JOURNAL OF COMPUTER AND SYSTEM SCIENCES 51, 110–134 (1995)

On the Expressive Power of Datalog: Tools and a Case Study*

PHOKION G. KOLAITIS†

Computer and Information Sciences, University of California, Santa Cruz, California 95064

AND

MOSHE Y. VARDI‡

IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120

Received March 23, 1993

We study here the language Datalog($\neq$), which is the query language obtained from Datalog by allowing equalities and inequalities in the bodies of the rules. We view Datalog($\neq$) as a fragment of an infinitary logic L^{ω} and show that L^{ω} can be characterized in terms of certain two-person pebble games. This characterization provides us with tools for investigating the expressive power of Datalog($\neq$). As a case study, we classify the expressibility of *fixed subgraph homeomorphism* queries on directed graphs. S. Fortune, J. Hopcroft, and J. Wyllie (*Theoret. Comput. Sci.* **10** (1980), 111–121) classified the computational complexity of these queries by establishing two dichotomies, which are proper only if $P \neq NP$. Without using any complexity-theoretic assumptions, we show here that the two dichotomies are indeed proper in terms of expressibility in Datalog($\neq$).

1. INTRODUCTION

The study of rule-based query languages has been a focal area of research in database theory during the past few years [Ull89]. A significant part of this research has concentrated on the language Datalog, which, in its purest form, is the language of function-free and negation-free Horn clauses. As a query language, Datalog can be viewed as the data sublanguage of general logic programming. Datalog programs always express queries that are computable in time bounded by a polynomial in the size of the underlying database. Moreover, in spite of its syntactical simplicity, Datalog can capture queries that are complete for polynomial time computations, such as the *path systems* query [Coo74].

The exact expressive power of Datalog is completely understood in certain cases. More specifically, if only *ordered* databases are considered, then a query is expressible by a Datalog program if and only if it is computable in polynomial time. This result is implicit in [Imm86, Var82] and stated explicitly in [Pap85, BG87].

The picture, however, changes dramatically on general (unordered) databases. In this case, as pointed out first in [CH85], there are natural polynomial time properties that are not expressible by any Datalog program. This follows easily from the fact that, in general, Datalog prgrams compute queries that are *strongly monotone*; i.e., they are preserved by adding tuples to database predicates (EDBs), or by adding elements to the universe of the database, or by identifying elements of the universe of the database (i.e., collapsing multiple elements into a single element). Thus, if a query is not strongly monotone, then there is no Datalog program that expresses it.

One natural way to increase the expressive power of Datalog, without changing the semantics, is to allow equalities $u = v$ and inequalities $u \neq v$ in the bodies of the rules. After all, these constructs are quite natural in a database query language. This extension of Datalog, which we will denote by Datalog($\neq$), was considered first by Shmueli [Shm87], who showed that it is more expressive than Datalog, and then it was investigated further in [GMSV87, AG89], where it was shown that Datalog($\neq$) differs from Datalog in more aspects than just expressive power. If a query is computed by a Datalog($\neq$) program, then it has the property that it is *monotone*; i.e., it is preserved by adding new elements to the universe of the database and by adding tuples to the database predicates. As a result, there is no Datalog($\neq$) program that expresses the complement of the transitive closure query, or any other polynomial time query that lacks the above monotonicity property.

The preceding remarks establish that the Datalog($\neq$) queries and, a fortiori, the Datalog queries, form a proper subset of the polynomial time queries. So far, however, not

* A preliminary version of this paper appeared in "Proceedings 9th ACM Symp. on Principles of Database Systems, March 1990," pp. 61–71.

† Partially supported by NSF Grant CCR-8905038.

‡ Present address: Computer Science Department, Rice University, Houston, Texas 77251.

0022-0000/95 $12.00

much progress has been made towards a complete understanding of which monotonic queries are expressible in Datalog variants. Lakshmanan and Mendelzon [LM89] studied this problem and made some progress by showing that the *even simple path* query, i.e., the query whether a directed graph contains a simple path of even length between two distinguished nodes, is not expressible in Datalog($\neq$). Even simple path is a monotone query that is known to be NP-complete. Although the result of Lakshmanan and Mendelzon [LM89] is an immediate consequence of P $\neq$ NP, the point of it is that it can be established directly without appealing to complexity-theoretic conjectures.

In this paper we continue the study of the expressive power of Datalog($\neq$) by developing tools and focusing on a case study. The main feature of our approach is the consideration of Datalog variants as fragments of the logic $L_{\infty\omega}$. This is an infinitary logic that extends first-order logic by allowing infinitary conjunctions and disjunctions (cf. [Kei71]). It turns out that Datalog($\neq$) can be viewed as a strict fragment of L^{ω}, the latter being the existential negation-free fragment of $L_{\infty\omega}$, where, in addition, formulas are restricted to contain a finite number of distinct variables. Since L^{ω} contains Datalog($\neq$), one can prove negative results about the expressive power of Datalog($\neq$) by proving negative results about the expressive power of L^{ω}.

The advantage of dealing with L^{ω}, instead of directly dealing with Datalog($\neq$), is that the expressive power of L^{ω} can be completely characterized in terms of *existential pebble games*, which are a refinement of the pebble games considered in Immerman [Imm82]. Indeed, a variant of existential pebble games, called *inductive pebble games*, was already used by Lakshmanan and Mendelzon [LM89] to prove results directly about Datalog($\neq$). However, the exact relationship between inductive pebble games and Datalog($\neq$) is unclear, while, in contrast, we show that existential pebble games completely characterize the expressive power of L^{ω}.

We demonstrate the usefulness of our tools by considering a class of queries that we call *pattern-based* queries. Intuitively, a structure satisfies a pattern-based query Q if it contains a pattern from a set of patterns that can be generated in polynomial time. The even simple path query of Lakshmanan and Mendelzon is an example of a pattern-based query. We show that for pattern-based queries there is an intimate relationship between their computational complexity and their expressibility in L^{ω}. More precisely, we use our game-theoretic characterization of L^{ω} to show that if a pattern-based query is expressible in L^{ω}, then it can be evaluated in polynomial time. It follows immediately that if P $\neq$ NP, then no pattern-based NP-complete query, such as the even simple path query, is expressible in L^{ω}.

The connection between complexity and expressibility is further illuminated by considering the class of *fixed subgraph homeomorphism* queries. Here we have a fixed pattern graph H and we have to determine whether it can be homeomorphically embedded in a given directed graph. Thus, fixed subgraph homeomorphism queries form a subclass of pattern-based queries. The complexity of fixed subgraph homomorphism queries was completely classified by Fortune *et al.* [FHW80] in terms of two dichotomies. They defined a class C of pattern graphs and showed that H-subgraphs homeomorphism query is (1) polynomial for pattern graphs H in C, (2) NP-complete for pattern graphs H in the complement $\bar{C}$ of C, and (3) polynomial for acyclic input graphs. Of course, the two dichotomies, pattern graphs in C versus pattern graphs in $\bar{C}$ and general input graphs versus acyclic input graphs, are proper only if P $\neq$ NP.

Without using any complexity-theoretic assumptions, we show here that the two dichotomies are indeed proper in terms of expressibility in Datalog($\neq$). More specifically, on the positive side we show that the H-subgraph homeomorphism query is expressible in Datalog($\neq$) for every pattern graph H in the class C, and we also prove that if only acyclic graphs are allowed as inputs, then the H-subgraph homeomorphism query is expressible in Datalog($\neq$) for every pattern graph H. The first result uses the reduction in [FHW80] of H-subgraph homeomorphism queries for pattern graphs H in C to network flow queries and the well-known max-flow min-cut theorem (cf. [Bol79]). For acyclic input graphs, Fortune *et al.* [FHW80] show that all fixed subgraph homeomophism queries can be reduced to questions about winning strategies in certain *single-player* pebble games on the input graph. These questions, in turn, can be answered by a polynomial time algorithm. A careful analysis of this algorithm shows that it can be expressed in fixpoint logic.[1] Unfortunately, it does not seem possible to express this algorithm in Datalog($\neq$). We get around this difficulty by offering a reduction of fixed subgraph homeomorphism queries over acyclic input graphs to questions about winning strategies in certain *two-player* pebble games on the input graph. Unlike the single-player games, we show that questions about winning strategies of the two-player games can be answered by Datalog($\neq$) programs.

Finally, on the negative side we demonstrate that if the pattern graph H is in the complement $\bar{C}$ of C, then the fixed subgraph homeomorphism query is *not* expressible in L^{ω} and, a fortiori, is not expressible in Datalog($\neq$) as well. The proof is an interesting illustration of how the game-theoretic characterization of L^{ω} can be used to translate a complexity lower-bound proof to an expressibility lower-bound proof.

[1] It is perhaps worth pointing out that the proof of this fact makes a crucial use of the *stage comparison* theorem from [Mos74]. This is a nontrivial result in fixpoint logic, which holds for all structures (finite or infinite) and which has played an important role in the study of fixpoint logic on finite structures [Imm86, GS86].

One of the main difficulties in game-theoretic proofs of lower bounds is finding the right structures on which to play the game. The idea underlying our proof is to play the game on structures generated by the reduction in NP-hardness result of [FHW80].

2. DATALOG($\neq$)

It is well known (cf. [Au79, Gai82]) that many natural graph-theoretic queries, such as transitive closure, connectivity, acyclicity, and two-colorability, are not expressible in relational calculus, or, equivalently, they are not first-order definable. The above queries, however, are expressible in *fixpoint logic*, which is first-order logic augmented with the *least fixpoint operator* for *positive* formulas [AU79, CH82].

We now consider Datalog($\neq$), which in terms of expressive power is equivalent to a fragment of fixpoint logic. A Datalog($\neq$) program is a finite collection of rules of the form

$$t_0 \leftarrow t_1, t_2, \ldots, t_l.$$

The expression t_0 is the *head* of the rule, while $t_1, \ldots, t_l$ form the *body of the rule*. The head of the rule is an atomic formula $S(x_1, \ldots, x_n)$, where S is a relational symbol. The expressions in the body can be equalities $x_j = x_k$, inequalities $x_j \neq x_k$, or atomic formulas $R(x_1, \ldots, x_m)$. Note that negated atomic formulas are not allowed in a Datalog($\neq$) program. The relational symbols appearing in the heads of the rules form the *intensional database predicates* (IDBs), while the others are the *extensional database predicates* (EDBs). One of the IDB predicates is designated as the *goal* predicate of the program.

A Datalog program is a Datalog($\neq$) program such that the body of every rule consists entirely of atomic formulas; i.e., no equalities or inequalities are allowed.

Both Datalog and Datalog($\neq$) have uniform semantics that are usually defined in terms of least fixpoints of monotone operators. For simplicity, assume first that π is a Datalog($\neq$) program with a single IDB predicate S of arity r and let σ be the vocabulary of the EDB predicates of π. On every structure $\mathbf{A}$ over the vocabulary σ the program π gives rise to an operator $\Theta_{\mathbf{A}}$ between r-ary relations on the universe A of $\mathbf{A}$. If S is an r-ary relation on $\mathbf{A}$, then the value $\Theta_{\mathbf{A}}(S)$ of the operator $\Theta_{\mathbf{A}}$ on S is the relation obtained by applying all rules of π, where the IDB predicate of π is being interpreted by S and the EDB predicates of π are being interpreted by the relations of the structure $\mathbf{A}$. Since the IDB predicate of π occurs positively in the bodies of the rules of π, we have that the operator $\Theta_{\mathbf{A}}$ is *monotone*; i.e., if $S \subseteq S'$, then $\Theta_{\mathbf{A}}(S) \subseteq \Theta_{\mathbf{A}}(S')$. The semantics π^∞ of the Datalog($\neq$) program π on $\mathbf{A}$ is defined to be the *least fixpoint* $\Theta_{\mathbf{A}}^\infty$ of the operator $\Theta_{\mathbf{A}}$ on $\mathbf{A}$. This is guaranteed to exist, because $\Theta_{\mathbf{A}}$ is a monotone operator.

Least fixpoint semantics are generally viewed as *declarative* semantics. In the case of Datalog($\neq$), the above semantics has also a *procedural* counterpart that is obtained by iterating the operator $\Theta_{\mathbf{A}}$ until its least fixpoint is reached. Given a structure $\mathbf{A}$, the IDB predicate S is initialized to be empty. Then the rules of π are applied repeatedly and tuples are added to the IDB predicate until this iteration stabilizes, i.e., until no new tuples can be added to S. More formally, the *stages* $\Theta_{\mathbf{A}}^n$, $n \geq 1$, of the operator $\Theta_{\mathbf{A}}$ are defined by the induction

$$\Theta_{\mathbf{A}}^1 = \Theta_{\mathbf{A}}(\varnothing) \qquad \text{and} \qquad \Theta_{\mathbf{A}}^{n+1} = \Theta_{\mathbf{A}}(\Theta_{\mathbf{A}}^n).$$

It can now be shown (cf. [CH85]) that the least fixpoint of $\Theta_{\mathbf{A}}$ is equal to the union of its stages, i.e.,

$$\Theta_{\mathbf{A}}^\infty = \bigcup_{n=1}^{\infty} \Theta_{\mathbf{A}}^n.$$

Actually, if $\mathbf{A}$ is a finite structure with s elements, then for some $n_0 \leq s^r$ we have that

$$\Theta_{\mathbf{A}}^\infty = \Theta_{\mathbf{A}}^{n_0} = \Theta_{\mathbf{A}}^n \qquad \text{for all} \quad n \geq n_0.$$

If the Datalog($\neq$) program π has more than one IDB predicate, then the semantics is defined by considering a system of monotone operators and iterating all operators of this system simultaneously until the least fixpoint of the system is reached. In this case, the semantics of π is the relation expressed by its goal predicate after the iteration terminates. In what follows, π^∞ will denote the semantics of a Datalog($\neq$) program π.

Chandra and Harel [CH85] were the first to point out that Datalog and its extensions are related to the existential fragment of fixpoint logic. More precisely, their results imply that Datalog($\neq$) is equivalent to the negation-free existential fragment of fixpoint logic, while Datalog is equivalent to the negation and inequality-free existential fragment of fixpoint logic.

We discuss now in more detail the connections between Datalog($\neq$) and fixpoint logic. For simplicity, we focus again on the case in which the program π has a single IDB predicate S. With every rule of π we can associate an existential first-order formula that has as matrix the body of the rule and existential quantifiers applied to the variables that occur only in the body (and not in the head) of the rule. Let $\varphi(\mathbf{w}, S)$ be the disjunction of all the existential first-order formulas obtained from the rules of π this way. Note that every relation symbol over the vocabulary $\sigma \cup \{S\}$ has only positive occurrences in $\varphi(\mathbf{w}, S)$. It is also easy to see that the formula $\varphi(\mathbf{w}, S)$ defines the operator $\Theta_{\mathbf{A}}$ uniformly on all structures $\mathbf{A}$. This means that the value $\Theta_{\mathbf{A}}(S)$ is given as

$$\Theta_{\mathbf{A}}(S) = \{\mathbf{a} : \mathbf{A}, \mathbf{a} \models \varphi(\mathbf{w}, S)\},$$

for every structure $\mathbf{A}$ and every r-ary relation S on A. It follows that the semantics of π coincides with the least fixpoint of the formula $\varphi(\mathbf{w}, S)$. This shows that Datalog($\neq$) is contained in the negation-free existential fragment of fixpoint logic. Moreover, the converse also holds; namely, least fixpoints of negation-free existential first-order formulas can be obtained from Datalog($\neq$) programs (cf. [CH85]).

EXAMPLE 2.1. For a concrete example, let π_1 be the Datalog($\neq$) program

$$T(x, y, w) \leftarrow E(x, y),\ w \neq x,\ w \neq y$$
$$T(x, y, w) \leftarrow E(x, z),\ T(z, y, w),\ w \neq x.$$

The operator $\Theta_{\mathbf{A}}$ associated with π_1 is defined by the negation-free existential first-order formula $\varphi(x, y, w, T)$, where

$$\varphi(x, y, w, T) \equiv (E(x, y) \wedge w \neq x \wedge w \neq y) \vee (\exists z)(E(x, z) \wedge T(z, y, w) \wedge w \neq x).$$

On any graph $\mathbf{A} = (V, E)$ the least fixpoint π_1^∞ of $\Theta_{\mathbf{A}}$ expresses the query "is there a *w-avoiding* path from x to y?" (i.e., "is there a path from x to y to that does not go through w?").

If π is a Datalog program, then the formula $\varphi(\mathbf{w}, S)$ that defines the operator $\Theta_{\mathbf{A}}$ is actually a negation-free and inequality-free existential first-order formula.

EXAMPLE 2.2. Let π_2 be the Datalog program

$$S(x, y) \leftarrow E(x, y)$$
$$S(x, y) \leftarrow E(x, z), S(z, y),$$

which computes the transitive closure query TC. The operator $\Theta_{\mathbf{A}}$ associated with π_2 is definable by the formula

$$\varphi(x, y, S) \equiv E(x, y) \vee (\exists z)(E(x, z) \wedge S(z, y)),$$

which is, indeed, both an inequality-free and negation-free existential first-order formula.

As mentioned in the Introduction, Datalog($\neq$) programs compute monotone queries, while Datalog programs compute queries that are strongly monotone. This difference stems from the lack of inequalities in Datalog programs. To illustrate this point, note that the query "is there a path from x to y that does not go through w?" in Example 2.1 is *not* expressible in Datalog, because this query need not be preserved when elements of the database universe are identified.

3. INFINITARY LOGICS WITH A FIXED NUMBER OF VARIABLES

The limited expressive power of first-order logic is due to its finitary syntax and to the absence of any recursion mechanism. Higher expressive power can be achieved by augmenting the syntax of first-order logic with infinitary formation rules or with fixpoint operators that act as iteration constructs. In this section we consider certain infinitary logics and investigate their relation to Datalog($\neq$).

During the 1960s and the 1970s logicians investigated in depth (cf. [BF85]) the infinitary logic $L_{\infty\omega}$, which, in addition to the rules of first-order logic, allows for disjunctions $\bigvee \Phi$ and conjunctions $\bigwedge \Phi$ over any (possibly infinite) set of formulas. More formally, the syntax of the infinitary logic $L_{\infty\omega}$ is defined as follows.

DEFINITION 3.1. Let σ be a vocabulary consisting of relational and constant symbols and let $\{v_1, ..., v_n, ...\}$ be a countable set of variables. The class $L_{\infty\omega}$ of *infinitary formulas* over σ is the smallest collection of formulas such that

- it contains all first-order formulas over σ.
- if φ is a formula of $L_{\infty\omega}$, then so is $\neg\varphi$,
- if ψ is a formula of $L_{\infty\omega}$ and v_i is a variable, then $(\forall v_i)\varphi$ and $(\exists v_i)\varphi$ are also formulas of $L_{\infty\omega}$.
- if Ψ is a set of $L_{\infty\omega}$ formulas, then $\bigvee \Psi$ and $\bigwedge \Psi$ are also formulas of $L_{\infty\omega}$.

Note that in the context of infinitary logic it is quite meaningful to consider infinite vocabularies, since a sentence can refer to infinitely many relations and constants. Since, however, our interest is focused here on finite structures, we adopt the following proviso.

Proviso. All vocabularies under consideration are assumed to be finite.

The concept of a *free variable* in a formula of $L_{\infty\omega}$ is defined in the same way as for first-order logic. We use the notation $\varphi(u_1, ..., u_m, ...)$ to denote that φ is a formula of $L_{\infty\omega}$ whose free variables are among the variables $u_1, ..., u_m, ...$ A *sentence* of $L_{\infty\omega}$ is a formula φ of $L_{\infty\omega}$ with no free variables. The semantics of $L_{\infty\omega}$ is a direct extension of the semantics of first-order logic, with $\bigvee \Psi$ interpreted as a disjunction over all formulas in Ψ and $\bigwedge \Psi$ interpreted as a conjunction. If $\mathbf{A}$ is a structure over σ and $a_1, ..., a_m, ...$ is a sequence of elements from the universe of $\mathbf{A}$, then we write

$$\mathbf{A}, a_1, ..., a_m, ..., \models \varphi(u_1, ..., u_m, ...)$$

to denote that the structure $\mathbf{A}$ *satisfies* the formula φ of $L_{\infty\omega}$ when each variable u_i is interpreted by the element a_i from the universe of $\mathbf{A}$, $i \geq 1$.

To illustrate the gain in expressive power, consider the well-known fact that the property "there are an even num-

ber of elements" is not expressible by any first-order sentence on finite structures. Let ρ_n be a first-order sentence stating that "there are exactly n elements." Then the infinitary sentence $\bigvee_{n=1}^{\infty} \rho_{2n}$ asserts that "there are an even number of elements."

In general, infinitary formulas may have an infinite number of distinct free or bound variables. We will now focus attention on fragments of $L_{\infty\omega}$ in which the total number of distinct variables is required to be finite.

DEFINITION 3.2. Let k be a positive integer.

- The *infinitary logic with k variables*, denoted by $L^k_{\infty\omega}$, consists of all formulas of $L_{\infty\omega}$ with at most k distinct variables.
- The infinitary logic $L^{\omega}_{\infty\omega}$ consists of all formulas of $L_{\infty\omega}$ with a finite number of distinct variables. Thus,

$$L^{\omega}_{\infty\omega} = \bigcup_{k=1}^{\infty} L^k_{\infty\omega}.$$

- We write $L^k_{\omega\omega}$ for the collection of all first-order formulas with at most k distinct variables.

It should be pointed out that a variable may have an infinite number of occurrences in a formula of $L^{\omega}_{\infty\omega}$.

The family $L^{\omega}_{\infty\omega}$ of the infinitary languages $L^k_{\infty\omega}$, $k \geq 1$, was introduced first by Barwise [Bar77], as a tool for studying fixpoint logic on infinite structures. It turned out, however, that these languages have, in addition, interesting uses in theoretical computer science. Indeed, they underlie much of the work in [Imm82, dR87, LM89] and they have been also studied in their own right in [Kol85, KV90]. Intuitively, a formula φ of $L^k_{\infty\omega}$ corresponds to a relational-algebra expression e_φ (cf. [Ull89]) with infinitary unions and intersections, such that all subexpressions of e_φ have arity of at most k.

The following examples illustrate the expressive power of $L^{\omega}_{\infty\omega}$.

EXAMPLE 3.3. *Cardinalities of total orders.* Assume that the vocabulary σ consists of a binary relation symbol $<$ and we are considering only the structures in which the interpretation of $<$ is a total order. Let τ_n be a first-order sentence asserting that "there are at least n elements." In general, τ_n requires n distinct variables. Immerman and Kozen [IK89] pointed out, however, that on total orders τ_n is equivalent to a sentence in $L^2_{\omega\omega}$. For example, τ_4 can be written as

$$(\exists x)(\exists y)(x < y \wedge (\exists x)(y < x \wedge (\exists y)(x < y))).$$

It follows that on total orders the sentence ρ_n asserting that "there are exactly n elements" is also in $L^2_{\omega\omega}$, since it is equivalent to $\tau_n \wedge \neg\tau_{n+1}$. As a result, on total orders properties such as "there are an even number number of elements," "the universe is finite," etc. are expressible in $L^2_{\infty\omega}$. In general, if P is any set of positive integers, then the property "the cardinality of the total order is a member of P" is expressible in $L^2_{\infty\omega}$, since it is definable by

$$\bigvee_{n \in P} \rho_n.$$

This implies, in particular, that $L^2_{\infty\omega}$ can express nonrecursive queries on total orders.

EXAMPLE 3.4. *Paths and connectivity.* Assume that the vocabulary σ consists of a single binary relation E and let $p_n(x, y)$ be a first-order formula over σ asserting that there is a path of length n from x to y. The obvious way to write $p_n(x, y)$ requires $n+1$ distinct variables, namely

$$(\exists x_1)\cdots(\exists x_{n-1}) \times (E(x, x_1) \wedge E(x_1, x_2) \wedge \cdots \wedge E(x_{n-1}, y)).$$

It is known, however, that each $p_n(x, y)$ is equivalent to a formula in $L^3_{\omega\omega}$, i.e., a first-order formula with at most three distinct variables x, y, z (cf. [Imm82]). To see this, put

$$p_1(x, y) \equiv E(x, y)$$

and assume, by induction on n, that $p_{n-1}(x, y)$ is equivalent to a formula in $L^3_{\omega\omega}$. Then

$$p_n(x, y) \equiv (\exists z)[E(x, z) \wedge (\exists x)(x = z \wedge p_{n-1}(x, y))].$$

Thus, the transitive closure query TC is expressible in $L^3_{\infty\omega}$.

More generally, if P is *any* set of positive integers, then the property "x and y are connected by a path whose length is a number in P" is expressible in $L^3_{\infty\omega}$ via the formula:

$$\bigvee_{n \in P} p_n(x, y).$$

Typical interesting instances of this kind of query are: "x and y are connected by a path of even lengh," "x and y are connected by a path whose length is a perfect square," and so on. This also shows that $L^3_{\infty\omega}$ can express nonrecursive queries.

Note that in the preceding example 3.4 only existential quantifiers were used in the formulas. Moreover, no negated atomic formulas were used. These observations motivate the following definition.

DEFINITION 3.5. Let L^k be the collection of all formulas of $L^k_{\infty\omega}$, $k \geq 1$, that are obtained from atomic formulas, equalities, and inequalities using infinitary disjunctions,

infinitary conjunctions, and existential quantification only. We also put

$$L^{\omega} = \bigcup_{k=1}^{\infty} L^k$$

and call this logic the *existential negation-free fragment of* $L^{\omega}_{\infty\omega}$.

Our next result shows that L^{ω} is powerful enough to express every Datalog($\neq$) query.

THEOREM 3.6. *If π is a* Datalog($\neq$) *program, then π^{∞} is definable by a formula of L^{ω}. Moreover, if π is a Datalog program, then π^{∞} is definable by an inequality-free formula of L^{ω}.*

Proof. For simplicity, we present first in detail the proof for the case in which the Datalog($\neq$) program π has a single IDR predicate S of arity r.

Let σ be the vocabulary of the EDB predicates of π and consider the monotone operator $\Theta_{\mathbf{A}}$ whose least fixpoint on a structure $\mathbf{A}$ over σ is the semantics of π on $\mathbf{A}$. As explained in the previous section, there is an existential formula $\varphi(w_1, ..., w_r, S)$ over $\sigma \cup \{S\}$ that defines $\Theta_{\mathbf{A}}$ uniformly on all structures. Thus,

$$\Theta_{\mathbf{A}}(S) = \{(a_1, ..., a_r) \in A^r : \mathbf{A}, a_1, ..., a_r \models \varphi(w_1, ..., w_r, S)\}$$

for every structure $\mathbf{A}$ over σ and every r-ary relation S on the universe A of $\mathbf{A}$. Moreover, every predicate symbol from $\sigma \cup \{S\}$ has only positive occurrences in the formula $\varphi(w_1, ..., w_r, S)$.

Assume that the total number of distinct variables (both free and bound) occurring in $\varphi(w_1, ..., w_r, S)$ is equal to l. Using induction on n, we will show that every stage $\Theta^n_{\mathbf{A}}$, $n \geqslant 1$, of the operator $\Theta_{\mathbf{A}}$ is definable uniformly on all structures $\mathbf{A}$ over σ by an existential first-order formula $\varphi^n(w_1, ..., w_r)$ that is negation-free and has all its variables among those of φ together with r new variables $y_1, ..., y_r$ not occurring in φ. Thus, each $\varphi^n(w_1, ..., w_r)$ will turn out to be an existential negation-free first-order formula at most $l+r$ distinct variables.

The claim is obvious for the first stage $\Theta^1_{\mathbf{A}}$ of $\Theta_{\mathbf{A}}$, since

$$\Theta^1_{\mathbf{A}} = \Theta_{\mathbf{A}}(\varnothing) = \{(w_1, ..., w_r) : \mathbf{A} \models \varphi(w_1, ..., w_r, \varnothing)\}$$

and $\varphi(w_1, ..., w_r, S)$ is an existential negation-free formula over the vocabulary $\sigma \cup \{S\}$.

Assume that the induction hypothesis holds for $\Theta^n_{\mathbf{A}}$ and let $\varphi^n(w_1, ..., w_r)$ be an existential negation-free formula with at most $l+r$ variables that defines $\Theta^n_{\mathbf{A}}$ uniformly on all structures. From the definition of the stages and the properties of $\varphi(w_1, ..., w_r, S)$, we have that

$$\Theta^{n+1}_{\mathbf{A}} = \{(a_1, ..., a_r) : \mathbf{A}, a_1, ..., a_r \models \varphi(w_1, ..., w_r, \Theta^n_{\mathbf{A}})\}.$$

We would like to apply the induction hypothesis and substitute $\Theta^n_{\mathbf{A}}$ in the expression above by its defining formula $\varphi^n(w_1, ..., w_r)$ without increasing the total number of variables. To achieve this, replace in $\varphi(w_1, ..., w_r, S)$ every occurrence of a subformula of the form $S(t_1, ..., t_r)$ by the expression

$$(\exists y_1) \cdots (\exists y_r)[(y_1 = t_1) \wedge \cdots \wedge (y_r = t_r) \wedge (\exists w_1) \cdots (\exists w_r) \times ((w_1 = y_1) \wedge \cdots \wedge (w_r = y_r) \wedge \varphi^n(w_1, ..., w_r)].$$

The resulting expression yields a formula $\varphi^{n+1}(w_1, ..., w_r)$ of $L^{l+r}_{\omega\omega}$ that defines $\Theta^{n+1}_{\mathbf{A}}$ uniformly on all structures. Moreover, $\varphi^{n+1}(w_1, ..., w_r)$ is an existential negation-free first-order formula.

It now follows that the least fixpoint $\pi^{\infty} = \Theta^{\infty}_{\mathbf{A}}$ of the operator Θ is definable by a formula of L^{l+r}. Indeed, we have that

$$\Theta^{\infty}_{\mathbf{A}} = \bigcup_{n=1}^{\infty} \Theta^n_{\mathbf{A}} = \left\{(a_1, ..., a_r) : \mathbf{A}, a_1, ..., a_r \models \bigvee_{n=1}^{\infty} \varphi^n(w_1, ..., w_r)\right\}.$$

Note also that if π is a Datalog program, then $\bigvee_{n=1}^{\infty} \varphi^n(w_1, ..., w_r)$ is actually an inequality-free formula of L^{l+r}, since in this case each $\varphi^n(w_1, ..., w_r)$ turns out to be an existential negation-free and inequality-free first-order formula with at most $l+r$ variables.

The general case requires only minor modifications. Indeed, if the Datalog($\neq$) program π has more than one IDB predicate, then the proof proceeds by considering a system of monotone operators and showing by simultaneous induction that all stages of the system are definable by existential negation-free first-order formulas with at most $l+r$ variables. ∎

We have, thus, established that Datalog($\neq$) is contained in L^{ω}. We should point out that this containment is a proper one, since L^{ω} can express nonrecursive queries, while Datalog($\neq$) queries are always computable in polynomial time. Note also that there are queries of very low complexity that are not expressible in L^{ω}. For example, the *parity query* (i.e., "are there an even number of elements in the universe of the structure?") is not expressible in $L^{\omega}_{\infty\omega}$ and, consequently, is also not expressible in L^{ω} (cf. [KV90]). Thus, in general, there is no connection between the expressibility of a query in L^{ω} and its computational complexity.

The preceding Theorem 3.6 constitutes a refinement of an earlier result to the effect that on every fixed structure the infinitary logic $L^{\omega}_{\infty\omega}$ can express every fixpoint query. That result appeared first in print in Barwise [Bar77] and Immerman [Imm82], but is actually anticipated in the unpublished Ph.D. thesis of Rubin [Rub75].

4. EXPRESSIVENESS AND PEBBLE GAMES

The results of the previous section imply that, in order to show that a particular query Q is not expressible in Datalog($\neq$), it is enough to establish that Q is not definable in L^ω. To characterize expressibility in L^ω, we define a certain relation between structures.

DEFINITION 4.1. Let k be a positive integer and let $\mathbf{A}$ and $\mathbf{B}$ be two structures over the vocabulary σ. Assume also that $a_1, ..., a_m$ and $b_1, ..., b_m$ are finite sequences of distinct elements from the universes of $\mathbf{A}$ and $\mathbf{B}$, respectively, where $1 \leqslant m \leqslant k$. We write

$$(\mathbf{A}, a_1, ..., a_m) \preccurlyeq^k (\mathbf{B}, b_1, ..., b_m)$$

to denote that for every formula $\varphi(u_1, ..., u_m)$ of L^k with free variables among $u_1, ..., u_m$ if $\mathbf{A}, a_1, ..., a_m \models \varphi(u_1, ..., u_m)$, then $\mathbf{B}, b_1, ..., b_m \models \varphi(u_1, ..., u_m)$.

In particular, we write $\mathbf{A} \preccurlyeq^k \mathbf{B}$ to denote that every sentence of L^k that is true in $\mathbf{A}$ is also true in $\mathbf{B}$.

It is obvious that the relation $\preccurlyeq^k$ is reflexive and transitive. It will turn out, through, that it is not symmetric. The connection between expressibility in L^k and the relation $\preccurlyeq^k$ is described by the following proposition.

PROPOSITION 4.2. *Let $\mathscr{C}$ be a class of finite structures over the vocabulary σ and let k be a positive integer. Then the following statements are equivalent:*

1. *The class $\mathscr{C}$ is L^k-definable; i.e., there is a sentence φ of L^k such that for any finite structure $\mathbf{A}$ over σ we have that*

$$\mathbf{A} \in \mathscr{C} \Leftrightarrow \mathbf{A} \models \varphi.$$

2. *If $\mathbf{A}$ and $\mathbf{B}$ are finite structures over σ such that $\mathbf{A} \in \mathscr{C}$ and $\mathbf{A} \preccurlyeq^k \mathbf{B}$, then $\mathbf{B} \in \mathscr{C}$.*

Proof. The direction (1) $\Rightarrow$ (2) follows from the definition of $\preccurlyeq^k$. For the other direction, assume that statement (2) holds for the class $\mathscr{C}$. Let $\mathbf{A}_0, \mathbf{A}_1, ...$ be an enumeration of all finite structures over σ up to isomorphism (there are countably many isomorphism classes of finite structures over σ).

Let $i \geqslant 0$. If $j \neq i$ and $\mathbf{A}_i \not\preccurlyeq^k \mathbf{A}_j$, then there is a sentence φ_j of L^k such that φ_j holds in $\mathbf{A}_i$ and fails in $\mathbf{A}_j$. Thus, the conjunction $\bigwedge_{\mathbf{A}_i \not\preccurlyeq^k \mathbf{A}_j} \varphi_j$ holds for $\mathbf{A}_i$, but fails on all structures $\mathbf{A}$ such that $\mathbf{A}_i \not\preccurlyeq^k \mathbf{A}$. We denote this conjunction, which is a sentence of L^k, by Φ_i.

We now claim that the countable disjunction $\bigvee_{\mathbf{A}_i \in \mathscr{C}} \Phi_i$, which is a sentence of L^k, defines the class $\mathscr{C}$. Assume that $\mathbf{A} \in \mathscr{C}$. Then $\mathbf{A}$ is isomorphic to $\mathbf{A}_l$ for some $l \geqslant 0$. Thus, $\mathbf{A} \models \bigvee_{\mathbf{A}_i \in \mathscr{C}} \Phi_i$, since $\mathbf{A} \models \Phi_l$. Conversely, assume that $\mathbf{A}$ is a finite structure satisfying Φ_l, where $\mathbf{A}_l \in \mathscr{C}$. Since Φ_l fails on all structures $\mathbf{A}_j$ such that $\mathbf{A}_l \not\preccurlyeq^k \mathbf{A}_j$, it follows that $\mathbf{A}_l \preccurlyeq^k \mathbf{A}$. Consequently, $\mathbf{A} \in \mathscr{C}$.

The relation $\preccurlyeq^k$ can be characterized in terms of certain infinitary pebble games. The games we consider here are asymmetric versions of the *k-pebble games* introduced by Barwise [Bar77] and Immerman [Imm82] in the study of the infinitary logics $L^k_{\infty\omega}$, $k \geqslant 1$.

DEFINITION 4.3. Let σ be a vocabulary consisting of relational symbols and constant symbols. Let $\mathbf{A}$, $\mathbf{B}$ be two structures over σ and let $c_1, ..., c_l$ and $d_1, ..., d_l$ be the intepretations of the constant symbols of σ on $\mathbf{A}$ and $\mathbf{B}$, respectively.

The *existential k-pebble game* between Players I and II on the structures $\mathbf{A}$ and $\mathbf{B}$ is played as follows. Each player has k pebbles: Player I has pebbles $p_1, ..., p_k$ and Player II has pebbles $q_1, ..., q_k$. A position in the game is a placement of some of the pebbles on the elements of $\mathbf{A}$ and $\mathbf{B}$; the p_i's are placed on elements of $\mathbf{A}$ and the q_i's are placed on elements of $\mathbf{B}$. A pebble q_i will be placed on $\mathbf{B}$ immediately after the pebble p_i is placed on $\mathbf{A}$. Initially, no pebble is placed. In each round of the game, Player I picks up some pebble, say p_i. If p_i is placed on $\mathbf{A}$, then Player I removes p_i from $\mathbf{A}$, and Player II responds by removing the pebble q_i from $\mathbf{B}$. If p_i is not placed on an element of $\mathbf{A}$, then Player I places p_i on some element of $\mathbf{A}$, and Player II responds by placing q_i on some element of $\mathbf{B}$.

Let $i_1, ..., i_m$ be the indices of the pebbles that are placed on $\mathbf{A}$ (and $\mathbf{B}$) after the ith round. Let $a_{i_1}, ..., a_{i_m}$ ($b_{i_1}, ..., b_{i_m}$) be the elements of $\mathbf{A}$ ($\mathbf{B}$) pebbled by the pebbles $p_{i_1}, ..., p_{i_m}$ ($q_{i_1}, ..., q_{i_m}$) after the ith round. If the mapping h with

$$h(a_{i_j}) = b_{i_j}, \qquad 1 \leqslant j \leqslant m,$$

and

$$h(c_j) = d_j, \qquad 1 \leqslant j \leqslant l,$$

is *not* a one-to-one homomorphism[0] between the substructures of $\mathbf{A}$ and $\mathbf{B}$ with universes

$$\{a_{i_1}, ..., a_{i_k}\} \cup \{c_1, ..., c_l\}$$

and

$$\{b_{i_1}, ..., b_{i_k}\} \cup \{d_1, ..., d_l\},$$

respectively, then Player I wins the game. Otherwise, the game continues. Player II wins the game if he has a *winning strategy* that allows him to continue playing "forever," i.e., if Player I can never win a round of the game.

[2] A *one-to-one homomorphism* from a structure $\mathbf{A}$ into a structure $\mathbf{B}$ is a one-to-one mapping h from the domain of $\mathbf{A}$ into the domain of $\mathbf{B}$ such that the constants of $\mathbf{A}$ are mapped to the corresponding constants of $\mathbf{B}$ and if $(w_1, ..., w_n)$ is a tuple in a relation R of $\mathbf{A}$, then $(h(w_1), ..., h(w_n))$ is a tuple in the relation R of $\mathbf{B}$.

We give next two examples that illustrate existential k-pebble games and reveal that the relation "Player II wins the existential k-pebble game on **A** and **B**" is not a symmetric relation on pairs of structures.

EXAMPLE 4.4. *Paths of different lengths.* Assume that the vocabulary σ consists of a single binary relation symbol E. Let **A** be a graph that is a directed path with m vertices and let **B** be a directed path with n vertices, where $n > m \geq 2$. Assume that **A** has vertices $\{a_1, ..., a_m\}$ and edges (a_i, a_{i+1}), $1 \leq i < m$, while **B** has vertices $\{b_1, ..., b_n\}$ and edges (b_j, b_{j+1}), $1 \leq j < n$.

It is quite clear that Player II wins the existential k-pebble game on **A** and **B** for any $k \leq 1$. His strategy is to simply "copy" on **B** the moves of Player I on **A**. More precisely, whenever Player I places a pebble on a node a_i of **A**, Player II responds by placing a pebble on the node b_i of **B**.

In contrast, we claim that Player I can win the existential two-pebble game on **B** and **A** (and, hence, Player I can also win the existential k-pebble game on **B** and **A** for any $k \geq 2$). Player I begins by placing his two pebbles on the nodes b_1, b_2 of **B**. Player II has to respond by placing his two pebbles on two consecutive nodes of **A**; actually, his best option is to place the two pebbles on a_1, a_2. Player I now removes the pebble from b_1 and places it on b_3. Player II's only option is to remove the pebble from a_1 and place it on a_3. Player I can continue playing this way, moving along the path on **B** and forcing Player II to move along the path on **A**. After a while, Player II has two pebbles on b_{m-1}, b_m and Player I has his corresponding pebbles on a_{m-1}, a_m. Since $n > m$, Player I can win the game in his next move by removing the pebble from b_{m-1} and placing it on b_{m+1}. At this point, Player II can no longer maintain a one-to-one homomorphism.

EXAMPLE 4.5. *Disjoint paths.* Assume that the vocabulary σ consists of a single binary relation E. Let **A** be a structure that is made up of two disjoint directed paths each with $2n+1$ vertices for some $n \geq 1$ and let **B** be a structure that is made up of two directed paths each with $2n+1$ vertices intersecting only at their $(n+1)$th vertex. Assume that the two paths in **A** have $a_1, ..., a_{2n+1}$ and $a'_1, ..., a'_{2n+1}$ as vertices, while the two paths in **B** have $b_1, ..., b_n, b_{n+1}, ..., b_{2n+1}$ and $b'_1, ..., b'_n, b'_{n+1}, ..., b'_{2n+1}$ as vertices, with $b_{n+1} = b'_{n+1}$.

We now claim that Player I can win the existential three-pebble game on **A** and **B**. His strategy is to first move two pebbles along consecutive vertices on the path of the a_i's on **A** in such a way that Player II is forced to move two pebbles along the corresponding vertices on the path of the b_i's (see Example 4.4). This is done until there are pebbles on a_n, a_{n+1} and on b_n, b_{n+1}. After this, Player I keeps one pebble on a_{n+1} and starts moving two pebbles along consecutive vertices on the path of the a'_i's, while forcing Player II to place pebbles along the corresponding vertices on the path of the b'_i's. Eventually, Player II loses because he is forced to remove a pebble from b'_{n+1} and place it on b_{n+1}, at which point the function between the pebbled vertices is not one-to-one.

In the above definition, we have described in a rather informal way what it means to say that "Player II wins the existential k-pebble game." The concept of a *winning strategy* for Player II in the existential k-pebble game is being made precise in what follows.

DEFINITION 4.6. Assume that **A** and **B** are two structures over the vocabulary σ and let c_j, $1 \leq j \leq l$, and d_j, $1 \leq j \leq l$, be the interpretations of the constant symbols of σ on **A** and **B**, respectively.

A *partial one-to-one homomorphism* between **A** and **B** is a function h such that its domain is a finite subset of the universe of **A** containing the constants $c_1, ..., c_l$ of **A**, its range is a finite subset of the universe of **B** containing the constants $d_1, ..., d_l$ of **B**, $h(c_j) = d_j$, $1 \leq j \leq l$, and such that h is a one-to-one homomorphism between the substructures of **A** and **B** with universes the domain and range of h respectively.

DEFINITION 4.7. Let k be a positive integer and let **A** and **B** be two structures over the vocabulary σ. We say that *Player II has a winning strategy for the existential k-pebble game* on **A** and **B** if there is a nonempty family $\mathcal{H}$ of partial one-to-one homomorphisms between **A** and **B** such that:

- $\mathcal{H}$ is *closed under subfunctions*: if $f \in \mathcal{H}$ and $\{(c_1, d_1), ..., (c_l, d_l)\} \subseteq g \subseteq f$ (as sets of ordered pairs), then $g \in \mathcal{H}$.
- $\mathcal{H}$ has *the forth property up to k*: if $f \in \mathcal{H}$ and $|f| < k + l$, then for any element $a \in \mathbf{A}$ there is an element b in **B** such that the function $f \cup \{(a, b)\}$ is in $\mathcal{H}$.

It turns out that $\preccurlyeq^k$ can be characterized in terms of the existential k-pebble game according to the following crucial result.

THEOREM 4.8. *Let k be a positive integer and let* **A**, **B** *be two structures over the vocabulary* σ. *Then the following two statements are equivalent*:

1. $\mathbf{A} \preccurlyeq^k \mathbf{B}$.
2. *Player* II *has a winning strategy for the existential k-pebble game on* **A** *and* **B**.

Proof. Let c_1, c_l and $d_1, ..., d_l$ be the interpretations of the constant symbos of σ and **A** and **B**, respectively. Assume first that $\mathbf{A} \preccurlyeq^k \mathbf{B}$. We have to show that there is a family $\mathcal{H}$ of partial one-to-one homomorphisms between **A** and **B** that provides Player II with a winning strategy for the existential k-pebble game.

The desired family $\mathcal{H}$ consists of all partial one-to-one

homomorphisms h between $\mathbf{A}$ and $\mathbf{B}$ such that the following hold:

- The domain of h is a set of the form $\{c_1, ..., c_l, a_1, ..., a_m\}$ and the range of h is a set of the form $\{d_1, ..., d_l, b_1, ..., b_m\}$, where $m \leqslant k$;
- $h(c_j) = d_j$, for all $j \leqslant l$, and $h(a_i) = b_i$, for all $i \leqslant m$;
- $(\mathbf{A}, a_1, ..., a_m) \leqslant^k (\mathbf{B}, b_1, ..., b_m)$.

We show now that $\mathscr{H}$ has the required properties:

1. $\mathscr{H}$ is non-empty, because $\mathbf{A} \leqslant^k \mathbf{B}$ and, thus, the function h with $h(c_j) = d_j$, $1 \leqslant j \leqslant l$, is a member of H.

2. It is clear from the definitions that $\mathscr{H}$ is closed under subfunctions.

3. It remains to show that $\mathscr{H}$ has the forth property up to k. Assume that $h \in \mathscr{H}$ and $|h| = m + l < k + l$. Then there are sequences of distinct elements $a_1, ..., a_m$ in $\mathbf{A}$ and $b_1, ..., b_m$ in $\mathbf{B}$, such that $h(a_i) = b_i$, $1 \leqslant i \leqslant m$, and

$$(\mathbf{A}, a_1, ..., a_m) \leqslant^k (\mathbf{B}, b_1, ..., b_m).$$

We claim that for any element a in A that is different from $a_1, ..., a_m$ there is an element b in B such that b is different from $b_1, ..., b_m$ and

$$(\mathbf{A}, a_1, ..., a_m, a) \leqslant^k (\mathbf{B}, b_1, ..., b_m, b).$$

Assume that no such $b \in B$ exists for a certain $a \in A$. Then for every $b \in B$ that is different from $b_1, ..., b_m$ there is a formula $\psi_b(v_1, ..., v_m, v)$ of L^k over σ such that

$$(\mathbf{A}, a_1, ..., a_m, a) \models \psi_b(v_1, ..., v_m, v)$$

and

$$(\mathbf{B}, b_1, ..., b_m, b) \not\models \psi_b(v_1, ..., v_m, v).$$

Hence,

$$(\mathbf{A}, a_1, ..., a_m) \models (\exists v)\Big((v_1 \neq v) \wedge \cdots \wedge (v_m \neq v) \wedge \bigwedge_{b \in B} \psi_b(v_1, ..., v_m, v)\Big),$$

and, at the same time,

$$(\mathbf{B}, b_1, ..., b_m) \not\models (\exists v)\Big((v_1 \neq v) \wedge \cdots \wedge (v_m \neq v) \wedge \bigwedge_{b \in B} \psi_b(v_1, ..., v_m, v)\Big).$$

This, however, is a contradiction, since

$$(\exists v)\Big((v_1 \neq v) \wedge \cdots \wedge (v_m \neq v) \wedge \bigwedge_{b \in B} \psi_b(v_1, ..., v_m, v)\Big)$$

is an L^k sentence and $(\mathbf{A}, a_1, ..., a_m) \leqslant^k (\mathbf{B}, b_1, ..., b_m)$. Assume now that Player II has a winning strategy for the existential k-pebble game on $\mathbf{A}$ and $\mathbf{B}$. Let $\mathscr{H}$ be a family of partial one-to-one homomorphisms providing Player II with a winning strategy for this game. We will show, by induction on the construction of L^k formulas, that if $\psi(v_1, ..., v_m)$ is a formula of L^k whose variables are among $v_1, ..., v_k$ and whose free variables are among $v_1, ..., v_m$, where $m \leqslant k$, then the following property $(*)$ holds:

$(*)$ For all $h \in \mathscr{H}$ with $|h| \geqslant l + m$ and for any elements (not necessarily distinct) $a_1, ..., a_m$ from the domain of h, we have that if

$$\mathbf{A}, a_1, ..., a_m \models \psi(v_1, ..., v_m),$$

then

$$\mathbf{B}, h(a_1), ..., h(a_m) \models \psi(v_1, ..., v_m).$$

Once property $(*)$ is established, then we will be able to conclude that $\mathbf{A} \leqslant^k \mathbf{B}$ by applying this property to sentences of L^k.

The base case in the induction is obvious, since atomic formulas and inequalities are preserved under one-to-one homomorphisms. The inductive steps for infinitary disjunction $\bigvee$ and infinitary conjunction $\bigwedge$ are straightforward using the induction hypothesis.

Assume that the formula $\psi(v_1, ..., v_m)$ is of the form $(\exists v)$ $\chi(v_1, ..., v_m, v)$ and that the property $(*)$ holds for the formula $\chi(v_1, ..., v_m, v)$. Let h be a one-to-one homomorphism in $\mathscr{H}$ such that $|h| \geqslant l + m$. We have to show that if $a_1, ..., a_m$ are arbitrary elements (not necessarily distinct) from the domain of h such that

$$\mathbf{A}, a_1, ..., a_m \models (\exists v)\, \chi(v_1, ..., v_m, v),$$

then

$$\mathbf{B}, h(a_1), ..., h(a_m) \models (\exists v)\, \chi(v_1, ..., v_m, v).$$

Note that, by our assumption about the variables of ψ, we must have that v is a variable v_j, for some j such that $1 \leqslant j \leqslant k$. We now distinguish two cases, namely the case where $j > m$ and the case where $j \leqslant m$.

If $j > m$, then it must also be the case that $m < k$. Let $a \in A$ be such that

$$\mathbf{A}, a_1, ..., a_m, a \models \chi(v_1, ..., v_m, v).$$

Consider the subfunction h^* of h with domain the set $\{c_1, ..., c_l, a_1, ..., a_m\}$. Note that h^* is a member of $\mathscr{H}$, since $\mathscr{H}$ is closed under subfunctions. By the forth property of $\mathscr{H}$ applied to h^* and a, there is an element $b \in B$ such that $h^* \cup \{(a, b)\}$ is in $\mathscr{H}$. By applying the induction hypothesis to $\chi(v_1, ..., v_m, v)$ and to $h^* \cup \{(a, b)\}$, we infer that

$$\mathbf{B}, h(a_1), ..., h(a_m), b \models \chi(v_1, ..., v_m, v)$$

and, hence,

$$\mathbf{B}, h(a_1), ..., h(a_m) \models (\exists v)\, \chi(v_1, ..., v_m, v).$$

Finally, assume that $j \leqslant m$. In this case v is the variable v_i, for some i with $1 \leqslant i \leqslant m$. Note also that the free variables of the formula χ are among the variables $v_1, ..., v_m$ and that

$$\mathbf{A}, a_1, ..., a_{i-1}, a_{i+1}, ..., a_m \models (\exists v_i)\, \chi(v_1, ..., v_m).$$

Let g be the subfunction of h with domain the set

$$\{c_1, ..., c_l, a_1, ..., a_{i-1}, a_{i+1}, ..., a_m\}.$$

Observe that $|g| \leqslant l + m - 1 < l + k$ and that g is a member of $\mathscr{H}$, since $\mathscr{H}$ is closed under subfunctions. Let $a \in \mathbf{A}$ be such that

$$\mathbf{A}, a_1, ..., a_{i-1}, a, a_{i+1}, ..., a_m \models \chi(v_1, ..., v_m).$$

By the forth property of $\mathscr{H}$ applied to g and a, there is an element $b \in B$ such that $g \cup \{(a, b)\}$ is in $\mathscr{H}$. By applying the induction hypothesis to $\chi(v_1, ..., v_m)$ and to $g \cup \{(a, b)\}$, we infer that

$$\mathbf{B}, g(a_1), ..., g(a_{i-1}), b, g(a_{i+1}), ..., g(a_m) \models \chi(v_1, ..., v_m)$$

and, hence,

$$\mathbf{B}, h(a_1), ..., h(a_m) \models (\exists v_i)\, \chi(v_1, ..., v_m),$$

since $g(a_j) = h(a_j)$ for $j \neq i$ and the satisfaction relation depends only on the free variables of a formula. ▮

The infinitary syntax of the logic L^k was used in a crucial way in the proof of the preceding Theorem 4.8. A careful examination of this proof reveals, however, that if the structure $\mathbf{B}$ is finite, then one can restrict attention to the first-order fragment of L^k. This is made precise in the following result.

COROLLARY 4.9. *Let k be positive integer, let $\mathbf{A}$ be a structure over the vocabulary σ and let $\mathbf{B}$ be a finite structure over σ. Then the following three statements are equivalent:*

1. $\mathbf{A} \preccurlyeq^k \mathbf{B}$.

2. *Every first-order sentence of the infinitary logic L^k that is true on $\mathbf{A}$ is also true on $\mathbf{B}$.*

3. *Player II has a winning strategy for the existential k-pebble game on $\mathbf{A}$ and $\mathbf{B}$.*

Proof. The argument used in the proof of Theorem 4.8 goes here through virtually unchanged. In particular, the implication (2) $\Rightarrow$ (3) can be established using the same argument as the one for the implication (1) $\Rightarrow$ (2) in Theorem 4.8. One need only observe that the conjunction over the universe B of the structure $\mathbf{B}$ is now a conjunction over a finite set. Thus, the resulting formula is actually a first-order formula of L^k. ▮

As a consequence of Proposition 4.2 and Theorem 4.8, we obtain next a game-theoretic characterization of definability in the logic L^k for classes of finite structures.

THEOREM 4.10. *Let k be a positive integer and let $\mathscr{C}$ be a class of finite structures over the vocabulary σ. Then the following two statements are equivalent:*

1. *The class $\mathscr{C}$ is L^k-definable, i.e., there is a sentence ψ of L^k such that for any finite structure $\mathbf{A}$ over σ we have that $\mathbf{A} \in \mathscr{C} \Leftrightarrow \mathbf{A} \models \psi$.*

2. *If $\mathbf{A}$ and $\mathbf{B}$ are finite structures over σ such that $\mathbf{A} \in \mathscr{C}$ and Player II has a winning strategy for the existential k-pebble game on $\mathbf{A}$ and $\mathbf{B}$, then $\mathbf{B} \in \mathscr{C}$.*

Remark 4.11. It follows from Corollary 4.9 that the sentences Φ_i in the proof of Proposition 4.2 can be taken to be countable conjuctions of first-order sentence of L^k. Thus, every sentence in L^k is equivalent to a countable disjunction of countable conjunctions of first-order sentences of L^k. This can be viewed as a *normal form* for sentences of L^ω on finite structures.

Theorem 4.10 provides a tool for showing that certain queries are not expressible in the existential negation-free fragment L^ω of $L^\omega_{\infty\omega}$ on finite structures and, a fortiori, not expressible in Datalog($\neq$) on finite structures. More specifically, in order to prove that a query Q is not expressible in L^ω on finite structures it suffices to show that for every $k \geqslant 1$ there are two structures $\mathbf{A}_k$ and $\mathbf{B}_k$ such that $\mathbf{A}_k$ satisfies Q, $\mathbf{B}_k$ does not satisfy Q, and Player II has a winning strategy for the existential k-pebble game on $\mathbf{A}_k$ and $\mathbf{B}_k$. Theorem 4.10 also guarantees that this method is *complete*. Namely, if Q is not expressible in L^ω, then such structures $\mathbf{A}_k$ and $\mathbf{B}_k$ must exist for every $k \geqslant 1$.

Remark 4.12. 1. One can derive refinements of Theorems 4.8, 4.9, and 4.10 that apply to Datalog. For this we have to consider a variant of the existential k-pebble game in which Player I wins if at some point of the game the corresponding substructures of $\mathbf{A}$ and $\mathbf{B}$ are not *homomorphic*. The result in this case is that Player II has a winning strategy in the modified existential k-pebble game

if and only if every inequality-free sentence of L^k that is true in $\mathbf{A}$ is also true in $\mathbf{B}$.

2. Lakshmanan and Mendelzon [LM89] used a variant of existential k-pebble games, which they called *inductive pebble games*, to show that on finite directed graphs the *even simple path* query, i.e., the query "is there a simple path of even length between two distinguished nodes s and t?" is not expressible in Datalog($\neq$). At first sight, the inductive pebble games appear to be different than the existential k-pebble games, because Player II has the option to play in an "asynchronous" manner and match the moves of Player I at some later stage. It is possible to show, however, that "asynchronicity" is not a genuinely new feature. In particular, one can prove that no advantage is gained by using one type of game over the other in establishing that a query is not expressible in L^ω and, a fortiori, in Datalog($\neq$).

5. L^ω-EXPRESSIBILITY AND COMPUTATIONAL COMPLEXITY

We have observed before that there is no connection between the expressibility of a query in L^ω and its computational complexity. In this section we show that for a certain class of monotone queries, which includes the "even simple path" query of [LM89], definability in L^ω implies that the query is in PTIME. This, in turn makes it possible to infer, assuming P $\neq$ NP, that the NP-complete queries in that class are not expressible in L^ω.

DEFINITION 5.1. A query Q is *pattern-based* if there is a polynomial-time function α such that

1. $\alpha(\mathbf{B})$ is a set of finite structures, called *pattern structures*, for every finite structure $\mathbf{B}$,
2. if $\mathbf{A}$ is in $\alpha(\mathbf{B})$, then $\mathbf{A}$ satisfies Q, and
3. $\mathbf{B}$ satisfies Q if and only if there is a pattern structure $\mathbf{A} \in \alpha(\mathbf{B})$ and a one-to-one homomorphism from $\mathbf{A}$ into $\mathbf{B}$.

The function α is called the *pattern generator of* Q.

Note that every polynomial-time query is pattern based; simply set $\alpha(\mathbf{B})$ to be $\{\mathbf{B}\}$ or $\varnothing$, depending on whether $\mathbf{B}$ satisfies Q or not.

EXAMPLE 5.2. 1. *Even simple path query.* Consider the even simple path query, which is known to be NP-complete [LM89]. It holds for a directed graph G if there is a simple path of even length between two distinguished points s and t. If G has n nodes, then let $\alpha(G)$ consist of all directed graphs $P = (V, E)$, where $V = \{1, ..., k\}$, $1 < k \leqslant n$, k is odd, and $E = \{(1, 2), ..., (k-1, k)\}$. Clearly, there is a simple path of even length between s and t in G if and only if there is a one-to-one homomorphism from some member of $\alpha(G)$ into G mapping 1 to s and k to t. Thus, the even simple path query is a pattern-based query.

2. *Fixed subgraph homeomorphism queries.* Given directed graph H and G and a one-to-one mapping m of the nodes of H into the nodes of G, we say that H is *homeomorphic* to a subgraph of G if there exists a mapping from the edges of H to pairwise node-disjoint simple paths in G such that the edge (s, t) is mapped to a simple path from $m(s)$ to $m(t)$. The *H-subgraph homeomorphism* query, for a fixed graph H, is to determine for an input graph G and a node mapping m whether H is homeomorphic to a subgraph of G. It is easy to see that this query is pattern-based.

We will show that if a query Q is both pattern-based and expressible in L^ω, then it is in PTIME. We first need a result about the complexity of games (see [IL90] for related results).

PROPOSITION 5.3. *For every k, there is a polynomial-time algorithm that determines who wins the existential k-pebble game on a given pair $\mathbf{A}$, $\mathbf{B}$ of finite structures.*

Proof. The proof is based on three observations. The first observation is that there are at most $(n+1)^{2k}$ configurations in the game, where n is the maximum number of elements in either $\mathbf{A}$ or $\mathbf{B}$, since each pebble can be placed in at most $n+1$ different ways, i.e., placed on one of the n elements or not placed at all, and there are at most k pebbles on each structure. It follows that Player I wins the game if and only if he wins the game in a polynomial number of moves. More precisely, let $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c, m)$ denote the assertion that Player I wins the existential k-pebble game on the pair $\mathbf{A}$, $\mathbf{B}$ starting in a configuration c in m moves. Then, Player I wins the game if and only if $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c_0, (n+1)^{2k})$, where c_0 is the starting configuration of the existential k-pebble game (i.e., the configuration with no pebble placed).

The second observation is that the question whether $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c, m)$ holds is polynomially reducible to the question whether $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c'', m-1)$ holds for certain configurations c''. To test whether $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c, m)$ holds, we simply need to find a configuration c' that is reachable from c by a move of Player I such that $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c'', m-1)$ holds for all configurations c'' that are reachable from c' by a move of Player II. There are at most kn configurations c' that are reachable from c by a move of Player I, and for each c' there are at most n configurations c'' that are reachable from c' by a move of Player II. It follows that we can determine in polynomial time whether $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c_0, (n+1)^{2k})$ holds by evaluating whether $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c, i)$ holds for $i = 1, ..., (n+1)^{2k}$ for all configurations c, of which there are at most $(n+1)^{2k}$.

The third observation is that the game is *determined*, i.e., either Player I or Player II wins the game. This follows from a simple application of König's lemma to the game tree of the existential k-pebble game. Thus, either $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c_0,$

$(n+1)^{2k}$) holds and Player I wins the game, or $\mathrm{Win}_k(\mathbf{A}, \mathbf{B}, c_0, (n+1)^{2k})$ does not hold and Player II wins the game. ▮

The idea underlying our next result is that instead of testing whether a pattern structure **A** can be embedded in **B** we can test whether Player II wins the game played on the pair **A** and **B**.

PROPOSITION 5.4. *Let Q be a pattern-based query with a pattern generator α such that Q is expressible in L^k. Then a finite structure* **B** *satisfies Q if and only if there is a pattern structure $\mathbf{A} \in \alpha(\mathbf{B})$ such that Player II wins the existential k-pebble game on the pair* **A**, **B**.

Proof. If **B** satisfies Q, then there is a pattern structure $\mathbf{A} \in \alpha(\mathbf{B})$ and a one-to-one homomorphism h of **A** into **B**. It follows that II wins the existential k-pebble game on **A**, **B** by using the homomorphism h: if Player I places his lth pebble on an element a of **A**, then Player II places his lth pebble on the element $h(a)$ of B.

Conversely, if II wins the existential k-pebble game on **A**, **B** for some pattern structure $\mathbf{A} \in \alpha(\mathbf{B})$, then, by Theorem 4.8, **B** satisfies Q, since by definition **A** satisfies Q. ▮

By combining Propositions 5.4 and 5.3 we obtain the desired result.

THEOREM 5.5. *Let Q be a pattern-based query that is expressible in L^ω. Then Q can be answered in polynomial time.*

COROLLARY 5.6. *Suppose* $\mathrm{P} \neq \mathrm{NP}$. *Then the even simple path query is not expressible in L^ω. Similarly, if a fixed subgraph homeomorphism query is NP-complete, then it is not expressible in L^ω.*

The results in Lakshmanan and Mendelzon [LM89] actually yield a stronger result for the even simple path query, namely that this query is not expressible in L^ω (without assuming that $\mathrm{P} \neq \mathrm{NP}$). In the next section we show that this also the case for those fixed subgraph homeomorphism queries that are NP-complete.

6. A CASE STUDY: FIXED SUBGRAPH HOMEOMORPHISM QUERIES

Let $H = (V_H, E_H)$ be fixed directed graph with nodes $v_1, \ldots, v_l$ and edges $e_1, \ldots, e_m$. Let $G = (V, E, s_1, \ldots, s_l)$ be a directed graph with l distinguished points $s_1, \ldots, s_l$ such that $s_i \neq s_j$ for $i \neq j$. We say that H is *homeomorphic to the distinguished subgraph of G* if G contains m pairwise node-disjoint simple paths $p_1, \ldots, p_m$[3] such that if the edge e_i in H is from v_j to $v_{j'}$, then p_i is a path from s_j to $s_{j'}$.

[3] Two simple paths are node-disjoint if they have no point in common, except that endpoints may be equal.

The *H-subgraph homeomorphism* query is to determine, for a fixed graph H, whether H is homeomorphic to the distinguished subgraph of an input graph G. We use the term *pattern graphs* to refer to such graphs H. As we observed earlier, fixed subgraph homeomorphism queries are pattern-based queries. In what follows we assume that pattern graphs have no isolated nodes, since if they do, then we can remove all isolated nodes and obtain a fixed subgraph homeomorphism query that is equivalent to the original one.

Fortune *et al.* [FHW80] classified the computational complexity of fixed sub-graph homeomorphism queries in terms of two different dichotomies that are described in what follows.[4]

The first dichotomy is expressed in terms of the collection C of all directed graphs with a distinguished node called the *root* and with the property that either the root is the head of every edge or the root is the tail of every edge (note that the root can be both the head and the tail of an edge, when the graph contains a self-loop from the root to itself). Fortune *et al.* [FHW80] showed that the H-subgraph homeomorphism query can be answered in polynomial time if the pattern graph H is in C, but it is NP-complete if the pattern graph H is in the complement $\bar{C}$ of C.

For the second dichotomy Fortune *et al.* [FHW80] considered the restriction of the subgraph homeomorphism problem to acyclic graphs. They showed that if only acyclic graphs are allowed as inputs, then the H-subgraph homeomorphism query can be answered in polynomial time for every pattern graph H.

Note that these two dichotomies yield a computational distinction only if $\mathrm{P} \neq \mathrm{NP}$. In this section, without using any complexity theoretic assumptions, we establish that the two dichotomies are indeed proper in terms of expressibility in Datalog($\neq$). More specifically, on the positive side we show that the H-subgraph homeomorphism query is expressible in Datalog($\neq$) for every pattern graph H in the class C. We also prove that if only acyclic graphs are allowed as inputs, then the H-subgraph homeomorphism query is expressible in Datalog($\neq$) for every pattern graph H. Finally, on the negative side we demonstrate that if the pattern graph H is in the complement $\bar{C}$ of C, then the fixed subgraph homeomorphism query is *not* expressible in L^ω and, a fortiori, is not expressible in Datalog($\neq$) as well. The rest of the present section is devoted to the proofs of these results.

6.1. *Positive Results*

As mentioned earlier, Fortune *et al.* [FHW80] exhibited polynomial-time algorithms for certain cases of the fixed

[4] We note that Fortune *et al.* actually consider *multigraphs*; i.e., they allow multiple edges between pairs of nodes, while we consider only graphs; i.e., we do not allow multiple edges between pairs of nodes. Their results, however, apply to graphs as well.

subgraph homeomorphism problem. We strengthen here these results by establishing that in these cases the fixed subgraph homeomorphism problem is actually expressible in Datalog($\neq$).

THEOREM 6.1. *Let H be a pattern graph in the class C. Then the H-subgraph homeomorphism query is expressibly in* Datalog($\neq$).

Proof. Fortune *et al.* show that for pattern graphs H in C the H-subgraph homeomorphism problem can be reduced to a network flow problem, where we want to know whether the input graph, viewed as an appropriate directed network with node capacities, can carry a flow greater or equal than the outdegree k of the root (or the indegree k of the root if the root is the tail of every edge). This reduction establishes that the H-subgraph homeomorphism query is decidable in polynomial time for pattern graphs H in C.

Using the max-flow min-cut theorem (cf. [Bol79]) and the fact that k is fixed, we can express this query by a Datalog($\neq$) program. We first illustrate the proof for the case where H has nodes v, v_1, v_2 and edges $(v, v_1), (v, v_2)$. In this case if G is an input graph and s, s_1, s_2 are the distinguished nodes of G, then the H-subgraph homeomorphism query $Q(s, s_1, s_2)$ becomes "are there node-disjoint simple paths from s to s_1 and to s_2?"

Recall that if w is a node in a graph, then a path π in the graph is *w-avoiding* if π does not go through w. More generally, if $t_1, ..., t_l$ are nodes in a graph, then a path π in the graph is *$\{t_1, ..., t_l\}$-avoiding* if π does not go through any of the nodes $t_1, ..., t_l$.

Let $Q'(s, s_1, s_2)$ be the query: "is there a path $w_1 = s, w_2, ..., w_m = s_2$ from s to s_2 such that for every w_i, $2 \leqslant i \leqslant m$, there is a w_i-avoiding path from s to s_1?" We claim that $Q'(s, s_1, s_2)$ holds precisely when there are node-disjoint simple paths from s to s_1 an so s_2 (i.e., when $Q(s, s_1, s_2)$ holds). Indeed, it is first of all obvious that if there are node-disjoint simple paths from s to s_1 and to s_2, then $Q'(s, s_1, s_2)$ holds. Conversely, assume that $Q'(s, s_1, s_2)$ holds in G, but that any two simple paths from s to s_1 and from s to s_2 intersect at some node. Then by Menger's theorem for directed graphs (or, equivalently, by the max-flow min-cut theorem) (cf. Bollobas [Bol79]) there is a node w in G such that any two simple paths from s to s_1 and from s to s_2 intersect at w. This, however, violates the fact that $Q'(s, s_1, s_2)$ holds.

Thus, it remains to show that the query $Q'(x, y, z)$ is expressible in Datalog($\neq$). Let $T(u, v, w)$ be the query "is there a w-avoiding path from u to v?" As it was shown in Example 2.1, T is expressible in Datalog($\neq$). Consider now the following program in which T is viewed as an EDB predicate:

$$Q'(s, s_1, s_2) \leftarrow E(s, s_2), T(s, s_1, s_2)$$

$$Q'(s, s_1, s_2) \leftarrow Q'(s, s_1, w), E(w, s_2), T(s, s_1, s_2).$$

It is easy to verify that the above program does indeed compute Q'.

For the general case, suppose that the pattern graph H has a root with outdegree k (the case where the root has indegree k is analogous), and, furthermore, assume for now that H does not contain a self-loop. In this case the input graph G has distinguished distinct nodes $s, s_1, ..., s_k$, and the H-subgraph homeomorphism query $Q_H(s, s_1, ..., s_k)$ asks whether there are k node-disjoint simple paths from s to $s_1, ..., s_k$. To show that this query is expressible in Datalog($\neq$), we prove the expressibility of a stronger query, where the input graph contains also distinguished nodes $t_1, ..., t_l$ and the query is whether there are k node-disjoint simple $\{t_1, ..., t_l\}$-avoiding paths from s to $s_1, ..., s_k$. Denote this query by $Q_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$. Note that the original H-subgraph homeomorphism query $Q_H(s, s_1, ..., s_k)$ is simply $Q_{k,0}(s, s_1, ..., s_k)$. We prove expressibility of $Q_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$ in Datalog($\neq$) by induction on k.

For $k = 1$, the query $Q_{1,l}(s, s_1, t_1, ..., t_l)$ is a generalization of the query of Example 2.1 (indeed the query in that example is simply $Q_{1,1}$) and is expressed by the program

$$Q_{1,l}(s, s_1, t_1, ..., t_l) \leftarrow E(s, s_1), \quad s \neq t_1, ..., s \neq t_l, \quad s_1 \neq t_1, ..., s_1 \neq t_l$$

$$Q_{1,l}(s, s_1, t_1, ..., t_l) \leftarrow Q_{1,l}(s, w, t_1, ..., t_l), \quad E(w, s_1), \quad s_1 \neq t_1, ..., s_1 \neq t_l.$$

Assume inductively that we have shown how to express $Q_{k-1,l}$, $l \geqslant 0$, in Datalog($\neq$). Let $Q'_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$ be the query "is there a path $w_1 = s, w_2, ..., w_m = s_k$ from s to s_k such that for every w_i, $2 \leqslant i \leqslant m$, there are $k-1$ node-disjoint simple $\{w_i, t_1, ..., t_l\}$-avoiding paths from s to $s_1, ..., s_{k-1}$ (i.e., $Q_{k-1,l+1}(s, s_1, ..., s_{k-1}, w_i, t_1, ..., t_l)$ holds). We claim that $Q'_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$ holds precisely when $Q_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$ holds.

Indeed, it is obvious that if there are node-disjoint simple *$\{t_1, ..., t_l\}$-avoiding* paths from s to $s_1, ..., s_k$ then $Q'_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$ holds. Conversely, assume that $Q'_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$ holds, but there are no node-disjoint simple *$\{t_1, ..., t_l\}$-avoiding* path from s to $s_1, ..., s_k$. Then by Menger's theorem for directed graphs, or, equivalently, by the max-flow min-cut theorem (cf. Bollobas [Bol79]), there are nodes $u_1, ..., u_{k-1}$ such that any $\{t_1, ..., t_l\}$-avoiding path from s to any one of the nodes $s_1, ..., s_k$ must go through one of the nodes $u_1, ..., u_{k-1}$. Since $Q'_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$ holds, there is a path $w_1 = s, w_2, ..., w_m = s_k$ from s to s_k such that for every w_i, $2 \leqslant i \leqslant m$, there are $k-1$ node-disjoint simple $\{w_i, t_1, ..., t_l\}$-avoiding paths from s to $s_1, ..., s_{k-1}$. We know that there are i, j, where $2 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant k-1$, such that $w_i = u_j$. It follows that there are no $k-1$ node-disjoint simple $\{w_i, t_1, ..., t_l\}$-avoiding paths from s to $s_1, ..., s_{k-1}$, which is a contradiction.

It remains to show that $Q'_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$ is expressible in Datalog($\neq$):

$$Q'_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$$
$$\leftarrow E(s, s_k), Q_{k-1,l+1}(s, s_1, ..., s_k, t_1, ..., t_l)$$
$$Q'_{k,l}(s, s_1, ..., s_k, t_1, ..., t_l)$$
$$\leftarrow Q'_{k,l}(s, s_1, ..., w, t_1, ..., t_l), E(w, s_k),$$
$$Q_{k-1,l+1}(s, s_1, ..., s_k, t_1, ..., t_l).$$

Finally, we have to deal with the case in which H does contain a self-loop. In this case, the input graph G has distinguished distinct nodes s, s_1, ..., s_k, and the H-subgraph homeomorphism query $Q_H(s, s_1, ..., s_k)$ asks whether there are $k+1$ node-disjoint simple paths from s to $s, s_1, ..., s_k$. Clearly, $Q_H(s, s_1, ..., s_k)$ holds if and only if either $Q_{k,0}(s, s_1, ..., s_k)$ holds and G contains a self-loop on s or there is node s_{k+1} that is distinct from $s, s_1, ..., s_k$, such that there is an edge from s_{k+1} to s and $Q_{k+1,0}(s, s_1, ..., s_{k+1})$ holds. Thus, it is easy to see that Q_H can be expressed in Datalog($\neq$). ∎

As mentioned earlier, [FHW80] showed that if the input graphs are restricted to be acyclic, then all fixed subgraph homeopmorphism queries can be answered in polynomial time. We prove next that in this case the queries are actually expressible in Datalog($\neq$).

THEOREM 6.2. *Let H be a fixed graph. Then there is a* Datalog($\neq$) *program π_H that expresses the H-subgraph homeomorphism problem for acyclic graphs.*

Proof. Consider a pattern graph H with nodes $v_1, ..., v_k$. We denote the edge between v_i and v_j, if it exists, by e_{ij}. Here the input graph G has distinguished distinct nodes $s_1, ..., s_k$, and we have to determine whether H is homeomorphic to the distinguished subgraph of G. Consider the following pebble game between two players on a graph G. (We note that this game is different from the existential pebble game described in Section 4. In particular, the current game is played on a single structure, while the games in Section 4 were played on pairs of structures.) To each edge e_{ij} in H there corresponds a pebble p_{ij}. A position in the game consists of a placement of a subset of the pebbles on the nodes of G. Initially, all pebbles are placed on G: each pebble p_{ij} is placed on s_i. A round of the game consists of the following:

- Player I points to one of the pebbles, say p_{ij}, that is placed on a node u.
- Player II moves p_{ij} to a node v such that there is an edge from u to v, there is no other pebble on v, and v is not a distiguished node of G except perhaps s_j. If v is s_j, then Player II removes p_{ij} from G.

Player I (resp., II) wins if II (resp., I) cannot make a move. Thus, Player II wins if and only if all pebbles have been removed from G. We claim that Player II has a winning strategy if and only if H is homeomorphic to the distinguished subgraph of G. The proof is almost identical to the proof of Lemma 4 in [FHW80], which relates homeomorphism in acyclic graphs to a certain *single*-player pebble game.

Suppose first that H is homeomorphic to the distinguished subgraph of G. That is, for each edge e_{ij} in H the input graph G contains a node-disjoint simple path π_{ij} such that π_{ij} is a path from s_i to s_j and the paths are pairwise node-disjoint. Then Player II's strategy is simply to move the pebble p_{ij} along the path π_{ij}. Ultimately, p_{ij} arrives at s_j and is removed.

Suppose now that Player II has a winning strategy. Define the *level* of a node in G to be the length of the longest path in G from that node. Levels are well-defined, since G is acyclic. Suppose that Player I always points to a pebble on a node with a maximal level among all pebbled nodes (ties are broken arbitrarily). We know that Player II wins even against this strategy for Player I, since Player II has a winning strategy. Clearly, pebble p_{ij} traces a path π_{ij} from s_i to s_j. We claim that these paths are pairwise node-disjoint. Suppose that the paths π_{ij} and π_{mn} intersect at a node v that is not their endpoint. Note that, by the rules of the game, v is not a distinguished node of G. Suppose p_{mn} was placed on v first. It must have been removed from v before p_{ij} was placed on v. Suppose that p_{ij} was placed on a node u when p_{mn} was removed from v. Clearly, there is a path from u to v. Thus, the level of u is higher than the level of v, and Player I should have pointed to p_{ij} and not to p_{mn}, which is a contradiction.

It remains to show that the existence of a winning strategy for Player II can be expressed in Datalog($\neq$). We demonstrate this on a simple example and leave the general case to the reader. Consider a particular fixed subgraph homeomorphism query: the *two node-disjoint paths* query. Here we have to determine whether an input graph G contains node-disjoint simple paths between distinguished nodes s_1 and t_1 and between distiguished nodes s_2 and t_2. The four distinguished nodes are distinct. This can be determined by considering the following pebble game between two players on the graph G. Here there are two pebbles, p_1 and p_2. A position in the game consists of a placement of a subset of the pebbles on the nodes of G. Initially, p_1 is placed on s_1 and p_2 is placed on s_2. A round of the game consists of the following:

- Player I points to one of the pebbles, say p_i, that is placed on a node u.
- Player II moves p_i to a node v such that there is an edge from u to v, there is no other pebble on v, and v is not a distinguished node of G except perhaps t_i. If v is t_i, then Player II removes p_i from G.

Player I (resp., II) wins if II (resp., I) cannot make a move.

It is easy to show that Player II has a winning strategy in the above game if and only if the query $D(s_1, s_2)$ holds:

$$
\begin{aligned}
&D(t_1, t_2) \leftarrow \\
&D(t_1, y) \leftarrow E(y, y'), D(t_1, y'), y \neq s_1, y \neq t_1, y' \neq s_2 \\
&D(x, t_2) \leftarrow E(x, x'), D(x', t_2), x \neq s_2, x \neq t_2, x' \neq s_1 \\
&D(x, y) \leftarrow x \neq y, x \neq s_2, x \neq t_2, x \neq t_1, \\
&\qquad y \neq s_1, y \neq t_1, y \neq t_2, \\
&\qquad E(y, y'), D(x, y'), y' \neq s_2 \\
&\qquad E(x, x'), D(x', y), x' \neq s_1. \quad \blacksquare
\end{aligned}
$$

It should be pointed out that the preceding positive results presented in Theorem 6.1 and Theorem 6.2 depend on the use of inequalities in Datalog($\neq$) programs in a crucial way. Indeed, fixed subgraph homeomorphism queries are not expressible in Datalog (without inequalities), since such queries are not preserved under homomorphisms.

6.2 *Negative Results*

We focus now on fixed subgraph homeomorphism queries in which the pattern graph H is in the complement $\bar{C}$ of the class C. Note that the results of [FHW80] and the preceding Corollary 5.6 imply that if P $\neq$ NP, then the H-subgraph homeomorphism query is not expressible in L^ω, for any pattern graph H in $\bar{C}$.

We will derive here the same conclusion without appealing to the assumption that P $\neq$ NP. The main idea behind the proof is to convert a complexity lower-bound proof to an expressibility lower-bound proof. For this, we first analyze the NP-hardness proof of the fixed subgraph homeomorphism queries for pattern graphs H in $\bar{C}$ and extract certain graphs on which we play existential k-pebble games. We then apply the game-theoretic characterization of L^ω given by Theorem 4.10 to obtain the lower bound on the expressibility of the fixed subgraph homeomorphism queries.

Recall that the class C consists of all directed graphs with a distinguished node and with the property that either this distinguished node is the head of every edge or it is the tail of every edge. It follows that the complement $\bar{C}$ of the class C is the collection of all directed graphs G such that G contains at least one of the following subgraphs:

1. A graph H_1 that consists of two disjoint edges; i.e., H_1 has four distinct nodes s_1, s_2, s_3, s_4 and two disjoint edges (s_1, s_2), (s_3, s_4).

2. A graph H_2 that is a path of length two through three distinct nodes; i.e., H_2 has three distinct nodes s_1, s_2, s_3 and edges (s_1, s_2), (s_2, s_3).

3. A graph H_3 that is a cycle of length two; i.e., H_3 has two distinct nodes s_1, s_2 and edges (s_1, s_2), (s_2, s_1).

Note that these three graphs give rise to the following natural fixed subgraph homeomorphism queries:

1. Given a directed graph G and four distinct nodes s_1, s_2, s_3, s_4, are there two node-disjoint simple paths from s_1 to s_2 and from s_3 to s_4?

2. Given a directed graph G and three distinct nodes s_1, s_2, s_3, is there a simple path from s_1 to s_3 that goes through s_2?

3. Given a directed graph G and two distinct nodes s_1 and s_2, is there a simple cycle containing both s_1 and s_2?

We wish to establish that if the pattern graph H is in the class $\bar{C}$, then the fixed subgraph homeomorphism query with pattern H is not expressible in L^ω. The following lemma and the preceding description of the class $\bar{C}$ imply that it is enough to establish this only for the subgraph homeomorphism queries with pattern H_1, H_2, and H_3.

LEMMA 6.3. *Assume that F_1 and F_2 are two directed graphs such that F_1 is a subgraph of F_2. If the fixed subgraph homeomorphism query with pattern F_1 is not expressible in L^ω, then the fixed subgraph homeomorphism query with pattern F_2 is not expressible in L^ω as well.*

Proof. We have to show that there is no $k \geq 1$ such that the fixed subgraph homeomorphism query with pattern F_2 is expressible in the infinitary logic L^k. Assume that F_1 has l nodes $s_1, \ldots, s_l$ and that F_2 has m nodes $s_1, \ldots, s_l, s_{l+1}, \ldots, s_m$. As explained earlier, we can assume without loss of generality that F_1 and F_2 contain no isolated nodes. Since the fixed subgraph homeomorphism query with pattern F_1 is not expressible in L^ω, the direction (2) $\Rightarrow$ (1) of Theorem 4.10 implies that for each $k \geq 1$ there is a graph A_k with distinguished nodes $a_1, \ldots, a_l$ and a graph B_k with distinghuished nodes $b_1, \ldots, b_l$ such that the following hold:

1. F_1 is homeomorphic to the distinguished subgraph of A_k;

2. F_1 is not homeomorphic to the distinguished subgraph of B_k;

3. Player II has a winning strategy for the existential k-pebble game on the graphs $(A_k, a_1, \ldots, a_l)$ and $(B_k, b_1, \ldots, b_l)$.

Let $F_2 - F_1$ be the graph consisting of the edges of F_2 that are not in F_1, together with the nodes that are incident to these edges. We construct a graph A'_k with distinguished nodes $a_1, \ldots, a_l, a'_{l+1}, \ldots, a'_m$ by adding a copy of $F_2 - F_1$ to A_k, where a node s_i of $F_2 - F_1$ that happens to be also a node of F_1 is identified with the associated distinguished node a_i of A_k. Moreover, the nodes $s_{l+1}, \ldots, s_m$ of F_2, which are not nodes of F_1, are associated with the distinguished nodes $a'_{l+1}, \ldots, a'_m$ of A'_k. In a similar manner we construct also a graph B'_k with distinguished nodes $b_1, \ldots, b_l$, $b'_{l+1}, \ldots, b'_m$.

We now claim that the following hold:

1. F_2 is homeomorphic to the distinguished subgraph of A'_k;

2. F_2 is not homeomorphic to the distinguished subgraph of B'_k;

3. Player II has a winning strategy for the existential k-pebble game on the graphs $(A'_k, a_1, ..., a_l, a'_{l+1}, ..., a'_m)$ and $(B'_k, b_1, ..., b_l, b'_{l+1}, ..., b'_m)$.

The first fact is obvious in view of the corresponding fact for F_1 and A_k above. For the second fact, one can show that the construction described above has the property that if F_2 is homeomorphic to the distinguished subgraph of B'_k, then F_1 is homeomorphic to the distinguished subgraph of B_k. Actually, this is contained in the proof of Lemma 1 in [FHW80] and is established by induction on the number of edges in $F_2 - F_1$.

Finally, Player II can win the existential k-pebble game on A'_k and B'_k by playing as follows. If Player I places a pebble on a node of A'_k that happens to be a node of A_k, then Player II responds by placing a pebble on a node of B_k according to his winning strategy for the existential k-pebble game on the graphs $(A_k, a_1, ..., a_l)$ and $(B_k, b_1, ..., b_l)$. If Player I places a pebble on a node a'_j of A'_k that is not a node of A_k, then Player II places a pebble on the corresponding node b'_j of B'_k.

It now follows from the direction (1) $\Rightarrow$ (2) of Theorem 4.10 that the fixed subgraph homeomorphism query with pattern F_2 is not expressible in L^k. ∎

The preceding Lemma 6.3 was inspired from Lemma 1 in [FHW80], which asserts that if F_1 is a subgraph of F_2 and the fixed subgraph heomeomorphism query with pattern F_1 is NP-hard, then the fixed subgraph homeomorphism query with pattern F_2 is also NP-hard.

Our next goal is to show that the fixed subgraph homeomorphism query with pattern H_1 (i.e., "are there two node-disjoint simple paths betwen two given pairs of distinct vertices?") is not expressible in L^ω. The proof of this result requires a thorough understanding of the proof that the fixed subgraph homeomorphism query with pattern H_1 is NP-hard. We now present a detailed proof of this NP-hardness result, which is the main technical accomplishment of [FHW80].

Fortune *et al.* [FHW80] show that the SATISFIABILITY problem for Boolean formulas in conjunctive normal form has a polynomial-time reduction to the fixed subgraph homeomorphism query with pattern H_1. A key ingredient of their reduction is a special graph, called a *switch*, that is depicted in Fig. 1. If p is a simple path in the switch, then we say that *p is passing through the switch* if it starts at a node having indegree zero and ends at a node having outdegree zero. Among the several paths that are passing through the switch, we single out the following six:

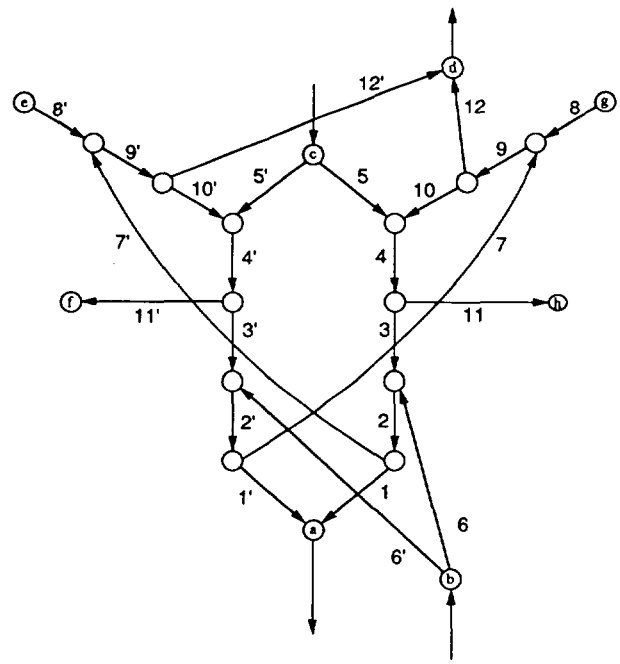

FIGURE 1

1. $p(c, a)$ is the path $5 \to 4 \to 3 \to 2 \to 1$.
2. $p(b, d)$ is the path $6' \to 2' \to 7 \to 9 \to 12$.
3. $p(e, f)$ is the path $8' \to 9' \to 10' \to 4' \to 11'$.
4. $q(c, a)$ is the path $5' \to 4' \to 3' \to 2' \to 1'$.
5. $q(b, d)$ is the path $6 \to 2 \to 7' \to 9' \to 12'$.
6. $q(g, h)$ is the path $8 \to 9 \to 10 \to 4 \to 11$.

The crucial combinatorial properties of the switch are presented in the lemma below, whose proof can be easily derived by inspecting the switch (cf. also [FHW80]).

LEMMA 6.4. *Suppose that two node-disjoint paths are passing through the switch, with one starting at node b and the other ending at node a. Then the path ending at a must start at c and the path starting at b must end at d. Moreover, these two paths must be either $p(c, a)$ and $p(b, d)$ or $q(c, a)$ and $q(b, d)$. In the first case, $p(e, f)$ is the only path that is passing through the switch and is node-disjoint from these two paths, while in the second case $q(g, h)$ is the only such path.*

We describe now the reduction of the SATISFIABILITY problem to the fixed subgraph homeomorphism query with pattern H_1, as given in [FHW80]. Assume that φ is a Boolean formula in conjunctive normal form with variables $x_1, ..., x_k$ and clauses $c_1, ..., c_l$. Using this formula we will construct a graph G_φ that will give rise to an instance of the fixed subgraph homeomorphism query with pattern H_1.

The graph G_φ has two main "building blocks," one for the variables of φ and one for its clauses. In addition, there are

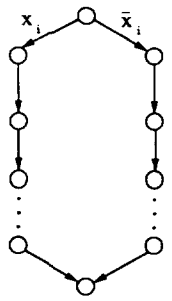

FIGURE 2

several switches, one for each occurrence of a literal in a clause of φ.

The building blocks for the variables $x_1, ..., x_k$ of φ are suitable copies of the graph of Fig. 2, one for each variable x_i. The left column of vertical edges is associated with the literal x_i, while the right column is associated with the literal $\bar{x}_i$. The number of edges in each vertical column is equal to the number of occurrences of its associated literal in the formula φ. When we construct the graph G_φ later on, each vertical edge will be replaced by a path of the form $q(g, h)$ from an appropriate switch. The building blocks for the variables are linked together by connecting the bottom node of the graph for x_i with the top node of the graph for x_{i+1}.

The building block for the clauses $c_1, ..., c_l$ of φ has nodes $n_0, ..., n_l$ with certain node-disjoint paths from n_{j-1} to n_j $1 \leqslant j \leqslant l$. The number of paths from the node n_{j-1} to the node n_j is equal to the number of literals occurring in the clause c_j, $1 \leqslant j \leqslant l$.

The graph G_φ is now constructed from these building blocks and the switches as follows (see Fig. 3):

1. For each occurrence of a literal y in a clause c_j, we replace one of the vertical edges in the column associated with y by the path $q(g, h)$ of the switch associated with this occurrence. We also set one of the paths from the node n_{j-1} to n_j to be the path $p(e, f)$ of the same switch.

2. We link all the switches together by first imposing an arbitrary order on the switches and then connecting the d

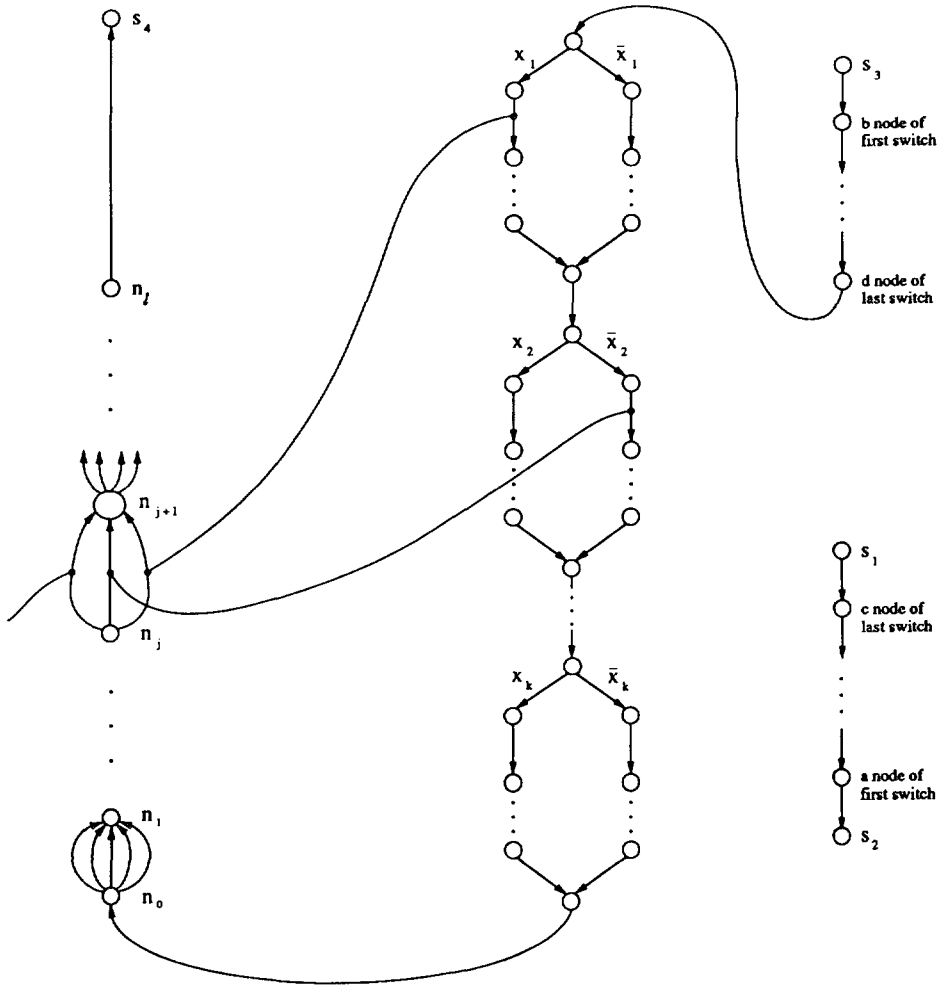

FIGURE 3

node of each switch to the b node of the next switch and the a node of each switch to the c node of the previous switch (see Fig. 4).

3. We link the building blocks for the variables to the building block for the clauses by adding an edge from the bottom of the building block for the variable x_k to the node n_0.

4. Finally, we add four distinguished nodes s_1, s_2, s_3, s_4 and the following five edges: an edge from s_1 to the c node of the last switch; an edge from the a node of the first switch to s_2; an edge from s_3 to the b node of the first switch; an edge from the d node of the last switch to the top of the building block for variable x_1; finally, an edge from n_l to s_4.

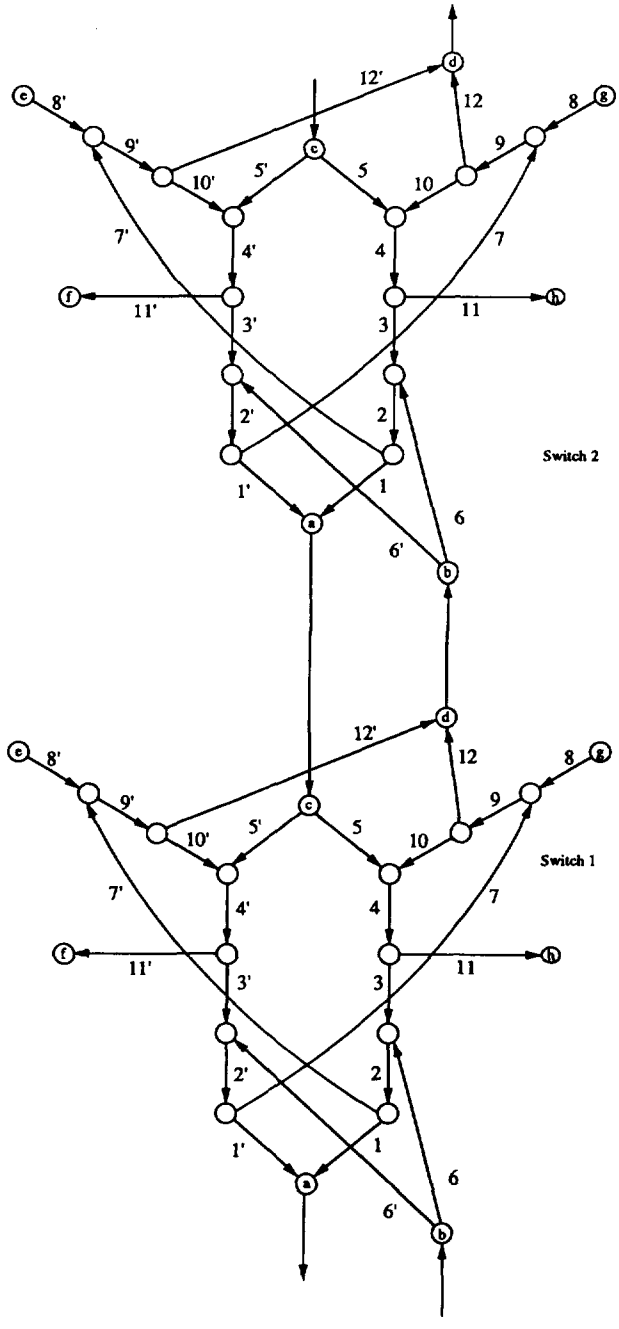

FIGURE 4

Figures 5 and 6 depict the graphs G_φ for the cases where φ is the formula $x_1 \vee \bar{x}_1$ and the formula $x_1 \wedge \bar{x}_1$, respectively.

Fortune *et al.* [FHW80] showed that the SATISFIABILITY problem with instance a Boolean formula φ reduces to the H_1-subgraph homeomorphism query with input the graph $(G_\varphi, s_1, s_2, s_3, s_4)$. In other words, they showed that a Boolean formula φ is satisfiable if and only if in the graph G_φ there are two node-disjoint simple paths from s_1 to s_2 and from s_3 to s_4.

Assume first that the formula φ is satisfiable. In this case, any fixed satisfying assignment for φ can be used to construct two node-disjoint simple paths p_1 from s_1 to s_2 and p_2 from s_3 to s_4. The part of p_2 along the building blocks for the variables of φ is specified by requiring that if a literal y is true under the satisfying assignment, then p_2 goes through the column associated with the complement $\bar{y}$ of y in the building blocks for the variables. Thus, p_2 is required to contain paths of the form $q(g, h)$ from some of the switches. The part of p_2 along the building block for the clauses is specified by requiring that p_2 links n_{j-1} to n_j, $1 \leqslant j \leqslant l$, by going through a path $p(e, f)$ from a switch associated with an occurrence in clause c_j of a literal that is true under this satisfying assignment. Consequently, for every switch in the graph $\mathcal{G}_\varphi$ at most one of the paths $p(e, f)$ and $q(g, h)$ from that switch is committed to be part of the path p_2. We can now extend this specification to a simple path p_2 from s_3 to s_4 as follows: if p_2 contains the path $p(e, f)$ of a switch, then we require that p_2 also contains the path $p(b, d)$ of the same switch; if p_2 contains the path $q(g, h)$ of a switch, then we require that p_2 also contains the path $q(b, d)$ of the same switch; if p_2 contains neither the path $p(e, f)$ nor the path $q(g, h)$ of some switch, then we choose one of the paths $p(b, d)$ or $q(b, d)$ of that switch and require p_2 to contain the path we chose. We can also find a simple path p_1 from s_1 to s_2 that is node-disjoint from p_2. We only have to require that p_1 contains either the path $p(c, a)$ or the path $q(c, a)$ of every switch, depending on whether p_2 contains the path $p(b, d)$ or the path $q(b, d)$ of that switch.

Assume next that in the graph G_φ there are two node-disjoint simple paths p_1 from s_1 to s_2 and p_2 from s_3 to s_4. Let us examine the possible routings of these two paths through the graph G_φ; see Figs. 3 and 4. Path p_1 leaves the first switch at node a and path p_2 enters the first switch at node b. Since b, c, e, and g are the only nodes of the first switch through which a path can enter the switch, it follows that the intersection of p_1 with the first switch is a path passing through the switch. Similarly, since a, d, f, and h are the only nodes of the first switch through which a path can leave the switch, it follows that the intersection of p_2 with the first

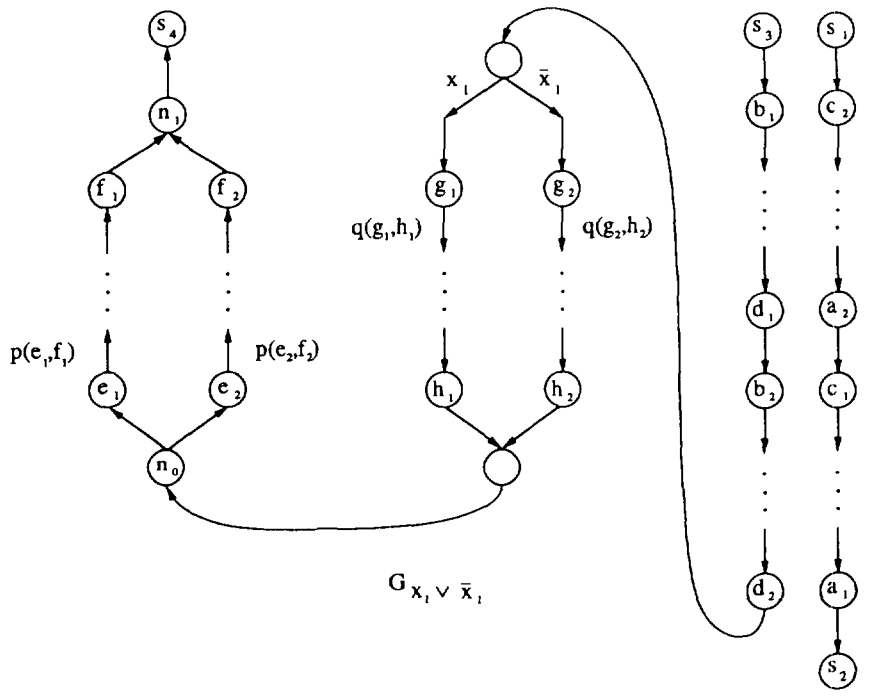

FIGURE 5

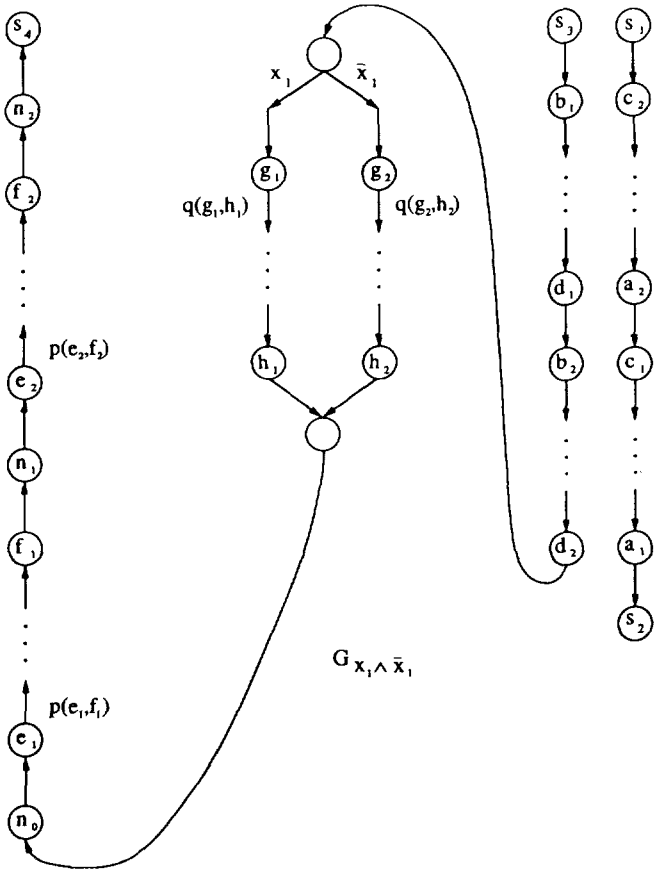

FIGURE 6

switch is also a path passing through the switch. Since these intersections are node-disjoint simple paths, Lemma 6.4 implies that p_1 must enter the first switch at c and path p_2 must leave the first switch at d. This, in turn, implies that p_1 must leave the second switch at node a of it and path p_2 must enter the second switch at node b of it. By repeating the previous argument for each switch, we infer that path p_1 must pass through the a node and through the c node of each switch, while path p_2 must pass through the b node and through the d node of each switch. In addition, Lemma 6.4 implies that path p_1 must be made up of segments that are paths of the form $p(a, c)$ or $q(a, c)$, and for each switch exactly one of the paths $p(a, c)$, $q(a, c)$ is a segment of p_1. Similarly, the part of p_2 from s_3 to the top of the building blocks for the variables consists of segments that are paths of the form $p(b, d)$ or $q(b, d)$, and for each switch exactly one of the paths $p(b, d)$, $q(b, d)$ is a segment of p_2.

Let us examine the routing of path p_2 through the building blocks of the variables. Since p_2 is a simple path that is disjoint from p_1, the preceding analysis shows that, whenever p_2 enters a node g of some switch, it must travel along the path $q(g, h)$ of that switch and exit at node h. As a result, we have that p_2 contains exactly one of the two vertical columns of each building block associated with a variable of φ. Similarly, whenever p_2 enters a node e of some switch, it must travel along the path $p(e, f)$ of that switch and exit at node f. Thus, p_2 must go through every node n_j and, moreover, it must reach n_j from n_{j-1} by traveling along a path $p(e, f)$ of some switch.

We now set a literal in φ to be true if and only if the path p_2 contains the vertical column associated with the literal $\bar{y}$. The preceding comments show that this truth assignment is well defined. Moreover, this assignment satisfies the formula φ. Indeed, assume that p_2 reaches n_j from n_{j-1} by traveling along a path $p(e, f)$ of some switch that is associated with the occurrence of a literal y in clause c_j, $1 \leqslant j \leqslant l$. In this case, p_2 can not contain the path $q(g, h)$ of the same switch and, hence, it must contain the vertical column associated with $\bar{y}$. As a result, y is true and clause c_j is satisfied. This completes the proof that the fixed subgraph homeomorphism query with pattern H_1 is NP-hard.

We are ready to return to our goal, which is to show that the fixed subgraph homeomorphism query with pattern H_1 is not expressible in the infinitary logic L^ω. This will be achieved by playing existential k-pebble games on appropriate graphs that we will extract from the preceding NP-hardness proof. Since the winning strategy for Player II in these games will turn out to be quite complicated, we introduce first certain k-pebble games on Boolean formulas that will be used as an auxiliary device to describe the moves of Player II in the existential k-pebble games.

Usually, a *truth assignment* is understood to be a mapping from a set $\{x_1, ..., x_k\}$ of variables to the set {**true**, **false**} of truth values. In what follows, we will be considering "extended" truth assignments in which we keep tract of the truth values assigned to *literals*, i.e., both variables x_i and negated variables $\bar{x}_i$, $1 \leqslant i \leqslant k$. The understanding is that if $\bar{x}_i$ is assigned value **true**, then x_i is assigned value **false** at the same time, and vice versa. Similarly, if $\bar{x}_i$ is assigned value **false**, then x_i is assigned value **true** at the same time and vice versa.

DEFINITION 6.5. Let k be a positive integer and let φ be a Boolean formula in conjunctive normal form with variables $x_1, ..., x_m$ and clauses $c_1, ..., c_n$. The *k-pebble game between Players I and II on the formula φ* has the following rules:

Player I moves first by placing a pebble either on one of the literals $x_1, \bar{x}_1, ..., x_m, \bar{x}_m$ or on one of the clauses $c_1, ..., c_n$ of the formula φ. If the pebble is placed on a literal, then Player II has to assign a truth value to it; otherwise, Player II has to select a literal from the clause pebbled by Player I and assign the value **true** to it. The game continues this way until each player has made k moves. If at some point during these moves a literal has been assigned value both **true** and **false** by Player II, then Player I wins the game; otherwise, Player I removes some of his pebbles and the game resumes until again he has placed k-pebbles and Player II has responded.

We say that *Player II wins the k-pebble game on the formula φ* if he has a *winning strategy* that allows him to continue playing the game "forever."

If the formula φ is satisfiable, then Player II wins the k-pebble game for every $k \geqslant 1$, by playing according to a satisfying assignment for φ. On the other hand, if φ is an unsatisfiable Boolean formula with k variables, then Player I wins the $(k+1)$-pebble game on φ. Indeed, Player I places one pebble on each positive literal of φ during his first k moves, forcing this way Player II to determine a truth assignment for all the literals of φ. Since φ is unsatisfiable, there is a clause c that is not satisfied by this truth assignment. Player I wins the $(k+1)$-pebble game on φ by placing his last pebble on the clause c. It should be pointed out that there are unsatisfiable formulas with k variables for which Player I can win the game with as few as two pebbles. Consider, for example, the formula

$$x_1 \wedge x_2 \wedge \cdots \wedge x_k \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \cdots \vee \bar{x}_k).$$

Player I wins the two-pebble game on the above formula, as follows. In his first move Player I places a pebble on the clause $(\bar{x}_1 \vee \bar{x}_2 \vee \cdots \vee \bar{x}_k)$. If Player II responds by making the literal $\bar{x}_i$ true, then Player I wins by placing his second pebble on the clause consisting of the literal x_i.

Let φ_k, $k \geqslant 1$, be the *complete Boolean formula on the variables* $x_1, ..., x_k$. This is the only formula in conjunctive normal form that has 2^k distinct clauses, each with k distinct literals. For example, φ_2 is the formula

$$(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2).$$

Note that each φ_k, $k \geqslant 1$, is an unsatisfiable formula and, consequently, Player I wins the $(k+1)$-pebble on φ_k. In contrast, Player II wins the k-pebble game on the formula φ_k by playing according to the following strategy. Suppose that Player I places a pebble on a literal y. If the truth value of y is determined at present, the Player II maintains this truth value; otherwise, Player II assigns a truth value (**true** or **false**) to y arbitrarily. Suppose that Player I places a pebble on a clause c of φ_k. This means that at most $(k-1)$ pairs of literals of the form $(x_i, \bar{x}_i)$ have their truth values determined at present. Thus, c contains at least one literal whose truth value is not determined at present. Player II wins by selecting such a literal from c and assigning value **true** to it.

THEOREM 6.6. *Let H_1 be a graph consisting of two disjoint edges. Then the fixed subgraph homeomorphism query with pattern H_1 is not expressible in the infinitary logic L^ω.*

Proof. We have to show that there is no $k \geqslant 1$ such that the fixed subgraph homeomorphism query with pattern H_1 is expressible in the infinitary logic L^k. Theorem 4.10 implies that it is enough to show that for every $k \geqslant 1$ we can find a graph A_k with distinguished nodes w_1, w_2, w_3, w_4 and a graph B_k with distinguished nodes s_1, s_2, s_3, s_4 such that the following hold:

1. In the graph A_k there are two node-disjoint simple paths from w_1 to w_2 and from w_3 to w_4;

2. In the graph B_k there is no pair of node-disjoint simple paths from s_1 to s_2 and from s_3 to s_4;

3. Player II has a winning strategy for the existential k-pebble game on the graphs $(A_k, w_1, w_2, w_3, w_4)$ and $(B_k, s_1, s_2, s_3, s_4)$.

The graph A_k will consists of just two node-disjoint simple paths. The lengths of these paths will be determined later on.

Let φ_k be the complete Boolean formula on the variables $x_1, ..., x_k$ and let G_{φ_k} be the graph associated with it in the reduction of the SATISFIABILITY problem to the fixed subgraph homeomorphism query with pattern H_1. We take $(B_k, s_1, s_2, s_3, s_4)$ to be the graph $(G_{\varphi_k}, s_1, s_2, s_3, s_4)$, where s_1, s_2, s_3, s_4 are the nodes introduced at the very end of the construction of G_{φ_k}. Since φ_k is an unsatisfiable formula, there is no pair of node-disjoint simple paths from s_1 to s_2 and from s_3 to s_4.

We now focus on certain paths from s_1 to s_2 and from s_3 to s_4 in G_{φ_k}, which we will call *standard* paths. Intuitively, a standard path would be a possible member of a pair of node-disjoint simple paths from s_1 to s_2 and from s_3 to s_4, if such a pair existed. More precisely, a *standard path from* s_1 *to* s_2 is a simple path from s_1 to s_2 such that for every switch of the graph G_{φ_k} the path goes through both node c and node a of that switch by containing as a segment exactly one of the paths $p(c, a)$ and $q(c, a)$ of that switch. Note that there are exponentially many standard paths from s_1 to s_2, but they are all of the same length, since the paths $p(c, a)$ and $q(c, a)$ of every switch have the same length.

A *standard path from* s_3 *to* s_4 is a path from s_3 to s_4 such that from s_3 to the top of the building blocks for the variables in G_{φ_k} it goes through both node b and node d of every switch by containing as a segment exactly one of the paths $p(b, d)$ and $q(b, d)$ of that switch. A standard path from s_3 to s_4 is also required to contain exactly one vertical column of the building block of each variable. In addition, it should go through every node n_j, $0 \leqslant j \leqslant 2^k$, and it should reach node n_j from node n_{j-1}, $1 \leqslant j \leqslant 2k$, by traveling along a path $p(e, f)$ of some switch. Note that again there are exponentially many standard paths from s_3 to s_4, but they are all of the same length, since the paths $p(b, d)$ and $q(b, d)$ have the same length in every switch, and all literals have the same number of occurrences in φ_k, which implies that all vertical columns are of the same length. It should also be pointed out that, unlike standard paths from s_1 to s_2, no standard path from s_3 to s_4 is a simple path. Indeed, it is not hard to show that if a standard path from s_3 to s_4 were simple, then φ_k would be satisfiable.

Let A_k be a graph consisting of two node-disjoint simple paths from a node w_1 to a node w_2 and from a node w_3 to a node w_4 such that the length of the first path is equal to the length of a standard path from s_1 to s_2 in the graph G_{φ_k}, while the length of the second is equal to the length of a standard path from s_3 to s_4 in G_{φ_k}.

Let p be a standard path from s_1 to s_2 in B_k. Since the path from w_1 to w_2 in A_k is a simple path having the same length as p, there is a natural mapping f between nodes in these two paths, such that if x is the ith node in the path from w_1 to w_2, then $f(x)$ is the ith node in the path p. In this case, we say that $f(x)$ is the node in p *corresponding* to the node x in the path from w_1 to w_2. Similarly, if q is a standard path from s_3 to s_4 in B_k, then there is a natural mapping g between nodes in the path from w_3 to w_4 in A_k and nodes in q, such that if z is the jth node in the path from w_3 to w_4, then $g(x)$ is the jth node in q. Again, we say that $g(z)$ is the node in q *corresponding* to the node z in the path from w_3 to w_4.

The proof of the theorem will be complete once we establish that Player II has a winning strategy in the existential k-pebble game on the graphs $(A_k, w_1, w_2, w_3, w_4)$ and $(B_k, s_1, s_2, s_3, s_4)$. The idea behind the strategy of Player II is as follows. If Player I places a pebble on a node in the path from w_1 to w_2 in A_k, then Player II will place a pebble on the corresponding node in some standard path from s_1 to s_2. On the other hand, if Player I places a pebble on a node in the path from w_3 to w_4, then Player II will place a pebble on the corresponding node in some standard path from s_3 to s_4. In addition, Player II has to ensure that the mapping from the pebbled nodes of A_k to the corresponding pebbled nodes of B_k is a one-to-one homomorphism. Player II will achieve this by making sure that at all times for every switch of the graph G_{φ_k} all pebbles on nodes of that switch are either on correponding nodes in the paths $p(c, a)$, $p(b, d)$, $p(e, f)$ of the switch or on corresponding nodes in the paths $q(c, a)$, $q(b, d)$, $q(g, h)$ of the switch.

We now describe the winning strategy of Player II in more detail. If Player I places a pebble on a node in A_k whose corresponding node is one of the nodes a, b, c, d of a switch, or one of the nodes linking one building block for a variable to the next, or one of the nodes n_j, $0 \leqslant j \leqslant 2^k$, then Player II places a pebble on the corresponding node in B_k. This is the trivial part of Player II's strategy. We focus next on the nontrivial part of his strategy and from now on we assume that all moves of Player I are on nodes other than the ones described above. In this case Player II views also each move of Player I as a move in a k-pebble game on the formula φ_k that is played simultaneously with the existential k-pebble game on the graphs $(A_k, w_1, w_2, w_3, w_4)$ and $(B_k, s_1, s_2, s_3, s_4)$. Player II keeps track of the current configuration of the k-pebble game on φ_k and uses his winning strategy for this game to determine his moves on the existential k-pebble game. Recall that during a k-pebble game on a formula in conjunctive normal form Player I challenges Player II to either assign a truth value to a literal or to select a literal from a clause and assign value **true** to it. Player II

has to keep playing in such a way that no literal has ever value both **true** and **false**. We describe next how Player II interprets the moves of Player I as moves in the k-pebble game on the formula φ_k. We distinguish the following four cases:

Case 1. Assume that Player I places a pebble on a node in the path from w_1 to w_2. The response of Player II in the existential k-pebble game will be to place a pebble on the corresponding node in a standard path from s_1 to s_2. The node that will be pebbled by Player II is going to be either in the path $p(c, a)$ of some switch or in the path $q(c, a)$ of the same switch. The switch is determined by the distance from w_1 of the node pebbled by Player I. Player II interprets also the move of Player I as a challenge in the k-pebble game on φ_k to assign a truth value to the literal y with which this switch is associated in G_{φ_k}.

Case 2. Assume that Player I places a pebble on a node in the path from w_3 to w_4 such that the corresponding node on a standard path is at a position between s_3 and the node at the top of the building blocks for the variables in G_{φ_k}. The response of Player II in the existential k-pebble game will be to place a pebble on the corresponding node in a standard path from s_3 to s_4. The node that will be pebbled by Player II is going to be either in the path $p(b, d)$ of some switch or in a path $q(b, d)$ of the same switch. Again, the switch is determined by the move of Player I. In addition, Player II interprets the move of Player I as a challenge in the k-pebble game on φ_k to assign a truth value to the literal y with which this switch is associated in G_{φ_k}.

Case 3. Assume that Player I places a pebble on a node in the path from w_3 to w_4 such that the corresponding node in a standard path is the vertical column for some variable x_i or in the vertical column for its negation $\bar{x}_i$. Player II will respond in the existential k-pebble game by selecting one of the two vertical columns and placing a pebble on the corresponding node in the column. He also views this move of Player I as a challenge in the k-pebble game on φ_k to assign a truth value to the literal x_i.

Case 4. Finally, assume that Player I places a pebble on a node in the path from w_3 to w_4 such that the corresponding node in a standard path is on the segment between node n_{j-1} and node n_j for some j. Player II will respond in the existential k-pebble game by placing a pebble on the corresponding node in one of the paths $p(e, f)$ from n_{j-1} to n_j. Thus, the move of Player I determines a clause c_j. Player II views this move of Player I as a challenge in the k-pebble game to select a literal from clause c_j and assign value **true** to it.

In all cases, the move of Player II in the k-pebble game will also enable him to determine his pending move in the existential k-pebble game. Player II keeps track of the k-pebble game on the formula φ_k by maintaining at all times a record of all literals that have a truth value assigned to them. He also keeps track of all the pebbled nodes that have caused a variable or a literal to acquire and maintain its present truth value. In particular, a truth value is removed from a literal as soon as no pebbled node forces it to have a truth value.

The moves of Player II in the k-pebble game on φ_k and in the existential k-pebble game are now described in more detail. Each of the first three cases above branches into two subcases depending on whether or not a truth value is currently assigned to the literal y (Cases 1 and 2) or to the variable x_i (Case 3) determined by the move of Player I. In Case 1, if the literal y does not have a truth value at present, then Player II assigns value **true** to it and pebbles the corresponding node in the path $p(c, a)$ of the switch associated with y. If y has a truth value, then Player II maintains this value in the k-pebble game on φ_k and moves as follows in the existential k-pebble game: if y has value **true**, then he pebbles the corresponding node in the path $p(c, a)$ of the switch; otherwise, he pebbles the corresponding node in the path $q(c, a)$ of the same switch. In Case 2, if the literal y does not have a truth value at present, then Player II assigns value **true** to it and pebbles the corresponding node in the path $p(b, d)$ of the switch associated with y. If y has a truth value, then Player II maintains this value in the k-pebble game on φ_k and moves as follows in the existential k-pebble game: if y has value **true**, then he pebbles the corresponding node in the path $p(b, d)$ of the switch; otherwise, he pebbles the corresponding node in the path $q(b, d)$ of the same switch.

In Case 3, if the variable x_i does not have a truth value at present, then Player II assigns value **true** to it and pebbles the corresponding node in the vertical column associated with the literal $\bar{x}_i$. If x_i has a truth value, then Player II maintains it and moves as follows in the existential k-pebble game: if x_i has value **true**, then he pebbles the appropriate node in the vertical column associated with $\bar{x}_i$; otherwise, he pebbles the corresponding node in the vertical column associated with x_i.

Finally, we describe the moves of Player II in Case 4. Since Player I has k pebbles and one of them has just been used to determine a clause c_j, at least one of the k literals occurring in the clause c_j does not have a truth value assigned to it at present. Player II selects such a literal and assigns value true to it. He also pebbles the corresponding node in the path $p(e, f)$ associated with the literal he selected.

We claim that the above constitutes a winning strategy for Player II in the existential k-pebble game between the graphs $(A_k, w_1, w_2, w_3, w_4)$ and $(B_k, s_1, s_2, s_3, s_4)$. For this, we have to verify that at all times during the game the mapping from the pebbled nodes of A_k to the corresponding pebbled nodes of B_k is a one-to-one homomorphism. Note

that the moves of Player II in the k-pebble game on φ_k are according to his winning strategy for this game. Thus, no literal is ever set to both true and false during the k-pebble game. In turn, this implies that at all times during the existential k-pebble game the following is true for every switch in the graph G_{φ_k}: all pebbles of Player II on nodes of that switch are either on nodes in the paths $p(c, a)$, $p(b, d)$, $p(e, f)$ of the switch or on nodes in the paths $q(c, a)$, $q(b, d)$, $q(g, h)$ of the switch. From this and from the fact that Player II always places his pebbles on nodes in standard paths corresponding to nodes pebbled by Player I, it follows that at all times the mapping between the pebbled nodes of A_k and the pebbled nodes of B_k is a homomorphism. Moreover, since in each case the three paths are pairwise disjoint, it follows that Player II is never forced to place two pebbles on the same node. Thus, the mapping between the pebbled nodes of A_k and the corresponding pebbled nodes of B_k is also one-to-one. The proof of the theorem is now complete. ∎

THEOREM 6.7. *Let H be a pattern graph in $\bar{C}$. Then the fixed subgraph homeomorphism query with pattern H is not expressible in the infinitary logic L^ω.*

Proof. Recall that the class $\bar{C}$ can be described as consisting of all directed graphs that contain at least one of the pattern graphs H_1, H_2, H_3 as a subgraph. We have just proved that the fixed subgraph homeomorphism query with pattern H_1 is not expressible in L^ω. The same conclusion can be derived for the pattern graphs H_2 and H_3, by first modifying slightly the graphs A_k and B_k, $k \geq 1$, in the proof of Theorem 6.6 and then playing the existential k-pebble game on the modified graphs. The pattern graph H_2 is a path of length two going through three distinct nodes. In this case we modify A_k and B_k by identifying node w_2 with node w_3 and node s_2 with node s_3. The pattern graph H_3 is a cycle of length two. For this pattern graph we modify A_k by identifying node w_1 with node w_4 and node w_2 with node w_3. We also modify B_k by identifying node s_1 with node s_4 and node s_2 with node s_3.

The conclusion of the theorem for arbitrary pattern graph H in $\bar{C}$ follows now immediately from the above by appealing to Lemma 6.3. ∎

The preceding Theorems 4.10, 6.6, and 6.7 provide us with tools for establishing that certain other pattern-based queries on directed graphs are not expressible in L^ω and, a fortiori, are not expressible in Datalog($\neq$). Consider, for example, the "even simple path query": given a directed graph G and two nodes s, t of G, is there a simple path of even length from s to t in G? As mentioned earlier, Lakshmanan and Mendelzon [LM89] established that this query is not expressible in Datalog($\neq$). We conclude by deriving this result as a corollary to the preceding Theorem 6.6.

COROLLARY 6.8. *The "even simple path" query on directed graphs is not expressible in the infinitary logic L^ω.*

Proof. Towards a contradiction, assume that there is a positive integer k such that the "even simple path" query is definable by a sentence of the infinitary logic L^k. We will use this assumption to show that the fixed subgraph homeomorphism query with pattern H_1 is expressible in L^{2k}, which will violate Theorem 6.6.

The proof is based on a reduction of the "two disjoint paths" query to the "even simple path" query. Given a directed graph G with distinguished nodes s_1, s_2, s_3, s_4, let G^* be the graph obtained from G by *doubling* every edge of G, adding a new node t, adding an edge from s_2 to s_3, and adding an edge from s_4 to t. Here, *doubling* every edge means that every edge (u, v) of G is replaced by a pair of edges (u, w) and (w, v), where w is a new node. It is easy to verify that there are two node-disjoint simple paths from s_1 to s_2 and from s_3 to s_4 in G if and only if there is a simple path of even length from s_1 to t in G^*.

Let A be a directed graph with distinguished nodes a_1, a_2, a_3, a_4 such that there are two node-disjoint simple paths from a_1 to a_2 and from a_3 to a_4 in A, and let B be a directed graph with distinguished nodes b_1, b_2, b_3, b_4 such that $(A, a_1, a_2, a_3, a_4) \leqslant^{2k} (B, b_1, b_2, b_3, b_4)$. We will show that there are two node-disjoint paths from b_1 to b_2 and from b_3 to b_4 in B. Once this is established, we will be able to infer that the fixed subgraph homeomorphism query with pattern H_1 is expressible in L^{2k}, by appealing to Theorem 4.10.

Let (A^*, a_1, t) and (B^*, b_1, t') be the result of applying the above reduction to the graphs (A, a_1, a_2, a_3, a_4) and (B, b_1, b_2, b_3, b_4). We now claim that $(A^*, a_1, t) \leqslant^k (B^*, b_1, t')$. This will be established by showing that Player II has a winning strategy for the existential k-pebble game on (A^*, a_1, t) and (B^*, b_1, t'). In addition to this game, Player II plays simultaneously an auxiliary existential $2k$-pebble game on (A, a_1, a_2, a_3, a_4) and (B, b_1, b_2, b_3, b_4). Player II keeps track of the auxiliary $2k$-pebble game at all times and uses his winning strategy in that game to determine his moves in the k-pebble game. The auxiliary game and the strategy of Player II for the k-pebble game are now described in detail. If Player I places a pebble on a node u of A^* that is also a node of A, then in the auxiliary game Player II views this as a move of Player I on u. Player II responds in the k-pebble game by pebbling the node u' of B that is pebbled by him in the auxiliary $2k$-pebble game according to his winning strategy. If Player I places a pebble on node w of A^* that is not in A, then there is a unique edge (u, v) in A such that (u, w) and (w, v) are edges of A^*. Player II views this as a sequence of two moves in the auxiliary game, namely as Player I placing a pebble first on u and then on v. Let u' and v' be the moves of Player II in the auxiliary game played according to his winning strategy. Since (u, v) is an edge of A, we must have that (u', v') is an

edge of B. As a result, there is a unique node w' in B^* such that (u', w') and (w', v') are edges of B^*. Then Player II responds in the existential k-pebble game by placing a pebble on w'. Finally, if Player I places a pebble on t, then Player II places a pebble on t'. It is not hard to verify that the above constitutes indeed a winning strategy for Player II in the existential k-pebble game on (A^*, a_1, t) and (B^*, b_1, t').

The existence of two node-disjoint simple paths from a_1 to a_2 and from a_3 to a_4 in the graph A implies that there is a simple path of even length from a_1 to t in A^*. Since $(A^*, a_1, t) \preccurlyeq^k (B^*, b_1, t')$ and since we assumed that the "even simple path" query is definable by a sentence of L^k, it follows that there is a simple path of even length from b_1 to t' in B^* and, consequently, there are two node-disjoint simple paths from b_1 to b_2 and from b_3 to b_4 in B. Theorem 4.10 implies now that the fixed subgraph homeomorphism query with pattern H_1 is expressible in L^{2k}, which, however, is a contradiction. ∎

7. CONCLUDING REMARKS

Our emphasis in this paper was on the study of the expressive power of Datalog($\neq$) by viewing it as a fragment of the infinitary logic L^ω. We showed that the game-theoretic characterization of the expressive power of L^ω provides us with tools for investigating the expressive power of Datalog($\neq$). We believe that these techniques are of interest not only from a database-theoretic perspective but also from a complexity-theoretic perspective. These technique enabled us to show that the dichotomies described in [FHW80] for fixed subgraph homeomorphism problems are indeed proper in terms of expressibility in Datalog($\neq$). This is similar in spirit to the result in [AF90], which uses expressibility in a certain logic to show that reachability in directed graphs is harder than reachability in undirected graphs. Our results indicate that the use of expressibility in Datalog variants to prove separation results deserves further study.

Finally, we note that in a recent paper Afrati *et al.* [ACY91] develop further tools to explore the expressive power of Datalog variants. In particular, using these tools they show that Datalog($\neq$) does not express all monotonic polynomial-time queries.

ACKNOWLEDGMENTS

We are grateful to N. Alon, R. Fagin, V. S. Lakshmanan, A. Mendelzon, N. Megiddo, and M. Yannakakis for stimulating discussions. In particular, playing pebble games with N. Alon led to Theorem 6.2.

REFERENCES

[ACY91] F. Afrati, S. S. Cosmadakis, and M. Yannakakis, On Datalog vs. polynomial time, *in* "Proceedings, 10th ACM Symposium on Principles of Database Systems, 1991."

[AF90] M. Ajtai and R. Fagin, Reachability is harder for directed than for undirected finite graphs, *J. Symbolic Logic* **55**, No. 1 (1990), 113–150.

[AG89] M. Ajtai and Y. Gurevich, DATALOG vs. first-order logic, *in* "Proceedings, 30th IEEE Symp. on Foundations of Computer Science, 1989," pp. 142–146.

[AU79] A. V. Aho and J. D. Ullman, Universality of data retrieval languages, *in* "Proceedings, 6th ACM Symp. on Principles of Programming Languages, 1979," pp. 110–117.

[Bar77] J. Barwise, On Moschovakis closure ordinals, *J. Symbolic Logic* **42** (1977), 292–296.

[BF85] J. Barwise and S. Feferman (Eds.), "Model-Theoretic Logics," Springer-Verlag, New York/Berlin, 1985.

[BG87] A. Blass and Y. Gurevich, Existential fixed-point logic, *in* "Computation Theory and Logic" (E. Börger, Ed.), Lecture Notes in Computer Science, Vol. 270, pp. 20–36, Springer-Verlag, New York/Berlin, 1987.

[Bol79] B. Bollobas, "Graph Theory," Springer-Verlag, New York/Berlin, 1979.

[CH82] A. Chandra and D. Harel, Structure and complexity of relational queries, *J. Comput. System Sci.* **25** (1982), 99–128.

[CH85] A. Chandra and D. Harel, Horn clause queries and generalizations, *J. Logic Programming* **1** (1985), 1–15.

[Coo74] S. A. Cook, An observation of time-storage trade-off, *J. Comput. System Sci.* **9** (1974), 308–316.

[dR87] M. de Rougemont, Second-order and inductive definability on finite structures, *Z. Math. Logik Grundlag. Math.* **33** (1987), 47–63.

[FHW80] S. Fortune, J. Hopcroft, and J. Wyllie, The directed homeomorphism problem, *Theoret. Comput. Sci.* **10** (1980), 111–121.

[Gai82] H. Gaifman, On local and nonlocal properties, *in* "Logic Colloquium '81" (J. Stern, Ed.), pp. 105–135, North Holland, Amsterdam, 1982.

[GMSV87] H. Gaifman, H. Mairson, Y. Sagiv, and M. Y. Vardi, Undecidable optimization problems for database logic programs, *in* "Proceedings, 2nd IEEE Symp. on Logic in Computer Science, 1987," pp. 106–115.

[GS86] Y. Gurevich and S. Shelah, Fixed-point extensions of first-order logic, *Ann. Pure Appl. Logic* **32** (1986), 265–280.

[IK89] N. Immerman and D. Kozen, Definability with bounded number of bound variables, *Inform. and Comput.* **83** (1989), 121–139.

[IL90] N. Immerman and E. S. Lander, Describing graphs: A first-order approach to graph canonization, *in* "Complexity Theory Retrospective" (A. Selman, Ed.), Springer-Verlag, 1990.

[Imm82] N. Immerman, Upper and lower bounds for first-order expressibility, *J. Comput. System Sci.* **25** (1982), 76–98.

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

[Kei71] H. J. Keisler, "Model Theory for Infinitary Logic," North Holland, Amsterdam, 1971.

[Kol85] Ph. G. Kolaitis, On asymptotic probabilities of inductive queries and their decision problem, *in* "Logics of Programs '85" (R. Parikh, Ed.), Lecture Notes in Computer Science, Vol. 193, pp. 153–166, Springer-Verlag, New York/Berlin, 1985.

[KV90] Ph. G. Kolaitis and M. Y. Vardi, Infinitary Logic and 0–1 Laws, *Inform. and Computation* **98** (1992), 258–294.

[LM89] V. S. Lakshmanan and A. O. Mendelzon, Inductive pebble games and the expressive power of Datalog, *in* "Proceedings, 8th ACM Symposium on Principles of Database Systems, 1989," pp. 301–310.

[Mos74] Y. N. Moschovakis, "Elementary Induction on Abstract Structures," North Holland, Amsterdam, 1974.

[Pap85] C. H. Papadimitriou, A note on the expressive power of Prolog, *Bull. EATCS* **26** (1985), 21–23.

[Rub75] A. Rubin, Ph.D. thesis, California Institute of Technology, 1975.

[Shm87] O. Shmueli, Decidability and expressiveness aspects of logic queries, *in* "Proceedings, ACM Symposium on Principles of Database Systems, 1987," pp. 237–249.

[Ull89] J. D. Ullman, "Database and Knowledge-Base Systems," Vol. I, II, Comput.-Sci. Press, Rockville, MD, 1989.

[Var82] M. Y. Vardi, The complexity of relational query languages, *in* "Proceedings, 14th ACM Symp. on Theory of Computing, 1982," pp. 137–146.

Printed in Belgium