ТРУДЫ МАТЕМАТИЧЕСКОГО ИНСТИТУТА ИМ. В.А. СТЕКЛОВА, 2011, т. 274, с. 297–313

УДК 510.67

Сравнение выразительной силы некоторых языков запросов для баз данных[1]

©2011 г. М. А. Тайцлин[2]

Поступило в марте 2011 г.

Выдающемуся математику и замечательному человеку
Сергею Ивановичу Адяну к его 80-летию

Рассматриваются некоторая версия языка SQL и версия стратифицированного Дейталога, и доказывается взаимная транслируемость одного языка в другой.

Решающую роль в моей научной жизни сыграл мой учитель выдающийся русский математик Анатолий Иванович Мальцев. После смерти Анатолия Ивановича я столкнулся с большими трудностями, которые вряд ли смог бы преодолеть, если бы не имел постоянной поддержки Сергея Ивановича Адяна. Для меня было большой честью и очень приятно получить приглашение участвовать в этом сборнике научных работ.

1. ВВЕДЕНИЕ

После знаменитых работ Кодда (см. [9, 10]) общепринятой моделью базы данных является реляционная модель, в которой база данных мыслится как конечный набор конечных таблиц. Каждая таблица при этом имеет заранее фиксированное число столбцов с фиксированными названиями. Имя самой таблицы тоже заранее фиксировано. Меняться могут только число строк и сами строки таблиц.

Существует несколько языков, которые удобны для извлечения сведений о хранящейся информации в базе данных. Наиболее распространенным (а в последнее время, возможно, и единственным) из них стал язык SQL. Описание стандартов SQL и приемов программирования на SQL можно найти во многих руководствах, например в [18, 20–22].

Как было замечено еще Коддом, SQL является по существу некоторой модификацией языка логики предикатов. Главным недостатком старых стандартов SQL было то, что существуют очень просто распознаваемые запросы, которые нельзя задать в языке SQL. Например, связность графа нельзя задать формулой логики предикатов. Это было замечено еще в [8]. По этой причине в последних стандартах SQL стали допускать рекурсивные вызовы.

Другим достаточно известным языком является язык Пролог. Но давно замечено, что, возможно, более эффективной стратегией по сравнению со стратегией Пролога является стратегия поиска “снизу вверх” (bottom-up evaluation; см., например, [13]). Так возник язык Дейталог. Как самостоятельный раздел логического программирования Дейталог развивается с 1977 г. после известного симпозиума по логике и базам данных (см. [1]).

При допущений в правилах отрицаний понимание смысла правил становится неоднозначным. Предлагалось много разных вариантов понимания отрицаний, но ни один из них не

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 08-01-00241).

[2]Кафедра информатики, Факультет прикладной математики и кибернетики, Тверской государственный университет, Тверь, Россия.
E-mail: Michael.Taitslin@tversu.ru

является достаточно удовлетворительным. По этой причине возник стратифицированный Дейталог, в котором допускаются отрицания, но только тех отношений, вычисление которых уже закончено.

При задании элементов таблиц возникает проблема, откуда выбираются эти элементы. Удобно заранее фиксировать область, из которой берутся элементы таблиц. Но тогда немедленно возникает возможность использовать отношения, заданные на этом универсуме, в формулировке запросов. Это уже допускалось в самых первых стандартах SQL. В этой ситуации многое зависит от выбора универсума.

Здесь мы в качестве универсума берем множество натуральных чисел. Если остановиться на таком выборе, то ответы на запросы, задаваемые даже формулами логики предикатов, могут не быть конечными таблицами. Например, запрос $x > 2$ имеет в качестве ответа бесконечное множество. Канеллакис с соавторами (см. [14]) предложил рассматривать конечно представимые (finitely representable) отношения вместо конечных таблиц. В этом случае ответ на SQL-запрос, например не содержащий рекурсивных вызовов, снова конечно представим. Конечно представленными эти авторы называли отношения, которые заданы бесквантворными формулами.

Но для того, чтобы ответ на конечно представленный запрос был конечно представленным, удобно наряду с обычным линейным порядком рассматривать для всех натуральных p так называемые дырявые порядки (gap-orders) $<_p$. В таком случае, если еще добавить константы для всех натуральных чисел и отношения $x =_p y$ для формул $x + p = y$, каждая формула становится эквивалентной положительной бесквантворной.

Третьим языком, который давно изучается, является язык логики неподвижных точек (fixed point logic).

Эта логика использовалась многими авторами (см., например, [2–5, 7, 11, 12, 15]). Один из основных результатов здесь — точная характеризация запросов, вычислимых за полиномиальное время, как запросов, задаваемых формулами логики неподвижных точек (теорема Ливчака–Иммермана–Варди). Отличие нашего подхода состоит только в том, что мы рассматриваем структуру, которая не является конечной. Поэтому не каждый оператор неподвижной точки дает результат. В некоторых случаях процесс может никогда не закончиться.

Взаимная транслируемость в определенном смысле формул логики неподвижных точек и стратифицированных дейталоговских программ тоже хорошо известна (см. об этом, например, в [15]). Наши рассмотрения имеют ту особенность, что предметная область бесконечна.

Здесь мы доказываем, что при определенном выборе определений все три языка имеют равную выразительную силу.

2. ДЕЙТАЛОГ

Мы рассматриваем множество $\omega = \{0, 1, 2, \dots\}$ натуральных чисел вместе с отношением $<$ обычного линейного порядка и с выделенным элементом 0. Наряду с отношением $<$ мы будем также рассматривать отношения $<_p$ для фиксированных положительных натуральных чисел p. Здесь $x <_p y$ означает, что $(x+p) < y$, другими словами, что между x и y располагаются еще p различных чисел, отличных от x и y. Под $x <_0 y$ мы понимаем просто $x < y$. Пусть $x =_i y$ для натурального положительного числа i есть сокращение для $x <_{(i-1)} y \,\&\, \neg\, x <_i y$. Пусть $x =_0 y$ есть $x = y$. Хотя 0 и эти отношения $<_p$, $=_p$ задаются на этом линейно упорядоченном множестве натуральных чисел формулами логики предикатов, мы добавляем их для того, чтобы в полученном обогащении каждая формула была эквивалентна не содержащей отрицаний бесквантворной. Под формулой мы здесь и далее понимаем формулу логики предикатов (см., например, [19]), не содержащую импликаций. Все другие не объясняемые здесь термины и обозначения тоже взяты из [19].

Лемма 2.1. *В системе* $(\omega, 0, \{=_p, <_p \mid p \in \omega\})$ *каждая формула эквивалентна бескванторной формуле, не содержащей отрицаний. В системе* $(\omega, =, \{p, <_p \mid p \in \omega\})$ *каждая экзистенциальная формула, не содержащая отрицаний, эквивалентна бескванторной формуле, не содержащей отрицаний.*

Доказательство. Как обычно, достаточно рассмотреть случай формулы вида $(\exists x)\Phi$, где Φ — бескванторная формула. Достаточно рассмотреть только случай, когда Φ является конъюнкцией базисных формул (другими словами, атомных формул и отрицаний атомных формул). Но отрицание формулы $x < y$ можно заменить на $y < x \vee y = x$, а при $p > 0$ отрицание формулы $x <_p y$ можно заменить на

$$y < x \vee y = x \vee x =_1 y \vee \ldots \vee x =_{(p-1)} y.$$

Существует конечное число различных возможностей взаимного расположения встречающихся свободных переменных между собой, x и 0. Если x попадает в интервал между y и свободным переменным z (y при этом может быть свободным переменным или 0), то достаточно квантор по x убрать, соответствующим образом исправив расстояние между y и z. $\square$

Формулы, не содержащие отрицаний, мы называем положительными.

Пусть $\Sigma = \langle 0, \{=_p, <_p \mid p \in \omega\}\rangle$.

Мы называем *атомами* атомные формулы сигнатуры $\Sigma_0 = \langle =, \{p, <_p \mid p \in \omega\}\rangle$.

Далее мы рассматриваем некоторый конечный набор символов отношений с указанием числа аргументных мест (местности или арности) каждого символа. Эти символы мы называем *входными* или *внешними*, понимая при этом, что их интерпретации представляют хранящуюся информацию. Другими словами, нам задана *база данных*. Указанный конечный набор символов отношений с указанием местности каждого отношения называется *схемой этой базы данных*.

Для написания программы стратифицированного Дейталога нужен еще другой конечный набор символов отношений, не содержащий входных символов. Эти символы отношений называют *внутренними* символами или внутренними именами. Каждый внутренний символ тоже имеет число аргументных мест. Внутренние символы иногда делят на *выходные*, в которых содержится результат работы программы, и *промежуточные*, которые служат для хранения промежуточных результатов. Все внутренние символы разбиты на ранги. Ранги начинаются с 1 и охватывают несколько последовательных натуральных чисел. Всегда имеются внутренние символы ранга 1. Внутренние символы ранга $(i + 1)$ имеются только в том случае, когда имеются внутренние символы ранга i. Имеется также бесконечное множество предметных переменных. Как обычно, предметные переменные можно мыслить себе как слова в алфавите, например, из | и x. Например, x является переменной и если y является переменной, то $y|$ тоже является переменной. Но конкретный способ изображения переменных, разумеется, не является существенным.

Смысл ранжирования состоит в том, что мы рассматриваем стратифицированное отрицание. Можно использовать отрицание внутреннего имени, но только после того, как уже закончено вычисление значения этого имени.

Формальное определение состоит в следующем. Программа стратифицированного Дейталога является конечной последовательностью правил. Каждое правило состоит из головы и тела.

Голова правила имеет вид $P(x_1, \ldots, x_n)$, где P является внутренним символом местности n, а $x_1, \ldots, x_n$ — это последовательность попарно различных переменных. Ранг правила — это ранг символа P. При этом P называется именем правила.

Тело правила либо пусто, либо является конечной последовательностью формул.

Каждая формула в теле правила имеет один из следующих видов:

- атом;
- $Q(y_1, \ldots, y_k)$, где Q — внешний символ местности k, а $y_1, \ldots, y_k$ — последовательность переменных;
- $Q(y_1, \ldots, y_k)$, где Q — внутренний символ местности k, ранг которого не превосходит ранг правила, а $y_1, \ldots, y_k$ — последовательность переменных;
- $\neg Q(y_1, \ldots, y_k)$, где Q — внутренний символ местности k, ранг которого *строго меньше* ранга правила, а $y_1, \ldots, y_k$ — последовательность переменных.

Обычно голова правила отделяется от его тела знаком $\leftarrow$.

Не требуется, чтобы каждая переменная, входящая в тело правила, входила в его голову. Смысл правила в том, что если можно так подобрать значения переменных, не входящих в голову правила, что все формулы тела правила станут истинными, то этот набор значений переменных головы правила включается в интерпретацию имени правила. Формальное определение работы программы стратифицированного Дейталога состоит в следующем.

Конечным представлением мы называем приписывание каждому внешнему символу отношения такой положительной бескванторной формулы сигнатуры Σ_0, что эта формула содержит различных переменных не более, чем местность этого символа. Переменные мы считаем упорядоченными (например, по числу | после первой буквы, если переменными являются слова, начинающиеся на эту букву, после которой идут символы |). Добавляя при необходимости конъюнктивные члены вида $z = z$, можно считать, что число различных переменных в приписанной формуле равно местности рассматриваемого внешнего символа. При этом для определенности мы считаем первую переменную значением первой координаты отношения, вторую — второй и т.д., последнюю — последней. Заметим, что каждая конечная таблица заведомо может быть конечно представлена. Действительно, кортеж $(i_1, \ldots, i_k)$ представляется формулой $(x_1 = i_1 \,\&\, \ldots \,\&\, x_k = i_k)$. Таблица из некоторого конечного числа строк представляется дизъюнкцией таких формул.

Мы рассматриваем линейный порядок на переменных, имея в виду, что переменные занумерованы натуральными числами и что этот порядок индуцируется порядком на номерах переменных. Это отношение порядка мы обозначаем символом $<$. Ясно, что это не приводит к смешению с порядком на значениях этих переменных.

Здесь и далее в таких случаях мы считаем, что $x_1 < \ldots < x_k$. Это означает, что мы считаем, что номер x_1 всегда меньше номера x_2, номер x_2 меньше номера x_3 и т.д., номер x_{k-1} меньше номера x_k. Запись $Q(y_1, \ldots, y_k)$ означает, что местность Q равна k, что $y_1 < \ldots < y_k$ в рассматриваемом упорядочении переменных и что список $y_1, \ldots, y_k$ — это список всех переменных, входящих в формулу $Q(y_1, \ldots, y_k)$. Формула $Q(y_1, \ldots, y_k)$ истинна для заданного кортежа $(i_1, \ldots, i_k)$ натуральных чисел, если эта формула истинна, когда y_1 принимает значение i_1, $\ldots$, y_k принимает значение i_k.

Состояние базы данных — это некоторое конечное представление этой базы данных. Состояние σ базы данных приписывает каждому внешнему символу R формулу $\sigma(R)$. Начальное состояние всем внутренним символам приписывает пустые множества кортежей натуральных чисел, а внешним символам приписывает формулы, которые в процессе работы программы не меняются. Далее множества, приписанные внутренним символам, возрастают по следующим правилам.

Работа программы проходит по шагам. Шаги занумерованы парами (i, j) положительных натуральных чисел. Первый шаг занумерован парой $(1, 1)$. На шаге (i, j) происходит изменение множеств, приписанных внутренним символам ранга i. Пары (i, j) упорядочены лексикографически.

Мы скажем, что для заданного кортежа $(i_1, \ldots, i_k)$ натуральных чисел и символа Q местности k формула $Q(y_1, \ldots, y_k)$ истинна, если на этом шаге этот кортеж включен в множество кортежей, приписанных этому символу. При этом внешнему символу приписывается множество кортежей, на которых истинна приписанная этому символу формула. Мы скажем, что для заданного кортежа $(i_1, \ldots, i_k)$ натуральных чисел и символа Q местности k формула $\neg Q(y_1, \ldots, y_k)$ истинна на шаге (i, j), если Q является внутренним символом, ранг которого меньше i, и на предыдущих шагах этот кортеж не был включен в множество кортежей, приписанных этому символу.

Если на шаге (i, j) окажется, что нет внутренних символов ранга i, то рассматриваемая программа останавливается. Результатом работы программы являются множества, приписанные внутренним символам. В противном случае на шаге (i, j) могут измениться только множества, приписанные внутренним символам ранга i.

Пусть P является внутренним символом ранга i и местности n. Кортеж $(j_1, \ldots, j_n)$ включается на шаге (i, j) в множество, приписанное символу P, в следующих случаях:

- если этот кортеж был включен в множество, приписанное этому символу, ранее;
- если в рассматриваемой программе найдется правило

$$P(x_1, \ldots, x_n) \leftarrow \Phi_1 \ldots \Phi_m$$

  и можно так подобрать натуральные числа в качестве значений переменных, не входящих в голову этого правила, что после придания x_1 значения j_1, $\ldots$, x_n значения j_n все формулы $\Phi_1, \ldots, \Phi_m$ окажутся истинными.

Если кортеж $(j_1, \ldots, j_n)$ включается на шаге (i, j) в множество, приписанное символу P, и этот кортеж не был включен в это множество на предыдущих шагах, то этот кортеж добавляется в это множество. Если никакой кортеж не добавляется ни в какое множество на шаге (i, j), то программа переходит к выполнению шага $(i + 1, 1)$. В противном случае программа переходит к выполнению шага $(i, j + 1)$.

Лемма 2.2. *После шага (i, j) (перед началом выполнения следующего шага) каждое множество, приписанное внутреннему символу, может быть задано бескванторной положительной формулой сигнатуры Σ. При этом после шага $(1, i)$ каждое множество, приписанное внутреннему символу, может быть задано бескванторной положительной формулой сигнатуры Σ_0.*

Доказательство. Лемма доказывается индукцией. Перед шагом $(1, 1)$ заключение леммы, очевидно, выполнено. Если это заключение выполнено перед выполнением некоторого шага, то оно выполнено после выполнения этого шага. Это следует из леммы 2.1. Дело в том, что каждое правило, имя которого есть P, меняет только формулу, приписанную этому P. При этом эта формула заменяется дизъюнкцией этой формулы и еще некоторой экзистенциальной формулы сигнатуры Σ. $\square$

Таким образом, если программа останавливается, то результат работы программы есть некоторый набор бескванторных положительных формул сигнатуры Σ.

Смысл рассмотрения положительных бескванторных формул вместо конечных таблиц как раз в том и состоит, что для заданных конечных таблиц результат работы дейталоговской программы может не быть набором конечных таблиц, но всегда задается набором положительных бескванторных формул.

Если все внутренние символы имеют ранг 1, то программа всегда останавливается (см. [6, Theorem 4.1]). Так как мы рассматриваем несколько другую ситуацию, а формулировка и доказательство теоремы 4.1 записаны в [6] не совсем аккуратно, мы приводим здесь полное доказательство.

Назовем дейталог-программу *нестратифицированной*, если все внутренние символы имеют ранг 1 или если внутренним символам вообще не приписаны ранги. Этот термин является общепринятым. Если внутренним символам вообще не приписаны ранги, то мы будем считать, что всем внутренним символам приписан ранг 1. Назовем дейталог-программу *безопасной*, если эта программа останавливается на каждом состоянии.

Теорема 2.3. *Каждая нестратифицированная программа является безопасной.*

Предварительно напомним два хорошо известных замечания, доказательства которых приведены, например, в [6] (см. там факты 4.3 и 4.4). Мы рассматриваем частичный порядок на конечных последовательностях натуральных чисел одной и той же длины. Из двух различных таких последовательностей мы считаем первую меньшей, а вторую большей, если каждый член первой последовательности не превосходит соответствующего члена второй последовательности. Соответствующими мы называем первые члены, вторые члены и т.д., наконец, последние члены. В данном множестве таких последовательностей некоторую последовательность мы называем *минимальной* (*максимальной*), если в этом множестве нет меньшей (большей) последовательности, чем рассматриваемая последовательность. Две такие последовательности называются сравнимыми, если одна из них меньше другой.

Замечание 2.4. В любом множестве конечных последовательностей натуральных чисел одной и той же длины подмножество минимальных последовательностей конечно.

Замечание 2.5. Пусть k — положительное натуральное число. Любая последовательность попарно различных конечных множеств конечных последовательностей длины k, составленных из натуральных чисел, в которой каждая последовательность из каждого следующего множества либо не является сравнимой ни с какой последовательностью из предыдущего, либо меньше или равна некоторой последовательности из предыдущего, конечна.

Доказательство теоремы 2.3. В программе используются конечное число натуральных чисел и конечное число переменных для каждого имени. Поэтому для каждого внутреннего имени существует только конечное число возможных взаимных расположений этих переменных и чисел. Под взаимным расположением мы здесь понимаем конъюнкцию всех верных формул видов $x < y$ и $x = y$, в которых каждое из x и y — это либо одно из рассматриваемых чисел, либо одна из рассматриваемых переменных. Так как возможных взаимных расположений только конечное число, то достаточно доказать, что для фиксированного внутреннего имени и фиксированного возможного взаимного расположения найдется такой шаг работы программы, после которого в множество, приписанное рассматриваемому внутреннему имени, не добавляется никаких кортежей с рассматриваемым взаимным расположением. Так как возможных значений каждой переменной, не превосходящей некоторого числа, только конечное число, то для каждого взаимного расположения мы будем изучать наборы значений только тех переменных, которые больше всех чисел. Для такого набора мы будем рассматривать кортеж расстояний между соседними элементами набора и между наименьшим элементом набора и наибольшим числом. Более подробно, пусть n — это наибольшее из всех встречающихся в программе натуральных чисел. Зафиксируем некоторое возможное взаимное расположение встречающихся в рассматриваемой программе натуральных чисел и переменных рассматриваемого внутреннего имени. Пусть $z_1, \ldots, z_k$ — это набор всех тех переменных в порядке возрастания их значений, которые больше n. Так как мы фиксировали взаимное расположение, то этот набор определяется однозначно. С каждым кортежем $(d_1, \ldots, d_k)$ значений $z_1, \ldots, z_k$ свяжем кортеж натуральных чисел $(d_1 - n, d_2 - d_1, \ldots, d_k - d_{k-1})$. Этот кортеж назовем набором расстояний для кортежа $(d_1, \ldots, d_k)$. Мы будем называть кортеж $(d_1, \ldots, d_k)$ кортежем *с минимальными расстояниями* среди некоторого множества таких кортежей, если набор расстояний для рассматриваемого кортежа минимален (нет в этом множестве кортежа с другим набором расстояний, для которого каждая координата в наборе расстояний не превосходит соответствующей

координаты в наборе расстояний для рассматриваемого кортежа). Аналогично мы будем говорить, что второй кортеж превосходит первый по расстояниям, если каждая координата в наборе расстояний для первого кортежа не превосходит соответствующей координаты в наборе расстояний для второго кортежа.

Лемма 2.6. *Если на некотором шаге $(1,i)$ вычислений некоторый кортеж включается в множество кортежей, приписанных некоторому внутреннему имени, то на этом шаге в это множество включаются и все превосходящие по расстояниям кортежи.*

Доказательство. Посмотрим, что происходит на шаге $(1,i)$. Некоторый набор $(j_1,\dots,j_n)$ добавляется в множество, приписанное некоторому внутреннему символу P, если существует правило

$$P(x_1,\dots,x_n) \leftarrow \Phi_1 \dots \Phi_m$$

и можно так подобрать натуральные числа в качестве значений переменных, не входящих в голову этого правила, что после придания x_1 значения j_1, $\dots$, x_n значения j_n все формулы $\Phi_1,\dots,\Phi_m$ окажутся истинными. Эти формулы являются бесквантормыми положительными формулами сигнатуры Σ_0, приписанными внешним символам, либо атомами, либо бесквантормыми положительными формулами сигнатуры Σ_0, приписанными внутренним символам перед началом выполнения шага $(1,i)$. Как уже отмечалось, каждое правило, имя которого есть P, меняет только формулу, приписанную этому P. При этом формула заменяется дизъюнкцией этой формулы и еще некоторой экзистенциальной положительной формулы сигнатуры Σ_0. Как уже отмечалось в лемме 2.2, полученная дизъюнкция эквивалентна бесквантормой положительной формуле сигнатуры Σ_0. Осталось заметить, что если бесквантормой положительной формуле сигнатуры Σ_0 удовлетворяет некоторый кортеж натуральных чисел, то этой формуле удовлетворяет и любой превосходящий по расстояниям кортеж. Но это совершенно очевидно. □

Поэтому достаточно понять, что после некоторого шага работы программы новых кортежей с минимальными расстояниями не появляется. Это вытекает из приведенных замечаний. □

Теорема 2.3 относится к нестратифицированным программам, однако стратифицированная программа может моделировать работу произвольной машины Тьюринга и не останавливаться, если эта машина Тьюринга не останавливается (подробности см. в [17]).

Дейталоговская программа может задавать транзитивное замыкание входного отношения. Например, для входного бинарного символа P и внутреннего бинарного символа Q программа может содержать правила

$$Q(x,y) \leftarrow P(x,y), \qquad Q(x,y) \leftarrow Q(x,z)\ P(z,y).$$

Если P произвольно, то эта программа может не останавливаться, но для конечно представленного P эта программа всегда остановится. Это показывает, что теорема 2.3 не совсем тривиальна. Она, в частности, утверждает, что для конечно представленного бинарного отношения длина минимального пути между двумя вершинами в соответствующем бесконечном графе всегда для любых двух вершин, для которых существует путь из первой вершины во вторую, не превосходит заранее заданного натурального числа.

Возможно, простейшей программой, которая не останавливается для конечно представленного P, является следующая программа:

$$Q(x) \leftarrow P(x), \qquad Q(x) \leftarrow Q(z)\ z =_2 x.$$

Эту программу можно переписать как стратифицированную, приписывая $Q(x)$ ранг 2, добавляя бинарный внутренний символ Q_1 и правила

$$Q_1(x,y) \leftarrow x <_2 y, \qquad Q(x) \leftarrow P(x), \qquad Q(x) \leftarrow Q(z)\ z <_1 x\ \neg Q_1(z,x).$$

Если $P(x)$ задана как $x = 2$, программа не останавливается. Это показывает, что для справедливости утверждения о том, что нестратифицированные программы всегда останавливаются, под атомами надо понимать атомные формулы сигнатуры Σ_0.

Аналогичный пример показывает, что для справедливости теоремы 2.3 нужно требовать, чтобы внешним символам были приписаны бесквантворные положительные формулы сигнатуры Σ_0. Однако если не заботиться о справедливости теоремы 2.3, то внешним символам можно присваивать и любые формулы сигнатуры Σ. От этого класс отношений, получаемых в результате работ дейталоговских программ, не расширится. Другими словами, любую формулу сигнатуры Σ можно получить как результат работы некоторой дейталоговской программы на входе, заданном набором бесквантворных положительных формул сигнатуры Σ_0.

3. SQL

3.1. Простые случаи. Мы предлагаем такую версию SQL, при которой элементы строк таблиц берутся из множества натуральных чисел, а, кроме имен таблиц, в запросах используются только символы сигнатуры $\Sigma = \langle 0, \{=_p, <_p \mid p \in \omega\}\rangle$. В этом случае каждое натуральное число p представимо в виде x с условием $0 =_p x$. Как уже отмечалось, каждая конечная таблица задается некоторой бесквантворной положительной формулой сигнатуры Σ_0. Но, как и ранее, мы будем рассматривать более общий случай произвольной бесквантворной положительной формулы сигнатуры Σ_0, интерпретация которой может не быть конечной таблицей.

В дальнейшем нам удобно бесквантворные положительные формулы сигнатуры Σ называть таблицами, а кортежи значений переменных такой формулы, при которых эта формула истинна, называть строками этой таблицы.

Итак, фиксируется некоторый конечный набор символов отношений с указанием числа аргументных мест каждого символа. Эти символы мы называем *входными* или *внешними*, понимая при этом, что их интерпретации представляют хранящуюся информацию. Другими словами, нам задана база данных. Каждому внешнему символу приписана бесквантворная положительная формула сигнатуры Σ_0, при этом число переменных в приписанной формуле совпадает с числом аргументных мест этого символа. Как уже отмечалось, переменные линейно упорядочены и для определенности мы считаем первую переменную значением первой координаты отношения, вторую — второй и т.д., последнюю — последней. Приписанные формулы могут быть произвольными, но последовательность переменных мы считаем зафиксированной для каждого внешнего символа. Другими словами, мы считаем заданными названия столбцов каждой таблицы. Мы говорим, что переменная приписана внешнему символу или входит в него, если эта переменная входит в формулу, приписанную этому символу.

SQL-программа выдает конечный набор формул в качестве результата своей работы.

Например, имея таблицу $R(x, y)$, мы желаем найти строки, в которых $x <_2 y$. Это можно задать следующим запросом:

```
select *
  from R
  where x <2 y
```

В результате мы получим таблицу, которая будет содержать те и только те строки из таблицы R, в которых $x + 2 < y$.

Формальное определение состоит в следующем[3]. SQL-программа представляет собой конечную последовательность запросов. В простейшем случае запрос определяется следующим

[3]В определениях конструкций SQL я частично использую обозначения из неопубликованной рукописи С.М. Дудакова (см. [23]). Хотя, конечно, здесь рассматривается другая ситуация.

образом:

```
select <список аргументов>
  from <имя таблицы>
```

Здесь ⟨имя таблицы⟩ — это один из внешних символов; ⟨список аргументов⟩ — это последовательность переменных, каждая из которых входит в формулу, приписанную этому внешнему символу. В списке аргументов аргументы разделяются запятой.

Если список аргументов представляет собой все аргументы, расположенные в том же порядке, вместо списка аргументов можно написать $*$.

Если мы хотим наложить некоторое условие на извлекаемые строки, понадобится инструкция `where`:

```
select <список аргументов>
  from <имя таблицы>
  where <условие>
```

В качестве условия может быть использована любая такая бесквантороная положительная формула сигнатуры Σ, что переменные этой формулы встречаются среди переменных формулы, приписанной внешнему символу ⟨имя таблицы⟩.

В более сложном случае запрос имеет вид

```
select <список аргументов>
  from <список таблиц>
  where <условие>
```

Здесь ⟨список таблиц⟩ — это последовательность внешних символов. В списке таблиц внешние символы разделяются запятой. В этом случае может оказаться, что два разных внешних символа имеют общую переменную. Чтобы различать, из какой таблицы выбирать значение этой переменной, ⟨список аргументов⟩ представляет собой последовательность разделенных запятыми выражений следующего вида:

$$\langle\text{внешний символ}\rangle.\langle\text{одна из переменных, приписанных этому символу}\rangle\,. \tag{1}$$

При этом ⟨внешний символ⟩ должен встречаться в последовательности ⟨список таблиц⟩. В этом случае ⟨условие⟩ — это бесквантороная положительная формула сигнатуры Σ, переменные которой имеют вид (1), где ⟨внешний символ⟩ встречается в последовательности ⟨список таблиц⟩. Если какая-то переменная приписана только одному из внешних символов, входящих в ⟨список таблиц⟩, в последовательности ⟨список аргументов⟩ можно просто писать эту переменную без указания, к какому внешнему символу она приписана.

В некоторых случаях удобно переменным результирующей таблицы присваивать новые имена. Иногда, например, для сравнения между собой различных полей строки одной и той же таблицы в последовательности ⟨список таблиц⟩ удобно несколько раз повторить один и тот же внешний символ. В этих случаях можно использовать конструкцию

```
select <список аргументов>
  from <список таблиц>
  where <условие>
```

в которой ⟨список таблиц⟩ — это последовательность разделенных запятыми выражений следующего вида:

$$\langle\text{внешний символ}\rangle\ \langle\text{новое имя для него}\rangle\,. \tag{2}$$

Здесь ⟨новое имя для него⟩ — это новый, не встречавшийся ранее внешний символ, а ⟨список аргументов⟩ — это последовательность разделенных запятыми выражений следующего вида:

$$\langle\text{новое имя внешнего символа}\rangle.\langle\text{одна из переменных, приписанных этому символу}\rangle \text{ as } \langle\text{новое имя для этой переменной}\rangle\,. \quad (3)$$

При этом ⟨новое имя для этой переменной⟩ — это переменная, которая ранее не использовалась. Новые имена для переменных должны быть выбраны таким образом, чтобы разные переменные получили разные новые имена; ⟨условие⟩ в этом случае — это бескванторная положительная формула сигнатуры Σ, переменные которой имеют вид

$$\langle\text{новое имя внешнего символа}\rangle.\langle\text{одна из переменных, приписанных этому символу}\rangle\,. \quad (4)$$

При этом ⟨внешний символ⟩ встречается в последовательности ⟨список таблиц⟩.

3.2. Группировка данных и агрегатные функции. Список группировки является списком полей, разделенных запятыми. Семантика оператора `group by` такая: все строки объединяются в несколько групп таким образом, что каждую группу образуют строки, у которых значения всех столбцов из списка группировки совпадают. Затем из каждой группы формируется одна строка.

Для того чтобы из каждой группы выбрать одно значение для какого-либо поля или сгенерировать новое значение, используются агрегатные функции: `max` — для выбора максимального значения, `min` — для минимального, `sum` — для нахождения суммы в группе значений некоторой координаты, не включенной в список группировки. Еще одна агрегатная функция `count(*)` предназначена для подсчета количества строк в группе.

Если строк в группе конечное число, то определения этих функций очевидны. В другом случае определена только `min`. Во всяком случае все агрегатные функции определены для конечных таблиц.

С учетом этих соображений наиболее общий запрос без подзапросов имеет вид

```
select <список аргументов>
  from <список таблиц>
  where <условие>
  group by <список группировки>
  having <условие1>
```
(5)

Здесь ⟨список таблиц⟩ определяется по-прежнему как последовательность разделенных запятыми выражений вида (2); ⟨список группировки⟩ — это последовательность разделенных запятыми выражений вида (4). Выражения, входящие в список группировки, называются элементами списка группировки; ⟨список аргументов⟩ — это последовательность разделенных запятыми выражений следующих видов:

$$\langle\text{элемент списка группировки}\rangle \text{ as } \langle\text{новое имя для этой переменной}\rangle\,, \quad (6)$$

$$\langle\text{агрегатная функция}\rangle \text{ as } \langle\text{новое имя для этой функции}\rangle\,. \quad (7)$$

Переменная входит в запрос тогда и только тогда, когда она имеет вид ⟨новое имя для этой переменной⟩ в выражении вида (6) или (7) из последовательности ⟨список аргументов⟩. В этом случае ⟨условие1⟩ — это бескванторная положительная формула сигнатуры Σ, переменные которой имеют вид ⟨новое имя для этой переменной⟩ в выражении вида (6) или (7) из последовательности ⟨список аргументов⟩. В этом случае ⟨условие⟩ — это бескванторная положительная формула сигнатуры Σ, переменные которой имеют вид (4), где ⟨внешний символ⟩ встречается в последовательности ⟨список таблиц⟩.

3.3. Действия с запросами.

3.3.1. *Объединение и пересечение запросов.* Аргументами в теоретико-множественных операциях с запросами являются запросы, в которые входит одинаковое число переменных. Все рассматриваемые операции имеют по два аргумента каждая. Предполагается, что каждая переменная входит в первый аргумент тогда и только тогда, когда она входит во второй.

Объединение запросов Запрос1 и Запрос2 задается запросом

```
(Запрос1)
 union
(Запрос2)
```

Вторая из записей

```
((Запрос1)
 union
(Запрос2))
 union
(Запрос3)
```

```
(Запрос1)
 union
(Запрос2)
 union
(Запрос3)
```

рассматривается как сокращение для первой.

Для пересечения запросов Запрос1 и Запрос2 используется конструкция `intersect`:

```
(Запрос1)
 intersect
(Запрос2)
```

Как и в случае объединения, при пересечении нескольких запросов можно опускать скобки.

В результирующий запрос при объединении и при пересечении входят те и только те переменные, которые входят в первый аргумент.

3.3.2. *Представления.* Как и ранее, каждый внутренний символ имеет ранг и число аргументных мест. Ранги начинаются с 1 и охватывают несколько последовательных натуральных чисел. Всегда имеются внутренние символы ранга 1. Внутренние символы ранга $(i+1)$ имеются только в том случае, когда имеются внутренние символы ранга i.

Представление запроса создается с помощью оператора `create view` вида

```
create view <имя представления> as
  <SQL-запрос>
```
(8)

или вида

```
create view <имя представления> as
  not <SQL-запрос>
```
(9)

Команда (9) требует, чтобы имени представления приписывалась формула, являющаяся отрицанием формулы, приписанной данному SQL-запросу.

Имя представления — это новый внутренний символ. Его местность — это число переменных, входящих в запрос. Переменная входит в представление тогда и только тогда, когда она входит в создающий это представление запрос. Мы считаем, что для каждого запроса имеется представление.

При определении запроса в (5) в последовательности ⟨список таблиц⟩ можно включать не только внешние символы отношений, но и имена представлений. Поэтому ⟨список таблиц⟩ теперь — это последовательность разделенных запятыми выражений следующих видов:

$$\langle\text{внешний символ}\rangle\ \langle\text{новое имя для него}\rangle\,, \tag{10}$$

$$\langle\text{имя представления}\rangle\ \langle\text{новое имя для него}\rangle\,. \tag{11}$$

В остальных определениях вместо "⟨внешний символ⟩" надо писать "⟨внешний символ⟩ или ⟨имя представления⟩".

При этом могут возникать рекурсивные вызовы. Однако должны соблюдаться следующие условия.

Назовем рангом представления ранг имени этого представления. Представление используется в другом представлении в случаях (8) и (9), если имя первого представления встречается в последовательности ⟨список таблиц⟩ для второго представления. Представление используется в объединении или пересечении представлений, если оно используется в одном из объединяемых или пересекаемых представлений.

Требуется, чтобы ранг любого представления не был меньше никакого ранга используемого представления в случае (8) и был строго больше каждого ранга используемого представления в случае (9). Ранг объединения или пересечения представлений должен быть строго больше ранга каждого аргумента.

3.4. Семантика. Напомним, что формулы, сопоставленные элементам последовательности ⟨список таблиц⟩, мы называем таблицами. Если задано несколько таблиц, то их тензорным произведением будем называть таблицу, число столбцов которой (число переменных результирующей формулы) равно сумме чисел столбцов перемножаемых таблиц, а каждая строка которой получается соединением строки первой таблицы со строкой второй таблицы и т.д., наконец, соединением со строкой последней таблицы и все такие соединения входят в тензорное произведение. При этом предполагается, что перемножаемые формулы попарно не имеют общих переменных, а переменные тензорного произведения — это все переменные перемножаемых таблиц и только они. Это означает, что тензорное произведение таблиц задается конъюнкцией формул, задающих эти таблицы.

Вычисление запроса проходит по шагам. Шаги занумерованы парами (i, j) положительных натуральных чисел. Первый шаг занумерован парой $(1, 1)$. На шаге (i, j) происходит вычисление запросов ранга i. Пары (i, j) упорядочены лексикографически.

До начала вычисления (до первого шага) всем запросам приписаны пустые множества кортежей.

Пусть до шага (i, j) всем запросам приписаны бесквантворные положительные формулы сигнатуры Σ. На шаге (i, j) вычисляются новые значения запросов ранга i.

Если запрос ранга i является объединением или пересечением запросов, то в результирующую таблицу входят те и только те строки, которые входят хотя бы в одну таблицу или соответственно одновременно в обе таблицы.

Пусть запрос ранга i имеет вид (5). Прежде всего составляем тензорное произведение этих таблиц. Это тензорное произведение в качестве столбцов имеет все столбцы перемножаемых таблиц, а в качестве строк имеет всевозможные строки, получаемые из строки первой таблицы путем присоединения строки второй таблицы и т.д., наконец, путем присоединения строки последней таблицы. Из полученной таблицы выбираем теперь только строки, удовлетворяющие формуле ⟨условие⟩. В полученной таблице группируем (объединяем) строки с совпадающими полями из последовательности ⟨список группировки⟩. В полученной таблице оставляем только столбцы из списка ⟨список аргументов⟩. В полученной таблице оставляем только строки, удовлетворяющие формуле ⟨условие1⟩. Полученную таблицу объединяем с таблицей, сопоставленной рассматриваемому запросу, и получаем таблицу, сопоставленную рассматриваемому запросу после шага (i, j).

Если представление ранга i имеет вид (8), то этому представлению сопоставляется та же таблица, которая сопоставлена создающему это представление запросу. Если представление ранга i имеет вид (9), то этому представлению сопоставляется отрицание того отношения, которое сопоставлено создающему это представление запросу. Так как ранг этого создающего

запроса меньше i, то формула, сопоставленная этому запросу, уже найдена до шага (i,j) и, следовательно, до этого шага уже найдено отрицание этой формулы. Как уже отмечалось (лемма 2.1), это отрицание тоже эквивалентно бесквантворной положительной формуле сигнатуры Σ.

Если никакая таблица не меняется на шаге (i,j), то вычисление переходит к выполнению шага $(i+1,1)$. В противном случае вычисление переходит к выполнению шага $(i,j+1)$. Если на шаге (i,j) окажется, что нет внутренних символов ранга i, то вычисление заканчивается. Результатом являются таблицы, приписанные внутренним символам.

Теорема 3.1. *После каждого шага вычисления (i,j) формулы, приписанные внутренним символам, являются бесквантворными положительными формулами сигнатуры Σ.*

Доказательство. Пусть до шага (i,j) всем запросам приписаны бесквантворные положительные формулы сигнатуры Σ. Остается доказать, что формулы, полученные после шага (i,j), тоже являются бесквантворными положительными формулами сигнатуры Σ.

В самом деле, тензорное произведение таблиц задается конъюнкцией формул, задающих эти таблицы; ⟨условие⟩ выделяет из этой конъюнкции те строки, которые выбираются. Если на полученную формулу навесить кванторы существования по столбцам, не включенным в ⟨список аргументов⟩, добавить столбцы, соответствующие включенным агрегатным функциям, вместе с определениями этих функций и оставить только строки, удовлетворяющие формуле ⟨условие1⟩, то и получим нужную формулу. Осталось заметить только, что агрегатные функции задаются бесквантворными положительными формулами сигнатуры Σ.

Итак, для заданной бесквантворной положительной формулы $\phi(\overline{x},\overline{y})$ сигнатуры Σ и заданного набора $\overline{a}$ натуральных чисел найдем наименьший $\overline{b}$, удовлетворяющий $\phi(\overline{a},\overline{b})$. Наборы упорядочены лексикографически.

Надо сначала написать правило, выделяющее среди наборов $\overline{z}$, удовлетворяющих $P(\overline{x},\overline{z})$, такие, для которых в этом множестве есть меньший в лексикографическом порядке. Дополнение этого множества выделяет минимальный набор.

Для определения максимального набора надо сначала написать правило, выделяющее среди наборов $\overline{z}$, удовлетворяющих $P(\overline{x},\overline{z})$, такие, для которых в этом множестве есть больший в лексикографическом порядке. Дополнение этого множества выделяет максимальный набор. Множество конечно тогда и только тогда, когда наибольший набор существует.

Пусть $P(\overline{x},\overline{y})$ — внешний символ для формулы $\phi(\overline{x},\overline{y})$ (или внутренний символ ранга i для представления, которому сопоставлена эта формула). Рассмотрим правила

$$Q(\overline{x},\overline{y},\overline{u}) \leftarrow P(\overline{x},\overline{z})\ \ P(\overline{x},\overline{y})\ \ P(\overline{x},\overline{u})\ \ \overline{y} <^1 \overline{z}\ \ \overline{z} <^1 \overline{u};$$

$$\mathrm{NOT}\, Q(\overline{x},\overline{y},\overline{u}) \leftarrow P(\overline{x},\overline{y})\quad P(\overline{x},\overline{u})\ \ \overline{y} <^1 \overline{u}\ \ \neg Q(\overline{x},\overline{y},\overline{u}),$$

в которых внутренний символ Q имеет ранг i, а внутренний символ $\mathrm{NOT}\, Q$ имеет ранг $(i+1)$. В этих правилах $<^1$ означает внутренний символ ранга 1, которому сопоставлено отношение лексикографического сравнения наборов. Правила для вычисления значения этого символа задать тривиально. В случае, когда длина набора $\overline{y}$ равна 1, правило одно и тело такого правила имеет вид $x < y$. Ясно, что справедливость $\mathrm{NOT}\, Q(\overline{x},\overline{y},\overline{u})$ означает, что $\overline{u}$ является следующим за $\overline{y}$ в лексикографическом порядке набором в множестве наборов $\overline{z}$, удовлетворяющих $P(\overline{x},\overline{z})$.

Теперь в случае, когда множество наборов $\overline{z}$, удовлетворяющих $P(\overline{x},\overline{z})$, конечно, можно подсчитать число таких наборов.

Предварительно надо иметь сложение натуральных чисел:

$$\mathrm{Sum}(x,0,x) \leftarrow;\ \ \mathrm{Sum}(x,y,z) \leftarrow u =_1 y\ \ \mathrm{Sum}(x,u,v)\ \ v =_1 z.$$

Для подсчета числа элементов группы надо ввести функцию NUMBER$(\overline{x}, \overline{y}, z)$. Для минимального $\overline{y}$ это число z равно 1. Следующее правило имеет вид

$$\text{NUMBER}(\overline{x}, \overline{y}, z) \leftarrow \text{NOT}\, Q(\overline{x}, \overline{u}, \overline{y})\ \ \text{NUMBER}(\overline{x}, \overline{u}, v)\ \ v =_1 z.$$

Ясно, что NUMBER$(\overline{x}, \overline{y}, z)$ для максимального $\overline{y}$ дает z, равное числу элементов группы. Аналогично вычисляется сумма элементов некоторого столбца для группы.

Это заканчивает доказательство утверждения, что после каждого шага вычисления всем запросам приписаны бесквантворные положительные формулы сигнатуры Σ. $\square$

Назовем SQL-запрос и стратифицированную дейталог-программу эквивалентными, если для заданных значений внешних символов они выдают один и тот же результат или одновременно не останавливаются.

Предыдущее доказывает также следующую теорему.

Теорема 3.2. *По каждому* SQL*-запросу можно эффективно построить эквивалентную стратифицированную дейталог-программу. В частности, если каждому внешнему символу приписана бесквантворная положительная формула сигнатуры Σ_0, а заданный* SQL*-запрос при этом останавливается, то ответом является набор бесквантворных положительных формул сигнатуры Σ.*

Обратное утверждение совершенно очевидно и доказывается индукцией по максимальному рангу, используемому в программе.

Теорема 3.3. *Для каждой стратифицированной дейталог-программы можно эффективно построить эквивалентный* SQL*-запрос.*

4. ЛОГИКА НЕПОДВИЖНЫХ ТОЧЕК (FPL)

Мы будем рассматривать формулы сигнатуры $\Sigma_1 = \langle \Sigma, Q_1^{(n_1)}, \ldots, Q_\ell^{(n_\ell)} \rangle$, получаемой обогащением сигнатуры Σ дополнительными n_i-местными символами отношений Q_i для $i = 1, \ldots, \ell$.

Рассматриваем формулы сигнатуры $\langle \Sigma_1, P^{(n)} \rangle$, получаемой обогащением сигнатуры Σ_1 дополнительным n-местным символом отношения P, в которые символ P входит положительно. Напомним, что символ P входит положительно в некоторую формулу, если эта формула имеет вид $P(z_1, \ldots, z_n)$, в которой $z_1, \ldots, z_n$ являются некоторыми переменными, либо имеет вид дизъюнкции двух формул, в каждую из которых символ P входит положительно, либо имеет вид конъюнкции двух формул, в каждую из которых символ P входит положительно, либо эта формула вообще не содержит P.

Мы будем рассматривать алгебраические системы сигнатуры Σ_1, носитель которых есть множество натуральных чисел, а символы Q_i для $i = 1, \ldots, \ell$ интерпретируются бесквантворными положительными формулами сигнатуры Σ_0 и при этом выполнены соглашения из определения конечного представления базы данных. Другими словами, нам заданы схема и состояние некоторой базы данных.

Для этого состояния и формулы $\phi(x_1, \ldots, x_n)$ сигнатуры $\langle \Sigma_1, P^{(n)} \rangle$, в которую P входит положительно, мы определяем формулу $\text{FP}(\phi)(x_1, \ldots, x_n)$. Формулу $\text{FP}(\phi)(x_1, \ldots, x_n)$ мы называем неподвижной точкой формулы $\phi(x_1, \ldots, x_n)$ на рассматриваемом состоянии.

Но в действительности мы будем рассматривать более общий случай. Если интерпретация неподвижной точки формулы $\phi(x_1, \ldots, x_n)$ уже определена и является бесквантворной положительной формулой сигнатуры Σ, то мы расширяем схему базы данных, добавляя символ отношения местности n к схеме Σ_1, и обогащаем состояние, считая интерпретацию этой неподвижной точки интерпретацией добавленного символа. Дальнейшие определения не зависят от того, на каком шаге расширения схемы базы данных мы находимся.

Интерпретация формулы $\mathrm{FP}(\phi)(x_1,\dots,x_n)$ строится следующим образом. Для данного состояния σ базы данных интерпретация $\mathrm{FP}(\phi)(x_1,\dots,x_n)$ есть $\bigcup_{i=1}^{\infty} P_i$, где P_0 пусто и

$$P_i = \bigl\{(a_1,\dots,a_n) \in \omega^n \mid (\sigma, P_{i-1}, a_1,\dots,a_n) \models \phi(x_1,\dots,x_n)\bigr\}.$$

При этом истинность $(\sigma, P_{i-1}, a_1,\dots,a_n) \models \phi(x_1,\dots,x_n)$ означает истинность $\phi(x_1,\dots,x_n)$ в (σ, P_{i-1}) при $x_1 = a_1,\dots,x_n = a_n$. Как объяснено в [19], система (σ, P_{i-1}) является обогащением состояния σ, в котором интерпретацией символа P является отношение P_{i-1}.

По предыдущему каждое P_i задается бесквантворной положительной формулой сигнатуры Σ. Так как $\phi(x_1,\dots,x_n)$ содержит P положительно, то P_{i+1} содержит P_i для каждого i. Таким образом, если $P_{i+1} = P_i$ для какого-то i, то интерпретация формулы $\mathrm{FP}(\phi)(x_1,\dots,x_n)$ на рассматриваемом состоянии и есть P_i для этого i. Если же последовательность этих P_i строго возрастает (что было бы невозможно на конечном носителе), то значение формулы $\mathrm{FP}(\phi)(x_1,\dots,x_n)$ не определено.

Если формула не содержит P, то ее неподвижная точка совпадает с ней самой на каждом состоянии. Поэтому каждую формулу можно мыслить как неподвижную точку некоторой формулы. Так полученные формулы называются *формулами логики неподвижных точек*.

Формулы можно ранжировать следующим образом. Формулы, в которых нет неподвижных точек, имеют ранг 0. Неподвижные точки формул ранга 0 имеют ранг 1. Формулы, в которых встречаются неподвижные точки ранга 1, но нет неподвижных точек большего ранга, имеют ранг 1. Неподвижные точки формул ранга 1 имеют ранг 2. Вообще формулы, в которых встречаются неподвижные точки ранга i, но нет неподвижных точек большего ранга, имеют ранг i. Неподвижные точки формул ранга i имеют ранг $(i+1)$.

Если формула содержит неподвижные точки, то на некоторых состояниях значение этой формулы может быть неопределенным. По предыдущему если значение формулы определено на некотором состоянии, то это значение является бесквантворной положительной формулой сигнатуры Σ.

Имеет место следующая

Теорема 4.1. *Для каждой формулы логики неподвижных точек можно эффективно построить такую дейталоговскую стратифицированную программу, что на каждом состоянии результат работы построенной программы совпадает со значением заданной формулы. При этом построенная программа не останавливается на указанном состоянии тогда и только тогда, когда на этом состоянии значение заданной формулы не определено.*

Доказательство этой теоремы рутинно и проводится индукцией по рангу рассматриваемой формулы.

Обратная теорема более содержательна. Ее доказательство использует отмеченный еще в книге Московакиса (см. [16]) факт, что индукция по нескольким параметрам в некоторых случаях сводится к индукции по одному параметру. Поэтому мы приведем это доказательство. Хотя наше построение не является чем-то совсем оригинальным, но в рассматриваемом случае несопоставимо проще, чем общее рассуждение Московакиса.

Теорема 4.2. *Для каждой дейталоговской стратифицированной программы можно эффективно построить такой набор формул логики неподвижных точек, что на каждом состоянии результат работы построенной программы совпадает со значением этого набора формул. При этом заданная программа не останавливается на указанном состоянии тогда и только тогда, когда на этом состоянии значение построенного набора формул не определено.*

Доказательство. Понятно, что достаточно рассмотреть только случай правил ранга i, предполагая, что для внутренних символов меньшего ранга формулы логики неподвижных точек, задающие значения этих символов, уже построены. Пусть имеется k внутренних символов

ранга i. Пусть m — наибольшее возможное число аргументов у этих символов. Если какой-то из этих символов имеет меньше m аргументов, мы будем предполагать, что аргументов у этого символа ровно m, повторяя последний аргумент нужное число раз. Введем новый символ Q ранга i и местности $(m+1)$.

Пусть Q_j — m_j-местный символ ранга i. Можно предполагать, что голова каждого правила с именем Q_j есть $Q_j(x_1,\dots,x_{m_j})$. Каждое правило с именем Q_j заменяем на новое правило с головой $Q(x_0,x_1,\dots,x_m)$ и телом, которое включает в себя атом $x_0 = j$, все входящие в тело рассматриваемого правила атомы и, кроме того, все входящие в тело рассматриваемого правила выражения вида

- $Q(y_1,\dots,y_k)$, где Q — внешний символ местности k, а $y_1,\dots,y_k$ — последовательность переменных;
- $Q(y_1,\dots,y_k)$, где Q — внутренний символ местности k, ранг которого меньше i, а $y_1,\dots,y_k$ — последовательность переменных;
- $\neg Q(y_1,\dots,y_k)$, где Q — внутренний символ местности k, ранг которого меньше i, а $y_1,\dots,y_k$ — последовательность переменных.

Что касается входящих в тело рассматриваемого правила выражений вида $Q_\ell(z_1,\dots,z_{m_\ell})$, где Q_ℓ — m_ℓ-местный символ ранга i, то вместо этих выражений в тело нового правила включаем последовательности

$$Q(z_0,z_1,\dots,z_{m_\ell},\dots,z_{m_\ell}) \quad z_0 = \ell.$$

Кроме того, если $m > m_j$, в тело нового правила добавляем последовательность

$$x_{m_{j+1}} = x_{m_j} \quad \dots \quad x_m = x_{m_j}.$$

После этих преобразований у нас для каждого ранга имеется ровно один внешний символ этого ранга.

Далее мы тело каждого правила заменяем на конъюнкцию всех элементов этого тела, а затем рассматриваем дизъюнкцию так полученных конъюнкций для всех правил с именем Q, на которую навешиваем кванторы существования по всем переменным, не входящим в голову правила. Так мы получаем формулу, неподвижная точка которой и дает значение нового символа Q ранга i. Для получения значения старого символа Q_j достаточно добавить конъюнкцию $x_0 = j$ и, если $m > m_j$, еще $(x_{m_{j+1}} = x_{m_j} \,\&\, \dots \,\&\, x_m = x_{m_j})$. $\square$

Благодарности. Я благодарен Льву Дмитриевичу Беклемишеву за замечания, которые способствовали улучшению первоначального текста статьи, в частности первоначального доказательства теоремы 2.3.

СПИСОК ЛИТЕРАТУРЫ

1. *Abiteboul S., Hull R., Vianu V.* Foundations of databases. Reading, MA: Addison-Wesley, 1995.
2. *Abiteboul S., Vardi M.Y., Vianu V.* Fixpoint logics, relational machines, and computational complexity // Structure in complexity theory: Proc. 7th Annu. Conf., Boston, 1992. Los Alamitos, CA: IEEE Comput. Soc. Press, 1992. P. 156–168.
3. *Abiteboul S., Vianu V.* Fixpoint extensions of first-order logic and Datalog-like languages // Logic in computer science: Proc. 4th Annu. Symp., Pacific Grove, CA (USA), 1989. Washington, DC: IEEE Comput. Soc. Press, 1989. P. 71–79.
4. *Afrati F., Cosmadakis S.S., Yannakakis M.* On Datalog vs. polynomial time // Principles of database systems: Proc. 10th ACM SIGACT–SIGMOD–SOGART Symp., Denver, 1991. New York: Assoc. Comput. Mach., 1991. P. 13–25.
5. *Ajtai M., Gurevich Y.* DATALOG vs. first-order logic // Foundations of computer science: Proc. 30th Annu. Symp. Los Alamitos, CA: IEEE Comput. Soc. Press, 1989. P. 142–146.

6. *Belegradek O.V., Stolboushkin A.P., Taitslin M.A.* On problems of databases over a fixed infinite universe // Logic, algebra, and computer science: Helena Rasiowa in memoriam. Warszawa: Pol. Acad. Sci., Inst. Math., 1999. P. 23–62. (Banach Center Publ.; V. 46).
7. *Blass A., Gurevich Y.* Existential fixed-point logic // Computation theory and logic / Ed. by E. Börger. Berlin: Springer, 1987. P. 20–36. (Lect. Notes Comput. Sci.; V. 270).
8. *Chandra A., Harel D.* Structure and complexity of relational queries // J. Comput. and Syst. Sci. 1982. V. 25. P. 99–128.
9. *Codd E.F.* A relational model of data for large shared data banks // Commun. ACM. 1970. V. 13. P. 377–387.
10. *Codd E.F.* Relational completeness of data base sublanguages // Data base systems / Ed. by R. Rustin. Englewood Cliffs, NJ: Prentice-Hall, 1972. P. 65–98.
11. *Immerman N.* Relational queries computable in polynomial time // Inf. and Control. 1986. V. 68. P. 86–104.
12. *Immerman N.* Languages that capture complexity classes // SIAM J. Comput. 1987. V. 16. P. 760–778.
13. *Kanellakis P.C., Goldin D.Q.* Constraint programming and database query languages // Theoretical aspects of computer software: Proc. 2nd Intern. Symp. TACS'94. Berlin: Springer, 1994. P. 96–120. (Lect. Notes Comput. Sci.; V. 789).
14. *Kanellakis P.C., Kuper G.M., Revesz P.Z.* Constraint query languages // J. Comput. and Syst. Sci. 1995. V. 51, N 1. P. 26–52.
15. *Libkin L.* Elements of finite model theory. Berlin: Springer, 2004.
16. *Moschovakis Y.N.* Elementary induction on abstract structures. Amsterdam: North-Holland, 1974.
17. *Stolboushkin A.P., Taitslin M.A.* Safe stratified Datalog with integer order does not have syntax // ACM Trans. Database Syst. 1998. V. 23, N 1. P. 100–109.
18. *Дейт К.Дж.* Введение в системы баз данных. 8-е изд. М.: Вильямс, 2005.
19. *Дудаков С.М., Тайцлин М.А.* Трансляционные результаты для языков запросов в теории баз данных // УМН. 2006. Т. 61, № 2. С. 3–66.
20. *Клайн К.* SQL. Справочник. М.: Кудиц-Образ, 2006.
21. *Уилтон П., Колби Дж.* SQL для начинающих. М.: Диалектика, 2006.
22. *Форта Б.* Освой самостоятельно SQL. 10 минут на урок. 3-е изд. М.: Вильямс, 2005.
23. *Дудаков С.М.* SQL. http://homepages.tversu.ru/~p000101/dudakov.pdf