Adding for-loops to first-order logic*

Frank Neven† Martin Otto Jurek Tyszkiewicz‡
Jan Van den Bussche

revised version, August 2000

Abstract

We study the query language BQL: the extension of the relational algebra with for-loops. We also study FO(FOR): the extension of first-order logic with a for-loop variant of the partial fixpoint operator. In contrast to the known situation with query languages which include while-loops instead of for-loops, BQL and FO(FOR) are not equivalent. Among the topics we investigate are: the precise relationship between BQL and FO(FOR); inflationary versus non-inflationary iteration; the relationship with logics that have the ability to count; and nested versus unnested loops.

1 Introduction

Much attention in database theory (or finite model theory) has been devoted to extensions of first-order logic as a query language [AHV95, EF95]. A seminal paper in this context was that by Chandra in 1981 [Cha81], where he added various programming constructs to the relational algebra and compared the expressive power of the various extensions thus obtained. One

*A preliminary version of this paper was presented at the 7th International Conference on Database Theory, Jerusalem, 1999.

†Research Assistant of the Fund for Scientific Research, Flanders.

‡Research begun at RWTH Aachen, supported by a German Research Council DFG grant, continued at the University of Warsaw, supported by the Polish Research Council KBN grant 8 T11C 002 11, and on leave at the University of New South Wales, supported by the Australian Research Council ARC grant A 49800112 (1998–2000).

such extension is the language that we denote here by BQL: a programming-language-like query language obtained from the relational algebra by adding *assignment statements*, *composition*, and *for-loops*. Assignment statements assign the result of a relational algebra expression to a relation variable; composition is obvious; and for-loops allow a subprogram to be iterated exactly as many times as the cardinality of the relation stored in some variable.

For-loops of this kind have since received practically no attention in the literature. In contrast, two other iteration constructs, namely least or inflationary fixpoints, and while-loops, have been studied extensively. In the present paper we take some steps towards the goal of understanding for-loops in query languages as well as fixpoints and while-loops are understood.

The variant of BQL with while-loops instead of for-loops, called RQL, was introduced by Chandra and Harel [CH82]. In the same paper these authors also introduced, in the context of query languages, the extension of first-order logic with the least fixpoint operator; we denote this logic here by FO(LFP). One can also use a *partial* fixpoint operator to obtain a logic, called FO(PFP), with the same expressive power as RQL [AV91].

Here, we introduce the FOR operator, which iterates a formula (called the "body formula") precisely as many times as determined by the cardinality of the relation defined by another formula (called the "head formula"). In contrast to the equivalence of RQL and FO(PFP), FO(FOR) is not equivalent to, but strictly stronger than, BQL. The reason for this turns out to be the presence of free variables in the head formula, acting as parameters; the restriction of FO(FOR) that disallows such parameters is equivalent to BQL.

The question whether FO(LFP) is strictly weaker than FO(PFP) is a famous open problem, since Abiteboul and Vianu showed that it is equivalent to whether PTIME is strictly contained in PSPACE [AV95]. In FO(LFP) we can equivalently replace the least fixpoint operator by the *inflationary* fixpoint operator IFP. So the PTIME versus PSPACE question is one of inflationary versus non-inflationary iteration. Since the FOR operator is non-inflationary in nature, one may wonder about the expressive power of the inflationary version of FOR, which we call IFOR. We show that FO(IFOR) lies strictly between FO(IFP) and FO(FOR). Since in FO(FOR) we can define parity, FO(FOR) is not subsumed by FO(PFP), and conversely FO(PFP) can only be subsumed by FO(FOR) if PSPACE equals PTIME, since FO(PFP) equals PSPACE on ordered structures and FO(FOR) is contained in PTIME.

A natural question is how FO(FOR) relates to FO(IFP, #), the extension of FO(IFP) with counting. Actually, FO(FOR) is readily seen to be subsumed by FO(IFP, #). We show that this subsumption is strict, by showing that one cannot express in FO(FOR) that two sets have the same cardinality.[1] We also show that the restriction of FO(IFP, #) that allows modular counting only, is strictly subsumed by FO(FOR).

The main technical question we focus on in this paper is that of nesting of for-loops. It is known that nested applications of while-loops in RQL, or of the PFP operator in FO(PFP), do not yield extra expressive power; a single while-loop or PFP operator suffices [EF95]. In the case of BQL, however, we show that nesting does matter, albeit only in a limited way: one level of nesting already suffices. In the case of FO(FOR), there are two kinds of nesting of the FOR operator: in body formulas, and in head formulas. Regarding bodies, we show that nested applications of the FOR operator in body formulas again do matter, although here we do not know whether nesting up to a certain level is sufficient. Regarding heads, we show that the restriction of FO(FOR) that allows only head formulas that are "pure" first-order, is weaker than full FO(FOR). By "pure" we mean that the formula cannot mention relation variables from surrounding FOR operators; from the moment this is allowed, we are back to full FO(FOR).

This paper is further organized as follows. After recalling the definitions of pebble games and fixpoint logic in Section 2, we define BQL and FO(FOR) and investigate their interrelationship and relationship with other logics in Section 3. In Section 4, we examine the nesting of for-loops in both BQL and FO(FOR). We end with a discussion in Section 5.

2 Preliminaries

Throughout the paper we use the terminology and notation of mathematical logic [EFT94]. For background on database theory we refer to Abiteboul, Hull, and Vianu [AHV95], and for finite model theory to Ebbinghaus and Flum [EF95], Immerman [Imm98], and Otto [Ott97].

A *relational vocabulary* τ is what in the field of databases is known as a relational schema; a *structure* over τ is what is known as an instance of that schema with an explicit domain. (Structures are always assumed to be finite in this paper.) We denote the domain of a τ-structure $\mathcal{A}$ by A, and the interpretation of the relation symbol R in $\mathcal{A}$ by $R^{\mathcal{A}}$.

[1] The analogous result for BQL (which is weaker than FO(FOR)) was stated by Chandra in the early eighties [Cha81, Cha88], but no proof has been published.

A k*-ary query* $\mathcal{Q}$ is a computable function that maps each τ-structure $\mathcal{A}$ to a subset of A^k, such that if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic via π then $\pi(\mathcal{Q}(\mathcal{A})) = \mathcal{Q}(\mathcal{B})$. We also call a nullary query a *Boolean* query.

The query language of first-order logic (the relational calculus) is denoted by FO. For any natural number k, FO^k denotes the k-variable fragment of FO, i.e., the set of FO formulas that use only the variables $\{x_1, \dots, x_k\}$.

The logic $\mathcal{L}^k_{\infty\omega}$ is defined as follows: (i) it contains all FO^k formulas; (ii) if φ is an $\mathcal{L}^k_{\infty\omega}$ formula so are $\neg\varphi$ and $(\exists x_i)\varphi$ for each $i = 1, \dots, k$; (iii) if Φ is a set of $\mathcal{L}^k_{\infty\omega}$ formulas then $\bigvee \Phi$ is an $\mathcal{L}^k_{\infty\omega}$ formula.

The semantics of $\mathcal{L}^k_{\infty\omega}$ is a direct extension of the semantics of first-order logic with $\bigvee \Phi$ being interpreted as the disjunction over all formulas in Φ; hence, neglecting the interpretation of the free variables,

$$\mathcal{A} \models \bigvee \Phi \qquad \Leftrightarrow \qquad \text{for some } \varphi \in \Phi,\ \mathcal{A} \models \varphi.$$

Let $\mathcal{A}$ and $\mathcal{B}$ be two structures, and let $\bar{a} = a_1, \dots, a_\ell$ and $\bar{b} = b_1, \dots, b_\ell$ be sequences of elements of A and B respectively with $\ell \leq k$. If for every $\mathcal{L}^k_{\infty\omega}$ formula $\varphi(\bar{x})$, $\mathcal{A} \models \varphi[\bar{a}]$ if and only if $\mathcal{B} \models \varphi[\bar{b}]$, then we say that $(\mathcal{A}, \bar{a})$ and $(\mathcal{B}, \bar{b})$ are k*-equivalent*. This k-equivalence can be nicely characterized by pebble games. The k-pebble game is a game with infinitely many rounds played by two players, the *Spoiler* and the *Duplicator*, on two structures $\mathcal{A}$ and $\mathcal{B}$ in the following way. Each structure has k pebbles numbered from 1 to k. Initially, pebble i is on element a_i in $\mathcal{A}$ and on element b_i in $\mathcal{B}$ for $i = 1, \dots, \ell$. In each round the Spoiler chooses a structure, say $\mathcal{A}$, picks up one of its k pebbles, say i, and places it on an element $a \in A$. The Duplicator then answers by placing pebble i of $\mathcal{B}$ on an element $b \in B$. The Spoiler wins the game if the mapping $\bar{a}' \to \bar{b}'$, where $\bar{a}'$ and $\bar{b}'$ are the current pebbled elements, is not a partial isomorphism between $\mathcal{A}$ and $\mathcal{B}$. We say that the Duplicator *wins* the k-pebble game on $\mathcal{A}$ and $\mathcal{B}$ if the Duplicator has a strategy preventing the Spoiler from winning.

A proof of the next proposition can, e.g., be found in Ebbinghaus and Flum's book [EF95].

Proposition 2.1 *Two structures $(\mathcal{A}, \bar{a})$ and $(\mathcal{B}, \bar{b})$ are k-equivalent if and only if the Duplicator wins the k-pebble game on $\mathcal{A}$ and $\mathcal{B}$.*

Let us briefly recall the syntax and semantics of FO(PFP) and FO(IFP). Let $\varphi(\bar{x}, \bar{y}, X, \overline{Y})$ be an FO formula over $\tau \cup \{X, \overline{Y}\}$, where X is an n-ary relation variable, $\bar{x}$ is of length n, and $\overline{Y}$ is a tuple of relation variables.

On any τ-structure $\mathcal{A}$ expanded with interpretations for the first-order and relational parameters $\bar{y}$ and $\overline{Y}$, φ defines the stages $\varphi^0(\mathcal{A}) := \emptyset$ and $\varphi^i(\mathcal{A}) := \{\bar{a} \mid \mathcal{A} \models \varphi[\bar{a}, \varphi^{i-1}(\mathcal{A})]\}$ for each $i > 0$. If there exists an i_0 such that $\varphi^{i_0}(\mathcal{A}) = \varphi^{i_0+1}(\mathcal{A})$, then we say that *the partial fixpoint of φ on $\mathcal{A}$ exists*, and define it to be $\varphi^{i_0}(\mathcal{A})$; otherwise we define it as the empty set. We obtain FO(PFP) by augmenting FO with the rule $[\mathrm{PFP}_{\bar{x},X}\varphi](\bar{t})$, which expresses that $\bar{t}$ belongs to the partial fixpoint of φ. For FO(IFP) we consider the stages $\tilde{\varphi}^0(\mathcal{A}) := \emptyset$ and $\tilde{\varphi}^i(\mathcal{A}) := \tilde{\varphi}^i(\mathcal{A}) \cup \{\bar{a} \mid \mathcal{A} \models \varphi[\bar{a}, \tilde{\varphi}^{i-1}(\mathcal{A})]\}$, for each $i > 0$. Here, there always exists an i_0 such that $\tilde{\varphi}^{i_0}(\mathcal{A}) = \tilde{\varphi}^{i_0+1}(\mathcal{A})$. We call $\tilde{\varphi}^{i_0}(\mathcal{A})$ *the inflationary fixpoint of φ on $\mathcal{A}$*. We obtain FO(IFP) by augmenting FO with the rule $[\mathrm{IFP}_{\bar{x},X}\varphi](\bar{t})$, which expresses that $\bar{t}$ belongs to the inflationary fixpoint of φ.

3 Query languages with for-loops

3.1 BQL

Let τ be a vocabulary. The set of BQL *programs over* τ is inductively defined as follows:

(i) if X is a relation variable of arity n and $\varphi(x_1, \ldots, x_n, \overline{X})$ is an FO formula over the vocabulary τ and the relation variables $\overline{X}$, then the assignment

$$X := \{\bar{x} \mid \varphi(\bar{x}, \overline{X})\}$$

is a BQL program;[2]

(ii) if P_1 and P_2 are BQL programs then the composition

$$P_1; P_2$$

is a BQL program; and

(iii) if P is a BQL program and X is a relation variable, then the for-loop

$$\textbf{for } |X| \textbf{ do } P \textbf{ od}$$

is a BQL program.

[2]Although Chandra's BQL is an extension of the relational algebra, we use FO which is known to be equivalent to the former [AHV95].

The semantics of BQL programs of the form (i) or (ii) is defined in the obvious way; for BQL programs of the form (iii) the subprogram P is iterated as many times as the cardinality of the relation stored in variable X prior to entering the loop. We now define this formally. Assume given an infinite set $\mathcal{X}$ of relation variables. We denote the arity of a variable X by $arity(X)$. For a BQL program P, we denote the set of variables occurring in P by $var(P)$. Further, a *valuation* v on a structure $\mathcal{A}$ is a mapping from $\mathcal{X}$ to relations over A such that $v(X) \in A^{arity(X)}$ for all X. Usually, we are only interested in valuations w.r.t. the variables occurring in a specific program. To emphasize this, we sometimes say P-valuation rather than just valuation. Given an initial valuation η on $\mathcal{A}$, the program P determines a final valuation $val[P, A, \eta]$ defined inductively as follows:

(i) If P is of the form $Y := \{\bar{x} \mid \varphi(\bar{x}, \overline{Y})\}$, then define

$$val[P, \mathcal{A}, \eta](X) \;:=\; \begin{cases} \{\bar{a} \mid \mathcal{A} \models \varphi[\bar{a}, \eta]\} & \text{if } Y = X; \\ \eta(X) & \text{otherwise.} \end{cases}$$

We abuse notation and let $\varphi[\bar{a}, \eta]$ denote the formula where each relation variable Y' in $\overline{Y}$ is interpreted by $\eta(Y')$.

(ii) If P is of the form $P_1; P_2$, then define

$$val[P_1; P_2, \mathcal{A}, \eta] \;:=\; val[P_2, \mathcal{A}, val[P_1, \mathcal{A}, \eta]].$$

(iii) If P is of the form **for** $|X|$ **do** P' **od**, then define

$$val[P, \mathcal{A}, \eta] := val[P^{(m)}, \mathcal{A}, \eta],$$

with m the number of tuples in the relation $\eta(X)$ and where, for $i > 0$,

$$\begin{aligned} val[P^{(0)}, \mathcal{A}, \eta] &:= \eta \\ val[P^{(i)}, \mathcal{A}, \eta] &:= val[P', \mathcal{A}, val[P^{(i-1)}, \mathcal{A}, \eta]]. \end{aligned}$$

A query $\mathcal{Q}$ is *expressible* in BQL if there exists a BQL program P and a variable $X \in var(P)$ such that for every structure $\mathcal{A}$, $\mathcal{Q}(\mathcal{A}) = val[P, \mathcal{A}, \eta_\emptyset](X)$, where $\eta_\emptyset$ denotes the valuation that maps each variable to the empty relation. We refer to X as the *output variable* of P. To express Boolean queries in BQL, we adopt the convention that the result of the query is true if and only if X is non-empty.

Example 3.1 Let $\tau_G = \{E\}$ be the vocabulary of graphs; so E is the binary edge relation. Consider the following BQL programs:

$$X := \{x \mid x = x\};\ Y := \emptyset;\ \mathbf{for}\ |X|\ \mathbf{do}\ Y := \{x \mid \neg Y(x)\}\ \mathbf{od},$$

$$X := E;\ \mathbf{for}\ |X|\ \mathbf{do}\ X := \{(x,y) \mid X(x,y) \vee (\exists z)(X(x,z) \wedge E(z,y))\}\ \mathbf{od}.$$

The first program computes, in variable Y, the parity of the domain, and the second program computes the transitive closure of E. ■

By a standard technique [CH82], one can simulate every FO(IFP) formula by a BQL program. It is well known that it is not expressible in FO(PFP) whether the cardinality of a set is even. Hence, since we just saw in the above example that this is expressible in BQL, FO(PFP) does not subsume BQL and BQL strictly subsumes FO(IFP). Furthermore, since all BQL queries are clearly PTIME and FO(IFP) captures PTIME on ordered structures [Imm86, Var82], BQL does the same.

For later use, we also briefly recall the syntax and semantics of RQL [CH82]. An RQL *program* is inductively defined in the same way as a BQL program with (iii) replaced by

(iii) if P' is an RQL program then $P \equiv \mathbf{while}\ X \neq \emptyset\ \mathbf{do}\ P'\ \mathbf{od}$ is an RQL program.

For each natural number i, the mapping $val[P^{(i)}, \mathcal{A}, \eta]$ is as defined above. The semantics of P is now defined as the mapping

$$val[P, \mathcal{A}, \eta] := val[P^{(m)}, \mathcal{A}, \eta],$$

where m is the smallest natural number such that $val[P^{(m)}, \mathcal{A}, \eta](X) = \emptyset$. If such an m does not exist then $val[P, \mathcal{A}, \eta]$ is defined as the empty set.

3.2 FO(FOR)

We next introduce the logic FO(FOR). The crucial construct in the formation of FO(FOR) formulas is the following. Suppose $\varphi(\bar{x}, \bar{y}, X, \overline{Y})$ and $\psi(\bar{z}, \bar{u}, \bar{y}, \overline{Y})$ are formulas and $\bar{x}$, $\bar{u}$ and X are of the same arity. Then the following FO(FOR) formula ξ is obtained from ψ and φ through the FOR-constructor:

$$\xi(\bar{u}, \bar{y}, \overline{Y}) \quad := \quad [\mathrm{FOR}^{\#\bar{z}:\psi}_{\bar{x},X}\varphi](\bar{u}).$$

The formula ψ is called the *head formula*, and φ is called the *body formula* of ξ. For each τ-structure $\mathcal{A}$ expanded with interpretations for the parameters $\bar{y}$ and $\overline{Y}$, and for any tuple of elements $\bar{a}$: $\mathcal{A} \models \xi[\bar{a}]$ if and only if $\bar{a} \in \varphi^m(\mathcal{A})$ where m equals the cardinality of the set $\{\bar{c} \mid \mathcal{A} \models \psi[\bar{c}, \bar{a}]\}$. Here $\varphi^m(\mathcal{A})$ is as defined in Section 2.

Example 3.2 Consider the following FO(FOR) formulas over τ_G:

$$\xi_1(u, v) \quad \equiv \quad [\mathrm{FOR}^{\#s,t:E(s,t)}_{x,y,X} E(x, y) \vee (\exists z)(X(x, z) \wedge E(z, y))](u, v)$$

and

$$\xi_2(x) \quad \equiv \quad \neg(\exists z)[\mathrm{FOR}^{\#y:E(x,y)}_{z,Z} \neg Z(z)](z).$$

The formula ξ_1 defines the transitive closure of E, and ξ_2 expresses that vertex x has even outdegree. ■

We now define simultaneous FO(FOR) which allows the simultaneous iteration of body formulas of for-loops. As is the case for fixpoint logic, this does not increase the expressiveness of the formalism. If $\varphi_1(\bar{x}_1, \bar{y}, \overline{Y}, X_1, \ldots, X_n)$, $\ldots$, $\varphi_n(\bar{x}_n, \bar{y}, \overline{Y}, X_1, \ldots, X_n)$ is a system of formulas where X_i and $\bar{x}_i$ are of the same arity, for each $i = 1, \ldots, n$, then

$$\xi(\bar{u}, \bar{y}, \overline{Y}) \quad := \quad [\text{S-FOR}^{\#\bar{z}:\psi}_{\bar{x}_1,X_1,\ldots,\bar{x}_n,X_n} \varphi_1, \ldots, \varphi_n](\bar{u})$$

is an FO(S-FOR) formula. On a structure $\mathcal{A}$ expanded with interpretations for the first-order and relational parameters $\bar{y}$ and $\overline{Y}$, for $j = 1, \ldots, n$, consider the stages defined by

$$\begin{aligned} \varphi_j^0(\mathcal{A}) &:= \emptyset; \\ \varphi_j^{i+1}(\mathcal{A}) &:= \{\bar{a} \mid \mathcal{A} \models \varphi_j[\bar{a}, \varphi_1^i(\mathcal{A}), \ldots, \varphi_n^i(\mathcal{A})]\}. \end{aligned}$$

Then $\mathcal{A} \models \xi[\bar{a}]$ if and only if $\bar{a} \in \varphi_1^m(\mathcal{A})$ where m equals the cardinality of the set $\{\bar{c} \mid \mathcal{A} \models \psi[\bar{c}, \bar{a}]\}$.

Each FO(S-FOR) formula can be transformed into an FO(FOR) formula by applying the usual encoding of the variables $X_1, \ldots, X_n$ into one relation variable of larger arity [EF95].

Proposition 3.3 *Each* FO(S-FOR) *formula is equivalent to an* FO(FOR) *formula.*

Proof. Consider the FO(S-FOR) formula ξ defined above. For $i = 1, \ldots, n$, let the arity of $\bar{x}_i$ be k_i and let $k = \max\{k_i \mid i \in \{1, \ldots, n\}\} + n$.

Define $\varphi'(z_1, \ldots, z_k, Z, \bar{y}, \overline{Y})$ as

$$(\exists v)(\exists w) v \neq w \wedge \left(\begin{array}{c} (\varphi'_1(z_1, \ldots, z_{k_1}, \bar{y}, \overline{Y}, Z) \wedge \delta_1(z_1, \ldots, z_k, v, w)) \\ \vee (\varphi'_2(z_1, \ldots, z_{k_2}, \bar{y}, \overline{Y}, Z) \wedge \delta_2(z_1, \ldots, z_k, v, w)) \\ \vdots \\ \vee (\varphi'_n(z_1, \ldots, z_{k_n}, \bar{y}, \overline{Y}, Z) \wedge \delta_n(z_1, \ldots, z_k, v, w)) \end{array} \right).$$

In the above, for each i, the formula φ'_i is obtained from φ_i by replacing any occurrence of $X_j(\bar{t})$ by

$$Z(\bar{t}, \underbrace{v, \ldots, v}_{k-k_j-j \text{ times}}, \underbrace{w, \ldots, w}_{j \text{ times}}),$$

and $\delta_i(z_1, \ldots, z_k, v, w)$ is the formula

$$z_{k_i+1} = \ldots = z_{k-i} = v \wedge z_{k-i+1} = \ldots = z_k = w.$$

Then ξ is equivalent to

$$(\exists v)(\exists w) \left(v \neq w \wedge [\mathrm{FOR}^{\#\bar{z}:\psi}_{\bar{z},Z} \varphi'](\bar{u}, v, \ldots, v, w) \right).$$

This simulation only works for structures that contain at least two elements. One-element structures can be treated separately because, up to isomorphism, there are only a finite number of them. ■

Clearly, all queries definable in FO(FOR) are in PTIME. We show in Section 3.5 that there are PTIME queries that are not definable in FO(FOR). However, for every PTIME query $\mathcal{Q}$ on graphs there is a formula $\varphi \in$ FO(FOR) such that $\mathcal{Q}(G) \neq \varphi(G)$ for a vanishingly small fraction of n element graphs G. Indeed, Hella, Kolaitis and Luosto showed that a canonical ordering is definable on almost all graphs in FO(IFP) plus the even quantifier [HKL96]. For future reference we call the query that expresses this ordering the *HKL query*; it will be used extensively in Section 4.2. Since we saw in the last example that the even quantifier is expressible in FO(FOR), the HKL query is also expressible in FO(FOR). Since FO(FOR) can also easily simulate FO(IFP), FO(FOR) thus captures PTIME on almost all graphs.

3.3 BQL versus FO(FOR)

We now show that every BQL query is definable in FO(FOR).

Proposition 3.4 *Every query expressible in* BQL *is definable in* FO(FOR).

Proof. The proof proceeds by induction on the structure of BQL programs. Let P be a BQL program. For each $X \in \mathit{var}(P)$ we construct an FO(FOR) formula $\varphi_X^P(\bar{x}, \overline{X})$, such that for each τ-structure $\mathcal{A}$ and P-valuation η,

$$\mathit{val}[P, \mathcal{A}, \eta](X) \quad = \quad \{\bar{a} \mid \mathcal{A} \models \varphi_X^P[\bar{a}, \eta]\}.$$

Here, $\overline{X}$ is an enumeration of $\mathit{var}(P)$. This proves the proposition, since the query expressed by P in output variable X equals $\{\bar{a} \mid \bar{a} \in \mathit{val}[P, \mathcal{A}, \eta_\emptyset](X)\}$, which by the above equals $\{\bar{a} \mid \mathcal{A} \models \varphi_X^P[\bar{a}, \eta_\emptyset]\}$. Note that substituting the empty predicate for a relation symbol corresponds to making it false.

We can assume that a variable that appears in the head of a for-loop, never appears in the left-hand side of an assignment in the body of this for-loop. Indeed, consider the following program

P_1;
for $|Y|$ **do**
    P_2
od;
P_3,

where P_1, P_2, and P_3 are BQL programs. If Y occurs in the left-hand side of an assignment in P_2 then we modify the above program into

P_1;
$Z := Y$;
for $|Y|$ **do**
    P_2'
od;
$Y := Z$;
P_3,

where Z is a variable not occurring in P_1, P_2 or P_3, and P_2' is obtained from P_2 by replacing each occurrence of Y by Z.

1. If P is of the form $Y := \{\bar{x} \mid \varphi(\bar{x}, \overline{X})\}$, then define for each $X \in var(P)$
$$\varphi_X^P := \begin{cases} \varphi(\bar{x}, \overline{X}) & \text{if } X = Y; \\ X(x_1, \ldots, x_{arity(X)}) & \text{otherwise.} \end{cases}$$

2. If P is of the form $P_1; P_2$, then we can assume, w.l.o.g, that $\varphi_X^{P_1}$ and $\varphi_Z^{P_2}$ have no first-order variables in common. For each X in $var(P_2)$ define φ_X^P as the formula obtained from $\varphi_X^{P_2}$ by replacing any occurrence of an atomic formula $Y(\bar{y})$, where Y in $var(P_1)$, by the formula $\varphi_Y^{P_1}(\bar{y})$. Define φ_X^P as $\varphi_X^{P_1}$ for each $X \in var(P_1) - var(P_2)$.

3. If P is of the form **for** $|Y|$ **do** P' **od**, then we assume, w.l.o.g, that no $\varphi_X^{P'}$ and $\varphi_Z^{P'}$ have first-order variables in common, and that Y does not appear in the left-hand side of an assignment in P'.

   Define φ_Y^P as $Y(x_1, \ldots, x_{arity(Y)})$, and define for each $X \in var(P) - \{Y\}$
   $$\varphi_X^P(\bar{x}) \quad := \quad (Y = \emptyset \wedge X(\bar{x})) \vee (Y \neq \emptyset \wedge \rho_X(\bar{x})),$$
   where the ρ_X are as follows. Let $var(P) := \{Y, X_1, \ldots, X_n\}$. For each $i = 1, \ldots, n$, define
   $$\rho_{X_i}(\bar{x}_i) \ := \ (\exists u)[\text{S-FOR}^{\#\bar{y}:Y(\bar{y})}_{\bar{x}_i u_i v_i, X_i', (\bar{x}_j u_j v_j, X_j')_{1 \leq j \neq i \leq n}} \tau_i, (\tau_j)_{1 \leq j \neq i \leq n}](\bar{x}_i, u, u),$$
   where for $\ell = 1, \ldots, n$
   $$\begin{aligned} \tau_\ell(\bar{x}_\ell, u_\ell, v_\ell, \bar{X}') \ := \ & (X_\ell' = \emptyset \wedge (u_\ell = v_\ell \rightarrow \varphi_{X_\ell}^{P'}(\bar{x}_\ell))) \\ & \vee (X_j' \neq \emptyset \wedge (u_\ell = v_\ell \rightarrow \alpha_\ell)). \end{aligned}$$
   Here, α_ℓ is the formula obtained from $\varphi_{X_\ell}^{P'}$ by replacing each occurrence of an atomic formula $Z(\bar{z})$ by the formula $(\exists v) Z'(\bar{z}, v, v)$ for each variable $Z \in \{X_1, \ldots, X_n\}$. We assume w.l.o.g. that no $\varphi_X^{P'}$ contains the variable v. The arity of each X_i' is two more than the arity of X_i. The reason we do it like this, is that a for-loop in the logic always starts with the empty relation for each relation variable, while the variables in a for-loop in the algebra are initialized. Hence, in the simulation we initialize each variable X with the value it has after one iteration of P'. For this we use a tagging technique. The two extra columns make sure that each X_i' is empty only once, namely, in the first iteration.

   This simulation only works for structures with at least two elements. One-element structures can be treated separately because, up to isomorphism, there are only a finite number of them.

■

The converse of Proposition 3.4 does not hold. For non-Boolean queries this is readily seen. The relation variables in a BQL program always hold relations that are closed under indistinguishability in first-order logic with a fixed number of variables. To see this, let P be a BQL program. We choose k to be the maximum of all arities of variables X in P and of the number of distinct first-order variables used in assignment statements in P (free or bound). W.l.o.g, we assume that all atoms $X(y_1, \dots, y_\ell)$ are such that $y_1, \dots, y_\ell$ are pairwise distinct. E.g., $X(y_1, y_1, y_3)$ can be replaced with $(\exists y_3)(X(y_1, y_3, y_2) \wedge y_3 = y_1)$. We introduce some more notationn. If φ is an FO^k formula and $\pi : \{1, \dots, k\} \to \{1, \dots, k\}$ is a bijection, then φ^π denotes the formula obtained from φ by replacing, for each i, every occurrence of x_i (both free and bound) by $x_{\pi(i)}$.

Now, on any structure $\mathcal{A}$, all the for-loops of P can be unfolded into a sequence of assignment statements. This sequence can now be transformed into an equivalent FO^k formula defining the output variable by applying iteratively the following operation: replace the subprogram

$$X := \varphi(x_1, \dots, x_m, \overline{X});\ Y := \psi(x_1, \dots, x_n, \overline{X})$$

by the assignment

$$Y := \psi'(x_1, \dots, x_n, \overline{X}),$$

where ψ' is obtained from ψ by replacing each occurrence of an atomic formula $X(x_{i_1}, \dots, x_{i_m})$ by $\varphi^\pi(x_{i_1}, \dots, x_{i_m})$. Here, $\pi : \{1, \dots, k\} \to \{1, \dots, k\}$ is an arbitrary bijection such that for each $j = 1, \dots, m$, $\pi(j) = i_j$.

The query defined by ξ_2 in Example 3.2, however, is not closed under FO^k-indistinguishability for any k. Indeed, let $\mathcal{G}_k$ be the graph depicted in Figure 1. Clearly, the Duplicator can answer any move of the Spoiler in the k-pebble game played on $(\mathcal{G}_k, p)$ and $(\mathcal{G}_k, p')$. Hence, no FO^k formula can distinguish the node p from node p'. They are, however, clearly distinguished by the formula ξ_2 from Example 3.2 .

To separate BQL from FO(FOR) with a Boolean query, we need to do more work. Let $\mathcal{Q}_1$ be the query *Is there a node with even outdegree?* This query is definable in FO(FOR) by the sentence $(\exists x)\xi_2(x)$. However, we show that this query is not expressible in BQL. To this end we introduce some machinery that will prove to be useful in Section 3.5 as well.

One way to carry out separation or inexpressibility proofs w.r.t. FOR-constructs is to use structures which are simple enough to have few definable

$$\begin{array}{ccc} & p & \\ \swarrow & \cdots & \searrow \\ c_1 & & c_k \end{array} \qquad\qquad \begin{array}{ccc} & p' & \\ \swarrow & \cdots & \searrow \\ c'_1 & & c'_{k+1} \end{array}$$

Figure 1: The graph $\mathcal{G}_k$.

predicates (finitely many in each arity), whose cardinalities moreover can be succinctly described in terms of basic cardinality invariants.

A *complete atomic k-type* over a relational vocabulary τ and in variables $x_1, \ldots, x_k$ is a maximally consistent collection of atoms and negated atoms in the language τ in the given variables. As all types to be considered here will be atomic, we simply refer those as *types*. Such types correspond to complete descriptions of the isomorphism type of a k-tuple of elements in a τ-structure. As τ is always finite for our considerations, each type may be identified with a single quantifier-free formula, namely the conjunction of all members of that type. Thinking of τ and k as fixed, we let Atp stand for the finite set of all atomic k-types.

A *partial* atomic *k-type* corresponds to a partial description of an isomorphism type of an k-tuple. It may be formalized as an arbitrary subset of Atp, or as an arbitrary quantifier-free formula. The translation between these two formalizations is obvious: a formula corresponds to the set of all complete types compatible (i.e., logically consistent) with it, and a set of complete types corresponds to the disjunction over its members. E.g., for the language of one unary predicate P, the partial 2-type characterized by the formula $x_1 = x_2$ corresponds to the set of complete 2-types $\{\{x_1 = x_2, Px_1, Px_2\}, \{x_1 = x_2, \neg Px_1, \neg Px_2\}\}$, and is also characterized by the formula $(x_1 = x_2 \wedge Px_1 \wedge Px_2) \vee (x_1 = x_2 \wedge \neg Px_1 \wedge \neg Px_2)$ which is logically equivalent to $x_1 = x_2$. We want to admit the empty subset of Atp as a partial type, corresponding to any unsatisfiable formula (e.g., $\neg x_1 = x_1$). Let $\Theta = \mathcal{P}(\mathrm{Atp})$ (power set of Atp) be the set of partial types. A tuple $\bar{a}$ in $\mathcal{A}$ *realizes* a type θ iff $\mathcal{A} \models \theta[\bar{a}]$. We write $\mathrm{atp}_{\mathcal{A}}(\bar{a})$ for the complete type realized by $\bar{a}$ in $\mathcal{A}$. Over a particular structure $\mathcal{A}$ one often identifies a type with the set of those tuples that realize it:

$$\theta[\mathcal{A}] = \{\bar{a} \mid \mathcal{A} \models \theta[\bar{a}]\}.$$

The following is an ad-hoc definition for the purposes of our separation proofs.

Definition 3.5 Call a τ-structure $\mathcal{A}$ *simple* if its automorphism group acts transitively on its atomic types, i.e., if for any two $\bar{a}$ and $\bar{a}'$ in $\mathcal{A}$ which realize the same complete atomic type, there is an $\mathcal{A}$-automorphism taking $\bar{a}$ to $\bar{a}'$.

As definable predicates over any structure (definable in any reasonable logic) are necessarily closed under automorphisms, it is obvious that any definable predicate over a simple structure $\mathcal{A}$ must be a union of sets $\theta_i[\mathcal{A}]$ for some complete types θ_i, i.e., a set $\theta[\mathcal{A}]$ for some partial type $\theta \in \Theta$. Indeed, if $R \subset A^k$ is definable (in any logic), then

$$R = \theta[\mathcal{A}] \quad \text{where} \quad \theta = \bigvee\nolimits_{\bar{a} \in R} \mathrm{atp}_{\mathcal{A}}(\bar{a}).$$

We now apply this to the study of the semantics of BQL programs over simple structures. As the valuation transformation induced by a program passes from definable predicates to definable predicates, its semantics can actually be described as a mapping on partial types. But first we have to normalize the given program so that all its effects can be described in one and the same arity k. Given P we choose k to be the maximum of all arities of variables X in P and of the number of distinct first-order variables used in assignment statements in P (free or bound). It is checked that the semantics of P remains essentially unchanged if we replace any variable X of P whose arity is $s < k$ by a padded version X' or arity k, whose intended interpretation (during all stages of the evaluation of P) is

$$X' = \{(x_1, \ldots, x_s, x_{s+1}, \ldots, x_k) | (x_1, \ldots, x_s) \in X, x_s = x_{s+1} = \ldots = x_k\}.$$

In order to force this interpretation, we need merely replace any assignment $X := \{\bar{x} \mid \varphi(\bar{x})\}$ in P by the modified assignment $X' := \{\bar{x} \mid \varphi(\bar{x}) \wedge \bigwedge_{s \leq j \leq k} x_s = x_j\}$, and any $X(\bar{x})$ in a formula by $(\exists x_s) \ldots (\exists x_k)(X(\bar{x}) \wedge \bigwedge_{s \leq j \leq k} x_s = x_j)$. Note in particular that the replacement of X by X' preserves the cardinality of X, so that the use of X' in place of X in FOR-instructions is unproblematic.

From now on we assume that all variables in P are k-ary, and that no assignment statement in P uses first-order variables other than $x_1, \ldots, x_k$. Let $\overline{X} = (X_1, \ldots, X_l)$ list the variables of P, so that the semantics of P is formalized over $\mathcal{A}$ as a mapping

$$P \colon \big(\mathcal{P}(A^k)\big)^l \to \big(\mathcal{P}(A^k)\big)^l,$$

from an initial valuation to the final valuation for $\overline{X}$.

Definition 3.6 Let $\mathcal{A}$ be simple. Then $R \subset A^k$ is *admissible* if it is closed under automorphisms of $\mathcal{A}$, i.e., if $R = \theta[\mathcal{A}]$ for some partial type $\theta \in \Theta$. A valuation η on $\mathcal{A}$ is admissible if all $\eta(X)$ are admissible.

Note that any BQL program P can only produce admissible valuations (final or intermediate) when initialized with an admissible valuation of its variables. As the default valuation $\eta_\emptyset$ in particular is admissible, we need only consider admissible valuation throughout. For P as above, the semantics on admissible valuations over simple $\mathcal{A}$ is thus faithfully described by the mapping

$$\begin{array}{rcl} \Gamma_P^{\mathcal{A}} \colon \Theta^l & \longrightarrow & \Theta^l \\ (\theta_1, \ldots, \theta_l) & \longmapsto & (\hat{\theta}_1, \ldots, \hat{\theta}_l), \end{array}$$

defined by the requirement that

$$\mathit{val}[P, \mathcal{A}, \eta](X_i) = \hat{\theta}_i[\mathcal{A}] \ \text{ for } i = 1, \ldots, l,$$

where $\eta(X_j) = \theta_j[\mathcal{A}]$ for $j = 1, \ldots, l$.

For the following inductive definition of $\Gamma_P^{\mathcal{A}}$ we fix $\mathcal{A}$ and omit the superscripts where convenient. We first treat assignment statements (or first-order formulas) in variables $x_1, \ldots, x_k$, then composition and for-loops.[3] For the first-order steps in (A)–(C) we may think of a program P consisting of just the assignment $X_1 := \{\bar{x} | \varphi(\bar{x})\}$, and we merely specify the value of $\hat{\theta}_1$, as trivially $\hat{\theta}_i = \theta_i$ for $i > 1$.

(A) Atomic formulas: if φ is a τ-atom or an equality-atom, put $\hat{\theta}_1 = \varphi$; if φ is an $\overline{X}$-atom, $\varphi = X_i(x_{i_1}, \ldots, x_{i_k})$, put $\hat{\theta}_1 = \theta_i(x_{i_1} \ldots x_{i_k})$.

(B) The Boolean connectives translate straightforwardly to Boolean set operations on $\hat{\theta}_1$.

(C) Existential quantification: w.l.o.g., consider the case of $\varphi = (\exists x_i) X_1$. Then $\hat{\theta}_1$ is the set of all those complete types whose restrictions to variables $x_1, \ldots, x_{i-1}, x_{i+1}, x_k$ are compatible (logically consistent) with θ_1.

[3] In Section 3.5, a very similar analysis will be applied to FO(FOR) formulas where, in fact, only the way in which for-loops are treated has to be modified; (A)–(C) will be unchanged.

(D) Composition: if $P = P_1; P_2$, and if Γ_i is as desired for P_i, then clearly $\Gamma_P = \Gamma_2 \circ \Gamma_1$ (functional composition) is good for P.

(E) For-loops: let $P = \mathbf{for}\, |X_i|\, \mathbf{do}\, P_0\, \mathbf{od}$. Then $\Gamma_P^{\mathcal{A}} = \left(\Gamma_{P_0}^{\mathcal{A}}\right)^\nu$ (ν-fold iteration) where $\nu = |\theta_i[\mathcal{A}]|$.

Note that there are only finitely many mappings $\Gamma\colon \Theta^l \to \Theta^l$, since Θ itself is finite. It follows that for any particular Γ, there is some constant q_Γ such that $\Gamma^{\nu+q_\Gamma} = \Gamma^\nu$ for all sufficiently large ν. Taking for q the product of all q_Γ, we find that

$$\Gamma^{\nu+q} = \Gamma^\nu, \text{ for all } \Gamma \text{ and sufficiently large } \nu, \tag{1}$$

i.e., from some value of ν onwards, the ν-fold iteration of any Γ merely depends on $\nu \bmod q$ rather than on ν itself.

We exploit this behaviour over families of simple structures $\mathcal{A}$ which admit a transparent description of the crucial values for the cardinalities of the sets $\theta[\mathcal{A}]$ (the values for ν in the semantics of for-loops). Here are two examples, which we will both use for inexpressiblity proofs in the sequel. Both families are indexed by a pair of parameters n, m and the crucial counting values turn out to be polynomials in n and m.

Example 3.7 Let $\tau = \{U\}$ for a unary predicate U. Let $\mathcal{A}^{(n,m)}$ have n elements in U and m outside. Any complete atomic k-type θ is uniquely characterized by the following data:

- a partition of $\{x_1, \ldots, x_k\}$ in terms of membership in U and its complement;
- for each part the equivalence relation which θ induces on it in terms of $=$; let k_1 and k_2 be their indices.

Then the following polynomial describes the cardinality of the set $\theta[\mathcal{A}^{(n,m)}]$, provided $n \geq k_1$ and $m \geq k_2$:

$$p_\theta(n, m) = n(n-1)\cdots(n-k_1+1)m(m-1)\cdots(m-k_2+1).$$

For later use we note the following. Consider $s \leq k$ and a fixed complete s-type ρ. Then the number of extensions that any given s-tuple that realizes ρ has to tuples that realize a given (partial) k-type θ, is also described by a polynomial with positive integer coefficients, which is either constant or strictly monotone in at least one of n or m. More precisely,

$$|\{\bar{b}|(\bar{a}, \bar{b}) \in \theta[\mathcal{A}^{(n,m)}]\}| = p_{\theta/\rho}(n, m),$$

for any fixed $\bar{a}$ with $\mathrm{atp}(\bar{a}) = \rho$, where

$$p_{\theta/\rho} = \frac{p_{\theta \wedge \rho}}{p_\rho}.$$

Note that ρ in the denominator is regarded as an s-type. We merely need to consider the case of complete θ which are consistent with ρ. But this means that ρ is actually the restriction of θ to variables $x_1, \ldots, x_s$. The respective indices k_1, k_2 for θ and k'_1, k'_2 for ρ clearly satisfy $k'_i \leq k_i$, whence the quotient $p_{\theta/\rho}$ reduces to a polynomial by cancellation of common terms.

■

Example 3.8 Let $\tau = \{E\}$, E binary. Let $\mathcal{A}^{(n,m)}$ be such that E is an equivalence relation with precisely n equivalence classes each of which has precisely m elements. Any complete atomic k-type θ, which is realizable in $\mathcal{A}^{(n,m)}$, is uniquely characterized by the following data

- the equivalence relation which it induces on $\{x_1, \ldots, x_k\}$ in terms of E; let $\alpha_1, \ldots, \alpha_i$ be its classes.
- for each α_j the equivalence relation which θ induces on α_j in terms of $=$; let k_j be its index.

Then the following polynomial describes the cardinality of the set $\theta[\mathcal{A}^{(n,m)}]$ for $n \geq i$ and $m \geq \max_j k_j$:

$$p_\theta(n, m) = n(n-1)\cdots(n-i+1)\prod_{j=1}^{i}(m(m-1)\cdots(m-k_j+1)).$$

Note that $p_\theta(n, m)$ is a multiple of the product nm and strictly monotone in both n and m, and this property remains true for all realizable partial θ.

■

We now use Example 3.8 to separate BQL from FO(FOR) by a Boolean query. While clearly $\mathcal{A}^{(n,m)} \models (\exists x)\xi_2(x)$ if and only if m is even[4] (cf. Example 3.2), we show that no BQL program draws this distinction.

[4]Note that E is an equivalence relation.

In Section 3.5, we shall use similar techniques and Example 3.7 to show that FO(FOR) cannot define the subclass of those $\mathcal{A}^{(n,m)}$ with $n = m$ (the equicardinality query).

Theorem 3.9 BQL *is strictly weaker than* FO(FOR).

Proof. For the proof we use the $\mathcal{A}^{(n,m)}$ of Example 3.8 and show that no BQL program accepts exactly those $\mathcal{A}^{(n,m)}$ for which m is even. Let P be any BQL program in variables $X_1, \dots, X_l$, normalized so that the arity of every X_i is k and all assignments only use variables $x_1, \dots, x_k$. We then show the following.

Claim 3.10 *There is a constant q and finitely many polynomials $p(n,m)$, all of them multiples of nm, such that for all $\mathcal{A}^{(n,m)}$ with sufficiently large n, m, the mapping $\Gamma_P^{\mathcal{A}^{(n,m)}}$ only depends on the values $p(n,m) \bmod q$.*

This proves the desired inexpressibility: choosing a sufficiently large multiple of q for n, and m just sufficiently large, it follows that P cannot distinguish between $\mathcal{A}^{(n,m)}$ and $\mathcal{A}^{(n,m+1)}$.

For the proof of the claim, we use as polynomials $p(n,m)$ all the p_θ describing the cardinalities of sets $\theta[\mathcal{A}^{(n,m)}]$ for sufficiently large n, m, for those θ that are realized in $\mathcal{A}^{(n,m)}$. Choose q such that all Γ on Θ^l satisfy $\Gamma^{\nu+q} = \Gamma^\nu$ for all sufficiently large ν, cf. (1) above. We prove the claim for these choices, by induction on P and the corresponding Γ_P, following (A)–(E) above. The claim goes trivially through for (A)–(D), and it remains to discuss (E) concerning for-loops. This is in fact the only place where a dependency on certain $p_\theta(n,m) \bmod q$ comes up. Suppose then that our program is of the form **for** $|X|$ **do** P **od**, and that Γ_P only depends on the $p(n,m) \bmod q$ for all sufficiently large n, m. We have to show that the same is true of $(\Gamma_P)^\nu$ where $\nu = |\theta[\mathcal{A}^{(n,m)}]| = p_\theta(n,m)$ for some $\theta \in \Theta$. We distinguish two cases, according to whether or not $\theta[\mathcal{A}]$ is empty (θ unsatisfiable). (Note that this is a legitimate case distinction as θ is one of the arguments $\theta_1, \dots, \theta_l$ for Γ, and as $\theta[\mathcal{A}]$ is empty for all $\mathcal{A}^{(n,m)}$ or for none, provided n, m are sufficiently large.) If $\theta[\mathcal{A}]$ is empty, the claim is trivial. Otherwise $|\theta[\mathcal{A}^{(n,m)}]| = p_\theta(n,m)$ is strictly monotone in both n and m. It follows that $\nu = |\theta[\mathcal{A}^{(n,m)}]| = p_\theta(n,m)$ is sufficiently large for sufficiently large n, m, and therefore $(\Gamma_P)^\nu$ will indeed only depend on Γ_P and $p_\theta(n,m) \bmod q$. Using the inductive hypothesis on Γ_P, this proves the claim. ∎

It is natural to ask whether there is a fragment of FO(FOR) which is equivalent to BQL. The FO(FOR) formula ξ_2 of Example 3.2 uses an individual parameter x in its head formula. Hence, to find a fragment of FO(FOR) equivalent to BQL, one could try to exclude individual parameters from head formulas. This, however, is not enough, since they can be simulated by relational parameters in the head and individual parameters in the body as is shown in the following proposition.

Proposition 3.11 *For every* FO(FOR) *formula there exists an equivalent* FO(FOR) *formula that does not use individual parameters in head formulas (but can still use relational parameters in head formulas).*

Proof. The proof proceeds by induction on the structure of FO(FOR) formulas. We only consider the interesting case. Consider the formula

$$\xi(\bar{u},\bar{y},\overline{Y}) = [\mathrm{FOR}^{\#\bar{z}:\psi(\bar{z},\bar{u},\bar{y},\overline{Y})}_{\bar{x},X}\varphi(\bar{x},\bar{y},X,\overline{Y})](\bar{u}).$$

Define $\xi'(\bar{u},\bar{y},\overline{Y})$ as follows,

$$\begin{array}{rcl}\xi'(\bar{u},\bar{y},\overline{Y}) & := & (\forall\bar{q})[\mathrm{FOR}^{\#v:v=v}_{\bar{q},Q} \\ & & \quad((Q=\emptyset\wedge\bar{q}=(\bar{u},\bar{y}))\vee \\ & & \quad(Q=\{(\bar{u},\bar{y})\}\wedge\bar{q}=\bar{q})\vee \\ & & \quad(Q=\mathit{all}\wedge\bar{q}=\bar{q})) \\ & & \quad\wedge\ [\mathrm{FOR}^{\#\bar{z}:\alpha(\bar{z},Q)}_{\bar{x},X} \\ & & \qquad((Q=\emptyset\vee Q=\mathit{all})\wedge\bar{x}=\bar{x})\vee \\ & & \qquad(Q\neq\emptyset\wedge Q\neq\mathit{all}\wedge\varphi(\bar{x},\bar{y},X,\overline{Y})\ ](\bar{u})\ \ ](\bar{q})\,.\end{array}$$

Here Q is a relation variable whose arity equals the width of $\bar{y}$. Further, $Q=\mathit{all}$, $Q=\{(\bar{u},\bar{y})\}$, and $Q=\emptyset$ are abbreviations for $(\forall\bar{q})Q(\bar{q})$, $(\forall\bar{q})(Q(\bar{q})\leftrightarrow\bar{q}=(\bar{u},\bar{y}))$, and $(\exists\bar{q})(Q(\bar{q}))$, respectively. Finally, $\alpha(\bar{z},Q)$ is the formula

$$(\exists\bar{u})(\exists\bar{y})\big(Q(\bar{u},\bar{y})\wedge Q\neq\mathit{all}\wedge\psi(\bar{z},\bar{u},\bar{y},\overline{Y})\big)\vee\big((Q=\emptyset\vee Q=\mathit{all})\wedge\bar{z}=\bar{z})\big).$$

Let $\mathcal{A}$ be a structure with at least two elements, and let $\overline{T}$, $\bar{a}$, and $\bar{b}$ be interpretations for $\overline{Y}$, $\bar{u}$, and $\bar{y}$, respectively. We have to show that

$$\mathcal{A}\models\xi[\bar{a},\bar{b},\overline{T}]\quad\Leftrightarrow\quad\mathcal{A}\models\xi'[\bar{a},\bar{b},\overline{T}]\,.$$

Suppose $\mathcal{A}\models\xi[\bar{a},\bar{b},\overline{T}]$.

1. In the first iteration of the outer for-loop, $Q = \emptyset$. The inner for-loop will do $|A|^n$ iterations, with n the width of $\bar{z}$. The variable X then gets the value A^m, with m the width of $\bar{x}$, which of course contains $\bar{a}$. Furthermore, Q is set to $\{(\bar{a}, \bar{b})\}$. Note that we introduced free variables $\bar{u}$ and $\bar{y}$ in the body of the inner for-loop.

2. In the second iteration, the inner for-loop will do $|\{\bar{c} \mid \mathcal{A} \models \psi[\bar{c}, \bar{a}, \bar{b}, \overline{T}]\}|$ iterations as does ξ. By assumption, $\bar{a}$ belongs to the final value of X. So after the second iteration of the outer for-loop, $Q = A^r$, with r the sum of the width of $\bar{u}$ and $\bar{y}$.

3. For each further iteration, $\bar{a}$ will belong to the final value of X, and Q will stay the full relation. Hence, $\mathcal{A} \models \xi'[\bar{a}, \bar{b}, \overline{T}]$.

Conversely, suppose $\mathcal{A} \not\models \xi[\bar{a}, \overline{T}]$. Then the value of Q will alternate between the empty relation and $\{(\bar{a}, \bar{b})\}$. Hence, Q will never be the full relation, and $\mathcal{A} \not\models \xi'[\bar{a}, \bar{b}, \overline{T}]$.

■

Individual parameters occurring in body formulas can be eliminated, possibly at the expense of introducing extra individual parameters in head formulas.

Proposition 3.12 *For every* FO(FOR) *formula there exists an equivalent* FO(FOR) *formula that does not use individual parameters in body formulas (but can still have individual parameters in head formulas).*

Proof. Consider an application of the $\mathrm{FOR}_{\bar{x},X}$-operator to a formula $\varphi(\bar{x}, \bar{z}, X)$ with individual parameters $\bar{z}$ (relational parameters suppressed). We wish to eliminate these individual parameters, and can do so at the expense of a corresponding increase in the arity of X. Let the arity of X' be the sum of the arities of X and $\bar{z}$. The intended interpretation of the iteration stages for X' is $X'_n = \{(\bar{x}, \bar{z}) \mid \bar{x} \in \varphi^n[\bar{z}]\}$. This is achieved by renaming all bound variables in φ so that they are different from those in $\bar{z}$, and then replacing any atom of the form $X(\bar{y})$ by $X'(\bar{y}, \bar{z})$. Hence, if $X(\bar{y})$ occurs in a head formula we introduce the new individual parameters $\bar{z}$. Call the resulting formula $\varphi'(\bar{x}, \bar{z}, X')$. The following equivalence is then immediate:

$$[\mathrm{FOR}^{\#\bar{y}:\psi}_{\bar{x},X} \varphi(\bar{x}, \bar{z}, X)](\bar{x}) \;\equiv\; [\mathrm{FOR}^{\#\bar{y}:\psi}_{\bar{x}\bar{z},X'} \varphi'(\bar{x}, \bar{z}, X')](\bar{x}, \bar{z}).$$

■

Disallowing individual parameters in both head and body formulas leads to a fragment of FO(FOR) equivalent to BQL.

Proposition 3.13 *The fragment of* FO(FOR) *that does neither allow individual parameters in heads nor in bodies of for-loops is equivalent to* BQL.

Proof. It follows from the proof of Proposition 3.4 that any BQL query can be simulated in FO(FOR) without individual parameters in the head or in the body.

We next show that for any FO(FOR) formula $\xi(\bar{u}, \overline{Y})$ without individual parameters in the head or in the body, there exists a BQL program P_ξ, with output variable $X_{\text{out},\xi}$, such that for any P_ξ-valuation η and structure $\mathcal{A}$,

$$val[P_\xi, \mathcal{A}, \eta](X_{\text{out},\xi}) = \{\bar{a} \mid \mathcal{A} \models \xi[\bar{a}, \eta]\}.$$

The proof proceeds by induction on the structure of ξ. We only consider the interesting case. Let ξ be of the form

$$\xi(\bar{u}, \overline{Y}) := [\mathrm{FOR}^{\#\bar{z}:\psi(\bar{z},\overline{Y})}_{\bar{x},X} \varphi(\bar{x}, X, \overline{Y})](\bar{u}).$$

Then define P_ξ as the following BQL program

P_ψ;
$X := \emptyset$;
for $|X_{\text{out},\psi}|$ **do**
    P_φ;
    $X := X_{\text{out},\varphi}$ **od**;
$X_{\text{out},\xi} := X$.

■

3.4 Inflationary versus non-inflationary iteration

If, in the definition of the semantics of the FOR operator, we replace , the stages φ^m by $\tilde{\varphi}^m$ (cf. Section 2), we obtain the inflationary version of FOR which we denote by IFOR.

It is routine to verify that FO(IFOR) collapses to FO on sets (i.e., over vocabularies consisting of unary relation names only). Indeed, each FO(IFOR)-definable relation is a union of automorphism classes and each automorphism class is definable by a quantifier-free formula. Since there is a uniform bound on the number of quantifier-free definitions (up to logical equivalence) and the bodies of for-loops iterate in an inflationary manner, each for-loop in FO(IFOR) can only iterate a fixed number of times (independent of the input structure). Hence, each for-loop is definable in first-order logic. Consequently, it is not expressible in FO(IFOR) that the cardinality of a set is even (since this is not expressible in FO [EF95]). This implies that FO(IFOR) is strictly weaker than FO(FOR).

On the other hand, FO(IFOR) is strictly more expressive than FO(IFP). Indeed, consider the vocabulary $\tau = \{U, R\}$ with U unary and R binary, and let $\mathcal{Q}_2$ be the following Boolean query: $\mathcal{Q}_2(\mathcal{A})$ is true if $R^{\mathcal{A}}$ is a chain, i.e., a successor-structure, and $|U^{\mathcal{A}}| \geq |R^{\mathcal{A}}|$ (here $|U^{\mathcal{A}}|$ denotes the cardinality of the set $U^{\mathcal{A}}$). The query $\mathcal{Q}_2$ is not definable in FO(IFP). Indeed, if φ is an FO(IFP) sentence, then, for some k, φ is equivalent to an $\mathcal{L}^k_{\infty\omega}$ sentence [EF95]. Let $\mathcal{A}$ be a structure where $|U^{\mathcal{A}}| = k$ and $R^{\mathcal{A}}$ is a chain of $k+1$ elements not occurring in $U^{\mathcal{A}}$, and let $\mathcal{B}$ be a structure where $|U^{\mathcal{B}}| = 2k$ and $R^{\mathcal{B}}$ is a chain of $k+1$ elements not occurring in $U^{\mathcal{A}}$. The Duplicator can win the k-pebble game on $\mathcal{A}$ and $\mathcal{B}$ by following the following strategy: when the Spoiler picks an element of a chain, the Duplicator picks the corresponding element on the other chain; and, when the Spoiler picks an unpebbled element in U, then the Duplicator responds with an arbitrary unpebbled element in the set of the other structure (we may assume that no element ever has two pebbles on it). This means that $\mathcal{A}$ and $\mathcal{B}$ are indistinguishable in $\mathcal{L}^k_{\infty\omega}$. Hence, φ cannot define $\mathcal{Q}_2$ because $\mathcal{A}$ satisfies $\mathcal{Q}_2$ and $\mathcal{B}$ does not. However, $\mathcal{Q}_2$ can be defined in FO(IFOR) by the formula

$$\texttt{chain} \wedge (\forall z) \Big(\texttt{last}(z) \rightarrow [\mathrm{IFOR}^{\#x:U(x)}_{x,X} \texttt{first}(x) \vee (\exists y)(X(y) \wedge R(y,x))](z)\Big),$$

where $\texttt{chain}$ is an FO(FOR) sentence saying that R is a chain, and $\texttt{first}(x)$ and $\texttt{last}(z)$ define the first and the last element of the chain, respectively. This yields the following proposition.

Proposition 3.14 FO(IFOR) *lies strictly between* FO(IFP) *and* FO(FOR).

3.5 A comparison with logics that count

Inflationary fixpoint logic with counting [GO93, Ott96, Ott97], here denoted by FO(IFP, #), is a two-sorted logic. With any structure $\mathcal{A}$ with universe A, we associate the two-sorted structure $\mathcal{A}^* := \mathcal{A} \cup \langle\{0, \ldots, n\}; \leq\rangle$ with

$n = |A|$ and where $\leq$ is the canonical ordering on $\{0, \ldots, n\}$. The two sorts are related by *counting terms*: if $\varphi(x, \bar{y})$ is a formula, then $\#_x[\varphi]$ is a term of the second sort. For any interpretation $\bar{b}$ for $\bar{y}$, the value of this term equals the number of elements a that satisfy $\varphi(a, \bar{b})$. The IFP operator can be applied to relations of mixed sort. Counting terms can be extended to apply to the counting of tuples and to yield tuples that correspond to the $|A|$-adic expansion of numbers, without increasing the expressive power of $\mathrm{FO(IFP, \#)}$. We will assume some familiarity with this logic and refer the reader to the sources just cited.

Every FO(FOR) formula can readily be simulated in $\mathrm{FO(IFP, \#)}$.

Proposition 3.15 $\mathrm{FO(IFP, \#)}$ *is at least as powerful as* FO(FOR).

Proof. We count the number of tuples in the relation defined by the head formula and then iterate the body formula that number of times. The only problem is that $\mathrm{FO(IFP, \#)}$ iterates in an inflationary manner while FO(FOR) in general does not. We resolve this by tagging each stage by a different number. If

$$\xi(\bar{u}, \bar{y}, \overline{Y}) = [\mathrm{FOR}^{\#\bar{z}:\psi(\bar{z}, \bar{u}, \bar{y}, \overline{Y})}_{\bar{x}, X} \varphi(\bar{x}, \bar{y}, X, \overline{Y})]\bar{u},$$

where $\bar{z} = z_1 \ldots z_\ell$, then ξ is equivalent to

$$\begin{aligned}(\exists \bar{\lambda})\big(\bar{\lambda} = \#_{\bar{z}}\psi^* \wedge \\ &[\mathrm{IFP}_{\bar{\mu}, \bar{x}, X'}(\bar{\mu} = \bar{0} \wedge \varphi^*(\bar{x}, \bar{y}, \emptyset, \overline{Y})) \\ &\quad \vee\ (\exists \bar{\nu})(\exists \bar{w})(X'(\bar{\nu}, \bar{w}) \wedge \bar{\nu} < \bar{\lambda} \wedge \bar{\mu} = \bar{\nu} + 1 \wedge \varphi_1^*](\bar{\lambda}, \bar{u})\big).\end{aligned}$$

The cardinality of the relation defined by ψ is bounded by $|A|^\ell$ on an input structure $\mathcal{A}$. An $(\ell + 1)$-ary tuple then represents a number between 0 and $|A|^{\ell+1} - 1$ in $|A|$-adic notation. The formulas ψ^* and φ^* are the $\mathrm{FO(IFP, \#)}$ formulas equivalent to respectively ψ and φ; X' is a mixed relation of arity $(\ell + 1, \mathit{arity}(X))$; and φ_1^* is obtained from φ^* by replacing each occurrence of $X(\bar{v})$ by $X'(\bar{\nu}, \bar{v})$. ∎

We next show that the subsumption of FO(FOR) by $\mathrm{FO(IFP, \#)}$ is strict. In fact, we show that the equicardinality query $\{(A, U) \mid |U| = |A \setminus U|\}$ is not definable in FO(FOR).

As we did for BQL in Section 3.3, we analyze the semantics of FO(FOR) formulas over simple structures in terms of mappings on partial types. It is useful to put formulas into a restricted normal form, w.r.t. arities of relation variables, the number of distinct first-order variables used, and w.r.t. to the role of first-order (individual) parameters in for-loops.

By Proposition 3.12, we can assume no body formula contains individual parameters. Normalization of all relation variables to some common arity, which is larger than the number of different first-order variables used, is straightforward just as for BQL. So we can assume from now on that every FO(FOR) formula uses no other first-order variables than $x_1, \dots, x_k$, that all occurring relation variables are of arity k, and that FOR-applications are of the form $\mathrm{FOR}_{\bar{x},X}$ where $\bar{x} = (x_1, \dots, x_k)$. By trivial renamings of variables we may also assume that the heads in FOR-applications always are of the form $\#\bar{y} : \psi(\bar{x})$ where $\bar{y} = (x_{s+1}, \dots, x_k)$ for some $0 \le s \le k$ (relational parameters may occur but are suppressed for clarity).

Let $\varphi(X_1, \dots, X_l, x_1, \dots, x_k)$ be an FO(FOR) formula. Just as for BQL it is easy to see that over simple structures, the semantics of such φ for admissible valuations is faithfully represented by a mapping

$$\begin{array}{rcl} \Gamma_\varphi^{\mathcal{A}} \colon \Theta^l & \longrightarrow & \Theta \\ (\theta_1, \dots, \theta_l) & \longmapsto & \hat{\theta} \end{array}$$

such that

$$\hat{\theta}[\mathcal{A}] = \{\bar{a} \in A^k | \mathcal{A} \models \varphi[\theta_1[\mathcal{A}], \dots, \theta_l[\mathcal{A}], \bar{a}]\}.$$

Again there are only finitely many such mappings, whence their iterations in for-loops will eventually have to be periodic. The type of iteration we encounter here — for a FOR-application $\mathrm{FOR}_{\bar{x},X_i}$ to a body formula φ whose semantics is represented by Γ say — is of the form $\left(\Gamma^{(i)}\right)^\nu$ for

$$\begin{array}{rcl} \Gamma^{(i)} \colon \Theta^l & \longrightarrow & \Theta^l \\ \bar{\theta} & \longmapsto & (\theta_1, \dots, \theta_{i-1}, \Gamma(\bar{\theta}), \theta_{i+1}, \dots, \theta_l). \end{array}$$

Just as in the treatment of BQL, therefore, we find some modulus q such that for all sufficiently large ν and all Γ and i

$$\left(\Gamma^{(i)}\right)^{\nu+q} = \left(\Gamma^{(i)}\right)^{\nu}. \tag{2}$$

The inductive generation of the Γ_φ follows the steps (A)–(C) outlined for BQL. We concentrate on the FOR-step and consider the formula

$$\varphi(\overline{X}, \bar{x}) = [\mathrm{FOR}_{\bar{x},X_1}^{\#\bar{y}:\varphi_1(\bar{x})} \varphi_0(\overline{X}, \bar{x})](\bar{x}),$$

where $\bar{y} = (x_{s+1}, \dots, x_k)$ for some $0 \le s \le k$, and $\bar{x} = (x_1, \dots, x_k)$. Assume that Γ_0 and Γ_1 represent the semantics of φ_0 and φ_1 over some simple $\mathcal{A}$. Recall how the number of φ_0-iterations to be performed to determine whether $\mathcal{A} \models \varphi[\bar{a}]$ depends on $(a_1, \dots, a_s)$, the parameters in the counting expression $\#_{\bar{y}} : \varphi_1$. Over simple $\mathcal{A}$ this dependence reduces to a dependence on $\mathrm{atp}_{\mathcal{A}}(a_1, \dots, a_s)$. Clearly $\varphi(\bar{x}) \equiv \bigvee_\rho (\rho(\bar{x}) \wedge \varphi(\bar{x}))$ where ρ ranges over all complete s-types (regarded as partial k-types). Since Boolean operations are trivial, we may treat just one particular disjunct, or actually assume w.l.o.g. that we are dealing with $\varphi := \varphi \wedge \rho$ for some fixed ρ. Then the following mapping is adequate for φ:

$$\Gamma_\varphi^{\mathcal{A}}(\bar{\theta}) = \big((\Gamma_0^{\mathcal{A}})^{(1)}\big)^{\nu}(\bar{\theta}'),$$

where $\bar{\theta}' = (\emptyset, \theta_2, \dots, \theta_l)$ (initialization $X_1 := \emptyset$), and

$$\begin{array}{l} \nu = \nu(\bar{\theta}) = |\{\bar{b} \in A^{(k-s)} \mid (\bar{a}, \bar{b}) \in \Gamma_1^{\mathcal{A}}(\bar{\theta})[\mathcal{A}]\}| \\ \text{for any fixed } \bar{a} = (a_1, \dots, a_s) \text{ with } \mathrm{atp}(\bar{a}) = \rho. \end{array} \tag{3}$$

We use the family of structures $\mathcal{A}^{(n,m)}$ of Example 3.7, consisting of a set A of size $n + m$ and a subset $U \subset A$ of size n, to show the following separation.

Theorem 3.16 FO(FOR) *is strictly weaker than* FO(IFP, #)*, in fact the equicadinality query* $\{(A, U) \mid |U| = |A \setminus U|\}$ *is not definable in* FO(FOR).

Proof. We show that no sentence φ of FO(FOR) is satisfied by exactly those $\mathcal{A}^{(n,m)}$ of Example 3.7 where $n = m$.

Claim 3.17 *Let φ be an FO(FOR) sentence. There are a constant q and finitely many polynomials $p(n, m)$ with positive integer coefficients, such that for all $\mathcal{A}^{(n,m)}$ with sufficiently large n, m, the mapping $\Gamma_\varphi^{\mathcal{A}^{(n,m)}}$ only depends on the values $p(n, m) \bmod q$.*

This implies that φ cannot distinguish between $\mathcal{A}^{(n,n)}$ and $\mathcal{A}^{(n,n+q)}$ for sufficiently large n. It remains to argue for the claim. We use as polynomials the $p_{\theta/\rho}(n, m)$ already considered in Example 3.7, and for q the modulus of (2) above. Note that the $p_{\theta/\rho}$ exactly correspond to the counting values ν needed in (3). With these choices, the inductive proof of the claim is obvious for first-order steps. FOR-steps are treated according to the above preparation, always relative to some fixed ρ. We inductively assume that the Γ_i are determined by the values $p(n,m) \bmod q$. Then so are $(\Gamma_0)^{(1)}$ and $\theta = \Gamma_1(\bar{\theta})$, and hence also $\nu \bmod q = p_{\theta/\rho}(n,m) \bmod q$. Note that $\nu = p_{\theta/\rho}(n,m)$ is either constant or sufficiently large for all sufficiently large n, m. In either case we find that indeed $\left((\Gamma_0)^{(i)}\right)^{\nu}$ is determined by the $p(n,m) \bmod q$ as claimed. ■

Although FO(FOR) is strictly weaker than fixpoint logic with counting, it is strictly more expressive than fixpoint logic with modular counting only. The latter logic is defined as the extension of FO(IFP) with generalized quantifiers $D_n x \varphi(x, \bar{y})$, for each natural number $n \geq 2$, meaning that $\mathcal{A} \models D_n x \varphi(x, \bar{a})$ if and only if $|\{b \mid \mathcal{A} \models \varphi(b, \bar{a})\}| \equiv 0 \pmod{n}$.

We show that the query $\mathcal{Q}_2$ from Section 3.4 is not expressible in FO(IFP) plus modular counting:

Theorem 3.18 FO(FOR) *is strictly stronger than* FO(IFP) *plus modular counting.*

Proof. We first outline how modular counting can be simulated in FO(FOR). The formula $D_n x \varphi(x, \bar{y})$ is simulated by

$$(\forall x)(\neg\varphi(x, \bar{y})) \vee (\exists x_0)[\text{S-FOR}^{\#x:\varphi(x,\bar{y})}_{x_0, X_0, \ldots, x_{n-1}, X_{n-1}} \psi_0, \ldots, \psi_{n-1}](x_0),$$

where $\psi_0 := X_{n-1}(x_0) \wedge \neg X_0(x_0)$;

$$\begin{aligned} \psi_1 := ((X_0 = \emptyset \wedge X_1 = \emptyset) \rightarrow x_1 = x_1) \\ \wedge (X_0 \neq \emptyset \rightarrow X_0(x_1)) \wedge (X_1 \neq \emptyset \rightarrow x_1 \neq x_1), \end{aligned}$$

and for $i = 2, \ldots, n-1$, $\psi_i := X_{i-1}(x_i) \wedge \neg X_i(x_i)$. After i iterations of the body of the for-loop, if $i > 0$ then $X_j \neq \emptyset$ if and only if $i \equiv j \pmod{n}$.

For the separation, we use the following modicification of the pebble game defined in Section 2, which we call the (k, D_n) pebble game. In a round of the (k, D_n) pebble game on two structures $\mathcal{A}$ and $\mathcal{B}$, the Spoiler chooses a structure (say $\mathcal{A}$), one of the k pebbles (say i), a natural number m smaller than or equal to n, and a set $X \subseteq A$. The Duplicator then answers by choosing a set $Y \subseteq B$ such that $|X| \equiv |Y| \pmod{m}$. The Spoiler now puts pebble i on an element $b \in B$, whereafter the Duplicator puts pebble i on an element $a \in A$ such that $a \in X$ if and only if $b \in Y$. The winning conditions are defined just as for the ordinary pebble game.

It follows from the work of Kolaitis and Väänänen [KV95] that if the Duplicator wins the (k, D_n) pebble game on $\mathcal{A}$ and $\mathcal{B}$ with the pebbles initially placed on $\bar{a}$ and $\bar{b}$, then for any FO(IFP) formula φ with at most k different variables and that uses only quantifiers D_m with $m \leq n$:

$$\mathcal{A} \models \varphi(\bar{a}) \quad \Leftrightarrow \quad \mathcal{B} \models \varphi(\bar{b}).$$

Suppose towards a contradiction that $\mathcal{Q}_2$ is defined by the formula φ that uses at most k different variables and where n is a natural number such that for every quantifier D_m that occurs in φ, $m \leq n$ and $n! > k$. Let $\mathcal{A}$ be a structure with $|U^{\mathcal{A}}| = 2k \cdot n!$ and where $R^{\mathcal{A}}$ is a chain of $3k \cdot n!$ elements not occurring in $U^{\mathcal{A}}$, and let $\mathcal{B}$ be a structure with $|U^{\mathcal{B}}| = 3k \cdot n!$ and where $R^{\mathcal{B}}$ is a chain of $3k \cdot n!$ elements not occurring in $U^{\mathcal{A}}$. We now describe the winning strategy of the Duplicator in the (k, D_n) pebble game on $\mathcal{A}$ and $\mathcal{B}$. On the chain R, the Duplicator picks exactly the same elements in A (respectively B) as the Spoiler does in B (respectively A). Therefore, we restrict attention to moves in U. Since U is a simple set disjoint from R, the response of the Duplicator is obvious once the sets X and Y are chosen. Hence, we only discuss the choice of the latter.

Assume the elements $\bar{a}$ and $\bar{b}$ are pebbled and $\bar{a} \to \bar{b}$ is a partial isomorphism of $\mathcal{A}$ and $\mathcal{B}$. We denote the value of the ith pebble in A and B by $\pi_{\mathcal{A}}(i)$ and $\pi_{\mathcal{B}}(i)$, respectively.

1. The Spoiler chooses a subset X of $U^{\mathcal{A}}$ and a natural number $2 \leq m \leq n$.

   (a) If $|X| = |U^{\mathcal{A}}| - j$, for some $j \in \{0, \ldots, k\}$, then the Duplicator takes a subset Y in $U^{\mathcal{B}}$ of size $|U^{\mathcal{B}}| - j$ such that for every pebble i, $\pi_{\mathcal{A}}(i) \in X$ if and only if $\pi_{\mathcal{B}}(i) \in Y$. We have, $|X| \equiv |Y| \pmod{m}$ since $|U^{\mathcal{A}}| \equiv |U^{\mathcal{B}}| \pmod{m}$.

   (b) If $|X| < |U^{\mathcal{A}}| - k$ then the Duplicator takes a subset Y in $U^{\mathcal{B}}$ of size $|X|$ such that for every pebble i, $\pi_{\mathcal{A}}(i) \in X$ if and only if $\pi_{\mathcal{B}}(i) \in Y$. We trivially have $|X| = |Y| \pmod{m}$.

2. The Spoiler chooses a subset Y of $U^{\mathcal{B}}$ and a natural number $m \leq n$.

   (a) If $|Y| = |U^{\mathcal{B}}| - j$, for some $j \in \{0, \ldots, k\}$, then the Duplicator takes a subset X in $U^{\mathcal{A}}$ of size $|U^{\mathcal{A}}| - j$ such that for every pebble i, $\pi_{\mathcal{A}}(i) \in X$ if and only if $\pi_{\mathcal{B}}(i) \in Y$. We have, $|X| \equiv |Y| \pmod{m}$ as $|U^{\mathcal{A}}| \equiv |U^{\mathcal{B}}| \pmod{m}$.

(b) If $|Y| < |U^{\mathcal{A}}| - k$ then the Duplicator takes a subset X in $U^{\mathcal{A}}$ of size $|Y|$ such that for every pebble i, $\pi_{\mathcal{A}}(i) \in X$ if and only if $\pi_{\mathcal{B}}(i) \in Y$.

(c) If $|U^{\mathcal{A}}| - k \leq |Y| < |U^{\mathcal{B}}| - k$ then the Duplicator takes a subset X in $U^{\mathcal{A}}$ of size $k \cdot n! + (|Y| \bmod n!)$ such that for every pebble i, $\pi_{\mathcal{A}}(i) \in X$ if and only if $\pi_{\mathcal{B}}(i) \in Y$.

■

We end this section with the following observation. FO(IFP, #) formulas are not evaluated on the τ-structures themselves but on the expansion of these τ-structures with an initial fragment of the natural numbers. Hence, it is interesting to ask how FO(FOR) compares to FO(IFP) when we make these natural numbers also available to FO(FOR) formulas (as is the case for FO(IFP, #) but without the counting terms). It turns out that under these conditions, FO(FOR) becomes equally powerful as FO(IFP, #). For example, we can easily simulate $\mu = \#_x[\varphi]$ as $[\mathrm{FOR}^{\#x:\varphi}_{\mu,M}\, \mu := \mu + 1](\mu)$, where $\mu := \mu + 1$ is an abbreviation of the FO formula that defines μ as 1 in the first iteration and then subsequently increases μ in M by 1.

4 Nesting of for-loops

In this section we study the nesting of for-loops in both BQL and FO(FOR).

4.1 Nesting in BQL

The *nesting depth* of a BQL program P, denoted by *depth*(P), is inductively defined as follows:

(i) the depth of an assignment statement is 0;

(ii) $\mathit{depth}(P_1; P_2) := \max\{\mathit{depth}(P_1), \mathit{depth}(P_2)\}$; and

(iii) $\mathit{depth}(\textbf{for } |X| \textbf{ do } P \textbf{ od}) := \mathit{depth}(P) + 1$.

For $i \geq 0$, let BQL_i be the fragment of BQL consisting of BQL programs of nesting depth at most i. We refer to BQL_1 as *unnested* BQL. Note that BQL_0 programs do not have any for-loops at all.

Theorem 4.1 *Unnested* BQL *is strictly weaker than* BQL.

Proof. Take the vocabulary $\tau = \{E, C\}$, with E binary and C unary. We consider graphs (with edge relation E) of the form of a chain, where to each node of the chain is attached a separate non-empty set of nodes. The chain is distinguished in the structure by the unary relation C. The attached sets are of various sizes bounded above by the length of the chain. If n is the length of the chain and α_i is the size of the ith set, then this structure is denoted by $\alpha = (\alpha_1, \ldots, \alpha_n)$; this is an element of $\{1, \ldots, n\}^n$. We denote the graph associated to α by G_α. In Figure 2, an example of such a structure is depicted.

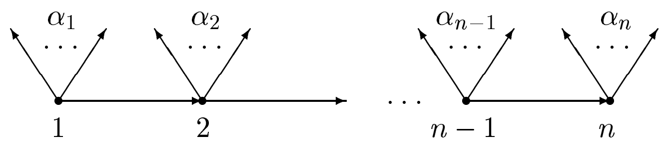

Figure 2: $\alpha = (\alpha_1, \ldots, \alpha_n)$

Now, let $\mathcal{Q}_3$ be the binary query defined by

$$\mathcal{Q}_3(G_\alpha) := \{(i, \alpha_i) \mid i \in \{1, \ldots, n\}\}.$$

By the numbers i and α_i we mean respectively the i-th and α_i-th element of the chain. Observe that this query is injective. I.e., if α and β are two different n-tuples then $\mathcal{Q}_3(G_\alpha) \neq \mathcal{Q}_3(G_\beta)$. We can express $\mathcal{Q}_3$ by the BQL program in Figure 3.

Suppose that P is an unnested BQL program that computes $\mathcal{Q}_3$. Let k be the maximum number of first-order variables used in P, let d be the number of (unnested!) for-loops in P, and let ℓ be the maximum arity of the relation variables that appear as heads of for-loops in P.

For any $k \geq 2$ and any n large enough there always exist two nonisomorphic graphs G_α and G_β with $\alpha, \beta \in \{k, \ldots, n\}^n$ in which the cardinalities of the relations occurring in the heads of all the for-loops in P are equal. Indeed, the number of possible sequences of cardinalities of heads in P is bounded by the polynomial $(n(n+1)+1)^{d\ell}$, because $n(n+1)$ is the maximal number of elements in such graphs, and there are d heads of arity at most ℓ, while there are exponentially many elements in $\{k, \ldots, n\}^n$.

These G_α and G_β are also indistinguishable in FO^k, because $\alpha_i, \beta_i \geq k$. Indeed, if the Spoiler pebbles the i-th element of the chain, then the Duplicator pebbles the i-th element of the other structure; if the Spoiler

$X := \emptyset;$
$Y := \{x \mid \texttt{first}(x)\};$
$V := \{x \mid C(x)\};$
for $|V|$ **do**
    $Z := \{x \mid (\exists y)(Y(y) \wedge E(y,x) \wedge \neg C(x))\}$
    $W := \emptyset;$
    for $|Z|$ **do**
        $W := \{x \mid (W = \emptyset \wedge \texttt{first}(x)) \vee (\exists y)(W(y) \wedge E(y,x) \wedge C(x))\};$
    od;
    $X := X \cup (Y \times W);$
    $Y := \{x \mid (\exists y)(Y(y) \wedge E(y,x) \wedge C(x))\};$
od;
$X_{\text{out}} := X;$

Figure 3: A BQL_2 program that computes $\mathcal{Q}_3$, where $\texttt{first}(x)$ is an abbreviation for $C(x) \wedge \neg(\exists y)(E(y,x))$.

pebbles an element in the i-th set, then the Duplicator pebbles an element in the i-th set of the other structure. Moreover, the result of P on input G_α, denoted by $P(\alpha)$, is indistinguishable in FO^k from the result of P on input G_β, denoted by $P(\beta)$, since in both structures P is evaluated as *the same* sequence of FO^k-definable substitutions. Indeed, recall that we chose G_α and G_β such that every for-loop iterates the same number of times on both structures.

Now, $P(\alpha)$ and $P(\beta)$ are indistinguishable subsets of chains of equal length, so they must in fact be *equal,* because every element of a chain is distinguishable from all others in FO^2 [Ott97]. Hence, the query expressed by P is not injective. This leads to the desired contradiction, because we noted that $\mathcal{Q}_3$ is injective. ■

Note that the query $\mathcal{Q}_3$ used in the above proof is not a Boolean query. It remains open whether there exists a Boolean query that separates BQL from unnested BQL.

We next show that the nesting hierarchy is not strict.

Theorem 4.2 BQL *is equivalent to* BQL_2.

Proof. We show that every BQL program is equivalent to a program in a normal form that contains both for- and while-loops. This program can then easily be transformed into an equivalent BQL_2 program. More precisely, we show that every BQL program is equivalent to a program of the form

$$P_1;\ \textbf{while}\ Y \neq \emptyset\ \textbf{do}\ P_2\ \textbf{od}, \tag{$\star$}$$

where P_1 does not contain a for-loop nor a while-loop; P_2 is an unnested BQL program; and the variable Y becomes empty on any structure after at most a polynomial number of iterations.

If on any input structure $\mathcal{A}$ the variable Y becomes empty after at most $|A|^\ell$ iterations then we say that Y is *bounded by* ℓ. We say that Y is *polynomially bounded* if it is bounded for some ℓ. A program that contains both while-loops and for-loops is a *mixed program*; such a program is in *mixed normal form* if it is of the form ($\star$).

Let us show that any program in mixed normal is in fact equivalent to a BQL_2 program. We first need some terminology. Let ψ be a first-order sentence over the vocabulary τ expanded with the relation variables. We define a mapping ρ_ψ from BQL programs to BQL programs inductively as follows:[5]

1. $\rho_\psi(X := \{\bar{x} \mid \varphi(\bar{x})\})\ :=\ (X := \textbf{if}\ \psi\ \textbf{then}\ \{\bar{x} \mid \varphi(\bar{x})\}\ \textbf{else}\ X)$;
2. $\rho_\psi(P_1; P_2)\ :=\ (\rho_\psi(P_1); \rho_\psi(P_2))$;
3. $\rho_\psi(\textbf{for}\ |X|\ \textbf{do}\ P\ \textbf{od})\ :=\ (\textbf{for}\ |X|\ \textbf{do}\ \rho_\psi(P)\ \textbf{od})$.

The mapping ρ_ψ is defined for RQL programs in a similar manner. (Recall that RQL is the extension of the relational calculus with while-loops defined at the end of Section 3.1.)

By the following lemma, it suffices to show that every BQL program can be brought into mixed normal form.

Lemma 4.3 *Each mixed program P in mixed normal form is equivalent to a* BQL_2 *program.*

Proof. If P is of the form

$$P_1;\ \textbf{while}\ Y \neq \emptyset\ \textbf{do}\ P_2;\ \textbf{od},$$

where Y is bounded by ℓ, then P is equivalent to

$$P_1;\ X := A^\ell;\ \textbf{for}\ |X|\ \textbf{do}\ \rho_{Y\neq\emptyset}(P_2)\ \textbf{od},$$

[5]Here, **if** ψ **then** $\{\bar{x} \mid \varphi(\bar{x})\}$ **else** X is a shorthand for $\{\bar{x} \mid (\psi \wedge \varphi(\bar{x})) \vee (\neg\psi \wedge X(\bar{x}))\}$.

where X does not occur in P_1 and P_2, and $X := A^\ell$ is an abbreviation for $X := \{(x_1, \ldots, x_\ell) \mid \bar{x} = \bar{x}\}$. ■

Programs in mixed normal form can easily be manipulated as shown in the next lemma.

Lemma 4.4 *If Q_1 and Q_2 are two programs in mixed normal form, then $Q_1; Q_2$ and* **while** $X \neq \emptyset$ **do** Q_1 **od** *are also equivalent to programs in mixed normal form, provided X is polynomially bounded. If Q is an unnested* BQL *program then $Q_1; Q$ and $Q; Q_1$ are equivalent to programs in mixed normal form.*

Proof. If Q_1 is in mixed normal form

$$P_0; \textbf{while } X_1 \neq \emptyset \textbf{ do } P_1 \textbf{ od},$$

and Q_2 is in mixed normal form

$$P_2; \textbf{while } X_3 \neq \emptyset \textbf{ do } P_3 \textbf{ od},$$

X_1 is bounded by ℓ and X_3 is bounded by ℓ', then $Q_1; Q_2$ is equivalent to the program in Figure 4. In this program phase1, phase2, phase3 and stop are nullary program variables not occurring in Q_1 or Q_2. They are used as Booleans in the standard way. The variable stop is bounded by $\ell + \ell'$.

The program **while** $X \neq \emptyset$ **do** Q_1 **od** is equivalent to the program in Figure 5, where stop, outerloop, and innerloop are nullary variables representing Booleans not occurring in Q_1. The variable stop is bounded by $\ell + \ell'$.

The cases $Q_1; Q$ and $Q; Q_1$ reduce to the first case by noticing that Q is equivalent to the following program in mixed normal form:

$$\text{ok} := \textbf{true}; \textbf{while } \text{ok} \textbf{ do } \text{ok} := \textbf{false}; Q \textbf{ od}.$$

■

We now show that every BQL program is equivalent to a program in mixed normal form. The proof proceeds by induction on the structure of BQL programs. The theorem then follows from Lemma 4.3.

The cases where P is of the form $X := \{\bar{x} \mid \varphi(\bar{x}, \overline{X})\}$ or where P is of the form $P_1; P_2$ with P_1 and P_2 in mixed normal form, follow from Lemma 4.4. Therefore, let P be of the form **for** $|X|$ **do** P' **od**, where P' is in mixed normal

P_0;
phase1 := $X_1 \neq \emptyset$;
phase2 := ¬phase1;
phase3 := **false**;
stop := **false**;
while ¬stop **do**
    $\rho_{\text{phase1}}(P_1)$;
    $\rho_{\text{phase2}}(P_2)$;
    $\rho_{\text{phase3}}(P_3)$;
    phase1 := phase1 **and** $X_1 \neq \emptyset$;
    phase2 := ¬phase1 **and** ¬phase2 **and** ¬phase3;
    phase3 := ¬phase1 **and** ¬phase2 **and** $X_3 \neq \emptyset$;
    stop := ¬phase1 **and** ¬phase2 **and** ¬phase3;
od;

Figure 4: The program in mixed normal form equivalent to $Q_1; Q_2$.

form. We choose k to be the maximum of all arities of variables in P and of the number of distinct first-order variables used in assignment statements in P (free or bound). Let v be the number of distinct variables used in P. As explained in Section 3.3, we can assume that all relation variables are k-ary.

Now, to simulate P, we first compute the k-variable Abiteboul-Vianu invariant $\pi_k(\mathcal{A})$ [AV95] of the input structure $\mathcal{A}$ by an RQL program $P^{<_k}$ in mixed normal form (note that this program does not use for-loops). The elements of $\pi_k(\mathcal{A})$ are the FO^k-equivalence classes of $\mathcal{A}$. Moreover, $\pi_k(\mathcal{A})$ provides us with a total order on the FO^k-equivalence classes. We will exploit this ordering to simulate some for-loops by while-loops whereafter we merge some of these while-loops together. Some care has to be taken, however, since this ordering is in general *not* a total ordering on $\mathcal{A}$. Since on any given structure every BQL program is equivalent to an FO^k formula (recall Section 3.3), every relation definable by a BQL program is a union of FO^k-equivalence classes on the input structure. In the following we refer to these classes by the natural numbers $\mathbf{1}, \ldots, \mathbf{N}$, where $\mathbf{N} = |\pi_k(\mathcal{A})|$ and $\mathcal{A}$ is the input structure under consideration. So, the output of a BQL program can be seen as a relation over $\{\mathbf{1}, \ldots, \mathbf{N}\}$.

In a relation variable D, we can now encode any number between 0 and $2^{\mathbf{N}} - 1$. Indeed, D represents the number zero if $D = \emptyset$ and the num-

stop := $X = \emptyset$;
outerloop := **true**;
while $\neg$stop **do**
    $\rho_{\text{outerloop}}(P_0)$;
    innerloop := $X_1 \neq 0$;
    $\rho_{\text{innerloop}}(P_1)$;
    outerloop := $X_1 = \emptyset$;
    stop := $X_1 = \emptyset \wedge X = \emptyset$;
od;

Figure 5: The program in mixed normal form equivalent to the program **while** $X \neq \emptyset$ **do** Q_1 **od**.

ber $\sum_{\mathbf{i} \in D} 2^{\mathbf{i}-1}$ otherwise. If D_1 and D_2 are two sets then the operations $\min\{D_1, D_2\}$, 'if $D_1 < D_2$ then $D_1 + 1$ else $\mathbf{0}$', and $D_1 - 1$ are expressible by single assignment statements. A tuple of ℓ sets $(D_1, \ldots, D_\ell)$ can now encode any number between 0 and $2^{\ell \cdot \mathbf{N}} - 1$. The operations described above can also be expressed for such tuples.

We now start with the simulation of P. Consider the input $\mathcal{A}$ with initial valuation η. We make a distinction between two cases: $|\eta(X)| < 2^{v \cdot \mathbf{N}}$ and $|\eta(X)| \geq 2^{v \cdot \mathbf{N}}$. In case $|\eta(X)| < 2^{v \cdot \mathbf{N}}$, using relations as counters, we can simulate the for-loop of P by a while-loop of the desired form. If, on the other hand, $|\eta(X)| \geq 2^{v \cdot \mathbf{N}}$, then we know that the execution of the loop will repeat a configuration because there are only $2^{v \cdot \mathbf{N}}$ assignments of values to v relation variables, and we can "shortcut" the computation of the for-loop. We now explain this in more detail.

In outline, the program equivalent to P that we are going to describe is of the form

$P^{<_k}$;
if $|\eta(X)| < 2^{v \cdot \mathbf{N}}$ **then** $\delta_1(P)$
    else $\delta_2(P)$

where $\delta_1(P)$ is the program in mixed normal form equivalent to P on structures $\mathcal{A}$ with initial valuation η where $|\eta(X)| < 2^{v \cdot \mathbf{N}}$, and $\delta_2(P)$ is the program in mixed normal form equivalent to P on structures $\mathcal{A}$ with initial valuation η where $|\eta(X)| \geq 2^{v \cdot \mathbf{N}}$. Using the ordering $<$, the cardinality test can be performed by the following program $P_{\text{card\_test}}$:

$X_{\text{card\_test}} := 0;$
for $|X|$ **do** $X_{\text{card\_test}} := \min\{X_{\text{card\_test}} + 1, 2^{v \cdot \mathbf{N}}\}$ **od**;
condition $:= X_{\text{card\_test}} \neq 2^{v \cdot \mathbf{N}}$.

Here, $X_{\text{card\_test}}$ stands for a v-tuple of relation variables, as explained above. The if-test can by replaced by while-loops in the standard way; see Figure 6. By applying Lemma 4.4 several times, this program can be transformed into

```
P^{<k};
P_card_test;
ok := true;
stop1 := false;
stop2 := false;
while ok do
        ok := false;
        while condition and ¬stop1 do
                stop1 := false;
                δ1(P)
        od;
        while ¬condition and ¬stop2 do
                stop2 := false;
                δ2(P)
        od
od;
```

Figure 6: Outline of the mixed program equivalent to P.

mixed normal form. It now only remains to define $\delta(P_1)$ and $\delta(P_2)$.

(i) The program $\delta_1(P)$ is defined as follows:

```
counter := X_card_test;
while counter ≠ 0 do
        counter := counter − 1;
        P′;
od.
```

Note that this while-loop is bounded by $arity(X)$. By applying Lemma 4.4 this program can be brought into mixed normal form.

(ii) The definition of $\delta_2(P)$ is based on the observation mentioned above that, if $|\eta(X)| \geq 2^{v\dot{\mathbf{N}}}$, the execution of the loop will repeat a configuration. Formally, let c be minimal such that for all $n \geq 2^{v\cdot\mathbf{N}}$, $val[P^{(n)}, \mathcal{A}, \eta] = val[P^{(n+c)}, \mathcal{A}, \eta]$. We say that c is the *cycle size* of P on $\mathcal{A}$ and η. Note that the cycle size is less than $2^{v\cdot\mathbf{N}}$. The program $\delta_2(P)$ is depicted in Figure 7. It simulates P by iterating its body first $2^{v\cdot\mathbf{N}}$ times and then $(|\eta(X)| - 2^{v\cdot\mathbf{N}}) \bmod c$ times. The former can be done by a single while loop as in case (i). In the figure, P_c is a program in mixed normal form that computes the cycle size, described below. We assume that the variables X', counter and rest do not appear in P'. Note that this is the only place, apart from the program $P_{\text{card\_test}}$, where a real for-loop appears. The while-loops in the program are bounded by $arity(X)$. Hence, the program can be put in mixed normal form by applying Lemma 4.4 several times.

P_c;
counter := $2^{v\cdot\mathbf{N}}$;
while counter $\neq 0$ **do**
    counter := counter $-$ 1;
    P';
od;
counter := 0;
for $|X|$ **do**
    counter := if counter $<$ cycle_size then counter + 1 else 0
od;
while counter $\neq 0$ **do**
    counter := counter $-$ 1;
    P';
od;

Figure 7: The program $\delta_2(P)$.

We complete the proof by describing the program P_c computing the cycle size. This program is shown in Figure 8. Here, $P_{\text{store\_initial\_values}}$ is the program that copies the initial values of the variables into some help variables, and $P_{\text{store\_current\_values\_}S}$ is the program that stores the current values in some help variables to which we refer as S. Finally, $P_{\text{restore\_initial\_values}}$ is the program that

```
P_store_initial_values;
counter := 2^{v·N};
while counter ≠ 0 do
        counter := counter − 1;
        P';
od;
P_store_current_values_S;
cycle_size := 0;
same_config := false;
while ¬same_config do
        P';
        same_config := current_config = S;
        cycle_size := cycle_size + 1;
od;
P_restore_initial_values;
```

Figure 8: The program P_c.

restores the initial values. Note that these programs are just sequences of assignments and consequently contain no while- or for-loop. Now, we know that after $2^{v \cdot \mathbf{N}}$ iterations the body of P repeats a configuration. Hence, we iterate P' that many times and then store the value of all its variables in S. Starting from these values we iterate P' until we obtain the same configuration (this happens after at most $2^{v \cdot \mathbf{N}}$ iterations) and we count the steps. This gives us the cycle size. The statement same_config := current_config $= S$ is an abbreviation for the assignment that assigns true to the variable same_config if the current values of the variables of P equal those stored in S, and assigns false to same_config otherwise. Again the while-loops are bounded by *arity*(X). ■

4.2 Nesting in FO(FOR)

There are two ways of nesting for-loops in FO(FOR): nesting in the head formulas and nesting in the body formulas. We show that nesting in the head does not give additional power, while nesting in the body does. It remains open whether nesting in the body up to a certain level is sufficient.

4.2.1 Nesting in the head

Let FH-FO(FOR) be the fragment of FO(FOR) that does not allow for-loops in its head formulas. That is, only first-order heads are allowed. We show that nesting in the head is dispensable.

Proposition 4.5 FO(FOR) *is equivalent to* FH-FO(FOR).

Proof. The proof proceeds by induction on the structure of FO(FOR) formulas. We only treat the interesting case. Let $\xi(\bar{u}, \bar{y}, \overline{Y})$ be the formula

$$[\mathrm{FOR}_{\bar{x},X}^{\#\bar{z}:\psi(\bar{z},\bar{u},\bar{y},\overline{Y})}\varphi(\bar{x}, \bar{y}, X, \overline{Y})](\bar{u}).$$

We can assume that both ψ and φ are FH-FO(FOR) formulas. On structures with at least two elements ξ is equivalent to the formula

$$[\text{S-FOR}_{\bar{u},U,\bar{z},Z}^{\#z:z=z}\sigma_1, \sigma_2](\bar{u}),$$

where σ_1 is the formula $[\mathrm{FOR}_{\bar{x},X}^{\#\bar{z}:Z(\bar{z})}\varphi(\bar{x}, \bar{y}, X, \overline{Y})](\bar{u})$, σ_2 is the formula ψ, and U and Z do not occur in ξ. Since the translation of FO(S-FOR) to FO(FOR) does not introduce additional nesting in the head formula (cf. Proposition 3.3), the above formula is equivalent to one in FH-FO(FOR). ■

The construction described above, however, introduces relational parameters in head formulas. We next show that in general one cannot get rid of these parameters. Let PFH-FO(FOR), FO(FOR) *with pure first-order heads*, be the fragment of FH-FO(FOR) that forbids relation variables in head formulas of for-loops.

To prove inexpressibility results for PFH-FO(FOR), we introduce an extended version of the k-pebble game defined in Section 2. First, the Duplicator has to preserve partial isomorphisms between the pebbles as in the ordinary k-pebble game. But on top of that, he must also make sure that for any FO^k formula $\varphi(\bar{x}, \bar{y})$, if we fill in some pebbled elements $\bar{a}$ from the first structure $\mathcal{A}$ and take the corresponding pebbled elements $\bar{b}$ in the second structure $\mathcal{B}$ (or vice versa) then $|\{\bar{a}' \mid \mathcal{A} \models \varphi[\bar{a}, \bar{a}']\}| = |\{\bar{b}' \mid \mathcal{B} \models \varphi[\bar{b}, \bar{b}']\}|$. This game provides us with the following tool.

Lemma 4.6 *Let $\mathcal{Q}$ be a Boolean query. If for every k, there exist structures $\mathcal{A}_k$ and $\mathcal{B}_k$ such that $\mathcal{Q}(\mathcal{A}_k) \neq \mathcal{Q}(\mathcal{B}_k)$, and the Duplicator has a winning strategy in the extended k-pebble game on $\mathcal{A}_k$ and $\mathcal{B}_k$, then $\mathcal{Q}$ is not definable in* PFH-FO(FOR).

Proof. Let $\mathcal{D}^k_{\infty\omega}$ be the logic $\mathcal{L}^k_{\infty\omega}$ extended with counting quantifiers $\exists^{=i}x_j$ (meaning that there are exactly i elements x_j such that ...), for all natural numbers i and for all $j = 1, \dots, k$, where the counting quantifiers are applied to FO^k formulas only. Note that $\mathcal{D}^k_{\infty\omega}$ is a fragment of $\mathcal{C}^k_{\infty\omega}$, that is, $\mathcal{L}^k_{\infty\omega}$ with counting quantifiers (see, e.g., [Ott97]).

In close analogy with the Immerman-Lander pebble game for $\mathcal{L}^k_{\infty\omega}$ with counting [IL90], one can show that if the Duplicator has a winning strategy in the extended k-pebble game on the structures $\mathcal{A}$ and $\mathcal{B}$ with the pebbles placed initially on $\bar{a}$ and $\bar{b}$ in $\mathcal{A}$ and $\mathcal{B}$, respectively, then for every $\mathcal{D}^k_{\infty\omega}$ formula $\varphi(\bar{x})$

$$\mathcal{A} \models \varphi[\bar{a}] \quad \Leftrightarrow \quad \mathcal{B} \models \varphi[\bar{b}].$$

Every PFH-FO(FOR) formula is equivalent to a $\mathcal{D}^k_{\infty\omega}$ formula, for some k. Since PFH-FO(FOR) formulas can have free relation variables, these can also occur in the corresponding $\mathcal{D}^k_{\infty\omega}$ formulas. However, no such relation variable will occur in the scope of a counting quantifier.

We proceed by induction on the structure of PFH-FO(FOR) formulas. The only interesting case is a formula $\xi(\bar{u}, \bar{y}, \overline{Y})$ of the form

$$[\mathrm{FOR}^{\#\bar{z}:\psi(\bar{z},\bar{u},\bar{y})}_{\bar{x},X} \varphi(\bar{x}, \bar{y}, X, \overline{Y})](\bar{u}),$$

with ψ an FO^{k_1} formula for some k_1. We can assume that φ is a $D^{k_2}_{\infty\omega}$ formula for some k_2, where no relation variable occurs in the scope of a counting quantifier. Then ξ is equivalent to the formula

$$\bigvee_{i=1}^{\infty} \left((\exists^{=i}\bar{z})\psi(\bar{z}, \bar{u}, \bar{y}) \wedge \varphi^i(\bar{x}, \bar{y}, \overline{Y})\right).$$

Here, φ^i is defined inductively as follows. For $i = 0$, define φ^0 as any false formula. For $i > 0$, φ^i is the formula obtained from φ by replacing any occurrence of an X-atom $X(v_1, \dots, v_s)$ by

$$(\exists z_1) \dots (\exists z_s) \left(\bar{v} = \bar{z} \wedge (\exists x_1) \dots (\exists x_s)(\bar{x} = \bar{z} \wedge \varphi^i(\bar{x}, \bar{y}, \overline{Y}))\right).$$

Here, $z_j = x_{k+j}$ for $j = 1, \dots, s$. Note that φ^i is a $\mathcal{D}^{k_2+s}$-formula, where s is the arity of X. Hence, ξ is equivalent to a $\mathcal{D}^{k_1+k_2+s}$-formula.

The lemma now follows. Indeed, let $\mathcal{Q}$ be a Boolean query that satisfies the conditions of the lemma. Take any PFH-FO(FOR) sentence ξ. By the above there exists an equivalent $\mathcal{D}^k_{\infty\omega}$ sentence φ for some k. By assumption the Duplicator wins the extended k-pebble game on $\mathcal{A}_k$ and $\mathcal{B}_k$. Hence, φ, and thus ξ, cannot express $\mathcal{Q}$. ■

Using the above lemma we can show the following.

Theorem 4.7 PFH-FO(FOR) *is strictly weaker than* FO(FOR).

Proof. Consider the following Boolean query over the vocabulary of graphs. $\mathcal{Q}_4(\mathcal{G})$ is true if

(i) every connected component of $\mathcal{G}$ is ordered by the HKL query (cf. the paragraph before Section 3.3);

(ii) the number of elements in each connected component is larger than the number of connected components; and

(iii) there are exactly two isomorphism types of connected components and they appear in equal numbers.

We first show that this query is definable in FO(FOR):

(i) Let $\eta(x, y)$ be the FO(FOR) formula that defines the HKL query. We can easily relativize it in the standard way to each connected component: replace each $(\exists v) \dots$ by $(\exists v)(\mathtt{path}(z, v) \wedge \dots$ and each $(\forall v) \dots$ by $(\forall v)(\mathtt{path}(z, v) \to \dots$, where z is a variable not occurring in φ and $\mathtt{path}(z, v)$ is the FO(FOR) formula expressing that there is a path in the graph between z and v. This gives us the formula $\eta'(z, x, y)$. We now only have to check whether for each node z the formula $\eta'(z, x, y)$ defines a linear order.

(ii) The next sentence checks requirement (ii):

$$(\forall x)\Big(\mathtt{first}(x) \to (\forall y)\big([\mathrm{FOR}^{\#z:\mathtt{first}(z)}_{y,Y} y = x \vee (\exists y')(Y(y') \wedge \mathtt{succ}(y', y))](y) \to \neg\mathtt{last}(y)\big)\Big)$$

Here, $\mathtt{first}$ ($\mathtt{last}$) is a formula defining the first (last) elements of all components, and $\mathtt{succ}(y', y)$ is a formula expressing that y is the successor of y' in the ordering of a component.

(iii) Let $x \cong y$ be the FO(FOR) formula saying that x and y are the first elements of different but isomorphic components. We can express this using the orderings, because if the two components are isomorphic, the isomorphism must respect the ordering and is thus unique. Expressing that the isomorphism between the orderings is an automorphism of the graph can be done in first-order logic.

Now the formula

$$(\exists x)(\exists y)(\mathtt{first}(x) \wedge \mathtt{first}(y) \wedge x \neq y \wedge x \ncong y \\ \wedge (\forall z)(\mathtt{first}(z) \rightarrow x \cong z \vee y \cong z))$$

states that there are exactly two isomorphism types. The next sentence expresses that the two isomorphism types have to occur in equal numbers;

$$(\forall x)(\mathtt{first}(x) \rightarrow \\ (\forall y)([\mathrm{FOR}^{\#z:x\cong z}_{y,Y} y = x \vee (\exists y')(Y(y') \wedge \mathtt{succ}(y', y))](y) \\ \leftrightarrow \\ [\mathrm{FOR}^{\#z:x\ncong z \wedge \mathtt{first}(z)}_{y,Y} y = x \vee (\exists y')(Y(y') \wedge \mathtt{succ}(y', y))](y)).$$

Using Lemma 4.6, however, we can prove that $\mathcal{Q}_4$ is not definable in PFH-FO(FOR). We first observe that for any k, there are arbitrary large non-isomorphic connected graphs G_k and H_k such that:

- every FO^k formula is equivalent to a quantifier free one over G_k and H_k;
- $|\{\bar{g} \mid G_k \models \varphi[\bar{g}]\}| = |\{\bar{g} \mid H_k \models \varphi[\bar{g}]\}|$ for every FO^k formula φ;
- G_k and H_k are ordered by the HKL query.

Indeed, recall that every FO^k-formula is equivalent to a quantifier-free one on almost all graphs; this follows from the satisfaction by almost all graphs of the extension axioms for k variables [EF95]. Hence, the first item is clear. We do not have to worry about the third item because the HKL query orders almost all graphs.

Up to logical equivalence there are only a constant number of quantifier-free formulas in k variables (say N) and in a graph of n vertices a relation definable by such a formula must have cardinality between 0 and n^k. Hence, there are only $(n^k+1)^N$ possible sequences of cardinalities of the definable relations. There are at least $2^{n^2}/n!$ nonisomorphic directed graphs on n vertices. (Indeed, there are exactly 2^{n^2} directed graphs on a fixed set of n vertices, and no isomorphism class among them has more than $n!$ elements, hence there are at least $2^{n^2}/n!$ isomorphism classes.) Consequently, there must be two nonisomorphic directed graphs on n vertices, say G_k and H_k, such that for every FO^k formula φ:

$$|\{\bar{g} \mid G_k \models \varphi[\bar{g}]\}| = |\{\bar{g} \mid H_k \models \varphi[\bar{g}]\}|.$$

We have choices of G_k and H_k for all large enough n, so we can use an $n > 2k+2$.

Let $\mathcal{A}_k$ be the disjoint union of $k+1$ copies of G_k and $k+1$ copies of H_k, and let $\mathcal{B}_k$ be the disjoint union of k copies of G_k and $k+2$ copies of H_k. We show that the Duplicator has a winning strategy in the extended k-pebble game on $\mathcal{A}_k$ and $\mathcal{B}_k$. The theorem then follows by Lemma 4.6 because $\mathcal{A}_k$ satisfies query $\mathcal{Q}_4$ but $\mathcal{B}_k$ does not.

In the game, for pebbled elements $\bar{a}$ and $\bar{b}$, we only have to show that

$$|\{\bar{a}' \mid \mathcal{A}_k \models \varphi[\bar{a},\bar{a}']\}| = |\{\bar{b}' \mid \mathcal{B}_k \models \varphi[\bar{b},\bar{b}']\}|, \tag{4}$$

for all FO^k formulas φ. To simplify the proof, we add the relation $\sim$ to the vocabulary. In both $\mathcal{A}_k$ and $\mathcal{B}_k$ this relation holds for two nodes if they appear in the same connected component. The following lemma now says that in (4) we only have to consider quantifier-free formulas.

Lemma 4.8 *On $\mathcal{A}_k$ and $\mathcal{B}_k$ each FO^k formula is equivalent to a quantifier-free one.*

Proof. This follows by an Ehrenfeucht-Fraïssé game argument. First note that if $\bar{a} \mapsto \bar{b}$ is a partial isomorphism in the presence of $\sim$, then the Duplicator wins the ordinary k-pebble game on $(\mathcal{A}_k, \bar{a})$ and $(\mathcal{B}_k, \bar{b})$. Indeed, he just plays independently in each component. Because all of them satisfy the extension axioms for k variables, and $\bar{a} \mapsto \bar{b}$ is a partial isomorphism, he can always find vertices to answer all the moves of the Spoiler. It follows that $(\mathcal{A}_k, \bar{a})$ and $(\mathcal{B}_k, \bar{b})$ satisfy the same FO^k formulas if they have the same atomic type in the edge relation and $\sim$. Hence, every FO^k formula is equivalent to a union of atomic types. ■

In the extended game on $\mathcal{A}_k$ and $\mathcal{B}_k$, the Duplicator uses an "exact mirror strategy": play to win the standard game, plus *always respond in isomorphic components and according to an isomorphism.* That is, he ensures that the partial isomorphism required for the game can always be extended to a partial isomorphism defined for all the members of the connected components in which the pebbles are located. Because, initially, there are no pebbles on the board and there are enough copies of the graphs H_k and G_k in both structures, the Duplicator can easily maintain this strategy.

It remains to show that under this strategy the conditions of the extended k-pebble game are always satisfied. The key observation is that the cardinalities of FO^k definable relations (even if $\sim$ can be used) are functions of cardinalities of certain quantifier-free definable relations over the components. In the absence of pebbles these cardinalities are equal in components isomorphic to G_k and H_k, and the components which contain parameters are isomorphic.

Let $\bar{a}$ ($\bar{b}$) be a sequence of pebbled elements in $\mathcal{A}_k$ ($\mathcal{B}_k$). We show a one-to-one correspondence between $\alpha = \{\bar{a}' \mid \mathcal{A}_k \models \varphi[\bar{a}, \bar{a}']\}$ and $\beta = \{\bar{b}' \mid \mathcal{B}_k \models \varphi[\bar{b}, \bar{b}']\}$, where φ is a quantifier-free FO^k formula using $\sim$. The partial isomorphism of the pebbled elements extends to a partial isomorphism of the components and can be further extended to a partial isomorphism defined for all the elements except those of one connected component isomorphic to G_k in $\mathcal{A}_k$, and one isomorphic to H_k in $\mathcal{B}_k$. Call these components *exceptional.*

Take $\bar{a}' \in \alpha$ and split it into the *exceptional subtuple* $e(\bar{a}')$, which consists of the elements coming from the exceptional component, and the rest called the *normal subtuple* $n(\bar{a}')$. Assume for convenience that $\bar{a}' = n(\bar{a}'), e(\bar{a}')$. The normal subtuple $n(\bar{a}')$ is put into correspondence according to the above mentioned partial isomorphism into a subtuple we denote $n(\bar{b}')$. Moreover, the sets

$$\gamma := \{\bar{c} \in \text{exceptional-component-of}(\mathcal{A}_k) \mid \mathcal{A}_k \models \varphi[\bar{a}, n(\bar{a}'), \bar{c}]\}$$

and

$$\delta := \{\bar{d} \in \text{exceptional-component-of}(\mathcal{B}_k) \mid \mathcal{B}_k \models \varphi[\bar{b}, n(\bar{b}'), \bar{d}]\}$$

are described by the same parameter-free FO^k formula evaluated over the exceptional component, only. Indeed, consider $\varphi[\bar{a}, n(\bar{a}')]$ and $\varphi[\bar{b}, n(\bar{b}')]$ (with unsubstituted variables to be filled in by elements of the exceptional subtuples), and replace atoms (built with equality, $\sim$ and the edge relation) by their truth values wherever possible. For all atoms concerning subtuples of $\bar{a}, n(\bar{a}')$ and $\bar{b}, n(\bar{b}')$ we get the same truth values in both formulas, because the arguments in atoms are taken from isomorphic fragments of structures $\mathcal{A}$ and $\mathcal{B}$. For positive atoms consisting of one argument from $\bar{a}, n(\bar{a}')$ or $\bar{b}, n(\bar{b}')$ and one variable to be filled in by an element of the exceptional subtuple, we always get false. This is so because the two arguments are taken from distinct connected components, so there can be neither edge, $\sim$ nor equality between them. Finally, we replace the atoms $x \sim y$ to be filled in by elements coming from exceptional components by their logical value, which is true. After this replacement we get two identical quantifier- and parameter-free formulas consisting entirely of atoms concerning elements of the exceptional subtuples, as desired.

Since we already had a one-to-one correspondence between the non-exceptional subtuples of α and β, we thus get a full bijection between α and β, as had to be shown. ∎

4.2.2 Nesting in the body

Let FB-FO(FOR) be the fragment of FO(FOR) that does not allow for-loops in body formulas. That is, only first-order bodies are allowed.

For a graph $\mathcal{G}$, let $n * \mathcal{G}$ denote the disjoint union of n copies of $\mathcal{G}$. Let $\mathcal{Q}_5$ be the following query on graphs. $\mathcal{Q}_5(\mathcal{H})$ is true if there exists a connected graph $\mathcal{G}$ such that

(i) $\mathcal{G}$ is ordered by the HKL query;

(ii) $\mathcal{H} \cong n * \mathcal{G}$; and

(iii) $n < |\mathcal{G}|$.

This query is definable in FO(FOR): as in the proof of Theorem 4.7 we can check whether each connected component can be ordered by the HKL query, whether all connected components are of the same isomorphism type, and using the ordering of a connected component we can check whether $n < |\mathcal{G}|$.

We next show in a sequence of lemmas that $\mathcal{Q}_5$ is not definable in FB-FO(FOR) thus proving the following.

Theorem 4.9 FB-FO(FOR) *is strictly weaker than* FO(FOR).

The key idea towards the proof of this theorem will be that if $\mathcal{G}$ satisfies the k-variable extension axioms for sufficiently large k,[EF95] then first-order bodies of for-loops start to cycle after a bounded number of iterations, while $\mathcal{Q}_5$ requires counting up to $|\mathcal{G}|$.

In the following we fix a connected graph $\mathcal{G}$ that is ordered by the HKL query. This implies that $\mathcal{G}$ is rigid.

Definition 4.10 For natural numbers $n \leq m$, a formula $\varphi(\bar{x})$ is called (n,m)-*embedding preserved* if for every embedding $e : n * \mathcal{G} \to m * \mathcal{G}$ and every tuple $\bar{a}$ of elements of $n * \mathcal{G}$, we have

$$n * \mathcal{G} \models \varphi[\bar{a}] \quad \Leftrightarrow \quad m * \mathcal{G} \models \varphi[e(\bar{a})].$$

Note that the just introduced notion depends on the graph $\mathcal{G}$. Intuitively, if φ is (n,m)-embedding preserved, then its interpretation in $n * \mathcal{G}$ completely determines its interpretation in $m * \mathcal{G}$ because the latter can be completely covered by embeddings of the former. Also note that since $\mathcal{G}$ is rigid each embedding is completely determined by which connected component of $n * \mathcal{G}$ is mapped onto which connected component of $m * \mathcal{G}$.

Obviously, atomic formulas are (n,m)-embedding preserved for all $n \leq m$. In a sequence of lemmas we prove bounds on the values of n and m for which FB-FO(FOR) formulas are (n,m)-embedding preserved.

Lemma 4.11 (First-order case) *Let $n \geq k$. If φ and ψ are formulas with at most k variables which are (n,m)-embedding preserved, then so are $\neg\varphi$, $\varphi \vee \psi$ and $(\exists x)\varphi$.*

Proof. The cases of negation and disjunction are straightforward, so let us focus on $(\exists x)\varphi(x,\bar{y})$. If $n * \mathcal{G} \models (\exists x)\varphi[x,\bar{a}]$, then there exists a b such that $n * \mathcal{G} \models \varphi[b,\bar{a}]$. Let e be an embedding. By the induction hypothesis we get that $m * \mathcal{G} \models \varphi[e(b), e(\bar{a})]$. Hence, $m * \mathcal{G} \models (\exists x)\varphi[x, e(\bar{a})]$.

Conversely, suppose that $m * \mathcal{G} \models (\exists x)\varphi[x, e(\bar{a})]$ for an embedding e. Then there exists a c such that $m * \mathcal{G} \models \varphi[c, e(\bar{a})]$. We distinguish two cases:

(i) There exists a b such that $e(b) = c$. Then by the induction hypothesis, we get $n * \mathcal{G} \models \varphi[b,\bar{a}]$.

(ii) If c does not belong to the image of e, then it must belong to a connected component of $m * \mathcal{G}$ which is disjoint from the image of e. Since $\bar{a}$ is of length at most $k-1$, there is a connected component of $n * \mathcal{G}$, denoted by C, that does not contain elements from $\bar{a}$. Now define e' as the embedding that maps C to the connected component of $m * \mathcal{G}$ that contains c and that is equal to e on the other connected components of $n * \mathcal{G}$. This brings us to case (i).

■

The next lemma says that the cardinality of the relations defined by head-formulas can be expressed by polynomials.

Lemma 4.12 *For any* FO(FOR) *formula* $\varphi(\bar{x}, \bar{y})$ *and interpretations* $\bar{a}$ *for* $\bar{x}$ *in* $n * \mathcal{G}$*, there exists a polynomial* $p(m)$ *such that for every* m *for which* φ *is* (n, m)*-embedding preserved and for every embedding* $e : n * \mathcal{G} \to m * \mathcal{G}$

$$p(m) = \big|\{\bar{c} \mid m * \mathcal{G} \models \varphi[e(\bar{a}), \bar{c}]\}\big|.$$

Proof. Fix n, e and $\bar{a}$ as in the statement of the lemma. Observe that

$$\{\bar{c} \mid m * \mathcal{G} \models \varphi[e(\bar{a}), \bar{c}]\} = \left\{ e'(\bar{b}) \;\middle|\; \begin{array}{c} \bar{b} \in n * \mathcal{G}, n * \mathcal{G} \models \varphi[\bar{a}, \bar{b}] \\ e' : n * \mathcal{G} \to m * \mathcal{G} \text{ embedding with } e'(\bar{a}) = e(\bar{a}) \end{array} \right\}.$$

Indeed, the inclusion $\supseteq$ holds because φ is (n, m)-embedding preserved, and $\subseteq$ because φ is (n, m)-embedding preserved and the whole of $m * \mathcal{G}$ can be covered by embeddings of $n * \mathcal{G}$.

We are going to limit the choice of e' and $\bar{b}$ on the r.h.s. above so that every $\bar{c}$ on the l.h.s. above will be equal to $e'(\bar{b})$ for exactly one pair $(e', \bar{b})$. Then the cardinality we are going to compute will be equal to the number of the chosen pairs.

First, for tuples $\bar{b}, \bar{b}'$ of elements of $n * \mathcal{G}$ we write $\bar{b} \sim \bar{b}'$ iff there is an automorphism f of $n * \mathcal{G}$ with $f(\bar{a}) = \bar{a}$ and $f(\bar{b}) = \bar{b}'$. The relation $\sim$ is clearly an equivalence relation. Let Q be a set of representatives of the equivalence classes of $\sim$. We claim that

$$\{\bar{c} \mid m * \mathcal{G} \models \varphi[e(\bar{a}), \bar{c}]\} = \left\{ e'(\bar{b}) \;\middle|\; \begin{array}{c} \bar{b} \in n * \mathcal{G}, n * \mathcal{G} \models \varphi[\bar{a}, \bar{b}], \bar{b} \in Q \\ e' : n * \mathcal{G} \to m * \mathcal{G} \text{ embedding with } e'(\bar{a}) = e(\bar{a}) \end{array} \right\}.$$

The inclusion $\supseteq$ is obvious. To prove $\subseteq$, note that for any $\bar{b}$ satisfying $n * \mathcal{G} \models \varphi[\bar{a}, \bar{b}]$ there exists $\bar{b}' \in Q$ and an automorphism f of $m * \mathcal{G}$ which is the identity on $\bar{a}$ and such that $\bar{b} = f(\bar{b}')$. Then, for any embedding e' as above, $e'(\bar{b}) = (e' \circ f)(\bar{b}')$ and $e' \circ f$ is an embedding with $(e' \circ f)(\bar{a}) = e(\bar{a})$.

Now, for every $\bar{b}$ we introduce an equivalence relation $\approx_{\bar{b}}$ on embeddings e' satisfying $e'(\bar{a}) = e(\bar{a})$. It is defined by $e' \approx_{\bar{b}} e''$ iff $e'(\bar{b}) = e''(\bar{b})$. Because $\mathcal{G}$ is rigid it is equivalent to the statement that e' and e'' are equal on the components in which elements of $\bar{a}$ and $\bar{b}$ are located. Let $R_{\bar{b}}$ be a set of representatives of all the equivalence classes of $\approx_{\bar{b}}$.

Certainly,

$$\left\{ e'(\bar{b}) \;\middle|\; \begin{array}{c} \bar{b} \in n * \mathcal{G}, n * \mathcal{G} \models \varphi[\bar{a}, \bar{b}], \bar{b} \in Q \\ e' : n * \mathcal{G} \to m * \mathcal{G} \text{ embedding with } e'(\bar{a}) = e(\bar{a}) \end{array} \right\} =$$
$$\left\{ e'(\bar{b}) \;\middle|\; \bar{b} \in n * \mathcal{G}, n * \mathcal{G} \models \varphi[\bar{a}, \bar{b}], \bar{b} \in Q, e' \in R_{\bar{b}} \right\}.$$

Moreover, for $\bar{b}, \bar{b}' \in Q$ and $e' \in R_{\bar{b}}, e'' \in R_{\bar{b}'}$, if $e'(\bar{b}) = e''(\bar{b}')$ then $\bar{b} = \bar{b}'$ and $e' = e''$. Indeed, $e' \circ (e'')^{-1}$ is a partial automorphism of $n * \mathcal{G}$ sending $\bar{b}'$ to $\bar{b}$ and $\bar{a}$ to $\bar{a}$. By rigidity of $\mathcal{G}$ it is defined at least for the whole components in which $\bar{a}$ and $\bar{b}'$ are located, and hence it can be extended to a total automorphism. Consequently $\bar{b} = \bar{b}'$, because they are members of Q. It follows that $e' = e''$, because they are members of $R_{\bar{b}}$.

We have

$$|\{\bar{c} \mid m * \mathcal{G} \models \varphi[e(\bar{a}), \bar{c}]\}| = |\{(\bar{b}, e') \mid \bar{b} \in Q,\ n * \mathcal{G} \models \varphi[\bar{a}, \bar{b}],\ \text{and } e' \in R_{\bar{b}}\}|.$$

Let ℓ be the number of components in which elements from $\bar{a}$ are located. Let $k(\bar{b})$ denote the number of components in which elements of $\bar{b}$ are located, excluding those in which elements from $\bar{a}$ are located. Note that $k(\bar{b})$ is equal for $\sim$-equivalent tuples $\bar{b}$. For every $\bar{b} \in Q$, the number of equivalence classes of $\approx_{\bar{b}}$ is $(m - \ell)(m - \ell - 1) \ldots (m - \ell - k(\bar{b}) + 1)$. Indeed, $\mathcal{G}$ is rigid and all our freedom in constructing an embedding is restricted to the choice which component should be mapped onto which. Moreover, the choice for the components in which $\bar{a}$ is located has been already done, and all the choices for components in which there are no elements from $\bar{b}$ do not count, because all embeddings which differ only on those components are in the same $\approx_{\bar{b}}$ equivalence class. Counting the number of pairs, for the first component with elements from $\bar{b}$ and no elements from $\bar{a}$ we have $m - \ell$ choices, for the second $m - \ell - 1$, ... , and for the last, $k(\bar{b})$-th component we have $m - \ell - k(\bar{b}) + 1$ choices.

Consequently, the cardinality from the thesis of the lemma is

$$\sum_{\bar{b} \in Q, n*\mathcal{G} \models \varphi[\bar{a},\bar{b}]} (m-\ell)\dots(m-\ell-k(\bar{b})+1),$$

a polynomial in m. ∎

Analyzing the formula for p in the above proof we get the following.

Corollary 4.13 *If p is non-zero then $p(m) \geq m-k$, with k the number of variables in φ.*

Lemma 4.14 (FOR case) *Let k, n, d be natural numbers such that $n \geq d+k$, and let $\psi(\bar{z},\bar{u},\bar{y})$ in* FB-FO(FOR) *and $\varphi(\bar{x},X)$ in FO, both using at most k variables, be such that*

- *ψ is $(n, n(d!+1))$-embedding preserved;*
- *there are at most d different stages of φ in $n*\mathcal{G}$; and*
- *each stage formula φ^i is $(n, n(d!+1))$-embedding preserved.*

Then $[\mathrm{FOR}^{\#\bar{z}:\psi}_{\bar{x},X}\varphi](\bar{u})$ is $(n, n(d!+1))$-embedding preserved.

Proof. Let $m = n(d!+1)$. The stages of φ in $n*\mathcal{G}$ and $m*\mathcal{G}$ are in one-to-one correspondence because $m*\mathcal{G}$ can be completely covered by embeddings and the stages are (n,m)-embedding preserved. Consequently, the sequences of stages require the same number of steps to arrive to the first cyclic state and have the same cycle size.

Since ψ is (n,m)-embedding preserved, for every choice of parameters $\bar{a}$ to be substituted for $\bar{y},\bar{u}$, according to Lemma 4.12 there exists a polynomial $p(n)$ such that $|\{\bar{b} \mid n*\mathcal{G} \models \psi[\bar{b},\bar{a}]\}| = p(n)$ and $|\{\bar{c} \mid m*\mathcal{G} \models \psi[\bar{c},e(\bar{a})]\}| = p(m)$ for any embedding e (for the former we use that ψ is (n,n)-embedding preserved). Because $n \equiv m \pmod{c}$ for any $c \leq d$ we have that $p(n) \equiv p(m) \pmod{c}$. Consequently because there are at most d different stages, the for-loop halts in both structures in the same stage, irrespective of the chosen parameters in the head formula. Note that the condition $n \geq d+k$ ensures by Corollary 4.13 that $p(n) \geq d$, so the cycles in the sequence of stages must be achieved in both structures, unless the numbers of iterations to be performed in both structures are the same, because the polynomial is the constant zero. ∎

Lemma 4.15 *Query $\mathcal{Q}_5$ is not expressible in* FB-FO(FOR).

Proof. Towards a contradiction, suppose that the FB-FO(FOR) formula ξ defines $\mathcal{Q}_5$. Let ξ have no more than $k > 2$ first-order variables. As in the proof of Theorem 4.7, we know that for sufficiently large N there exists a graph on N vertices that satisfies the extension axioms for k variables and is ordered by the HKL query. Moreover, we may assume that the graph is rigid because almost all graphs are rigid [EF95]. So, we fix one such graph of cardinality $N > d+k+1$, where d is the maximal number of stages a formula of FO^k can induce in a disjoint union of graphs satisfying extension axioms with k variables (this number is finite as each FO^k formula is equivalent to a quantifier-free one using $\sim$, see Lemma 4.8).

By a straightforward induction using using Lemma 4.11 and Lemma 4.14, we see that ξ is $(n, n(d!+1))$-embedding preserved (with respect to $\mathcal{G}$) for each $n \geq d+k$. Moreover, each formula that is (n, m)-embedding preserved and (m, p)-embedding preserved is also (n, p)-embedding preserved. Therefore the sentence ξ is $(n, n(d!+1)^s)$-embedding preserved for any s, and, because $(d+k) * \mathcal{G} \models \xi$, we have $(d+k)(d!+1)^s * \mathcal{G} \models \xi$ for all s. However, the query $\mathcal{Q}_5$ is false in $n * \mathcal{G}$ for $n \geq N$, which leads to the desired contradiction. ■

The method we have used to distinguish FB-FO(FOR) and FO(FOR) is much stronger than necessary to do just that. We illustrate this now.

Let us call a first-order query $\varphi(X, \bar{x})$ *polynomial* if its number of stages is bounded by a polynomial in the cardinality of the structure. The question whether PFP with polynomial first order bodies is equivalent to the whole of PFP is equivalent to the question whether PTIME=PSPACE. Indeed, PFP with polynomial bodies is sandwiched between IFP and PFP, which are equal iff PTIME=PSPACE [AV95]. So when the latter equality holds, then PFP collapses to IFP (and thus to a fragment of PFP with polynomial bodies). On the other hand, if the equality does not hold, then in the ordered world PFP=PSPACE $\supsetneq$ PTIME, while PFP with polynomial bodies consists entirely of queries computable in PTIME, and thus strictly contained in PFP.

So the question of equivalence between two forms of first-order bodies in PFP, unrestricted and polynomial, is wide open. In the FOR world, however, the fragment with first-order polynomial bodies is weaker than the logic with arbitrary first-order bodies. Furthermore, this result does not depend on any complexity theoretic assumptions.

Theorem 4.16 FOR(FOR) *with polynomial* FO *formulas as bodies is strictly weaker than* FO(FOR) *with unrestricted FO bodies.*

Proof. Let $\mathcal{G}$ be an ordered set, i.e., a structure with no other relations besides the order. Define $\mathcal{Q}_6$ as the following query: $\mathcal{Q}_6(\mathcal{H})$ is true iff $\mathcal{H}$ is isomorphic to $n * \mathcal{G}$ and $n < 2^{|\mathcal{G}|}$. I.e., $\mathcal{H}$ should be a partial order consisting of incomparable chains of equal lengths and the number of chains should be smaller than 2 to the length of the chain. $\mathcal{Q}_6$ can be expressed in FO(FOR) with unrestricted FO bodies as a conjunction of:

- a first order formula saying that the structure is a poset which is a union of chains, i.e., for every element t, all elements smaller than t and all elements bigger than t are linearly ordered;
- a straightforward FO(FOR) formula saying that all chains are equally long; and
- the following formula

  $$\neg\big((\exists t)\,\mathrm{first}(t) \wedge (\forall y)\big(t \leq y \rightarrow [\mathrm{FOR}_{x,X}^{\#z:\mathrm{first}(z)} Succ(X,x,t)](y)\big)\big),$$

  where $Succ$ computes the successor of the current relation X viewed as a binary expansion of length equal to the length of the chain with first element t. Further, if the successor cannot be represented, then no increment is made. By selecting the first element of each chain in the head of the for-loop, the body of the for-loop iterates exactly n times. The output relation is not the whole chain exactly when the number of minimal elements z is smaller than 2 to the length of the chain minus 1.

Now we prove that $\mathcal{Q}_6$ cannot be defined in FO(FOR) with polynomial FO bodies.

Suppose $\mathcal{Q}_6$ is defined by an FO(FOR) formula ξ with k variables and polynomial FO formulas as bodies of all its for-loops. Let these polynomials be uniformly bounded by the polynomial $p(n)$.

By a straightforward structural induction ξ is $(n, n(p(|\mathcal{G}|)!+1))$-embedding preserved for each $n \geq p(|\mathcal{G}|) + k$, and therefore it is (by transitivity) $(n, n(p(|\mathcal{G}|)!+1)^s)$-embedding preserved for every s. Since $(p(|\mathcal{G}|)+k)*\mathcal{G} \models \xi$ for $\mathcal{G}$ of sufficiently large cardinality, $(p(|\mathcal{G}|) + k)(p(|\mathcal{G}|)! + 1)^s * \mathcal{G} \models \xi$ for all s. But $\mathcal{Q}_6$ does not have this property, a contradiction. ■

5 Discussion

We studied two languages based on for-loops: BQL and FO(FOR). BQL is a programming like language while FO(FOR) is based on a partial fixpoint operator. Actually, BQL and FO(FOR) can be considered as the for-loop variants of the loop languages RQL and FO(PFP). In contrast to the equivalence of the latter, the former are not equally expressive. In brief, this is because the use of parameters turns out to play a much more powerful role in for-loops than in while loops. One striking consequence of the strength of parameters in for-loops is that, unlike BQL, FO(FOR) can actually define queries whose output relation is not locally closed under k-variable equivalence for any k. [6]

We summarize the results obtained and mention some open problems:

BQL and FO(FOR). BQL is strictly subsumed by FO(FOR). The fragment of FO(FOR) in which individual parameters are admitted neither in heads nor in bodies, however, is equivalent to BQL.

Comparison with other logics. We compared FO(FOR) with other known logics. FO(FOR) and FO(PFP) are incomparable if PTIME $\neq$ PSPACE and FO(FOR) lies strictly between inflationary fixpoint-logic with modular counting and partial fixpoint-logic with (proper) counting. For the last separation, we showed that FO(FOR) cannot check whether two sets have the same cardinality. This result is a generalization of the same result for BQL announced by Chandra [Cha81, Cha88]. The inflationary variant of FO(FOR) lies strictly between inflationary fixed point logic and full FO(FOR). This separation should be contrasted with the corresponding issue in the case of while-loops. Recall that with while-loops, a separation of inflationary from full partial fixed-point logic would amount to no less than a separation of PTIME from PSPACE.

Nesting. Finally, we considered nesting of for-loops.

- Unnested BQL is strictly weaker than BQL, but one level of nesting suffices to simulate all of BQL. For the separation we used a non-Boolean

[6] Here, "local closure" would mean for some fixed k that the output relation is closed under k-variable equivalence within each individual structure; of course not even BQL is closed under "global" finite-variable equivalence as it allows modular counting.

query. It remains open whether there is a Boolean query definable in BQL but not in unnested BQL.

- In the case of FO(FOR), we have two kinds of nesting: nesting in the head and nesting in the body of FOR-operators. Nesting in body formulas matters although it remains open whether nesting up to a certain level is sufficient. Nesting in heads is essential when relational parameters are not admitted.

Some of the separations obtained are technically rather involved, using quite some of the machinery developed in finite model theory and descriptive complexity for the study of fixed-point logics, plus specially adapted game techniques and counting arguments, some of them based on random graphs.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AV91] S. Abiteboul and V. Vianu. Datalog Extensions for Database Queries and Updates. *Journal of Computer and System Sciences*, 43(1): 62-124, 1991.

[AV95] S. Abiteboul and V. Vianu. Computing with first-order logic. *Journal of Computer and System Sciences*, 50(2):309–335, 1995.

[CH82] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1):99–128, 1982.

[Cha81] A. Chandra. Programming primitives for database languages. In *Conference Record, 8th ACM Symposium on Principles of Programming Languages*, pages 50–62, 1981.

[Cha88] A. Chandra. Theory of database queries. In *Proceedings of the Seventh ACM Symposium on Principles of Database Systems*, pages 1–9. ACM Press, 1988.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.

[EFT94] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer-Verlag, second edition, 1994.

[GO93] E. Grädel and M. Otto. Inductive definability with counting on finite structures. In E. Börger, editor, *Computer Science Logic*, volume 702 of *Lecture Notes in Computer Science*, pages 231–247. Springer-Verlag, 1993.

[HKL96] L. Hella, Ph. G. Kolaitis, and K. Luosto. Almost everywhere equivalence of logics in finite model theory. *Bulletin of Symbolic Logic*, 2(4):422–443, 1996.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[Imm98] N. Immerman. *Descriptive Complexity*. Springer, 1998.

[IL90] N. Immerman and E. S. Lander. Describing graphs: a first-order approach to graph canonization. In A. Selman, editor, *Complexity Theory Retrospective*, pages 59–81. 1990.

[KV95] Ph. G. Kolaitis and Jouko A. Väänänen. Generalized quantifiers and pebble games on finite structures. *Annals of Pure and Applied Logic*, 74(1):23–75, 1995.

[Ott96] M. Otto. The expressive power of fixed-point logic with counting. *Journal of Symbolic Logic*, 61(1):147–176, 1996.

[Ott97] M. Otto. *Bounded Variable Logics and Counting*, volume 9 of *Lecture Notes in Logic*. Springer, 1997.

[Var82] M. Vardi. The complexity of relational query languages. In *14th ACM Symposium on Theory of Computing*, pages 137–146, 1982.