Polynomial Time Query Processing in Temporal Deductive Databases

Jan Chomicki
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
chomicki@cs.unc.edu

Abstract

We study conditions guaranteeing polynomial time computability of queries in temporal deductive databases. We show that if for a given set of temporal rules, the period of its least models is bounded from the above by a polynomial in the database size, then also the time to process yes-no queries (as well as to compute finite representations of all query answers) can be polynomially bounded. We present a bottom-up query processing algorithm **BT** that is guaranteed to terminate in polynomial time if the periods are polynomially bounded. Polynomial periodicity is our most general criterion, however it can not be directly applied. Therefore, we exhibit two weaker criteria, defining *inflationary* and *I-periodic* sets of temporal rules. We show that it can be decided whether a set of temporal rules is inflationary. I-periodicity is undecidable (as we show), but it can be closely approximated by a syntactic notion of *multi-separability.*

1 Introduction

In [7], we proposed *temporal deductive databases* (TDDs) as a tool for storing and retrieving information about infinite temporal phenomena. A temporal deductive database (TDD) consists of a finite set of temporal rules and a finite temporal database.

TDDs are an extension of DATALOG [14]. Function symbols are used in TDDs in a limited way: there is one unary function symbol **+1**, written in postfix, and terms built with this symbol can appear only in one distinguished argument of all predicates. TDDs have two desirable properties, not shared by arbitrary logic programs [16, 11]:

1. it is decidable whether a given atom is implied by a TDD [7] (*yes-no query processing*).
2. all query answers (possibly infinitely many) can be finitely represented using relational specifications [6] (*all-answers query processing*).

However, due to computational complexity considerations, TDDs are impractical in their full generality. Yes-no query processing is PSPACE-data-complete [12, 7, 5]. Relational specifications can also be computed in PSPACE [5] and their size may be exponential in the size of the database [6].

In this paper, we make a step towards making TDDs feasible. We seek *tractability*: restrictions on temporal rules that will guarantee a polynomial upper bound on yes-no and all-answers query processing. Most of the restrictions that we present here are *semantic* and express properties of least (Herbrand) models of sets of temporal rules. Others are *syntactic* and prescribe the form of sets of temporal rules.

Least models of TDDs have a repetitive, periodic structure [7]. We show that if for a given set of temporal rules, the period of its least models is bounded from the above by a polynomial in the database size, then also the time to process yes-no queries (as well as to compute relational specifications) can be polynomially bounded. We present a bottom-up query processing algorithm **BT** that is guaranteed to terminate in polynomial time if the periods are polynomially bounded. Polynomial periodicity is our most general criterion, however it can not be directly applied. Therefore, we exhibit two weaker criteria, defining *inflationary* and *I-periodic* sets of temporal rules.

Intuitively, an inflationary set of temporal rules ex-

0

presses an infinite phenomenon with the following property: if a fact P(**t0**,$\bar{a}$) is true, then also P(**t**,$\bar{a}$) is true for all **t**>**t0**. We show that it can be decided whether a set of temporal rules is inflationary.

A set of rules is *I-periodic* if there is a common, database-independent period for all its least models. This property is undecidable (as we show). It can, however, be approximated by a syntactic notion of *multi-separability* which prohibits mutual recursion and restricts the syntax of recursive rules. This approximation is quite close, because for every I-periodic set of rules Z, there is a multi-separable set of rules Z_1 such that for all databases D, the least model of $Z \wedge D$ is equal to the least model of $Z_1 \wedge D_1$. The database D_1 differs very little from D and can be obtained from it. Multi-separable sets of rules are more general than *separable* sets of rules discussed in a different context in [7].

Least models of inflationary and I-periodic sets of temporal rules have polynomially-bounded periods, thus in both cases the yes-no query processing algorithm **BT** is guaranteed to terminate in polynomial time.

The paper is organized as follows. We provide two examples of tractable TDDs in section 2. In section 3, we define basic notions. We also summarize known complexity results in this area. In section 4, we explain our notion of tractability and we prove that polynomial periodicity implies tractability. We study inflationary temporal rules in section 5 and I-periodic temporal rules in section 6. We review related work in section 7 and conclude with an analysis of possible extensions of this work in section 8.

2 Examples

We give here two examples of tractable TDDs. The set of rules in the first is I-periodic, in the second - inflationary.

Example: A travel agent has obtained the following specification from an airline: "flights to ski resorts are scheduled every seventh day during off-season, every second day during the winter and every day during winter holidays". This can be specified using temporal rules as follows:

```
plane(T+7,X) :- plane(T,X),
      resort(X), offseason(T).
plane(T+2,X) :- plane(T,X),
      resort(X), winter(T).
plane(T+1,X) :- plane(T,X),
      resort(X), holiday(T).
offseason(T+365) :- offseason(T).
winter(T+365) :- winter(T).
holiday(T+365) :- holiday(T).
```

The temporal database may look as follows[1]:

```
plane(01/01/90).
offseason(03/21/90).
...
offseason(12/19/90).
winter(12/20/89).
...
winter(03/20/90).
holiday(12/25/89).
holiday(01/01/90).
```

Now to verify whether a plane leaves to Hunter on a given day **t0**, it has to be checked whether `plane(t0,''Hunter'')` is implied by the rules and the database. We might also ask about all days when a plane leaves to Hunter and this query has infinitely many answers.

The set of rules in this example is multi-separable (but not separable), and consequently I-periodic. But it is not inflationary: take a database with nonempty `plane` relation but empty `offseason`, `winter` and `holiday` relations. □

Example: Consider the following set of rules:

```
path(K,X,X) :- node(X), null(K).
path(K+1,X,Z) :- edge(X,Y), path(K,Y,Z).
path(K+1,X,Y) :- path(K,X,Y).
```

This set of rules is inflationary, because of the third rule. If the predicates `node` and `edge` represent nodes and edges of a directed graph, the predicate `path` has no tuples in the database and the predicate `null` has only one tuple `null(0)` in the database, the meaning of `path(K,X,Y)` is *"there is a path of length at most* **K** *between the nodes* `X` *and* `Y`*"*.

The above set of rules is not I-periodic, because the length of a path in an arbitrary graph can not be bounded from above. □

3 Basic notions

We assume here that the reader is familiar with the syntax and the semantics of logic programs: [16],[11, Chapter 1 and 2].

[1] The actual dates that appear in the database are really abbreviations for terms of the form (...((0+1)+1)...+1). We can imagine that those abbreviations are expanded when the database is being input. Similarly, we could provide an abbreviation for intervals and represent `winter` and `offseason` as single tuples `winter(<12/20/89,03/20/90>)` and `offseason(<03/21/90,12/19/90>)`.

3.1 Syntax

In this section, we define the class of temporal deductive databases: our postulated extension of DATALOG.

Language. We assume that the language contains infinitely many variable, function and predicate symbols. As usual, we call 0-ary function symbols *constants.* Variables and constants are (disjointly) partitioned into *temporal* and *non-temporal* ones. We assume that there is exactly one temporal constant $\mathbf{0}$. A predicate symbol is either *temporal* or *non-temporal.*

Terms. A *non-temporal term* is either a non-temporal constant (a standard database constant) or a non-temporal variable. A *temporal term* is defined inductively:

1. the temporal constant $\mathbf{0}$ is a temporal term.
2. a temporal variable is a temporal term.
3. if $\mathbf{v}$ is a temporal term, then $\mathbf{v+1}$ is a temporal term.
4. there are no other temporal terms.

Term which do not contain any variables are called *ground.* The only ground non-temporal terms are constants, while ground temporal terms may be arbitrarily deep. A non-ground temporal term contains exactly one variable and this variable is temporal. All the temporal and non-temporal terms are distinct.

Temporal terms will appear in **bold** font. We will write $\mathbf{k}$ instead of $(\cdots((\mathbf{0}\underbrace{+\mathbf{1})+\mathbf{1})\cdots+\mathbf{1})}_{k\ times}$ and $\mathbf{t+k}$ instead of $(\cdots((\mathbf{t}\underbrace{+\mathbf{1})+\mathbf{1})\cdots+\mathbf{1})}_{k\ times}$. The integer k corresponds to the temporal term $\mathbf{k}$ in a natural way.

Formulas. If P is a temporal predicate symbol and R is a non-temporal predicate symbol, $\mathbf{v}$ is a temporal term and $\bar{x}$ is a vector of non-temporal terms of appropriate arity, then P($\mathbf{v}$,$\bar{x}$) is a temporal atom and R($\bar{x}$) is a non-temporal atom. The term $\mathbf{v}$ is the temporal argument and the elements of $\bar{x}$ are the non-temporal arguments of P($\mathbf{v}$,$\bar{x}$).

A *temporal database* is a finite set of tuples: ground atoms which can be temporal or non-temporal. Temporal Horn rules are defined in the standard way, except that they have to be built from the atoms defined above. All variables are universally quantified. A *temporal deductive database* (TDD) is a finite set of temporal rules and a temporal database. A *temporal query* is a first order formula without equality built from temporal and non-temporal atoms, standard logical connectives and quantifiers [2]. We will write $Q(x_1,\ldots,x_k)$ to denote the query Q with free variables $x_1,\ldots,x_k$.

A temporal rule r is *semi-normal* if r contains at most one temporal variable and if this variable appears in r, it has to appear as the temporal argument of some literal. A temporal rule r is *normal* if it is semi-normal and non-ground temporal terms in r are of depth at most 1.

For every set of temporal rules, there is a set of equivalent semi-normal rules and another set of equivalent normal rules. Both are obtained through the introduction of additional predicates and rules, as shown in [5]. Thus, we will assume that the rules are normal throughout this paper, except in section 6 where assuming only semi-normality is more convenient. We will also assume that rules do not contain any ground terms.

3.2 Semantics

By an *interpretation* we mean a Herbrand interpretation of a formula, i.e. an interpretation which is a subset of its Herbrand base, and by a *model* - a *Herbrand model.* Both the Herbrand universe and the Herbrand base of a temporal formula are infinite. By the results of [16], every temporal deductive database $Z \wedge D$ (where Z is a set of rules and D is a database) has a least (Herbrand) model denoted by $M_{Z\wedge D}$. This model is also the least fixpoint $LFP(Z,D)$ of a mapping $T_{Z\wedge D}$ from interpretations to interpretations:

$$T_{Z\wedge D}(I)=\{A : A = A_0\theta,\ \theta \textit{ is a ground substitution},\ A_0 : -A_1,\ldots,A_k \in Z,\ A_1\theta \in I,\ldots,A_k\theta \in I\} \cup D.$$

We have:

$$LFP(Z,D)=\textstyle\bigcup_{k=1}^{\infty} T^k_{Z\wedge D}(\emptyset).$$

In the following definitions, assume that M is a set [3] of temporal and non-temporal tuples and $\mathbf{t0}$ and $\mathbf{t1}$ are ground temporal terms.

Define the *snapshot* $M(\mathbf{t0})$ of M as:

$$M(\mathbf{t0})=\{A : (\exists P)(\exists \bar{x})(A = P(\mathbf{t0},\bar{x}) \textit{ and } A \in M)\}.$$

$M(\mathbf{t0})$ may be thought of as the result of the selection $\sigma_{\$1=t0}(M)$. Additionally, $M(\mathbf{t0})$ is always finite, because non-temporal arguments can assume only finitely many values.

Define the *segment* $M(\mathbf{t0...t1})$ of M as:

[2] We assume that both existential and universal quantifiers come in two different sorts: one quantifying over ground temporal terms and the other - over non-temporal constants.

[3] M does not have to be finite. However, it has to contain only finitely many constant and function symbols. Every Herbrand model of a temporal deductive database has this property.

$M(\mathbf{t0...t1})=\bigcup_{t0\leq t\leq t1} M(\mathbf{t})$.

Define the *non-temporal part* M^{nt} of M as the set of all the non-temporal tuples in M.

Define the *state* $M[\mathbf{t0}]$ of M as:

$$M[\mathbf{t0}]=\{B : (\exists P)(\exists \bar{x})(B = P(\bar{x}) \textit{ and } P(\mathbf{t0}, \bar{x}) \in M)\}.$$

$M[\mathbf{t0}]$ may be thought of as a result of "projecting out" the temporal arguments in the predicates in $M(\mathbf{t0})$. Therefore, it is a finite, function-free database. Every M has only finitely many different states. Moreover, if for every $\mathbf{t}$ we know $M[\mathbf{t}]$, we can reconstruct the entire set M.

A model M of a temporal deductive database $Z \wedge D$ (such that c is the max. depth of a temporal term in the database D) is *periodic* with period $(k-c, p)$ if:

$$(\forall \mathbf{t} \geq \mathbf{k})(M[\mathbf{t}]=M[\mathbf{t+p}]).$$

This definition assumes that the rules are normal. If they are only semi-normal, the equality of single states has to be replaced by the equality of g subsequent states (where g is the max. depth of a non-ground temporal term in Z).

Theorem 3.1 *[7] The least model $M_{Z\wedge D}$ is periodic with a period (b, p) such that $b+p$ is at most exponential in the size of the database D.*

3.3 Query processing

We adopt the view that TDDs define (possibly infinite) structures in which queries are evaluated.

Let Z be a set of temporal rules and D a temporal database. A substitution θ to the open variables of a temporal query Q is an *answer* to Q if $Q\theta$ is ground and $M_{Z\wedge D} \models Q\theta$. Query evaluation finds all answers to a given query. When Q is closed, query evaluation returns *"yes"* if $M_{Z\wedge D} \models Q$, *"no"* - otherwise. We assume that negative queries are evaluated using the Closed World Assumption.

In temporal deductive databases, the least model $M_{Z\wedge D}$ may be infinite and consequently there may be infinitely many different answers to a query. To handle this problem, we introduced in [6] the notion of a *relational specification*: a finite structure equivalent to the infinite least model. Here we present it in a slightly different formulation.

A *relational specification* $S_{Z\wedge D}$ of $L = M_{Z\wedge D}$ is a triple (T, B, W) where T is a finite set of ground temporal terms, B is a temporal database (by definition finite) and W is a finite set of ground rewrite rules whose both sides are temporal terms. We will call the terms in T *representative* and B - the *primary* database. We will write $\mathbf{t}\overset{W}{\leadsto}\mathbf{t0}$ to indicate that the ground term $\mathbf{t}$ can be rewritten to $\mathbf{t0}$ using the rules in W and no more rewritings are applicable ($\mathbf{t0}$ is a canonical form of $\mathbf{t}$).

A relational specification $S_{Z\wedge D} = (T, B, W)$ of $L = M_{Z\wedge D}$ satisfies the following conditions:

1. $B=\bigcup_{t\in T} L(\mathbf{t})\cup L^{nt}$.
2. for every ground temporal term $\mathbf{t}$, there is a term $\mathbf{t0}\in T$ such that $\mathbf{t}\overset{W}{\leadsto}\mathbf{t0}$ and $L[\mathbf{t}]=L[\mathbf{t0}]$. The term $\mathbf{t0}$ is called a representative of $\mathbf{t}$.

The last condition suggests how to evaluate a ground atomic query in $S_{Z\wedge D}$: rewrite the query atom using W until no more rewrite rules apply. If the rewritten atom is in B, answer *"yes"*, otherwise answer *"no"*. Similarly, an open query may be simply computed on the primary database B [4]. There will be finitely many answer substitutions, each representing possibly infinitely many original answer substitutions. The correspondence between those two types of substitutions are captured by the rewrite rules, so the rewrite rules themselves should be a part of the query answer.

The notion of a relational specification is applicable to functional deductive databases - a generalization of TDDs. Relational specifications are well-defined only if rules are *range-restricted*: every variable in the head has to appear also in the body. A similar requirement has been postulated for DATALOG [14]. In the rest of this paper, we will assume that the rules are range-restricted.

In [6, 5], we showed a procedure to compute a relational specification $S_{Z\wedge D} = (T, B, W)$. In the following, we will refer to the specification computed by this procedure as *the* relational specification $S_{Z\wedge D}$. This specification has the property that W is a terminating rewrite system and every ground temporal term has a single representative in T.

In the case of TDDs, the relational specification has a particularly simple form: the set W contains exactly one rewrite rule:

$$\mathbf{c+b+p} \rightarrow \mathbf{c+b}$$

where (b, p) is a period of $M_{Z\wedge D}$ and c is the max. depth of a temporal term in the database D. The procedure presented in [6] computes a minimal period (b, p) of $M_{Z\wedge D}$ which has the following property:

[4] This method is appropriate only if it leaves the query answer unchanged. We show that this is indeed the case for temporal queries.

$b + p \leq t + 1$

where t is the max. depth of a representative term in $S_{Z \wedge D}$.

Example: Take the following set of rules Z consisting of one rule:

```
even(T+2) :- even(T).
```

and the following database:

```
even(0).
```

The relational specification (T, B, W) of $M_{Z \wedge D}$ is as follows:

T={**0**, **1**}.
B={`even(0)`}.
W={**2**→**0**}.

We use the specification to answer queries. For example, the query `even(4)` will be first rewritten as `even(2)` and then as `even(0)`. The tuple `even(0)` is in the primary database B, thus the answer to the original query is *"yes"*.

On the other hand, the query `even(3)` will be rewritten as `even(1)` and no further. But the tuple `even(1)` is not in B, thus the answer is *"no"*.

An answer to an open query `even(X)` consists of the substitution **X=0** and the rewrite rule **2**→**0**. This answer represents infinitely many answer substitutions:

X=0, **X=2**, **X=4**...

□

The construction of the relational specification makes *explicit* the period of a TDD.

We will say that a query $Q(x_1, \ldots, x_k)$ is *invariant* with respect to relational specifications if for every set of temporal rules Z and every temporal database D:

$(\forall x_1, \ldots, x_k)$
$(M_{Z \wedge D} \models Q(x_1, \ldots, x_k)$ iff $B \models Q(r(x_1), \ldots, r(x_k)))$.

where $S_{Z \wedge D} = (T, B, W)$ is the relational specification of $M_{Z \wedge D}$ and:

$r(x_i) = x_i$ if x_i is non-temporal.
$r(x_i) = x_0$ such that $x_i \overset{W}{\rightsquigarrow} x_0$, if x_i is temporal.

It should be clear that for a representative term y, $r(y) = y$.

Proposition 3.1 *Every temporal query is invariant w.r.t. relational specifications.*

Proof: See Appendix. □

Therefore, any temporal query Q can be answered by first computing $S_{Z \wedge D} = (T, B, W)$ and then evaluating Q over B using W to rewrite ground temporal terms in the query. In the definition of query evaluation over B, the quantifiers binding temporal variables are interpreted as ranging over the set of representative terms.

3.4 Computational complexity

The graph $Gr(Z, Q)$ of a set of temporal rules Z and a temporal query Q is defined as:

$$Gr(Z, Q) = \{(\bar{d}, D) : M_{Z \wedge D} \models Q(\bar{d})\}.$$

where $\bar{d}$ is a vector of ground terms of appropriate arity and D is a temporal database. The complexity of $Gr(Z, Q)$ will be called the *data complexity* [4, 17] of Z and Q. We will be interested in the complexity of determining the membership in $Gr(Z, Q)$ (*yes-no query processing*) and the complexity of computing the relational specification of $M_{Z \wedge D}$ (*all-answers query processing*). Because all temporal queries are invariant w.r.t. relational specifications, all the answers to any temporal query can be obtained from $S_{Z \wedge D}$.

Theorem 3.2 *[12, 7] For a set of temporal rules Z and a ground atomic query Q, yes-no query processing is PSPACE-data-complete.*

Theorem 3.3 *[5] For a set of temporal rules Z, computing the relational specification $S_{Z \wedge D}$ of $M_{Z \wedge D}$ is PSPACE-data-complete. There is a set of temporal rules Z for which the size of $S_{Z \wedge D}$ is exponential in the size of D.*

The exponential size specification mentioned in the above theorem consists of exponentially many representative terms, an exponentially sized primary database and a single rule whose left side is an exponential ground temporal term. Clearly, the non-temporal part of $M_{Z \wedge D}$ (which is also a part of $S_{Z \wedge D}$) is always at most polynomial in size.

4 Tractability

The above results state that obtaining a *yes-no* answer to a ground atomic temporal query in TDDs can not be done in polynomial time unless P=PSPACE. The same is true of computing relational specifications. These facts motivate our interest in classes of temporal rules for which queries may be processed in polynomial time.

We say that a set of temporal rules Z is *tractable* if for every temporal database D, the relational specification $S_{Z \wedge D}$ can be computed in time polynomial in the size of D. The size of the database D is considered here to be $max(n, c)$ where n is the number of tuples in D and c the maximum depth of a temporal term in D. This is equivalent to the assumption that temporal terms in D are encoded in unary.

We argue that our notion of tractability is quite robust. The polynomially-sized relational specification $S_{Z \wedge D}$, once computed, can be used to answer queries. If a query is invariant w.r.t. $S_{Z \wedge D}$ and is computable in polynomial time on finite structures, then it is also computable in polynomial time on the possibly infinite $M_{Z \wedge D}$ (provided, of course, that the size of $S_{Z \wedge D}$ is polynomial in the size of D). We know that temporal queries, although quite general, satisfy both criteria: invariance and polynomial time computability on finite structures, and consequently are polynomial time computable.

There are other candidate criteria of tractability which differ from ours by considering smaller classes of queries. For example, one can take only ground atomic queries. But then a gap appears: more general yes-no queries, e.g. existentially quantified, can no longer be computed in polynomial time unless P=PSPACE (see the PSPACE lower bound example in [7]). A similar gap exists as well between queries using only existential quantifiers and queries using also universal quantifiers, c.f. an example presented in [9] [5]. Moreover, polynomial time computability of ground atomic queries does not guarantee polynomial time computability of all the answers to queries. The relational specification $S_{Z \wedge D}$ of all answers may be exponentially-sized, while ground atomic queries to $Z \wedge D$ are polynomial time computable [5].

We are going to prove now our most general result, a complete semantic characterization of tractable sets of temporal rules. Subsequently, we are going to use this result to infer the tractability of various classes of temporal rules.

Theorem 4.1 *Let Z be a set of temporal rules and D a temporal database. The size of the relational specification $S_{Z \wedge D}$ is polynomial in the size of D iff $S_{Z \wedge D}$ can be computed in time polynomial in the size of D.*

Proof: The right-to-left direction is trivial but the left-to-right is not.

We will show first that ground atomic queries to the TDD $Z \wedge D$ can be computed in time polynomial in $max(n, c, h)$ where n is the number of tuples in D, c is the max. depth of a temporal term in D and h is the depth of the temporal term in the ground atomic query Q. We have that

$$M_{Z \wedge D} \models Q \text{ iff } Z \wedge D \wedge \neg Q \text{ is unsatisfiable.}$$

Define

$$m = max(c, h) + range(Z \wedge D).$$

where $range(Z \wedge D)$ is the number of different states in $M_{Z \wedge D}$. If the size of $S_{Z \wedge D}$ is a polynomial in $max(n, c)$, so is $range(Z \wedge D)$. Therefore, m is a polynomial in $max(n, c, h)$.

The algorithm shown in Figure 1 determines whether $Z \wedge D \wedge \neg Q$. It works in time polynomial in $max(n, m)$. Its correctness can be justified in the same way as the correctness of a bottom-up algorithm in [7, 5]. Essentially, the algorithm constructs a ground hyper-resolution refutation of $Z \wedge D \wedge \neg Q$.

L':=D
repeat
    L:=$L'(\mathbf{0...m})$
    L':=$T_{Z \wedge D}(L)$
until $L(\mathbf{0...m})$=$L'(\mathbf{0...m})$ and L^{nt}=L'^{nt}
answer:=$L \models Q$

Figure 1: Algorithm **BT**

The size of representative terms is also polynomial in the size of D, thus the algorithm from [6], computing $S_{Z \wedge D}$, runs in polynomial time as well. □

For temporal rules, the size of $S_{Z \wedge D}$ is polynomially bounded iff there exists a polynomial period of $M_{Z \wedge D}$. Therefore, we will refer to the tractability criterion from Theorem 4.1 as *polynomial periodicity.*

Currently, we don't know whether polynomial periodicity is decidable. The weaker criterion: "polynomial time computability of ground atomic temporal queries" can be easily shown to be as hard as determining whether P=PSPACE. In the following, we will show several classes of temporal rules that have polynomial periods.

5 Inflationary rules

A predicate P is *derived* by a set of rules Z if it appears in the head of some rule in Z.

[5] The example in this paper uses Skolem functions, but in this context the difference is immaterial.

A set of temporal rules Z is *inflationary* if for all temporal databases D, all ground temporal terms $\mathbf{t}$, all vectors of non-temporal constants $\bar{x}$ of appropriate arity and all temporal predicates P derived by Z:

$$M_{Z\wedge D} \models P(\mathbf{t}, \bar{x}) \Rightarrow M_{Z\wedge D} \models P(\mathbf{t}+\mathbf{1}, \bar{x}).$$

This notion is inspired by the inflationary semantics for negation [10, 2]. The graph example from the introduction is inflationary. However, if we dropped the restriction to derived predicates in the above definition, that example would no longer be considered inflationary.

Theorem 5.1 *Every inflationary set Z of temporal rules is polynomially periodic and thus tractable.*

Proof: Take a temporal database D. Assume n is the number of tuples in D and c is the max.depth of a temporal term in D. The size of every state of $L = M_{Z\wedge D}$ is bounded by a polynomial $P_1(n)$ which is independent of c. Consider the states:

$$L[\mathbf{c+1}], L[\mathbf{c+2}], \ldots, L[\mathbf{c+s+1}].$$

where $s = P_1(n)+1$. Only derived predicates may appear in any of those states, because c is the max.depth of a temporal term in D. In the above sequence, there must be a pair of identical subsequent states $L[\mathbf{t1}]$ and $L[\mathbf{t1+1}]$, because the length of a sequence of differing states

$$L[\mathbf{c+1}] \subset L[\mathbf{c+2}] \subset L[\mathbf{c+3}] \subset \ldots$$

is at most $s+1$. If $L[\mathbf{t1}]$=$L[\mathbf{t1+1}]$, then also $L[\mathbf{t1}]$=$L[\mathbf{t1+2}]$ etc. Therefore, $(P_1(n)+1, 1)$ is a period of L and Theorem 4.1 can be applied. □

Theorem 5.2 *It is decidable whether a domain-independent set of temporal rules Z is inflationary.*

Proof: We show that Z is inflationary iff for all derived predicates P_i that appear in it:

$$P_i(\mathbf{1}, \bar{a}) \in M_{Z\wedge D_i}.$$

where $D_i = \{P_i(\mathbf{0}, \bar{a})\}$ and $\bar{a}$ is a vector of appropriate length consisting of pairwise-different non-temporal constants.

The necessity of the above condition is obvious. We show that it is also sufficient, i.e. that from

$$P_i(\mathbf{1}, \bar{a}) \in L_i = M_{Z\wedge D_i}$$

it follows that Z is inflationary. Assume $\bar{a}$ stands for a vector of l elements $a_1, \ldots, a_l$, $\bar{b}$ for a vector of l elements $b_1, \ldots, b_l$ and

$$P_i(\mathbf{u}, \bar{b}) \in L = M_{Z\wedge D}.$$

We are going to show that

$$P_i(\mathbf{u}+\mathbf{1}, \bar{b}) \in L.$$

Define a mapping G between ground, temporal or non-temporal, terms:

$G(a_i) = b_i$ for $i = 1, \ldots, l$.
$G(\mathbf{0}) = \mathbf{u}$.
$G(\mathbf{s}+\mathbf{1}) = G(\mathbf{s})+\mathbf{1}$
for all ground temporal terms $\mathbf{s}$.

The mapping G can be generalized to vectors of terms in an obvious way.

Define another mapping H between ground atoms:

$H(P(\mathbf{s}, \bar{x})) = P(G(\mathbf{s}), G(\bar{x}))$ for a temporal atom.
$H(R(\bar{x})) = P(G(\bar{x}))$ for a non-temporal atom.

We are going to show that for all ground atoms A:

$$A \in L_i \Rightarrow H(A) \in L.$$

If this is true, we can take $A = P_i(\mathbf{1}, \bar{a})$ and subsequently obtain

$$H(A) = P_i(G(\mathbf{1}), G(\bar{a})) = P_i(\mathbf{u}+\mathbf{1}, \bar{b}) \in L.$$

Because $A \in L_i = LFP(Z, D_i)$, we have that $A \in T^j_{Z\wedge D_i}(\emptyset)$ for some j. We prove now the thesis by the induction on j. If $j = 1$, A has to be the only database tuple $P_i(\mathbf{0}, \bar{a})$. By the original assumption

$$H(A) = P_i(\mathbf{u}, \bar{b}) \in L.$$

Assume that the thesis is true for some j and take an atom $A \in T^{j+1}_{Z\wedge D_i}(\emptyset)$. If it belongs to D_i, we reason as above. Otherwise, there must be a rule

$$r : A_0 :- A_1, \ldots, A_k \in Z$$

and a ground substitution θ such that

$$A_1\theta \in T^j_{Z\wedge D_i}(\emptyset), \ldots, A_k\theta \in T^j_{Z\wedge D_i}(\emptyset).$$

and $A_0\theta = A$. From the inductive assumption, for all A_p, $p = 1, \ldots, k$:

$$A_p\theta \in L_i \Rightarrow H(A_p\theta) \in L.$$

We show another substitution τ such that for all A_p, $p = 0, \ldots, k$:

$H(A_p\theta) = A_p\tau.$

Consequently, from

$A_1\tau \in L, \ldots, A_k\tau \in L$

it follows that $A_0\tau = H(A_0\theta) \in L$.

We specify now the substitution τ and prove that it satisfies the required condition. For the temporal variable $\mathbf{T}$:

$$\mathbf{T}\tau = G(\mathbf{T}\theta).$$

and similarly for every non-temporal variable y:

$$y\tau = G(y\theta).$$

Notice that $y\theta$ has to be one of the a_i that appear in D_i, by the range-restrictedness of rules. Also, $\mathbf{T}\theta$ is a ground temporal term and G is defined for every such term. Thus τ is well defined.

Consider $A_p = P(\mathbf{T}, \bar{x})$ where $\mathbf{T}$ is the temporal variable. Now:

$$H(A_p\theta) = H(P(\mathbf{T}\theta, \bar{x}\theta)) = P(G(\mathbf{T}\theta), G(\bar{x}\theta)) = \\ = P(\mathbf{T}\tau, \bar{x}\tau) = A_p\tau.$$

If A_p is a non-temporal atom, the argument is similar. If $A_p = P(\mathbf{T}+\mathbf{1}, \bar{x})$:

$$H(A_p\theta) = H(P(\mathbf{T}\theta + 1, \bar{x}\theta)) = P(G(\mathbf{T}\theta{+}\mathbf{1}), G(\bar{x}\theta)) = \\ = P(G(\mathbf{T}\theta){+}\mathbf{1}, G(\bar{x}\theta)) = P(\mathbf{T}\tau + 1, \bar{x}\tau) = A_p\tau.$$

□

6 I-periodicity

A set of temporal rules Z is *I-periodic* if there is a pair of integers (b_0, p_0) which is a period of $M_{Z\wedge D}$ for every temporal database D. We will call $i_Z = (b_0, p_0)$ an *I-period* of Z.

Clearly, an I-period of Z does not have to be (and usually isn't) a minimal period of $M_{Z\wedge D}$ for a given database D.

Theorem 6.1 *Every I-periodic set Z of temporal rules is polynomially periodic and thus tractable.*

However, I-periodicity is not an effective notion.

Theorem 6.2 *Testing I-periodicity is undecidable.*

Proof: By reduction from boundedness detection shown undecidable in [8]. A set of function-free rules S is *strongly k-bounded* if for every function-free database D, $LFP(S, D) = T^k_{S\wedge D}(\emptyset)$,

Take any set of function-free rules S. Create a set of temporal rules S' in the following way.

For every rule rule $r \in S$ create a temporal rule $r' \in S'$ which counts the iterations of r. For example, if r is:

```
a(X,Z) :- p(X,Y), a(Y,Z).
```

then r' is:

```
a(T+1,X,Z) :- p(T,X,Y), a(T,Y,Z).
```

For every predicate create a copying rule. E.g.

```
a(T+1,X,Y) :- a(T,X,Y).
```

Finally, transform every function-free database to a temporal database by extending every tuple with a temporal argument equal to $\mathbf{0}$. E.g.

```
a(0,b,c).
```

It is easy to see that S is strongly k-bounded iff S' is I-periodic (with the I-period equal to $(k, 1)$).

□

We define now syntactically the class of *multi-separable* rules guaranteed to be I-periodic. A temporal rule r is *time-only* if it is recursive and non-temporal arguments in all the occurrences of the recursive predicate are identical. A time-only rule is *reduced* if every non-temporal argument that appears in in its body appears also in its head.

Example: The following rule is time-only and reduced:

```
near(T+1,X,Y) :- near(T,X,Y),
      idle(T,X), idle(T,Y).
```

□

A temporal rule r is *data-only* if it is recursive and the temporal argument in all the temporal literals is identical.

Example: The following rule is data-only:

```
happy(T,X) :- happy(T,Y), friend(X,Y).
```

□

A set of rules is *multi-separable* if it is mutual-recursion free and all the rules defining a recursive predicate are either time-only or data-only.

Our main result here is Theorem 6.5 which states that multi-separable rules are I-periodic and therefore tractable. Without loss of generality, we may assume that time-only rules in a multi-separable set of temporal rules are reduced. The reduced form may be obtained through the introduction of additional predicates and additional non-recursive rules. This transformation preserves multi-separability. We assume here that the rules are semi-normal, because the normalization [6] introduces mutual recursion. Therefore, a multi-separable set of rules may become non-multi-separable after the normalization. However, the periodicity of a least model is the same for normal and semi-normal rules if for semi-normal rules it is redefined as suggested in section 3.

The following two theorems show that I-periodic and time-only rules are very closely related.

Theorem 6.3 *Every set of reduced time-only rules Z is I-periodic.*

Proof: The basic idea is as follows: there is only a finite, constant number of pairs $(b_1, p_1), \ldots, (b_k, p_k)$ such that for every database D, one of the above pairs is a period of $M_{Z \wedge D}$. This fact is true when Z is reduced time-only, even though the number of non-temporal constants that may appear in a database D is unbounded and the temporal terms in D can be arbitrarily large. The pair $(max_{i=1,\ldots,k}(b_i), \prod_{i=1,\ldots,k} p_i)$ is an I-period of Z. We show now how to obtain the periods $(b_1, p_1), \ldots, (b_k, p_k)$.

Initially, assume that all the predicates in Z are temporal and have exactly two arguments. Define the following equivalence relation $x \asymp_L y$ between non-temporal constants x and y where $L = LFP(Z, D)$:

$$x \asymp_L y \text{ iff } (\forall P)(\forall \mathbf{s})(P(\mathbf{s}, x) \in L \equiv P(\mathbf{s}, y) \in L).$$

Additionally, assume now that the only temporal term in the database D is $\mathbf{0}$. Then there is a very simple sufficient condition for $x \asymp_L y$, namely:

$$(\forall P)(P(\mathbf{0}, x) \in D \equiv P(\mathbf{0}, y) \in D) \Rightarrow x \asymp_L y.$$

This condition may be justified by noticing that $P(\mathbf{s}, x) \in L$ is derived solely on the basis of tuples with the second argument equal to x.

On the basis of the database D, we construct a *skeleton* database D' in the following way. For every equivalence class of $\asymp_L$, we choose one *delegate* constant. The database D' is formed by removing from D tuples containing non-delegate constants. We will say that a non-delegate constant is *eliminated* by the corresponding delegate constant. Denote $L' = LFP(Z, D')$. Notice that in view of the preceding paragraph, L' may be equivalently constructed by removing tuples with non-delegate constants from L. We show that the periods of L and L' are identical. More precisely:

$$(\forall \mathbf{t1})(\forall \mathbf{t2})(L[\mathbf{t1}]{=}L[\mathbf{t2}] \text{ iff } L'[\mathbf{t1}]{=}L'[\mathbf{t2}]).$$

Assume $L[\mathbf{t1}]{=}L[\mathbf{t2}]$. Take $P(\mathbf{t1}, z) \in L'$. Clearly, $P(\mathbf{t1}, z) \in L$ and consequently $P(\mathbf{t2}, z) \in L$. Now because z is a delegate, also $P(\mathbf{t2}, z) \in L'$.

Assume $L'[\mathbf{t1}]{=}L'[\mathbf{t2}]$. Take $P(\mathbf{t1}, z) \in L$. If z is a delegate, then $P(\mathbf{t1}, z) \in L'$. Consequently, $P(\mathbf{t2}, z) \in L'$ and $P(\mathbf{t2}, z) \in L$. Otherwise, assume that z has been eliminated by x. Clearly, $P(\mathbf{t1}, x) \in L'$ and as above $P(\mathbf{t2}, x) \in L$. Thus also $P(\mathbf{t2}, z) \in L$.

Therefore, in the construction of the I-period of Z, we have to consider only the periods of $LFP(Z, D)$ where D is a skeleton database.

Take a truth assignment T to literals $P(\mathbf{0}, x)$ for some constant x and all predicates P in Z. In any skeleton database D', there can be at most one (delegate) constant y such that:

$$(\forall P)(T(P(\mathbf{0}, x)) = true \equiv P(\mathbf{0}, y) \in D').$$

If s is the number of predicates in Z, then there are 2^s such truth assignments and consequently 2^{2^s} skeleton databases with possibly different periods. To obtain an I-period of Z, those periods are computed by the algorithm constructing relational specifications [6] and combined as suggested at the beginning of the proof.

We sketch now how to relax the initial assumptions. If the database D contains temporal terms different from $\mathbf{0}$, but the set of rules Z is normal, skeleton databases may be constructed as above. This is partly due to the fact that a period of $LFP(Z, D)$ is defined relative to the biggest temporal term in D. If Z is semi-normal, skeleton databases will contain tuples with the temporal argument greater than $\mathbf{0}$ but less than $\mathbf{g}$ (where g is the max. depth of a non-ground temporal term in Z). If non-temporal predicates are allowed in Z, skeleton databases will contain non-temporal tuples. Finally, if the arity of predicates is not restricted, the relation $\asymp_L$ should be defined between vectors of constants. $\Box$

Theorem 6.4 *For every I-periodic set of temporal rules Z, there is a mutual-recursion-free set of reduced time-only rules Z_1 such that for every temporal database D, there is a temporal database D_1 satisfying the following conditions:*

- $M_{Z \wedge D} {=} M_{Z_1 \wedge D_1}$.
- *biggest temporal terms in D and D_1 differ by a database-independent constant.*

Proof: If the I-period of Z is (b, p), then for every predicate P in Z, put in Z_1 the following rule:

P($\mathbf{T}$+$\mathbf{p}$,$\bar{x}$) :- P($\mathbf{T}$,$\bar{x}$).

Let $L = M_{Z \wedge D}$ and $\mathbf{c}$ be the biggest temporal term in D. Put in D_1 all such tuples P($\mathbf{t}$,$\bar{x}$)$\in L$ where $\mathbf{t} \leq \mathbf{c}+\mathbf{b}$. □

Theorem 6.5 *Every multi-separable set Z of temporal rules is I-periodic.*

Proof: Because Z is mutual-recursion-free, we can assign a different level number to every predicate. The proof is by induction on the level number. □

7 Related work

Recently, there has been a considerable interest in *temporal logic programming* [1, 3, 7], i.e. Horn rules capable of modelling infinite temporal phenomena. The approach in [1] and [3], is to pick a subset of Linear Temporal Logic which can be treated as a programming language. Usually, this means that the subset has well-defined operational semantics. The subset studied in those papers consists of Horn clauses with appropriately restricted occurrences of modal temporal operators. The semantics is patterned after the well-known semantics of standard logic programs [16].

We have been pursuing a different approach. Instead of extending the language of logic programs, we restricted it by designating one argument in predicates as *temporal*, i.e. containing terms built from $\mathbf{0}$ and the unary function symbol $+\mathbf{1}$. Clearly, such a language inherits the declarative and operational semantics of [16]. Moreover, it captures the same class of infinite temporal phenomena as the modal language of [3]. Having database applications in mind, we further restricted this language by disallowing function symbols in non-temporal arguments obtaining *temporal deductive databases*: a temporal extension of DATALOG.

In [7], we suggested a different, database-independent, approach to finite representation of infinite query answers based on the notion of *infinite objects*. However, it was applicable only to *separable* temporal rules, defined similarly to multi-separable rules in this paper, except for an additional requirement that recursive time-only rules have at most one temporal literal in the body. Even for separable rules, the approach of [7] produced exponential-size representations of query answers.

In [6], we studied a generalization of TDDs where more than one function sumbol is allowed. Unfortunately, for this class of rules the proof of Theorem 4.1 does not go through and no tractable subclasses have been identified.

8 Further work

This work raises a lot of interesting questions.

We have studied two tractable classes of temporal rules: inflationary and I-periodic rules. Both classes are useful and practical: inflationary sets of rules can be effectively recognized and I-periodic sets of rules are very closely approximated by syntactically defined multi-separable rules. Other useful tractable classes should exist as well.

How hard is the verification of polynomial periodicity - our criterion of tractability? Can this criterion be formulated without a reference to relational specifications? Is there a class of queries whose polynomial time computability will capture the same notion?

Intuitively, we would like all queries which are "easy" (PTIME-computable) on finite structures, be also PTIME-computable in tractable TDDs. However, in this paper we establish only that this correspondence holds for queries which are invariant w.r.t. relational specifications. There are very simple queries which are not invariant.

Example: Consider the equality of temporal terms:

$$E(\mathbf{s}, \mathbf{t}) \equiv \mathbf{s} = \mathbf{t}.$$

Take the following TDD $Z \wedge D$:

```
p(T+1) :- p(T).
p(0).
```

The relational specification $S_{Z \wedge D} = (T, B, W)$:

$T = \{\mathbf{0}\}$.
$B = \{\mathrm{p}(\mathbf{0})\}$.
$W = \{\mathbf{1} \Rightarrow \mathbf{0}\}$.

Take $y_0 = \mathbf{0}$ and $y_1 = \mathbf{1}$. We have that

$$r(y_0) = r(y_1) = \mathbf{0}.$$

. Therefore:

$$B \models E(r(y_0), r(y_1)).$$

but:

$$M_{Z \wedge D} \not\models E(y_0, y_1).$$

□

Such queries can not be evaluated by constructing the relational specification first. The notion of query evaluation becomes problematic, because the structure in which the query is to be evaluated, namely $M_{Z \wedge D}$, may be infinite. Alternatively, query evaluation may be defined syntactically, for example along the lines of Reiter's syntactic reconstruction of the Closed World Assumption [13]. But in the presence of function symbols, Reiter's construction leads to infinite formulas.

The next step after identifying tractable temporal rules is to study the methods of optimizing their evaluation. In particular, various methods of *rule rewriting* devised for DATALOG [15] might be applicable to temporal rules as well.

Acknowledgments

I am grateful to Tomasz Imielinski for suggesting that I work on this topic and for many discussions. Thanks go to Jack Minker and the University of Maryland Institute for Advanced Computer Studies for support during the early stages of the preparation of this paper. Tony Bonner provided several valuable e-mail comments.

References

[1] M. Abadi and Z. Manna. Temporal Logic Programming. In *IEEE Symposium on Logic Programming*, pages 4–16, San Francisco, Ca., September 1987.

[2] S. Abiteboul and V. Vianu. Procedural and Declarative Database Update Languages. In *ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, 1988.

[3] M. Baudinet. Temporal Logic Programming is Complete and Expressive. In *ACM SIGPLAN Symposium on Principles of Programming Languages*, 1989.

[4] A.K. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.

[5] J. Chomicki. *Functional Deductive Databases: Query Processing in the Presence of Limited Function Symbols*. PhD thesis, Rutgers University, New Brunswick, New Jersey, January 1990.

[6] J. Chomicki and T. Imielinski. Relational Specifications of Infinite Query Answers. In *ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, May 1989.

[7] J. Chomicki and T. Imielinski. Temporal Deductive Databases and Infinite Objects. In *ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, Austin, Texas, March 1988.

[8] H. Gaifman, Sagiv Y., H. Mairson, and M.Y. Vardi. Undecidable Optimization Problems for Database Logic Programs. In *IEEE Symposium on Logic in Computer Science*, 1987.

[9] T. Imielinski. Complexity of query processing in incomplete databases. Submitted.

[10] P.G. Kolaitis and C.H. Papadimitriou. Why not Negation by Fixpoint? In *ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 231–239, Austin,Texas, March 1988.

[11] J.W. Lloyd. *Foundations of Logic Programming.* Springer Verlag, 2nd edition, 1987.

[12] D.A. Plaisted. Complete Problems in the First-Order Predicate Calculus. *Journal of Computer and System Sciences*, 29:8–35, 1984.

[13] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modeling*, pages 191–233, Springer-Verlag, 1984.

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

[15] J.D. Ullman. *Principles of Database and Knowledge-Base Systems.* Volume 2, Computer Science Press, 1989.

[16] M.H. van Emden and R.A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, 1976.

[17] M.Y. Vardi. The Complexity of Relational Query Languages. In *ACM SIGACT Symposium on Theory of Computing*, pages 137–146, 1982.

A Appendix

We prove Proposition 3.1 in several steps. First, it immediately follows from the definition that any *atomic* open query is invariant w.r.t. relational specifications. Assume now that $S_{Z \wedge D} = (T, B, W)$ is the relational specification and the mapping r is defined as in Section 3. Quantifiers binding temporal variables in the query are interpreted as ranging over representative terms.

Lemma A.1 *If the query $Q(y, z_1, \ldots, z_k)$ is invariant w.r.t. relational specifications, then so is the query*

$$P(z_1, \ldots, z_k) \equiv \exists y\, Q(y, z_1, \ldots, z_k).$$

Proof: We have to prove that

$$M_{Z \wedge D} \models P(z_1, \ldots, z_k) \text{ iff } B \models P(r(z_1), \ldots, r(z_k)).$$

The left-hand side of this equivalence is equivalent to

$$M_{Z \wedge D} \models \exists y\, Q(y, z_1, \ldots, z_k).$$

which is in turn equivalent to

$$M_{Z \wedge D} \models Q(y_0, z_1, \ldots, z_k).$$

for some y_0. From the original assumption, this is equivalent to

$$B \models Q(r(y_0), r(z_1), \ldots, r(z_k)).$$

which in turn implies

$$B \models \exists y\, Q(y, r(z_1), \ldots, r(z_k)).$$

because $r(y_0)$ is a representative term. The last condition is equivalent to

$$B \models P(r(z_1), \ldots, r(z_k)).$$

Now assume:

$$B \models \exists y\, Q(y, r(z_1), \ldots, r(z_k)).$$

and let y_0 be such y. We have $r(y_0) = y_0$, because y_0 is either a representative term or a non-temporal term. Therefore:

$$B \models Q(r(y_0), r(z_1), \ldots, r(z_k)).$$

From the original assumption, it follows that

$$M_{Z \wedge D} \models Q(y_0, z_1, \ldots, z_k).$$

and

$$M_{Z \wedge D} \models \exists y\, Q(y, z_1, \ldots, z_k).$$

$\Box$

Lemma A.2 *If the query $Q(y, z_1, \ldots, z_k)$ is invariant w.r.t. relational specifications, then so is the query*

$$P(z_1, \ldots, z_k) \equiv \forall y\, Q(y, z_1, \ldots, z_k).$$

Proof: We have to prove that

$$M_{Z \wedge D} \models P(z_1, \ldots, z_k) \text{ iff } B \models P(r(z_1), \ldots, r(z_k)).$$

The left-hand side of this equivalence is equivalent to

$$M_{Z \wedge D} \models \forall y\, Q(y, z_1, \ldots, z_k).$$

We are going to verify that the above condition implies

$$B \models \forall y\, Q(y, r(z_1), \ldots, r(z_k)).$$

Take any y_0 within the scope of the above universal quantifier. We have $r(y_0) = y_0$, because either y_0 is a representative term or it is a non-temporal term. Therefore:

$$B \models Q(y_0, r(z_1), \ldots, r(z_k)).$$

follows from the original assumption and the fact that

$$M_{Z \wedge D} \models Q(y_0, z_1, \ldots, z_k).$$

for any term y_0.
Now assume that

$$B \models \forall y\, Q(y, r(z_1), \ldots, r(z_k)).$$

Take any term y_0. We have that

$$B \models Q(r(y_0), r(z_1), \ldots, r(z_k)).$$

because either $r(y_0)$ is a representative term or y_0 is a non-temporal term and $y_0 = r(y_0)$. In both cases, the above fact follows directly from the previous one. Now from the original assumption, we can infer that

$$M_{Z \wedge D} \models Q(y_0, z_1, \ldots, z_k).$$

Because y_0 was arbitrary, also:

$$M_{Z \wedge D} \models \forall y\, Q(y, z_1, \ldots, z_k).$$

□

Lemma A.3 *If the query* $Q(z_1, \ldots, z_k)$ *is invariant w.r.t. relational specifications, so is its negation:*

$$P(z_1, \ldots, z_k) \equiv \neg Q(z_1, \ldots, z_k).$$

Proof: Clearly:

$$M_{Z \wedge D} \not\models P(z_1, \ldots, z_k) \text{ iff } B \not\models P(r(z_1), \ldots, r(z_k)).$$

The left-hand side of this equivalence is equivalent to:

$$M_{Z \wedge D} \models \neg P(z_1, \ldots, z_k).$$

because we assume that negative queries are evaluated under the Closed World Assumption. The equivalence of the right-hand side to:

$$B \models \neg P(r(z_1), \ldots, r(z_k)).$$

follows from the fact that for all ground terms $a_1, \ldots, a_k$, the query $\neg P(r(a_1), \ldots, r(a_k))$ is correctly evaluated under the CWA applied just to B. Notice that we never evaluate in B atoms containing ground temporal terms which are not representative. □

Lemma A.4 *If* $Q_1(y_1, \ldots, y_k, z_1, \ldots, z_m)$ *and* $Q_2(x_1, \ldots, x_n, z_1, \ldots, z_m)$ *are invariant w.r.t. relational specifications, so are:*

$$(Q_1 \wedge Q_2)(y_1, \ldots, y_k, x_1, \ldots, x_n, z_1, \ldots, z_m)$$
$$\equiv Q_1(y_1, \ldots, y_k, z_1, \ldots, z_m) \wedge Q_2(x_1, \ldots, x_n, z_1, \ldots, z_m).$$
$$(Q_1 \vee Q_2)(y_1, \ldots, y_k, x_1, \ldots, x_n, z_1, \ldots, z_m)$$
$$\equiv Q_1(y_1, \ldots, y_k, z_1, \ldots, z_m) \vee Q_2(x_1, \ldots, x_n, z_1, \ldots, z_m).$$

Proof: Straightforward. □

All the above lemmas taken together imply Proposition 3.1.