Dynamic Deductive Data Bases with Steady Behaviour

Michael I. Dekhtyar
Department of Computer Science
Tver State University
Tver, Russia, 170000
mat@mat.tvegu.tver.su

Alexander Ja. Dikovsky
4 Miusskaya Sq.
Keldysh Inst. for Appl. Math.
Moscow, Russia, 125047
dikovsky@applmat.msk.su

Abstract

We consider deductive databases with updates and integrity constraints and introduce a new notion - that of bounded disturbance of an active medium. In these terms we define several types of notions of steady behavior of such deductive databases, which may open up new classes of applications of logically controlled databases. In this work we explore computational complexity of the concepts introduced.

1 Introduction

This work presents a novel approach to deductive databases with updates. Traditionally the frame of deductive data bases includes: a version of logic language **L** in which relations on data are specified; a formal system for **L**, normally, a resolution based one, which implements an operational semantics of formulas (queries); a set of axioms imposing implicit conditions on interpretations (DB states), e.g. CWA; and a set of integrity constraints (IC), i.e. invariant properties of evolving DB states [11]. In the recent decade extensive research was done in the field of deductive databases with updates (*dynamic DDBs*). Most attention was focussed on update languages and on the analysis of their expressibility and complexity (cf. [1, 15, 14, 4]) as well as of declarative and operational semantics of updates (cf. [16, 9, 5]), the latter being closely connected with analysis of revision and updates in propositional knowledge bases (cf. [13, 10]). The central problem in this field is determination of the effect of updates and validation of their consistency with IC. Updates violating IC are usually treated as abnormal. This is one of the points in which our approach to updates differs from the conventional one. We distinguish two sources of modifications of DB states: the *updates* proper caused by DDB control, and external *disturbances* of the medium in

violation, while the other one may restore the IC. There are various classes of important applications calling for such an approach to updates. A typical example is planning production processes with restrictions on the resources consumed. Specific transactions predefined by production technology might cause consumption and, possibly, exhaustion of some of the resources needed to continue the process. So, to maintain this process, we need periodical external replenishment of the resources, and successful planning should take this into account (see the example in the Appendix). Traditional formal means dealing with dynamic systems are analytical. Though advantageous in many respects, they are very sensitive to the growth of the number of parameters. For DDBs the number of DB attributes is rather a secondary factor. Threfore, the logical data bases approach to simulating the behavior of dynamic systems seems to be very promising when these systems are characterized by multiple parameters.

A deductive DDB $\mathcal{B}$ simulating the behavior of a dynamic system $\mathcal{S}$ represents its current states by DB states. We represent the interaction of $\mathcal{S}$ and its medium by trajectories, i.e. sequences of DB states of the form

$$\omega: \ \mathcal{E}_0 \xrightarrow{d_1} \mathcal{E}_1^* \overset{G_1}{\vdash} \mathcal{E}_1 \xrightarrow{d_2} \mathcal{E}_2^* \overset{G_2}{\vdash} \ ....$$

Each action $\mathcal{E}_i^* \overset{G_i}{\vdash} \mathcal{E}_i$ stands for a successful refutation in $\mathcal{B}$ of the goal G_i in the DB state $\mathcal{E}_i^*$, resulting in the new DB state $\mathcal{E}_i$. This is a predefined update of $\mathcal{B}$. Each step $\mathcal{E}_i \xrightarrow{d_i} \mathcal{E}_i^*$ stands for an external disturbance d_i. Suppose updates apply only to the states satisfying the IC of $\mathcal{B}$. Then the existence of an infinite trajectory ω means that the disturbances restore IC after all updates, and so $\mathcal{B}$ can behave steadily in the state $\mathcal{E}_0$. Moreover, if we find that all disturbances in the ω are bounded by some k with respect to a certain measure on disturbances, it will mean that the steadiness is attained at the cost of no more than k-bounded external disturbances. We call this sort of steady behavior a *k-bounded stability*. We also consider another kind of steady behavior manifested in the existence of a trajectory with empty disturbances which starts in a DB state $\mathcal{E}_0$ and leads to a DB state $\mathcal{E}_N$ obeying the IC. This property of a DDB indicates that it can *sometimes* restore the IC without external disturbances. We call such states $\mathcal{E}_0$ *promising*. In this paper we explore computational complexity of these two kinds of steady behavior with respect to several types of IC and in various classes of deductive DDBs. Bounded stability and promise properties are certainly not recursive equivalent in the class of unlimited deductive DDBs. Nonetheless, their complexity is the same in the classes of practical interest that we consider. E.g., when IC are monotonic and ground, while DDBs are ground, nonrecursive, positive (don't use negation) and never delete facts, they are of linear time complexity. If we admit nonmonotonic IC or negation, they become NP-complete. If, besides this, we allow deletion, they will be $PSPACE$-complete. If nonrecursive Datalog DDBs are permitted as well, then their complexity goes up to $SPACE(2^{poly})$-completeness, and

more, we admit recursion stratified with respect to elementary updates, then both are $PSPACE$-complete in ground case and $SPACE(2^{poly})$-complete in Datalog case.

This paper is organized as follows. Section 2 contains some necessary preliminaries. Section 3 introduces the basic notions of δ-bounded disturbance of medium, δ-stable trajectory and promising state. Section 4 presents results characterizing complexity of the promise problem in several classes of DDBs. Section 5 contains results on complexity of two modifications of the stability problem in the same classes of DDBs. Due to the lack of space we provide only brief sketches of some proofs of upper bounds.

2 Notation and Preliminaries

We consider logic programs with updates in a first-order language $\mathbf{L}$ as nondeterministic transformations of database (DB) states. Along with the set of predicate symbols $\mathbf{P}$ of $\mathbf{L}$ we admit in general the set $\mathbf{F}$ of its functors. To obtain a natural semantics of updates we, similarly to [14], split $\mathbf{P}$ into two disjoint parts $\mathbf{P}^e$ (*extensional* predicates) and $\mathbf{P}^i$ (*intensional* predicates). Accordingly, the Herbrand base $\mathbf{B}$ of $\mathbf{L}$ splits into $\mathbf{B}^e = \{p(t_1, ..., t_k) \mid t_i \in \mathbf{H}, p \in \mathbf{P}^e\}$ and $\mathbf{B}^i = \{q(u_1, ..., u_l) \mid u_i \in \mathbf{H}, q \in \mathbf{P}^i\}$, where $\mathbf{H}$ is the Herbrand universe of $\mathbf{L}$. DB states are subsets of $\mathbf{B}^e$. The heads of clauses of logic programs are always intensional atoms. The bodies of clauses can include two kinds of elementary database updates: $insert(A), delete(A)$, where A is an extensional atom. We also allow negative extensional atoms in the clause bodies. To save the size of the programs, the bodies may include the disjunction ";" as well as the update operator *change(A,B)* in place of the combination *delete(A), insert(B).*

We rely on the traditional SLD-refutation operational semantics of logic programs with the following specific features. The Prolog like leftmost selection computation rule is used with safe calls of elementary updates (i.e. all variables in the arguments of updates of the form $insert(A)$ or $delete(A)$ should be bound by ground terms by the moment of their call). Negation is treated as finite failure, which means in our very restricted situation that a negated atom in the current state cannot be unified to any fact in the current DB state. Successful refutations have a side effect of transforming initial DB states into new DB states. Any logic program with updates can thus be naturally associated with the following relation $\mathcal{E} \vdash \mathcal{E}'$ on DB states.

Definition 1 *Let $\mathcal{I}$ be a logic program, $\mathcal{E}$, $\mathcal{E}'$ be two DB states and $:- G$ be a goal. A refutation of $\mathcal{I} \cup \mathcal{E} \cup \{ :- G\}$ is a finite sequence $S = s_1, s_2, ..., s_n$ of triplets (computation states) of the form $s_i = (\mathcal{E}_i, G_i, \sigma_i)$, in which $\mathcal{E}_i$ is a DB state, G_i is a goal and σ_i is a substitution, $\mathcal{E}_1 = \mathcal{E}$, $G_1 = G$, σ_1 is the identity substitution ε, G_n is the empty goal*

$G_i = L_1, L_2, \ldots, L_k$.

(1) When L_1 *is an atom, then for some variant of a clause (possibly a unit one)* H :- $B_1, \ldots, B_r$ $(r \geq 0)$ *and for MGU* θ *of* $L_1 \circ \sigma_i$ *and* H *the equalities* $G_{i+1} = B_1, \ldots, B_r, L_2, \ldots, L_k$, $\sigma_{i+1} = \sigma_i \circ \theta$ *and* $\mathcal{E}_{i+1} = \mathcal{E}_i$ *hold as normally.*

(2) When $L_1 = \neg q(\bar{t})$ *for some* $q \in \mathcal{P}^e$ *and* $q(\bar{t}) \circ \sigma_i$ *cannot be unified to any fact in* $\mathcal{E}_i$, *then the equalities* $G_{i+1} = L_2, \ldots, L_k, \mathcal{E}_{i+1} = \mathcal{E}_i$ *and* $\sigma_{i+1} = \sigma_i$ *hold.*

(3) When $L_1 = insert(A)$ *and* $A \circ \sigma_i \in \mathbf{B}^e$ *then the equalities* $G_{i+1} = L_2, \ldots, L_k$, $\mathcal{E}_{i+1} = \mathcal{E}_i \cup \{ A \circ \sigma_i \}$ *and* $\sigma_{i+1} = \sigma_i$ *hold.*

(4) When $L_1 = delete(A)$ *and* $A \circ \sigma_i \in \mathbf{B}^e$ *then the equalities* $G_{i+1} = L_2, \ldots, L_k$, $\mathcal{E}_{i+1} = \mathcal{E}_i \setminus \{A \circ \sigma_i\}$ *and* $\sigma_{i+1} = \sigma_i$ *hold.*

When such a refutation exists, we write the usual $\mathcal{I} \cup \mathcal{E} \cup \{ :- G\} \Longrightarrow \Box$. *If in addition* $\mathcal{E}_n = \mathcal{E}'$, *then we say that the program* $\mathcal{I}$ *with the goal* $\{ :- G\}$ *transforms* $\mathcal{E}$ *into* $\mathcal{E}'$, *and denote this by* $\mathcal{E} \overset{G}{\vdash_{\mathcal{I}}} \mathcal{E}'$.

As is readily seen from this definition, we consider each goal for a logic program with updates as a pre-defined nondeterministic transformation of DB states. In contrast to [5], we do not extend the language of logic programs to reflect explicitly intermediate DB states of refutations. However, our semantics allows us to impose constraints on these states in a natural manner. E.g., a successful application of the clause

$$h \text{ :- } insert(son(jack, tom)), \neg ancestor(jack, tom), etc, ...$$

is possible only if the addition of the fact $son(jack, tom)$ to a current DB state does not conflict the acyclicity of the "ancestor" relation.

In real application databases data can usually be naturally stratified into classes where specific values of certain attributes are purely informational, i.e. their change does not affect operational properties of the data base. The data differing only in such "insignificant" features can be regarded as equivalent. The partitioning of data into classes of data equivalence sometimes permits reduction of a large and potentially infinite application domain to a finite and visible one. Some variations of this property can be found in theory. E.g., the so called C-genericity considered in several works [12, 2, 1, 6] defines equivalence of all constants to within some finite subset. We introduce the following more general notion.

Definition 2 *Let* $\equiv$ *be an equivalence on* $\mathbf{H}$. *It can be extended naturally onto* $\mathbf{B}^e$: $g(t_1, ..., t_n) \equiv g(t'_1, ..., t'_n)$ *iff* $t_i \equiv t'_i$ *for all* $1 \leq i \leq n$. *For any* $D_1, D_2 \subseteq \mathbf{B}^e$: *we set* $D_1 \Lessdot D_2$, *if there are such* $b > 0$ *and such a function* $\eta : D_1 \to D_2$ *that for every* $G \in D_1$ $G \equiv \eta(G)$, *and for every* $G' \in D_2$ $\mid \eta^{-1}(G') \mid < b$. $D_1 \Lessgtr D_2$ *if* $D_1 \Lessdot D_2$ *and* $D_2 \Lessdot D_1$.

A logic program $\mathcal{I} \cup \{ :- G\}$ *is* compatible with an equivalence $\equiv$ *on* $\mathbf{B}^e$ *if for any three states* $\mathcal{E}_1, \mathcal{E}'_1, \mathcal{E}_2 \subseteq \mathbf{B}^e$ *such that* $\mathcal{E}_1 \Lessgtr \mathcal{E}_2$ *and* $\mathcal{E}_1 \overset{G}{\vdash_{\mathcal{I}}} \mathcal{E}'_1$ *there exists such a state* $\mathcal{E}'_2$ *that* $\mathcal{E}'_1 \Lessgtr \mathcal{E}'_2$ *and* $\mathcal{E}_2 \overset{G}{\vdash_{\mathcal{I}}} \mathcal{E}'_2$.

to Turing machines. That is why the majority of natural algorithmic problems related to transformations of DB states are unsolvable. Nevertheless there are natural constraints on logic programs which hold generally in applications and provide solutions for some of these problems. One of the most realistic constraints of this kind is the property of stratifiability analogous in a sense to [3].

Definition 3 *Let $\mathcal{P}$ be a logic program. We say that a predicate p* refers to *a predicate q if there is a clause defining p in $\mathcal{P}$ with q in its body. We consider the relation "*depend on*" which is the reflexive and transitive closure of the relation "refer to". Maximal strongly connected components of the graph of the relation "depend on" are called* cliques. *$\mathcal{P}$ is called* dynamically stratified (d-stratified) *if in any of its clauses*

$$p(\bar{t}) \ :- \ p_1(\bar{t_1}), \ldots, p_i(\bar{t_i}), q(\bar{u}), p_{i+1}(\bar{t}_{i+1}), \ldots, p_r(\bar{t_r})$$

in which q is in the clique of p, all predicates $p_1, \ldots, p_i, p_{i+1}, \ldots, p_r$ are stationary, *i.e. do not depend on elementary updates.*

The main point of this definition is that DB updates are available only at the steps where a clique is changed. The next definition introduces a classification of logic programs by the form of their clauses.

Definition 4 *A logic program $\mathcal{P}$ is* stationary *if all its predicates are stationary. $\mathcal{P}$ is* positive *if it does not use negation. It is called* ground *if all its clauses are ground. It is* flat *if all terms in its clauses are variables or constants. We call $\mathcal{P}$* expanding *if the update delete/1 is not used in its clauses. And we call $\mathcal{P}$* productional *if it defines the unique intensional predicate q/0 and all its clauses are* productions, *i.e. have the form q :- Con_1, $\ldots, Con_k$, $Act_1, \ldots, Act_m$ where each Con_i is an extensional literal and each Act_j is an elementary update.*

The term "production" is borrowed from AI. Our clause q :- $Con_1, \ldots, Con_k$, $Act_1, \ldots, Act_m$ coresponds there to the conditional transformation of the form $CONDITION \Rightarrow ACTION$ where $CONDITION$ is the conjunction of the literals Con_i, and $ACTION$ is the sequence of updates $Act_1, \ldots, Act_m$. Similar rules have been used, say in the transaction language DL in [1], though in programs with somewhat different semantics.

3 Behavior of Dynamic Deductive Data Bases

The deductive data bases (DDBs) under consideration are in the line of the definition in [11]. They include an intensional logic program with updates, a pre-defined set of goals implementing DB state transactions as well as queries, integrity constraints embodied by a stationary logic program and a data equivalence. To describe the behavior of dynamic DDBs in media

properties of trajectories reflecting steady behavior.

Definition 5 *A* dynamic deductive data base(DDB) *in language* $\mathbf{L}$ *is a system*

$$\mathcal{B} =< \mathcal{I} \cup \{ \ :- \ G_1, \ldots, \ :- \ G_n \ \}, \ \mathcal{A} \cup \{ \ :- \ R \ \}, \ \equiv_i >$$

where:

- $\mathcal{I}$ *is a logic program with updates in* $\mathbf{L}$,

- *all goals* $G_i, \ i = 1, \ldots, n,$ *are either ground or stationary,*

- $\mathcal{A} \cup \{ \ :- \ \mathcal{R}\}$ *is a stationary logic program in* $\mathbf{L}$ *defining* integrity constraints (IC),

- $\equiv_i$ *is an equivalence on* $\mathbf{B}^e$ *(we call it a* data equivalence*), and logic programs* $\mathcal{I} \cup \{ \ :- \ G_i\}, \ i = 1, \ldots, n,$ *and* $\mathcal{A} \cup \{ \ :- \ \mathcal{R}\}$ *are compatible with* $\equiv_i$.

Ground goals are called updates. *Stationary goals are called* queries.

Now we turn to the main definition of this paper. Its core concept is the notion of trajectory with disturbance which makes behavior of a dynamic DDB reactive towards the effect of its medium.

Definition 6 *For any two disjoint sets* $\mathcal{D}^+, \mathcal{D}^- \subseteq \mathbf{B}^e$, *we define the* $(\mathcal{D}^+, \mathcal{D}^-)$-disturbance *as the relation on DB states* $\mathcal{E}_1 \stackrel{\mathcal{D}^+, \mathcal{D}^-}{\longrightarrow} \mathcal{E}_2$ *such that* $\mathcal{E}_1, \mathcal{E}_2 \subseteq \mathbf{B}^e$ *and* $\mathcal{E}_2 = (\mathcal{E}_1 \cup \mathcal{D}^+) \setminus \mathcal{D}^-$.

Let $\mathcal{B}$ *be a DDB. We define a* trajectory *of* $\mathcal{B}$ *as any finite or infinite sequence of the form*

$$\omega : \ \mathcal{E}_0 \stackrel{\mathcal{D}_1^+, \mathcal{D}_1^-}{\longrightarrow} \mathcal{E}_1^* \stackrel{G_{i_1}}{\vdash_{\mathcal{I}}} \mathcal{E}_1 \stackrel{\mathcal{D}_2^+, \mathcal{D}_2^-}{\longrightarrow} \mathcal{E}_2^* \stackrel{G_{i_2}}{\vdash_{\mathcal{I}}} \mathcal{E}_2 \ \ldots \ .$$

The sequence of pairs $d = (\mathcal{D}_1^+, \mathcal{D}_1^-), (\mathcal{D}_2^+, \mathcal{D}_2^-), \ldots$ *is called the* disturbance *of the trajectory* ω. ω *is* d-supported *if all states* $\mathcal{E}_i^*, \ i = 1, 2, \ldots$ *satisfy the IC* $\mathcal{A}$, *i.e.* $\mathcal{A} \cup \mathcal{E}_i^* \cup \{ \ :- \ \mathcal{R}\} \Longrightarrow \ \Box$.

A specific feature of this definition is that it reflects the sequential model of interaction between a dynamic DDBs and its media. They affect DB states in turn. d-supportedness of a trajectory means that all IC violations caused by DDB updates are compensated in this trajectory by subsequent disturbances. Notice that such a compensation is always possible at the cost of an unlimited increase in the size of the disturbance. Therefore we need reasonable restrictions on this size so as to obtain adequate notions of steady behavior.

Definition 7 *Let* $\mathcal{B}$ *be a DDB and* $\delta =< \mathcal{D}^+, \mathcal{D}^- >$ *with finite* $\mathcal{D}^+, \mathcal{D}^- \subseteq \mathbf{B}^e$. *A trajectory* ω *with the disturbance* $d = (\mathcal{D}_1^+, \mathcal{D}_1^-), (\mathcal{D}_2^+, \mathcal{D}_2^-), \ldots$ *is* δ-stable *if it is d-supported, infinite and for all* $k \geq 1$

$$\mathcal{D}_k^+ \Leqslant_i \mathcal{D}^+ \text{ and } \mathcal{D}_k^- \Leqslant_i \mathcal{D}^-.$$

this DB state.

We call a DB state $\mathcal{E}$ promising *for* $\mathcal{B}$ *if there is a finite trajectory of* $\mathcal{B}$ *with empty disturbance, i.e. with* $\mathcal{D}_k^+ = \mathcal{D}_k^- = \emptyset$ *for all* k, *starting in* $\mathcal{E}$ *and resulting in some DB state satisfying the IC* $\mathcal{A}$.

The above properties of steady behavior are rather weak. $\delta-$stability is an existential property. It does not reflect the behavior of the DDB along all possible trajectories. As to promise, it provides compensation of IC without external disturbances, though along some trajectory of indefinite length.

Example 1. Consider the following toy DDB $\mathcal{B}_1$ describing health condition in terms of three propositional extensional predicates $healthy, ill$ and $medicine$ (the last one representing the availability of the necessary remedy). The intensional logic program consists of two updates:

$upd_1 \ :- \ ill, medicine, delete(ill), delete(medicine), insert(healthy).$

$upd_2 \ :- \ healthy, change(healthy, ill).$

The IC describes satisfactory states:

$r \ :- \ healthy \ ; \ ill, medicine.$

This DDB is $(\{medicine\}, \emptyset)$-stable in any DB state satisfying the IC. Indeed, if the DB state contains $healthy$, then the second update upd_2 is applied. It might violate the IC in the case where $medicine$ is not present. However, the IC can be restored by the only possible nonempty disturbance. If the DB state contains ill and $medicine$, then the first update upd_1 is applied and a state containing $healthy$ arises. Observe that none of the DB states violating the IC is promising in $\mathcal{B}_1$. Meanwhile, if we replace the IC r by

$r_1 \ :- \ healthy \ ; \ \neg ill.$

then the state $\{ill, medicine\}$ does not satisfy the new IC r_1 though it is promising.

Example 2. Let us develop the preceding DDB by including two different diseases (ds_1, ds_2) and corresponding medicines (mdc_1, mdc_2) and let immunity (imm_1, imm_2) arise as a result of curing these diseases. The new DDB $\mathcal{B}_2$ has the following intensional logic program:

$upd_{1i} \ :- \ ds_i, mdc_i, delete(ds_i), delete(mdc_i), \ insert(imm_i). \ (i = 1, 2),$

$upd_2 \ :- \ \neg imm_1, \neg imm_2, \neg ds_1, \neg ds_2, insert(ds_1), insert(ds_2),$

$upd_{3i} \ :- \ imm_i, delete(imm_i). \ (i = 1, 2)$

and the IC:

$r \ :- \ (\neg ds_1; mdc_1), (\neg ds_2; mdc_2).$

The DDB $\mathcal{B}_2$ is $\delta-$stable for $\delta = (\{mdc_1, mdc_2\}, \emptyset)$ in all states satisfying the IC. This is rather obvious because in each DB state satisfying the IC at least one update is applicable, and moreover, the maximal disturbance transforms any DB state into a state obeying the IC. However, for $\delta_1 = (\{mdc_1\}, \emptyset)$ $\mathcal{B}_2$ is not δ_1-stable in any state. The reason is that any infinite d-supported trajectory includes an application of the update upd_2 which leads to a DB state including both diseases, hence for restoring the

the Appendix.

4 Complexity of Promise Problem

Promise guarantees the accessibility of a state obeying IC from a given state without disturbances. This corresponds to purely logical moves along a trajectory. Let $\mathbf{P}$ be a class of DDBs. The promise problem for this class is the problem of membership of pairs $(\mathcal{B}, \mathcal{E})$ in the set

$$PROMISE(\mathbf{P}) = \{(\mathcal{B}, \mathcal{E}) \mid \mathcal{E} \textit{ is a promising state of a DDB } \mathcal{B} \in \mathbf{P}\}.$$

The promise problem apparently includes the problem of satisfiability of IC, i.e. the halting problem for logic programs implementing IC. So in the class of all dynamic DDBs this problem is undecidable. That is why we consider below only those DDBs whose IC $\mathcal{A} \cup \{ :- \mathcal{R}\}$ satisfy one of the following conditions:

(IC0) $\mathcal{A}$ is positive and ground (IC0-constraints);

(IC1) $\mathcal{A}$ is ground (IC1-constraints);

(IC2) all intensional atoms in the bodies of clauses are ground (IC2-constraints).

We need to know the complexity of the IC checking under these restrictions, to account for its part in the complexity of promise and stability problems.

Theorem 1

(1) The problem of IC-satisfiability in a finite DB state $\mathcal{E} \subseteq \mathbf{B}^e$ w.r.t. IC1-constraints (IC0-constraints) $\mathcal{A}$ can be solved in linear time w.r.t. $| \mathcal{A} | + | \mathcal{E} |$.

(2) The problem of IC-satisfiability in a finite DB state $\mathcal{E} \subseteq \mathbf{B}^e$ w.r.t. IC2-constraints $\mathcal{A}$ is NP-complete.

A square time algorithm in the case of IC1-constraints is straightforward. A linear time algorithm in this case can be obtained by the method used in [7].

After we have reasonably narrowed the class of integrity constraints we should impose several workable conditions on intensional parts of DDBs and analyze their impact on computational complexity. By $PROD$ we denote the set of all DDBs with productional intensional programs $\mathcal{I}$. Within this class we introduce several subclasses:

- $PROF$ is the subclass of all DDBs in $PROD$ with flat intensional parts $\mathcal{I}$;
- $PROG$ is the subclass of all DDBs in $PROD$ with ground intensional parts;
- $PROG^-$ is the subclass of all DDBs in $PROG$ with expanding intensional parts;

tional parts (hence intensional parts of programs in $PROG^+$ do not use negation and the update $delete/1$).

The DDB $\mathcal{B}_1$ in example 1 belongs to $PROG^-$, whereas $\mathcal{B}_2$ in example 2 belongs to $PROG$. In all these classes one can find various important applications. E.g. spreadsheets can be directly represented as DDBs in $PROG$ with very simple IC1-constraints. The so called productional expert systems fall into the $PROF$ class and some of them even into $PROG^-$. The next theorem shows that the complexity of the promise problem in these classes of DDBs immediately depends on the kinds of nonmonotonic means used in their intensional parts.

Theorem 2

(1) The problem $PROMISE(PROG^+)$ w.r.t. IC0-constraints can be solved in linear time, and w.r.t. IC1-constraints it is NP-complete.

(2) The problem $PROMISE(PROG^-)$ w.r.t. IC2-constraints belongs to NP, and w.r.t. IC0-constraints it is NP-hard.

(3) The problem $PROMISE(PROG)$ w.r.t. IC2-constraints belongs to PSPACE, and w.r.t. IC0-constraints it is PSPACE-hard.

(4) The problem $PROMISE(PROF)$ w.r.t. IC2-constraints belongs to $SPACE(2^{poly})$. and w.r.t. IC0-constraints it is $SPACE(2^{poly})$-hard.

(5) The problem $PROMISE(PROD)$ is undecidable.

It is interesting to note that the promise problem is solvable in a class of d-stratified deductive DDBs substantially greater than $PROF$. We will use **GDS** and **FDS** to denote the classes of all DDBs having IC2-constraints and d-stratified and, resp., ground or flat intensional parts.

Theorem 3

(1) The problem $PROMISE(GDS)$ is $PSPACE$-complete.

(2) The problem $PROMISE(FDS)$ is $SPACE(2^{poly})$-complete.

Proof sketch. Upper bounds need some auxiliary concepts and lemmas. Computations of logic programs are naturally represented by computation trees. We adapt here the definition of a computation tree used in [8] for computations of Prolog programs.

Definition 8 *Let $\mathcal{I}$ be some logic program, $R = A_1, ..., A_n$, and c be its computation $\mathcal{E}_1 \stackrel{R}{\vdash_{\mathcal{I}}} \mathcal{E}_2$. Then the tree of this computation $t = t(c)$ is defined as follows. The nodes of t are pairs of the form (G, U) where G is one of the subgoals resolved in c and U is the substitution chosen for its resolution. The root of t is the pair $v_0 = (R, \varepsilon)$ and it has sons $(A_1, U_1), ..., (A_n, U_n)$. Let a node $v = (G_1, U)$ of t correspond to a subgoal G_1 resolved in c in a state $(\mathcal{E}, (G_1, ..., G_k), \sigma)$. In the case where G_1 is an elementary update or a negated atom, v is a leaf and $U = \varepsilon$. If G_1 is an extensional or intensional atom resolved by a unit clause B in $\mathcal{I} \cup \mathcal{E}$,*

is an intensional atom resolved by a clause H :- $B_1, \ldots, B_r (r \geq 1)$, *and* θ *is the MGU of* $G_1 \circ \sigma$ *and* H, *then* $U = \theta$ *and* v *has in* t r *sons:* $(B_1, U_1), ..., (B_r, U_r)$.

We associate with each node v *of* t *its input and output DB states* $\mathcal{E}^{in}(v), \mathcal{E}^{out}(v)$ *and its global contexts* $\sigma^{in}(v), \sigma^{out}(v)$. *For the root* v_0 *we set* $\mathcal{E}^{in}(v_0) = \mathcal{E}_1$ *and* $\mathcal{E}^{out}(v_0) = \mathcal{E}_2$ *and* $\sigma^{in}(v) = \varepsilon$. *For leaves* v *with negative subgoals,* $\mathcal{E}^{in}(v) = \mathcal{E}^{out}(v)$ *and* $\sigma^{in}(v) = \sigma^{out}(v)$. *For elementary update leaves* v, $\sigma^{in}(v) = \sigma^{out}(v)$ *and* $\mathcal{E}^{out}(v)$ *is obtained from* $\mathcal{E}^{in}(v)$ *by the corresponding elementary update (inserting or deleting a fact). For all other leaves* $v = (G, U)$, $\mathcal{E}^{in}(v) = \mathcal{E}^{out}(v)$ *and* $\sigma^{out}(v) = \sigma^{in}(v) \circ U$. *For an inner node* $v = (G, U)$ *with* r *sons* $v_1 = (B_1, U_1), ..., v_r = (B_r, U_r)$, *we set* $\mathcal{E}^{in}(v_1) = \mathcal{E}^{in}(v)$, $\mathcal{E}^{out}(v) = \mathcal{E}^{out}(v_r)$, $\mathcal{E}^{out}(v_i) = \mathcal{E}^{in}(v_{i+1})$ *for each* $1 \leq i < r$, $\sigma^{in}(v_1) = \sigma^{in}(v) \circ U$, $\sigma^{out}(v) = \sigma^{out}(v_r)$ *and* $\sigma^{out}(v_i) = \sigma^{in}(v_{i+1})$ *for each* $1 \leq i < r$. *For a node* $v = (G, U)$, *we call the literals* $inst^{in}(v) = G \circ \sigma^{in}(v)$ *and* $inst^{out}(v) = G \circ \sigma^{out}(v)$, *resp., an* in-instance *and an* out-instance of G in v.

One can easily verify that these definitions of input and output DB states and global contexts are correct. From this definition it follows that $\sigma^{out}(v_0)$ restricted to the variables of the goal R is the computed answer substitution of the computation c.

To illustrate these notions, let $\mathcal{I}$ be the logic program with clauses

s :- $e_1(X), a(X)$

$a(X)$:- $\neg e_2(X), insert(e2(X))$

$a(X)$:- $e_2(X), delete(e2(X))$

and with the goal :- s. Let $\mathcal{E}_0 = \{e_1(c_1), e_2(c_2)\}$ and $\mathcal{E}_1 = \{e_1(c_1), e_2(c_2), e_2(c_1)\}$ be two DB states. The computation tree $\mathcal{E}_0 \overset{s}{\vdash}_{\mathcal{I}} \mathcal{E}_1$ is shown in Fig. 1.

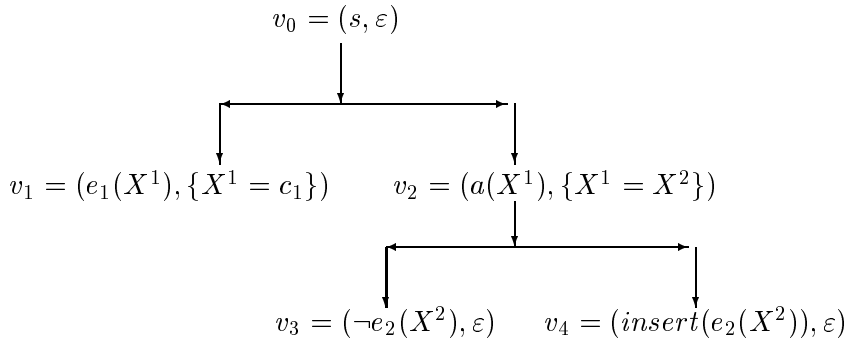

*Fig.*1

Definition 9 *Let* t *be some computation tree, and let* $v_1 = (A_1, U_1)$ *and* $v_2 = (A_2, U_2)$ *be nodes of* t *such that* v_1 *is an ancestor of* v_2. *We call this pair of nodes* contractable *if* $\mathcal{E}^{in}(v_1) = \mathcal{E}^{in}(v_2)$, $\mathcal{E}^{out}(v_1) = \mathcal{E}^{out}(v_2)$ *and there exists such a 1-1 renaming substitution* θ *that* $A_1 = A_2 \circ \theta$ *and for*

$\sigma^{out}(v_2)(X \circ \theta)$. *The tree* t *is* contracted *if it has no two contractable nodes.*

Two computation trees t_1, t_2 *with the roots* v_{01}, v_{02} *respectively, are* equivalent *if* $inst^{in}(v_{01}) = inst^{in}(v_{02})$, $inst^{out}(v_{01}) = inst^{out}(v_{02})$, $\mathcal{E}^{in}(v_{01}) = \mathcal{E}^{in}(v_{02})$ *and* $\mathcal{E}^{out}(v_{01}) = \mathcal{E}^{out}(v_{02})$.

In this computation tree we have, for example, $inst^{in}(v_1) = e_1(X^1)$, $inst^{out}(v_1) = e_1(c^1)$, $\mathcal{E}^{in}(v_4) = \mathcal{E}_0$, $\mathcal{E}^{out}(v_4) = \mathcal{E}_1$, $inst^{in}(v_3) = \neg e_2(c_1)$ and $\sigma^{out}(v_4) = \{X^1 = c_1, X^2 = X^1\}$.

Lemma 1 *(Contraction lemma). For any computation tree* t_1 *there exists an equivalent contracted computation tree* t_2.

Lemma 2 *Let* t *be a computation tree of* $\mathcal{E}_1 \overset{G}{\vdash_{\mathcal{I}}} \mathcal{E}_2$, $\mathbf{b}_{\mathcal{I}}$ *be the maximal length of clause bodies of* $\mathcal{I}$, $\mathbf{l}(t) = max\{\mid inst^{in}(v) \mid, \mid inst^{out}(v) \mid \mid v \in t\}$, *and* $\mathbf{db}(t) = max\{\mid \mathcal{E}^{in}(v) \mid, \mid \mathcal{E}^{out}(v) \mid \mid v \in t\}$. *Then the computation described by* t *can be simulated by a Turing machine in space* $O(max\{\mathbf{db}(t),\ depth(t) \times \mathbf{b}_{\mathcal{I}} \times \mathbf{l}(t)\})$.

Returning to the proof of upper bounds we observe that in any path of a contracted computation tree all nodes are pairwise not equivalent. Therefore by the contraction lemma, the depth of a computation tree for **GDS** can be bounded by $|\mathcal{I}|^2$. For **FDS** this depth can be bounded by $|\mathcal{I}|pc^a$, where p is the number of intensional predicates of $\mathcal{I}$, c is the number of its constants and a is the maximal arity of its intensional predicates. Therefore by lemma 2 we get the required upper bounds. □

5 Complexity of Stability Problems

In this section we explore the complexity of the problems of stability in a state and stability in multiple states.

Definition 10 *Let* $\mathcal{B}$ *be a DDB and* $\delta =< \mathcal{C}^+, \mathcal{C}^- >$ *with finite* $\mathcal{C}^+, \mathcal{C}^- \subseteq \mathbf{B}^e$. *A DB state* $\mathcal{E}$ *of* $\mathcal{B}$ *is called a* δ-state *if* $\mathcal{C}^+ \leqslant\!\!=_i \mathcal{E}$ *and for no* $t \in \mathcal{C}^-$ $t \leqslant\!\!=_i \mathcal{E}$ *is true.*

For each class $\mathbf{P}$ of DDBs we will consider only its subclass $\mathbf{P}^{peq}$ with "tractable" data equivalence relations, i.e. such that for some polynomial *pol* the problems $\mathcal{D}_1 \leqslant\!\!=_i \mathcal{D}_2$ and "*for no* $A \in \mathcal{D}_1$, $B \in \mathcal{D}_2$ $A \equiv_i B$" can be solved in space $pol(\mid \mathcal{D}_1 \mid + \mid \mathcal{D}_2 \mid)$ for any finite $\mathcal{D}_1$, $\mathcal{D}_2$. In doing so, in cases when we establish linear (polynomial) time complexity, we suppose these problems to be of linear (polynomial) time complexity either. In all proofs of lower bounds we use only the identity data equivalence.
We estimate the complexity of the following problems:

$STABLE^{\forall}(\mathbf{P}) = \{(\mathcal{B}, \delta, C^+, C^-) \mid \mathcal{B} \in \mathbf{P}^{peq}$ *is* $\delta -$ *stable in any* $(C^+, C^-) -$ *state*$\}$.

It is rather a surprising fact that in simple classes of deductive DDBs under consideration the complexity of the promise problem and the stability problem is the same. This is not the case in the class of unrestricted DDBs.

Theorem 4

(1) The problem $STABLE(PROG^+)$ *w.r.t. IC0-constraints can be solved in linear time, and w.r.t. IC1-constraints it is* NP*-complete.*

(2) The problem $STABLE(PROG^-)$ *w.r.t. IC2-constraints belongs to* NP*, and w.r.t. IC0-constraints it is* NP*-hard.*

Our proof of the upper bound for $STABLE(PROG^-)$ is based on the following assertion.

Lemma 3 *For any DDB* $\mathcal{B} \in PROG^-$ *and any DB state* $\mathcal{E}$ *the following statements are equivalent:*

(i) $\mathcal{B}$ *is* δ*-stable in* $\mathcal{E}$;

(ii) there exists a trajectory starting in $\mathcal{E}$ *in which some production is applied twice.*

Next theorems show that for the other classes we consider, the promise and the stability problems are again of the same complexity.

Theorem 5

(1) The problem $STABLE^{\forall}(PROG)$ *is* $PSPACE$*-complete.*

(2) The problem $STABLE^{\forall}(PROF)$ *is* $SPACE(2^{poly})$*-complete.*

(3) The problem $STABLE^{\forall}(GDS)$ *is* $PSPACE$*-complete.*

(4) The problem $STABLE^{\forall}(FDS)$ *is* $SPACE(2^{poly})$*-complete.*

Our proofs of upper bounds use the folowing auxilliary notions and facts. Let $\mathcal{B} =< \mathcal{I} \cup \{ \ :- \ g\}, \ \mathcal{A} \cup \{ \ :- \ R\}, \equiv_i >$ be a DDB. In the case where the update $\mathcal{E} \stackrel{g}{\vdash}_{\mathcal{I}} \mathcal{E}'$ is effected by a single application of a production $p \in \mathcal{I}$, we will simply write $\mathcal{E} \stackrel{p}{\vdash} \mathcal{E}'$. Let $\delta =< \mathcal{D}^+, \mathcal{D}^- >$. Let us denote the set of all extensional atoms occurring in $\mathcal{B}$ and δ by $\mathcal{K}^e$. We define the following bipartite graph $\Gamma = (\Pi_1, \Pi_2, \Delta)$. $\Pi_1 = 2^{\mathcal{K}^e}$ and $\Pi_2 = \{\mathcal{E} \in \Pi_1 \mid \mathcal{E}$ satisfies the IC$\}$ are two node components of Γ. The set of its arrows is defined as $\Delta = \{(\mathcal{E}_1, \mathcal{E}_2) \mid \mathcal{E}_1 \in \Pi_1, \mathcal{E}_2 \in \Pi_2, \ \mathcal{E}_1 \stackrel{\mathcal{D}_0^+, \mathcal{D}_0^-}{\longrightarrow} \mathcal{E}_2$ for some $\mathcal{D}_0^+ \subseteq \mathcal{D}^+, \mathcal{D}_0^- \subseteq \mathcal{D}^-\} \cup \{ \ (\mathcal{E}_2, \mathcal{E}_1) \mid \mathcal{E}_2 \in \Pi_2, \mathcal{E}_1 \in \Pi_1, \ \mathcal{E}_2 \stackrel{p}{\vdash} \mathcal{E}_1$ for some $p \in \mathcal{I}\}$. The graph Γ is very closely related to δ-stable trajectories. For $\mathcal{B} \in PROG$ $(\mathcal{B} \in GDS)$ this relation can be formulated as the following

Lemma 4 *There is a* δ*-stable trajectory in* $\mathcal{B}$ *starting in a state* $\mathcal{E}$ *iff there is a node* $\mathcal{E}_1 \in \Pi_2$ *such that there is a path in* Γ *from* $\mathcal{E} \cap \mathcal{K}^e \in \Pi_1$ *to* $\mathcal{E}_1$ *and there is a nonempty loop in* Γ *from* $\mathcal{E}_1$ *to* $\mathcal{E}_1$.

Γ with the only difference that Π_1 is the set of all states of $\mathcal{B}$. So instead of Lemma 4 we use here its following counterpart.

Lemma 5 *There is a δ-stable trajectory in $\mathcal{B}$ starting in a state $\mathcal{E}_0 \in \Pi_1$ iff there is a node $\mathcal{E}_1 \in \Pi_2$ such that there is a path in Γ from $\mathcal{E}_0$ to $\mathcal{E}_1$ and there is a nonempty loop in Γ from $\mathcal{E}_1$ to $\mathcal{E}_1$.*

It is interesting that $\forall$-stability is of no greater complexity then the stability in a DB state.

Theorem 6

(1) The problem $STABLE(PROG)$ is $PSPACE$-complete.
(2) The problem $STABLE(PROF)$ is $SPACE(2^{poly})$-complete.
(3) The problem $STABLE(GDS)$ is $PSPACE$-complete.
(4) The problem $STABLE(FDS)$ is $SPACE(2^{poly})$-complete.

Of course, the problem of stability is undecidable in the class $PROD$. The reason is the same as for the promise problem.

6 Conclusion

We have proposed a new model of interactive behavior of deductive databases and some characteristics of their steadiness with respect to the action of their active media. Even our very simplified examples show that various kinds of dynamic systems interacting with active media can be simulated by DDBs, and the properties of their steady behavior can be formulated and explored in the terms of this work. We consider here two weakest properties of steady behavior - promise and stability, showing that they are solvable in several classes of DDBs of interest in applications. There are other forms of steadiness we have not explored in this work. One of them is a $uniform\ \delta - stability$ which means that for **any** update of a DDB $\mathcal{B}$ there is a $\delta-$bounded disturbance restoring the IC. Another one is a $\delta - homeostacity$ which is dual to the stability property. It characterizes the so called *d-resistant* trajectories of $\mathcal{B}$, i.e., in the notation of the Definition 6, those in which all states $\mathcal{E}_i$, $i = 0, 1, ...$ satisfy the IC. Informally speaking, this means that for any external disturbance there exists an update which restores the IC. $\mathcal{B}$ is $\delta - homeostatic$ if it has this property for all $\delta-$bounded disturbances. We shall explore δ-homeostacity elsewhere.

Acknowledgements

This work was sponsored by the Russian Fundamental Studies Foundation (Grant 93-012-627).

[1] Abiteboul, S., Vianu, V., *Datalog extensions for database queries and updates,* Rapp. de Recherche n. 900, INRIA-Rocquencourt. Septembre, 1988.

[2] Aho, A.V., Ullman, J.D., *Universality of data retrieval languages, Proc. 6th ACM Symp. on Principles of Prog. Languages,* San Antonio, Texas, 110-117, 1979.

[3] Apt, K.R., Blair, H. and Walker A., *Towards a theory of declarative knowledge.* in: J. Minker (ed.) *Foundations of deductive databases and logic programming.* Morgan Kaufman Pub., Los Altos, 89-148, 1988.

[4] Bonner, A.J., *Hypothetical Datalog: complexity and expressibility.* Theoretical Computer Science, 76, 3-51, 1990.

[5] Bonner,A.G., Kifer, M., *Transaction logic programing,* In *Proc. of the Tenth Intern. Conf. on Logic Programming.* The MIT Press, 257-279, 1993.

[6] Chandra, A.K., Harel, D., *Computable queries for relational databases,* Journal of Computer and System Sciences, vol. 21, n.2 1980, 156-178.

[7] Dikovsky, A.Ja., *Linear time solutions to the problems related to automatic synthesis of acyclic programs.* Programming, 3, 1985.

[8] Dikovsky, A.Ja., *On computational complexity of Prolog programs.* Theoretical Computer Science, 119, 63-102, 1993.

[9] Dung, P.M., Representing actions in logic programming and its application in database updates. In *Proc. of the Tenth Intern. Conf. on Logic Programming,* The MIT Press, 222-238, 1993.

[10] Eiter, T., Gottlob, G., *On the complexity of propositional knowledge base revision, updates, and counterfactuals.* Artificial Intelligence, vol. 57, 227-270, 1992.

[11] Gallaire, H., Minker, J., Nicolas, J.-M., Logic and databases: a deductive approach, ACM Computing Surveys, vol. 16, n.2, 153-185, 1984.

[12] Hull, R., *Relative information capacity of simple relational database schemata.* SIAM J. of Computing, vol. 15, n.3 856-886, 1986.

[13] Katsuno, H., Mendelzon, A. O., *Propositional knowledge base revision and minimal change.* Artificial Intelligence, vol. 52, 253-294, 1991.

[14] Manchanda, S., Warren, D.S., *A logic-based language for database updates.* In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming,* Morgan-Kaufmann, Los Altos, CA, 363-394, 1988.

[15] Naqvi, S., Krishnamurthy, R., *Database updates in logic programming.* In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 251-262, 1988.

[16] Sadri, F., Kowalski, R., *A theorem proving approach to database integrity.* In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming,* Morgan-Kaufmann, Los Altos, CA, 313-362, 1988.

APPENDIX

A building company constructs dwelling houses. The financial support of its activity is based on payments from customer's contracts. The payments are put on the current account and are spent on materials and on monthly employee's salary.

1. A fragment of the DB scheme: $current\_account(Sum)$, $spare\_contract$ $(Customer)$, $contract\_state$ $(Contract\_number, Customer, Ready)$, ($Ready$ takes values in $\{no, yes\}$), $balance(Contract\_number$, Sum_to_pay, $Paid)$, $received$ $(Contract\_number$, $Paid\_sum)$, $month\_salary(Sum)$ (total month salary to pay), $paid\_in(Year$, $Month)$, $materials(Price)$, $assembly(Price)$, $profit\_rate(Percent)$, $minimal\_sum(Sum)$, (the minimum possible remainder of the sum on the account),...

2. A fragment of the intensional part of DDB:

$execute$:– /* Execution of a contract */
$salary\_paid$, $contract\_state(N, C, no)$, $materials(MPrice)$,
$assembly(APrice)$, $current\_account(CS)$, $CS > MPrice$,
$change(current\_account(CS), current\_account(CS - MPrice))$,
$change(contract\_state(N, C, no), contract\_state(N, C, yes))$,
$profit\_rate(Percent)$, $Price = (MPrice + APrice) * (1 + Percent/100)$,
$insert(balance(N, Price, 0))$.

$receive$:– /* Payments and accounts */
$received(N, Sum)$, $balance(N, S, R)$,
$change(balance(N, S, R), balance(N, S, R + Sum))$,
$change(current\_account(CS), current\_account(CS + Sum))$,
$delete(received(N, Sum))$.

pay :– /* Payment of salary */
$previous\_month(Y, M)$, /* built-in predicate */
$\neg paid\_in(Y, M)$, $month\_salary(T)$, $current\_account(C)$, $C > T$,
$change(current\_account(C), current\_account(C - T))$,
$insert(paid\_in(Y, M))$.

$salary\_paid$:–
$previous\_month(Y, M)$, $paid\_in(Y, M)$.

$search$:– /* Search for a contract */
$salary\_paid$, $\neg contract\_state(\_, \_, no)$, /* nothing to do */
$spare\_contract(Customer)$, $assign\_number(Number)$,
$insert(contract\_state(Number, Customer, no))$,
$delete(spare\_contract(Customer))$.

3. Integrity constraints (on finances only):

$admissible$:–
$current\_account(T)$, $minimal\_sum(Min)$,
$findall(P, (balance(\_, S, C), S > C, P\ is\ S - C), LP)$,
$sum(LP, Free), T + Free > Min$. /* Free - debit sum */

As we see in this example, any update of this DDB changes current states. Any operation implies consumption of materials and money for payments, which can lead to DB states violating the IC. These resources can only be restored externally in the form of new accessible contracts which are sought by the update $search/0$. E.g., in the state $\mathcal{E}_0$ containing facts $materials(800)$, $assembly(700)$, $profit\_rate(50)$, $current\_account(2200)$, $month\_salary(900)$ and $minimal\_sum(1000)$, this DDB

stability is lost if the monthly salary raises to $month\_salary(4500)$, while the other parameters and δ remain unchanged. The point is that in this state no update can be applied.