Юрий Нечитайлов[1]

ДЕСКРИПЦИОННЫЕ ЛОГИКИ КАК ФУНДАМЕНТ ДЛЯ РАЗРЕШЕНИЯ РЯДА АЛГОРИТМИЧЕСКИХ ВОПРОСОВ[2]

Аннотация. Проанализированы отличие и основные свойства баз даных и баз знаний как основных площадок для применения средств логического программирования. Приведено описание семейства дискрипционных логик, даны основные определения. Описана проблема баланса выразительной силы языка и вычислительной эффективности, приведены инструменты управления этими параметрами и описаны их сильные и слабые стороны. Предложены перечень и описание нестандартных алгоритмических проблем в системам логического программирования для их использования в качестве содержательной основы при формализации вопроса в рамках одной из логических систем.

Ключевые слова: адаптивная логика, обучающиеся системы, дескрипционная логика.

Iurii Nechitailov

DESCRIPTION LOGICS AS A BASIS FOR SOLVING OF A SET OF ALGORITHMIC PROBLEMS

Abstract. Some distinctions and properties of databases and knowledge bases as major platforms for logic programming are analysed. A review of the family of Description logics and its main definitions is given. A problem of balance between the expressive power of a language and its computational complexity is described. Some instruments to control these parameters are proposed. A list and description of non-standard algorithmic problems encountered in the logic programming is considered as a conceptual basis for the formalization of questions in the framework of a logical system.

Keywords: adaptive logic, learning systems, description logics.

Введение

Выбор теоретического фундамента для построения логики вопросов во многом зависит от исходных установок. Для дальнейшей формализации мы постараемся подобрать те типы вопросов, которые задаются относительно логических систем и которые представляют практический интерес. Пригодность к практической реализации обязывает, в погоне за выразительной силой языка, обратить внимание на вычислительную эффективность проектируемой системы. Кроме того, поскольку большинство практических реализаций логических систем основано на работе с базами знаний, вполне естественно было бы отвести вопросу

[1] *Нечитайлов Юрий Вячеславович*, кандидат философских наук, доцент кафедры логики Санкт-Петербургского государственного университета.

Iurii Nechitailov, PhD, docent, department of logic, Saint Petersburg State University.

[2] Работа выполнена при поддержке РГНФ, грант № 15-03-00166.

(как элементу логической системы) роль запроса к подобной базе знаний. Для начала же стоит обратить внимание на то, чем отличаются базы знаний от баз данных и каковы их основные черты.

1. Базы данных и базы знаний

В процессе автоматизации решений блока финансово-экономических и военных задач в начале 1960-х годов появились первые технологии, связанные с базами данных и основанные в то время на языке COBOL. Сам термин *база данных* (англ. database), как считается, вошёл в обиход на одном из симпозиумов компании System Development Corporation (SDC), посвящённых этой проблематике, состоявшемся в 1963 году в Санта-Монике (шт. Калифорния), где в том числе были представлены ранние программные системы — прообразы современных СУБД.

За время, прошедшее с тех пор, возможно в силу своего прикладного характера, этот термин получил ряд описательных определений, из которых не очень просто выявить, какие из признаков являются необходимыми и достаточными, позволяющими отличать базы данных от чего-либо ещё. Ситуацию усугубило появление другого, близкого по значению термина — *база знаний*. Он возник в ходе развития исследований в области интеллектуальных систем и искусственного интеллекта.

Тем не менее, попробуем обобщить сведения, доступные на данный момент. Проведённый анализ показал, что во многих ситуациях эти два термина понимаются как род и вид, т. е. база знаний — это особый вид базы данных.

Среди существенных признаков, характеризующих *базы данных*, выделяют следующие: связанность (организованность в блоки), структурированность, соотнесённость с предметной областью, наличие общих принципов описания, хранения и манипулирования, независимых от прикладных программ.

В дополнение к указанным признакам, *базы знаний* наделяют следующими: внутренняя интерпретируемость (выразимость одних знаний через другие, к примеру через задание отношений эквивалентности и включённости), связность с некоторой системой знаний (набором понятий более высокого уровня) и представленность самой системы знаний в этой же базе, динамичность (предоставление возможности роста базы знаний даже в момент обращения к ней, т. е. запись в базу знаний одним пользователем может происходить одновременно с получением информации другим), активность (новые знания могут приводить к изменению старых знаний или их временной блокировке; в первом случае речь идёт не о перезаписи, а о ревизии знаний).

Резюмируя, с точки зрения логики можно выделить следующие существенные различия между базами данных, не являющимися базами знаний, и собственно базами знаний. Во-первых, в базе знаний в явном виде могут храниться элементы логического аппарата для работы с ней. Во-вторых, база знаний основывается на моделях, подразумевающих «открытость мира», т. е. если она не содержит некоторое положение, то это не будет обязывать к принятию его отрицания. В-третьих, база знаний продолжит функционировать даже в случае возникновения неконсистентности.

Наблюдая за современными тенденциями развития баз данных, следует отметить, что колоссальный рост объёмов обрабатываемых данных, связанные с этим потребности децентрализации, распределённого хранения, шифрования, а также необходимость быстрого отклика привели к тому, что пала «священная корова» отцов-основателей первых систем управления баз данных — требование к их консистентности. В результате потребность в разработке моделей взаимодействия с такого рода данными оказалась актуальной, как никогда ранее. Для решения данной задачи имеет смысл обратиться, во-первых, к дескрипционным логикам — как основе практической реализации методов представления и обработки знаний, а во-вторых, к адаптивным логикам — как основе для манипулирования неконсистентными данными, для которых отсутствует количественная оценка степени правдоподобия.

2. Дескрипционные логики

Дескрипционные логики представляют собой семейство взаимосвязанных языков, каждый из которых имеет собственные синтаксис и семантику.

2.1. Основные термины

Базовыми синтаксическими элементами языка дескрипционной логики являются атомарные *концепт* и *роль*, соответствующие одноместному и двухместному предикатам языка математической логики. Концепты применяются для описания классов или свойств, роли — для описания отношений между концептами. Концепты и роли позволяют формировать *положения* (англ. statements). В теориях, описывающих базы знаний, различаются общие знания о понятиях и их взаимосвязях (интенсиональные знания), которые выражаются с помощью утверждений общего вида — *терминологий* (англ. terminology), или *аксиом*, а также знания об индивидуальных объектах, их свойствах и связях с другими объектами (экстенсиональные знания) — *утверждения* (англ. assertions) *об индивидах*. Такое различение связано с тем, что интенсиональные знания на практике реже изменяются, чем экстенсиональные. Опираясь на это, выделяются набор терминологических аксиом, называемый TBox, и набор утверждений об индивидах — Abox. Вместе они образуют *базу знаний*, или *онтологию*. Таким образом, в дескрипционной логике термин «база знаний» может использоваться как внутренний, с приданием ему соответствующей смысловой нагрузки. Во избежание разночтения, для указания на множество положений, составляющих TBox и ABox, будем пользоваться термином «онтология», а за термином «база знаний» сохраним общий смысл, о котором шла речь в предыдущем разделе. В ходе применения дескрипционной логики на практике онтологии могут связываться с различными предметными областями.

2.2. Базовая дескрипционная логика $\mathcal{ALC}$

Как и в случае традиционных (общетеоретических) логик, дескрипционные логики задаются через описание синтаксиса и семантики языка. Забегая вперёд, скажем, что язык *базовой дескрипционной логики* ($\mathcal{ALC}$) является разрешимым фрагментом первопорядковой логики.

2.2.1. Синтаксис $\mathcal{ALC}$

В качестве сигнатуры языка дескрипционной логики берётся кортеж $\langle \mathbb{N}_C, \mathbb{N}_R, \mathbb{N}_I \rangle$ не более чем счётных множеств атомарных концептов, ролей и индивидов (имён) соответственно. В дескрипционных логиках одним из аналогов высказывания (правильно построенной формулы, **ППФ**) является *концепт*. Как будет видно из определения, он соответствует не всем формам первопорядковых высказываний. Итак, для атомарного концепта $A \in \mathbb{N}_C$ и атомарной роли $R \in \mathbb{N}_R$ концепт C (ППФ) в форме Бэкуса-Наура (БНФ) задаётся следующим образом:

$$C := A \mid \top \mid \bot \mid (\neg C) \mid (C \sqcap C) \mid (C \sqcup C) \mid (\forall R.C) \mid (\exists R.C).$$

Символы $\neg$, $\sqcap$, $\sqcup$, $\forall$, и $\exists$ называются конструкторами. Языки дескрипционных логик различаются тем, какие конструкторы они позволяют использовать. Так, язык $\mathcal{ALC}$ позволяет использовать *отрицание* ($\neg$), *конъюнкцию* ($\sqcap$), *дизъюнкцию* ($\sqcup$), *ограничение на значения роли* ($\forall$) и *экзистенциальное ограничение* ($\exists$).

Концепты, в свою очередь, используются для построения набора *терминологий*, формирующих TBox и представляющих знание о терминах, их взаимосвязи и т. п. При этом для построения терминологий вводятся дополнительные бинарные отношения *вложенности* ($\sqsubseteq$) и *эквивалентности* ($\equiv$). Они используются в качестве внешних знаков терминологий и позволяют сформулировать, соответственно, аксиомы вложенности и аксиомы эквивалентности концептов. Терминологии являются ещё одним аналогом высказывания в дескрипционных логиках. Формально с помощью БНФ терминология t для концепта C определяется как

$$t := (C \sqsubseteq C) \mid (C \equiv C).$$

В некоторых дескрипционных логиках подобные положения формулируются не только относительно концептов, но и относительно ролей, образуя аксиомы вложенности и аксиомы эквивалентности ролей. Иногда в таких случаях аксиомы ролей выделяют в отдельный RBox.

Помимо этого, концепты используются для построения набора утверждений, формирующих (ABox) и описывающих знание о некотором предметном мире — об *индивидах*. Положения, образующие набор *утверждений* из ABox, если говорить в терминах математической логики, не содержат переменных, замещая их подстановками индивидных констант. В рамках дескрипционной логики это означает, что утверждения из ABox задаются относительно множества индивидов. Для формирования утверждений из ABox могут добавляться два бинарных отношения: *равенства индивидов* и *неравенства индивидов*. Для концепта C, роли $R \in \mathbb{N}_R$ и индивидов $a, b \in \mathbb{N}_I$ утверждением s из ABox будет:

$$s := C(a) \mid R(a, b) \mid a = b \mid a \neq b.$$

В записи $R(a, b)$, b называют R-последователем a.

2.2.2. Семантика $\mathcal{ALC}$

Семантика задаётся путём интерпретации индивидов, атомарных концептов и ролей. Формально интерпретация представляется парой $I = \langle \Delta^{\mathcal{I}}, .^{\mathcal{I}} \rangle$, где $\Delta^{\mathcal{I}}$ — непустое множество, называемое доменом, а $.^{\mathcal{I}}$— особая функция, называемая функцией интерпретации. Функция интерпретации сопоставляет каждому из атомарных концептов подмножество элементов домена, каждой атомарной роли — подмножество пар из указанного домена, а каждому индивиду — элемент домена. В формальной записи:

- если $A \subseteq N_C$, то $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$;
- если $R \subseteq N_R$, то $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$;
- если $a \in N_I$, то $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

Для сложных концептов интерпретация определяется по индукции:

- $\top$ интерпретируется как весь домен: $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$;
- $\bot$ интерпретируется как пустой домен: $\bot^{\mathcal{I}} = \varnothing$;
- отрицание концепта интерпретируется как дополнение множества: $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus (C)^{\mathcal{I}}$;
- конъюнкция концептов интерпретируется как пересечение множеств: $(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$;
- дизъюнкция концептов интерпретируется как объединение множеств: $(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$;
- ограничение на значения роли $(\forall R.C)$ интерпретируется как множество тех индивидов, у которых все R-последователи принадлежат интерпретации концепта C. Формально: $(\forall R.C)^{\mathcal{I}} = \{\alpha \in \Delta^{\mathcal{I}} \mid \forall \beta \in \Delta^{\mathcal{I}}[(\alpha, \beta) \in R^{\mathcal{I}} \Rightarrow \beta \in C^{\mathcal{I}}]\}$;
- экзистенциальное ограничение $(\exists R.C)$ интерпретируется как множество тех индивидов, у которых все R-последователи принадлежат интерпретации концепта C. Формально: $(\exists R.C)^{\mathcal{I}} = \{\alpha \in \Delta^{\mathcal{I}} \mid \exists \beta \in \Delta^{\mathcal{I}}[(\alpha, \beta) \in R^{\mathcal{I}} \;\&\; \beta \in C^{\mathcal{I}}]\}$.

2.3. Инструменты построения вывода

Логики сразу обратят внимание, что описание синтаксиса дескрипционных логик не содержит правил вывода. Данный факт может быть обусловлен двумя причинами. Во-первых, дескрипционные логики являются развитием идей, сформулированных в рамках фреймовых структур и семантических сетей. Основной практической задачей на начальном этапе была классификация, а не построение логического вывода. Во-вторых, на практике логический вывод даже на верхнем уровне реализуется не с помощью правил, как их привыкли

видеть логики, а с помощью алгоритмов, которые можно сопоставить с определёнными правилами. Так, например, обстоит дело и в случае такого известного языка логического программирования, как Пролог. При этом некоторые положения, лежащие в основе алгоритмов, имеют прямые аналоги в общетеоретических логических системах (как, например, правило резолюций и закон выявления), а другие — нет (как, например, правило унификации).

В дескрипционных логиках для построения вывода используются различные алгоритмы. Каждый из алгоритмов, что вполне естественно, будет давать свой набор результатов. Целесообразность выбора определённого алгоритма обуславливается, как выразительными характеристиками реализуемого языка, так и готовностью к неполноте ответов. Среди алгоритмов для построения выводов в дескрипционных логиках можно выделить алгоритмы, использующие структурную категоризацию (через сравнение синтаксической структуры заданных концептов), и алгоритмы, использующие табличные методы (к примеру, в работе Brachman *et al.* 1991 описан такой алгоритм для дескрипционной системы CLASSIC, основанной на фреймовых структурах).

2.4. Расширения $\mathcal{ALC}$

За прошедшие три десятилетия предложен ряд расширений базовой дескрипционной логики. Эти расширения были в определённой степени структурированы и достаточно подробно изучены. Среди прочего они связаны с добавлением свойств транзитивности ролей, иерархии ролей, обратимости ролей, функциональности ролей, ограничением кардинальности ролей, качественным ограничением кардинальности ролей и т. п.

К примеру, в расширении, именуемом $\mathcal{S}$ и позволяющим определение *транзитивных ролей*, в исходном атомарном множестве ролей $\mathbb{N}_R$ выделяется подмножество транзитивно замкнутых ролей $\mathbb{N}_{R+} \subseteq \mathbb{N}_R$, так что для роли $R \in \mathbb{N}_{R+}$, элементов домена α, β, $\gamma \in \Delta^{\mathcal{I}}$ и интерпретации $\mathcal{I}$ будет иметь место:

$$\text{если } (\alpha, \beta) \in R^{\mathcal{I}}, \text{ и } (\beta, \gamma) \in R^{\mathcal{I}}, \text{ то } (\alpha, \gamma) \in R^{\mathcal{I}}.$$

Остальные расширения именуются добавлением идентифицирующих их букв либо к $\mathcal{S}$, либо к $\mathcal{ALC}$ — к последнему, если в них не подразумевается задание свойства транзитивности ролей.

Дескрипционные логики, содержащие в имени $\mathcal{H}$, допускают задание *иерархии ролей*. Они, помимо TBox и ABox, содержат RBox, в котором формулируется набор терминологий с описанным ранее отношением вложенности.

Дескрипционные логики, содержащие в имени $\mathcal{I}$, допускают задание *обратимых ролей*.

Интересным расширением являются логики, содержащие в имени $\mathcal{O}$ и допускающие использование *номиналов*. В этих логиках допускается задание концептов через перечисление составляющих их индивидов (имён). Например,

$$C = \{a_1, \ldots, a_n\} \text{ для } a_1, \ldots, a_n \in \mathbb{N}_I.$$

В таком случае интерпретация будет вполне очевидна:

$$C^{\mathcal{I}} = \{a_1, \ldots, a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\} \subseteq \Delta^{\mathcal{I}}.$$

Дескрипционные логики, содержащие в имени $\mathcal{Q}$, позволяют задавать *качественное ограничение кардинальности ролей*. Такие логики, для любой простой роли R, концепта C и натурального числа n, позволяют определять концепты $\leqslant nR.C$, для которых существует не более n R-последователей в C или, аналогично не менее n R-последователей в C. Используя символ $\#$ для указания кардинальности, это можно определить формально:

$$(\leqslant R.C)^{\mathcal{I}} = \{a \mid \#\{(a,b) \in R^{\mathcal{I}} \ \&\ b \in C^{\mathcal{I}}\} \leqslant n\},$$

$$(\geqslant R.C)^{\mathcal{I}} = \{a \mid \#\{(a,b) \in R^{\mathcal{I}} \ \&\ b \in C^{\mathcal{I}}\} \geqslant n\}.$$

Производными дескрипционными логиками являются логики, содержащие в имени букву $\mathcal{N}$, и позволяющие *учёт кардинальности безотносительно «качества» R-последователей*. Ещё более частные логики, обозначающиеся буквой $\mathcal{F}$, позволяют описывать так называемую *функциональность ролей*, — когда имеется не более одного R-последователя.

Наконец, некоторые дескрипционные логики позволяют различать типы данных и задавать конкретные роли. В своём имени они содержат следующую комбинацию из трёх символов ($\mathcal{D}$). Для реализации этого задаются дополнительный домен $\Delta_D^{\mathcal{I}}$, непересекающиеся с основным $\Delta^{\mathcal{I}}$. Конкретные роли R_D задаются на $\Delta^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$.

3. Место дескрипционных логик в системе традиционных

Для логиков дескрипционные логики выглядят несколько необычно. Потому для привлечения их внимания немаловажно сопоставить хотя бы некоторые из дескрипционных логик с привычными (традиционными) логическими системами — пропозициональными, первопорядковыми, модальными и т. п.

3.1. Сопоставление с модальной логикой

Языки некоторых дескрипционных логик можно сопоставить с языками модальных логик. Этот факт был обнаружен Шилдом (Schild 1991) в 1991 году, который свёл базовую дескрипционную логику $\mathcal{ALC}$ к мультимодальной логике $\mathbf{K_n}$. При этом для атомарных концептов $A_1, \ldots, A_m$ и ролей $R_1, \ldots, R_n$ выполняется следующее сопоставление. Атомарные концепты переводятся в пропозициональные переменные $p_1, \ldots, p_m$. Отрицание ($\neg$), конъюнкция ($\sqcap$) и дизъюнкция ($\sqcup$) дескрипционной логики однозначно переводятся к аналогичным связкам ($\neg$, $\wedge$ и $\vee$) модальной логики. Формально, если f_T — функция, переводящая выражение языка дескрипционной логики на язык модальной, а C и D — концепты, то вышесказанное будет записано в виде:

$$\begin{array}{lcl} f_T(A_i) & = & p_i \\ f_T(\neg C) & = & \neg(f_T(C)) \\ f_T(C \sqcap D) & = & f_T(C) \wedge f_T(D) \\ f_T(C \sqcup D) & = & f_T(C) \vee f_T(D) \end{array}$$

К этому следует добавить перевод в модальности квантифицированных выражений для ролей, выполненный по аналогии с тем, как это делается для перевода из фрагмента первопорядковой логики (см., например, van Benthem 1984).

$$\begin{array}{lcl} f_T(\forall R_i.C) & = & \Box_i(f_T(C)) \\ f_T(\exists R_i.C) & = & \Diamond_i(f_T(C)) \end{array}$$

Проблемной точкой перевода на язык семантики возможных миров может показаться, в частности, независимость означивания атомарных пропозиций в различных мирах и, как следствие, определение тождественности атомарных пропозиций при переходе между мирами. Однако проблемы с этим нет. Каждый из атомарных концептов означивается на домене $\Delta^{\mathcal{I}}$. При этом элементы домена и рассматриваются в качестве точек соотнесения (т. е. значения истинности атомарных концептов $A_1, \ldots, A_m$ на каком-то элементе домена $d \in \Delta^{\mathcal{I}}$ будут соответствовать значениям истинности $p_1, \ldots, p_m$ в возможном мире $d \in W$). Таким образом, изменение A_i при переходе от одного элемента домена к другому будет соответствовать изменению p_i при переходе от одного мира к другому, а p_i в различных мирах будут рассматриваться как тождественные (те же самые, но принимающие различные значения). Заметим, что данная концепция тождественности отличается от аристотелевской, подразумевающей неизменность значения. Однако без подобного изменения в определении тождественности, семантика возможных миров оказалась бы не оправданной (всё равно что утверждение, что я вчера и я сегодня никак не связаны).

В 1994 году было показано (De Giacomo, Lenzerini 1994; Schild 1994), что расширение языка $\mathcal{ALC}$, позволяющее использовать регулярные выражения[3] для описания ролей, — $\mathcal{ALC}_{reg}$ — сопоставимо с пропозициональной динамической логикой.

3.2. Сопоставление с первопорядковой логикой

В 2001 году было показано (Lutz *et al.* 2001), что $\mathcal{ALC}$ сводима ещё и к фрагменту логики предикатов первого порядка — логике с двумя переменными (который является разрешимым).

Если же не ставить ограничение на количество переменных, то перевод будет вполне очевидным. Бесквантерные выражения переводятся в лоб. Для кванторных перевод будет следующим.

$$\begin{array}{lcl} f_T(\forall R_i.C) & = & \forall x(R_i(x,y) \supset f_T(C)(y)) \\ f_T(\exists R_i.C) & = & \exists x(R_i(x,y) \wedge f_T(C)(y)) \end{array}$$

[3]Регулярные выражения для ролей строятся с использованием бинарных операций объединения ($\sqcup$), композиции ($\circ$) и унарной звезды Клини ($*$).

Здесь x — переменная, не встречавшаяся ранее.

3.3. Эпистемическая интерпретация

Поскольку язык некоторых дескрипционных логик может быть переведён на язык модальных, такие дескрипционные логики получают средства для выражения модальностей и оперирования ими. Поскольку вдобавок они содержат инструменты первопорядковой логики, с их помощью оказывается возможным, в том числе, различать модальности de dicto (приписывание модальных характеристик высказыванию) и de re (приписывание модальных характеристик индивиду). Первые обуславливают истинность высказывания (дают понять, что оно истинно не всегда, везде и во всех отношениях, а только в рамках определённых контекстов), вторые соотносят ряд индивидов с тем(и), что ограничен(ы) модальностью (ограниченный модальностью индивид становится точкой соотнесения). Подробности можно найти, к примеру, в работе Баадера, Кюстерса и Волтера (Baader *et al.* 2010).

Указанный перевод позволяет представить на языке дескрипционных логик не только отношение достижимости и его свойства, но и возможные миры. В рамках TBox дескрипционная логика позволяет описывать характеристики ролей посредством соответствующих аксиом и, следовательно, накладывать условия на связываемые с ними отношения достижимости. А поскольку свойства модальных операторов определяются условиями, накладываемыми на отношение достижимости, характеристики моделируемой модальной системы можно задавать с помощью дескрипционной логики. В частности, дескрипционная логика позволяет задать модальность знания, приписав соответствующим ролям в рамках TBox свойства рефлексивности, транзитивности и эвклидовости. Для множества агентов $\mathbb{A} = \{1, \ldots, n\}$, ролей $\{K_i\}_{i \in \mathbb{A}} \in \mathbb{N}_R$ и пропозиции C это будет записано в форме аксиом:

$$((\forall K_i.C) \sqsubseteq C)$$

$$((\forall K_i.C) \sqsubseteq (\forall K_i.(\forall K_i.C)))$$

$$(\neg(\forall K_i.C) \sqsubseteq (\forall K_i.\neg(\forall K_i.C)))$$

Кроме того, оказывается, что дескрипционные логики позволяют задать отношение предпорядка между мирами, которое используется в эпистемо-доксастической логике (см. определение эпистемо-доксастической логики с использованием отношения предпорядка, например, в van Benthem, Smets (2015), в частности, чтобы в рамках одной модели определить сразу две сильные модальности: и знание, и убеждение. Достичь этого можно за счёт описания специально выделенных для этой цели ролей в рамках ABox, моделирующих предпорядок на элементах домена (и, соответственно, на ассоциируемых с ними возможных мирах). Для множества агентов $\mathbb{A} = \{1, \ldots, n\}$, ролей $\{O_i\}_{i \in \mathbb{A}} \in \mathbb{N}_R$ и индивидов $a_1, a_2, a_3 \in \mathbb{N}_I$ это может быть записано, к примеру, в форме множества утверждений:

$$O_7(a_1, a_1), O_7(a_2, a_2), O_7(a_3, a_3), O_7(a_1, a_3), O_7(a_2, a_3),$$

$$O_5(a_1, a_1), O_5(a_2, a_2), O_5(a_3, a_3), O_5(a_1, a_2), O_5(a_3, a_2).$$

С помощью приведённого множества утверждений ABox для трёх возможных миров a_1, a_2, a_3 и агентов 7 и 5 можно описать их предпочтения относительно сведений, содержащихся в данных мирах. В частности, представленное описание можно трактовать как то, что агент 7 считает мир 3 не менее достоверным[4], чем остальные, в то время как агент 5 считает таковым мир 2.

На заре построения теория пересмотра убеждений, в середине 1980-х, был сформулирован ряд принципов включения новых данных в систему существующих, известный как AGM-подход, — названный в честь авторов, внесших наибольший вклад в его создание (C. Alchourrón, P. Gärdenfors и D. Makinson, см. Alchourrón *et al.* 1985). В этом подходе различаются три способа включения новых данных: расширение, сужение и пересмотр. Как выяснилось в ходе развития динамических эпистемических логик, ряд положений, сформулированных в рамках AGM-подхода, накладывают некоторые ограничения на область их применения. В том числе это затронуло и некоторые построения, описывающие развитие эпистемических ситуаций, и привело к тому, что от ряда принципов AGM-подхода (AGM-принципов) пришлось отказаться. Одной из важных сторон, оказывающих влияние на выполнение или невыполнение AGM-принципов, является характер «поведения» предметной области. Благодаря тому, что семантика дескрипционных логик строится на базе домена $\Delta^{\mathcal{I}}$, оказывается возможным смоделировать различные варианты поведения предметной области. Во-первых, как крайний вариант, каждую из статических моделей, которые описывают эпистемические ситуации между динамическими переходами (обновлениями), можно связать с собственным доменом. Во-вторых, как другой крайний вариант, каждую из статических моделей можно связать с одним и тем же доменом. И, наконец, как промежуточный вариант, последовательные статические модели можно связать с последовательностью монотонно расширяющихся доменов. От этого будет зависеть, в том числе, выполнимость некоторых аксиом. Рассмотрим, к примеру, аксиомы $\neg K_7(\text{Инопланетянин} \equiv \bot)$ и $(K_7\neg(\text{Инопланетянин})) \equiv \top)$, первая из которых означает, что агент 7 не знает, что инопланетяне не существуют, а вторая — что ни один из тех, кого знает агент 7, не является инопланетянином. Эти две формулы выполнимы при построении моделей на основе расширяющихся доменов, но невыполнимы при построении моделей на основе одного домена.

Благодаря осуществлённому за прошедшие годы подобному переводу языка дескрипционных логик, появилась возможность перенести в них соответствующие результаты традиционных логик. На основе дескрипционных логик даже была построена теория вывода в секвенциальном и натуральном форматах (см. Rademaker 2012). Однако с точки зрения изучения возможностей практической реализации идей традиционных систем требуется рассмотреть вопрос о глубине возможного перевода в обратную сторону — выразительных средств традиционных систем на язык дескрипционных логик. Эта тема будет рассматривать-

[4]Допустимы и другие интерпретации, например что агент считает его более достоверным или, наоборот, менее достоверным.

ся в следующем разделе. Кроме того, в следующем разделе рассматривается, какие дополнительные возможности, с точки зрения выразительности языка, открываются благодаря использованию дескрипционных логик.

4. Дескрипционные логики как средство практической реализации традиционных систем

Преимущество дескрипционных логик обусловлено одной из причин их появления — потребностью в создании эффективного прикладного логического аппарата. Видя определённый успех в теоретическом и прикладном развитии дескрипционных логик, появившихся лишь в конце 70-х, имеет смысл оценить различные варианты использования их аппарата для развития и реализации традиционных логических систем.

4.1. Проблема баланса

На практике работа с онтологиями обычно реализуется с помощью компьютеров. В связи с этим, в отличие от общетеоретических логических систем (гильбертовского, генценовского типов и т. п.), в прикладных (на основе которых, к примеру, может быть задан язык Пролог) возникает ряд дополнительных задач, связанных с описанием организации доступа к данным, а также алгоритмов и стратегий их обработки.

Набор положений, составляющих онтологию, может быть слишком большим, данные могут храниться в какой-то особой кодировке, могут быть и другие факторы, затрудняющие их быстрое восприятие пользователем. Доступ к этим положениям осуществляется посредством организации специального интерфейса. С точки зрения аппарата прикладной логической системы это означает, что он должен содержать некий инструмент построения *запроса* к базе знаний.

Простейшими вариантами запроса являются запросы на детерминированный поиск и на проверку/распознание (к примеру, на выявление включённости положения в число исходных положений), более сложными — запросы на недетерминированный поиск и построение (к примеру, на решение вопроса об условиях выводимости из исходных положений).

Определение видов запросов является лишь одной из дополнительных задач, возникающих при задании аппарата прикладной логики. Определение путей оптимизации при поиске ответа на запрос является не менее важной и сложной задачей. Если бы не существовало никаких требований или ограничений, в том числе требований или ограничений по скорости вычислений, по объёму пространства, используемого для вычислений, которые определяют то, что называется *вычислительной сложностью*, то можно было бы позволить ставить сколь угодно сложные вопросы к онтологии. В некоторых случаях, возможно, пришлось бы подождать семь с половиной миллионов лет, как при ответе на «Величайший вопрос Жизни, Вселенной и Всего Такого», сформулированный сверхразумной расой существ, созданному ими компьютеру — Думателю (Deep

Thought) — второму по производительности за всё существование времени и Вселенной, или построить ещё более великий компьютер, включающий в себя живых существ как часть вычислительной системы, названный Земля и в силу своих огромных размеров воспринимавшийся некоторыми по ошибке как планета, для того чтобы узнать в чём собственно состоит этот Величайший вопрос (см.: Дуглас Адамс, «Путеводитель по галактике для путешествующих автостопом»). Однако и мы сами, и природа ставим ограничения на вычислительную сложность, которые вытекают в ряд требований, в том числе и в сфере прикладной логики. В сфере прикладной логики эти требования выражаются в форме алгоритмов и стратегий, которые мы используем для *поиска решения* на запрос и *оптимизации поиска*.

Во Вселенной мало что даётся даром, это относится и к оптимизации вычислений. Если мы пожелаем серьёзно «сэкономить» на времени вычислений или задействованном пространстве, это неизбежно скажется на гарантиях, связанных с полнотой (оценкой всех вариантов), качеством (выбором лучшего варианта) и правильностью (соответствию всем контекстам) формируемых ответов — факторами, относящимися к *выразительной силе* языка и его дескриптивным возможностям. Речь идёт именно о гарантиях полноты, качества и правильности, а не о полноте, качестве и правильности. Разница в том, что в некоторых частных случаях определённая «ограниченная» схема вычислений может давать полный, качественный и правильный результат, в то время как в других случаях она же может не давать никакого результата вовсе, — в таком случае говорить о гарантиях нельзя.

В логике подобная оппозиция, напоминающая ситуацию во вселенной «Звёздных войн», известна как «Золотое правило»: сохраняется неизменный баланс между выразительной силой языка и вычислительной сложностью рассуждений, выполняемых на его основе (см., например: Й. ван Бентем, «Модальная логика для открытых умов» (van Benthem 2010), с. 25), — чем выше выразительная сила, тем выше вычислительная сложность. К примеру, логика предикатов первого порядка является более выразительной, нежели пропозициональная модальная логика: с помощью первой можно различить все модели, описанные с помощью второй, но не наоборот. Вычислительная сложность первопорядковой логики также выше — она неразрешима, в отличие от пропозициональной модальной логики (для некоторых высказываний, записанных на языке первопорядковой логики, не существует алгоритма, определяющего принадлежность этих высказываний теории).

Дескрипционные логики, являясь прикладным теоретическим инструментом для задач представления и обработки знаний, как раз нацелены на поиск оптимального баланса вычислительных возможностей (разрешимости) и выразительной силы языка. Вопрос практической реализации связанных с этим идей выходит за рамки дескрипционных логик (как и данной статьи) и в последние годы относится прежде всего к сфере *Языков сетевых онтологий* (Web Ontology Languages, **OWL**), которые, в более подходящем для сети XML-формате, позволяют реализовать многое из того, что задумано в рамках де-

скрипционных логик, и даже гораздо больше. Конечно, упомянутый вопрос баланса остаётся актуальным и в случае OWL. Но там появляются дополнительные параметры, которые позволяют вмешиваться в отношение «сложность вычислений — выразительность языка». В ходе практической реализации повышение выразительности с одновременным сдерживанием сложности иногда достигается за счёт принесения в жертву гарантированности результата; подобное изменение алгоритмов приводит к тому, что, даже если решение существует, оно не всегда может быть найдено.

4.2. Инструмент выбора между эффективностью и выразительностью

Язык сетевых онтологий OWL демонстрирует прекрасную возможность практического выбора между эффективностью системы и её выразительностью. Так, версия OWL 1.0 имеет три спецификации: разрешимые OWL-Lite — соответствует дескрипционной логике $\mathcal{SHIF}(\mathcal{D})$, OWL-DL — соответствует дескрипционной логике $\mathcal{SHOIN}(\mathcal{D})$ и неразрешимая, но более выразительная OWL-Full, которая не соответствует ни одной дескрипционной логике. Разрабатываемая версия OWL 1.1 соответствует дескрипционной логике $\mathcal{SROIQ}(\mathcal{D})$. Таким образом для реализации предпочтения между эффективностью и выразительностью достаточно подобрать подходящую формальную основу.

Вторым вариантом решения проблемы эффективности при сохранении достаточной выразительности является оптимизация за счёт использования различных стратегий поиска, упорядочения данных и отсечения решений. Хотя нужно заметить, что подобные стратегии в некоторых случаях могут вызвать неполноту представленных решений, — даже если решения и имеются, они могут быть не найдены, или найдены, но не все или не самые оптимальные.

4.3. Постановка алгоритмической проблемы как вопрос к системе

В традиционных общетеоретических логических системах, включая те, что рассматриваются в теории естественного вывода, язык систем не подразумевает постановки никаких запросов или вопросов. Однако, выполняя экспликацию того, что появилось в рамках дескрипцонных логик, можно сказать, что в некотором смысле на метауровне подобный запрос всё же осуществляется. С точки зрения информатики такого рода запрос связан с одной из алгоритмических проблем, относящейся к решению вопроса о выполнимости, ответом на который является построение вывода в рамках заданной формальной системы. В математической логике, в модальных логиках и других общетеоретических системах это, пожалуй, единственный запрос, который принято ставить по отношению к заданной формальной системе. Именно ему посвящена определяемая в рамках этих систем разрешающая процедура. Более того, отсутствие разрешимости относительно подобного рода запроса в общем случае не является сигналом к отказу в разработке формального аппарата системы. В отличие от общетеоретических построений, прикладная область обязывает быть внимательным к вопросу баланса выразительности и эффективности, о чём шла

речь в предыдущем разделе. Таким образом, проблема разрешимости становится принципиальной.

Язык базовой дескрипционной логики $\mathcal{ALC}$ является разрешимым относительно вопроса о выполнимости правильно построенных формул, поскольку он сводим к соответствующим фрагментам модальной и первопорядковых логик. Кроме того, в дескрипционной логике можно различать несколько видов выполнимости.

1. **Выполнимость концепта C в интерпретации $\mathcal{I}$:** если $C^{\mathcal{I}} \neq \varnothing$.

2. **Выполнимость концепта C:** если существует интерпретация $\mathcal{I}$, в которой он выполняется.

3. **Выполнимость терминологии s вида $C_1 \sqsubseteq C_2$:** если в любой интерпретации $\mathcal{I}$ выполняется $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.

4. **Выполнимость концепта C относительно TBox:** если существует интерпретация, являющаяся моделью этого TBox, в которой данный концепт выполняется (интерпретация является моделью TBox, если она выполняет все его терминологии).

Вопрос о выполнимости — это один из вариантов вопросов, которые можно ставить по отношению к логической системе. И все указанные здесь варианты выполнимости можно без особых усилий инкорпорировать в язык традиционных логических систем. Для этого достаточно задать лишь некий аналог операции «теста». Сложнее обстоят дела с другими видами вопросов, которые можно поставить по отношению к дескрипционным логикам. Для включения описания таких видов вопросов в язык самой логики потребуется привлечение дополнительных инструментов. К ряду задач, которые могут быть сформулированы в форме вопроса относительно дескрипционной логической системы, относятся:

1. **Проверка совместимости TBox:** имеет ли заданный TBox хотя бы одну модель.

2. **Проверка онтологии на совместимость:** имеет ли заданная онтология хотя бы одну модель.

3. **Построение классификации терминологии:** найти полную иерархию концептов для заданного TBox относительно вложения.

4. **Принадлежность индивида a концепту C относительно онтологии O:** проверить имеет ли место $a^{\mathcal{I}} \in C^{\mathcal{I}}$ в любой модели онтологии O.

5. **Извлечь экземпляры концепта (matching of concept descriptions):** найти все экземпляры заданного концепта относительно заданной онтологии.

6. **Построить наиболее узкий концепт для группы индивидов (most specific concept):** построить наименьший (по вложению) концепт, описывающий группу индивидов из ABox и учитывающий все свойства этих индивидов, включая те, которыми обладают концепты из утверждений, в которые они входят, а также их отношения к другим индивидам.

7. **Найти наименьший обобщающий концепт для заданной последовательности концептов (least common subsumer):** найти наименьший (по вложению) концепт, в который вложены все концепты заданной последовательности относительно заданной онтологии.

8. **Построить ответ на запрос к онтологии:** выдать все наборы индивидов, которые удовлетворяют данному запросу относительно заданной онтологии (как в программировании с ограничениями).

Таким образом, дескрипционные логики содержат достаточно богатую содержательную основу для формализации вопроса в рамках формальной системы. Тем не менее, на практике даже аппарат дескрипционных логик может давать не совсем желательный результат. К примеру, если в онтологии окажутся несовместимые данные (на практике обычно имеет смысл вести речь о несовместимости данных в ABox), то ответы относительно следствий могут оказаться «тривиальными»: любое положение будет считаться следствием. В результате внесение хотя бы одного «плохого» положения в ABox приведёт к тому, что все положения онтологии (даже TBox) окажутся скомпрометированы. Один из способов борьбы с подобным явлением состоит в ослаблении логической системы, о чём пойдёт речь в следующем разделе.

4.4. Ослабление выводимости

Как ни странно это может прозвучать, одним из способов повышения выразительной силы языка в определённом смысле является ослабление лежащей в его основе логической системы. Действительно, чем больше сущностей некоторой предметной области способен различать язык, тем выше его выразительная сила. Логическая система, помимо прочего, задаёт классы эквивалентных формул и взаимосвязи между ними (с помощью отношения логического следования). Более слабые системы задают больше подобных классов эквивалентности, чем сильные. Такой ход для повышения выразительной силы языка в ряде случаев даже не приведёт к повышению вычислительной сложности. Инструменты логического программирования, с помощью которых могут быть реализованы идеи, заложенные в аппарат дескрипционных и первопорядковых логик, как раз содержат ряд возможностей для ослабления вывода.

В первую очередь вывод может быть ослаблен за счёт того, что некоторые программные средства содержат возможности применения декларативного стиля (воплощающего идеи первопорядковой логики или её фрагмента) наравне с процедурным (воплощающим идеи императивного программирования). К таким средствам можно отнести, в том числе, ECL^{i}PS^e, Classic и Loom.

В частности, язык логического программирования ECL^{i}PS^e, являющийся расширением языка Пролог, поддерживает в первую очередь декларативный стиль. Но при этом в ECL^{i}PS^e, благодаря наличию инструментов для определения операторов, амбивалентности его синтаксиса (в котором вне контекста не различаются функциональные и предикатные константы — имена), использованию метапеременных и наличию специальных операторов управления, поддерживается также и процедурный стиль. Последствия использования этих стилей, с точки зрения логики, проще всего увидеть, сравнив декларативный аналог импликации и оператор управления `->`, используемый в определении связи **if-then-else**. Импликацию нередко сравнивают с условием «если... то», оговариваясь при этом, что в естественном языке смысл связок, даже в случае применения к атомарным высказываниям, может серьёзным образом зависеть от контекста. Тем не менее даже такая формальная среда логического программирования, как Пролог, различает свой аналог импликации и свой аналог связи «если... то».

Инструментарий ECL^{i}PS^e и Пролога позволяет реализовать импликацию различными способами. Два из наиболее близких по смыслу, которые содержатся в самих интерпретаторах этих языков, связаны с оператором построения правил и условным оператором управления. Первый из них позволяет реализовать декларативный стиль программирования, второй — процедурный. Поскольку первый не существует лишь на бумаге, а выполняется компьютером, он также может получить процедурную интерпретацию, описывающую ход выполнения Пролог-программы. В свою очередь, для второго в ряде случаев может быть найдена декларативная интерпретация, описывающая его смысл с точки зрения языка логики. Разберём это подробнее.

База знаний Пролога, являющаяся одновременно и программой, состоит из набора утверждений (клауз), которые подразделяются на факты, описывающие безусловные истины, и правила, описывающие условные. Из этого ясно, что импликация в Прологе реализуется за счёт использования правил. С точки зрения логической интерпретации, в рамках правил устанавливается связь между предикатами. Один предикат используется для задания *головы* правила, другие (от одного и более) — для формирования *хвоста* (*тела*) правила[5]. К примеру, правило с головным двухместным предикатом `p0(X1, X2)` и двумя хвостовыми — одноместным `p1(Y1)` и трёхместным `p2(Y1, X1, X2)`, может быть задано следующим образом[6]:

```
p0(X1, X2) :- p1(Y1), p2(Y1, X1, X2).
```

Здесь `:-` — оператор, отделяющий голову правила (слева от него) от хвоста (справа). Предикаты хвоста отделяются друг от друга запятой, правило

[5]Предикат головы правила может повторяться в хвосте. Факт может рассматриваться как правило с пустым хвостом. Возможно придание пропозиционального характера правилам и фактам путём исключения аргументной части предикатов.

[6]Константные имена в Прологе начинаются со строчных букв, переменные — с заглавных.

оканчивается точкой. Логический аналог правила можно найти в секвенциальном исчислении LJ, при этом предикаты в хвосте правила будут сопоставляться с множеством формул антицедента, а предикат головы — с сукцедентом. Из этого становится понятной связь правил с импликацией. В случае правил, содержащих лишь константные имена, предикаты хвоста, вместо запятых, в антицеденте импликации связываются конъюнкциями, а предикат головы ставится в консеквент импликации. К примеру, правило

```
p(a, b) :- q(c), r(c, a, b).
```

соответствует формуле $((q(c) \wedge r(c, a, b)) \rightarrow p(a, b))$. Если правило содержит переменные, то они связываются кванторами всеобщности, областью действия которых является вся формула. Например, правило из первого примера соответствует формуле $\forall X_1, X_2, Y_1((p_1(Y_1) \wedge p_2(Y_1, X_1, X_2)) \rightarrow p_0(X_1, X_2))$, которая эквивалентна формуле $\forall X_1, X_2(\exists Y_1(p_1(Y_1) \wedge p_2(Y_1, X_1, X_2)) \rightarrow p_0(X_1, X_2))$.

Помимо выявления интерпретации правил на языке логики предикатов первого порядка, важно понимать, как влияют правила на поведение программы в ходе выполнения запросов к базе знаний. Когда выполняется запрос, содержащий некоторый предикат, Пролог, в первую очередь, пытается найти унифицируемый с ним предикат среди фактов и в головах правил. Если предикат не найден, Пролог выдаёт в качестве ответа «Нет». Если найден, выполняется унификация (в некотором смысле соответствующая подстановка переменных и сопоставление констант), и в случае обращения к правилам осуществляются дополнительные запросы по предикатам хвоста. Если в ходе выполнения запросов находится хоть одна подходящая унификация, она выдаётся в качестве ответа, иначе выдаётся «Нет».

Помимо правил, Пролог и ECL^{i}PS^e содержат особые операторы управления, действие которых не направлено на поддержку декларативного стиля программирования. Это не означает, что использование этих операторов обязательно приведёт к нарушению декларативного стиля, а означает, что для его сохранения придётся воспользоваться дополнительными навыками программирования и установить специальные ограничения на использование операторов управления. В общем случае они поддерживают лишь процедурный стиль. Рассмотрим самый близкий по смыслу к импликации оператор управления **if-then-else** и покажем последствие использования процедурного стиля. В Прологе **if** `A` **then** `B` **else** `C` записывается в виде:

```
A -> B ; C.
```

Здесь `A`, `B` и `C` могут быть заменены на любые куски программного кода, выполнимого Прологом. Результатом выполнения программы, написанной на языке Пролог, с использованием лишь декларативного стиля может быть *Да* (*Yes*) или *Нет* (*No*). Кроме того, может возникнуть *расхождение*, которое проявится в том, что программа не остановится и результат не будет выведен на экран. Если же программа задействует и процедурный стиль программирования, в качестве результата может получено *Прерывание* (*Abort*), как, например, в случае

деления на ноль. Схема оператора условного управления работает следующим образом: сначала осуществляется попытка выполнения того, что записано в `A`, если оно даёт ответ «Да», то осуществляется попытка выполнения того, что записано в `B`, а результат выполнения `B` станет результатом работы всего рассматриваемого куска программного кода. Если же выполнение того, что записано в `A`, даёт ответ «Нет», то осуществляется попытка выполнения того, что записано в `C`, и результат выполнения `C` станет результатом работы всего рассматриваемого куска программного кода.

Может показаться, что использованный здесь оператор `->` является прямым аналогом **if-then**, однако это не так. Программный код в форме

```
A -> B.
```

в случае отсутствия расхождения будет давать эффект, соответствующий логической формуле $A \wedge B$, и будет эквивалентен коду

```
A -> B ; fail.
```

который эквивалентен формуле $(A \to B) \wedge (\neg A \to \bot)$. Здесь, если процесс выполнения доходит до `fail`, рассматриваемый кусок программного кода даст ответ «Нет». Когда записывается обычное логическое выражение, подразумевается, что то, что записывается — утверждается. Поэтому с точки зрения логики $\bot \vee \top$ эквивалентно $\top$, в то время как с точки зрения операциональной семантики утверждается, что вначале может быть получен ответ «Нет», а затем «Да».

Прямым аналогом **if-then** будет код в форме

```
A -> B ; true.
```

Следует добавить, что операторы управления не могут включаться в базу знаний ECL^{i}PS^e непосредственно. Их можно использовать либо в хвосте правил, либо при формировании запросов. Поэтому на практике в чистом виде форму `A -> B ; true` можно встретить лишь в запросе. При построении программы нужно использовать подходящую модификацию, например, в форме

```
1  %                ( ABox):
2  rationalbeing(masha).
3  rationalbeing(vova).
4  rationalbeing(zhenya).
5  terrestrial(zhenya).
6  terrestrial(slava).
7  %                 ( TBox):
8  terrestrial(X) :- rationalbeing(X). %                 3
9
10 %
11 solveDom(A, C) :-
12   declareDom(Biengs),
```

```
13    searchDom([X], Biengs),
14    testDom(X, A, C).
15
16  %
17  declareDom(Biengs) :-
18    Biengs = [masha, vova, zhenya, slava, kolya].
19
20  %                          Vals          Vars
21  searchDom(Vars, Vals) :-
22    ( foreach(V,Vars), param(Vals) do member(V,Vals) ).
23
24  %
25  testDom(X, A, C) :-
26    ((rationalbeing(X), A = rationalbeing(X), C = terrestrial(X), !)  ;
27    ((terrestrial(X), A = (not rationalbeing(X)),
28    C = (terrestrial(X)), !);
29    (A = (not rationalbeing(X)), C = (not terrestrial(X))))).
```

Листинг 1. Код ECL^{i}PS^e: оператор построения правил в роли импликации

```
ifth(A, B) :- A -> B ; true.
```

Теперь сравним два рассмотренных аналога импликации Пролога.

Правило Пролога, согласно процедурной интерпретации, является частью процедуры, где его голова — это имя процедуры, а хвост — часть тела. Тогда понятно, что голова правила является просто «точкой входа». Успешное выполнение правила, если оно оказалось выбрано в ходе построения ответа на запрос, целиком обусловлено успехом выполнения его хвоста. Можно сказать, что голова не выполняется, а наследует результаты выполнения хвоста. С точки зрения декларативной интерпретации это означает, что успешное выполнение антецедента в некотором смысле порождает (конструирует) консеквент. Наличие или отсутствие подтверждения консеквента непосредственно определяется выполнимостью или невыполнимостью антецедента. На языке динамической логики это означает, что осуществляется тест антецедента, и в случае успеха устанавливается консеквент. Если выполнение антецедента оказывается неуспешным, правило не задействуется, а истинность предиката, связанного с головой, определяется другими факторами. Такая связь антецедента и консеквента является прямым аналогом сильного отношения следования. Утверждение антецедента неизбежно приводит к утверждению консеквента. Подобная интерпретация импликации подойдёт для моделирования детерминированных полных моделей или для описания концептуальных связей (т. е. устанавливаемых «по определению», а по результатам наблюдений). Правила Пролога позволяют выявить в базе знаний то, что сформулировано неявно.

Пример программного кода ECL^{i}PS^e, демонстрирующий возможность использования оператора построения правил в качестве импликации, приведён

```
?- solveDom(A, C).

A = rationalbeing(masha)
C = terrestrial(masha)
Yes (0.00s cpu, solution 1, maybe more)

A = rationalbeing(vova)
C = terrestrial(vova)
Yes (0.00s cpu, solution 2, maybe more)

A = rationalbeing(zhenya)
C = terrestrial(zhenya)
Yes (0.00s cpu, solution 3, maybe more)

A = (not rationalbeing(slava))
C = terrestrial(slava)
Yes (0.00s cpu, solution 4, maybe more)

A = (not rationalbeing(kolya))
C = (not terrestrial(kolya))
Yes (0.00s cpu, solution 5)
```

Листинг 2. Запрос и ответы: оператор построения правил в роли импликации

в листинге 1. Результат выполнения соответствующего запроса представлен в листинге 2.

Согласно процедурной интерпретации условного оператора управления, сначала выполняется часть, сопоставляемая с антицедентом, а в случае успеха, дополнительно, выполняется часть, связываемая с консеквентом. Оператор будет считаться отработавшим успешно, только если обе части выполнены успешно или же не выполнена успешно первая часть. Таким образом, условный оператор управления, в отличие от оператора построения правила, не является конструктором, устанавливающим концептуальные связи или выявляющим утверждения базы знаний, а позволяет протестировать, можно ли сопоставить соответствующую импликативную связь выбранным фрагментам сведений, которые удалось свести в единую базу знаний. При этом сама, даже успешно протестированная, пара (антицедент, консеквент) может соответствовать не только неустойчивым корреляциям, но и вовсе никак не связанным событиям. В отличие от оператора построения правила, в случае условного оператора управления наличие или отсутствие подтверждения консеквента определяется выполнимостью или невыполнимостью самого консеквента. Выполнимость или невыполнимость антицедента с этим может быть технически (формально) никак не связана. Пример программного кода $\mathrm{ECL}^i\mathrm{PS}^e$, демонстрирующий возможность использования условного оператора управления в качестве импликации, при-

```
%                ( ABox):
rationalbeing(masha).
rationalbeing(vova).
rationalbeing(zhenya).
terrestrial(zhenya).
terrestrial(slava).
%                               ( ABox):
ifth(A, C) :- A -> C ; true. %                  1

%
solveDom(A, C) :-
  declareDom(Biengs),
  searchDom([X], Biengs),
  testDom(X, A, C).

%
declareDom(Biengs) :-
  Biengs = [masha, vova, zhenya, slava, kolya].

%                        Vals          Vars
searchDom(Vars, Vals) :-
  ( foreach(V,Vars), param(Vals) do member(V,Vals) ).

%
testDom(X, A, C) :-
  ifth(rationalbeing(X), terrestrial(X)), %             1
  ((rationalbeing(X), A = rationalbeing(X), C = terrestrial(X), !)  ;
  ((terrestrial(X), A = (not rationalbeing(X)),
  C = (terrestrial(X)), !);
  (A = (not rationalbeing(X)), C = (not terrestrial(X))))).
```

Листинг 3. Код ECL^{i}PS^e: условный оператор управления в роли импликации

```
?- solveDom(A, C).

A = rationalbeing(zhenya)
C = terrestrial(zhenya)
Yes (0.00s cpu, solution 1, maybe more)

A = (not rationalbeing(slava))
C = terrestrial(slava)
Yes (0.00s cpu, solution 2, maybe more)

A = (not rationalbeing(kolya))
C = (not terrestrial(kolya))
Yes (0.00s cpu, solution 3)
```

Листинг 4. Запрос и ответы: условный оператор управления в роли импликации

ведён в листинге 3.

В листингах ответов 2 и 4 собраны комбинации значений истинности антецедента `A` и консеквента `C`, при которых каждый из аналогов импликации оказывается истинным. Если сравнить лишь варианты встречающихся комбинаций, не сопоставляя их с базой знаний, может показаться, что их результаты выполнения совпадают — $\{\langle\top,\top\rangle,\langle\bot,\top\rangle,\langle\bot,\bot\rangle\}$. Однако следует обратить внимание, что в 4 листинге часть значений, эквивалентных $\langle\top,\top\rangle$, отсутствует ввиду того, что условный оператор управления (в отличие от оператора построения правил) не представляет собой порождающую процедуру, генерирующую консеквенты.

В результате можно сделать вывод, что первый аналог импликации позволяет строить умозаключения в соответствии с принципом контрапозиции, а второй — нет, т. е. второй соответствует искомой, более слабой, версии. В системах Classic и Loom, позволяющих реализовать некоторые фрагменты дескрипционных логик, также существуют подобные аналоги импликации. Классической (сильной) импликации соответствует декларативное отношение вложенности $\sqsubseteq$, используемое для задания концептуальных свойств языка в форме аксиом TBox. Слабой импликации соответствует процедурное отношение $\Rightarrow$, используемое для описания выявленных связей между индивидами в рамках ABox.

Литература

Alchourrón *et al.* 1985 — *Alchourrón C., Gärdenfors P., Makinson D.* On the Logic of Theory Change: Partial Meet Contraction and Revision Functions // Journal of Symbolic Logic, Vol. 50, Issue 2. 1985. Pp. 510–530.

Apt, Wallace 2007 — *Apt K., Wallace M.* Constraint Logic Programming using ECL^{i}PS^e. Cambridge University Press, UK. 2007. 349 p.

Baader *et al.* 2010 — *Baader F., Küsters R., and Wolter F.* Extensions to Description Logics // Description Logic Handbook. Theory, implementation, and applications.

Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P. (eds), Cambridge University Press, UK. 2010. Pp. 237–282.

Brachman *et al.* 1991 — *Brachman R., McGuiness D., Patel-Schneider P., Resnick L., Borgida A.* Living with classic: when and how to use a kl-one-like language // Principles of Semantic Networks: Explorations in the Representation ofKnowledge. Sowa J. (ed.), MorganKaufmann, San Mateo. 1991. Pp. 401–456.

De Giacomo, Lenzerini 1994 — *De Giacomo G., Lenzerini M.* Boosting the correspondence between Description Logics and propositional dynamic logics // Proceedings of the 12$^{\text{th}}$ National Conference on Artificial Intelligence (AAAI'94), AAAI Press/The MIT Press. 1994. Pp. 205–212.

Lutz *et al.* 2001 — *Lutz C., Sattler U., Wolter F.* Modal Logic and the Two-Variable Fragment // Computer Science Logic. CSL 2001. Fribourg L. (eds), Lecture Notes in Computer Science, vol. 2142. Springer, Berlin, Heidelberg. 2001. Pp. 247–261.

Rademaker 2012 — *Rademaker A.* A Proof Theory for Description Logics. Springer. 2012. 109 p.

Schild 1991 — *Schild K.* A correspondence theory for terminological logics: Preliminary report // Proceedings of the 12$^{\text{th}}$ International Joint Conference on Artificial Intelligence (IJCAI'1991), Morgan Kaufmann Publishers, Sydney, Australia. 1991. Pp. 466–471.

Schild 1994 — *Schild K.* Terminological cycles and the propositional μ-calculus // Proceedings of the 4$^{\text{th}}$ International Conference on the Principles of Knowledge Representation and Reasoning (KR'94). Doyle J., Sandewall E., and Torasso P. (eds), Bonn (Germany). 1994. Pp. 509–520.

van Benthem 1984 — *van Benthem J.* Correspondence theory // Handbook of Philosophical Logic. Gabbay D., Guenthner F. (eds), vol. II. 1984. Pp. 167–247.

van Benthem 2010 — *van Benthem J.* Modal Logic for Open Minds. Center for the Study of Language and Information. Рукопись. 2 февраля 2010. 390 с.

van Benthem, Smets 2015 — *van Benthem J., Smets S.* Dynamic Logics of Belief Change // Handbook of Epistemic Logic, van Ditmarsch H., Halpern J., van der Hoek W., Kooi B. (eds), College Publications, 2015. Pp. 313–393.