QUERYING DATALOG PROGRAMS WITH TEMPORAL LOGIC

Alexander Tuzhilin

Information Systems Department
Stern School of Business
New York University

44 West 4th Street, Room 9-78
New York, NY 10012
atuzhilin@stern.nyu.edu
212-998-0832

To appear in: Acta Informatica

Working Paper Series
STERN IS-93-20

Abstract

Temporal logic queries on Datalog and negated Datalog programs are studied, and their relationship to Datalog queries on these programs is explored. It is shown that, in general, temporal logic queries have more expressive power than Datalog queries on Datalog and negated Datalog programs. It is also shown that an *existential domain-independent* fragment of temporal logic queries has the same expressive power as Datalog queries on negated Datalog programs with inflationary semantics. This means that for finite structures this class of queries has the power of the fixpoint logic.

Key Words and Phrases: Datalog, temporal logic queries, expressive power.

1 Introduction

Traditionally, the semantics of a Datalog query on a Datalog program is associated with the computation of the fixpoint of that program and evaluation of the query on that fixpoint [Ull88]. However, as will be explained below, it is also interesting to ask questions about the *intermediate* stages of the computation of the fixpoint. One way to ask questions about sequences of intermediate stages of the computation is to use *temporal logic*. Then it becomes interesting to know how temporal logic queries on sequences generated by Datalog programs are related to Datalog queries on fixpoints of these programs.

Clearly, a Datalog query defined by predicate $Q(x_1, \ldots, x_n)$ on a Datalog or negated Datalog, Datalog$^\neg$, program can be expressed as the temporal logic query $\{x_1, \ldots, x_n \mid \diamond\ Q(x_1, \ldots, x_n)\}$ on the intermediate stages of the fixpoint computation for that program, where $\diamond$ is the temporal operator *sometimes* [MP92]. The inverse question is more interesting: can temporal logic queries on Datalog or Datalog$^\neg$ programs be simulated with regular Datalog queries on (some other) Datalog or Datalog$^\neg$ programs? It follows from simple monotonicity considerations that the answer is negative for Datalog programs. However, the answer is much more involved for the Datalog$^\neg$ programs. In fact, this question constitutes the focal point of this paper.

The problem of studying intermediate stages in the computation of the fixpoint of Datalog and Datalog$^\neg$ programs is interesting for the following reasons. First, researchers have been studying intermediate stages of the fixpoint computations before. For example, Moschovakis in his book [Mos74] has a separate chapter on the stages of an inductive definition (Chapter 2) that contains the Stage Comparison Theorem among other results on intermediate stages of the fixpoint computation. Also, Abiteboul and Vianu [AV91] and Gurevich and Shelah [GS86] extensively use intermediate stages of the fixpoint computations in the proofs of their major results.

Second, intermediate stages of fixpoint computations can be used for the specifications of temporal databases. There have been several methods proposed in the past for the specification of infinite temporal databases, such as temporal logic programs [BCW93, Tuz93], production systems [KT89, TK91], and linear repeating points [BNW91]. Furthermore, as [AV91] shows, doubly negated Datalog, Datalog$^{\neg *}$ [AV91], is very closely related to certain types of production systems and, therefore, can also be used for the specifications of temporal databases. Of course, Datalog and Datalog$^\neg$ differ from Datalog$^{\neg *}$ in that Datalog$^{\neg *}$ supports negations in the head of a rule, whereas the other two languages do not. Nevertheless, Datalog and Datalog$^\neg$ can also be used for the specification of temporal data in those applications where data is non-decreasing over time, assuming that the semantics of these programs is defined in terms of the *intermediate* stages of

fixpoint computations.

Finally, intermediate stages of the fixpoint computation become very important in those Datalog extensions that do not guarantee the existence of the fixpoint, such as doubly negated Datalog, Datalog$^{\neg *}$ [AV91]. Since some Datalog$^{\neg *}$ programs do not have a fixpoint, fixpoint queries on these programs are not well-defined. To solve this problem, we can use temporal logic queries as an extension of Datalog queries on such programs.

Temporal logic as a query language has been studied before in the context of temporal databases [TC90, BNW91, GM91, CCT]. Studying temporal logic as a query language is important because it serves as a theoretical foundation for various temporal query languages and algebras proposed in the literature [CCT]. In this paper we continue studying temporal logic as a query language but on a special type of a temporal database generated by Datalog programs.

The rest of the paper is organized as follows. In Section 2, we define some preliminary concepts, including a temporal logic query language on Datalog and Datalog$^{\neg}$ programs. In Section 3, we analyze the relative expressive power of temporal logic and Datalog queries for Datalog and Datalog$^{\neg}$ programs. We also formulate the main result of the paper that a certain subset of temporal logic queries can be simulated with Datalog queries on Datalog$^{\neg}$ programs and show how it is related to the Stage Comparison Theorem [Mos74]. In Section 4, we prove this result. Proofs of the major technical lemmas stated in Section 4 are presented in the Appendix.

2 Preliminaries

In order to study the relationship between Datalog and temporal logic queries, we first have to define the meaning of these queries on Datalog and Datalog$^{\neg}$ programs.

Datalog queries on Datalog and Datalog$^{\neg}$ programs have been extensively studied before [Ull88]. Specifically, let P be a Datalog program and let E be a set of EDB predicates. Consider the sequence of database states $D_0, D_1, \ldots, D_n, \ldots$ where $D_{i+1} = EVAL(D_i)$ and $D_0 = E$. The mapping $EVAL$ computes new facts from the facts in D_i by applying all the Datalog rules in P simultaneously[1]. The meaning of a Datalog program is associated with the least fixpoint of the mapping $EVAL$, i.e., with the first value D_i in the sequence above for which $D_i = D_{i+1}$ [Ull88]. Similarly, the meaning of a Datalog$^{\neg}$ program under *inflationary semantics* [AV91, KP91], is defined as the fixpoint of the mapping

$$D_{i+1} = D_i \cup EVAL(D_i) \tag{1}$$

[1]For precise definition of $EVAL$ see [Ull88, p. 115]

A *Datalog query* for a Datalog or Datalog$^\neg$ program P is a predicate Q appearing among the IDB predicates of P. The answer to such a query is defined in the standard way [Ull88] as the instance of predicate Q taken at the fixpoint of P.

However, when we define the semantics of a temporal logic query on a Datalog program, we cannot assume the fixpoint semantics of the program as defined above because the meaning of temporal logic formulas is defined in terms of *sequences* of predicate instances appearing in them. To accommodate this difference, we associate the meaning of a Datalog or a Datalog$^\neg$ program with the *entire sequence* $D_0, D_1, \ldots, D_n, \ldots$ of the intermediate stages in the computation of the fixpoint of the program. We will call it *sequential* semantics, and, if no confusion arises, we will also call it *inflationary* for the Datalog$^\neg$ programs because the sequence of intermediate stages is still obtained with the inflationary equation (1).

To define temporal logic queries, we consider the future fragment of predicate temporal logic [MP92, Eme90], denoted as TL, with temporal operators $\circ$ (*next*), and **until** and with time defined with natural numbers. $\circ A$ is true at time t if A is true at time $t+1$. A **until** B is true at time t if B is true at some time $t' \geq t$ and A is true for all times t'' such that $t \leq t'' < t'$. In addition, we consider derived temporal operators *possibility* ($\diamond$) and *necessity* ($\Box$) that are defined as $\diamond A = TRUE$ **until** A, and $\Box A = \neg \diamond \neg A$. Finally, another derived temporal operator A **before** B is defined as being true at time t if for every time t' such that B is true at t', there is some time t'' such that $t \leq t'' < t'$ and A is true at time t'' [Kro87].

The semantics of a temporal logic formula is defined with a *temporal structure* K [Kro87], which specifies instances of all its predicates at *all* the times in the future. In particular, K_t defines instances of all the predicates appearing in the formula at time t. We make an assumption, natural in the database context, that domains of predicates *do not change over time*.

From the database perspective, a temporal structure can be viewed as an infinite sequence of database states, i.e. $D_0, D_1, D_2, \ldots$. In this paper, we assume that the temporal structure is defined by a Datalog or Datalog$^\neg$ program that generates a sequence of database states in the way described above[2]. We denote the temporal structure generated by a Datalog or Datalog$^\neg$ program P (for a set of EDBs) as K^P.

A temporal logic formula ϕ on a Datalog or Datalog$^\neg$ program P, with all the predicates in ϕ appearing in program P, defines a *query* $\{\mathbf{x} \mid \phi(\mathbf{x})\}$ on P[3]. The *answer* to this query is defined

[2]The stages of intermediate computations of Datalog and Datalog$^\neg$ programs can be associated with the discrete linear model of time. This means that one application of Datalog rules takes one time unit. Datalog literature usually calls intermediate steps of the computation *stages*, whereas the temporal logic literature calls them *time instances*. We will not adhere to any specific terminology in the paper and will use the two terms interchangeably.

[3]We will sometimes simplify the notation and refer to this query as ϕ.

with a first-order (time independent) predicate

$$o^*_P(\mathbf{x}) \;=\; K^P_0(o(\mathbf{x}))$$

where K^P is the temporal structure determined by program P (K^P_0 means that K^P is evaluated at time $t = 0$). In other words, the answer to query ϕ on P is the set of tuples $\mathbf{x}$ satisfying the temporal logic formula ϕ at time 0 with semantics determined by program P. For example, if $\phi(x) \equiv A(x)$ **until** $B(x)$, then $\phi^*_P(x)$ is true if $A(x)$ is always true from time $t = 0$ until $B(x)$ becomes true.

We associate some, generally infinite, domain DOM_P with a Datalog$^\neg$ program P and assume that all the constants in P and in EDB's of P come from this domain. Furthermore, we assume that all the Datalog$^\neg$ programs are *safe* in the sense defined in [Ull88]. This means that a Datalog$^\neg$ program with the inflationary semantics cannot introduce any new symbols when rules are applied to a database and that it always has a fixpoint.

3 Temporal Logic vs. Datalog Queries

In this section, we compare the expressive power of TL and Datalog queries on Datalog and Datalog$^\neg$ programs. If $Q(\mathbf{x})$ is a Datalog query on a Datalog or Datalog$^\neg$ program, then the TL query $\{\mathbf{x} \mid \diamond Q(\mathbf{x})\}$ simulates $Q(\mathbf{x})$ for the same program. Therefore, TL queries are at least as powerful as Datalog queries on Datalog and Datalog$^\neg$ programs. Furthermore, since $\{x \mid A(x)$ **before** $B(x)\}$ is not necessarily monotone in the initial value of B at time $t = 0$ for all Datalog programs and since Datalog queries on Datalog programs are always monotone in their predicates, we have the following proposition.

Proposition 1 *TL queries have more expressive power than Datalog queries for Datalog programs.*

In the rest of the paper, we will address the question of whether or not temporal logic queries can be expressed with Datalog queries on *Datalog*$^\neg$ programs. Clearly, the answer is negative for an unrestricted class of TL queries as the following example shows.

Let $\phi(x) = \neg \diamond A(x) \vee \diamond A(x)$. Certainly, $o(x)$ is true for all values of x and for all Datalog$^\neg$ programs. Therefore, if the domain of a program is infinite then this query produces an infinite answer. However, (safe) Datalog$^\neg$ programs can produce only finite answers. This means that the TL query ϕ cannot be expressed with a Datalog query on a Datalog$^\neg$ program.

This motivates the concept of *domain-independence* which is an extension of the same concept for standard relational queries [Ull88] to temporal logic and Datalog$^\neg$ programs. Let $dom_{P,\phi}$ be

the set of all the constants appearing in a TL query ϕ, in all the rules of a Datalog$^\neg$ program P, in all the EDB predicates of P, and the constants in all the future instances of predicates in P, i.e. constants inferred by program P[4]. Then a TL formula ϕ is *domain-independent* if for any Datalog$^\neg$ program P the predicate ϕ^*_P is the same for any domain DOM_P such that $DOM_P \supset dom_{P,\phi}$, i.e., ϕ^*_P does not depend on DOM_P.

We are ready to state the main result of the paper that says that for a certain subclass of domain-independent queries (which we will call existential queries) the following condition holds:

> for any Datalog$^\neg$ program P and a TL query ϕ on P from that class of queries, there is a Datalog$^\neg$ program P' and a Datalog query Q such that Q on P' and ϕ^*_P define the same mapping.

3.1 Relationship to the Stage Comparison Theorem

If a TL formula does not contain quantifiers then the main result follows from

1. the fact that the answer to a TL query without quantifiers can be expressed as a first-order formula in terms of the ordering predicates $S_{A<B}$ and $S_{A\leq B}$ [TK89][5];
2. the Stage Comparison Theorem [Mos74] that says that the ordering predicates $S_{A<B}$ and $S_{A\leq B}$ can be expressed as the least fixpoints of some first order formulas;
3. the fact that inflationary Datalog$^\neg$ programs have the power of inflationary fixpoints [AV91], and therefore least fixpoints [GS86].

However, if a TL formula contains quantifiers then it can be shown in the general case that the Step 1 in this argument is no longer valid. Therefore, it is not clear how the Stage Comparison Theorem can be applied to the general case when quantifiers appear in TL formulas. For this reason, we provide our own proof of the main result which is independent of the Stage Comparison Theorem.

The proof of the main result is structured as follows. First, we define a certain class of *configuration formulas* for a set of predicates appearing in a query. Second, we show that the answer to a TL query is equal to the disjunction of some set of configuration formulas. Third, we

[4]Since we consider only *safe* programs, no new constants will be added to $dom_{P,\phi}$ by applying rules from P. Therefore, the domain of a safe formula contains only constants in ϕ, in EDB predicates, and the constants of P.

[5]As will be defined below, an ordering predicate $S_{A<B}(x)$ ($S_{A\leq B}(x)$) is true if and only if x is inserted in predicate A *before* (or at the same time as) x is inserted in predicate B.

show how configuration formulas can be computed with Datalog$^\neg$ programs. In the next section we provide an example that illustrates these steps, and in Section 4 we prove the main theorem.

3.2 Example

Consider the following temporal logic formula on some Datalog$^\neg$ program P:[6]

$$\phi(x) \equiv C(x) \textbf{ until } (A(x) \wedge \neg B(x)) \tag{2}$$

We will show that there is a Datalog$^\neg$ program P' and a query Q such that Q and ϕ^*_P define the same mapping. This will be done in two parts. In the first part, we will show that ϕ^*_P can be expressed as a first-order formula over *ordering* predicates. In the second part, we will show how this first-order formula can be computed with Datalog$^\neg$ rules.

Let $t_A(x)$, $t_B(x)$, and $t_C(x)$ be the time instances when x is inserted into predicates A, B, and C respectively[7]. $t_A(x)$ and $t_B(x)$ are well defined, because under the inflationary semantics of Datalog$^\neg$, once a tuple is inserted into a predicate, it will never be removed from it. In general, $0 \leq t_A(x), t_B(x), t_C(x) \leq \infty$. $t_A(x) = \infty$ means that x is never inserted into A; similarly for t_B and for t_C.

We define *ordering predicates* $S_{A<B}(x)$, $S_{A=B}(x)$, $S_{A<C}(x)$, etc. as follows. $S_{A<B}(x)$ is true if and only if $t_A(x) < t_B(x)$. Other ordering predicates are defined similarly.

Part 1. In this part, we will show that ϕ^*_P can be expressed in first-order terms over some *ordering* predicates by evaluating operators in ϕ in the bottom-up manner. Let $\phi(x) = \phi''(x)$ **until** $\phi'(x)$, where $\phi'(x) = A(x) \wedge \neg B(x)$ and $\phi''(x) = C(x)$.

Step 1: $\phi'(x) \equiv A(x) \wedge \neg B(x)$.

Let $\tau'_1(x) = t_A(x)$, $\tau'_2(x) = t_B(x)$. By inspection, if $\tau'_1(x) < \tau'_2(x)$ then ϕ' is true on the interval $[\tau'_1(x), \tau'_2(x))$, and false outside of this interval. We will call such an interval *truth interval* because it specifies times when ϕ' is true. We will also call $\tau'_1(x)$ and $\tau'_2(x)$ *transition functions* since at these time points ϕ' changes its value.

For the reasons to be explained below, we will consider two cases: $\tau'_1(x) = 0$ and $\tau'_1(x) > 0$.

[6] Since this formula does not contain quantifiers, it can be simulated with a Datalog$^\neg$ program using the Stage Comparison Theorem as was specified in Section 3.1. However, introduction of quantifiers makes any non-trivial example unmanageable. Therefore, we selected an example without quantifiers in order to illustrate some of the major ideas used in the proof of the main theorem. The additional difficulties related to quantifiers will be addressed directly in the proof.

[7] These time instances are related to the stages of inductive definitions in [Mos74].

By inspection, if

$$\gamma_{11}(x) \equiv \tau_1'(x) < \tau_2'(x) \wedge \tau_1'(x) = 0$$

then ϕ' is true on the time interval $[0, \tau_2'(x))$ and if

$$\gamma_{12}(x) \equiv \tau_1'(x) < \tau_2'(x) \wedge \tau_1'(x) > 0$$

then ϕ' is true on $[\tau_1'(x), \tau_2'(x))$, where $\tau_1'(x) > 0$. Also, if

$$\gamma_{13}(x) \equiv \tau_1'(x) \geq \tau_2'(x)$$

then ϕ' is false for all times.

We call $\gamma_{11}, \gamma_{12}, \gamma_{13}$ *configuration formulas* for ϕ' because each formula uniquely determines the configuration of the temporal structure of ϕ': for all values of x satisfying a configuration formula, the temporal structure of $\phi'(x)$ has the same topology of its truth intervals (e.g. if x satisfies $\gamma_{12}(x)$ then $\phi(x)'$ is true on $[\tau_1'(x), \tau_2'(x))$ and false elsewhere). Note that $\gamma_{11}, \gamma_{12}, \gamma_{13}$ determine all possible configurations of truth intervals for ϕ' because they are mutually exclusive and collectively exhaustive.

Step 2: $\phi''(x) \equiv C(x)$.

Let $\tau''(x) = t_C(x)$. In this case, we have two configuration formulas $\gamma_{21}(x)$ and $\gamma_{22}(x)$. If

$$\gamma_{21}(x) \equiv \tau''(x) = 0$$

then $\phi''(x)$ is true for all times. If

$$\gamma_{22}(x) \equiv \tau''(x) > 0$$

then $\phi''(x)$ is true on the interval $[\tau''(x), \infty)$, where $\tau''(x) > 0$.

Step 3: $\phi(x) \equiv \phi''(x)$ **until** $\phi'(x)$.

To determine all possible configurations for ϕ, we have to consider pairwise combinations of configurations produced in Steps 1 and 2 (six combinations altogether).

1. $\gamma_{31}(x) \equiv \gamma_{11}(x) \wedge \gamma_{21}(x) \equiv \tau_1'(x) < \tau_2'(x) \wedge \tau_1'(x) = 0 \wedge \tau''(x) = 0$

   The temporal structure of $\phi(x)$ for the values of x satisfying $\gamma_{31}(x)$ has the configuration consisting of a single truth interval $[0, \tau_2'(x))$.

2. $\gamma_{32}(x) \equiv \gamma_{11}(x) \wedge \gamma_{22}(x)$.

   To determine the temporal structure of ϕ in this case, we have to consider two cases, i.e. when $\tau''(x) < \tau_2'(x)$ and when $\tau''(x) \geq \tau_2'(x)$. By inspection, in both cases the temporal structure has only one truth interval $[0, \tau_2'(x))$.

3. $\gamma_{33}(x) \equiv \gamma_{12}(x) \wedge \gamma_{21}(x)$.

In this case, the temporal structure for ϕ has only one truth interval $[0, \tau_2'(x))$.

4. $\gamma_{34}(x) \equiv \gamma_{12}(x) \wedge \gamma_{22}(x)$.

In this case, $\gamma_{12}(x) \wedge \gamma_{22}(x)$ is *not* a configuration formula for $\phi(x)$ because there can be different temporal structures for the values of x satisfying $\gamma_{34}(x)$. To get configuration formulas for this case, we have to split it into the following subcases:

(a) $\tau''(x) \leq \tau_1'(x)$

(b) $\tau_1'(x) < \tau''(x) \leq \tau_2'(x)$

(c) $\tau_2'(x) < \tau''(x)$

In subcase (4a), we get the configuration formula

$\gamma_{341}(x) \equiv \gamma_{34}(x) \wedge \tau''(x) \leq \tau_1'(x) \equiv \gamma_{12}(x) \wedge \gamma_{22}(x) \wedge \tau''(x) \leq \tau_1'(x)$

and the truth interval $[\tau''(x), \tau_2'(x))$.

Both subcases (4b) and (4c) have the same configuration determined by the truth interval $[\tau_1'(x), \tau_2'(x))$ and the combined configuration formula

$\gamma_{342}(x) \equiv \gamma_{12}(x) \wedge \gamma_{22}(x) \wedge \tau''(x) > \tau_1'(x)$

5. $\gamma_{35}(x) \equiv \gamma_{13}(x) \wedge \gamma_{21}(x)$.

The temporal structure for ϕ is always FALSE in this case for all the values of x satisfying $\gamma_{35}(x)$ and all the moments of time.

6. $\gamma_{36}(x) \equiv \gamma_{13}(x) \wedge \gamma_{22}(x)$.

The temporal structure for ϕ is also always FALSE in this case.

So far, we considered all possible configurations of temporal structure for ϕ and conditions (configuration formulas) that determine these configurations. Clearly, ϕ_P^* is equal to the disjunction of those configuration formulas, whose temporal structure is true at time $t = 0$. By inspection, this happens when one of the conditions $\gamma_{31}(x)$, or $\gamma_{32}(x)$, or $\gamma_{33}(x)$ is true, i.e.

$$\phi_P^*(x) = \gamma_{31}(x) \vee \gamma_{32}(x) \vee \gamma_{33}(x)$$

This expression can be simplified to

$$\phi_P^*(x) = \gamma_{11}(x) \vee \gamma_{12}(x) \wedge \gamma_{21}(x) \qquad (3)$$

If we substitute the expressions for $\gamma_{11}(x)$, $\gamma_{12}(x)$, $\gamma_{21}(x)$ in (3) as defined in Steps 1 and 2 above, then we conclude that $\phi^{\prec}_P(x)$ is true when either 1) $A(x)$ is true at time 0 and $B(x)$ is not true at that time or 2) $C(x)$ is true at 0 and $B(x)$ becomes true after $A(x)$. Notice that this observation coincides with the meaning of the *TL* query defined by (2), and this verifies the formula (3).

Part 2. It follows from the Stage Comparison Theorem [Mos74] and from the fact that Datalog$^\neg$ programs have the power of inflationary fixpoints [AV91] that ordering predicates can be computed with Datalog$^\neg$ programs. Therefore, $\phi^{\prec}_P$, as defined by (3), can be computed with a Datalog$^\neg$ program. However, in the proof of the main theorem (Lemma 8), we will show how configuration formulas can be computed directly with Datalog$^\neg$ programs. We delay the treatment of this issue until then. ∎

This example illustrates the major idea behind the proof of the main theorem that the answer to a *TL* query ϕ can be obtained by determining a finite number of configuration formulas. Furthermore, the answer to the query consists of the disjunction of those configuration formulas whose corresponding truth intervals start at time 0.

As was pointed out before, we did not consider quantifiers in this example and did not address the problems associated with them. We will deal with quantifiers directly in the proof of the main theorem.

4 Main Theorem

In this section, we prove the main result of this paper that existential domain-independent *TL* queries on Datalog$^\neg$ programs can be simulated with Datalog queries. In order to prove this result, we first provide some preliminary definitions.

A *TL* formula can have several references to the same predicate. Each such reference will be called an *occurrence* of a predicate in a formula. Two occurrences of the same predicate are *identical* if they have the same list of variables, e.g. $P(x_1, \ldots, x_k)$; otherwise, they are *distinct*. Let $A_1, \ldots, A_n$ be all the distinct occurrences of all the predicates from query ϕ. For example, the formula $B(x)$ **until** $(B(y) \wedge C(x, x))$ gives 3 predicates, which we could write as $A_1(x)$, $A_2(y)$, $A_3(x, x)$.

Let $\mathbf{x}$ be a sequence of some length m listing in some order all the variables of ϕ. For each predicate occurrence A_i, $\mathbf{x}_i$ will denote the actual variables of A_i in the order they appear. Thus in the above example, $m = 2$, $\mathbf{x} = (x, y)$, $\mathbf{x}_1 = (x)$, $\mathbf{x}_2 = (y)$, $\mathbf{x}_3 = (x, x)$. As defined in Section 3.2, let $t_{A_i}(\mathbf{x}_i)$ be the time instance when $\mathbf{x}_i$ is inserted into A_i. As before, $t_{A_i}(\mathbf{x}_i) = \infty$ means that $\mathbf{x}_i$

is never inserted into A_i.

Let $A_i(\mathbf{x}_i)$ and $A_j(\mathbf{x}_j)$ be two predicate occurrences in ϕ, and $\mathbf{x}$ be the sequence of variables consisting of variables from either $\mathbf{x}_i$ or $\mathbf{x}_j$. An *ordering predicate* $S_{A_i<A_j}(\mathbf{x})$ on a Datalog$^\neg$ program is true if and only if $t_{A_i}(\mathbf{x}_i) < t_{A_j}(\mathbf{x}_j)$. In other words, it is true if and only if $\mathbf{x}_i$ is inserted in A_i before $\mathbf{x}_j$ is inserted in A_j. Similarly, we define other types of ordering predicates: $S_{0<A_j}(\mathbf{x})$ is true if and only if $0 < t_{A_j}(\mathbf{x}_j)$, $S_{A_i=A_j}(\mathbf{x})$ is true if and only if $t_{A_i}(\mathbf{x}_i) = t_{A_j}(\mathbf{x}_j)$, and $S_{A_j<\infty}(\mathbf{x})$ if and only if $t_{A_j}(\mathbf{x}_j) < \infty$. As was pointed out before, the ordering predicates are related to the ordering relations defined in the Stage Comparison Theorem [Mos74]. In fact they are modified versions of ordering relations.

Let $\Omega_{A_1,\ldots,A_n}$ be the class of all the first-order formulas over all the distinct occurrences $A_1, A_2, \ldots, A_n$ of predicates from ϕ and over all the ordering predicates based on $A_1, A_2, \ldots, A_n$. For example, $(\forall y)(C(x,y) \Rightarrow S_{A<B}(x,y))$ is a formula from $\Omega_{A,B,C}$.

We next define a configuration for a *TL* formula. A *configuration* $C_\phi(\mathbf{x})$ for a *TL* formula ϕ is a triple $(\gamma(\mathbf{x}), \Gamma(\mathbf{x}), \delta)$, where $\gamma(\mathbf{x})$ is a first-order formula from $\Omega_{A_1,\ldots,A_n}$ called *configuration formula*, $\Gamma(\mathbf{x}) = \{\tau_0(\mathbf{x}), \tau_1(\mathbf{x}), \ldots, \tau_n(\mathbf{x}), \tau_{n+1}(\mathbf{x})\}$ is a set of *transition functions* such that $0 = \tau_0(\mathbf{x}) < \tau_1(\mathbf{x}) < \ldots < \tau_n(\mathbf{x}) < \tau_{n+1}(\mathbf{x}) = \infty$ for all values of $\mathbf{x}$ satisfying the configuration formula $\gamma(\mathbf{x})$, and δ is a boolean variable, called a *configuration type indicator* (or simply a configuration indicator). A configuration for a *TL* formula $\phi(\mathbf{x})$ has a property that the temporal structure of $\phi(\mathbf{x})$ does not change for the values of $\mathbf{x}$ satisfying the configuration formula $\gamma(\mathbf{x})$, and is determined by its transition functions. More specifically, ϕ is constant (either true or false) on an interval $[\tau_i(\mathbf{x}), \tau_{i+1}(\mathbf{x}))$, $i = 0, \ldots, n$, for all values of $\mathbf{x}$ such that $\gamma(\mathbf{x})$ is true and changes truth values across adjacent intervals, i.e. if ϕ is true on $[\tau_i(\mathbf{x}), \tau_{i+1}(\mathbf{x}))$ then it is false on $[\tau_{i+1}(\mathbf{x}), \tau_{i+2}(\mathbf{x}))$, and if ϕ is false on the first interval, it is true on the second one. In addition, δ determines the truth value of the first interval $[\tau_0(\mathbf{x}), \tau_1(\mathbf{x}))$ (and, therefore, the other intervals as well): if δ is true then ϕ is also true on the first interval, and if δ is false then ϕ is also false on the first interval. The intervals $[\tau_i(\mathbf{x}), \tau_{i+1}(\mathbf{x}))$ on which ϕ is true are called *truth intervals* of the configuration. An example of a configuration with $\delta = FALSE$ is shown in Fig. 1.

Intuitively, a configuration specifies one of the possible "topologies" of a temporal structure of a *TL* formula on a Datalog$^\neg$ program by specifying the set of transition points (transition functions) at which the formula changes values between *true* and *false*. In addition, the configuration formula specifies under what conditions this topology is valid. For instance, in Step 1 of Part 1 of the example presented in Section 3.2, if the configuration formula is $\gamma_{12}(x) \equiv \tau_1'(x) < \tau_2'(x) \wedge \tau_1'(x) > 0$ then ϕ' has the topology determined by two transition functions $\tau_1'(x)$ and $\tau_2'(x)$. The formula

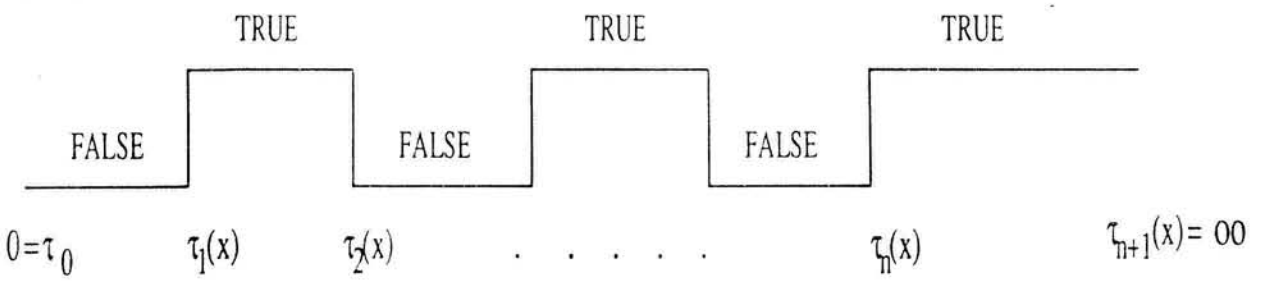

Figure 1: An example of a configuration.

$\phi'(x)$ is true for all the values of x satisfying $\gamma_{12}(x)$ and for all the moments of time t such that $\tau_1'(x) \leq t < \tau_2'(x)$ and false otherwise.

The proof of the main theorem is based on the idea that, if a TL formula is of a certain type (which we will call an "internally existential" formula), then the answer to the query defined by this formula is equal to a disjunction of a finite set of configuration formulas (that belong to $\Omega_{A_1,\ldots,A_n}$). This means that in order to simulate a TL query of that type, we only have to show how to simulate formulas from $\Omega_{A_1,\ldots,A_n}$ by Datalog$^\neg$ programs.

In the proof of the main theorem when we show that ϕ_P^ω belongs to $\Omega_{A_1,\ldots,A_n}$, i.e. it is "computable" by a Datalog$^\neg$ program, we have to show that transition functions in configurations of subformulas of ϕ are also "computable" in some sense. This motivates the following definition of a "computable" class of transition functions $\Phi_{A_1,\ldots,A_n}$ which is defined as follows:

1. $t_{A_i}(\mathbf{x}) \in \Phi_{A_1,\ldots,A_n}$, for $i = 1,\ldots,n$.

2. constant functions zero : $DOM_P^k - 0$ and infinity : $DOM_P^k - \infty$ belong to $\Phi_{A_1,\ldots,A_n}$ for all the values of $k = 1, 2, 3, \ldots$, where DOM_P is the underlying domain of the program P. If no confusion arises, we denote these functions as 0 and ∞.

3. if $\alpha(\mathbf{x}, y) \in \Phi_{A_1,\ldots,A_n}$ and $\gamma(\mathbf{x}, y) \in \Omega_{A_1,\ldots,A_n}$ then

$$g(\mathbf{x}) = \max_y\{\alpha(\mathbf{x}, y) \mid \gamma(\mathbf{x}, y)\} \in \Phi_{A_1,\ldots,A_n}$$

$$h(\mathbf{x}) = \min_y\{\alpha(\mathbf{x}, y) \mid \gamma(\mathbf{x}, y)\} \in \Phi_{A_1,\ldots,A_n}$$

For example, the formulas $\alpha(x, y) = \max_z\{t_A(x, y, z) \mid t_A(x, y, z) < \infty\}$ and $\beta(x) = \min_y\{\alpha(x, y) \mid \alpha(x, y) > 0\}$ are in Φ_A since, as it will be shown in Lemma 2, $\alpha(x, y) > 0$ belongs to Ω_A.

Lemma 2 *Let* $\alpha(\mathbf{x}) \in \Phi_{A_1,\ldots,A_n}$, $\beta(\mathbf{y}) \in \Phi_{A_1,\ldots,A_n}$. *Then the expressions* $0 < \alpha(\mathbf{x})$, $\alpha(\mathbf{x}) < \beta(\mathbf{y})$, $\alpha(\mathbf{x}) = \beta(\mathbf{y})$, $\alpha(\mathbf{x}) < \infty$ *belong to* $\Omega_{A_1,\ldots,A_n}$.[8]

Proof: The proof is done by induction on the number of operators in $\alpha(\mathbf{x})$ and $\beta(\mathbf{y})$ and is based on the observation that $\max_y\{\alpha(\mathbf{x},y) \mid \gamma_1(\mathbf{x},y)\} < \min_z\{\beta(\mathbf{x},z) \mid \gamma_2(\mathbf{x},z)\}$ is equivalent to $(\forall y)(\forall z)((\gamma_1(\mathbf{x},y) \wedge \gamma_2(\mathbf{x},z)) \Rightarrow (\alpha(\mathbf{x},y) < \beta(\mathbf{x},z)))$ ∎

As was mentioned before, we will show that for a certain class of *TL* formulas, temporal structures of these formulas are uniquely determined by a finite set of configurations. However, it is not true for arbitrary *TL* formulas. To see this, consider the formula $(\exists y)\phi(\mathbf{x},y)$. Assume that $\phi(\mathbf{x},y)$ has a configuration as shown in Fig. 2. For example, $\phi(\mathbf{x},y)$ could be $A(x,y) \wedge \neg B(x,y)$, the program P could be the transitive closure of predicates A and of B, and the initial instances of A and B at time $t = 0$ and values of x and y are such that $A(x,y)$ becomes true before $B(x,y)$. Then $(\exists y)\phi(\mathbf{x},y)$ can have arbitrarily many truth intervals (Fig. 3); the number of these intervals depends on $\mathbf{x}$ and can grow arbitrarily large in general. For example, assume that for $y = y_0$, $A(x,y_0)$ becomes true before $B(x,y_0)$, i.e. $t_A(x,y_0) < t_B(x,y_0)$, assume that the same is true for $y = y_1$, i.e. $t_A(x,y_1) < t_B(x,y_1)$, and that $t_B(x,y_0) < t_A(x,y_1)$. Also assume that for no y and no t, such that $t_B(x,y_0) < t < t_A(x,y_1)$, $A(x,y) \wedge \neg B(x,y)$ is true. In this case, $(\exists y)(A(x,y) \wedge \neg B(x,y))$ must be true between times $t_A(x,y_0)$ and $t_B(x,y_0)$, false between times $t_B(x,y_0)$ and $t_A(x,y_1)$, and true again between times $t_A(x,y_1)$ and $t_B(x,y_1)$. Note that there can be arbitrarily many pairs of such values of y_0 and y_1 in general, and therefore arbitrarily many truth intervals for $(\exists y)(A(x,y) \wedge \neg B(x,y))$ depending on the initial instances of predicates $A(x,y)$ and $B(x,y)$ at time $t = 0$. This means that the temporal structure of $(\exists y)\phi(\mathbf{x},y)$ cannot be determined by a finite number of configurations in general.

However, as Lemma 6 will show, if we restrict *TL* formulas to *internally existential* formulas (that will be defined below) then the temporal structure of a *TL* formula can be uniquely determined by a finite set of configurations. This motivates the following definitions.

Definition 3 *A TL formula is* normalized *if it has the form*

$$(\mathbf{Q_1}x_1)\ldots(\mathbf{Q_n}x_n)\phi(x_1,\ldots,x_n)$$

where $\mathbf{Q_i}$ *is either an existential or a universal quantifier, and* $\phi(x_1,\ldots,x_n)$ *is a TL formula such that*

[8] Note that this lemma trivially holds for the case when $\alpha(\mathbf{x}) = t_A(\mathbf{x})$ and $\beta(\mathbf{y}) = t_B(\mathbf{y})$ since $t_A(\mathbf{x}) < t_B(\mathbf{y})$ is the ordering predicate $S_{A<B}(\mathbf{x},\mathbf{y})$.

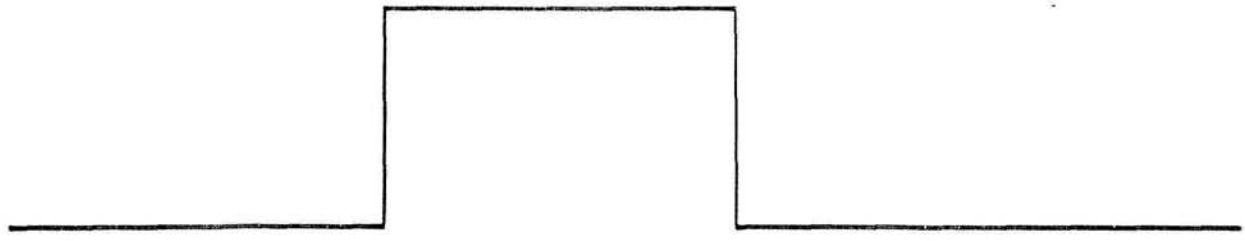

Figure 2: A configuration for $\phi(\mathbf{x}, y)$.

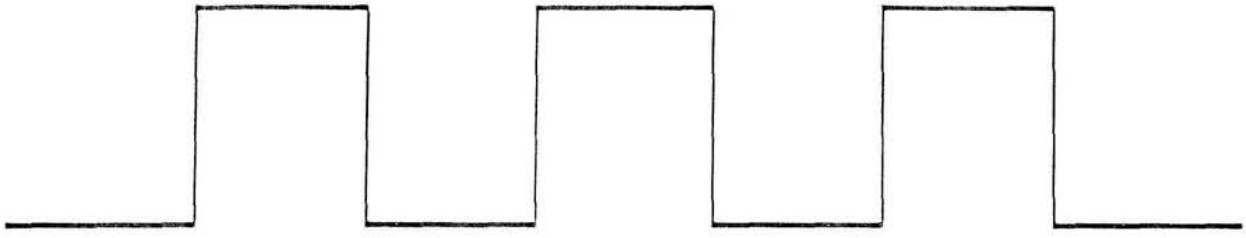

Figure 3: A configuration for $(\exists y)\phi(\mathbf{x}, y)$.

- *the existential quantifier cannot appear in a sub-expression* α **until** $(\exists x)\beta(x)$
- *the universal quantifier cannot appear in a sub-expression* $(\forall x)\alpha(x)$ **until** β
- *a quantifier cannot appear inside a first-order logic subexpression, i.e. a subexpression* $(\mathbf{Q}x)\phi(x)$ *can only appear either at the "outermost" level in the formula or inside a temporal operator.*

The following lemma says that any *TL* formula can be normalized.

Lemma 4 *For any TL formula, there exists an equivalent normalized TL formula.*

Proof: A *TL* formula can be normalized by "pulling" quantifiers out of the inner scopes of the formula using the classical equivalences of first-order logic and the following equivalences of temporal logic:

$$\alpha \textbf{ until } (\exists x)\beta(x) \equiv (\exists x)(\alpha \textbf{ until } \beta(x))$$

and

$$((\forall x)\alpha(x)) \textbf{ until } \beta \equiv (\forall x)(\alpha(x) \textbf{ until } \beta)$$

∎

Definition 5 *A TL formula is* existential, *if it has an equivalent* normalized *formula with only existential quantifiers. Furthermore, a TL formula is* internally existential, *if it is existential and its normalized form does not have quantifiers in front of the formula, i.e. if $n = 0$ in Definition 3.*

After we introduced all the necessary concepts, we are ready to state the main lemma of this paper. Intuitively, it says that the temporal structure of an internally existential *TL* formula on a Datalog$^\neg$ program is uniquely determined by a *finite* set of mutually exclusive, collectively exhaustive configurations. Furthermore, each configuration is "computable" in the sense that its configuration formula belongs to $\Omega_{A_1,\ldots,A_n}$ and its transition functions to $\Phi_{A_1,\ldots,A_n}$.

Lemma 6 *For all internally existential formulas $\phi(\mathbf{x})$ from TL there is a finite set of configurations $\mathcal{C}^i_\phi(\mathbf{x}) = (\gamma_i(\mathbf{x}), \Gamma_i(\mathbf{x}), \delta_i)$.*[9] *$i = 1, \ldots, m$, such that*

1. *if $\tau_{ij}(\mathbf{x})$ is a transition function in $\Gamma_i(\mathbf{x})$ then $\tau_{ij}(\mathbf{x}) \in \Phi_{A_1,\ldots,A_n}$;*
2. *$\gamma_i(\mathbf{x}) \wedge \gamma_j(\mathbf{x}) = FALSE$ for $i, j = 1, \ldots, m$: $i \neq j$;*
3. *$\bigvee_{i=1}^m \gamma_i(\mathbf{x}) = TRUE$;*

The proof of this lemma is presented in the Appendix. The major difficulty in the proof of this lemma is associated with handling quantifiers.

The next lemma is one of the two major lemmas to be used in the proof of the main theorem of the paper.

Lemma 7 *Let ϕ be an internally existential formula from TL. Let I_ϕ be the set of indexes i such that γ_i is a configuration formula for ϕ, as defined in Lemma 6, and the initial truth interval for γ_i starts at $t = 0$, i.e. its left endpoint is 0. Then for all* x

$$\phi^*_P(\mathbf{x}) = \bigvee_{i \in I_\phi} \gamma_i(\mathbf{x})$$

Proof: Follows from Lemma 6 and from the fact that $\gamma_1, \ldots, \gamma_m$, as defined in Lemma 6, comprise a collectively exhaustive set of formulas. ∎

Since configuration formulas are special cases of *TL* formulas without the temporal component, the concept of domain-independence is also applicable to them. The next lemma says, among other things, that domain-independent configuration formulas can be computed with Datalog$^\neg$ rules. As

[9] Note that $\gamma_i(\mathbf{x}) \in \Omega_{A_1,\ldots,A_n}$ for $i = 1, \ldots, m$ by the definition of the configuration formula.

was stated before, this result follows from the Stage Comparison Theorem [Mos74] and from the fact that Datalog$^\neg$ programs have the power of inflationary fixpoint queries [AV91]. However, we provide a constructive alternative proof of the same fact (by presenting Datalog$^\neg$ programs that simulate formulas from $\Omega_{A_1,\ldots,A_n}$) that is independent of the Stage Comparison Theorem.

Lemma 8 *Any domain-independent formula ϕ from $\Omega_{A_1,\ldots,A_n}$ on a Datalog$^\neg$ program P can be computed with a Datalog$^\neg$ program containing program P as a subprogram.*

The proof of this lemma is presented in the Appendix.

We are ready to state the main theorem of the paper now. Consider the existential formula $(\exists x_1)\ldots(\exists x_k)\phi(\mathbf{x}, x_1, \ldots, x_k)$ on Datalog$^\neg$ program P. If the internally existential formula $\phi(\mathbf{x}, x_1, \ldots, x_k)$ on program P can be simulated by query $Q'(\mathbf{x}, x_1, \ldots, x_k)$ on a Datalog$^\neg$ program P' then the existential formula $(\exists x_1)\ldots(\exists x_k)\phi(\mathbf{x}, x_1, \ldots, x_k)$ can be simulated by query $Q(\mathbf{x})$ on the Datalog$^\neg$ program obtained from P' by adding the rule $Q(\mathbf{x}) \leftarrow Q'(\mathbf{x}, x_1, \ldots, x_k)$ to P'. Then the main theorem of this paper immediately follows from this observation, from Lemmas 7, 8, and from the fact that γ_i in Lemma 7 belongs to $\Omega_{A_1,\ldots,A_n}$ for all values of $i = 1, \ldots, m$.

Theorem 9 *For any Datalog$^\neg$ program P and any existential domain independent TL query ϕ, there exists a Datalog$^\neg$ program P' and a Datalog query Q on P' such that ϕ^*_P and Q define the same mapping. Furthermore, P' contains P as a subprogram.*

5 Conclusions

In this paper, we study the expressive power of temporal logic queries on Datalog and inflationary Datalog$^\neg$ programs. The semantics of these programs is associated with the entire sequences of intermediate stages in the computation of the fixpoint rather than with the fixpoint itself. We compare the expressive power of temporal logic queries with Datalog queries on Datalog and Datalog$^\neg$ programs. We show that temporal logic queries have more expressive power than Datalog queries on Datalog programs. We also consider the existential domain-independent fragment of temporal logic. This fragment is interesting because existential *TL* queries can have only a finite set of possible configurations of its temporal structures. We show that existential domain-independent temporal logic queries have the same expressive power as Datalog queries on Datalog$^\neg$ programs. This means that on finite structures, existential temporal logic queries *on Datalog$^\neg$ programs* have the power of fixpoint queries [Mos74] since inflationary Datalog$^\neg$ programs have that power [AV91].

However, the question whether non-existential domain independent temporal logic queries can be simulated with Datalog queries on Datalog$^\neg$ programs remains open.

Acknowledgments

The author wishes to thank Zvi Kedem for many insightful discussions of the results presented in this paper and extensive comments on the previous drafts of the paper. The author also wishes to thank Gabi Kuper and Jim Clifford for useful comments on the earlier drafts of this paper.

References

[AV91] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*. 43:62–124. 1991.

[BCW93] M. Baudinet, J. Chomicki, and P. Wolper. Temporal deductive databases. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases*. Benjamin/Cummings, 1993. Forthcoming.

[BNW91] M. Baudinet, M. Niezette, and P. Wolper. On the representation of infinite temporal data and queries. In *Proceedings of PODS Symposium*, pages 280–290, 1991.

[CCT] J. Clifford, A. Croker, and A. Tuzhilin. On completeness of historical query languages. *TODS*. Forthcoming.

[Eme90] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, chapter 16, pages 996–1072. Elsevier Science Publishers, 1990.

[GM91] D. Gabbay and P. McBrien. Temporal logic and historical databases. In *International Conference on Very Large Databases*, 1991.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*. 32:265–280, 1986.

[KP91] P. G. Kolaitis and C. H. Papadimitriou. Why not negation by fixpoint? *Journal of Computer and System Sciences*. 43:125–144, 1991.

[Kro87] F. Kroger. *Temporal Logic of Programs*. Springer-Verlag, 1987. EATCS Monographs on Theoretical Computer Science.

[KT89] Z. M. Kedem and A. Tuzhilin. Relational database behavior: Utilizing relational discrete event systems and models. In *Proceedings of PODS Symposium*, pages 336–346, 1989.

[Mos74] Y. Moschovakis. *Elementary Induction on Abstract Structures*. North Holland, 1974.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[TC90] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal relational completeness. In *International Conference on Very Large Databases*, pages 13–23, 1990.

[TK89] A. Tuzhilin and Z. M. Kedem. Using temporal logic and datalog to query databases evolving in time. Technical Report 484, New York University, 1989.

[TK91] A. Tuzhilin and Z. M. Kedem. Modeling dynamics of databases with relational discrete event systems and models. Working Paper IS-91-5, Stern School of Business, NYU, 1991.

[Tuz93] A. Tuzhilin. Applications of temporal databases to knowledge-based simulations. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases*. Benjamin Cummings, 1993.

[Ull88] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

Appendix: Proofs of Lemmas 6 and 8

Proof of Lemma 6

As the first step of the proof, we replace ϕ with an equivalent formula that does not contain *next* operators as follows. Since *next* operator ($\circ$) commutes with $\exists$, $\wedge$, $\neg$, and **until** operators, it can be "pushed inside" ϕ. For example, the formula

$$\phi(x,y) \equiv \circ(A(x,y) \textbf{ until } (\circ A(y,y) \wedge \neg B(x,y)))$$

is equivalent to

$$\phi'(x,y) \equiv \circ A(x,y) \textbf{ until } (\circ^2 A(y,y) \wedge \neg \circ B(x,y)))$$

After that, we remove all the *next* operators $\circ^k A$ from ϕ by replacing them with the equivalent expressions obtained by computing the value of A at the k-th iteration of program P (this can be done because each application of rules in program P corresponds to one time unit). More precisely, if P has m rules containing A in their heads, i.e. $A \leftarrow p_1, \ldots, A \leftarrow p_m$, then $\circ^k A$ can be replaced by $\circ^{k-1}(p_1 \vee \ldots \vee p_m)$. The process of eliminating the *next* operator $\circ$ can be continued inductively after this formula is normalized. Clearly, the resulting formula is also internally existential.

As an example, consider formula ϕ' on the Datalog$^\neg$ program

$$A(x,y) \leftarrow A(x,y) \wedge \neg B(x,y) \tag{4}$$

$$B(x,y) \leftarrow A(x,u) \wedge A(u,z) \wedge B(z,y) \tag{5}$$

We replace the first component of ϕ', $\circ A(x,y)$, with $\phi'_1(x,y) = A(x,y) \wedge \neg B(x,y)$. We also replace $\circ^2 A(y,y)$ with $\phi'_2(y) = \circ A(y,y) \wedge \neg \circ B(y,y) = (A(y,y) \wedge \neg B(y,y)) \wedge \neg(A(y,u) \wedge A(u,z) \wedge B(z,y))$. Finally, we replace the third component of ϕ', $\circ B(x,y)$, with $\phi'_3(x,y) = A(x,u) \wedge A(u,z) \wedge B(z,y)$. Putting the components together, we replace the formula ϕ' with the new formula $\phi'_1(x,y)$ **until** $(\phi'_2(y) \wedge \neg \phi'_3(x,y)$ that contains no *next* operators.

Let ϕ be the new internally existential *TL* formula without any *next* operators in it. Since ϕ is an internally existential formula, existential quantifiers appear in it only in expressions $(\exists y)\alpha(\mathbf{x},y)$ **until** $\beta(\mathbf{x})$, i.e. only on the left-hand side of the **until** operator (we denote this combination of $\exists$ and **until** as $\exists$/**until**). Therefore, we prove the lemma by induction on the number of operators in ϕ, which are $\neg$, $\wedge$, **until**, and $\exists$/**until** operators. At each inductive step, we maintain a finite set of configurations for ϕ satisfying conditions of the lemma and show how each of the operators generates a new finite set of configurations satisfying the same conditions.

Base case. If $\phi(\mathbf{x}) = A(\mathbf{x})$ then there are two configurations: $\mathcal{C}_1(\mathbf{x}) = (S_{0=A}(\mathbf{x}), \{0, \infty\}, TRUE)$ and $\mathcal{C}_2(\mathbf{x}) = (S_{0<A}(\mathbf{x}), \{0, t_A(\mathbf{x}), \infty\}, FALSE)$.

Negation. If $\phi(\mathbf{x}) = \neg\phi'(\mathbf{x})$ and $\phi'(\mathbf{x})$ has a set of configurations $(\gamma_i(\mathbf{x}), \Gamma_i(\mathbf{x}), \delta_i)$ for $i = 1, \ldots, k$ then $\phi(\mathbf{x})$ has the set of configurations $(\gamma_i(\mathbf{x}), \Gamma_i(\mathbf{x}), \neg\delta_i)$.

Temporal operator **until**. Let $\phi(\mathbf{x}) = \phi'(\mathbf{x})$ **until** $\phi''(\mathbf{x})$. Assume that $\mathcal{C}_{\phi'}(\mathbf{x}) = (\gamma'(\mathbf{x}), \{\tau'_0(\mathbf{x}), \tau'_1(\mathbf{x}), \ldots, \tau'_{n'}(\mathbf{x})\}, \delta')$ be *some* configuration for $\phi'(\mathbf{x})$ and $\mathcal{C}_{\phi''}(\mathbf{x}) = (\gamma''(\mathbf{x}), \{\tau''_0(\mathbf{x}), \tau''_1(\mathbf{x}), \ldots, \tau''_{n''}(\mathbf{x})\}, \delta'')$ be *some* configuration for $\phi''(\mathbf{x})$.

As was shown in the example in Section 3.2 (Step 3.4), $\gamma'(\mathbf{x}) \wedge \gamma''(\mathbf{x})$ does not define a configuration formula for ϕ because it is not clear what the transition functions for this configuration formula are. A configuration formula for ϕ is obtained from $\gamma'(\mathbf{x}) \wedge \gamma''(\mathbf{x})$ by conjuncting it with the condition determining the relative ordering of the transition functions for ϕ' and ϕ''. Only in this

case we can guarantee that ϕ has the same set of truth intervals for all the values of $\mathbf{x}$ satisfying the configuration formula.

To make this argument formal, we define a *merging* $\mathcal{M}(\mathbf{x}) = \{\tau_0(\mathbf{x}), \tau_1(\mathbf{x}), \ldots, \tau_n(\mathbf{x})\}$ of two sets of transition functions $\{\tau'_0(\mathbf{x}), \tau'_1(\mathbf{x}), \ldots, \tau'_{n_1}(\mathbf{x})\}$ and $\{\tau''_0(\mathbf{x}), \tau''_1(\mathbf{x}), \ldots, \tau''_{n_2}(\mathbf{x})\}$, where n is some number in the range of $[max\{n_1, n_2\}, n_1 + n_2]$, as follows. Each element in $\mathcal{M}(\mathbf{x})$ is either $\tau'_i(\mathbf{x})$, or $\tau''_j(\mathbf{x})$, or both (in case $\tau'_i(\mathbf{x}) = \tau''_j(\mathbf{x})$). Furthermore, the relative ordering of transition functions $\Gamma'(\mathbf{x}) = \{\tau'_i(\mathbf{x})\}_{i=1,n_1}$ and $\Gamma''(\mathbf{x}) = \{\tau''_j(\mathbf{x})\}_{j=1,n_2}$ is preserved in the merging, i.e. if $\tau'_p(\mathbf{x}) < \tau'_q(\mathbf{x})$ then their corresponding counterparts $\tau_{p'}(\mathbf{x})$ and $\tau_{q'}(\mathbf{x})$ in the merging $\mathcal{M}(\mathbf{x})$ satisfy the condition $\tau_{p'}(\mathbf{x}) < \tau_{q'}(\mathbf{x})$ (the similar condition also holds for the transition functions in $\Gamma''(\mathbf{x})$). Note that we can have many mergings of two sets of transition functions depending on the number n and on a specific embedding of the two sets in the merging.

Two configurations $\mathcal{C}_{\phi'}$, $\mathcal{C}_{\phi''}$ and a merging $\mathcal{M}_k(\mathbf{x}) = \{\tau_{k0}(\mathbf{x}), \tau_{k1}(\mathbf{x}), \ldots, \tau_{kn}(\mathbf{x})\}$ of their transition functions determine a configuration for ϕ because ϕ has the same temporal structure (set of truth intervals) for all the tuples $\mathbf{x}$ satisfying the configuration formula

$$\tau'_i(\mathbf{x}) \wedge \tau''_j(\mathbf{x}) \wedge \bigwedge_{l=1}^{n} (\tau_{kl-1}(\mathbf{x}) < \tau_{kl}(\mathbf{x}))$$

The set of transition functions for this configuration is obtained from the transition functions in $\mathcal{M}_k(\mathbf{x})$ by removing those functions $\tau_{kl}(\mathbf{x})$ from it for which ϕ has the same value (either TRUE or FALSE) on the adjacent intervals $[\tau_{kl-1}(\mathbf{x}), \tau_{kl}(\mathbf{x}))$ and $[\tau_{kl}(\mathbf{x}), \tau_{kl+1}(\mathbf{x}))$. Finally, the configuration type indicator δ_k is equal to the value of ϕ on the first interval $[\tau_{k0}(\mathbf{x}), \tau_{k1}(\mathbf{x}))$.

Conjunction. Let $\phi(\mathbf{x}) = \phi'(\mathbf{x}) \wedge \phi''(\mathbf{x})$. The configurations for ϕ are produced in a very similar manner as the configurations for the **until** operator, and therefore we omit the proof.

Until *Combined with Existential Quantifier* ($\exists$/**until**).

Let

$$\phi(\mathbf{x}) = ((\exists y)\phi'(\mathbf{x}, y)) \textbf{ until } \phi''(\mathbf{x}) \tag{6}$$

We considered $\exists$/**until** combination and not just an existential quantifier because, as we mentioned in Section 4, the lemma does not hold for the existential quantifier: $(\exists y)\phi'(\mathbf{x}, y)$ can have an unbounded number of configurations depending on the particular value of $\mathbf{x}$.

However, if an existential quantifier appears in the left operand of the **until** operator, as in formula (6), then we claim that the number of transition functions in any configuration for $\phi(\mathbf{x})$ based on some configurations for $\phi'(\mathbf{x}, y)$ and $\phi''(\mathbf{x})$ is *not greater* than the number of transition functions in the corresponding configuration for $\phi''(\mathbf{x})$. The proof of the claim follows from the

observation that each truth interval in ϕ contains a truth interval of ϕ''. To see this, let $[\tau_1'(\mathbf{x}), \tau_2'(\mathbf{x}))$ and $[\tau_1''(\mathbf{x}), \tau_2''(\mathbf{x}))$ be truth intervals in some configurations for formulas $(\exists y)\phi'(\mathbf{x}, y)$ and $\phi''(\mathbf{x})$ respectively. Then if $[\tau_1'(\mathbf{x}), \tau_2'(\mathbf{x}))$ does not intersect any truth interval for $\phi''(\mathbf{x})$ then transition functions $\tau_1'(\mathbf{x}), \tau_2'(\mathbf{x})$ do not appear in any configuration for $\phi(\mathbf{x})$ (this follows from the definition of **until**). If $\tau_1'(\mathbf{x}) < \tau_1''(\mathbf{x}) < \tau_2'(\mathbf{x})$ then ϕ has a truth interval $[\tau_1'(\mathbf{x}), \alpha)$, where $\tau_2''(\mathbf{x}) \leq \alpha$. Finally, if $\tau_1''(\mathbf{x}) < \tau_1'(\mathbf{x}) < \tau_2''(\mathbf{x})$ then ϕ has a truth interval $[\tau_1''(\mathbf{x}), \alpha)$, where $\tau_2''(\mathbf{x}) \leq \alpha$.

We next show that the bounded number of transition functions in any configuration of ϕ obtained from configurations for $\phi'(\mathbf{x}, y)$ and $\phi''(\mathbf{x})$ is defined with a finite number of configurations. Furthermore, we explicitly determine these configurations and show that they satisfy conditions of the lemma.

To determine the set of configurations for $\phi(\mathbf{x})$, we introduce some preliminary concepts. Let $truth\_int(\mathcal{C}_\psi(\mathbf{x}), i)$ be a function determining whether $[\tau_i(\mathbf{x}), \tau_{i+1}(\mathbf{x}))$ is a truth interval for the configuration $\mathcal{C}_\psi(\mathbf{x}) = (\gamma(\mathbf{x}), \{\tau_0(\mathbf{x}), \tau_1(\mathbf{x}), \ldots, \tau_n(\mathbf{x})\}, \delta)$, i.e. whether $\psi(\mathbf{x})$ is true on this interval for the values of $\mathbf{x}$ satisfying $\gamma(\mathbf{x})$. This function is TRUE if i is even for $\delta = TRUE$ or i is odd for $\delta = FALSE$; otherwise, it is FALSE.

Let ϕ' from (6) have m configurations $\mathcal{C}^k_{\phi'}(\mathbf{x}, y) = (\gamma_k(\mathbf{x}, y), \{\tau_{k0}(\mathbf{x}, y), \tau_{k1}(\mathbf{x}, y), \ldots, \tau_{kn_k}(\mathbf{x}, y)\}, \delta_k)$, $k = 1, \ldots, m$. Then define a "point cover" function for a formula ϕ', $pnt\_cover_{\phi'}(\mathbf{x}, t)$, to be true if there exists y and a configuration $\mathcal{C}^k_{\phi'}(\mathbf{x}, y)$ such that one of its truth intervals $[\tau_{kj}(\mathbf{x}, y), \tau_{kj+1}(\mathbf{x}, y))$ covers the point t, i.e. $\tau_{kj}(\mathbf{x}, y) \leq t < \tau_{kj+1}(\mathbf{x}, y)$. To proceed further with the proof, we need the following lemma:

Lemma 10 *If $\alpha(\mathbf{x}) \in \Phi_{A_1, \ldots, A_n}$ then* pnt_cover$_{\phi'}(\mathbf{x}, \alpha(\mathbf{x}))$ *can be expressed as a formula from $\Omega_{A_1, \ldots, A_n}$.*

Proof: The proof is based on the observation that $pnt\_cover_{\phi'}(\mathbf{x}, t)$ is equivalent to

$$(\exists y) \bigvee_{k=1}^{m} \bigvee_{truth\_int(\mathcal{C}^k_{\phi'}(\mathbf{x}, y), i)} (\gamma_k(\mathbf{x}, y) \wedge \tau_{ki}(\mathbf{x}, y) \leq t < \tau_{ki+1}(\mathbf{x}, y))$$

Then the result immediately follows from Lemma 2. ∎

Let $int\_cover_{\phi'}(\mathbf{x}, \alpha, \beta)$ be an "interval cover" function for a formula ϕ' that is true if and only if $pnt\_cover_{\phi'}(\mathbf{x}, t)$ is true for all values of t from the interval $[\alpha, \beta)$, i.e. $\alpha \leq t < \beta$. As for the point-cover function, we need the following lemma in order to proceed further with the proof:

Lemma 11 *If $\alpha(\mathbf{x})$ and $\beta(\mathbf{x})$ are in $\Phi_{A_1, \ldots, A_n}$ then* int_cover$_{\phi'}(\mathbf{x}, \alpha(\mathbf{x}), \beta(\mathbf{x}))$ *can be expressed as a formula from $\Omega_{A_1, \ldots, A_n}$.*

Proof: Let ϕ' have m configurations $\mathcal{C}^k_{\phi'}(\mathbf{x}, y) = (\gamma_k(\mathbf{x}, y), \{\tau_{k0}(\mathbf{x}, y), \tau_{k1}(\mathbf{x}, y), \ldots, \tau_{kn_k}(\mathbf{x}, y)\}, \delta_k)$. The proof is based on the observation that $int\_cover_{\phi'}(\mathbf{x}, \alpha(\mathbf{x}), \beta(\mathbf{x}))$ is true if and only if one of the conditions holds: 1) there exists some y, some configuration for that y, and some truth interval for that configuration that covers the entire interval $[\alpha(\mathbf{x}), \beta(\mathbf{x}))$; or 2) there exists some y, some configuration for that y, and some transition function $\tau(\mathbf{x}, y)$ for that configuration such that $\alpha(\mathbf{x}) \leq \tau(\mathbf{x}, y) < \beta(\mathbf{x})$, and that any transition function $\tau(\mathbf{x}, z)$ appearing between $\alpha(\mathbf{x})$ and $\beta(\mathbf{x})$ must be covered by a point-cover function, i.e. $pnt\_cover_{\phi'}(\mathbf{x}, \tau(\mathbf{x}, z))$ must be true. This observation can be formalized by stating that $int\_cover_{\phi'}(\mathbf{x}, \alpha(\mathbf{x}), \beta(\mathbf{x}))$ is equivalent to

$(\exists y) \bigvee_{k=1}^{m} \bigvee_{truth\_int(\mathcal{C}^k_{\phi'}(\mathbf{x},y),i)} (\tau_{ki}(\mathbf{x}, y) \leq \alpha(\mathbf{x}) \wedge \beta(\mathbf{x}) < \tau_{ki+1}(\mathbf{x}, y))$

$\vee$

$\bigvee_{k=1}^{m} \bigvee_{i=1}^{n_k} (\exists y)(\alpha(\mathbf{x}) \leq \tau_{ki}(\mathbf{x}, y) < \beta(\mathbf{x})) \wedge \bigwedge_{k=1}^{m} \bigwedge_{i=1}^{n_k} (\forall z)((\alpha(\mathbf{x}) \leq \tau_{ki}(\mathbf{x}, z) < \beta(\mathbf{x})) \Rightarrow$
$pnt\_cover_{\phi'}(\mathbf{x}, \tau_{ki}(\mathbf{x}, z)))$

It follows from Lemmas 10 and 2 that this formula belongs to $\Omega_{A_1, \ldots, A_n}$. ∎

Let $\mathcal{C}_{\phi''}(\mathbf{x}) = (\gamma''(\mathbf{x}), \{\tau''_0(\mathbf{x}), \tau''_1(\mathbf{x}), \ldots, \tau''_n(\mathbf{x})\}, \delta'')$ be a configuration for ϕ'' and let j be such that $[\tau''_j(\mathbf{x}), \tau''_{j+1}(\mathbf{x}))$ is a truth interval for this configuration (i.e. $truth\_int(\mathcal{C}_{\phi''}(\mathbf{x}), j)$ holds). If the interval $[\tau''_0(\mathbf{x}), \tau''_1(\mathbf{x}))$ is a truth interval then we let j range from 2 to the left endpoint of the last truth interval (i.e. if $[\tau''_{n-1}(\mathbf{x}), \tau''_n(\mathbf{x}))$ is a truth interval then j will go until $n-1$; if $[\tau''_n(\mathbf{x}), \infty)$ is a truth interval then j will go until n). If $[\tau''_1(\mathbf{x}), \tau''_2(\mathbf{x}))$ is a truth interval then we let j range from 1 to the left endpoint of the last truth interval.

Let

$$\xi_{j1}(\mathbf{x}) = pnt\_cover_{\phi'}(\mathbf{x}, \tau''_j(\mathbf{x})) \wedge \neg pnt\_cover_{\phi'}(\mathbf{x}, \tau''_{j-1}(\mathbf{x})) \quad (7)$$

$$\xi_{j2}(\mathbf{x}) = \neg pnt\_cover_{\phi'}(\mathbf{x}, \tau''_j(\mathbf{x})) \quad (8)$$

$$\xi_{j3}(\mathbf{x}) = int\_cover_{\phi'}(\mathbf{x}, \tau''_{j-1}(\mathbf{x}), \tau''_j(\mathbf{x})) \quad (9)$$

Each configuration $\mathcal{C}_{\phi''}(\mathbf{x}) = (\gamma''(\mathbf{x}), \{\tau''_0(\mathbf{x}), \tau''_1(\mathbf{x}), \ldots, \tau''_n(\mathbf{x})\}, \delta'')$ for ϕ'' defines several configurations for ϕ as follows. For each truth interval $[\tau''_j(\mathbf{x}), \tau''_{j+1}(\mathbf{x}))$ for this configuration select some $\xi_{ji_j}(\mathbf{x})$, where $i_j = 1, 2$, or 3. Consider the formula

$$\gamma_l(\mathbf{x}) = \gamma''(\mathbf{x}) \wedge \bigwedge_{truth\_int(\mathcal{C}_{\phi''}(\mathbf{x}), j)} \xi_{ji_j}(\mathbf{x}) \quad (10)$$

We claim that it is a configuration formula for ϕ, i.e., for all values of $\mathbf{x}$ satisfying (10), ϕ has the same configuration of its temporal structure. To see this, we examine the meanings of functions $\xi_{j1}(\mathbf{x}), \xi_{j2}(\mathbf{x}), \xi_{j3}(\mathbf{x})$ now.

Function $\xi_{j1}(\mathbf{x})$ defines the situation when some truth interval for some configuration for some y in $\phi'(\mathbf{x}, y)$ intersects the left endpoint of the truth interval $[\tau''_j(\mathbf{x}), \tau''_{j+1}(\mathbf{x}))$, and no truth interval of no configuration for no y in $\phi'(\mathbf{x}, y)$ intersects the right endpoint of the truth interval $[\tau''_{j-2}(\mathbf{x}), \tau''_{j-1}(\mathbf{x}))$. In this case, a configuration for $\phi(\mathbf{x})$ based on $\mathcal{C}_{\phi''}(\mathbf{x})$ has the structure as shown in Fig. 4 (it has a transition function $\tau_j(\mathbf{x})$ between $\tau''_{j-1}(\mathbf{x})$ and $\tau''_j(\mathbf{x})$ shown with the dotted line).

Function $\xi_{j2}(\mathbf{x})$ defines the situation when no truth interval for no configuration for no y in $\phi'(\mathbf{x}, y)$ intersects the left endpoint of the truth interval $[\tau''_j(\mathbf{x}), \tau''_{j+1}(\mathbf{x}))$. In this case, a configuration for $\phi(\mathbf{x})$ based on $\mathcal{C}_{\phi''}(\mathbf{x})$ has the structure as shown in Fig. 5 (it has $\tau''_j(\mathbf{x})$ as its transition function).

Function $\xi_{j3}(\mathbf{x})$ defines the situation when all the points between two truth intervals $[\tau''_{j-2}(\mathbf{x}), \tau''_{j-1}(\mathbf{x}))$ and $[\tau''_j(\mathbf{x}), \tau''_{j+1}(\mathbf{x}))$ are "covered" by some truth interval of some configuration for ϕ'. In this case, a configuration for $\phi(\mathbf{x})$ based on $\mathcal{C}_{\phi''}(\mathbf{x})$ has the structure as shown in Fig. 6 (the dotted line in this figure means that the truth interval for the configuration of ϕ based on $\mathcal{C}_{\phi''}(\mathbf{x})$ *contains* interval $[\tau''_{j-1}(\mathbf{x}), \tau''_j(\mathbf{x}))$); in other words, no transition function for $\mathcal{C}_\phi(\mathbf{x})$ lies between $\tau''_{j-1}(\mathbf{x})$ and $\tau''_j(\mathbf{x})$.

As it follows from Lemmas 10 and 11, $\xi_{j1}(\mathbf{x})$, $\xi_{j2}(\mathbf{x})$, and $\xi_{j3}(\mathbf{x})$ belong to $\Omega_{A_1,\ldots,A_n}$.

After all the preliminary concepts were introduced, we are ready to define the finite set of configurations $\mathcal{C}^l_\phi(\mathbf{x}) = (\gamma_l(\mathbf{x}), \Gamma_l(\mathbf{x}), \delta_l)$ for $\phi(\mathbf{x})$.

Each configuration $\mathcal{C}_{\phi''}(\mathbf{x}) = (\gamma''(\mathbf{x}), \{\tau''_0(\mathbf{x}), \tau''_1(\mathbf{x}), \ldots, \tau''_n(\mathbf{x})\}, \delta'')$ for ϕ'', and each choice of $i_j \in \{1, 2, 3\}$ for j taken over all truth intervals of $\mathcal{C}_{\phi''}$ determine the configuration formula (10).

Each choice of predicates $\xi_{ji_j}(\mathbf{x})$ taken over $i_j = 1$ or 2 or 3 for all values of j, such that $truth\_int(\mathcal{C}_{\phi''}(\mathbf{x}), j)$ holds, gives rise to one instance of formula (10). This means that there are no more than $\sum_m 3^{\lceil n \rceil/2}$ different configurations for ϕ for each configuration $\mathcal{C}_{\phi''}(\mathbf{x})$ for ϕ''.[10]

The configuration formula $\gamma_l(\mathbf{x})$ defines the configuration $\mathcal{C}^l_\phi(\mathbf{x})$ which has the set of transition functions $\Gamma_l(\mathbf{x})$. To finish the proof of the lemma, we have to show that these transition functions are "computable," i.e., they belong to $\Phi_{A_1,\ldots,A_n}$. The proof of this proceeds as follows. Take a configuration $\mathcal{C}_{\phi''}(\mathbf{x}) = (\gamma''(\mathbf{x}), \{\tau''_0(\mathbf{x}), \tau''_1(\mathbf{x}), \ldots, \tau''_n(\mathbf{x})\}, \delta'')$ for ϕ'' and initially add all the transition functions $\tau''_j(\mathbf{x})$ to $\Gamma_l(\mathbf{x})$. Then this set of transition functions $\Gamma_l(\mathbf{x})$ will be modified according to the following rules. For each j, such that $[\tau''_j(\mathbf{x}), \tau''_{j+1}(\mathbf{x}))$ is a truth interval for that configuration, consider the following cases depending on the value of i_j in ξ_{ji_j} in formula (10):

1. $i_j = 1$. Drop $\tau''_j(\mathbf{x})$ from $\Gamma_l(\mathbf{x})$ and add a new transition function $\tau_j(\mathbf{x})$ to

[10] We cannot say that $\sum_m 3^{\lceil n \rceil/2}$ is the exact estimate because we cannot choose all three formulas ξ for the first and the last truth intervals in certain cases. But we can safely say that the estimate is an upper bound on the number of configurations generated by $\mathcal{C}_{\phi''}(\mathbf{x})$.

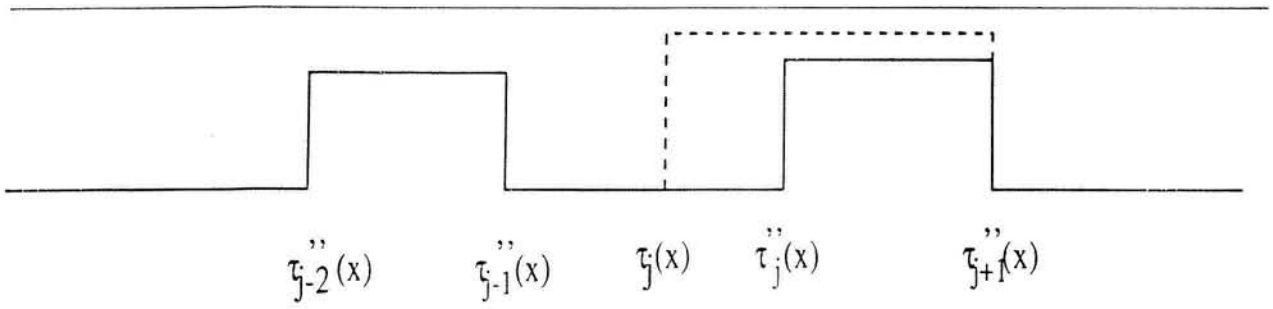

Figure 4: Configuration for $\phi(\mathbf{x})$ if Condition $\xi_{j1}(\mathbf{x})$ Holds.

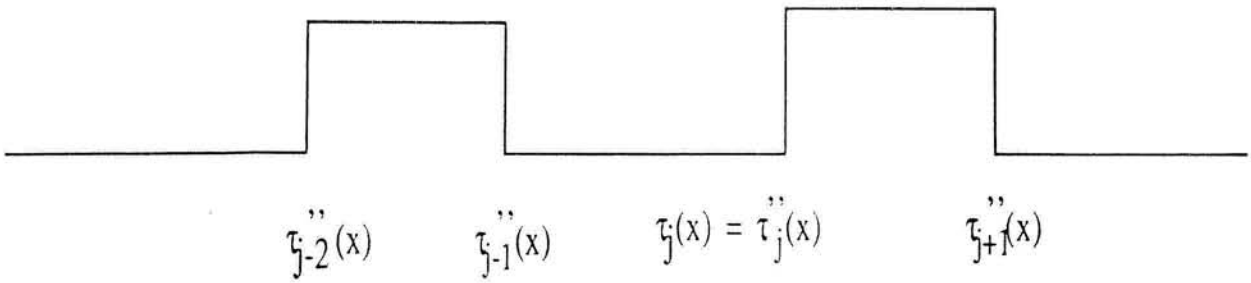

Figure 5: Configuration for $\phi(\mathbf{x})$ if Condition $\xi_{j2}(\mathbf{x})$ Holds.

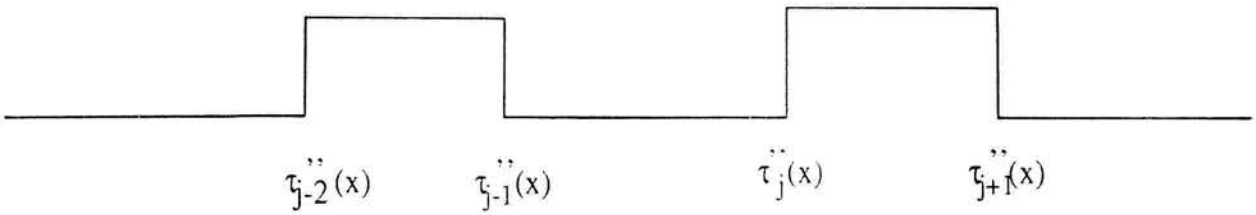

Figure 6: Configuration for $\phi(\mathbf{x})$ if Condition $\xi_{j3}(\mathbf{x})$ Holds.

it (see Fig. 4) that is defined as follows. If ϕ' has configurations $\mathcal{C}^k_{\phi'}(\mathbf{x}, y) = (\gamma'_k(\mathbf{x}, y), \{\tau'_{k0}(\mathbf{x}, y), \tau'_{k1}(\mathbf{x}, y), \ldots, \tau'_{kn_k}(\mathbf{x}, y)\}, \delta'_k)$ then $\tau_j(\mathbf{x}) = min_{ki}\{\lambda_{jki}(\mathbf{x})\}$, where

$$\lambda_{jki}(\mathbf{x}) = min_y\{\tau'_{ki}(\mathbf{x}, y) \mid int\_cover_{\phi'}(\mathbf{x}, \tau'_{ki}(\mathbf{x}, y), \tau''_j(\mathbf{x}))\}$$

As it follows from the definition of $\tau_j(\mathbf{x})$, it must be somewhere between points $\tau''_{j-1}(\mathbf{x})$ and $\tau''_j(\mathbf{x})$ in Fig. 4.

Note that $\tau_j(\mathbf{x}) \in \Phi_{A_1,\ldots,A_n}$ because $int\_cover_{\phi'}(\mathbf{x}, \tau'_{ki}(\mathbf{x}, y), \tau''_j(\mathbf{x})) \in \Omega_{A_1,\ldots,A_n}$ and because of the definition of $\Phi_{A_1,\ldots,A_n}$.

2. $i_{mj} = 2$. $\tau''_{mj}(\mathbf{x})$ remains in $\Gamma_l(\mathbf{x})$. This corresponds to the fact that there are no additional transition points added to Fig. 5.

3. $i_{mj} = 3$. Both $\tau''_{mj}(\mathbf{x})$ and $\tau''_{mj-1}(\mathbf{x})$ are removed from $\Gamma_l(\mathbf{x})$. This corresponds to the fact that the configuration in Fig. 6 "absorbs" points $\tau''_{j-1}(\mathbf{x})$ and $\tau''_j(\mathbf{x})$.

Finally, δ_l is determined for $\mathcal{C}^l_\phi(\mathbf{x})$ based on whether or not $[\tau_{l0}(\mathbf{x}), \tau_{l1}(\mathbf{x}))$ is the truth interval in that configuration.

It follows from (10) and from the fact that $\xi_{ji_j}(\mathbf{x}) \in \Omega_{A_1,\ldots,A_n}$ that $\gamma_l(\mathbf{x}) \in \Omega_{A_1,\ldots,A_n}$. Furthermore, it is easy to see that $\gamma_l(\mathbf{x})$'s form a mutually exclusive, collectively exhaustive set of formulas. Finally, it follows from the construction of transition functions that all of them belong to $\Phi_{A_1,\ldots,A_n}$. This completes the proof of the lemma.

Proof of Lemma 8

In the proof, we utilize some of the techniques developed in [AV91]. To begin with, we extend program P with rules that compute the 0-ary predicate FP_P. FP_P becomes true two steps after the time the fixpoint of P is reached. The new program P'' is obtained by adding some new rules to P as follows. For each IDB predicate $A_i(\mathbf{x}_i)$ in P, $i = 1, \ldots, n$, we introduce three auxiliary predicates $A'_i(\mathbf{x}_i)$, $A''_i(\mathbf{x}_i)$, and $prev\_unless\_last_i(\mathbf{x}_i)$ in P''. Predicates A'_i and A''_i are shifted backwards relative to A_i by one and two time units respectively. They are defined with the following rules in P'':

$A'_i(\mathbf{x}_i) \leftarrow A_i(\mathbf{x}_i)$

$A''_i(\mathbf{x}_i) \leftarrow A'_i(\mathbf{x}_i)$

Predicate $prev\_unless\_last_i$ is equal to predicate A_i'' at all the times until the *last* stage before the fixpoint of P'' is reached (i.e. all the predicates A_i'' stop changing)[11]. When this moment is detected by program P'', FP_P is set to be true. Formally, predicates $prev\_unless\_last_i$ and FP_P are obtained as follows. Replace each occurrence of A_i in every rule of P with A_i'' and replace each rule of the form $A_i(\mathbf{x}_i) \leftarrow \alpha_i(\mathbf{x}_i)$ in P with n rules $prev\_unless\_last_i(\mathbf{x}_i) \leftarrow \alpha_i(\mathbf{x}_i) \wedge A_j(\mathbf{x}_j) \wedge \neg A_j'(\mathbf{x}_j)$ for $j = 1, \ldots, n$ and add these rules to P''. Note that predicate $prev\_unless\_last_i$ is not updated to α_i by these rules only at the time when $A_j(\mathbf{x}_j) = A_j'(\mathbf{x}_j)$ for all $j = 1, \ldots, n$ (i.e. one step after the fixpoint of P is reached). In addition, add the following n rules to P'':

$$FP_P \leftarrow A_i''(\mathbf{x}_i) \wedge \neg prev\_unless\_last_i(\mathbf{x}_i)$$

for $i = 1, \ldots, n$.

It is easy to see that there is some i such that predicate $prev\_unless\_last_i$ is equal to A_i'' at all the stages before the fixpoint of P'' is reached and differs from A_i'' at the fixpoint of P'' and all the subsequent stages. Therefore, FP_P becomes true at the fixpoint time of P'' which occurs two stages after the fixpoint of P.

After we presented a program computing the fixpoint of P, we are ready to prove the lemma itself. The proof proceeds by induction on the number of operators in ϕ. Without loss of generality, we consider operators $\wedge$, $\neg$, and $\exists$. At each stage, following [AV91], we will prove by induction that there exists a Datalog$^\neg$ program P_ϕ, a predicate Q_ϕ, and a 0-ary predicate $done_\phi$ such that

1. Q_ϕ, as a Datalog query on P_ϕ, is equivalent to ϕ_P^*.
2. $done_\phi$ becomes true within a finite number of stages after the fixpoint of program P_ϕ is reached (more precisely, after 2 stages).
3. The program P_ϕ does not change values of predicates in ϕ.

If there are no operators in ϕ then there are two cases to consider. First, $\phi = A$, where A is a predicate. This case is trivial. Second, ϕ is equal to one of the ordering predicates as defined in Section 4. The result in this case follows from the Stage Comparison Theorem [Mos74, GS86] and from the fact that Datalog$^\neg$ has the power of inflationary fixpoint queries [AV91]. However, this can be proven in a different way as follows.

1. If $\phi(\mathbf{x})$ is $S_{0<A}(\mathbf{x})$ then P_ϕ consists of P and the following rules:

   $$B(\mathbf{x}) \leftarrow A(\mathbf{x}) \wedge \neg T$$

[11] We would like to credit the idea of predicate $prev\_unless\_last_i$ to [AV91].

$T \leftarrow \neg T$

$Q_\phi(\mathbf{x}) \leftarrow FP_P \wedge A(\mathbf{x}) \wedge \neg B(\mathbf{x})$

$done_\phi \leftarrow FP_P$

where the 0-ary predicate T is initially false.

2. If $\phi(\mathbf{x})$ is $S_{A_i<A_j}(\mathbf{x})$ then P_ϕ consists of P and the following rules:

   $Q_\phi(\mathbf{x}) \leftarrow A_i(\mathbf{x}_i) \wedge \neg A_j(\mathbf{x}_j)$

   $done_\phi \leftarrow FP_P$

   where FP_P is the fixpoint predicate obtained as specified before.

3. If $\phi(\mathbf{x})$ is $S_{A_i=A_j}(\mathbf{x})$ then P_ϕ consists of P and the following rules:

   $A'_i(\mathbf{x}_i) \leftarrow A_i(\mathbf{x}_i)$

   $A'_j(\mathbf{x}_j) \leftarrow A_j(\mathbf{x}_j)$

   $Q_\phi(\mathbf{x}) \leftarrow A_i(\mathbf{x}_i) \wedge \neg A'_i(\mathbf{x}_i) \wedge A_j(\mathbf{x}_j) \wedge \neg A'_j(\mathbf{x}_j)$

   $done_\phi \leftarrow FP_P$

   Note that A'_i and A'_j are trailing predicates: they are obtained from A_i and A_j respectively by shifting these predicates one time unit backwards.

4. If $\phi(\mathbf{x})$ is $S_{A_i<\infty}(\mathbf{x})$ then P_ϕ consists of P and the rule

   $Q_\phi(\mathbf{x}) \leftarrow FP_P \wedge A_i(\mathbf{x})$

   $done_\phi \leftarrow FP_P$

We consider operators now.

And: If $\phi(\mathbf{x}) \equiv \phi'(\mathbf{x}) \wedge \phi''(\mathbf{x})$ then P_ϕ consists of $P_{\phi'}$, $P_{\phi''}$ and the following rules

$Q_\phi(\mathbf{x}) \leftarrow done_{\phi'} \wedge done_{\phi''} \wedge Q_{\phi'}(\mathbf{x}) \wedge Q_{\phi''}(\mathbf{x})$

$done_\phi \leftarrow done_{\phi'} \wedge done_{\phi''}$

Existential Quantifier: If $\phi(\mathbf{x}) \equiv (\exists y)\phi'(\mathbf{x}, y)$ then P_ϕ consists of $P_{\phi'}$ and the rules:

$Q_\phi(\mathbf{x}) \leftarrow done_{\phi'} \wedge Q_{\phi'}(\mathbf{x}, y)$

$done_\phi \leftarrow done_{\phi'}$

Negation: If $\phi(\mathbf{x}) \equiv \neg\phi'(\mathbf{x})$ then P_ϕ consists of $P_{\phi'}$ and the following rules:

$dom(\mathbf{x}) \leftarrow dom(x)^{|\mathbf{x}|}$

$Q_\phi(\mathbf{x}) \leftarrow done_{\phi'} \wedge dom(\mathbf{x}) \wedge \neg Q_{\phi'}(\mathbf{x})$

$done_\phi \leftarrow done_{\phi'}$

where $|\mathbf{x}|$ in the rule above means the arity of vector $\mathbf{x}$ and $dom(x)^{|\mathbf{x}|}$ is the Cartesian product of $dom(x)$ taken $|\mathbf{x}|$ times. Predicate $dom(x)$ defines the set of all the constants appearing in EDB predicates and in the rules. It is obtained with the rules $dom(x_{ij}) \leftarrow A_i(x_{i1}, \ldots, x_{in_i})$, $j = 1, 2, \ldots, n_i$, ranging over all predicates A_i of P and also with the facts $dom(c_i)$, where c_i are the constants appearing in the rules of P.

Note that the rules computing negation are safe (because we added predicate $dom(\mathbf{x})$ to one of the rules). However, notice that Q_ϕ is equivalent to $\phi_P^\sim$ only because of domain independence of ϕ.

Also note that throughout all the inductive stages, the program P_ϕ consists of the original program P and additional rules added in order to compute ϕ. Therefore, P is a subprogram of P_ϕ.