Multi Dimensional CTL and Stratified Datalog

Theodore Andronikos
Department of Informatics
Ionian University
andronikos@ionio.gr

Vassia Pavlaki *
Department of Electrical and Computer Engineering
National Technical University of Athens
vpavlaki@softlab.ece.ntua.gr

Abstract

In this work we define Multi Dimensional CTL (MD-CTL in short) by extending CTL which is the dominant temporal specification language in practice. The need for Multi Dimensional CTL is mainly due to the advent of semi-structured data. The common path nature of CTL and XPath which provides a suitable model for semi-structured data, has caused the emergence of work on specifying a relation among them aiming at exploiting the nice properties of CTL. Although the advantages of such an approach have already been noticed [36, 26, 5], no formal definition of MD-CTL has been given. The goal of this work is twofold; a) we define MD-CTL and prove that the "nice" properties of CTL (linear model checking and bounded model property) transfer also to MD-CTL, b) we establish new results on stratified Datalog. In particular, we define a fragment of stratified Datalog called Multi Branching Temporal (MBT in short) programs that has the same expressive power as MD-CTL. We prove that by devising a linear translation between MBT and MD-CTL. We actually give the exact translation rules for both directions. We further build on this relation to prove that query evaluation is linear and checking satisfiability, containment and equivalence are EXPTIME–complete for MBT programs. The class MBT is the largest fragment of stratified Datalog for which such results exist in the literature.

1 Introduction

Temporal logics are modal logics used for the description and specification of the temporal ordering of events [21]. The flow of time can be perceived as either linear or branching. According to the former view at each moment there is only one possible future (a typical example is Linear Temporal Logic), whereas according to the latter, at each moment time may follow different paths which represent different possible futures [19, 32]. The most prominent example of the second category are CTL (Computational Tree Logic), CTL*, and μ-calculus.

A major breakthrough in this area was the discovery of algorithmic methods for verifying temporal logic properties of finite Kripke structures. In such structures each state can be characterized by a fixed number of atomic propositions. Therefore, the question of checking a program having specific properties is reduced to that of checking whether a temporal logic formula holds on the Kripke structure that represents the program (hence the term *model checking*). Model checking has been widely used for verifying the correctness of, or finding design errors in many real-life systems [14]. Through the 1990s, CTL has become the dominant temporal specification language for industrial use [47, 11] mostly due to its balance of expressive power and efficient model checking complexity.

The recent advent of semi-structured data has caused the emergence of work on exploiting the nice properties of CTL. Some attempts have been made to relate CTL with XML query languages like XPath mainly because of their common path nature. Although not much work has been done in this direction, the advantages of such approach have been noticed [36, 26, 5]. Still, only for limited fragments of XPath this relation is possible [36]. Trying to extend the results to larger fragments calls for *Multi Dimensional CTL* [5] (another term as it appears in [26] is *multimodal CTL*). An intriguing question is whether the "nice" properties of CTL transfer also to Multi Dimensional CTL (MD-CTL in short). In the present work we give a positive answer to this question. We give a formal definition of MD-CTL and prove that it admits linear time model checking and exhibits the "bounded model property". We further build on these "nice" properties of MD-CTL to establish results on *stratified Datalog* [2, 12].

*The project is co - funded by the European Social Fund (75%) and National Resources (25%) - Operational Program for Educational and Vocational Training II (EPEAEK II) and particularly the Program HERAKLEITOS.

The introduction of Datalog played a crucial role in the design of declarative, logic-oriented database languages due to Datalog's ability to express recursive queries. Datalog is a rule-based language that has simple and elegant semantics based on the notion of minimal model or least fixpoint. This leads to an operational semantics that can be implemented efficiently, as demonstrated by a number of prototypes of deductive database systems [38, 41, 20]. In order to express queries of practical interest, negation is allowed in the bodies of Datalog rules. Of particular interest is stratified negation. In *stratified Datalog* [2, 12] negation is allowed in any predicate under the constraint that negated predicates are computed in previous strata. Stratified logic programs were introduced and studied first by Chandra and Harel [12] and soon attracted the interest of researchers [2, 48, 33, 39, 30, 29]. Simple, intuitive semantics leading to efficient implementation exists for stratified Datalog. Stratified logic programs were used to handle negation in the NAIL! system developed at Stanford University [37].

In the present work we define the class of Multi Branching Temporal (MBT in short) Datalog programs which are built up from a finite number of binary *extensional database predicates* (EDBs) and an arbitrary number of unary EDBs. We prove that the class MBT has the same expressive power as MD-CTL by devising a linear translation between MBT and MD-CTL. We further build on this relation to prove that checking *containment* of queries, i.e. checking whether one query yields a subset of the result of the other, and *equivalence* i.e. checking whether two queries produce the same answer, are EXPTIME-complete for programs belonging to the class MBT. *Satisfiability* is another problem of wide interest. A predicate in a Datalog program is said to be *satisfiable* if, for some database the program computes a non empty relation for it [35]. We prove that for the class MBT satisfiability is EXPTIME-complete and query evaluation is linear. The class MBT is the largest fragment of stratified Datalog for which such results exist in the literature. The only other result that exists is the decidability of the *equivalence* and *satisfiability* problems for programs consisting of unary only EDB predicates [35, 27].

Recently researchers have tried to exploit the nice properties of CTL by relating it to semi-structured query languages like XPath. In [26] Gottlob and Koch observe that axes such as "following" and "following-sibling" used extensively in XPath call for "multimodal CTL". They state that a possible translation could be a generalization of the one from CTL to Datalog LITE given in [25] sketching only the embedding into Datalog. In this work we give the exact translation rules and devise an embedding for *both* directions. Concerning the second aim of the present work, that is establishing results on stratified Datalog, our motivation was based on the observation that although Datalog has been the subject of research last decades, few results on stratified Datalog exist in the literature. Table 1 summarizes these results.

	Stratified negation	MBT Programs (Stratified negation with unary EDB predicates & finite number of binary EDB predicates)	Stratified negation with unary EDB predicates
Containment	undecidable	EXPTIME–complete [Section 7]	decidable [35, 27]
Equivalence	undecidable	EXPTIME–complete [Section 7]	decidable [35, 27]
Satisfiability	undecidable	EXPTIME–complete [Section 7]	decidable [35, 27]
Evaluation	polynomial	linear [Section 7]	linear [Section 7]

Table 1: Results on Query Containment, Equivalence, Satisfiability and Evaluation for fragments of Datalog with stratified negation

While trying to define the fragment of stratified Datalog that has the same expressive power with MD-CTL we have to deal with the fact that MD-CTL is interpreted over infinite paths. This means that finite Multi Dimensional Kripke structures over which we interpret MD-CTL have to be total with respect to their accessibility relations R_k. However, relational databases over which Datalog programs are interpreted do not have any constraints, i.e. the input could be any structure. This is the reason why there exist translations for the one direction i.e. from fragments of CTL to fragments of Datalog [25]. To overcome the problem, we add a self loop in those nodes that do not have a successor in R_k. In particular, we encompass it in the definition of the MBT class allowing thus for any input database to be captured. The example that follows explains further this point.

Example 1.1 *Consider the following Datalog program:*

$$\begin{cases} G(x) \longleftarrow R(x,y), H(y) \\ G(x) \longleftarrow H(x), \neg A(x) \\ H(x) \longleftarrow P(x), Q(x) \\ A(x) \longleftarrow R(x,y) \end{cases}$$

The above program returns the same set of ground facts for G on any pair of databases which only differ in adding self loops in R on these nodes that do not have a successor in R. ▲

Another important observation is that our results go through because we prove that the *bounded model property* (if a formula is satisfiable, then it is satisfiable in a "small" finite model, where "small" means of size bounded by some function of the length of the input formula [21]) holds also in MD-CTL. In simple words this means that if there is a model for a MD-CTL formula then there is also a finite model of bounded size. In CTL (and therefore in MD-CTL) we consider infinite models. So if the *bounded model property* did not hold, then our results could not carry over to Datalog where finite input is assumed.

The contributions of the present work can be summarized as follows:

- We introduce the Multi Dimensional Computation Tree Logic (in short MD-CTL).
- We prove that the nice properties of CTL (linear model checking, and bounded model property) transfer also to MD-CTL. We further prove that the validity problem for MD-CTL is EXPTIME–complete.
- We define a fragment of stratified Datalog called Multi Branching Temporal (MBT in short) programs which has the same expressive power with MD-CTL.
- We give the exact translation rules for both directions between MD-CTL and MBT. The rules are succinct and the translation is linear in both directions.
- We prove that the query evaluation for MBT programs is linear by reduction to the model checking problem for MD-CTL formulae. Moreover, the satisfiability and containment problems are proved to be EXPTIME–complete by reduction to the validity problem for MD-CTL formulae.

The rest of the paper is organized as follows. Section 1.4 gives an overview of related work. Section 2 is a preliminary section that reviews CTL, Datalog, and stratified Datalog. In Section 3 we formally define Multi Dimensional CTL (MD-CTL) and in Section 4 we prove that the nice properties of CTL transfer also to MD-CTL. Section 5 defines the class of Multi Branching Temporal programs which is a fragment of stratified Datalog that has the same expressive power with MD-CTL and defines an embedding from MD-CTL to the class of MBT programs. Section 6 gives an embedding from MBT programs to MD-CTL and discusses the technical challenges that arise. In Section 7 we prove that query evaluation for MBT programs is linear and that checking satisfiability and containment is EXPTIME–complete. Finally, Section 8 discusses possible future research directions.

1.1 Related work

Recently researchers have tried to exploit the known nice properties of CTL by relating it to semi-structured query languages like XPath. Miklau and Suciu [36] were the first to notice that the fragment of XPath consisting of predicates (or filters) [ ], wildcards $*$ and the descendant axis //, can be expressed in a fragment of ECTL (the existential CTL). In an independent work [26] Gottlob and Koch defined the logical core of XPath (CXPath in short) and noticed that CXPath can be encoded in plain CTL without any extensions. That is, they used only the axes "self", "child", "descendant-or-self" and "descendant". Reverse axes such as "parent", "ancestor", etc. can be dealt with by using CTL with past operators [21]. In the same work Gottlob et al. observe that the remaining axes such as "following" and "following-sibling" require multimodal CTL. In fact, they notice that multimodal CTL with past operators can still be checked in linear time. The proof they suggest consists of a generalization of the translation from CTL to Datalog LITE given in [25]. Another related work is [4] where Afanasieva defines *Many Dimensional CTL* (CTL_{Δ} in short) in order to embed $\mathcal{X}CPath$ (a fragment of XPath) into CTL. In Section 3 we explain how our work differs from [4].

One effective approach for efficiently implementing model checking consists in translating temporal logics to Logic Programming [34]. Logic Programming (LP) has been successfully used as an implementation platform for verification systems such as model checkers. Translations of temporal logics such as

CTL or μ-calculus into logic programming can be found in [40, 8, 13]. [8] presents the LMC project which uses XSB, a logic programming system that extends Prolog-style SLD resolution with tabled resolution. The database query language Datalog has inspired work in [25], where the language Datalog LITE is introduced. Datalog LITE is a variant of Datalog that uses stratified negation, restricted variable occurrences and a limited form of universal quantification in rule bodies. Datalog LITE is shown to encompass CTL and the alternation-free μ-calculus. In the same work Gottlob et al. notice that CTL can be embedded into stratified Datalog. Research on model checking in the modal μ-calculus is pursued in [51] where the connection between modal μ-calculus and Datalog$^{\neg}$ is observed. Results about the parallel complexity of Datalog$^{\neg}$ queries and a reduction of $\wedge$-free formulae of the alternation-free modal μ-calculus to Datalog$^{\neg}$ is used to derive results about the parallel computational complexity of this fragment of modal μ-calculus.

The work in [24] shows how the model checking problem for CTL can be reduced to the query evaluation problem for fragments of Datalog. In particular, a direct and modular translation from the temporal logics CTL, ETL, FCTL (CTL extended with the ability to express fairness) and the modal μ-calculus to Monadic inf-Datalog with built-in predicates was given. It is called inf-Datalog because the semantics differs from the conventional Datalog least fixed point semantics, in that some recursive rules (corresponding to least fixed points) are allowed to unfold only finitely many times, whereas others (corresponding to greatest fixed points) are allowed to unfold infinitely many times. In [1] an embedding of CTL into a fragment of Datalog$_{Succ}$ that is Datalog with the successor operator was devised.

Concerning containment of queries, the majority of research refers to conjunctive queries without negation. In [35, 27] checking equivalence of stratified Datalog programs and satisfiability were proved to be decidable but only for programs with unary EDB predicates. In the present work we extend this result to a fragment of stratified Datalog which contains also a finite number of binary EDB predicates. We call this fragment Multi Branching Temporal (MBT) Datalog programs. For the class MBT we prove that checking containment, equivalence and satisfiability are EXPTIME-complete and query evaluation is linear. The class MBT is the largest fragment of stratified Datalog for which such results exist.

2 Preliminaries

2.1 Syntax and Semantics of CTL

Temporal logics are classified as linear or branching according to the way they perceive the nature of time. In linear temporal logics every moment has a unique future (successor), whereas in branching temporal logics every moment may have more than one possible futures. Branching temporal logic formulae are interpreted over infinite trees or graphs that can be unwound into infinite trees. CTL (Computational Tree Logic) [9, 17] is a branching temporal logic that uses the path quantifiers **E** ("there exists a path") and **A** ("for all paths"). A path is an infinite sequence of states such that each state and its successor are related by the transition relation. CTL formulae contain temporal operators as well. For instance, to assert that "there is a path on which property ψ_1 holds until ψ_2 becomes true" we write $\mathbf{E}(\psi_1 \mathbf{U} \psi_2)$, where **U** is a temporal operator. The syntax of CTL dictates that each usage of a temporal operator must be preceded by a path quantifier. These pairs consisting of the path quantifier and the temporal operator can be nested arbitrarily, but must have at their core a purely propositional formula.

In this paper AP denotes the set of atomic propositions: $\{p_0, p_1, p_2, \ldots\}$ from which formulae are built. Various temporal operators are listed in the literature as part of the CTL syntax. Two of them, **X** and **U**, form a complete set from which all other (future) operators can be expressed. It is convenient to assume that CTL formulae are in *existential normal form*. This means that the universal path quantifier **A** is cast in terms of its dual existential path quantifier **E** using negation and a third temporal operator $\widetilde{\mathbf{U}}$. For instance, instead of writing $\mathbf{A}(\psi_1 \mathbf{U} \psi_2)$, we equivalently write $\neg \mathbf{E}(\neg\psi_1 \,\widetilde{\mathbf{U}}\, \neg\psi_2)$. The $\widetilde{\mathbf{U}}$ operator was initially introduced in [46, 31] precisely for this purpose.

The syntax of CTL formulae is given by rules $\mathbf{S_1} - \mathbf{S_3}$:

- $\mathbf{S_1}$: Atomic propositions and $\top$ are CTL formulae.
- $\mathbf{S_2}$: If φ and ψ are CTL formulae then so are $\neg\varphi$ and $\varphi \wedge \psi$.
- $\mathbf{S_3}$: If φ and ψ are CTL formulae then $\mathbf{EX}\varphi$, $\mathbf{E}(\varphi \mathbf{U} \psi)$ and $\mathbf{E}(\varphi \widetilde{\mathbf{U}} \psi)$ are CTL formulae.

The semantics of CTL is defined in terms of temporal Kripke structures. A temporal Kripke structure $\mathcal{K}$ is a directed labeled graph with node set W, arc set R and labeling function V. $\mathcal{K}$ need not be a tree; however, it can be turned into an infinite labeled tree if *unwound* from a given state s_0 (see [21] and [45] for details).

Definition 2.1 *Let AP be the set of atomic propositions. A* Kripke structure *$\mathcal{K}$ for AP is a tuple $\langle W, R, V \rangle$, where:*

- W *is the set of states,*
- $R \subseteq W \times W$ *is the* total *accessibility relation, and*
- $V : W \longrightarrow 2^{AP}$ *is the valuation that determines which atomic propositions are true at each state.*

A Kripke structure $\mathcal{K} = \langle W, R, V \rangle$ *is* finite *if* W *is finite.* ⊣

Definition 2.2 *A path* π *of* $\mathcal{K}$ *is an infinite sequence* $s_0, s_1, s_2, \ldots$ *of states of* W*, such that* $R(s_i, s_{i+1})$*,* $i \geq 0$*.* ⊣

The notation $\mathcal{K}, s \models \varphi$ means that "the formula φ holds at state s of $\mathcal{K}$". The meaning of $\models$ is formally defined as follows:

Definition 2.3

- $\models \top$ *and* $\not\models \bot$
- $\mathcal{K}, s \models p \Longleftrightarrow p \in V(s)$*, for an atomic proposition* $p \in AP$
- $\mathcal{K}, s \models \neg\varphi \Longleftrightarrow \mathcal{K}, s \not\models \varphi$
- $\mathcal{K}, s \models \varphi \vee \psi \Longleftrightarrow \mathcal{K}, s \models \varphi$ *or* $\mathcal{K}, s \models \psi$
- $\mathcal{K}, s \models \varphi \wedge \psi \Longleftrightarrow \mathcal{K}, s \models \varphi$ *and* $\mathcal{K}, s \models \psi$
- $\mathcal{K}, s \models \mathbf{E}\varphi \Longleftrightarrow$ there exists a path $\pi = s_0, s_1, \ldots$, with initial state $s = s_0$, such that $\mathcal{K}, \pi \models \varphi$
- $\mathcal{K}, s \models \mathbf{A}\varphi \Longleftrightarrow$ for every path $\pi = s_0, s_1, \ldots$, with initial state $s = s_0$ it holds that $\mathcal{K}, \pi \models \varphi$
- $\mathcal{K}, \pi \models \mathbf{X}\varphi \Longleftrightarrow \mathcal{K}, \pi^1 \models \varphi$
- $\mathcal{K}, \pi \models \varphi \mathbf{U} \psi \Longleftrightarrow$ there exists $i \geq 0$ such that $\mathcal{K}, \pi^i \models \psi$ and for all j, $0 \leq j < i$, $\mathcal{K}, \pi^j \models \varphi$
- $\mathcal{K}, \pi \models \varphi \widetilde{\mathbf{U}} \psi \Longleftrightarrow$ for all $i \geq 0$ such that $\mathcal{K}, \pi^i \not\models \psi$ there exists j, $0 \leq j < i$, such that $\mathcal{K}, \pi^j \models \varphi$ ⊣

A CTL state formula φ is *satisfiable* if there exists a Kripke structure $\mathcal{K} = \langle W, R, V \rangle$ such that $\mathcal{K}, s \models \varphi$, for some $s \in W$. In this case $\mathcal{K}$ is a *model* of φ. If $\mathcal{K}, s \models \varphi$ for every $s \in W$, then φ is *true* in $\mathcal{K}$, denoted $\mathcal{K} \models \varphi$. If $\mathcal{K} \models \varphi$ for every $\mathcal{K}$, then φ is *valid*, denoted $\models \varphi$. If $\mathcal{K} \models \varphi$ for every *finite* $\mathcal{K}$, we say that φ is valid with respect to the class of *finite* Kripke structures, denoted $\models_f \varphi$.

2.2 Model Checking and Complexity

Model checking is the problem of verifying the conformance of a finite state system to a certain behavior, i.e. verifying that the labeled transition graph satisfies the formula that specifies the behavior. Hence, given a labeled transition graph $\mathcal{K}$, a state s and a temporal formula φ, the model checking problem for $\mathcal{K}$ and φ is to decide whether $\mathcal{K}, s \models \varphi$. The *size* of the labeled transition system $\mathcal{K}$, denoted $|\mathcal{K}|$, is taken to be $|W| + |R|$ and the *size* of the formula φ, denoted $|\varphi|$, is the number of symbols in φ.

For CTL formulae the model checking problem is known to be P–hard [42], something that renders the development of efficient parallel algorithms highly improbable. However, there exist efficient algorithms that solve this problem in $O(|\mathcal{K}| \cdot |\varphi|)$ time [10]. Although in most practical applications the crucial factor is $|\mathcal{K}|$, which is much larger than $|\varphi|$, it is insightful to examine how the two parameters $|\mathcal{K}|$ and $|\varphi|$ affect the complexity. This can be done by introducing the following two complexity measures for the model checking problem [49]:

- *data* complexity, which assumes a *fixed* formula and *variable* Kripke structures, and
- *program* or *formula* complexity, which refers to *variable* formulae over a *fixed* Kripke structure.

CTL model checking is NLOGSPACE–complete with respect to data complexity and its formula complexity is in $O(\log |\varphi|)$ space [42].

Another important problem for CTL is the *validity* problem, that is deciding whether a formula φ is valid or not. This problem is much harder; it has been shown to be EXPTIME-complete [45].

Theorem 2.1 (*Validity*) [45] *The validity problem for CTL is EXPTIME–complete.*

CTL exhibits an important property, namely the *bounded model* property: if a formula φ is satisfiable, then φ is satisfiable in a structure of bounded cardinality. As Vardi remarks in [45] this property is stronger than the finite model property which says that if φ is satisfiable, then φ is satisfiable in a finite structure.

Theorem 2.2 (*Bounded Model Property*) [21] *If a CTL formula φ has a model, then φ has a model with at most $2^{c|\varphi|}$ states, for some $c > 0$.*

In Section 4 we prove that model checking for MD-CTL is linear and Theorems 2.1 and 2.2 also hold for MD-CTL.

2.3 Datalog

Datalog [43] is a query language for relational databases. An *atom* is an expression of the form $E(x_1, \ldots, x_r)$, where E is a predicate symbol and $x_1, \ldots, x_r$ are either variables or constants. A *ground fact* (or *ground atom*) is an atom of the form $E(c_1, \ldots, c_r)$, where $c_1, \ldots, c_r$ are constants. From a logic perspective, a relation corresponding to a predicate symbol E is just a finite set of ground facts of E and a relational database D is a finite collection of relations. To simplify notation, in the rest of this paper we use the same symbol for the relation and the predicate symbol; which one is meant will be clear from the context.

A *database schema* $\mathfrak{D}$ [15] is an ordered tuple $\langle W, E_1, \ldots, E_n \rangle$, where W is the *domain* of the schema and $E_1, \ldots, E_n$ are predicate symbols, each with its associated arity. Given a database schema $\mathfrak{D}$, the set of all ground facts formed from E_1, ..., E_n using as constants the elements of W is denoted $\mathcal{HB}(W)$. A *database* D over $\mathfrak{D}$ is a *finite* subset of $\mathcal{HB}(W)$; in this case, we say that $\mathfrak{D}$ is the underlying schema of D. The *size* of a database D, denoted $|D|$, is the number of ground facts in D.

Definition 2.4 *A* Datalog program Π *is a finite set of function-free Horn clauses, called* rules. *Rules have the following form:*

$$G(x_1, \ldots, x_n) \longleftarrow B_1(y_{1,1}, \ldots, y_{1,n_1}), \ldots, B_k(y_{k,1}, \ldots, y_{k,n_k}) \quad (1)$$

where:

- $x_1, \ldots, x_n$ *are variables,*
- $y_{i,j}$*'s are either variables or constants,*
- $G(x_1, \ldots, x_n)$ *is the* head *of the rule, and*
- $B_1(y_{1,1}, \ldots, y_{1,n_1}), \ldots, B_k(y_{k,1}, \ldots, y_{k,n_k})$ *are the* body *of the rule.* ⊣

Predicates that appear in the head of some rule are called *IDB* predicates (intensional database predicates), while predicates that appear only in the bodies of the rules are called *EDB* predicates (extensional database predicates). Each Datalog program Π is associated with an ordered pair of database schemas $(\mathfrak{D}_i, \mathfrak{D}_o)$, called the *input-output schema*, as follows: $\mathfrak{D}_i$ and $\mathfrak{D}_o$ have the same domain and contain exactly the EDB predicates and the IDB predicates of Π, respectively. Given a database D over $\mathfrak{D}_i$, the set of ground facts for the IDB predicates of Π, which can be deduced from D by applications of the rules of Π, is the output database D' (over $\mathfrak{D}_o$), denoted $\Pi(D)$. In that sense, databases over $\mathfrak{D}_i$ are mapped to databases over $\mathfrak{D}_o$ via Π. In the sequel of the paper we assume without explicitly mentioning it, that the input databases for a Datalog program Π have the appropriate schema.

Definition 2.5 *Given a Datalog program* Π *we distinguish an IDB predicate G that we call the* goal (*or* query) *predicate of* Π. *Let D be an input database; The* query evaluation *problem for G and D is to compute the set of ground facts of G in $\Pi(D)$, denoted $G_\Pi(D)$.* ⊣

The *dependency graph* of a Datalog program is a directed graph with nodes the set of IDB predicates of the program; there is an arc from predicate B to predicate G if there is a rule with head an instance of G and at least one occurrence of B in its body. The *size* of a rule r, denoted $|r|$, is the number of symbols appearing in r. Given a Datalog program $\Pi = \begin{cases} r_n \\ \ldots \\ r_0 \end{cases}$, the *size* of Π, denoted $|\Pi|$, is $|r_0| + \ldots + |r_n|$.

The data complexity of Datalog is polynomial. However, it has been shown that Datalog only captures a proper subset of monotonic polynomial-time queries if no order is present [3].

2.3.1 Stratified Datalog

Intuitively, *stratified Datalog* is a fragment of Datalog with negation allowed in any predicate under the constraint that negated predicates are computed in previous strata. Stratified Datalog has strictly higher expressive power than Datalog. Indeed, stratified programs can be divided into layers, so that at any given layer all predicates occurring negatively have been defined at a lower layer. Note that the first layer is negation-free. It follows that every Datalog program can be viewed as a stratified program with a single layer. To be more precise, each head predicate of Π is a head predicate in precisely one stratum Π_i and appears only in the body of rules of higher strata Π_j ($j > i$) [25]. This means that:

1. If G is the head predicate of a rule that contains a negated subgoal B, then B is in a lower stratum than G.
2. If G is the head predicate of a rule that contains a non negated subgoal B, then the stratum of G is at least as high as the stratum of B.

In other words a program Π is stratified, if there is an assignment $str()$ of integers $0, 1, \ldots$ to the predicates in Π, such that for each clause r of Π the following holds: if G is the head predicate of r and B a predicate in the body of r, then $str(G) \geq str(B)$ if B is non negated, and $str(G) > str(B)$ if B is negated.

Example 2.1 *For the stratified program:*

$$\begin{cases} P \longleftarrow \neg Q \\ Q \longleftarrow \neg R \\ R \longleftarrow S \end{cases}$$

$str()$ is the following: $str(R) = str(S) = 0, str(Q) = 1$ and $str(P) = 2$. ▲

The dependency graph can be used to define strata in a given program as follows: whenever there is a rule with head predicate G and negated subgoal predicate B, there is no path from G to B. There is no recursion through negation in the dependency graph of a stratified program. For more details on stratified Datalog see [43, 50].

2.3.2 Bottom-up evaluation and complexity

The *bottom-up evaluation* of a query initializes the IDB predicates to be empty and repeatedly applies the program rules to add tuples to the IDB predicates, until no new tuples can be added [6, 43, 50]. In stratified Datalog strata are used in order to structure the computation in a bottom-up fashion. The head predicates of a given stratum are evaluated only after all head predicates of the lower strata have been computed. This way any negated subgoal is treated as if it were an EDB relation.

There are two main complexity measures for Datalog and its extensions.

- *data complexity* which assumes a *fixed* Datalog program and *variable* input databases, and
- *program complexity* which refers to *variable* Datalog programs over a *fixed* input database.

In general, Datalog is P–complete with respect to data complexity and EXPTIME–complete with respect to program complexity [44, 28]. The queries computable by stratified logic programs contain properly those computable by Datalog programs, since the former are closed under negation, while the latter are not [29]. Although there are different semantics for negation in Logic Programming (e.g., stratified negation, well-founded semantics, stable model semantics, etc.), for stratified programs these semantics coincide. Stratified programs have a unique stable model which coincides with the stratified model, obtained by partitioning the program into an ordered number of strata and computing the fixpoints of every stratum in their order. As shown in [29], stratified Datalog can only express a proper subset of fixpoint queries. Datalog with stratified negation is P–complete with respect to data complexity and EXPTIME–complete with respect to program complexity [2]. An excellent survey regarding these issues is [15].

3 Multi Dimensional CTL

In this section we introduce the *Multi Dimensional Computation Tree Logic* (in short MD-CTL). Before proceeding to the definition we discuss some of the technical challenges that arise and how our approach differs from previous ones.

A key issue when generalizing CTL to more dimensions is how to handle the "totality" requirement of the accessibility relation. In CTL this presents no problem; the unique accessibility relation R is required to be total. In multi dimensional CTL however there can be two different approaches:

- demand that every accessibility relation R_k, $1 \leq k \leq n$ be total, or
- demand that the union of all the accessibility relations $\bigcup_{1\leq k\leq n} R_k$ be total.

This affects directly the definition of the path in the multi dimensional Kripke structure. Recall that formulae are interpreted over infinite paths. So, if the first approach is taken then k-paths may be defined. For instance, a k-path is comprised of states connected only by R_k. In contrast, if the second approach is adopted, then such a definition of paths is not permissible because it does not guarantee that a path formed from a specific R_k is infinite. In this case, it becomes necessary to allow paths to consist of states connected via all accessibility relations, in order to ensure that they are infinite. In this work we have chosen the first approach because we find the resulting semantics closer to the spirit of CTL and less restrictive. The second approach seems more appropriate for interpreting multi dimensional CTL formulae over finite trees. We refer the reader to [4] to see the difference when the second approach is taken, precisely for reducing $\mathcal{X}CPath$ query evaluation to CTL model checking.

3.1 Multi Dimensional CTL

Multi Dimensional CTL is an extension of CTL to a finite number of mutually independent dimensions. MD-CTL as opposed to CTL has many, instead of one, dimensions. In CTL we talk about paths but in MD-CTL we need to be more precise and talk about paths with respect to a particular dimension. We define *a path along dimension k*, for simplicity called k-path, to be an *infinite* sequence of states such that each state and its successor are related by the transition relation that corresponds to dimension k. We use the path quantifiers $\mathbf{E}_k$ and $\mathbf{A}_k$, meaning "there exists a k-path" and "for all k-paths", respectively. For instance, to assert that "property φ is always true on every k-path" or that "there is a k-path on which property ψ_1 is true until ψ_2 becomes true" we write $\mathbf{A}_k\mathbf{G}_k\varphi$ and $\mathbf{E}_k(\psi_1\mathbf{U}_k\psi_2)$, respectively. We adopt the *existential normal form*, where the universal quantifier is cast in terms of its dual existential quantifier using negation. So, instead of writing $\mathbf{A}_k(\psi_1\,\mathbf{U}_k\,\psi_2)$, we use the equivalent formula $\neg\mathbf{E}_k(\neg\psi_1\,\widetilde{\mathbf{U}}_k\,\neg\psi_2)$.

3.1.1 Syntax and Semantics

In the following paragraphs we formally define MD-CTL formulae by giving its syntax and semantics. We fix the number n of dimensions and for each dimension $k, 1 \leq k \leq n$, we use the "future" temporal operators $\mathbf{X}_k$, $\mathbf{U}_k$ and $\widetilde{\mathbf{U}}_k$. Rules $\mathbf{MS_1} - \mathbf{MS_3}$ provide the syntax of MD-CTL formulae:
$\mathbf{MS_1}$: Atomic propositions and $\top$ are MD-CTL formulae.
$\mathbf{MS_2}$: If φ and ψ are MD-CTL formulae then $\neg\varphi$ and $\varphi \wedge \psi$ are MD-CTL formulae.
$\mathbf{MS_3}$: If φ and ψ are MD-CTL formulae then so are $\mathbf{E}_k\mathbf{X}_k\varphi$, $\mathbf{E}_k(\varphi\,\mathbf{U}_k\,\psi)$ and $\mathbf{E}_k(\varphi\,\widetilde{\mathbf{U}}_k\,\psi)$, $1 \leq k \leq n$.

MD-CTL formulae are interpreted over *Multi Dimensional* Kripke structures (MD-Kripke structures in short), defined as follows:

Definition 3.1 *A MD-Kripke structure $\mathcal{K}$ for a set AP of atomic propositions is a tuple $\langle W, R_1, \ldots, R_n, V\rangle$, where:*

- *W is the set of states,*
- *$R_k \subseteq W \times W$ is the accessibility relation along dimension k (in the sequel of the paper just k-accessibility relation) which must be* total, *i.e. $\forall s \exists t R_k(s,t)$, and*
- *$V : W \longrightarrow 2^{AP}$ is a valuation that determines which atomic propositions are true at each state.*

A MD-Kripke structure $\mathcal{K} = \langle W, R_1, \ldots, R_n, V\rangle$ is finite *if W is finite.* ⊣

Definition 3.2 *A path π along dimension k (k-*path *in short) is an* infinite *sequence $s_0, s_1, s_2, \ldots$ of states of W, such that $R_k(s_i, s_{i+1})$, $i \geq 0$. π^i denotes the path $s_i, s_{i+1}, s_{i+2}, \ldots$* ⊣

Although in MD-Kripke structures the set of states W can be *infinite*, in this work we study relational databases, where the universe is *finite*. Hence, in the sequel of the paper we consider finite structures. In CTL and, therefore, in MD-CTL the computation paths are *infinite*, which means that in order for the accessibility relations R_k to be meaningful, they must be *total*, something that Definition 3.1 asserts ([31]):

$$\forall x \exists y \; R_k(x, y) \tag{2}$$

The notation $\mathcal{K}, s \models \varphi$ means that the formula φ holds at state s of $\mathcal{K}$. The meaning of $\models$ is formally defined as follows:

Definition 3.3

- $\models \top$,
- $\mathcal{K}, s \models p \iff p \in V(s)$, *for* $p \in AP$,
- $\mathcal{K}, s \models \neg\varphi \iff \mathcal{K}, s \not\models \varphi$,
- $\mathcal{K}, s \models \varphi \wedge \psi \iff \mathcal{K}, s \models \varphi$ *and* $\mathcal{K}, s \models \psi$,
- $\mathcal{K}, s \models \mathbf{E}_k \mathbf{X}_k \varphi \iff$ there exists a k-path $\pi = s_0, s_1, \ldots$, with initial state $s = s_0$, such that $\mathcal{K}, s_1 \models \varphi$,
- $\mathcal{K}, s \models \mathbf{E}_k(\varphi \mathbf{U}_k \psi) \iff$ there exists a k-path $\pi = s_0, \ldots, s_i, \ldots$, with initial state $s = s_0$, such that $\mathcal{K}, s_i \models \psi$, $(i \geq 0)$, and for all j, $0 \leq j < i$, $\mathcal{K}, s_j \models \varphi$, *and*
- $\mathcal{K}, s \models \mathbf{E}_k(\varphi \widetilde{\mathbf{U}}_k \psi) \iff$ there exists a k-path $\pi = s_0, \ldots, s_i, \ldots$, with initial state $s = s_0$, which has the following property: for all $i \geq 0$ such that $\mathcal{K}, s_i \not\models \psi$ there exists j, $0 \leq j < i$, such that $\mathcal{K}, s_j \models \varphi$. ⊣

We can think of $\mathbf{E}_k(\psi_1 \, \widetilde{\mathbf{U}}_k \, \psi_2)$ as saying that there exists a k-path on which: (1) either ψ_2 always holds, or (2) the first occurrence of $\neg\psi_2$ is strictly preceded by an occurrence of ψ_1.

A MD-CTL formula φ is *satisfiable* if there exists a MD-Kripke structure $\mathcal{K} = \langle W, R_1, \ldots, R_n, V \rangle$ such that $\mathcal{K}, s \models \varphi$, for some $s \in W$. In this case $\mathcal{K}$ is a *model* of φ. If $\mathcal{K}, s \models \varphi$ for every $s \in W$, then φ is *true* in $\mathcal{K}$, denoted $\mathcal{K} \models \varphi$. If $\mathcal{K} \models \varphi$ for every $\mathcal{K}$, then φ is *valid*, denoted $\models \varphi$. If $\mathcal{K} \models \varphi$ for every *finite* $\mathcal{K}$, we say that φ is valid with respect to the class of *finite* MD-Kripke structures, denoted $\models_f \varphi$. The *truth set* of a formula φ with respect to a structure $\mathcal{K}$, is the set of states of $\mathcal{K}$ at which φ is true.

Definition 3.4 (*Truth set*) *Given a MD-CTL formula φ and a MD-Kripke structure $\mathcal{K} = \langle W, R_1, \ldots, R_n, V \rangle$, the* truth set *of φ with respect to $\mathcal{K}$, denoted $\varphi[\mathcal{K}]$, is $\{s \in W \mid \mathcal{K}, s \models \varphi\}$.* ⊣

4 The "nice" properties of MD-CTL

In this section we prove that the nice properties of CTL transfer also to MD-CTL. In particular, we prove that (1) model checking for MD-CTL formulae is linear (Section 4.1), (2) MD-CTL exhibits the bounded model property (Section 4.2), and (3) validity for MD-CTL formulae is EXPTIME–complete (Section 4.3). The proofs are extensions of the corresponding proofs for CTL. This is mainly due to the way we defined MD-CTL by avoiding to mix dimensions. That is, by treating each dimension in separate, CTL formulae are just the fragment of MD-CTL formulae corresponding to a particular dimension.

4.1 Model checking for MD-CTL is linear

Clarke et al. in [10] presented an algorithm for verifying that a Kripke structure $\mathcal{K}$ meets a specification expressed in CTL. The complexity of the algorithm is linear in both the size of the formula and the size of the Kripke structure. In this subsection we argue that the algorithm in [10] (pp. 249-251) can be extended in a straightforward way to apply to MD-CTL formulae and MD-Kripke structures. Moreover, the complexity remains linear in both the size of the formula and the size of the structure. We present the extended algorithm which is the following:

($\mathbf{A_1}$) The procedure label_graph(φ) labels with φ those states of $\mathcal{K}$ for which φ holds. label_graph(φ) handles $3 \times n + 4$ cases, depending on whether φ is $\top$, or an atomic proposition, or has one of the following forms: $\neg\phi_1$, $\psi_1 \wedge \psi_2$, $\mathbf{E}_k\mathbf{X}_k\psi_1$, $\mathbf{E}_k(\psi_1\ \mathbf{U}_k\ \psi_2)$ and $\mathbf{E}_k(\psi_1\ \widetilde{\mathbf{U}}_k\ \psi_2)$ $(1 \leq k \leq n)$.

($\mathbf{A_2}$) The procedures for handling formulae of the form $\mathbf{E}_k\mathbf{X}_k\psi_1$ and $\mathbf{E}_k\ (\psi_1\ \mathbf{U}_k\ \psi_2)$ are the same as in [10], only now they use the corresponding k-accessibility relation R_k instead of R.

($\mathbf{A_3}$) The operator $\widetilde{\mathbf{U}}$, which was not used in [10], necessitates the introduction of the new recursive procedure eu_tilde(φ, k, s, b). eu_tilde(φ, k, s, b) determines whether φ is true at state s, when φ is of the form $\mathbf{E}_k(\psi_1\ \widetilde{\mathbf{U}}_k\ \psi_2)$. Upon termination the boolean variable b is true iff $\mathcal{K}, s \models \varphi$ and false otherwise. The specifics are shown in pseudocode in Figure 1.

```
procedure label_graph(φ)
begin
...
  {main operator is E_k Ũ_k}
  begin
      ST:=empty_stack;
      for all s ∈ W do
          marked(s):=false;
          for all s ∈ W do
          if ¬marked(s) then
              eu_tilde(φ,k,s,b)
  end
      ...
end   {label_graph}

procedure eu_tilde(φ,k,s,b)
begin
  if marked(s) then
  begin
      if labeled(s, φ) or stacked(s) then
          b:=true;
      else
          b:=false;
      return;
  end
  marked(s):=true;
  if labeled(s, ψ1) and labeled(s, ψ2) then
      begin
          add_label(s, φ);
          b:=true;
          return;
      end
  else if ¬labeled(s, ψ2) then
      begin
          b:=false;
          return;
      end
  push(s,ST);
  for all t such that R_k(s,t) do
  begin
      eu_tilde(φ,k,t,b1);
      if b1 then
          begin
              pop(ST);
              add_label(s, φ);
              b:=true;
              return;
          end
  end
  pop(ST);
```

```
    b:=false;
    return;
end   {eu_tilde}
```

Figure 1: The pseudocode for the recursive procedure `eu_tilde`.

The correctness of the above algorithm is based on the following facts:

- If s is already marked, then:
  1. If s is labeled with φ, this means that φ holds at s, so b is set to `true` and the procedure terminates.
  2. If s is in the stack, this means that there is a cycle containing s in $\mathcal{K}$ and ψ_2 is true at all states of this cycle. By construction the stack has the following property: if states $t_0, \ldots, t_i$ are in the stack, then $R_k(t_j, t_{j+1})$, $0 \leq j < i$ and ψ_2 holds at t_j, $0 \leq j \leq i$ (see [10] pages 258-259 for details). This in turn implies that φ holds at s, so again b is set to `true` and the procedure terminates.
  3. If none of the above holds, then a depth-first search from s must have been completed and φ is false at s. Therefore, b is set to `false` and the procedure terminates.
- If s is not marked, i.e. it is visited for the first time, then:
  1. If s is labeled with ψ_1 and ψ_2, meaning that ψ_1 and ψ_2 hold at s, then φ also holds at s. Consequently, s is labeled with φ, b is set to `true` and the procedure terminates.
  2. If ψ_2 is false at s, then φ is also false at s, so b is set to `false` and the procedure terminates.
  3. If none of the above holds, then ψ_2 holds at s but ψ_1 does not. Hence, a depth-first search from s is initiated; s is pushed onto the stack and its k-successors are visited in a systematic fashion. If φ is true at a k-successor of s, then φ is also true at s, in which case s, after popped from the stack, is labeled with φ, b is set to `true` and the procedure terminates. If however φ is false at all the k-successors of s, then φ too is false at s. Therefore, s is popped from the stack, b is set to `false` and the procedure terminates.

The main observation regarding the complexity is that the time for each call of `eu_tilde`(φ, k, s, b) is proportional to the number of outgoing R_k edges from s (excluding of course the time required by recursive calls). Therefore, all calls to `eu_tilde` require time proportional to the number of states plus the number of edges, i.e. time $O(|W| + |R_1| + \ldots + |R_n|) = O(|\mathcal{K}|)$. Given a formula φ we apply the `label_graph` procedure to the subformulae of φ beginning with the simplest and gradually working our way towards φ. It requires a trivial induction to check that the number of subformulae of φ is $O(|\varphi|)$ (see [10] for details). Thus, we derive the next theorem.

Theorem 4.1 *The model checking problem for MD-CTL formulae is linear in the size of the formula and in the size of the Kripke structure, that is given a MD-CTL formula φ and a finite MD-Kripke structure $\mathcal{K}$, $\varphi[\mathcal{K}]$ can be compiled in time $O(|\mathcal{K}| \cdot |\varphi|)$.*

4.2 Bounded model property

Emerson and Halpern in [18] proved that CTL has the *bounded model property*: if a formula is satisfiable, then it is satisfiable in a finite model of size bounded by some function of the length of the input formula. One way to prove such results for modal logics is to "collapse" a (possibly infinite) model by identifying states according to an equivalence relation of small finite index and then showing that the resulting finite quotient structure is still a model for the formula in question. Fischer and Ladner used this technique to prove that PDL has the small model property [23]. Emerson and Halpern in [18] demonstrated how this technique fails when applied to CTL. Still, the Fischer-Ladner quotient structure obtained from a CTL model may be viewed as a "pseudo-model" which can be unwound into a model still small. That is, they proved that a satisfiable CTL formula is satisfiable in a small finite model obtained from the "pseudo model" resulting from the Fischer-Ladner quotient construction.

The way they used to construct this pseudo-model resembles the one used in [7] to prove the corresponding results for DPDL. In particular, Emerson and Halpern reproved the results in [7] using the fixpoint characterizations [16] of the temporal operators to construct a tableau that may be viewed as a small pseudo-model. We prove now that the bounded model property transfers also to MD-CTL.

Theorem 4.2 (*Bounded Model Property*) *If a MD-CTL formula φ has a model, then it has a model with at most $2^{c|\varphi|}$ states, for some $c > 0$.*

Proof:
The proofs of Lemmata 3.8, 4.3, 4.4, 4.5, 4.6 in [18] pages 7-12 go through with minor modifications. In particular:

- The notion of fragment which is appropriate for Kripke structures with one accessibility relation must be refined to accommodate MD-Kripke structures. Thus, we define a k-fragment as in [18] page 7, only that now we use the k-th accessibility relation R_k.
- Although the path quantifier **A** is part of the syntax of CTL in [18], the temporal operator $\widetilde{\mathbf{U}}$ is not. So every formula of the form $\mathbf{E}_k(\psi_1 \,\widetilde{\mathbf{U}}_k\, \psi_2)$ is replaced with the equivalent $\neg\mathbf{A}_k(\neg\psi_1 \,\mathbf{U}_k\, \neg\psi_2)$. ■

4.3 Validity

In this subsection we show that the validity problem for MD-CTL, i.e. deciding whether a given MD-CTL formula holds at all states in all transition systems, is decidable. To be more precise, the validity problem for MD-CTL is EXPTIME–complete.

In [18] pages 12-13 Emerson and Halpern give an algorithm for deciding if a CTL formula φ is satisfiable that runs in time $2^{c|\varphi|}$ for some $c > 0$. This algorithm can also be applied to MD-CTL formulae provided that each occurrence of $\mathbf{E}_k(\psi_1 \,\widetilde{\mathbf{U}}_k\, \psi_2)$ is replaced by $\neg\mathbf{A}_k(\neg\psi_1 \,\mathbf{U}_k\, \neg\psi_2)$. In this way we derive as an upper bound that the validity problem for MD-CTL formulae is in EXPTIME. Furthermore, we know from [45] that the validity problem for CTL is EXPTIME–complete. In view of the fact that CTL is a special case of MD-CTL, this provides a lower bound for MD-CTL. Hence, the next theorem follows immediately:

Theorem 4.3 (*Validity*) *The validity problem for MD-CTL is EXPTIME–complete.*

5 Embedding MD-CTL into stratified Datalog

In this section, we define an embedding from MD-CTL into a fragment of stratified Datalog that we call the class of *Multi Branching Temporal* (in short MBT) programs. The results of this section combined with the results of Section 6 prove that the class MBT has the same expressive power with MD-CTL over finite structures. We start with the formal definition of MBT programs and then proceed to show how MD-Kripke structures can be seen as relational databases and MD-CTL formulae as relational queries. Finally we give the exact rules of this embedding.

5.1 Multi Branching Temporal programs

Multi Branching Temporal programs are built up from unary and binary EDB predicates and contain only unary and binary IDB predicates. One particular IDB predicate is chosen to be the goal predicate of the program. Inductively if Π_1, Π_2 are MBT programs with goal predicates G_1, G_2, respectively, and with disjoint sets of IDB predicates (with the exception of A_k and W which are the same in all programs) then Π is the union of the rules of Π_1, Π_2 and one of the following five sets of rules.

$$\begin{cases} G(x) \longleftarrow W(x), \neg G_1(x) \\ \Pi^n_{dom} \end{cases} \qquad \begin{cases} G(x) \longleftarrow G_1(x), \neg A_k(x) \\ G(x) \longleftarrow R_k(x,y), G_1(y) \\ A_k(x) \longleftarrow R_k(x,y) \end{cases}$$

$$\{\ G(x) \longleftarrow G_1(x), G_2(x)$$

$$\begin{cases} G(x) \longleftarrow G_2(x) \\ G(x) \longleftarrow G_1(x), R_k(x,y), G(y) \end{cases} \qquad \begin{cases} G(x) \longleftarrow G_1(x), G_2(x) \\ G(x) \longleftarrow G_2(x), \neg A_k(x) \\ G(x) \longleftarrow B_k(x,x) \\ G(x) \longleftarrow G_2(x), R_k(x,y), G(y) \\ B_k(x,y) \longleftarrow G_2(x), R_k(x,y), G_2(y) \\ B_k(x,y) \longleftarrow G_2(x), R_k(x,u), B_k(u,y) \\ A_k(x) \longleftarrow R_k(x,y) \end{cases}$$

For a more succinct presentation we use the *program operators* $\overline{[\cdot]}$, $\bigwedge[\cdot,\cdot]$, $X_k[\cdot]$, $\bigcup_k[\cdot,\cdot]$ and $\widetilde{\bigcup}_k[\cdot,\cdot]$. The correspondence between the programs and the operators is shown in Figure 2. Π^n_{dom} is a convenient abbreviation of the set of rules depicted above and the superscript n indicates that the predicate symbols

$R_1, \ldots, R_n$ are used in the construction of the program. Π_1 and Π_2 are MBT programs with goal predicates G_1 and G_2 respectively. The subscript k appearing in the operators $X_k[\cdot]$, $\bigcup_k[\cdot,\cdot]$ and $\widetilde{\bigcup}_k[\cdot,\cdot]$ corresponds to R_k, e.g., operator $X_1[\cdot]$ contains R_1, $X_2[\cdot]$ contains R_2 etc. G and B_k are "fresh" predicate symbols, i.e. they must not appear in Π_1 or Π_2. In contrast, A_k may occur in Π_1 or Π_2.

Definition 5.1

- *The programs* $\begin{cases} G(x) \longleftarrow W(x) \\ \Pi^n_{dom} \end{cases}$ *and* $G(x) \longleftarrow P_i(x)$ *are MBT$_n$ programs and G is their goal predicate.*
- *If* Π_1 *and* Π_2 *are MBT$_n$ programs with goal predicates G_1 and G_2 respectively, then* $\overline{[\Pi_1]}$, $\bigwedge[\Pi_1, \Pi_2]$, $X_k[\Pi_1]$, $\bigcup_k[\Pi_1, \Pi_2]$ *and* $\widetilde{\bigcup}_k[\Pi_1, \Pi_2]$, *where* $1 \leq k \leq n$, *are also MBT$_n$ programs with goal predicate G.*
- *The class MBT is the union of the MBT$_n$ subclasses:*

$MBT = \bigcup_{n \geq 0} MBT_n$ ⊣

An MBT program will, generally, contain one or more unary G_i IDBs and, possibly, one or more binary $B_{k,j}$ IDBs. In fact the number of the latter is equal to the number of applications of the $\widetilde{\bigcup}_k$ operator. As already mentioned all these predicate symbols are different, whereas $A_1, \ldots, A_n$ and W are the same in all programs. The intuition behind $W, A_1, \ldots, A_n$ and B_k, is the following:
• $W(x)$ means that x belongs to the "domain" of the database, i.e. appears in the relations that comprise the database.
• $A_k(x)$ means that state x has at least one k-successor.
• $B_k(x, y)$ captures the notion of a k-path from state x to state y, such that G_2 holds at every state along this path. In view of the fact that G_2 corresponds to a MD-CTL formula (let's say ψ), $B_k(x, x)$ asserts the existence of a cycle having the property that ψ holds at every state of this cycle.

Note that the goal predicate is always in the last stratum and captures the meaning of the query better than any other IDB predicate. The following proposition proves that the MBD class is a fragment of stratified Datalog.

Operators used in the definition of MBT programs.

$$\Pi^n_{dom} = \begin{cases} W(x) \longleftarrow R_1(x,y) \\ W(x) \longleftarrow R_1(y,x) \\ \ldots \\ W(x) \longleftarrow R_n(x,y) \\ W(x) \longleftarrow R_n(y,x) \\ W(x) \longleftarrow P_0(x) \\ \ldots \\ W(x) \longleftarrow P_m(x) \end{cases} \qquad \overline{[\Pi_1]} = \begin{cases} G(x) \longleftarrow W(x), \neg G_1(x) \\ \Pi_1 \\ \Pi^n_{dom} \end{cases}$$

$$\bigwedge[\Pi_1, \Pi_2] = \begin{cases} G(x) \longleftarrow G_1(x), G_2(x) \\ \Pi_1 \\ \Pi_2 \end{cases}$$

$$X_k[\Pi_1] = \begin{cases} G(x) \longleftarrow G_1(x), \neg A_k(x) \\ G(x) \longleftarrow R_k(x,y), G_1(y) \\ A_k(x) \longleftarrow R_k(x,y) \\ \Pi_1 \end{cases} \qquad \bigcup_k[\Pi_1, \Pi_2] = \begin{cases} G(x) \longleftarrow G_2(x) \\ G(x) \longleftarrow G_1(x), R_k(x,y), G(y) \\ \Pi_1 \\ \Pi_2 \end{cases}$$

$$\widetilde{\bigcup}_k[\Pi_1, \Pi_2] = \begin{cases} G(x) \longleftarrow G_1(x), G_2(x) \\ G(x) \longleftarrow G_2(x), \neg A_k(x) \\ G(x) \longleftarrow B_k(x,x) \\ G(x) \longleftarrow G_2(x), R_k(x,y), G(y) \\ B_k(x,y) \longleftarrow G_2(x), R_k(x,y), G_2(y) \\ B_k(x,y) \longleftarrow G_2(x), R_k(x,u), B_k(u,y) \\ A_k(x) \longleftarrow R_k(x,y) \\ \Pi_1 \\ \Pi_2 \end{cases}$$

Figure 2: The query operators used in the definition of Multi Branching Temporal (in short MBT) programs.

Proposition 5.1 *Every MBT program is stratified.*

Proof:
Given that Π_1, Π_2 are stratified programs, any set of rules that might be added to Π_1, Π_2 according to Definition 5.1 in order to form program Π preserves the stratification of the program. ■

5.2 From formulae and structures to queries and databases

We embed MD-CTL into the class of MBT programs via a mapping $\mathbf{h} = (h_f, h_s)$ such that:

1. h_f maps MD-CTL formulae into MBT programs: given a formula φ, $h_f(\varphi)$ is a program Π with unary goal predicate G.
2. h_s maps MD-Kripke structures to relational databases, i.e. $h_s(\mathcal{K})$ is a database D.
3. This mapping is sound and complete in the following sense:

$$\varphi[\mathcal{K}] = G_\Pi(D), \text{where } \Pi = h_f(\varphi) \text{ and } D = h_s(\mathcal{K})$$

The exact mapping h_f is given below, where Π_i corresponds to the formula $\psi_i, i = 1, 2$.

Definition 5.2 *Let φ be an n-dimensional CTL formula. The MBT$_n$ program $h_f(\varphi)$ is defined recursively as follows:*

1. *If $\varphi \equiv \top$ or $\varphi \equiv p_i$, then $h_f(\varphi)$ is $\begin{cases} G(x) \longleftarrow W(x), \\ \Pi^n_{dom} \end{cases}$ and $\{\ G(x) \longleftarrow P_i(x)$ respectively.*
2. *If $\varphi \equiv \neg\psi_1$ or $\varphi \equiv \psi_1 \wedge \psi_2$, then $h_f(\varphi)$ is $\overline{[\Pi_1]}$ and $\bigwedge[\Pi_1, \Pi_2]$, respectively.*
3. *If $\varphi \equiv \mathbf{E}_k\mathbf{X}_k\psi_1$ or $\varphi \equiv \mathbf{E}_k(\psi_1\ \mathbf{U}_k\ \psi_2)$ or $\varphi \equiv \mathbf{E}_k(\psi_1\ \widetilde{\mathbf{U}}_k\ \psi_2)$, then $h_f(\varphi)$ is $X_k[\Pi_1]$, $\bigcup_k[\Pi_1, \Pi_2]$ and $\widetilde{\bigcup}_k[\Pi_1, \Pi_2]$, respectively.* ⊣

The construction of MBT programs that correspond to MD-CTL formulae can be performed very efficiently. This is formalized in Proposition 5.2. The proof is a direct consequence of Definition 5.2.

Proposition 5.2 *Given a MD-CTL formula φ, the corresponding MBT program Π is of size $O(|\varphi|)$ and is constructed in time $O(|\varphi|)$.*

Finite MD-Kripke structures can be viewed as relational databases. Definition 5.3 states formally the details of this mapping.

Definition 5.3 *Let AP be a finite set $\{p_0, \ldots, p_m\}$ of atomic propositions and let $\mathcal{K} = \langle W, R_1, \ldots, R_n, V\rangle$ be an n-dimensional Kripke structure for AP. Then $h_s(\mathcal{K})$ is the database $\langle R_1, \ldots, R_n, P_0, \ldots, P_m\rangle$, where $P_i = \{s \in W \mid p_i \in V(s)\}$ contains the states at which p_i is true $(0 \leq i \leq m)$.*

In addition, to $\mathcal{K}$ corresponds the database schema $\mathfrak{D}_\mathcal{K} = \langle W, R_1, \ldots, R_n, P_0, \ldots, P_m\rangle$, with domain the set of states W, binary predicate symbols $R_1, \ldots, R_n$ and unary predicate symbols $P_0, \ldots, P_m$. A database schema of this form is called a MD-Kripke schema. ⊣

As we have already pointed out, for simplicity we use the same notation, e.g., $R_1, \ldots, R_n, P_0, \ldots, P_m$ both for the predicate symbols and the relations. The context makes clear whether $R_1, \ldots, R_n, P_0, \ldots, P_m$ stand for predicate symbols or relations. The following proposition is a straightforward consequence of Definition 5.3.

Proposition 5.3 *A finite MD-Kripke structure $\mathcal{K}$ can be converted into a relational database $D = h_s(\mathcal{K})$ of size $O(|\mathcal{K}|)$ which can be constructed in time $O(|\mathcal{K}|)$.*

Notice that the relations $R_1, \ldots R_n$ of $h_s(\mathcal{K})$ are *total*. Moreover, every k-path $s_0, s_1, s_2, \ldots$ of $\mathcal{K}$ gives rise to the k-path $s_0, s_1, s_2, \ldots$ in $h_s(\mathcal{K})$ and vice versa: if $s_0, s_1, s_2, \ldots$ is a k-path in $h_s(\mathcal{K})$, then

$$R_k(s_i, s_{i+1}), \text{ for every } i \geq 0 \tag{3}$$

The next proposition states formally the basic property of $B_k(x, x)$.

Proposition 5.4 *$B_k(s, s)$ holds iff there exists a finite k-path $s_0, \ldots, s_r$ in D such that $s_0 = s_r = s$ and $G_2(s_i)$, for every i, $0 \leq i \leq r$.*

To prove Theorem 5.1 we need the next proposition, which is basically just a simple application of the pigeonhole principle.

Proposition 5.5 *Let $\mathcal{K} = \langle W, R_1, \ldots, R_n, V \rangle$ be a finite MD-Kripke structure and let $s_0, \ldots, s_i, \ldots, s_j, \ldots, s_r$ be a finite k-path in $\mathcal{K}$, where $r \geq |W|$. Then, there exists a state $s \in W$ such that $s_i = s_j = s$.*

We proceed to the main result of this section, which asserts that the mapping from MD-CTL formulae to the class of MBT programs is sound and complete. The proof is presented in the Appendix to facilitate the flow of the paper.

Theorem 5.1 *Let $\mathcal{K}$ be a finite n-dimensional Kripke structure and let D be the corresponding relational database. If φ is a n-dimensional CTL formula and Π its corresponding MBT_n program, then the following holds:*

$$\varphi[\mathcal{K}] = G_{\Pi}(D) \tag{4}$$

Proof: The proof can be found in the Appendix.

6 Embedding stratified Datalog into MD-CTL

In the previous section we embedded MD-CTL into the MBT class. In this section we work on the opposite direction, that is we define an embedding from MBT into MD-CTL. This is achieved via a mapping $\mathbf{f} = (f_q, f_d)$ such that:

1. f_q maps MBT programs into MD-CTL formulae, and
2. f_d maps relational databases to MD-Kripke structures.

This mapping is also sound and complete in the following sense:

$$G_{\Pi}(D) = \varphi[\mathcal{K}]$$

The correspondence of MBT programs to MD-CTL formulae is given below (subformula ψ_i corresponds to subprogram $\Pi_i, i = 1, 2$).

Definition 6.1 *Given a MBT_n program Π, $f_q(\Pi)$ is the mapping defined recursively as follows:*

1. *If* $\Pi = \left\{ \begin{array}{l} G(x) \longleftarrow W(x) \\ \Pi^n_{dom} \end{array} \right.$ *or* $\Pi = \{ \ G(x) \longleftarrow P_i(x) \ $ *, then $f_q(\Pi)$ is $\top$ and p_i, respectively.*
2. *If $\Pi = \overline{[\Pi_1]}$ or $\Pi = \bigwedge[\Pi_1, \Pi_2]$, then $f_q(\Pi)$ is $\neg\psi_1$ and $\psi_1 \wedge \psi_2$, respectively.*
3. *If $\Pi = X_k[\Pi_1]$ or $\Pi = \bigcup_k[\Pi_1, \Pi_2]$ or $\Pi = \widetilde{\bigcup}_k[\Pi_1, \Pi_2]$, then $f_q(\Pi)$ is $\mathbf{E}_k\mathbf{X}_k\psi_1$, $\mathbf{E}_k(\psi_1 \, \mathbf{U}_k \, \psi_2)$ and $\mathbf{E}_k(\psi_1 \, \widetilde{\mathbf{U}}_k \, \psi_2)$, respectively.* $\dashv$

The following proposition asserts that the construction of MD-CTL formulae that correspond to MBT programs can be performed very efficiently. The proof is an immediate consequence of Definition 6.1.

Proposition 6.1 *Given a MBT Π, the corresponding MD-CTL formula φ, which is of size $O(|\Pi|)$, can be constructed in time $O(|\Pi|)$.*

At this point, we discuss the technical challenges of this embedding. In MD-Kripke structures every accessibility relation R_k is total and as a result the corresponding relational database contains a total binary relation R_k. However, a database relation is not necessarily total. A relation that is not total may not give rise to the (infinite) paths necessary for the interpretation of MD-CTL formulae. To overcome this problem we define the *total closure* R^t_k of an arbitrary binary relation R_k with respect to a domain W as follows:

$$R^t_k = R_k \cup \{(x, x) \mid x \in W \text{ and } \nexists y \text{ such that } R_k(x, y)\} \tag{5}$$

In simple words, if R_k is not total, we can still get a total relation by adding a self loop to the states that have no successors. Note that if R_k is already total then $R^t_k = R_k$. Now a relational database can be transformed into a finite MD-Kripke structure in a meaningful way. Definition 6.2 gives the details of this transformation. Note that by a straightforward translation some Datalog rules might not be safe (i.e. they may have variables that do not occur in nonnegated body subgoals). Thus, we introduce a number of rules which essentially define the domain by an IDB predicate which is used in rules for safety.

Definition 6.2 *Let D be any database over the MD-Kripke schema $\mathfrak{D}_{\mathcal{K}} = \langle U, R_1, \ldots, R_n, P_0, \ldots, P_m \rangle$. The domain W of D is defined as $W = W_{R_x} \bigcup W_{R_y} \bigcup W_P$, where:*

$$W_{R_x} = \bigcup_{k=0}^{n} \{x \in U \mid R_k(x, y)\}$$

$$W_{R_y} = \bigcup_{k=0}^{n} \{y \in U \mid R_k(x, y)\}$$

$$W_P = \bigcup_{i=0}^{m} \{x \in U \mid P_i(x)\}$$

Let D^t be the total *database $\langle R_1^t, \ldots, R_n^t, P_0, \ldots, P_m \rangle$, where R_k^t, $1 \leq k \leq n$, is the total closure of R_k with respect to W. Then $f_d(D)$ is the finite MD-Kripke structure $\langle W, R_1^t, \ldots, R_n^t, V \rangle$, where $V(s) = \{p_i \in AP \mid P_i(s)\}$.* $\dashv$

$f_d(D)$ is well-defined because $R_1^t, \ldots, R_n^t$ are total as required by Definition 2.1. The next proposition follows directly from Definition 6.2.

Proposition 6.2 *A relational database $D = \langle R_1, \ldots, R_n, P_0, \ldots, P_m \rangle$ over a MD-Kripke schema with domain W can be transformed into a finite MD-Kripke structure $\mathcal{K} = f_d(D)$ of size $O(|W| + |R_1| + \ldots + |R_n|) = O(|D|)$ in time $O(|D|)$.*

The main result of this section is that the mapping $\mathbf{f} = (f_q, f_d)$ is sound a complete: $G_\Pi(D) = \varphi[\mathcal{K}]$, where $\varphi = f_q(\Pi)$ and $\mathcal{K} = f_d(D)$. In order to prove it we have to show that MBT programs cannot distinguish between a database D and the corresponding total database D^t, i.e. are invariant under total closure.

Theorem 6.1 *Given a MBT program Π with goal predicate G and a database D with MD-Kripke schema, the following holds:*

$$G_\Pi(D) = G_\Pi(D^t) \tag{6}$$

Proof: The proof can be found in the Appendix.

Theorem 6.2 *Let D be a relational database over a MD-Kripke schema and let $\mathcal{K}$ be the corresponding finite MD-Kripke structure. If Π is a MBT and φ its corresponding MD-CTL formula, then the following holds:*

$$G_\Pi(D) = \varphi[\mathcal{K}] \tag{7}$$

Proof:
From Theorem 6.1 we know that $G_\Pi(D) = G_\Pi(D^t)$. Further, we can show that $G_\Pi(D^t) = \varphi[\mathcal{K}]$ – the proof is similar to the proof of Theorem 5.1 and therefore is omitted. This completes the proof. ■

Theorem 6.2 proves that the mapping from MBT programs to MD-CTL formulae is sound and complete. This result in association with Theorem 5.1 leads to the following conclusion:

Theorem 6.3 *MD-CTL over finite structures has the same expressive power with the class of MBT programs.*

7 Results on Stratified Datalog

The results of Sections 5 and 6 established that MD-CTL and MBT programs have the same expressive power. In this section we build on this relation to show that the class MBT is an efficient fragment of stratified Datalog in the sense that: (1) query evaluation is linear and (2) checking satisfiability, containment and equivalence are EXPTIME-complete.

7.1 Query Evaluation

Definitions 5.2 and 6.1 in essence provide algorithms for constructing a MBT program which corresponds to a MD-CTL formula and vice versa. The importance lies in that the translation can be carried out efficiently in both directions. This is formalized by Propositions 5.2 and 6.1, which, together with Propositions 5.3 and 6.2, suggest an efficient method for performing program evaluation in this fragment. Suppose that we are given a database D (with a MD-Kripke schema), a program Π with goal G and we want to evaluate G on D, i.e. to compute $G_\Pi(D)$. This can be done as follows:

1. From Π and D construct the corresponding φ and $\mathcal{K}$ respectively. This step requires $O(|\Pi| + |D|)$ time and results in a formula φ of size $O(|\Pi|)$ and a MD-Kripke structure $\mathcal{K}$ of size $O(|D|)$.
2. Apply a model checking algorithm for $\mathcal{K}$ and φ. The algorithm will compile the truth set $\varphi[\mathcal{K}]$, i.e. the set of states of $\mathcal{K}$ on which φ is true. According to Theorem 6.2 $\varphi[\mathcal{K}]$ is exactly the outcome of the evaluation of G on D.

Since model checking algorithms for MD-CTL run in $O(|\mathcal{K}| \cdot |\varphi|)$ time (see Theorem 4.1), the following theorem holds.

Theorem 7.1 *Given a MBT program Π with goal G and a database D, the evaluation of G on D can be done in $O(|D||\Pi|)$ time.*

The above result establishes that for the class MBT the query evaluation has linear program and data complexity.

7.2 Satisfiability

In this section we show that checking the satisfiability of a MBT program can be reduced to checking the satisfiability of a MD-CTL formula. We begin by stating Theorem 7.2 which is a straightforward extension of Theorem 4.3, and on which we build later to argue about the satisfiability of MBT programs.

Theorem 7.2 (*Satisfiability*) *The satisfiability problem for MD-CTL is EXPTIME–complete.*

Definition 7.1 (*Satisfiability for Datalog programs*) *An IDB predicate G of a program Π is* satisfiable *if there exists a database D, such that $G_\Pi(D) \neq \emptyset$.* ⊣

Proposition 7.1 *Let Π be a MBT program with goal predicate G and let φ be the corresponding MD-CTL formula; φ is satisfiable iff G is satisfiable.*

Proof:
($\Rightarrow$) Suppose that φ is satisfiable; then there exists a MD-Kripke structure $\mathcal{K} = \langle W, R_1, \ldots R_n, V\rangle$, such that $\mathcal{K}, s \models \varphi$, for some $s \in W$. If $\mathcal{K}$ is finite, then by Theorem 5.1 we obtain that $s \in G_\Pi(D)$, where D is the database that corresponds to $\mathcal{K}$. If $\mathcal{K}$ is infinite, then by Theorem 4.2 there exists a finite MD-Kripke structure $\mathcal{K}_f = \langle W_f, R_{f_1}, \ldots, R_{f_n}, V_f\rangle$ such that $\mathcal{K}_f, s' \models \varphi$, for some $s' \in W_f$. Invoking again Theorem 5.1 we derive that $s' \in G_\Pi(D)$, where D is the database that corresponds to $\mathcal{K}_f$. We conclude that in both cases G is satisfiable.
($\Leftarrow$) Suppose now that G is satisfiable. This means that there exists a database $D = \langle R_1, \ldots, R_n, P_0, \ldots, P_m\rangle$ over a MD-Kripke schema with domain W, such that $G_\Pi(D) \neq \emptyset$. Hence, by Theorem 6.2 we obtain that $\varphi[\mathcal{K}] \neq \emptyset$, where $\mathcal{K}$ is the finite MD-Kripke structure that corresponds to D. This of course implies that there exists a state $s \in W$ such that $\mathcal{K}, s \models \varphi$, i.e. φ is satisfiable. ■

Proposition 7.1 proves that the problem of checking the satisfiability of the unary goal predicates of MBT programs is decidable. The following proposition deals with the case of the binary $B_k(x, y)$ predicates.

Proposition 7.2 *Let Π be a MBT program and let B_k be a binary IDB predicate of Π. Deciding the satisfiability of B_k can be reduced to deciding the satisfiability of a goal predicate G of another MBT program in polynomial time.*

Proof:
If Π contains a binary IDB predicate $B_k(x, y)$, then it contains a subprogram $\Pi' = \widetilde{\bigcup}_k[\Pi_1, \Pi_2]$. Let φ, ψ_1 and ψ_2 be the MD-CTL formulae corresponding to Π', Π_1 and Π_2. According to Definition 6.1, $\varphi \equiv \mathbf{E}_k(\psi_1 \,\widetilde{\mathbf{U}}_k\, \psi_2)$. Let us consider now the MD-CTL formula $\varphi'' \equiv \varphi \wedge \neg\mathbf{E}_k(\top\, \mathbf{U}_k\, \psi_1)$. Let Π'' be the MBT program corresponding to φ'' and let G be the goal predicate of Π''. But then B_k is satisfiable iff G is satisfiable. Finally, it is easy to see that the above reduction takes place in polynomial time.

In order to complete the proof we have to argue about the satisfiability of every IDB predicate. A MBT program may contain the following IDB predicates: W, A_k, G_i and B_k . The first two predicates are always satisfiable. Propositions 7.1 and 7.2 handle the remaining two predicates by reducing their satisfiability to the satisfiability of the appropriate MD-CTL formulae.

The following theorem is an immediate consequence of Theorem 7.2 and Propositions 7.1 and 7.2. ■

Theorem 7.3 *Checking satisfiability for MBT programs is EXPTIME–complete.*

7.3 Containment

The problem of checking the containment of MBT programs can be reduced to that of checking the implication of MD-CTL formulae. First we give some basic definitions regarding the notion of containment for Datalog programs and MD-CTL formulae.

Definition 7.2 (*Containment of Datalog queries*) *Given two Datalog queries* Π_1 *and* Π_2 *with goal predicates* G_1 *and* G_2*, we say that* Π_1 *is* contained *in* Π_2*, denoted* $\Pi_1 \sqsubseteq \Pi_2$*, if and only if for every database* D*,* $G_{1_{\Pi_1}}(D) \subseteq G_{2_{\Pi_2}}(D)$*.* Π_1 *and* Π_2 *are* equivalent*, denoted* $\Pi_1 \equiv \Pi_2$*, if* $\Pi_1 \sqsubseteq \Pi_2$ *and* $\Pi_2 \sqsubseteq \Pi_1$*.* ⊣

A notion of containment for MD-CTL formulae can also be cast in terms of truth sets.

Definition 7.3 (*Containment of MD-CTL formulae*) *Given two MD-CTL formulae* φ_1 *and* φ_2*, we say that* φ_1 *is* contained *in* φ_2*, denoted* $\varphi_1 \sqsubseteq \varphi_2$*, if and only if for every* finite *MD-Kripke structure* $\mathcal{K}$*,* $\varphi_1[\mathcal{K}] \subseteq \varphi_2[\mathcal{K}]$*.* ⊣

Suppose we have two MD-CTL formulae φ_1 and φ_2; we say that φ_1 *implies* φ_2 if for every MD-Kripke structure $\mathcal{K} = \langle W, R_1, \ldots, R_n, V\rangle$ and for every $s \in W$, $\mathcal{K}, s \models \varphi_1$ implies that $\mathcal{K}, s \models \varphi_2$. If φ_1 implies φ_2, then formula $\varphi_1 \rightarrow \varphi_2$ is valid and vice versa. Hence, we can use the notation $\models \varphi_1 \rightarrow \varphi_2$ to assert that φ_1 implies φ_2. The following corollary follows directly from Theorem 4.3.

Corollary 7.1 (*Implication*) *The problem of deciding whether a MD-CTL formula* φ_1 *implies a MD-CTL formula* φ_2 *is EXPTIME–complete.*

Note that it is sufficient to consider only finite structures because as we explained in Section 4.2 MD-CTL has the bounded model property.

Proposition 7.3 *Given two MD-CTL formulae* φ_1 *and* φ_2 *the following are equivalent:*
1. $\varphi_1 \sqsubseteq \varphi_2$
2. $\models \varphi_1 \rightarrow \varphi_2$

Proof:
$(1 \Rightarrow 2)$ $\varphi_1 \sqsubseteq \varphi_2$ means that for every finite MD-Kripke structure $\mathcal{K} = \langle W, R_1, \ldots, R_n, V\rangle$, $\varphi_1(\mathcal{K}) \subseteq \varphi_2(\mathcal{K})$. This implies that if $s \in \varphi_1(\mathcal{K})$, then $s \in \varphi_2(\mathcal{K})$. Therefore, for every finite MD-Kripke structure $\mathcal{K} = \langle W, R_1, \ldots, R_n, V\rangle$ and for every $s \in W$, $\mathcal{K}, s \models \varphi_1$ implies $\mathcal{K}, s \models \varphi_2$, i.e.$\models_f \varphi_1 \rightarrow \varphi_2$.

It remains to consider the infinite case; we will prove that the next two assertions are equivalent:
(a) $\models_f \varphi_1 \rightarrow \varphi_2$
(b) $\models \varphi_1 \rightarrow \varphi_2$

It is obvious that (b) implies (a). To show that (a) also implies (b) let us assume to the contrary that (a) holds and (b) does not. This means that $\varphi_1 \wedge \neg\varphi_2$ is satisfiable, i.e. it has a model $\mathcal{K}$. $\mathcal{K}$ cannot be finite because of (a). It must, therefore, be infinite. Then, from Theorem 4.2 we obtain that $\varphi_1 \wedge \neg\varphi_2$ has a finite model $\mathcal{K}_f$, which is absurd because of (a).
$(2 \Rightarrow 1)$ $\models \varphi_1 \rightarrow \varphi_2$ means that for every MD-Kripke structure $\mathcal{K} = \langle W, R_1, \ldots, R_n, V\rangle$, $\mathcal{K} \models \varphi_1 \rightarrow \varphi_2$. Consequently, for every $s \in W$, $\mathcal{K}, s \models \varphi$ implies that $\mathcal{K}, s \models \psi$, or in other words, $\varphi_1(\mathcal{K}) \subseteq \varphi_2(\mathcal{K})$. Thus, $\varphi_1 \sqsubseteq \varphi_2$. ■

The following theorem is a direct consequence of Corollary 7.1 and Proposition 7.3.

Theorem 7.4 *Checking containment for MD-CTL formulae is EXPTIME–complete.*

Theorem 7.5 proves the EXPTIME–completeness of the containment problem for MBT programs.

Theorem 7.5 *Checking containment for MBT programs is EXPTIME–complete.*

Proof:
Let Π_1, Π_2 be MBT programs with goal predicates G_1, G_2 and let φ_1, φ_2 be the corresponding MD-CTL formulae. We shall prove that $\Pi_1 \sqsubseteq \Pi_2$ iff $\varphi_1 \sqsubseteq \varphi_2$.
$(\Rightarrow)$ Suppose that $\Pi_1 \sqsubseteq \Pi_2$, but it is not the case that $\varphi_1 \sqsubseteq \varphi_2$, i.e. there exists a finite MD-Kripke structure $\mathcal{K}'$ such that $\varphi_1[\mathcal{K}'] \not\subseteq \varphi_2[\mathcal{K}']$. Let D' be the database that corresponds to $\mathcal{K}'$; then by Theorem 5.1 we get that $G_{1_{\Pi_1}}(D') \not\subseteq G_{2_{\Pi_2}}(D')$. But this is absurd because the fact that $\Pi_1 \sqsubseteq \Pi_2$ implies that for every database D, $G_{1_{\Pi_1}}(D) \subseteq G_{2_{\Pi_2}}(D)$. Thus, it must be the case that $\varphi_1 \sqsubseteq \varphi_2$.
$(\Leftarrow)$ Suppose that $\varphi_1 \sqsubseteq \varphi_2$, but it is not the case that $\Pi_1 \sqsubseteq \Pi_2$, i.e. there exists a database D' such that

$G_{1_{\Pi_1}}(D') \not\subseteq G_{2_{\Pi_2}}(D')$. Let $\mathcal{K}'$ be the finite MD-Kripke structure that corresponds to D'. Theorem 6.2 then implies that $\varphi_1[\mathcal{K}'] \not\subseteq \varphi_2[\mathcal{K}']$, which is absurd because $\varphi_1 \sqsubseteq \varphi_2$ means that for every finite MD-Kripke structure $\mathcal{K}$, $\varphi_1(\mathcal{K}) \subseteq \varphi_2(\mathcal{K})$. Hence, it must be the case that $\Pi_1 \sqsubseteq \Pi_2$. ■

The following theorem is an immediate consequence of Theorem 7.5.

Theorem 7.6 *Checking equivalence of MBT programs is EXPTIME–complete.*

8 Conclusions

In this work we introduced the *multidimensional* computation tree logic (MD-CTL) and we proved that the "nice" properties of CTL (linear model checking and bounded model property) transfer also to MD-CTL. We exploited these properties to establish results on stratified Datalog for which not much work exists in the literature. In particular, we defined a fragment of stratified Datalog called the class of Multi Branching Temporal (MBT) programs that has the same expressive power with MD-CTL. To prove that we devised a translation from both directions between MD-CTL formulae and MBT programs by giving the exact translation rules. We further built on this relation to prove that checking satisfiability, containment and equivalence problems are EXPTIME–complete for MBT programs. The MBT class is the largest fragment of stratified Datalog for which these problems are known to be decidable. We also proved that query evaluation is linear.

In future work we plan to extend our approach to CTL* (Full Branching Time Logic) [22, 19]. CTL is a proper and less expressive fragment of CTL*. Although we believe that the extension is feasible, having considered and investigated the problem for a short time, we think that the translation of CTL* will introduce additional non-trivial complications. Another future direction is to investigate larger fragments of stratified Datalog for which query containment, equivalence, satisfiability and evaluation are decidable.

References

[1] F. Afrati, T. Andronikos, V. Pavlaki, E. Foustoucos, and I. Guessarian. From CTL to Datalog. In *ACM, Principles of Computing and Knowledge: Paris C. Kanellakis Memorial Workshop*, 2003.

[2] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. *In Proc. Work. on Foundations of Deductive Databases and Logic Programming*, pages 89–148, 1988.

[3] F. Afrati, S. Cosmadakis, and M. Yannakakis. On Datalog vs. Polynomial Time. In *ACM PODS*, pages 13–25, 1991.

[4] Loredana Afanasiev. XML query evaluation via CTL model checking. Master's thesis, Department of Sciences University Gabriele d' Annunzio (Pescara), 2004.

[5] L. Afanasiev, M. Franceschet, M. Marx, and M. de Rijke. CTL model checking for processing simple XPath queries. In *Proc. of Temporal Representation and Reasoning TIME*, 2004.

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

[7] M. Ben-Ari, J. Y. Halpern, and A. Pnueli. Deterministic propositional dymanic logic: Finite models, complexity and completeness. *JCSS*, 25(3):402–417, 1982.

[8] B. Cui, Y. Dong, X. Du, K. N. Kumar, C. R. Ramakrishnan, I. V. Ramakrishnan, A. Roychoudhury, S. A. Smolka, and D. S. Warren. Logic programming and model checking. In *PLIP/ALP'98*, volume 1490, pages 1–20. LNCS, Springer, 1998.

[9] E. M. Clarke and E. A. Emerson. Design and verification of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logics of Programs*, volume 131, pages 52–71. LNCS, Springer, 1981.

[10] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[11] E. M. Clarke, O. Grumberg, and D. Long. Verification tools for finitestate concurrent systems. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, volume Lecture Notes in Computer Science, pages 124–175. SpringerVerlag, 1993.

[12] A. Chandra and D. Harel. Horn clauses queries and generalizations. *In Journal of Logic Programming*, 2(1):1–15, 1985.

[13] W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In *Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, pages 358–375, 1998.

[14] E. M. Clarke and J. M. Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[15] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[16] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of paralled programs as fixpoints. In *Int. Colloq. Automata Lang. and Programming*, pages 169–181, 1980.

[17] E.A. Emerson and E.M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[18] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.

[19] E. A. Emerson and J. Y. Halpern. Sometimes and not never revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[20] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonic reasoning. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 364–375, 1997.

[21] E. A. Emerson. *Temporal and modal logic*. Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, 1990.

[22] E. A. Emerson and A. P. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, 1984.

[23] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18(2):194–211, 1979.

[24] I. Guessarian, E. Foustoucos, T. Andronikos, and F. Afrati. On temporal logic versus datalog. *Theoretical Computer Science*, 303(1):103–133, 2003.

[25] G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Transactions on Computational Logic*, 3(1):42–79, 2002.

[26] G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *LICS*, 2002.

[27] A. Halevy, I. S. Mumick, Y. Sagiv, and O. Shmueli. Static analysis in datalog extensions. *Journal of the ACM*, 48(5):971–1012, 2001.

[28] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1–3):86–104, 1986.

[29] P. C. Kolaitis. The expressive power of stratified logic programs. *Information and Computation*, 90(1):50–66, 1991.

[30] Apt K. and J. M. Pugin. Maintenance of stratified databases viewed as a belief revision system. In *Proc. PODS*, pages 136–145, 1987.

[31] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[32] L. Lamport. Sometimes is sometimes "not never": on the temporal logic of programs. In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 174–185, 1980.

[33] V. Lifschitz. On the declarative semantics of logic programs with negation. *Foundations of Deductive Databases and Logic Programming*, pages 177–192, 1988.

[34] J. L. Lloyd. *Foundations of Logic Programming*. Berlin: Springer, 1987.

[35] A. Levy, I. S. Mumick, Y. Sagiv, and O. Shmueli. Equivalence, query-reachability, and satisfiability in Datalog extensions. In *ACM PODS*, pages 109–122, 1993.

[36] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *PODS*, 2002.

[37] K. Morris, J. D. Ullman, and A. Van Gelder. Design overview of the NAIL! system. In *Third International Conference on Logic Programming*, volume 225, pages 554–568. Springer-Verlag, 1986.

[38] S. Naqvi and S. Tsur. *A Logic Language for Data and Knowledge Bases*. Computer Science Press, 1989.

[39] T. Przymusinski. On the declarative semantics of deductive databases and logic programs. *Foundations of Deductive Databases and Logic Programming*, pages 193–216, 1988.

[40] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *Proceedings of the 9th International Conference on Computer Aided Verification*, pages 143–154, 1997.

[41] R. Ramakrisnhan, D. Srivastava, and S. Sudarshan. CORAL Control, Relations and Logic. In *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 238–250, 1992.

[42] Ph. Schnoebelen. The complexity of temporal logic model checking. In *Advances in Modal Logic, vol, 4. King's College Publications.*, pages 437–459, 2003.

[43] J. D. Ullman. *Principles of Databases and Knowledge Base Systems, Vol. I and II*. Computer Science Press, 1988.

[44] M. Y. Vardi. The complexity of relational query languages. In *ACM STOC*, pages 137–146, 1982.

[45] M.Y. Vardi. Why is modal logic so robustly decidable?. In *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184, 1997.

[46] M. Y. Vardi. Sometimes and not never re-revisited: on branching vs. linear time. In *Proc. 9th Intl Conf. on Concurrency Theory, D. Sangiorgi and R. de Simone (eds.), Springer-Verlag, Lecture Notes in Computer Science 1466*, pages 1–17, September 1998.

[47] M. Y. Vardi. Branching vs. linear time: Final showdown. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–22, 2001.

[48] A. Van Gelder. Negation as failure using tight derivations for general logic programs. *Foundations of deductive databases and logic programming*, pages 149–176, 1988.

[49] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1rst Symp. on Logic in Computer Science*, pages 322–331, 1986.

[50] C. Zaniolo, S. Ceri, Ch. Faloutsos, R. Snodgrass, V. S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann Publishers, Inc. San Fransisco California, 1997.

[51] S. Zhang, O. Sokolsky, and S. A. Smolka. On the parallel complexity of model checking in the Modal Mu-Calculus. In *Proceedings of Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 154–163, 1994.

9 Appendix: Proofs

9.1 Proof of Theorem 5.1

We prove that (4) holds by induction on the structure of φ. To increase the readability of the proof, we use the subscripts in the goal predicates to denote the corresponding formula. For instance, we write $G_{\mathbf{E}_k\mathbf{X}_k\psi}$ to denote that G is the goal predicate of the program corresponding to $\mathbf{E}_k\mathbf{X}_k\psi$.

1. If $\varphi \equiv \top$ or $\varphi \equiv p$, where $p \in AP$, then the corresponding programs are those of Definition 5.2.(1):
   - $(\Rightarrow)$ $\mathcal{K}, s \models \top \Rightarrow s \in W \Rightarrow$ (by the totality of the accessibility relations) there exists $t \in W$ such that $(s,t) \in R_k$ for some k $(1 \leq k \leq n) \Rightarrow s \in W_{\Pi^n_{dom}}(D) \Rightarrow s \in G_\top(D)$.
     $(\Leftarrow)$ $s \in G_\top(D) \Rightarrow s \in W_{\Pi^n_{dom}}(D) \Rightarrow s$ appears in one of $R_1, \ldots, R_n, P_0, \ldots, P_n \Rightarrow s \in W \Rightarrow \mathcal{K}, s \models \top$.
   - $\mathcal{K}, s \models p \Leftrightarrow p \in V(s) \Leftrightarrow P(s)$ is a ground fact of $D \Leftrightarrow s \in G_p(D)$.
2. If $\varphi \equiv \neg\psi$ or $\varphi \equiv \psi_1 \wedge \psi_2$, then the corresponding programs are shown in Definition 5.2.(2).
   - $\neg$: $(\Rightarrow)$ $\mathcal{K}, s \models \varphi \Rightarrow \mathcal{K}, s \models \neg\psi \Rightarrow \mathcal{K}, s \not\models \psi \Rightarrow$ (by the induction hypothesis) $s \notin G_\psi(D) \Rightarrow s \in G_\varphi(D)$.
     $(\Leftarrow)$ $s \in G_\varphi(D) \Rightarrow s \notin G_\psi(D) \Rightarrow$ (by the induction hypothesis) $\mathcal{K}, s \not\models \psi \Rightarrow \mathcal{K}, s \models \neg\psi \Rightarrow \mathcal{K}, s \models \varphi$.
   - $\wedge$: $(\Rightarrow)$ $\mathcal{K}, s \models \varphi \Rightarrow \mathcal{K}, s \models \psi_1$ and $\mathcal{K}, s \models \psi_2 \Rightarrow$ (by the induction hypothesis) $s \in G_{\psi_1}(D)$ and $s \in G_{\psi_2}(D) \Rightarrow s \in G_{\psi_1}(D) \cap G_{\psi_2}(D) \Rightarrow s \in G_\varphi(D)$.
     $(\Leftarrow)$ $s \in G_\varphi(D) \Rightarrow s \in G_{\psi_1}(D) \cap G_{\psi_2}(D) \Rightarrow s \in G_{\psi_1}(D)$ and $s \in G_{\psi_2}(D) \Rightarrow$ (by the induction hypothesis) $\mathcal{K}, s \models \psi_1$ and $\mathcal{K}, s \models \psi_2 \Rightarrow \mathcal{K}, s \models \varphi$.
3. If $\varphi \equiv \mathbf{E}_k\mathbf{X}_k\psi$, then the corresponding program is that of Definition 5.2.(3).
   - $(\Rightarrow)$ $\mathcal{K}, s \models \mathbf{E}_k\mathbf{X}_k\psi \Rightarrow$ there exists a k-path $\pi = s_0, s_1, s_2, \ldots$ with initial state $s_0 = s$, such that $\mathcal{K}, \pi \models \mathbf{X}_k\psi \Rightarrow \mathcal{K}, \pi^1 \models \psi$ for the path $\pi^1 = s_1, s_2, \ldots \Rightarrow \mathcal{K}, s_1 \models \psi \Rightarrow$ (by the induction hypothesis) $s_1 \in G_\psi(D)$. Furthermore, from (3) we know that $R_k(s_0, s_1)$ holds. From the second rule of Π_φ, by combining $G_\psi(s_1)$ with $R_k(s_0, s_1)$, we derive $G_\varphi(s_0)$ and, thus, $s_0 \in G_\varphi(D)$.
   - $(\Leftarrow)$ Let us assume that $s \in G_\varphi(D)$. In this case the database relation R_k is total. Hence, $s \in A_k(D)$ and the first rule does not add new states to $G_\varphi(D)$. From the rules of program Π_φ there exists a s_1 such that $R_k(s, s_1)$ and $G_\psi(s_1)$ hold. By the induction hypothesis we get $\mathcal{K}, s_1 \models \psi$. Let $\pi = s_0, s_1, s_2, \ldots$ be any k-path with initial state $s_0 = s$ and second state s_1. Clearly, then $\mathcal{K}, \pi^1 \models \psi \Rightarrow \mathcal{K}, \pi \models \mathbf{X}_k\psi \Rightarrow \mathcal{K}, s \models \varphi$.
4. If $\varphi \equiv \mathbf{E}_k(\psi_1 \, \mathbf{U}_k \, \psi_2)$, then the corresponding program is that of Definition 5.2.(3).
   - $(\Rightarrow)$ $\mathcal{K}, s \models \mathbf{E}_k(\psi_1 \, \mathbf{U}_k \, \psi_2) \Rightarrow$ there exists a k-path $\pi = s_0, s_1, s_2, \ldots$ with initial state $s_0 = s$, such that $\mathcal{K}, \pi^i \models \psi_2$ and $\mathcal{K}, \pi^j \models \psi_1 \Rightarrow \mathcal{K}, s_i \models \psi_2$ and $\mathcal{K}, s_j \models \psi_1$ $(0 \leq j \leq i-1) \Rightarrow s_i \in G_{\psi_2}(D)$ and $s_j \in G_{\psi_1}(D)$ $(0 \leq j \leq i-1)$ (by the induction hypothesis). From (3) we know that $R_k(s_r, s_{r+1})$, $0 \leq r < i$. From the first rule of Π_φ : $G_\varphi(x) \longleftarrow G_{\psi_2}(x)$ we derive that $G_\varphi(x)$. Successive applications of the second rule of Π_φ : $G_\varphi(x) \longleftarrow G_{\psi_1}(x)$, $R_k(x,y)$, $G_\varphi(y)$ yield $G_\varphi(s_{i-1})$, $G_\varphi(s_{i-2})$, $\ldots$, $G_\varphi(s_1)$, $G_\varphi(s_0)$. Thus, $s_0 \in G_\varphi(D)$.
   - $(\Leftarrow)$ For the inverse direction, suppose that $s \in G_\varphi(D)$. From the rules of program Π_φ there exists a state s_i (possibly $s_i = s$) such that $G_{\psi_2}(s_i)$ holds. In addition, there exists a sequence of states $s_0 = s, s_1, \ldots, s_i$ such that $R_k(s_r, s_{r+1})$ and $G_{\psi_1}(s_r)$ $(0 \leq r < i)$. By the induction hypothesis we get that $\mathcal{K}, s_i \models \psi_2$ and $\mathcal{K}, s_j \models \psi_1$ $(0 \leq j \leq i-1)$. Let $\pi = s_0, s_1, s_2, \ldots, s_i, \ldots$ be any k-path with initial segment $s_0, s_1, \ldots, s_i$. Then, $\mathcal{K}, \pi^i \models \psi_2$ and $\mathcal{K}, \pi^j \models \psi_1$ $(0 \leq j \leq i-1)$, i.e. $\mathcal{K}, \pi \models \varphi$.
5. If $\varphi \equiv \mathbf{E}_k(\psi_1 \, \widetilde{\mathbf{U}}_k \, \psi_2)$, then the corresponding program is that of Definition 5.2.(3).
   - $(\Rightarrow)$ Recall from Section 2 that $\mathcal{K}, s \models \mathbf{E}_k(\psi_1 \, \widetilde{\mathbf{U}}_k \, \psi_2)$ means that there exists a k-path $\pi = s_0, s_1, s_2, \ldots$ with initial state $s_0 = s$, such that either (1) $\mathcal{K}, \pi^i \models \psi_2$, for every $i \geq 0$, or (2) $\mathcal{K}, \pi^i \models \psi_1 \wedge \psi_2$ and $\mathcal{K}, \pi^j \models \psi_2$, $0 \leq j \leq i-1$. We examine both cases:
     (a) In the first case $\mathcal{K}, s_i \models \psi_2$, for every $i \geq 0$. The induction hypothesis gives that $s_i \in G_{\psi_2}(D)$, for every $i \geq 0$. Let $s_0, s_1, s_2, \ldots, s_r$ be an initial segment of π, with $n \geq |W|$. From Proposition

5.5 we know that in the aforementioned sequence there exists a state t such that $t = s_k = s_l$, $0 \leq k < l \leq r$. Then Proposition 5.4 implies that $(s_k, s_k) \in B_k(D)$. From the third rule of Π_φ: $G_\varphi(x) \longleftarrow B_k(x,x)$, we derive that $G_\varphi(s_k)$. Successive applications of the rule $G_\varphi(x) \longleftarrow G_{\psi_2}(x), R_k(x,y), G_\varphi(y)$ yield $G_\varphi(s_{k-1})$, $G_\varphi(s_{k-2})$, ..., $G_\varphi(s_1)$, $G_\varphi(s_0)$. Accordingly, $s_0 \in G_\varphi(D)$.
(b) In the second case, we know that $\mathcal{K}, s_i \models \psi_1 \wedge \psi_2$ and $\mathcal{K}, s_j \models \psi_2$, $0 \leq j \leq i-1$. By the induction hypothesis we get that $s_i \in G_{\psi_1}(D)$ and $s_j \in G_{\psi_2}(D)$, $0 \leq j \leq i$. From the first rule of Π_φ: $G_\varphi(x) \longleftarrow G_{\psi_1}(x), G_{\psi_2}(x)$, we derive that $G_\varphi(s_i)$. Successive applications of the fourth rule of Π_φ: $G_\varphi(x) \longleftarrow G_{\psi_2}(x)$, $R_k(x,y)$, $G_\varphi(y)$ yield $G_\varphi(s_{i-1})$, $G_\varphi(s_{i-2})$, ..., $G_\varphi(s_1)$, $G_\varphi(s_0)$. Therefore, $s_0 \in G_\varphi(D)$.

($\Leftarrow$) For the inverse direction, suppose that $s_0 \in G_\varphi(D)$. We define $G_\varphi(D,r)$ to be the set of ground facts of the IDB predicate G_φ that have been computed during the first r rounds of the evaluation of the last stratum of the program Π_φ. We shall prove that for every $t \in G_\varphi(D,r)$, there exists a k-path $\pi = t_0, t_1, t_2, \ldots$ with initial state $t_0 = t$, such that $\mathcal{K}, \pi \models \varphi$. We use induction on the number of rounds r.

(a) If $r = 1$, then t must appear either due to the first rule of Π_φ: $G_\varphi(x) \longleftarrow G_{\psi_1}(x), G_{\psi_2}(x)$ or due to the third rule of Π_φ: $G_\varphi(x) \longleftarrow B_k(x,x)$ if the IDB predicate B_k belongs to a previous stratum. The database relation R_k is total, meaning that $t \in A_{k_\varphi}(D)$, and, thus, t could not have appeared from an application of the second rule. Note that if B_k is in the last stratum, then, of course, t could not have appeared due to the third rule. In the former case $t \in G_{\psi_1}(D) \cap G_{\psi_2}(D)$; the induction hypothesis for ψ_1 and ψ_2 means that $\mathcal{K}, t \models \psi_1 \wedge \psi_2$, which immediately implies that $\mathcal{K}, \pi \models \varphi$ for any k-path $\pi = t_0, t_1, t_2, \ldots$ with initial state $t_0 = t$. In the latter case, $(t,t) \in B_{k_\varphi}(D)$ and, in view of Proposition 5.4, this implies the existence of a finite sequence $t_0, t_1, \ldots, t_l$, such that $t_0 = t_l = t$ and $\mathcal{K}, t_j \models \psi_2$, $0 \leq j \leq l$. Consider the path $\pi = (t_0, t_1, \ldots, t_l)^\omega$; for this path we have $\mathcal{K}, \pi \models \varphi$.

(b) We show now that the claim holds for $r+1$, assuming that it holds for r. Suppose that t first appeared in $G_\varphi(D, r+1)$ during round $r+1$. This could have happened either because of the third rule: $G_\varphi(x) \longleftarrow B_k(x,x)$ or because of the fourth rule: $G_\varphi(x) \longleftarrow G_{\psi_2}(x), R_k(x,y), G_\varphi(y)$.
In the first case $(t,t) \in B_{k_\varphi}(D)$. Then Proposition 5.4 asserts the existence of a finite sequence $t_0, t_1, \ldots, t_l$ of states, such that $t_0 = t_l = t$ and $\mathcal{K}, t_j \models \psi_2$, $0 \leq j \leq l$. Consider the path $\pi = (t_0, t_1, \ldots, t_l)^\omega$; for this path we have $\mathcal{K}, \pi \models \varphi$.
In the second case, we know that $G_{\psi_2}(t)$ and that there exists a t_1 such that $R_k(t, t_1)$ and $G_\varphi(t_1)$. By the induction hypothesis, we get that $\mathcal{K}, t \models \psi_2$ and that $\mathcal{K}, t_1 \models \varphi$. Immediately then we conclude that $\mathcal{K}, \pi \models \varphi$, for the k-path $\pi = t_0, t_1, t_2, \ldots$ with $t_0 = t$. ■

9.2 Proof of Theorem 6.1

We prove that (6) holds by induction on the structure of the program Π.

1. If $\Pi = \begin{cases} G(x) \longleftarrow W(x) \\ \Pi^n_{dom} \end{cases}$, then:
($\Rightarrow$) $s \in W_{\Pi^n_{dom}}(D) \Rightarrow D$ contains a ground fact of the form $P_i(s)$ or $R_k(s,t)$ or $R_k(t,s)$, for some k, $1 \leq k \leq n$. Obviously, D^t also contains this ground fact, which means that $s \in W_{\Pi^n_{dom}}(D^t)$.
($\Leftarrow$) $s \in W_{\Pi^n_{dom}}(D^t) \Rightarrow D^t$ contains a ground fact of the form $P_i(s)$ or $R_k(s,t)$ or $R_k(t,s)$, or $R_k(s,s)$, for some k, $1 \leq k \leq n$. In the first three cases D also contains this ground fact; however, D may not contain a ground fact of D^t that has the form $R_k(s,s)$. If D does not contain $R_k(s,s)$, this implies (recall the definition of R^t_k) that D contains a fact $P_i(s)$ or a fact $R_k(t,s)$ for some constant t, but does not contain any fact of the form $R_k(s,u)$. But then we would have that $s \in W_{\Pi^n_{dom}}(D)$ due to $P_i(s)$ or $R_k(t,s)$.

2. If $\Pi = \{ \; G(x) \longleftarrow P_i(x) \;$, then $s \in G_\Pi(D) \Leftrightarrow P_i(s)$ is a ground fact of $D \Leftrightarrow P_i(s)$ is a ground fact of D^t $\Leftrightarrow s \in G_\Pi(D^t)$.

3. If $\Pi = \overline{[\Pi_1]}$, then $s \in G_\Pi(D) \Leftrightarrow s \in W_{\Pi^n_{dom}}(D)$ and $s \notin G_{1_{\Pi_1}}(D)$. Reasoning as above we conclude that $s \in W_{\Pi^n_{dom}}(D) \Leftrightarrow s \in W_{\Pi^n_{dom}}(D^t)$. Furthermore, by the induction hypothesis with respect to Π_1, we get that $s \in G_{1_{\Pi_1}}(D) \Leftrightarrow s \in G_{1_{\Pi_1}}(D^t)$. Hence, $s \in G_\Pi(D) \Leftrightarrow s \in G_\Pi(D^t)$.

4. If $\Pi = \bigwedge[\Pi_1, \Pi_2]$, then $s \in G_\Pi(D) \Leftrightarrow s \in G_{1_{\Pi_1}}(D)$ and $s \in G_{2_{\Pi_2}}(D) \Leftrightarrow$ (by the induction hypothesis) $s \in G_{1_{\Pi_1}}(D^t)$ and $s \in G_{2_{\Pi_2}}(D^t) \Leftrightarrow s \in G_\Pi(D^t)$.

5. If $\Pi = X_k[\Pi_1]$, then:
($\Rightarrow$) Suppose that $s \in G_\Pi(D)$; this is a result of either the first or the second rule of Π. If it is due to the first rule, then $s \in G_{1_{\Pi_1}}(D)$ and D does not contain a ground fact of the form $R_k(s,u)$, for any constant u. If it is due to the second rule, D contains a ground fact $R_k(s,u)$, for some constant u, and $u \in G_{1_{\Pi_1}}(D)$. In the former case, the induction hypothesis implies that $s \in G_{1_{\Pi_1}}(D^t)$. Moreover, by construction D^t contains the ground fact $R_k(s,s)$. Hence, $s \in G_\Pi(D^t)$ because of the second rule of Π. In the latter case, the induction hypothesis implies that $u \in G_{1_{\Pi_1}}(D^t)$. Taking into account that D^t contains $R_k(s,u)$, we conclude that $s \in G_\Pi(D^t)$ because of the second rule of Π.
($\Leftarrow$) Suppose that $s \in G_\Pi(D^t)$. Let us assume for a moment that s appears in $G_\Pi(D^t)$ due to an application of the first rule of Π. This would imply that $s \notin A_{k_\Pi}(D^t)$. But this is absurd because R_k^t is total by construction (i.e. $\forall s \exists u R_k(s,u)$) meaning that $s \in A_{k_\Pi}(D^t)$. This shows that when evaluating Π on "total" databases, such as D^t, the first rule of Π is redundant. Hence, s must appear in $G_\Pi(D^t)$ as a result of an application of the second rule of Π. This means that D^t contains a ground fact of the form $R_k(s,u)$, for some constant u (possibly $s = u$), and $u \in G_{1_{\Pi_1}}(D^t)$. The induction hypothesis gives that $u \in G_{1_{\Pi_1}}(D)$ $(*)$. If D contains the ground fact $R_k(s,u)$, then $s \in G_\Pi(D)$ due to the second rule of Π. If however D does not contain the ground fact $R_k(s,u)$, then by the definition of R_k^t we deduce that: (a) D contains no ground fact of the form $R_k(s,v)$, for any v, meaning that $s \notin A_{k_\Pi}(D)$ $(**)$ and (b) the ground fact in D^t is actually $R_k(s,s)$, i.e. $s = u$, which, in view of $(*)$, means that $s \in G_{1_{\Pi_1}}(D)$ $(***)$. By $(**)$ and $(***)$ we conclude that $s \in G_\Pi(D)$ due to the first rule of Π.

6. If $\Pi = \bigcup_k[\Pi_1, \Pi_2]$, then:
($\Rightarrow$) Suppose that $s \in G_\Pi(D)$; from the rules of the program Π we see that there is a s_i (possibly $s_i = s$) such that $s_i \in G_{2_{\Pi_2}}(D)$. In addition, there exists a sequence $s_0 = s$, s_1, ..., s_i such that D contains the ground facts $R_k(s_r, s_{r+1})$ and $s_r \in G_{1_{\Pi_1}}(D)$ $(0 \leq r < i)$. By construction D^t also contains the ground facts $R_k(s_r, s_{r+1})$ $(0 \leq r < i)$. Further, the induction hypothesis implies that $s_i \in G_{2_{\Pi_2}}(D^t)$ and $s_r \in G_{1_{\Pi_1}}(D^t)$ $(0 \leq r < i)$. Consequently, by successive applications of the second rule, we conclude that $s \in G_\Pi(D^t)$.
($\Leftarrow$) Suppose that $s \in G_\Pi(D^t)$. Consider a minimal sequence $s_0 = s, s_1, \ldots, s_i$ (possibly $s_i = s$) such that D^t contains the ground facts $R_k(s_r, s_{r+1})$, $s_r \in G_{1_{\Pi_1}}(D^t)$ and $s_r \notin G_{2_{\Pi_2}}(D^t)$ $(0 \leq r < i)$ and $s_i \in G_{2_{\Pi_2}}(D^t)$. D contains also the facts $R_k(s_r, s_{r+1})$ $(0 \leq r < i)$. Let us assume that D does not contain $R_k(s_k, s_{k+1})$, for some k, $0 \leq k < i$. This means that $s_k = s_{k+1} = \ldots = s_i$ (recall (5)), which in turn implies that $s_i \notin G_{2_{\Pi_2}}(D^t)$, i.e, a contradiction. Thus, we have established that D also contains the facts $R_k(s_r, s_{r+1})$ $(0 \leq r < i)$. Now, the induction hypothesis implies that $s \in G_{2_{\Pi_2}}(D)$ and $s_r \in G_{1_{\Pi_1}}(D)$ $(0 \leq r < i)$. Consequently, by successive applications of the second rule, we conclude that $s \in G_\Pi(D)$.

7. If $\Pi = \widetilde{\bigcup}_k[\Pi_1, \Pi_2]$, then let $G_\Pi(D, r)$ and $G_\Pi(D^t, r)$ be the sets of ground facts of G that have been computed during the first r rounds of the evaluation of the last stratum of Π on D and D^t, respectively. We shall prove that $s \in G_\Pi(D, r) \Leftrightarrow s \in G_\Pi(D^t, r)$ using induction on the number of rounds r.
i. We prove the claim for $r = 1$.
($\Rightarrow$) Let $s \in G_\Pi(D, 1)$; s appears due to one of the first three rules of Π. If it is due to the first rule: $G(x) \longleftarrow G_1(x), G_2(x)$, then $s \in G_{1_{\Pi_1}}(D) \cap G_{2_{\Pi_2}}(D)$ and the induction hypothesis pertaining to Π_1 and Π_2 gives that $s \in G_{1_{\Pi_1}}(D^t) \cap G_{2_{\Pi_2}}(D^t)$, which immediately implies that $s \in G_\Pi(D^t, 1)$. If it is due to the second rule: $G(x) \longleftarrow G_2(x), \neg A_k(x)$, then $s \in G_{2_{\Pi_2}}(D)$ and D does not contain a ground fact of the form $R_k(s,u)$, for any constant u. The induction hypothesis with respect to Π_2 implies that $s \in G_{2_{\Pi_2}}(D^t)$. Moreover, by construction D^t contains the ground fact $R_k(s,s)$. Then, by the fifth rule of Π: $B_k(x,y) \longleftarrow G_2(x), R_k(x,y), G_2(y)$, $(s,s) \in B_{k_\Pi}(D^t)$ and, consequently, by the third rule $s \in G_\Pi(D^t, 1)$. If it is due to the third rule: $G(x) \longleftarrow B_k(x,x)$, then $(s,s) \in B_{k_\Pi}(D)$. This means that D contains a sequence of ground facts $R_k(s_0, s_1), R_k(s_1, s_2)$, ..., $R_k(s_l, s_{l+1})$ with $s_m \in G_{2_{\Pi_2}}(D)$, $0 \leq m \leq l+1$, and $s_0 = s_{l+1} = s$. Using the induction hypothesis pertaining to Π_2 we obtain $s_m \in G_{2_{\Pi_2}}(D^t)$, $0 \leq m \leq l+1$. Further, by construction D^t contains all the facts of D and,

therefore, $(s,s) \in B_{k_\Pi}(D^t)$. Finally, by the third rule we conclude that $s \in G_\Pi(D^t, 1)$.
($\Leftarrow$) Let $s \in G_\Pi(D^t, 1)$; s appears either due to the first or due to the third rule of Π. The totality of R_k^t precludes the use of the second rule. If it is due to the first rule, a trivial invocation of the induction hypothesis pertaining to Π_1 and Π_2 gives that $s \in G_\Pi(D, 1)$. If it is due to the third rule, then $(s,s) \in B_{k_\Pi}(D^t)$ and $s \in G_{2_{\Pi_2}}(D^t)$. We distinguish two cases, depending on whether D^t contains the ground fact $R_k(s,s)$ or not. Let us first consider the case where $R_k(s,s)$ is in D^t. If $R_k(s,s)$ is also in D, then, of course, $(s,s) \in B_{k_\Pi}(D)$ and, consequently, $s \in G_\Pi(D, 1)$. So, let us assume that D does not contain $R_k(s,s)$. This means that D contains no ground fact of the form $R_k(s,u)$, for any u, or, in other words, that $s \notin A_{k_\Pi}(D)$. Then, if we apply the second rule of Π, using the induction hypothesis to derive that $s \in G_{2_{\Pi_2}}(D)$, we conclude that $s \in G_\Pi(D, 1)$. Let us now consider the case where $R_k(s,s)$ is not in D^t. This means that D^t contains a sequence of ground facts $R_k(s_0, s_1), R_k(s_1, s_2), \ldots, R_k(s_l, s_{l+1})$ with $s_m \in G_{2_{\Pi_2}}(D^t)$, $0 \leq m \leq l+1$, and $s_0 = s_{l+1} = s$. Without loss of generality we may assume that this sequence does not contain any fact of the form $R_k(u,u)$[1]. D also contains the facts $R_k(s_0, s_1), R_k(s_1, s_2), \ldots, R_k(s_l, s_{l+1})$; for suppose to the contrary that one of these facts is not present in D. But then this "missing" fact must be of the form $R_k(u,u)$, which is absurd. Hence, using the induction hypothesis to derive that $s_m \in G_{2_{\Pi_2}}(D)$, $0 \leq m \leq l+1$, we deduce that $(s,s) \in B_{k_\Pi}(D)$, and, consequently, that $s \in G_\Pi(D, 1)$.
ii. We show now that the claim holds for $r+1$, assuming that it holds for r.
($\Rightarrow$) Suppose that s first appeared in $G_\Pi(D, r+1)$ during round $r+1$. This could have happened due to one of the first four rules of Π. In case one of the first three rules is used, by reasoning as above, we conclude that $s \in G_\Pi(D^t, r+1)$. So, let us suppose that the fourth rule: $G(x) \longleftarrow G_2(x), R_k(x,y), G(y)$ is used. This implies that $s \in G_{2_{\Pi_2}}(D)$ and that there exists a s_1 such that $R_k(s, s_1)$ and $s_1 \in G_\Pi(D, r)$. By construction D^t also contains $R_k(s, s_1)$. Moreover, the induction hypothesis with respect to the number of rounds gives that $s_1 \in G_\Pi(D^t, r)$ and the induction hypothesis with respect to Π_2 gives that $s \in G_{2_{\Pi_2}}(D^t)$. Thus, by the fourth rule we derive that $s \in G_\Pi(D^t, r+1)$.
($\Leftarrow$) Suppose now that s first appeared in $G_\Pi(D^t, r+1)$ during round $r+1$. This could have happened due to one of the first four rules of Π. In case one of the first three rules is used, then by reasoning as before, we obtain that $s \in G_\Pi(D, r+1)$. So, let us suppose that the fourth rule: $G(x) \longleftarrow G_2(x), R_k(x,y), G(y)$ is used. This implies that $s \in G_{2_{\Pi_2}}(D^t)$ and that there exists a s_1 such that $R_k(s, s_1)$ and $s_1 \in G_\Pi(D^t, n)$. The fact that s first appeared in $G_\Pi(D^t, r+1)$ during round $r+1$ means that $s \neq s_1$ because if $s = s_1$, then s would belong to $G_\Pi(D^t, r)$. This in turn implies that D contains $R_k(s, s_1)$. Invoking the induction hypothesis we get that $s_1 \in G_\Pi(D, r)$ and $s \in G_{2_{\Pi_2}}(D)$. Thus, by the fourth rule we derive that $s \in G_\Pi(D, r+1)$.

The bottom-up evaluation of Datalog programs guarantees that there exists $n_0 \in \mathbb{N}$ such that $G_\Pi(D, n_0) = G_\Pi(D, r)$ for every $r > n_0$, meaning that $G_\Pi(D) = G_\Pi(D, n_0)$. Similarly, $G_\Pi(D^t) = G_\Pi(D^t, n_0)$ and, hence, $G_\Pi(D) = G_\Pi(D^t)$. ■

[1]To see why, let us suppose that it contains the fact $R_k(u,u)$. This means that the sequence is $R_k(s_0, s_1), R_k(s_1, s_2), \ldots, R_k(s_i, u), R_k(u,u), R_k(u, s_{i+3}), R_k(s_{i+3}, s_{i+4}), \ldots, R_k(s_l, s_{l+1})$. But then consider the sequence $R_k(s_0, s_1), R_k(s_1, s_2), \ldots, R_k(s_i, u), R_k(u, s_{i+3}), R_k(s_{i+3}, s_{i+4}), \ldots, R_k(s_l, s_{l+1})$ that also gives rise to $(s,s) \in B_{k_\Pi}(D^t)$ without containing $R_k(u,u)$.