УДК 681.3.06

М.И.Дехтярь, А.Я.Диковский

ДИНАМИЧЕСКИЕ ДЕДУКТИВНЫЕ БАЗЫ ДАННЫХ [1]

Аннотация

Понятие дедуктивной базы данных расширяется с тем, чтобы предложения ее интенсиональной части могли динамически изменять состояние экстенсиональной части. Предложено понятие стабильного поведения дедуктивной базы данных, ориентированное на моделирование стабильных динамических процессов. Исследована вычислительная сложность проблемы стабильности и связанных с нею проблем для различных классов дедуктивных баз данных.

Введение. В этой работе мы ставим перед собой цель развить существующее понятие дедуктивной базы данных с тем, чтобы обеспечить возможность моделирования динамических систем логическими средствами. При этом центральное место в работе занимает исследование свойства стабильности динамической базы.

Представим себе динамическую систему $\mathcal{D}$, которая может находиться в состояниях из некоторого (возможно бесконечного) множества $\mathcal{S}$. Функционирование системы определяется оператором $\vdash$, представляющим собой бинарное отношение на $\mathcal{S}$: $\vdash \subset \mathcal{S} \times \mathcal{S}$. Имеется некоторый критерий допустимости состояний, т.е. некоторое их свойство $\mathcal{A} \subset \mathcal{S}$. Система $\mathcal{D}$ функционирует в среде, способной воздействовать на ее состояния с помощью другого оператора $\longrightarrow$ на $\mathcal{S}$: $\longrightarrow \subset \mathcal{S} \times \mathcal{S}$. Тогда общее поведение системы в среде описывается ее траекторией, т.е. последовательностью вида

$$\omega : \mathcal{E}_0 \longrightarrow \mathcal{E}_1^* \vdash \mathcal{E}_1 \longrightarrow \mathcal{E}_2^* \vdash ...,$$

в которой $\mathcal{E}_i$, $\mathcal{E}_i^* \in \mathcal{S}$ - состояния. Что означает, что система $\mathcal{D}$ стабильна? Что значит, что она гомеостатична? Содержательно свойство стабильности $\mathcal{D}$ состоит в следующем. Переход $\vdash$ возможен только из допустимого состояния, но в результате этого перехода свойство допустимости состояния может быть утрачено (например, переход осуществляется за счет потребления "ресурсов", не воспроизводимых системой $\mathcal{D}$.) Поэтому, чтобы обеспечить дальнейшее функционирование $\mathcal{D}$, требуется воздействие внешней среды $\longrightarrow$ (например, восполнение "ресурсов"), восстанавливающее свойство допустимости состояния. После этого снова становится возможен переход $\vdash$, и т.д.. Понятно, что разумное определение понятия стабильности предполагает некоторую оценку степени воздействия среды на состояния. Поэтому естественно предположить, что имеется некоторая мера различия между состояниями $\| \mathcal{E}_1, \mathcal{E}_2 \|$. Это позволяет уточнить понятие стабильности так: система $\mathcal{D}$ *стабильна в состоянии* $\mathcal{E}_0$ *с уровнем* δ, если имеется бесконечная траектория ω указанного выше вида, в которой все состояния $\mathcal{E}_i^*$ допустимы и для любого $j \geq 0$ $\| \mathcal{E}_j, \mathcal{E}_{j+1}^* \| \leq \delta$.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (93-012-627).

Свойство гомеостатичности двойственно свойству стабильности. Содержательный его смысл состоит в том, что из любого состояния, получаемого из допустимого состояния внешним воздействием определенного уровня (возмущением из внешней среды), система может вновь достичь допустимое состояние. Говоря несколько более точно, система $\mathcal{D}$ *гомеостатична с уровнем* δ, если для любых состояний $\mathcal{E}_0$, $\mathcal{E}_1^*$ таких, что $\mathcal{E}_0$ допустимо, $\mathcal{E}_0 \longrightarrow \mathcal{E}_1^*$ и $\| \mathcal{E}_0, \mathcal{E}_1^* \| \leq \delta$, существует допустимое состояние $\mathcal{E}_1$ такое, что $\mathcal{E}_1^* \vdash \mathcal{E}_1$. Представляет интерес и более слабый аналог свойства гомеостатичности. Система $\mathcal{D}$ *слабо гомеостатична в состоянии* $\mathcal{E}_0$ *с уровнем* δ, если существует бесконечная ее траектория ω, указанного выше вида, в которой все состояния $\mathcal{E}_i$ допустимы и для любого $j \geq 0$ $\| \mathcal{E}_j, \mathcal{E}_{j+1}^* \| \leq \delta$.

Для описания поведения динамических систем традиционно используется аналитический аппарат, который, как известно, весьма чувствителен к возрастанию числа параметров задач. Между тем, многие динамические системы могут быть описаны в виде базы данных, фиксирующей текущее состояние, и теории (логической схемы базы данных), описывающей допустимые состояния и законы их изменения. Для подобных баз и теорий число параметров задачи - весьма второстепенный фактор. Поэтому представляется весьма перспективным подход к моделированию динамических систем с помощью дедуктивных баз данных. При этом, конечно, дедуктивные базы данных должны допускать внелогические средства изменения состояний.

Понятие дедуктивной базы данных [1], [2] принадлежит парадигме "логического программирования". В логической программе, интерпретируемой как дедуктивная база данных, выделяют две части: экстенсиональную и интенсиональную. Экстенсиональная часть $\mathcal{E}$ включает предикаты, определяемые исключительно фактами, т.е. атомарными логическими формулами вида $p(t_1, ..., t_n)$, в которых p - имя $n-$ местного предиката (отношения) и t_i - термы. Для дедуктивных баз типично ограничение, допускающее в качестве термов лишь константы или переменные (плоские термы). Интенсиональная часть $\mathcal{I}$ включает предикаты, в определениях которых используются Хорновские предложения, т.е. формулы импликативного фрагмента логики первой ступени вида

$$A_1 \ \& \ \ldots \& \ A_k \ \rightarrow \ A,$$

в которых посылки и заключения содержат атомарные формулы A_i, A. Таким образом, экстенсиональная часть описывает текущее состояние некоторой предметной области (базы данных), а интенсиональная часть в аксиоматическом виде описывает "знания"об этой области. Интерпретатор $\Longrightarrow$ дедуктивной базы данных реализует оператор, определяемый интенсиональной частью и извлекающий всевозможные следствия (ответы на запросы) из состояния базы данных. Точнее, для формулы-запроса G он строит опровержение

$$\mathcal{I} \ \cup \ \mathcal{E} \ \cup \ \{ \ \neg \ G \circ \theta \ \} \ \Longrightarrow \ \square$$

для подходящего частного случая G (θ - подстановка значений вместо переменных G). Конструируемая интерпретатором подстановка θ является "ответом на запрос" G . При этом интерпретатор корректен в том смысле, что $\forall(G \circ \theta)$ логически следует из $\mathcal{I} \cup \mathcal{E}$. Таким образом, дедуктивные базы данных ставят в более широкие логические рамки обычные базы данных.

Общепринято считать, что интерпретатор $\Longrightarrow$ не производит побочных эффектов, т.е. не изменяет состояние базы $\mathcal{E}$, и в этом смысле может быть назван *статическим*. Между тем, как уже было отмечено выше, для задач моделирования поведения динамических систем необходимо, чтобы интерпретатор дедуктивных баз данных имел возможность изменять состояния базы, т.е. добавлять, удалять или заменять факты состояний $\mathcal{E}$. Заметим, что при этом должны использоватся операторы, которые реально, а не условно (как, например, в [3]) изменяют состояние базы. Такого рода средства динамического изменения логической программы типичны для стандартных систем логического программирования [4], [5]. Включая аналогичные средства в предложения интенсиональной части базы (у нас это - операторы add/1, delete/1 и
change/2), мы превращаем интерпретатор $\Longrightarrow$ в, вообще говоря, немонотонный оператор на состояниях базы. При этом интерпретация запроса G соответствует выполнению некоторой функции моделируемой системы.

В первом разделе работы приведен пример дедуктивной динамической базы данных, иллюстрирующий указанные особенности таких систем.

Во втором разделе собраны все необходимые исходные понятия. В частности, здесь уточняется операционная семантика логической программы с динамическими операторами. Основное своеобразие нашего варианта семантики определяется выбранной стратегией вычисления: подцели в теле выбранного предложения устраняются в естественном порядке, т.е. как в Прологе. Как известно [6], [7], статические логические программы полны относительно денотационной семантики [8] при любой стратегии. Поэтому для статических программ наш выбор ничем не оправдан. Однако для логических программ с динамическими операторами порядок устранения динамических подцелей определяет порядок изменения состояния базы данных и доступа к ней. Поэтому некоторая определенная стратегия обязательно должна быть выбрана, а выбранная нами является наиболее естественной.

В третьем разделе приводится новое определение дедуктивной базы данных, расширенное на случай динамических операторов. Здесь определяется свойство перспективности состояния базы данных, состоящее в том, что из него достижимо без внешних коррекций допустимое состояние. В общем случае это свойство, конечно, алгоритмически неразрешимо. Мы классифицируем наиболее интересные разрешимые случаи и оцениваем их вычислительную сложность. Оказывается, что в случае, когда интенсиональные предложения представляют собой базисные подукции без отрицаний и база данных может лишь монотонно увеличиваться, эта проблема разрешима за линейное время. Заметим, что это линейное время в сильном смысле, т.е. шаг имеет константный вес по отношению к переходу по ссылке. Фактически здесь используется модификация алгоритма из [9]. Стоит допустить отрицания, и проблема становится NP-полной. Если же в продукциях разрешить удаление фактов из базы данных, то проблема PSPACE-полна. Более того, если в продукциях разрешить использование сложных небазисных термов, то проблема становится алгоритмически неразрешимой. Другой используемый нами критерий классификации - это расслоенность логических программ в смысле [10]. Если интенсиональная часть дедуктивной базы данных базисна и расслоена относительно динамических операторов, то в ней проблема перспективности состояний относится к PSPACE. Если ослабить свойство базисности, допуская плоские термы, то проблема становится

$SPACE(2^{poly})$-полной.

Последний раздел посвящен свойству стабильности. Проблема стабильности оказывается PSPACE-полной уже в классе дедуктивных баз, с базисными интенсиональными частями, расслоенными относительно динамических операторов Если снять условие базисности, то проблема становится $SPACE(2^{poly})$-полной.

Среди работ, подводящих к этой проблематике и оказавших на нее влияние, следует определенно назвать [11].

1. Пример динамической дедуктивной базы данных. Рассмотрим пример, относящийся к проблеме финансового и материального обеспечения строительного предприятия. Строительное предприятие "Коттедж"строит индивидуальные дома. Его финансирование происходит за счет платежей заказчиков по контрактам, а расходы связаны с закупкой необходимых материалов и выплатой зарплаты сотрудникам. Подразумеваемая семантика используемых в примере предикатов соответствует их названиям. Следуя традиционному синтаксису логических программ, мы записываем импликацию
A_1 & ...& A_k $\rightarrow$ A в виде A :- A_1, ..., A_k .

1. Фрагмент схемы состояний базы:

```
  тек_счет(Сумма),
  своб_заказ(Заказчик),
  состояние_заказа(Номер,Заказчик,Готовность),
    /* признак Готовность принимает значения из
       {no,yes} */
  склад(Вид_материала,Количество_в_наличии),
  потребность(Вид_материала,Количество),
  оплата(Номер_заказа,Сумма_к_оплате,Оплачено),
  получено(Номер_заказа,Переведенная_сумма),
  зарплата(Год,Месяц,Сумма)
    /* общая начисленная зарплата за месяц */,
  выплачено_за(Год,Месяц),
  цена(Материал,Цена),
  стоим_работ(Стоимость),
  рентабельность(Процент),
  комплект(Комплект),
    /* комплект - список вида [M1:K1,...,Mr:Kr], Mi
       - деталь, Ki - количество экземпляров; можно
       обойтись и без списка, увеличив число
       аргументов */
  мин_сум(Сумма),
    /* сумма - минимально допустимый остаток средств
       на счете */
  мин_зарплата(Сумма),
  численность(Число_сотрудников),
  ...
\medskip
```

2. Фрагмент интенсиональной части базы

```
/*** Выполнение этапа ***/

выполнить :-
     зарплата_выплачена,
     состояние_заказа(N,C,no),
     оценить(Цена),
     change(состояние_заказа(N,C,no),
            состояние_заказа(N,C,yes)),
     add(оплата(N,Цена,0)).

/*** Получение оплаты ***/

получить :-
     получено(N,Sum),
     (оплата(N,S,C),
      change(оплата(N,S,C),
             оплата(N,S,C+Sum))
      ;
      not оплата(N,_,_),
      add(оплата(N,0,Sum)), /* авансирование */
      тек_счет(T),
      change(тек_счет(T),
             тек_счет(T+Sum))
     ),
     delete(получено(N,Sum)).

/* Выплата заработной платы */

выплатить :-
     предыдущий_месяц(Y,M),
 /* встроенный предикат */
     not оплачено_за(Y,M),
     (зарплата(Y,M,T)
     ;
      not зарплата(Y,M,_),
      мин_зарплата(S),
      численность(N),
      T = S*N
     ),
     тек_счет(C),
     C > T,
     change(тек_счет(C),тек_счет(C-T)),
     add(оплачено_за(Y,M)).

 зарплата_выплачена :-
     предыдущий_месяц(Y,M),
     оплачено_за(Y,M).
```

```
/*** Поиск заказа ***/

поиск :-
     зарплата_выплачена,
     not состояние_заказа(_,_,no),
 /* нечего делать */
     своб_заказ(Заказчик),
     присвоить_номер(Номер),
     add(состояние_заказа(Номер,Заказчик,no)),
     delete(своб_заказ(Заказчик)).

/*** Выполнение и оценка этапа ***/

оценить(Цена) :-
     комплект(Комплект),
     для_вып(Комплект,0,Сумма,возм,Рез),
     ifthenelse(Рез=возм,
            (forall(member(M:K,Комплект),
                    change(склад(M,Q),склад(M,Q-K))),
             стоим_работ(W),
             Стоимость = W+Сумма,
             рентабельность(Процент),
             Цена = Стоимость*(1 + Процент/100),
             change(тек_счет(T),тек_счет(T-W)))).

/*** Оценка наличия материалов на складе ***/

для_вып([],Цена,Цена,Рез,Рез).
для_вып([M:K|R],P,Цена,_,Рез) :-
     склад(M,Q),
     Q < K,
     приобрести(M,K-Q),
     для_вып(R,_,_,невозм,Рез).
для_вып([M:K|R],P,Цена,Рз,Рез) :-
     склад(M,Q),
     Q >= K,
     цена(M,Price),
     для_вып(R,P+K*Price,Цена,Рз,Рез).

/*** Приобретение материалов ***/

приобрести(M,Q) :-
     цена(M,Price),
     тек_счет(T),
     T > Q*Price,
     change(тек_счет(T),
            тек_счет(T-Q*Price)),
     склад(M,K),
     change(склад(M,K),
```

```
            склад(M,K+Q)).
```

3. Условие допустимости состояния (только по финансам)

```
допустимо :-
     тек_счет(T),
     мин_сум(Мин),
     (T > Мин
      ;
      T є Мин,
      findall(P,
                (оплата(_,S,C),
                 S > C,
                 P is S-C),
               LP),
      sum(LP,Своб),
 /* Своб - дебеторская задолженность */
      T + Своб > Мин).
```

Как видим, в этом примере выполнение любой операции (т.е. выполнение любого запроса) приводит к изменению состояния базы. Здесь, так же как и во многих других подобных системах, основной проблемой является проблема поиска такой последовательности операций, при которой, начиная с допустимого состояния, можно всякий раз снова достигать некоторое допустимое состояние, и таким образом обеспечивать бесконечное функционирование системы. Иначе говоря, в этом примере основной является проблема стабильного поведения базы. Приведенный пример как раз и демонстрирует ситуацию, когда выполнение операций связано с потреблением "ресурсов", которые не восполняются базой и, таким образом, должны восполняться в результате внешнего воздействия среды. Этими ресурсами являются свободные заказы. Выполнение заказа связано с потреблением материалов и выплатой заработной платы, а средства на то и другое перечисляются заказчиками при наличии заказов. При фиксированных ценах и данном уровне рентабельности число заказов, необходимых для возобновления производства, как раз и определяет уровень стабильности системы. Факты, описывающие искомые свободные заказы, потребляются в результате выполнения операции *поиск/0*, когда возникает простой. При этом добавляться в базу данных они могут только внешним по отношению к системе образом.

2. Обозначения и определения.

Зафиксируем сигнатуру $\mathbf{\Sigma}$, состоящую из трех непересекающихся алфавитов $< \mathcal{F}, \mathcal{P}^e, \mathcal{P}^i >$. Будем считать, что алфавиты $\mathbf{\Sigma}$ содержат выражения вида a/n. n называется *арностью* символа a. Символы из $\mathcal{F}$ называются *функциональными* (0-арные символы называются *константами*). Символы из $\mathcal{P}^e$ являются именами *экстенсиональных предикатов,* а - из $\mathcal{P}^i$ - именами *интенсиональных предикатов*. 0-арные предикатные символы называются *пропозициональными*. Обозначим через $\mathbf{Var}$ счетное множество переменных и через $\mathbf{T}_{\mathcal{F}}$ - множество всех термов в алфавите $\mathcal{F} \cup \mathbf{Var}$. Далее обозначим через $\mathbf{H}_{\mathcal{F}}$ множество *базисных* термов над $\mathcal{F}$, т.е. термов, не содержащих символов переменных. Через $\mathbf{A}^e$ ($\mathbf{A}^i$) обозначим множество

всех атомов вида $p(t_1, ..., t_n)$, где $p/n \in \mathcal{P}^e$ ($\mathcal{P}^i$) и $t_1, ..., t_n \in \mathbf{T}_\mathcal{F}$, а через $\mathbf{B}^e$ ($\mathbf{B}^i$) - множество всех базисных атомов того же вида, в которых $t_1, ..., t_n \in \mathbf{H}_\mathcal{F}$.

В приложениях часто конкретные значения некоторых аргументов (атрибутов, параметров) термов и атомов играют служебную роль, поскольку существенные свойства баз данных и их преобразований не зависят от этих значений. Термы (атомы), отличающиеся лишь такими "несущественными"аргументами, естественно считать информационно эквивалентными. В приведенном выше примере можно считать несущественными конкретные значения параметра "номер_заказа". Разбиение всех термов и атомов на классы информационно эквивалентных часто позволяет сводить бесконечную или очень большую конечную предметную область к конечной или вполне обозримой. Пусть $\equiv$ - некоторое отношение эквивалентности на $\mathbf{H}_\mathcal{F}$. Его можно естественным образом распространить и на $\mathbf{B}^e$: $g(t_1, ..., t_n) \equiv g(t'_1, ..., t'_n) \rightleftharpoons t_i \equiv t'_i$ для всех $1 \leq i \leq n$. Для любых D_1, $D_2 \subset \mathbf{B}^e$ $D_1 \Leftarrow D_2$, если существует такая инъекция $\eta : D_1 \rightarrow D_2$, что для каждого $G \in D_1$ $G \equiv \eta(G)$. $D_1 \Leftrightarrow D_2$, если $D_1 \Leftarrow D_2$ и $D_2 \Leftarrow D_1$.

Мы будем рассматривать логические программы в сигнатуре $\mathbf{\Sigma} =< \mathcal{F}, \mathcal{P}^e, \mathcal{P}^i >$, состоящие из расширенных Хорновских предложений вида

ГОЛОВА

или

ГОЛОВА :– *ТЕЛО.*

ГОЛОВА - это некоторый интенсиональный атом из $\mathbf{A}^i$. ТЕЛО строится из *литералов*, т.е. атомов из $\mathbf{A}^i \cup \mathbf{A}^e$ или отрицаний атомов из $\mathbf{A}^e$, и динамических подцелей вида $add(atom)$ – добавить атом, $delete(atom)$ – удалить атом, $change(atom_1, atom_2)$ – заменить $atom_1$ на $atom_2$, в которых $atom$, $atom_1$ и $atom_2 \in \mathbf{A}^e$, с помощью операторов "," (конъюнкции) и ";" (дизъюнкции). Таким образом, в рассматриваемом случае все атомы в негативных литералах и динамических подцелях являются экстенсиональными. Предложение, у которого все атомы в голове и в теле являются базисными, называется *базисным.*

Определение 1. Логическая программа *в* $\mathbf{\Sigma}$ - *это конечное множество* $\mathcal{P} = \mathcal{I} \cup \mathcal{E} \cup \{ :- \mathcal{R} \}$, *где*

- $\mathcal{I}$ - интенсиональная компонента *программы* $\mathcal{P}$ - *непустое множество расширенных Хорновских предложений;*
- $\mathcal{E}$ - *подмножество* $\mathbf{B}^e$, *называемое* экстенсиональной компонентой или состоянием базы данных $\mathcal{P}$;
- $\mathcal{R}$ - *конъюнкция литералов.* $\{ :- \mathcal{R} \}$ *называется* целью.

Для определения операционной семантики логической программы $\mathcal{P}$ обобщим процедуру опровержения из [7] следующим образом. На каждом шаге вычисления его состояние определяется тройкой ($\mathcal{E}_c$, $\mathcal{R}_c$, σ_c), где $\rfloor\!\dashv\!\updownarrow\mathcal{E}_c$ - текущее состояние базы данных, $\mathcal{R}_c$ - текущая цель - список недоказанных подцелей и σ_c - подстановка. В начале вычисления $\mathcal{E}_c = \mathcal{E}$, $\mathcal{R}_c = \mathcal{R}$ и $\sigma_c = \varepsilon$ (ε - тождественная подстановка). На очередном

шаге вычисления в цели $\mathcal{R}_c = G_1, G_2, \ldots, G_k$ выбирается первая (слева) подцель G_1 и в зависимости от ее вида происходит преобразование текущего состояния вычисления. Если первая подцель G_1 является дизъюнктивной, т.е. $G_1 = (L_1, \ldots, L_l ; R_1, \ldots, R_r)$, то новой целью будет либо $L_1, \ldots, L_l, G_2, \ldots, G_k$, либо $R_1, \ldots, R_r, G_2, \ldots, G_k$. Если $G_1 \in \mathbf{A}^i \cup \mathbf{A}^e$ и в $\mathcal{I} \cup \mathcal{E}_c$ имеется предложение (факт), голова которого унифицируется с G_1, то выбирается бесконфликтный вариант (т.е. результат взаимно-однозначной замены переменных на ранее не встречавшиеся переменные) одного из них $H :- B_1, \ldots, B_r$, определяется наиболее общий унификатор θ такой, что $G_1 \circ \sigma_c \circ \theta = H \circ \theta$, цель $\mathcal{R}_c$ заменяется на $B_1, \ldots, B_r, G_2, \ldots, G_k$ и подстановка σ_c заменяется на $\sigma_c \circ \theta$. Если $G_1 = \neg\, g(\bar{t})$ для $g(\bar{t}) \in \mathbf{A}^e$, то G_1 успешно удаляется из $\mathcal{R}_c$, если $g(\bar{t})$ не унифицируется ни с одним фактом из $\mathcal{E}_c$. Пусть $A, A_1, A_2 \in \mathbf{B}^e$. Тогда вызов $G_1 = add(A)$ успешно удаляется из $\mathcal{R}_c$, а побочный эффект этого вызова заключается в том, что факт A добавляется к $\mathcal{E}_c$, если его там не было . Если $G_1 = delete(A)$ и A есть в $\mathcal{E}_c$, то G_1 удаляется из $\mathcal{R}_c$ и A удаляется из $\mathcal{E}_c$. Если A_1 есть в $\mathcal{E}_c$, то $G_1 = change(A_1, A_2)$ удаляется из $\mathcal{R}_c$ и при этом факт A_1 удаляется из $\mathcal{E}_c$, а факт A_2 добавляется к $\mathcal{E}_c$, если он там ранее отсутствовал. Если в текущем состоянии вычисления цель $\mathcal{R}_c$ пуста, то будем считать, что опровержение начальной цели $\mathcal{R}$ завершено успешно и обозначать это обычным образом:

$$\mathcal{I} \cup \mathcal{E} \cup \{ :- \mathcal{R} \} \Longrightarrow \Box.$$

Если для состояния $\mathcal{E}' \subset \mathbf{B}^e$ существует успешное вычисление в заключительном состоянии которого $\mathcal{E}_c = \mathcal{E}'$, то будем говорить, что программа $\mathcal{I}$ с целью $\{ :- \mathcal{R} \}$ *переводит* состояние $\mathcal{E}$ в состояние $\mathcal{E}'$ и писать $\mathcal{E} \overset{\mathcal{R}}{\vdash_{\mathcal{I}}} \mathcal{E}'$.

Определение 2. *Логическая программа* $\mathcal{I} \cup \{ :- \mathcal{R} \}$ согласована *с отношением эквивалентности* $\equiv$ *на* $\mathbf{B}^e$, *если для любых трех подмножеств* $\mathcal{E}_1, \mathcal{E}_1', \mathcal{E}_2 \subset \mathbf{B}^e$ *таких, что* $\mathcal{E}_1 \Leftrightarrow \mathcal{E}_2$ *и* $\mathcal{E}_1 \overset{\mathcal{R}}{\vdash_{\mathcal{I}}} \mathcal{E}_1'$ *существует* $\mathcal{E}_2'$ *такое, что* $\mathcal{E}_1' \Leftrightarrow \mathcal{E}_2'$ *и* $\mathcal{E}_2 \overset{\mathcal{R}}{\vdash_{\mathcal{I}}} \mathcal{E}_2'$.

Так как класс логических программ является универсальным, т.е. обладает вычислительной силой машин Тьюринга, то большинство интересных проблем, связанных с преобразованиями баз данных логическими программами, алгоритмически неразрешимы. Поэтому мы будем рассматривать некоторые ограниченные подклассы логических программ, включающие большинство встречающихся на практике случаев.

Определение 3. *Пусть* $\mathcal{P}$ *- логическая программа. Следуя [10], скажем, что ее предикат* p ссылается на *предикат* q, *если в* $\mathcal{P}$ *есть предложение, определяющее* p, *в котором вызывается* q. *Пусть отношение* "зависит от" *является рефлексивно-транзитивным замыканием отношения "ссылается на". Максимальные сильно-связные компоненты графа отношения "зависит от"называются* кликами. *Назовем логическую программу* динамически-расслоенной (д-расслоенной), *если в любом ее предложении вида*

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

в котором предикаты p и q принадлежат одной и той же клике, все предикаты $p_1, \ldots, p_i, p_{i+1}, \ldots, p_r$ *являются статическими, т.е. не зависят от динамических предикатов.*

Смысл этого определения состоит в том, что в д-расслоенных логических программах состояния изменяются только при смене клики, и поэтому число различных состояний на одной ветви дерева вывода не превосходит числа клик.

Определение 4. *Логическая программа* $\mathcal{P}$ *называется* статической, *если все ее предикаты являются статическими (иначе говоря, в ее предложениях не вызываются динамические предикаты* add/1, delete/1, change/2,*).* $\mathcal{P}$ *называется* позитивной *если в ее предложениях нет отрицаний,* базовой, *если в них нет переменных и* расширяющей, *если в них нет вызовов динамических предикатов delete/1, change/2. Назовем* $\mathcal{P}$ продукционной *если она описывает единственный пропозициональный предикат* G *и все предложения* $\mathcal{P}$ *имеют вид:* $G :- B_1, ..., B_k, Act_1, ..., Act_m$ *, где* B_i *- это вызов экстенсионального предиката или его отрицания, а* Act_j *- вызов динамического предиката. Такое предложение, по-существу, представляет продукцию вида* $Условие \Rightarrow$
Действие, где Условие является конъюнкцией формул B_i, *а Действие - последовательностью операций над базой данных* $Act_1, ..., Act_m$.

3. Дедуктивные базы данных.

Определение 5. Дедуктивной базой данных (ДБД) *в сигнатуре* $\mathbf{\Sigma} = < \mathcal{F}, \mathcal{P}^e, \mathcal{P}^i >$ *называется система*

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

где

- *для всякого конечного* $\mathcal{E}$ *из* $\mathbf{B}^e$ *(называемого* экстенсионалом *или* состоянием $\mathcal{B}$*) и каждого* $i = 1, \ldots, n$ *множество* $\mathcal{I} \cup \mathcal{E} \cup \{ :- G_i \}$ *является логической программой;*
- *каждая цель* G_i, $i = 1, \ldots, n$, *является либо пропозициональной, либо статической. Пропозициональные цели называются* модификациями, *а статические -* запросами;
- *логическая программа* $\mathcal{A} \cup \{ :- \mathcal{R} \}$ *является статической - она называется* критерием допустимости (состояний);
- $\equiv_i$ *- отношение эквивалентности на* $\mathbf{B}^e$, *называемое* информационной эквивалентностью;
- *логические программы* $\mathcal{I} \cup \{ :- G_i \}$, $i = 1, \ldots, n$ *и* $\mathcal{A} \cup \{ :- \mathcal{R} \}$ *согласованы с* $\equiv_i$.

Состояние $\mathcal{E}$ *называется* допустимым, *если* $\mathcal{A} \cup \mathcal{E} \cup \{ :- \mathcal{R} \} \Longrightarrow \Box$.

Определение 6. *(Операционная семантика ДБД). Пусть* $\mathcal{E}_1, \mathcal{E}_2 \subset \mathbf{B}^e$ *- два состояния ДБД* $\mathcal{B}$, *а* $\mathcal{D}^+, \mathcal{D}^- \subset \mathbf{B}^e$ *таковы, что* $\mathcal{E}_2 = (\mathcal{E}_1 \cup \mathcal{D}^+) \setminus \mathcal{D}^-$. *В этом случае мы будем писать* $\mathcal{E}_1 \overset{\mathcal{D}^+,\mathcal{D}^-}{\longrightarrow} \mathcal{E}_2$. *Назовем* траекторией *ДБД* $\mathcal{B}$ *конечную или бесконечную последовательность*

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

в которой каждое G_i *- либо запрос, либо непустая последовательность модификаций. Последовательность пар*

$$(\mathcal{D}_1^+, \mathcal{D}_1^-), \ (\mathcal{D}_2^+, \mathcal{D}_2^-) \ , ...$$

называется коррекцией *траектории* ω, *а последовательность целей* G_{i_1}, G_{i_2}, ... - управляющей последовательностью *траектории* ω. *Траектория* ω *называется* допустимой, *если она бесконечна и все ее состояния* $\mathcal{E}_i^*$ *являются допустимыми.*

В приведенном выше примере ДБД "Коттедж"программа $\mathcal{I}$ состоит из предложений, определяющих предикаты *выполнить/0, получить/0, выплатить/0, поиск/0, оценить/1, для вып/5, приобрести/2.* Все четыре пропозициональных предиката из этого списка являются возможными модификациями G_i. Критерий допустимости $\mathcal{A}$ состоит из одного предложения, задающего статический предикат *допустимо/0*, являющийся также и целью $\mathcal{R}$.

Отметим, что в обеих логических программах ДБД "Коттедж"помимо указанных интенсиональных, экстенсиональных и динамических предикатов использованы также и некоторые встроенные предикаты (арифметические, $sum/2$ - суммирование элементов списка целых, $findall/3$, и др.). Такого рода средства безусловно должны присутствовать в прикладных версиях дедуктивных баз данных. Однако поскольку для ДБД все основные разрешающие процедуры являются по-существу символическими, включение таких предикатов в формальное определение ДБД было бы неоправданным.

Естественными вопросами, возникающими для ДБД, являются вопросы о возможности перехода из недопустимого состояния в допустимое (с использованием или без использования коррекций) и вопросы о существовании траекторий, удовлетворяющих тем или иным условиям. Для целей сложностного анализа этих проблем мы будем предполагать, что состояния логических программ и баз данных конечны.

Определение 7. *Назовем состояние* $\mathcal{E} \subset \mathbf{B}^e$ перспективным *для ДБД* $\mathcal{B}$, *если существует конечная траектория* $\mathcal{B}$ *с пустой коррекцией (т.е. для каждого* j $\mathcal{D}_j^+ = \emptyset$ *и* $\mathcal{D}_j^- = \emptyset$ *), переводящая* $\mathcal{E}$ *в некоторое допустимое состояние.*

Пусть, например, в описанной выше ДБД "Коттедж"состояние $\mathcal{E}$ = { *тек_счет(100), мин_сум(200), получено(17, 200)*}. Нетрудно проверить, что оно не является допустимым, но после модификации *получить/0* перейдет в состояние $\mathcal{E}_1$ = { *тек_счет(300), мин_сум(200), оплата(17,0,200)*}, которое уже будет допустимо. Следовательно, $\mathcal{E}$ - перспективное состояние.

Пусть $\mathbf{P}$ - это некоторый класс ДБД. Проблема перспективности для этого класса состоит в проверке принадлежности пар вида $(\mathcal{B}, \mathcal{E})$ множеству $ПРСП(\mathbf{P}) = \{(\mathcal{B}, \mathcal{E}) \mid \mathcal{E}$ *конечно и перспективно для ДБД* $\mathcal{B} \in \mathbf{P}\}$. Для класса всех ДБД эта проблема очевидно неразрешима, так как, например, она включает в себя проблему остановки логической программы-критерия допустимости. Поэтому далее мы будем рассматривать лишь ДБД, у которых критерий допустимости $\mathcal{A} \cup \{ :- \mathcal{R} \}$ удовлетворяет одному из условий:

(Д1) программа $\mathcal{A}$ является базисной (Д1-критерий);

(Д2) все атомы с интенсиональными предикатами в телах предложений являются базисными (Д2-критерий).

Теорема 1. *(1) Проблема проверки по любой программе* $\mathcal{A}$, *удовлетворяющей условию (Д1), и по конечному состоянию* $\mathcal{E} \subset \mathbf{B}^e$ *свойства допустимости разрешима за линейное время (от* $|\mathcal{A}| + |\mathcal{E}|$*).*

(2) Проблема проверки по любой программе $\mathcal{A}$, *удовлетворяющей условию (Д2), и по конечному состоянию* $\mathcal{E} \subset \mathbf{B}^e$ *свойства допустимости является NP-полной.*

Обозначим через ПРОП класс всех ДБД с интенсиональными продукционными логическими программами $\mathcal{I}$. Выделим в нем следующие подклассы:

- ПРОП - подкласс ДБД в интенсиональных программах которых допускаются только 0-местные функциональные символы (т.е. все аргументы атомов - переменные или константы);
- ПРОБ - это подкласс ДБД с базисными программами $\mathcal{I}$;
- $ПРОБ^-$ - подкласс ПРОБ, в котором программы $\mathcal{I}$ являются расширяющими;
- $ПРОБ^+$ - подкласс ПРОБ, в котором программы $\mathcal{I}$ являются позитивными и расширяющими (т.е. в них отсутствуют отрицания и операторы удаления и изменения фактов состояния).

Все алгоритмические проблемы, которые мы рассматриваем в этой работе, разрешимы в указанных узких подклассах ПРОД . Однако не следует считать эти классы чересчур бедными и имеющими чисто технический смысл. Внутри них мы обнаруживаем логические программы, полезные для многих важных приложений. Так, класс $ПРОБ^+$ содержит программы, вычисляющие замыкания функциональных зависимостей в реляционных базах данных. Другим примером являются электронные таблицы, непосредственно представимые в виде ДБД с интенсиональными логическими программами из класса ПРОБ и простыми (Д1)-критериями допустимости. Отметим также, что все так называемые продукционные экспертные системы попадают в класс ПРОП, а некоторые из них - даже в класс $ПРОБ^-$. Оказывается, что даже в этих простейших классах логических программ сложность проблемы перспективности состояний существенно зависит от того, какие механизмы немонотонности в них допускаются. Возможность использовать в программах сложные термы сразу же приводит к неразрешимости этой проблемы.

Теорема 2. *(1) Проблема* $ПРСП(ПРОБ^+)$ *по отношению к (Д1)-критериям разрешима за линейное время.*

(2) Проблема ПРСП(ПРОБ$^-$) по отношению к (Д2)-критериям принадлежит NP и по отношению к (Д1)-критериям является NP-трудной.

(3-4) Проблемы ПРСП(ПРОБ) и ПРСП(ПРОП) по отношению к (Д2)-критериям принадлежат PSPACE и по отношению к (Д1)-критериям являются PSPACE-трудными.

(5) Проблема ПРСП(ПРОД) по отношению к (Д1)-критерию алгоритмически неразрешима.

В продукционных программах возможности интенсиональной компоненты по управлению выводом и изменениям базы данных весьма невелики. Гораздо больше средств для этого предоставляют д-расслоенные программы. Мы обозначим через *ДПР* класс ДБД с д-расслоенными программами $\mathcal{I}$, в которых допускаются только 0-местные функциональные символы, и через *ДБР* - класс ДБД с базисными д-расслоенными программами $\mathcal{I}$.

Теорема 3. *(1) Проблема* ПРСП(ДБР) *по отношению к (Д2)-критериям является PSPACE-полной.*

(2) Проблема ПРСП(ДПР) *по отношению к (Д2)-критериям является* $SPACE(2^{poly})$*-полной.*

4. Стабильность.

Наиболее важным для приложений ДБД являются свойства ее стабильности по отношению к немонотонным преобразованиям состояний, связанным с допустимыми траекториями. Стабильные ДБД позволяют моделировать поведение устойчивых динамических систем логическими средствами. Мы предлагаем следующее определение свойства стабильности.

Определение 8. *Пусть* $\mathcal{B}$ *- ДБД в сигнатуре* Σ. Покрытие - *это пара* конечных *множеств* $\delta = < \mathcal{D}_H^+, \mathcal{D}_H^- >$, *где* $\mathcal{D}_H^+$, $\mathcal{D}_H^- \subset \mathbf{B}^e$.

ДБД $\mathcal{B}$ стабильна с покрытием δ на состоянии $\mathcal{E}_0$, *если существует такая допустимая траектория* ω, *начинающаяся с* $\mathcal{E}_0$, *что для всех* $j > 0$

$$\mathcal{D}_j^+ <\equiv_i \mathcal{D}_H^+ \text{ и } \mathcal{D}_j^- <\equiv_i \mathcal{D}_H^-.$$

Пусть $\mathcal{C}^+$, $\mathcal{C}^- \subset \mathbf{B}^e$. $\mathcal{B}$ стабильна с покрытием δ на $(\mathcal{C}^+, \mathcal{C}^-)-$состояниях, *если* $\mathcal{B}$ *стабильна с покрытием* δ *на любом состоянии* $\mathcal{E}$, *таком что* $\mathcal{C}^+ \Leftarrow \mathcal{E}$ *и для каждого* $t \in \mathcal{C}^-$ *неверно, что* $t \Leftarrow \mathcal{E}$.

Например, передположим, что в примере "Коттедж"в исходном состоянии $\mathcal{E}_0$ цена комплекта, определяемая предикатом *для_вып/5*, составляет 800, а остальные параметры определяются фактами *стоимость_работ(700), рентабельность(50), тек_счет(2200), мин_зарплата(100), численность(9), мин_сум(1000).* Тогда нетрудно проверить, что эта ДБД стабильна с покрытием $\delta = \{$*своб_заказ(_), получено*$(\_, 2250)\ \}$ на состоянии $\mathcal{E}_0$. Соответствующую допустимую траекторию можно описать условным регулярным выражением:

$\{$ *выплатить* $\circ$ **своб_заказ(_)** $\circ$ *поиск* $\circ\ \emptyset \circ$ *выполнить* $\circ$
получено $\circ$ *получить* $\circ\ \emptyset\ \}^*$.

(Жирным шрифтом выделены коррекции; $\emptyset$ - пустая коррекция.) Однако δ не гарантирует стабильность ДБД для всех исходных состояний. Например, стабильность будет нарушена для состояния, в котором минимальная

зарплата возрастет до 500, а остальные параметры останутся прежними. Дело в том, что в таком состоянии невозможна ни одна модификация.

Пусть $\mathbf{P}$ - это некоторый класс ДБД. Обозначим через СТБЛ($\mathbf{P}$) множество троек вида $(\mathcal{B}, \delta, (\mathcal{C}^+, \mathcal{C}^-))$ таких, что $\mathcal{B}$ принадлежит $\mathbf{P}$ и при этом стабильна с покрытием δ на $(\mathcal{C}^+, \mathcal{C}^-)$– состояниях.

Теорема 4. *(1) Проблема СТБЛ(ДБР) по отношению к (Д2)-критериям является PSPACE-полной.*

(2) Проблема СТБЛ(ДПР) по отношению к (Д2)-критериям является $SPACE(2^{poly})$*-полной.*

Заключение.

Проблема стабильности в классе ПРОП, конечно же (и по тем же причинам, что и проблема перспективности) алгоритмически неразрешима.

Рассматриваемые в этой работе классы логических программ и ДБД в основном определяются чисто синтаксическими условиями, распознаваемыми в полиномиальное время.

Как указывалось во введении, двойственным по отношению к стабильности является понятие гомеостатичности, при котором внешние коррекции нарушают допустимость состояний (вносят возмущения), а модификации восстанавливают допустимость (устраняют возмущения). При подходящем уточнении этого понятия для него оказываются справедливы результаты, аналогичные теореме 4.

Авторы благодарны М.В.Хомякову за содержательные обсуждения примеров стабильного поведения логических программ и возможных подходов к созданию гомеостатических логических сред.

Список литературы

[1] Ceri S., Gottlob G., Tanka L. *Logic programming and databases.* Springer-Verlag, Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, 1989.

[2] Lobo J., Minker J., Rajasekar A. *Foundations of disjunctive logic programming.* The MIT Press. Cambridge, Massachusetts. London,England, 1992.

[3] Bonner A.J. *Hypothetical Datalog: complexity and expressibility.*// Theoretical Computer Science, 1990 - vol.76, N 1.

[4] Clocksin W.F., Mellish C.S. *Programming in Prolog.* 2nd edition, Springer Verlag, New York, 1984.

[5] Sterling L., Shapiro E. *The art of Prolog. Advanced programming techniques.* The MIT Press, Massachusetts. London, England, 1986.

[6] Clark K.L. *Predicate logic as a computational formalism.*// Research Rept. 79/59, Department of Computing, Imperial College, 1979.

[7] Lloyd J.W. *Foundations of logic programming.* Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1984.

[8] Apt K.R., Van Emden M.H. *Contributions to the theory of logic programming.*// Journal of the ACM, 1982 - vol. 29, N 3.

[9] Диковский А.Я. *Решение в линейное время алгоритмических проблем, связанных с синтезом ациклических программ.* // Программирование, 1985, 3.

[10] Apt K.R., Blair H. and Walker A. *Towards a theory of declarative knowledge.* // J. Minker (ed.) Proc. of the Workshop on Foundations of Deductive Databases and Logic Programming, Washington, 18-22 Aug., 1986.

[11] Маслов С.Ю. *Теория дедуктивных систем и ее применения.* М., "Радио и связь", 1986.