Recognition of Deductive Data Base Stability *

Michael I. Dekhtyar[1] and Alexander Ja. Dikovsky[2]

[1] Dept. of CS, Tver St. Univ. 33 Zheljabova str. Tver, Russia, 170013
dekhtyar@tversu.ac.ru
[2] Keldysh Inst. for Appl. Math. 4 Miusskaya sq. Moscow, Russia, 125047
dikovsky@keldysh.msk.ru

Abstract. New notions are introduced of globally stable behavior of deductive data base reacting to stimuli of its active medium. Various problems of integrity constraints restoration after updates fit this general frame. We explore the computational complexity of the problem of stability of a deductive data base in a given DB state with respect to its medium.

1 Introduction

In this work the deductive data bases (*DDBs*) serve as a model of interactive discrete dynamic systems with complex states described by relations over values of multiple parameters. At any given moment the system state is represented by a data base state (*DB state*), i.e. a finite set of extensional facts. Possible actions (moves) of the system in given states can be represented by some binary relation $\vdash$. External stimuli of the active medium of the system (*disturbances*) also change states and can be represented by another binary relation $\xrightarrow{d}$. In our approach the fundamental difference between these two relations is determined by the level of our knowledge about them. The theory of behavior of the system is assumed to be known. This being so, we describe it in the form of a logic program with updates $\mathcal{P}$ over DB states. More specifically, a move from state $\mathcal{E}_1$ to state $\mathcal{E}_2$, i.e. the action $\mathcal{E}_1 \vdash \mathcal{E}_2$ is described as a computation of some predefined goal $:- a$ of $\mathcal{P}$ transforming the first DB state to the second: $\mathcal{E}_1 \overset{a}{\vdash}_{\mathcal{P}} \mathcal{E}_2$. As for the disturbances, one can always estimate the set of possible disturbances and evaluate the effect of a committed disturbance on the current state. However *one cannot predict in the given state which disturbance is to be committed*. E.g., in a bank there exists the list of feasible services required by its clients (disturbances) and the corresponding transactions (actions), but one cannot predict exactly which orders for the services emerge at the given moment.

Local behavior of the system in the current state $\mathcal{E}_0$ is described as one interaction with its medium in this state. There are two types of interactions depending on what comes first, a medium disturbance or a system action: $\mathcal{E}_0 \xrightarrow{d}$

* This work was sponsored by INTAS (Grant 94-2412) and the Russian Fundamental Studies Foundation (Grants 95-01-01321 and 96-01-00395).

$\mathcal{E}_1^* \overset{a}{\vdash}_{\mathcal{P}} \mathcal{E}_1$ and $\mathcal{E}_0 \overset{a}{\vdash}_{\mathcal{P}} \mathcal{E}_1^* \overset{d}{\longrightarrow} \mathcal{E}_1$. Normally, one can distinguish between feasible and not feasible interactions depending on a criterion of admissibility of system states. Each sensible interaction applies to an admissible state $\mathcal{E}_0$ and yields an admissible state $\mathcal{E}_1$. However, the intermediate state $\mathcal{E}_1^*$ may in general be inadmissible, in which case the reaction compensates for the ruinous stimuli. We represent the admissibility criterion by some *integrity constraints (IC)* expressed by a formula or a (logic) program Φ over DB states. In terms of the IC the feasibility of the interaction is expressed as follows: the interaction of the form $\mathcal{E}_0 \overset{d}{\longrightarrow} \mathcal{E}_1^* \overset{a}{\vdash}_{\mathcal{P}} \mathcal{E}_1$ or $\mathcal{E}_0 \overset{a}{\vdash}_{\mathcal{P}} \mathcal{E}_1^* \overset{d}{\longrightarrow} \mathcal{E}_1$ is *feasible* if $\mathcal{E}_0 \models \Phi$ and $\mathcal{E}_1 \models \Phi$. Thus, the discrete dynamic system is represented by the *deductive data base (DDB)* $\mathcal{B} =< \mathcal{P} \cup GOALS, \Phi >$, its medium is described by the relation $\overset{d}{\longrightarrow}$, and their interaction is expressed in terms of feasible interactions.

Global behavior of the system in the current state $\mathcal{E}_0$ is represented by sequences of (feasible) interactions starting in $\mathcal{E}_0$: the *(feasible) trajectories*. Basically, we have two types of trajectories corresponding to the two types of interactions above: *disturbance-action* trajectories

$$\mathcal{E}_0 \overset{d}{\longrightarrow} \mathcal{E}_1^* \overset{a_1}{\vdash} \mathcal{E}_1 \overset{d}{\longrightarrow} \mathcal{E}_2^* \overset{a_2}{\vdash} \mathcal{E}_2 \ldots$$

and *action-disturbance* trajectories

$$\mathcal{E}_0 \overset{a_1}{\vdash} \mathcal{E}_1^* \overset{d}{\longrightarrow} \mathcal{E}_1 \overset{a_2}{\vdash} \mathcal{E}_2^* \overset{d}{\longrightarrow} \mathcal{E}_2 \ldots$$

Infinite feasible disturbance-action trajectories represent the homeostatic behavior of the DDB in spite of disturbances of its medium: the DDB always manages to restore the IC along this trajectory if and when they are violated by the medium disturbances. Such trajectories are called *homeostatic*. Dually, the infinite feasible action-disturbance trajectories represent the stable behavior of the DDB owing to disturbances of its medium: the medium always compensates along the trajectory for possible ruinous actions of the DDB. They are called *stable*.

Trajectories of each type form a tree with the root $\mathcal{E}_0$: $T_{da}(\mathcal{E}_0)$ and $T_{ad}(\mathcal{E}_0)$ respectively. A number of natural properties of interactive behavior of $\mathcal{B}$ in a given DB state can be formalized in terms of these trees.

Definition 1 *Let $\mathcal{B}$ be a DDB and $\overset{d}{\longrightarrow}$ be a disturbance relation. Let $Q_1, Q_2 \in \{\forall, \exists\}$. Then $\mathcal{B}$ is $d-Q_1Q_2$-*stable in DB state *$\mathcal{E}_0$ if in the tree $T_{ad}(\mathcal{E}_0)$ there is a Q_1Q_2-subtree in which all branches are infinite stable trajectories. $\mathcal{B}$ is $d-Q_1Q_2$-*homeostatic in DB state *$\mathcal{E}_0$ if in the tree $T_{da}(\mathcal{E}_0)$ there is a Q_1Q_2-subtree in which all branches are infinite homeostatic trajectories.*

In [4] we introduce $\exists\exists$-stability and explore its computational complexity. In [5] the $\forall\exists$-homeostaticity is introduced and investigated.

Another interesting class of properties of steady interactive behavior is the class of *total* properties of feasible interactions, i.e. properties which hold *in any* DB state. One such property has attracted widespread attention because it corresponds to an important and long known scenario in DDBs applications. This

is the property of *total homeostaticity:*

$$\forall \mathcal{E}, \mathcal{E}_1^* \ (\mathcal{E} \models \Phi \ \& \ \mathcal{E} \xrightarrow{d} \mathcal{E}_1^* \Rightarrow \exists a, \mathcal{E}_1 \ (\mathcal{E}_1^* \overset{a}{\vdash} \mathcal{E}_1 \ \& \ \mathcal{E}_1 \models \Phi))$$

equivalent to $\forall\exists$-homeostaticity in any admissible DB state. The so called revision algorithms [10] are applied to DB states of a propositional DDB, violating the IC in order to transform them to DB states satisfying the IC. In fact the same problem is investigated in [11,12]. Here the IC is expressed by a much more general revision program $\mathcal{P}$ and partial algorithms are proposed transforming a given inadmissible DB state $\mathcal{E}$ into a stable model of $\mathcal{P}$ (in the sense of [7]), "induced" by $\mathcal{E}$. In both cases updates are treated implicitly: the initial inadmissible DB state may be viewed as the result of a ruinous update. In [9] elementary updates (insertion or deletion of a ground literal) are explicit, the IC Φ is expressed by Datalog programs extended by simple productions (rules), and algorithms are described "constructing" a model of Φ consistent with an input update. So in these papers DB updates correspond to destructive disturbances and the algorithms enforcing the IC correspond to restoring DDB actions in our terms. This is in fact the basic scenario in the active data bases (cf. [3,8]), where the disturbances are either the elementary updates or some external events recognizable by action rules. Here again the total homeostaticity is guaranteed by some strategy over actions.

It should be stressed that while in the cited literature one is always interested in models which **guarantee** the total homeostaticity, we are investigating the inverse problem: *given a DB state* $\mathcal{E}$, *a DDB* $\mathcal{B}$ *in certain class of DDBs* $\mathcal{C}$, *an IC* Φ *in a class of ICs* $\mathcal{I}$, *and a disturbance relation* $\xrightarrow{d}$ *restricted in some sense, one should find out whether* $\mathcal{B}$ *is stable (homeostatic) in* $\mathcal{E}$ *with respect to* $\xrightarrow{d}$.

As far as we know, the property of stable behavior went unnoticed in the literature. Meanwhile, this property is peculiar to many interactive systems whose operation involves consumption, compensation, and restoration of some resources. The typical examples are plants, trade enterprises, warehouses, etc. Their medium is formed of consumers and some financial or material sources necessary to support their successful and unlimited operation. In this paper we explore the computational complexity of the property of $\forall\exists$-stability and $\forall\forall$-stability of DDBs. The $\forall\exists$-stability means that for any DDB action violating the IC there exists some disturbance of the medium, properly restricted, which restores the IC. In the case of the $\forall\forall$-stability each restricted disturbance should restore the IC. Not surprisingly, these properties are undecidable in the class of all DDBs. However, they are solvable in various classes of DDBs via reasonable classes of IC of interest for real applications. E.g., we prove that in the nonrecursive $DATALOG^{\neg}$ case where DDBs are structureless production systems the $\forall\forall$-stability is exponential-space complete, while the $\forall\exists$-stability is deterministic-double-exponential-time complete. If DDBs are ground (i.e. variableless) production systems, then the first property is $PSPACE$-complete, while the other is deterministic-exponential-time complete. In some more narrow classes of DDBs the complexity of both properties is quite tractable. For example, in the class of ground production systems which never delete facts from

states, and under monotonic integrity constraints they are solved in polynomial time. Although in many interesting situations the complexity of stability recognition appears to be too high, there are natural methods of factorization of the DB states space (described elsewhere), which can substantially decrease the size of the problems in practical DDB applications.

2 Basic Notation and Definitions

We consider a 1st order language $\mathbf{L}$ in a signature containing: pairwise disjoint sets $\mathbf{P}^e$ of *extensional predicates*, $\mathbf{P}^q$ of *intensional query predicates,* and $\mathbf{P}^u$ of *intensional update predicates,* and a set of function (and constant) symbols $\mathbf{F}$. $\mathbf{H}$ denotes the Herbrand universe of $\mathbf{L}$, and $\mathbf{B}^e$, $\mathbf{B}^i$ are for the extensional and intensional Herbrand bases. *DB states* are finite subsets of $\mathbf{B}^e$. We consider *logic programs with updates* in $\mathbf{L}$ with clauses of the form $Head :- Body$, where $Head$ is an intensional atom $p(\bar{t})$ with $p \in \mathbf{P}^q \cup \mathbf{P}^u$, and $Body$ is a (possibly empty) sequence of:

- literals of the form $q(\bar{u})$ or $\neg r(\bar{v})$ with $r \notin \mathbf{P}^u$, (i.e. only atoms with extensional and intensional query predicates, can be negated),
- elementary DB updates $insert(Fact)$, $delete(Fact)$, where $Fact$ is an extensional atom,
- assignments of the form $X := e$ and arithmetical constraints of the form $e_1 < e_2$ with e, e_1, e_2 being arithmetical expressions.

So we subsume that numerals and arithmetic operators are included into $\mathbf{F}$. The semantic scheme below will show that they are interpreted in the standard way.

Definition 2 *Let $\mathcal{P}$ be a logic program with updates. We say that a predicate p* refers to *a predicate q if there is a clause $p(\bar{t}) :- \bar{\alpha}, q(\bar{s}), \bar{\beta}$ in $\mathcal{P}$. 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 $G(\mathcal{P})$ of the relation "depend on" are called* cliques. *A predicate (a goal, a subgoal) is* stationary *if it does not depend on elementary updates, assignments, and update predicates.*

We distinguish the following important class of logic programs with updates.

Definition 3 *A logic program with updates $\mathcal{P}$ is* stratified *if:*

1) all query predicates are stationary;

2) let $\mathcal{P}^q$ be the set of definitions of all query predicates in $\mathcal{P}$: $\mathcal{P}^q = \mathcal{P}(\mathbf{P}^q)$. Then for any DB state $\mathcal{E}$ the logic program $\mathcal{P}^s = \mathcal{P}^q \cup \mathcal{E}$ is stratified in the sense of [1].

A logic program with updates $\mathcal{P}$ is update (u-) stratified *if it is stratified and in every clause $p(\bar{u}) :- \alpha_1, ..., \alpha_i, q(\bar{v}), \alpha_{i+1}, ..., \alpha_r$ in which p, q are update predicates belonging to the same clique of $G(\mathcal{P})$, all predicates in α_i are* stationary.

The essence of the u-stratifiability is in point 3 which says that DB updates are available only at the steps where a clique is changed. This constraint provides tight bounds to the number of elementary DB updates fired in the course of a transaction caused by an update predicate call.

Example 1 *Let* $\mathbf{P}^e = \{e/1\}$ *and* $\mathbf{P}^u = \{a/0, b/0, c/0\}$. *Then the following program is u-stratified:*
$a \ :- \ e(0), b.$
$a \ :- \ insert(e(0)).$
$b \ :- \ \neg e(0), a.$
$b \ :- \ e(I), I_1 := I + 1, \neg e(I_1), delete(e(I)), insert(e(I_1)), c.$
$c \ :- \ e(I), I > 0, I_1 := 2 * I, \neg e(I_1), insert(e(I_1)).$

Here the dependency subgraph on $G(\mathbf{P}^u)$ consists of two cliques: $\{a, b\} \rightarrow \{c\}$.

Throughout this paper we consider only stratified logic programs with updates and call them simply logic programs with updates.

The operational semantics of a logic program with updates $\mathcal{P}$ is represented by a relation of the form $\mathcal{E} \cup \{ \ :- u\} \ \overset{\theta}{\vdash_{\mathcal{P}}} \ \mathcal{E}' \cup \{ \ :- v\}$. Intuitively it says that $\mathcal{P}$ reduces via the answer substitution θ the goal $:- u$ when in the input DB state $\mathcal{E}$, to the goal $:- v$ and changes $\mathcal{E}$ to the output DB state $\mathcal{E}'$. The following rule schemes 1 - 7 define this relation formally (index $\mathcal{P}$ is dropped). The letters u, v, ϕ in these schemes stand for sequences of subgoals, $\Box$ is for the empty goal, θ, θ', and σ are for substitutions and for an MGU, ε denotes the identity substitution, e, e' denote arithmetical expressions, A, H stand for intensional atoms, $Fact$ is for an extensional atom, and $stable\_mod(\mathcal{E}' \cup \mathcal{P}^q)$ denotes the (unique) stable model (in the sense of [7]) of the stratified program $\mathcal{E}' \cup \mathcal{P}^q$.

1. $$\frac{}{\mathcal{E} \cup \{ \ :- u\} \overset{\varepsilon}{\vdash} \mathcal{E} \cup \{ \ :- u\}}$$

2. $$\frac{\mathcal{E} \cup \{ \ :- u\} \overset{\theta}{\vdash} \mathcal{E}' \cup \{ \ :- A, v\}, \ (H \ :- \phi) \in \mathcal{P} \ \textit{and} \ A \circ \theta \circ \sigma = H \circ \sigma, \ \textit{or} \ A \circ \theta \circ \sigma \in \mathcal{E}' \ \textit{and} \ \phi = \emptyset}{\mathcal{E} \cup \{ \ :- u\} \overset{\theta \circ \sigma}{\vdash} \mathcal{E}' \cup \{ \ :- \phi, v\}}$$

3. $$\frac{\mathcal{E} \cup \{ \ :- u\} \overset{\theta}{\vdash} \mathcal{E}' \cup \{ \ :- X := e, \ v\}, \ e \circ \theta \ \textit{is ground}, \ \theta' = \theta[X \backslash val(e, \theta)]}{\mathcal{E} \cup \{ \ :- u\} \overset{\theta'}{\vdash} \mathcal{E}' \cup \{ \ :- v\}}$$

4. $$\frac{\mathcal{E} \cup \{ \ :- u\} \overset{\theta}{\vdash} \mathcal{E}' \cup \{ \ :- e < e', \ v\}, \ e \circ \theta, \ e' \circ \theta \ \textit{are ground and} \ val(e, \theta) < val(e', \theta)}{\mathcal{E} \cup \{ \ :- u\} \overset{\theta}{\vdash} \mathcal{E}' \cup \{ \ :- v\}}$$

5. $$\frac{\mathcal{E} \cup \{ \ :- u\} \overset{\theta}{\vdash} \mathcal{E}' \cup \{ \ :- insert(Fact), \ v\}, \ Fact \circ \theta \ \textit{is ground}}{\mathcal{E} \cup \{ \ :- u\} \overset{\theta}{\vdash} (\mathcal{E}' \cup \{Fact \circ \theta)\}) \cup \{ \ :- v\}}$$

6. $$\frac{\mathcal{E} \cup \{ \ :- u\} \overset{\theta}{\vdash} \mathcal{E}' \cup \{ \ :- delete(Fact), \ v\}, \ Fact \circ \theta \ \textit{is ground}}{\mathcal{E} \cup \{ \ :- u\} \overset{\theta}{\vdash} (\mathcal{E}' \backslash \{Fact \circ \theta)\}) \cup \{ \ :- v\}}$$

7. $$\frac{\mathcal{E}\cup\{\ :\!-\ u\}\overset{\theta}{\vdash}\mathcal{E}'\cup\{\ :\!-\ \neg A,\ v\},\ stable\_mod(\mathcal{E}'\cup\mathcal{P}^q)\models\neg A\circ\theta}{\mathcal{E}\cup\{\ :\!-\ u\}\overset{\theta}{\vdash}\mathcal{E}'\cup\{\ :\!-\ v\}}$$

These rules support leftmost strategy of subgoals evaluation. Rules scheme 1 introduces the identity answer substitution and serves as an axioms scheme in the derivations. The premises of rules schemes 2-7 consist of two parts: (i) a sequent of the form $\mathcal{E}\cup\{\ :\!-\ u\}\ \overset{\theta}{\vdash_{\mathcal{P}}}\ \mathcal{E}'\cup\{\ :\!-\ g,v\}$, where g is the subgoal to be resolved by the rule, and (ii) a condition of applicability of the rule. The conclusion of each scheme consists of the derived sequent of the same form. Rule 2 is the standard resolution step which resolves the leftmost subgoal A by a new variant of the clause $(H\ :\!-\ \phi)\in\mathcal{P}$, or by the fact $A\in\mathcal{E}'$. In this rule σ denotes an MGU. Rules 3 and 4 deal with arithmetic. All variables in expressions e,e' should be instantiated (by numbers). Rules 3, 4 do not change DB states. $val(e,\theta)$ denotes the value of the arithmetical expression e in the environment θ. The assignment $X:=e$ changes the answer substitution θ binding X by $val(e,\theta)$. Rules 5 and 6 are the only rules changing DB states. They describe the effect of elementary updates. Specifically, rule 5 changes the DB state $\mathcal{E}'$ to the DB state $\mathcal{E}'\cup\{Fact\circ\theta)\}$. Rule 7 indicates that the negation is resolved in the (unique) stable model $stable\_mod(\mathcal{E}'\cup\mathcal{P}^q)$. It should be noted that our choice of negation semantics is quite arbitrary. Any negation semantics "effectively computable" on finite models will do here.

Rules 1 - 7 associate with each update predicate $a/0$ the following nondeterministic transaction operator $\overset{a}{\vdash_{\mathcal{P}}}$ on DB states: $\mathcal{E}\ \overset{a}{\vdash_{\mathcal{P}}}\ \mathcal{E}'\Longleftrightarrow\mathcal{E}\cup\{\ :\!-\ a\}\ \overset{\theta}{\vdash_{\mathcal{P}}}\ \mathcal{E}'\cup\{\Box\}$ for some θ.
A DDB $\mathcal{B}=<\mathcal{P}\cup\{\ :\!-\ a_1,\ldots,\ :\!-\ a_n\ \},\ \Phi>$ includes an intensional logic program with updates $\mathcal{P}$, a predefined set of 0-ary goals $\{a_1,\ldots,a_n\}$ implementing DB state transactions, and integrity constraints (IC) embodied by a relation Φ on DB states. The behavior of DDB $\mathcal{B}$ is defined by the transaction relation $\vdash_{\mathcal{B}}$ which is the union of relations $\overset{a_i}{\vdash_{\mathcal{P}}}$, $i=1,...,n$.

3 Problem Classes

We explore the computational complexity of the following problems. Given a DDB $\mathcal{B}=<\mathcal{P}\cup\{\ :\!-\ a_1,\ldots,\ :\!-\ a_n\ \},\ \Phi>$, a disturbance relation $\overset{d}{\longrightarrow}$ (in some representation), and a DB state $\mathcal{E}$, one should check whether $\mathcal{B}$ is $\forall\exists$- (respectively $\forall\forall$-) stable in $\mathcal{E}$ with respect to $\overset{d}{\longrightarrow}$.

It is readily seen that both problems are undecidable if no restrictions are imposed on their main parameters: logic programs with updates, ICs and disturbance relations. In this section we introduce various restrictions which come about naturally and guarantee the decidability of these problems.

Logic programs. Logic programs we consider are always u-stratified. However, this condition does not assure the solvability of the stability recognition prob-

lems by itself. We classify the restrictions to logic programs by the form of their clauses.

Definition 4 *A logic program* $\mathcal{P}$ *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 either variables or constants.* $\mathcal{P}$ *is* expanding *if the update delete/1 is not used in its clauses. We call* $\mathcal{P}$ branching *if it is not recursive, i.e. its dependency graph* $G(\mathcal{P})$ *has no cycles. 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* $q \text{ :- } Con_1, \ldots, Con_k, Act_1, \ldots, Act_m$ *where each* Con_i *is an extensional literal and each* Act_j *is an elementary update. These are exactly the productions used in AI, so we keep their usual syntax:*

$$Con_1 \& \ldots \& Con_m \Longrightarrow Act_1, \ldots, Act_n.$$

We consider the following classes of u-stratified logic programs:

- USF is the class of u-stratified flat programs;
- USG is the class of u-stratified ground programs;
- $BRAF$ is the class of branching flat programs;
- $BRAG$ is the class of branching ground programs;
- $BRAG^x$ is the class of expanding programs in $BRAG$;
- $BRAG^+ = BRAG^p \cap BRAG^x$.

For each class of branching programs in this list we consider the corresponding class of productional logic programs: $PROF$, $PROG$, $PROG^x$, and $PROG^+$. The last class is the smallest one. Programs in $PROG^+$ have only productional rules which do not use negation and the update $delete/1$.

Integrity constraints. For a DB state $\mathcal{E}$ and an IC Φ we denote by $\mathcal{E} \models \Phi$ the relation "Φ is true in $\mathcal{E}$". The nature of IC Φ is immaterial here. What is essential is that Φ has a constructive representation in some formal language (a logical formula, a set of productions, or a polynomial time Turing machine, etc.) for which there is a universal algorithm checking the property $\mathcal{E} \models \Phi$.

Definition 5 *An IC* Φ *is preserved upwards if whenever* $\mathcal{E} \subseteq \mathcal{E}'$ *and* $\mathcal{E} \models \Phi$, *then* $\mathcal{E}' \models \Phi$.

Below we analyze the complexity of stability problems using ICs in the following three classes.

IC_0 : Φ is preserved upwards and the problem $\mathcal{E} \models \Phi$ is in **P**.

IC_1 : the problem $\mathcal{E} \models \Phi$ is in **P**.

IC_2 : the problem $\mathcal{E} \models \Phi$ is in **PSPACE**.

E.g., the class IC_0 contains ICs expressed by positive closed quantifier-free formulas, and some monotone graph properties (e.g., connectivity). The well-known functional dependencies and a number of properties of model size (e.g., parity) belong to the class IC_1. The class IC_2 contains the ICs expressed by the 1st order formulas, etc.

Disturbance relations. The constraints we impose on disturbance relations $\xrightarrow{d}$ are expressed in terms of the following *change set:*

$c(d, \mathcal{E}) = \{(D^+, D^-) |$ *there is* $\mathcal{E}' : \ \mathcal{E} \xrightarrow{d} \mathcal{E}', \ D^+ = \mathcal{E}' \setminus \mathcal{E}, \ and \ D^- = \mathcal{E} \setminus \mathcal{E}'\}$.

Let $\delta = (\Delta^+, \Delta^-)$ be a pair of two finite subsets of $\mathbf{B}^e$. We define a δ-*disturbance* as a binary relation on DB states such that for every $\mathcal{E}$ $c(\delta, \mathcal{E}) = \{(D^+, D^-) | D^+ \subseteq \Delta^+, D^- \subseteq \Delta^-\}$.

We finish this section by a definition of stability problem. For a class of logic programs with updates $\mathbf{P}$, a class of ICs $\mathbf{I}$, and a pair of quantifiers $\mathbf{Q_1}, \mathbf{Q_2}$ $STABLE^{\mathbf{Q_1 Q_2}}(\mathbf{P} + \mathbf{I}) =$

$\{(< \mathcal{P}, \Phi >, \delta, \mathcal{E}) \mid \mathcal{P} \in \mathbf{P},\ \Phi \in \mathbf{I},\ < \mathcal{P}, \Phi >\ is\ \delta - Q_1 Q_2 - stable\ in\ \mathcal{E}\}$.

Example 2 *Imagine a population of organisms in which the breeding rules are phrased in terms of values of some characteristic feature as follows: a new individual emerges only if its parents have some different values of this feature, and exactly one value is inherited. For the case of three possible values* a, b, c *this rule is expressed very simply in our terms.*
Productional logic program $\mathcal{P}$ *:*

$$\alpha \& \beta \Longrightarrow delete(\beta), \quad \alpha, \beta \in \{a, b, c\}.$$

IC $\Phi = (a\&b \vee a\&c \vee b\&c)\&\neg(a\&b\&c)$. $\delta = (\{a, b, c\}, \emptyset)$.
It is clear that this biological system is δ-$\forall\exists$-*stable in all admissible states. However, it is not* δ_1-$\forall\exists$-*stable in any DB state when* $\delta_1 = \{a\}$.

4 Complexity of $\forall Q$-stability

In this section we establish complexity bounds to the problems of $\forall\forall-$ and $\forall\exists$-stability in the classes of DDBs defined above. Our results show that such features of DDBs as nonmonotone ICs, negations in conditions, deletions of facts, and the use of variables in logic programs can increase the complexity of these problems dramatically, whereas the restricted form (u-stratifiability) of recursion does not change the complexity bounds. Because of space limitations some proofs are omitted here, the others are only sketched.

In the theorems to follow the finite sets of facts $\mathcal{E}$ (Δ^+, Δ^-, etc.) are supposed to be represented in some standard coding. The size of the code $|\mathcal{E}|$ is $O(C_\mathcal{E} * FN_\mathcal{E})$, where $C_\mathcal{E}$ depends on maximal arity of extensional predicates and on constants used in $\mathcal{E}$, and $FN_\mathcal{E}$ denotes the number of facts in $\mathcal{E}$. This differs from the encoding used in the finite model complexity where predicates are represented by the sequences of their values on *all* possible data. Below for each Turing machine complexity class $CLASS(F)$ $ACLASS(F)$ denotes the corresponding complexity class for the alternating Turing machines [2].

Theorem 1

(1) The problem $STABLE^{\forall\forall}(PROG^+ + IC_0)$ *is solved in polynomial time.*
(2) The problem $STABLE^{\forall\forall}(BRAG^+ + IC_1)$ *is co-NP-complete.*
(3) The problem $STABLE^{\forall\forall}(PROG^x + IC_1)$ *is* $PSPACE$*-complete.*
(4) The problem $STABLE^{\forall\forall}(BRAG + IC_2)$ *is* $PSPACE$*-complete.*
(5) The problem $STABLE^{\forall\forall}(BRAF + IC_2)$ *is* $SPACE(2^{poly})$*-complete.*
(6) The problem $STABLE^{\forall\forall}(USG + IC_2)$ *is* $PSPACE$*-complete.*
(7) The problem $STABLE^{\forall\forall}(USF + IC_2)$ *is* $SPACE(2^{poly})$*-complete.*
(8) The problem $STABLE^{\forall\forall}(PROD + IC_0)$ *is undecidable.*

Sketch of the proof. (1) Let $\mathcal{P} \in PROG^+$, $\Phi \in IC_0$ and $\delta = (D^+, D^-)$. Then it follows from monotonicity of $\mathcal{P}$ and Φ that $(< \mathcal{P} \cup \{ :- g\}, \Phi >, \delta, \mathcal{E}) \in STABLE^{\forall\forall}(PROG^+ + IC_0)$ iff there is at least one production of $\mathcal{P}$ which applies to $\mathcal{E}$, and for all DB states $\mathcal{E}^*$ such that $\mathcal{E} \vdash_{\mathcal{P}} \mathcal{E}^*$ DB state $\mathcal{E}_1 = \mathcal{E}^* \backslash D^-$ satisfies the IC Φ. Since model checking $\mathcal{E}_1 \models \Phi$ can be done in polynomial time we get the polynomial time algorithm to solve the problem (1).

Due to the following reduction, in the proofs of all the other points of theorem 1 we can take advantage of the methods used in [4] for the problem of $\exists\exists$-stability and the following problem called in [4] a promise problem :
$PROMISE(\mathbf{P} + \mathbf{I}) = \{(< \mathcal{P}, \Phi >, \delta, \mathcal{E}) \mid \mathcal{P} \in \mathbf{P}, \Phi \in \mathbf{I},$ and there is a finite action-disturbance δ-trajectory starting in $\mathcal{E}$ and resulting in some DB state $\mathcal{E}'$ satisfying the IC $\Phi\}$.

Lemma 1 *Let* $\mathbf{P}$ *be any class of programs in the assertions (2)-(8) of theorem 1, and* $\mathbf{I} \in \{IC_1, IC_2\}$. *Then*

$$STABLE^{\forall\forall}(\mathbf{P} + \mathbf{I}) \equiv_{poly} \neg PROMISE(\mathbf{P} + \mathbf{I})$$

□

The problem of $\forall\exists-$stability turns to be more complex than that of $\forall\forall-$stability.

Theorem 2

(1) The problem $STABLE^{\forall\exists}(PROG^+ + IC_0)$ *is solved in polynomial time.*

(2) The problem $STABLE^{\forall\exists}(PROG^+ + IC_1)$ *is* $PSPACE-$*complete.*

(3) The problem $STABLE^{\forall\exists}(BRAG^x + IC_2)$ *is* $PSPACE$*-complete.*

(4) The problem $STABLE^{\forall\exists}(PROG + IC_1)$ *is* $APSPACE-$ *(i.e.* $DEXPTIME-$*) complete.*

(5) The problem $STABLE^{\forall\exists}(PROF + IC_2)$ *is* $ASPACE(2^{poly})$*-complete.*

(6) The problem $STABLE^{\forall\exists}(USG + IC_1)$ *is* $APSPACE-$ *(i.e.* $DEXPTIME-$*) complete.*

(7) The problem $STABLE^{\forall\exists}(USF + IC_2)$ *is* $ASPACE(2^{poly})$*-complete.*

(8) The problem $STABLE^{\forall\exists}(PROD + IC_0)$ *is undecidable.*

Sketch of the proof. (1) To get a polynomial time algorithm in this case it is enough to notice that $(< \mathcal{P} \cup \{: -g\}, \Phi >, \delta, \mathcal{E}) \in STABLE^{\forall\exists}(PROG^+ + IC_0)$ iff there is at least one production of $\mathcal{P}$ which applies to $\mathcal{E}$ and for all DB states $\mathcal{E}^*$ such that $\mathcal{E} \vdash_{\mathcal{P}} \mathcal{E}^*$ DB state $\mathcal{E}_1 = \mathcal{E}^* \cup D^+$ satisfies the IC Φ.

The lower bounds (2)-(7) in this theorem are proven by simulation of successful computations of alternating Turing machines in terms of $\forall\exists-$stable trajectories of DDBs in the classes above. We sketch here the proof of only one lower bound, that of the point (2). To prove this lower bound we construct for any alternating Turing machine $\mathcal{M}$ a DDB $\mathcal{B}_{\mathcal{M}} = < \mathcal{P} \cup \{ :- g\}, \Phi >$ and some δ, such that $\mathcal{M}$ accepts an input word $x = a_{i_1}, ..., a_{i_n}$ in time bounded by a polynomial p, iff the DDB $\mathcal{B}_{\mathcal{M}}$ is $\delta - \forall\exists-$stable in a DB state $\mathcal{E}_x$ (constructed from x. Productions of $\mathcal{P}$ simulate $\forall$-instructions of $\mathcal{M}$ as follows: the instruction $q_l^{\forall} a_k \rightarrow q_r^{\exists} a_p S$ is simulated by the set of productions
$q(i, l, t)$ & $a(i, k, t) \implies$

$$insert(s(t+1)),\ insert(q(i+S,r,t+1)),\ insert(a(i,p,t+1))$$
($N = p(n)$, $1 \le i \le N,\ 1 \le t \le N-1$).
Here $q(i,j,t)$ means that "at the step t the head of $\mathcal{M}$ visits the cell i in state q_j", $a(i,k,t)$ means that "at the step t the cell i contains the symbol a_k", and $s(t)$ remembers the current step number t. For the accepting final state q_a of $\mathcal{M}$ $\mathcal{P}$ includes the final productions:
$q(i,a,t) \implies$ **none** $(1 \le i,t \le N)$.
δ-disturbances simulate $\exists$-steps of $\mathcal{M}$ as follows: $\delta = (\mathcal{D}^+, \mathcal{D}^-)$, and
$\mathcal{D}^+ = \{q(i,l,2t), a(i,k,t), s(2t)\ |q_l \in Q^\forall,\ 1 \le i \le N,\ 1 \le t \le N/2\}$,
$\mathcal{D}^- = \{q(i,l,2t+1), a(i,k,2t+1)\ |q_l \in Q^\exists,\ 1 \le i \le N,\ 0 \le t \le N/2\}$.

DB states represent prefixes of computations of $\mathcal{M}$, and the initial DB state $\mathcal{E}_x = \{s(0), q(1,0,0), a(1,i_1,0), \ldots, a(n,i_n,0), a(n+1,0,0), \ldots, a(N,0,0)\}$ represents the starting instantaneous description.
The following IC Φ filters out DB states representing feasible computations:
$$\Phi = A\ \&\ B\ \&\ C\ \&\ D\ \&\ E\ \&\ F\ \&\ G,$$
where: A expresses that a sequence of steps of $\mathcal{M}$ represented by DB state is continuous and finishes at an even ($\forall$-) step. B expresses that each cell contains a single symbol and the head visits a unique cell. C expresses that if $s(t)$ lacks in the DB-state then the step t is not represented by this state. D expresses that the symbols in the cells not visited by the head are not changed. E expresses that the states of $\mathcal{M}$ and the symbols in visited cells are changed by the corresponding instructions of $\mathcal{M}$. F expresses that $\mathcal{E}_x$ is included in the current DB state. G expresses that the computation represented by the DB state does not reject x.

Therefore, a DB state $\mathcal{E}$ satisfies the IC Φ iff it represents an initial segment of a computation of $\mathcal{M}$ starting in $\mathcal{E}_x$ and finishing in some $\forall$-state. Moreover, any stable trajectory of $\mathcal{B}_\mathcal{M}$ has to reach a DB state which represents a successful computation of $\mathcal{M}$ on x. Then it continues by a final production of $\mathcal{P}$ and by the empty disturbance. Now the theorem follows from the assertion that the $\forall\exists$-subtrees of stable trajectories in $T_{ad}(\mathcal{E}_x)$ one-to-one correspond to the $\forall\exists$-subtrees of successful computations of $\mathcal{M}$ on x. □

5 Conclusion

The concept of $\forall\exists$-stable behavior of DDBs proposed in this paper substantially generalizes the following property: "for any action of the DDB violating the IC there exists a bounded external update which restores the IC". This concept applies naturally to the analysis of the behavior of interactive discrete dynamic systems with complex states in active medium, whose operation involves consumption, compensation, and restoration of resources, e.g. plants, trade enterprises, warehouses, etc. Among various possible types of steady behavior of such systems, we consider here only two kinds of stable behavior: $\forall\forall$- and $\forall\exists$-stability. Meanwhile, in some other application domains we encounter other types of steady behavior as well, e.g. $\exists\exists$-stability or homeostaticity of various types, which have been formalized and explored in our previous papers [4, 5].

We consider rather broad classes of DDBs. Therefore, not surprisingly, the complexity of the properties of $\forall\forall$- and $\forall\exists$-stability in these classes sometimes turns out to be very high. However, in real applications the intensional description of system behavior is composed from a variety of independent small modules. For such DDBs our results guarantee quite tractable upper complexity bounds. Very efficient CAD systems can be created on the ground of the proposed concepts, which can support an interactive experimental analysis of behavior of complex discrete dynamic systems in broad classes of applications.

References

1. 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.
2. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J., *Alternation.* J. Ass. Comput. Mach., v.28, n.1, 114-133, 1981.
3. Dayal, U., Hanson,E., and Widom, J., *Active database systems.* In W. Kim (ed.) *Modern Database Systems.* 436-456, Addison Wesley, 1995.
4. Dekhtyar, M.I., Dikovsky, A.Ja., *Dynamic Deductive Data Bases with Steady Behavior.* In L. Sterling (ed.) *Proc. of the 12th International Conf. on Logic Programming,* The MIT Press, 183-197, 1995.
5. Dekhtyar, M.I., Dikovsky, A.Ja., *On Homeostatic Behavior of Dynamic Deductive Data Bases.* In: D. Bjorner, M.Broy, I.Pottosin (eds.) *Proc. 2nd Int. A.P.Ershov Memorial Conference "Perspectives of System Informatics". Lecture Notes in Computer Science.* 1996, Vol. 1181, 420-432.
6. Eiter, T., Gottlob, G., *On the complexity of propositional knowledge base revision, updates, and counterfactuals.* Artificial Intelligence, vol. 57, 227-270, 1992.
7. Gelfond, M., Lifschitz, V., *The stable semantics for logic programs.* In: R.Kovalsky and K.Bowen (eds.) *Proc. of the 5th Intern. Symp. on Logic Programming.* 1070-1080, Cambridge, MA, 1988, MIT Press.
8. Gottlob, G., Moercotte, G., Subrahmanian, V.S., *The PARK semantics for Active Databases.* In *Proc. of EDBT'96.* Avignon, France, 1996.
9. Halfeld Ferrari Alves, M., Laurent, D., Spyratos, N. *Update rules in Datalog programs.* Rapport de Recherche n. 1024, 01 / 1996, Université de Paris-Sud, Centre d'Orsay, LRI.
10. Katsuno, H., Mendelzon, A. O., *Propositional knowledge base revision and minimal change.* Artificial Intelligence, vol. 52, 253-294, 1991.
11. Marek, V.W., Truszciński, M. *Revision programming, database updates and integrity constraints.* In: *International Conference on Data Base theory,* ICDT, LNCS n. 893, 368-382, 1995.
12. Przymusinski, T.C., Turner, H., *Update by Means of Inference Rules.* In: V.W.Marek, A.Nerode, M.Truszczyński (eds.) *Logic Programming and Nonmonotonic Reasoning,* Proc. of the Third Int. Conf. LPNMR'95, Lexington, KY, USA, 166-174, 1995.