if we restrict FO to counting-only queries?
Satisfiability of 2-SyCALC is undecidable 1/3

FO is undecidable when querying undirected unlabeled graphs without self-loops.

Encode graphs as structures

Encode graph \( G = (V, E) \) as the structure \( \text{enc}(G) = (V, \gamma) \) with

\[
\gamma = \left\{ \left( \{m, n\}, m \right) \middle| (m, n) \in E \right\} \cup \left\{ \left( \{n\}, n \right) \middle| n \in V \right\}.
\]

\( \text{enc}(G) \) always satisfies the SyCALC query:

\[
\text{enc-graph} = \text{set-ids} \land \forall e \exists M \exists N ((\langle M, N \rangle \land \Box (e, M) \land \Box (e, N)) \Rightarrow \forall X ((\langle M, X \rangle \land (N, X)) \Rightarrow \neg \Box (e, X))).
\]
Satisfiability of 2-SyCALC is undecidable 1/3

FO is undecidable when querying undirected unlabeled graphs without self-loops.

Encode graphs as structures
Encode graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ as the structure $\text{enc}(\mathbf{G}) = (\mathbf{V}, \gamma)$ with

$$\gamma = \{(\{m, n\}, m), (\{m, n\}, n) \mid (m, n) \in \mathbf{E}\} \cup \{(\{n\}, n) \mid n \in \mathbf{V}\}.$$  

$\text{enc}(\mathbf{G})$ always satisfies the SyCALC query:

$$\text{enc-graph} = \text{set-ids} \land \
\forall e \exists M \exists N \ (((M \neq N) \land \Gamma(e, M) \land \Gamma(e, N)) \Rightarrow \
\forall X ((M \neq X) \land (N \neq X) \Rightarrow \neg \Gamma(e, X))).$$
Satisfiability of 2-SyCALC is undecidable 2/3

FO is undecidable when querying undirected unlabeled graphs without self-loops.

Encode FO queries as SyCALC queries
Encode Boolean FO query $\phi$ by $\text{enc}(\phi) = \text{enc-}\text{graph} \land \tau(\phi)$ with

$$
\tau(M = N) \equiv M = N; \\
\tau(E(M, N)) \equiv (M \neq N) \land \exists e (\Gamma(e, M) \land \Gamma(e, N)); \\
\tau(e_1 \lor e_2) \equiv \tau(e_1) \lor \tau(e_2); \\
\tau(\neg e) \equiv \neg \tau(e); \\
\tau(\exists N e) \equiv \exists N \tau(e).
$$

Lemma

Let $G$ be a graph and let $\phi$ be a Boolean FO query. Then,

$$[[\phi]]_G = [[\text{enc}(\phi)]]_{\text{enc}(G)}.$$
Satisfiability of 2-SyCALC is undecidable 3/3

FO is undecidable when querying undirected unlabeled graphs without self-loops.

**Proposition**

*If* \( \varphi \) *is a Boolean FO query, then* \( \text{enc}(\varphi) \) *is a 2-SyCALC query.*

**Proof.**

If \( S_1 \) and \( S_2 \) are structures that are 2-counting-equivalent, and \( \llbracket \text{set-ids} \rrbracket_{S_1} = \llbracket \text{set-ids} \rrbracket_{S_2} = \text{true} \), then \( S_1 \) and \( S_2 \) are isomorphic.

**Lemma**

*Let* \( \varphi \) *be a Boolean FO query. If there exists a structure* \( S \) *satisfying* \( \text{enc}(\varphi) \), *then we can construct from* \( S \) *a graph satisfying* \( \varphi \).
FO is undecidable when querying undirected unlabeled graphs without self-loops.

Proposition

If $\phi$ is a Boolean FO query, then $\text{enc}(\phi)$ is a 2-SyCALC query.

Proof.

If $S_1$ and $S_2$ are structures that are 2-counting-equivalent, and $[\text{set-ids}]_{S_1} = [\text{set-ids}]_{S_2} = \text{true}$, then $S_1$ and $S_2$ are isomorphic.

Lemma

Let $\phi$ be a Boolean FO query. If there exists a structure $S$ satisfying $\text{enc}(\phi)$, then we can construct from $S$ a graph satisfying $\phi$.

Theorem

The satisfiability problem is undecidable for 2-SyCALC queries.
What about weaker counting-only languages?

- 1-SyCALC.
- SimpleCALC.
- (QuineCALC).

These classes have the finite model property.
What about weaker counting-only languages?

- 1-SyCALC.
- SimpleCALC.
- (QuineCALC).

These classes have the finite model property.
Satisfiability of 1-SyCALC is decidable 1/2

Definition
Let $k, d \geq 0$. Structures $S_1 = (N_1, \gamma_1)$ and $S_2 = (N_2, \gamma_2)$ are \textit{d-partial $k$-counting-equivalent} if, for every pair of sets of objects $I$ and $E$ with $|I \cup E| \leq k$, either

1. $[\text{gcount}(I; E)]_{S_1} = [\text{gcount}(I; E)]_{S_2} \leq d$; or
2. $d < [\text{gcount}(I; E)]_{S_i} < |N_i| - d$, $i \in \{1, 2\}$,
3. $|N_1| - [\text{gcount}(I; E)]_{S_1} = |N_2| - [\text{gcount}(I; E)]_{S_2} \leq d$.

Lemma
Let $Q$ be a SyCALC query with set name quantifier depth $d$ and let $S_1$ and $S_2$ be $d$-partial $k$-counting-equivalent structures with $k = |\text{adom}(S_1)| = |\text{adom}(S_2)|$. Then $[Q]_{S_1} = [Q]_{S_2}$. 
Satisfiability of 1-SyCALC is decidable 2/2

**Proposition**

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

**Proposition**

Let $\mathbf{Q}$ 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, $[e]_\mathbf{S} \neq \emptyset$ if and only if there exists a structure $\mathbf{S}' = (\mathbf{N}', \gamma')$ with $|\mathbf{N}'| \leq 2d + 1$, $|\text{adom}(\mathbf{S}')| \leq r(2d + 1)$, and $[e]_{\mathbf{S}'} \neq \emptyset$. 

Theorem

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

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

Proposition

Let $\mathcal{Q}$ be a 1-SyCALC query with set name quantifier depth $d$ and object quantifier depth $r$, and let $\mathcal{S} = (N, \gamma)$ be a structure. Then, $\mathcal{S}, \emptyset$ if and only if there exists a structure $\mathcal{S}' = (N', \gamma')$ with $|N'| \leq 2d + 1$, $|\text{dom}(\mathcal{S}')| \leq r(2d + 1)$, and $\mathcal{S}', \emptyset$.

Theorem

The satisfiability problem is decidable for 1-SyCALC queries.
Satisfiability of SimpleCALC is decidable

Proposition (Reminder)

Let $Q(x_1, \ldots, x_m)$ be a SimpleCALC-$k$ query. There exists an $n$ such that

$$\langle o_1, \ldots, o_k \rangle \in [Q]_S \iff \langle o_1, \ldots, o_k \rangle \in [Q]_{S|_A}$$

for some $A$ with $|A| \leq n$ and $\{o_1, \ldots, o_k\} \subseteq A$.

Proposition

Let $S = (N, \gamma)$ be a structure with $|\text{adom}(S)| = z$, and let $d \geq 0$. There exists a structure $S' = (N', \gamma')$ with $|N'| \leq (d + 1) \cdot 2^z$ such that $S$ and $S'$ are $d$-partial z-counting-equivalent structures.
Satisfiability of SimpleCALC is decidable

**Proposition (Reminder)**

Let \( Q(x_1, \ldots, x_m) \) be a SimpleCALC-\( k \) query. There exists an \( n \) such that

\[
\langle o_1, \ldots, o_k \rangle \in [Q]_S \iff \langle o_1, \ldots, o_k \rangle \in [Q]_{S|_A}
\]

for some \( A \) with \( |A| \leq n \) and \( \{o_1, \ldots, o_k\} \subseteq A \).

**Proposition**

Let \( S = (N, \gamma) \) be a structure with \( |\operatorname{adom}(S)| = z \), and let \( d \geq 0 \). There exists a structure \( S' = (N', \gamma') \) with \( |N'| \leq (d + 1) \cdot 2^z \) such that \( S \) and \( S' \) are \( d \)-partial \( z \)-counting-equivalent structures.

**Theorem**

Satisfiability is decidable for SimpleCALC queries, and is \( \text{NEXPTIME} \)-hard for SimpleCALC-\( k \) queries, \( k \geq 2 \).
Conclusion and discussion
An overview of our results

<table>
<thead>
<tr>
<th>QuineCALC</th>
<th>SimpleCALC</th>
<th>c.-o. SyCALC</th>
<th>c.-o. queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3-SyCALC</td>
<td>3-counting-only</td>
</tr>
<tr>
<td>QuineCALC-2</td>
<td>SimpleCALC-2</td>
<td>2-SyCALC</td>
<td>2-counting-only</td>
</tr>
<tr>
<td>QuineCALC-1</td>
<td>SimpleCALC-1</td>
<td>1-SyCALC</td>
<td>1-counting-only</td>
</tr>
<tr>
<td>QuineCALC-0 ≡ SimpleCALC-0 ≡ 0-SyCALC</td>
<td></td>
<td></td>
<td>0-counting-only</td>
</tr>
</tbody>
</table>
An overview of our results

<table>
<thead>
<tr>
<th>First-order definable queries (SyCALC)</th>
<th>c.-o. queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuineCALC</td>
<td>SimpleCALC</td>
</tr>
<tr>
<td>QuineCALC-3</td>
<td>SimpleCALC-3</td>
</tr>
<tr>
<td>QuineCALC-2</td>
<td>SimpleCALC-2</td>
</tr>
<tr>
<td>QuineCALC-1</td>
<td>SimpleCALC-1</td>
</tr>
<tr>
<td>QuineCALC-0 \equiv SimpleCALC-0 \equiv 0-SyCALC</td>
<td>0-counting-only</td>
</tr>
</tbody>
</table>
An overview of our results

<table>
<thead>
<tr>
<th>QuineCALC-0</th>
<th>SimpleCALC-0</th>
<th>0-SyCALC</th>
<th>0-counting-only</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuineCALC-1</td>
<td>SimpleCALC-1</td>
<td>1-SyCALC</td>
<td>1-counting-only</td>
</tr>
<tr>
<td>QuineCALC-2</td>
<td>SimpleCALC-2</td>
<td>2-SyCALC</td>
<td>2-counting-only</td>
</tr>
<tr>
<td>QuineCALC-3</td>
<td>SimpleCALC-3</td>
<td>3-SyCALC</td>
<td>3-counting-only</td>
</tr>
</tbody>
</table>

First-order definable queries (SyCALC)
Future work: generalize bag-of-sets

- ‘Return students that take courses offered by two departments.’
  \[
  \{\langle x \rangle \mid \exists C \exists D_1 \exists D_2 \ (SC(x, C) \land CD(C, D_1) \land CD(C, D_2))\}\]

- ‘Return student-department pairs in which the student only takes courses offered by that department.’
  \[
  \{\langle x, y \rangle \mid |\{z \mid SC(x, z) \land DC(y, z)\}| = |\{z \mid SC(x, z)\}|\}\]
Future work

▶ Are SimpleCALC-\( k \) and ‘\( \text{gcount}(\mathcal{X}; \mathcal{Y}) \geq c \)’-queries equivalent? E.g. ‘Return pairs of distinct students which take the same courses.’

\[
\{ \langle x, y \rangle \mid (x \neq y) \land \text{gcount}(x; y) = \text{gcount}(y; x) = 0 \}
\]

▶ Decision problems: is a SimpleCALC-\( k \) query \( l \)-counting-only? E.g. ‘Are there at least two students taking courses?’

\[
\{ \langle \rangle \mid \exists x \exists y \exists X \exists Y ((x \neq y) \land \Gamma(x, X) \land \Gamma(y, Y)) \}
\]

▶ Possibilities for query optimization?