УДК 004.82:004.7

Е.М. Бениаминов

На пути к стандарту представления знаний в Веб алгебраическими средствами

Современные системы открытых связанных данных, включая Wikidata, построены на основе стандартов представления онтологий типа RDF, RDFstar, RDFa, OWL и стандартов языков запросов типа SPARQL, GraphQL и т.д. В то же время, стандарты по представлению аксиом в онтологиях на языках SWRL, RIF и даже OWL мало используются, так как вычисления в онтологиях, представленных в Веб, на основании аксиом и правил вывода получили малое распространение. В связи с этим предлагается использовать богатый опыт теории алгебраических вычислений и алгебраического представления знаний для создания удобного универсального средства построения онтологий, ориентированных на вычисления ответов на запросы с использованием аксиом и правил переписывания для построения ответов на запросы, а не только фактов. Рассматривается возможность построения стандарта алгебраического языка представления онтологий в Веб: Algebraic Web Ontology Language (Algebraic OWL) и некоторые элементы такого стандарта. Используется опыт и методологии подходов Common Algebraic Specification Language (CASL), системы Mathematica, языков программирования Haskell и Prolog.

Ключевые слова: *компьютерные онтологии, представление знаний, библиотеки онтологий, связанные данные, многосортные алгебры, алгебраический подход*

DOI: 10.36535/0548-0027-2022-10-2

МОТИВАЦИЯ И НЕФОРМАЛЬНОЕ ВВЕДЕНИЕ

Простейший распространенный способ представления онтологий в Веб – это использовать стандарт *Resurs Definition Format* (RDF) [1] и представлять онтологии в виде графов, в которых вершинами являются уникальные имена классов, их элементы и некоторые элементы стандартных типов данных, а ребрами, связывающими некоторые из этих вершин, являются свойства классов и элементов также названные уникальными именами. На логическом уровне такие онтологии моделируются конечным множеством элементов и конечным множеством двуместных предикатов.

В стандарте RDF, а также в его расширении RDFS [2] выделяются имена некоторых классов и свойств, которые будут общими для всех онтологий, представленных по этому стандарту. В RDF, RDFS и OWL [3, 4] такими именами классов являются *rdfs:Class*, *owl:Thing* и другие, а именами свойств – *rdfs:subclassOf*, *rdf:type*, с помощью которых в онтологии выделяются подклассы классов и элементы классов. В RDFS фиксировано имя свойства *rdfs:label*, с помощью которого для каждого элемента онтологии можно задать имя элемента в моделируемой области, соответствующе этому элементу онтологии. Естественно, онтологии моделируют какую-то ситуацию, внешнюю по отношению к ней, и имена классов, лейблы элементов и свойств отсылают к совокупностям элементов, элементам и их свойствам этой внешней ситуации. Кроме того, по уникальному имени элемента онтологии обычно можно получить ссылку на адрес в Веб, где этот элемент описывается.

В соответствии с этими стандартами факт, что Энштейн награжден нобелевской премией будет представляться в онтологии тройкой (ребром графа) *{Энштейн награжден нобелевская_премия}*, где *Энштейн* и *нобелевская_премия* – это вершины графа, а *награжден* – это свойство, связывающее эти вершины. В функциональном обозначении этот факт записывается в виде: *награжден(Энштейн, нобелевская_премия)*. Некоторые трудности в этом подходе, связанные с тем ограничением, что могут использоваться только бинарные отношения для представления онтологий, появляются, если, например, в онтологии нужно еще отразить, когда Энштейн был награжден нобелевской премией и за что.

Для преодоления этой трудности в соответствии со стандартом RDFstar [5] разрешается вводить именованные подграфы и для этих имен можно вводить новые свойства или использовать конструкцию (операцию угловых скобок) *<<Энштейн награжден нобелевская_премия>>*, которая делает эту тройку новой вершиной в графе, и ее можно связывать свойствами с другими его вершинами. В *Wikidata* [6], чтобы определить свойства ребра, используются префикс *p:* для имени этого ребра и префикс *ps:* для свойства. Геометрически это означает, что на ребре ставится новая вершина или несколько вершин, и свойства от этих

новых вершин соответсвуют некоторым утверждениям о данном ребре.

В стандарте OWL [3, 4] вводится много других операций, строящих новые классы и свойства из уже существующих классов и свойств, удовлетворяющих некоторым условиям. Примерами таких операций являются теоретико-множественные операции над классами, которые строят новые классы, или взятие обратного свойства для некоторого свойства или операция композиции свойств, которая порождает новое свойство. При этом, в OWL можно задавать некоторые типы аксиом, включая равенство (*owl:equivalentClass, owl:equivalentProperty, owl:sameAs*) двух выражений. Ответы на запросы к онтологиям, построенным в языке OWL, хорошо бы получать с учетом аксиом и, в частности, проверять построенные онтологии на непротиворечивость. Эти функции с разной степени точности выполняют отдельно разрабатываемые программы (*reasoners*).

Множества, на которых действует система операций и между композициями некоторых из этих операций выполняются равенства, удобно моделировать, применив алгебраический подход. Поэтому естественно использовать алгебраический подход моделирования (с его богатым опытом применения в программировании) для моделирования онтологий, на которых задана система операций и аксиом.

ОБЩЕЕ ОПИСАНИЕ АЛГЕБРАИЧЕСКИХ СРЕДСТВ МОДЕЛИРОВАНИЯ

В этом разделе дается краткое описание средств и возможностей алгебраического подхода к моделированию онтологий. Основное средство моделирования в этом подходе – это алгебра. Алгебра, как математическая структура, – это набор множеств и набор операций, действующих на элементах этих множеств.

Источники алгебраического моделирования

По алгебраическому моделированию имеется обширная литература, включая описание языка *CASL* [7], или языка программирования *Haskell* [8], не говоря уже о математической литературе на эту тему. Для моделирования онтологий нам понадобятся алгебры с условными операциями и условными соотношениями, т. е. некоторые операции можно применить, если их аргументы в этой алгебре удовлетворяют определенным равенствам, и некоторые элементы удовлетворяют некоторому равенству (аксиоме), если эти элементы удовлетворяют некоторым условиям. Поэтому здесь мы будем во многом следовать описанию языка *The Maude System* [9, 10] – системы, которая поддерживает в своих спецификациях алгебраическую логику условных равенств и переписываний выражений. Ряд идей алгебраического моделирования взят из системы представления знаний и создания библиотек онтологий ЭЗОП [11, 12]. В основе этой системы используется алгебраический (категорный) подход к представлению знаний, модульная коллективная разработка библиотек онтологий и открытый язык описания онтологий, который формируется самими пользователями при их создании.

Спецификация алгебры, инициальная реализация, аппроксимации, вычисления

В спецификациях алгебр, как и в онтологиях, алгебра задается именами классов, типов данных, именами операций с указанием откуда и куда действует каждая операция (из каких классов и типов данных в какие действует операция) и именами некоторых элементов классов.

Кроме того, на множестве классов и элементов задано отношение класс-подкласс и принадлежность элементов классам. Используя имена элементов и переменных для элементов классов и типов, стандартным способом определяются термы – правильно построенные выражения из переменных, констант и имен операций с учетом типов аргументов операций, типов результатов операций и типов переменных и констант. Аксиомы алгебры задаются в виде равенств некоторых термов одного типа или условных равенств.

По каждой такой спецификации алгебры может быть формально построена соответствующая ей инициальная алгебра, которая определяется как множество всех термов (выражений) этой алгебры без переменных, профакторизованное по минимальному отношению эквивалентности, порожденному аксиомами этой алгебры. Такая инициальная алгебра является идеальным объектом, в котором отражаются все знания, зафиксированные и выводимые из ее спецификации. В общем случае она бесконечна и ее отношение эквивалентности может быть алгоритмически неразрешимым подмножеством в множестве всех пар термов этой алгебры. Для того чтобы использовать алгебраические модели, нужно либо ограничить тип аксиом алгебры так, чтобы отношение эквивалентности на множестве ее термов было алгоритмически разрешимым и проблема равенства в ней разрешалась эффективно, либо в каждый момент времени использовать эффективно разрешимую аппроксимацию инициальной алгебры, состоящую из разрешимого (или конечного) подмножества термов алгебры с эффективно разрешимым подмножеством в отношении его эквивалентности.

Заметим что в инициальной алгебре, построенной по спецификации алгебры, отражаются все знания, вытекающие из спецификации алгебры, а в аппроксимации отражаются эффективно доступные знания, вытекающие из спецификации алгебры, добытые к текущему моменту. В частности, некоторые аппроксимации могут быть представлены в виде базы данных.

Между аппроксимациями инициальной алгебры естественным образом может быть определено отношение, соответствующее тому, что одна аппроксимация является более точной, чем другая. Это означает, что в более точной аппроксимации доступно большее множество термов алгебры и, если некоторое равенство термов доступно в менее точной аппроксимации, то оно доступно и в более точной аппроксимации.

Предложим структуры, в которых можно было бы хранить такие аппроксимации для эффективного вычисления в алгебрах.

Предлагаем рассматривать два типа вычислений в алгебрах: 1) в текущей аппроксимации инициальной алгебры (быстрые эффективные вычисления) на ос-

новании уже добытых знаний и 2) с расширением аппроксимации с добавлением новых термов к текущей аппроксимации алгебры и добавлением новых равенств между термами, которые выводятся из аксиом этой алгебры.

Особый интерес для моделирования онтологий представляют конечные аппроксимации алгебр, соответствующие онтологиям. Конечные аппроксимации создаются по некоторому конечному множеству термов данной алгебры, на котором строится отношение эквивалентности, порожденное конечным множеством пар термов из этого множества. Для элементов этих пар их равенство уже выведено из аксиом алгебры. Таким образом, в конечной аппроксимации алгебры множество ее термов распадается на непересекающиеся подмножества, состоящие из попарно равных между собой термов. В каждом таком подмножестве равных выражений выбирается один элемент, который является представителем – дескриптором элементов этого подмножества. Именно этот элемент (дескриптор) будет выдаваться после вычислений в аппроксимации в ответ на вопрос: чему равен некоторый терм? (В ответ на такой запрос выдается дескриптор, равный в данной аппроксимации этому терму.)

Естественно потребовать также, чтобы в конечной аппроксимации алгебры, если некоторый терм принадлежит конечной аппроксимации, то и все подтермы этого терма должны принадлежать этой аппроксимации.

Отношение равенства в алгебре на множестве термов удовлетворяет аксиоме конгруэнции. Это значит, что любая операция, примененная к равным аргументам, выдает равные значения.

В этих предположениях операция вычисления *calcOnApprox* термов на конечной аппроксимации должна выдавать каждому терму $F(t_1,\ldots, t_k)$ из аппроксимации его дескриптор $calcOnApprox(F(t_1,\ldots, t_k))$. При этом выполняется равенство:

$$calcOnApprox(F(t_1,\ldots, t_k))= calcOnApprox(F(calcOnApprox\ (t_1),\ldots, calcOnApprox\ (t_k))).$$

Поэтому для вычисления термов в аппроксимации достаточно знать только дескрипторы термов с именами операций алгебры и с аргументами-дескрипторами.

Хранение аппроксимации алгебры в том или ином виде называется внутренним текущим представлением алгебры в компьютерной системе по ее спецификации. Быстрые приближенные вычисления в алгебре выполняются по внутреннему представлению. Примером внутреннего представления алгебры чисел в компьютерных системах являются машинные числа.

Обычные программы логического вывода (*reasoners*), применяемые для OWL-онтологий также строят не всевозможные термы из имен онтологий (например, не все возможные пересечения и объединения классов, а только те, которые используются в определении данной онтологии). Важным при построении внутреннего представления онтологии считается использование аксиом предиката *isSubclassOf* вида:

если *isSubclassOf(?a, ?b) AND isSubclassOf(?b, ?c),* то *isSubclassOf(?a, ?c);*

если *isSubclassOf(?a, ?b) AND isSubclassOf(?b, ?a),* то *?a=?c;*

если *isSubclassOf(?a,?b) AND isSubclassOf(?a,?c),* то *isSubclassOf(?a, П(?b,?c)),*

где *П(?b,?c)* – операция пересечение классов *?b* и *?c.*

Кроме того, важно использование аксиом для предиката принадлежности элемента классу, который здесь будет называться *hasType*[1]. Аксиомы для этого предиката имеют вид:

если *hasType (?e, ?a) AND isSubclassOf(?a, ?b),* то *hasType (?e, ?b);*

если *hasType (?e, ?a) AND hasType (?e, ?b),* то *hasType (?e, П(?a,?b)).*

Что считается противоречием в алгебре? Разные имена, введённые в определении (спецификации) алгебры, считаются разными элементами (исключения составляют имена переменных и имена синонимов уже введенных имен). Если из аксиом алгебры выводится равенство двух имен (например, *True=False*), то это считается противоречием, на которое должна указать система вывода. В онтологиях реакцией на такое указание должна быть проверка на совпадение объектов, соответствующих этим именам в предметной области: если объекты совпадают, то имена объявляются синонимами. В противном случае, требуется изменение аксиом, вызвавших это равенство.

В OWL также введено имя пустого класса *owl:Nothing* с такой аксиомой, что если для некоторого выражения *e* выводится выражение *hasType(e, Nothing),* то это противоречие.

С учетом приведенных аксиом конечную аппроксимацию алгебры онтологии можно представить в виде базы данных с отношениями *descriptor*, *resultOp, subclassOf, elementOf.*

В отношении *descriptor* описываются все дескрипторы, входящие в аппроксимацию. Указывается выражение дескриптора с уникальными именами операций и констант, входящих в его выражение, и внутренний идентификатор этого дескриптора.

Для данной аппроксимации в отношении *resultOp* описываются результаты операций онтологии, примененных к дескрипторам. В этом отношении для имени операции онтологии и списка идентификаторов дескрипторов аргументов операции дается идентификатор дескриптора, который является результатом этой операции для этих аргументов. Вместо идентификаторов можно использовать ссылочные данные на дескрипторы (адреса дескрипторов) в отношении *descriptor.*

В отношении *subclassOf* описывается отношение, состоящие из пар идентификаторов дескрипторов для неравных классов, связанных отношением класс-подкласс, между которыми нет классов, отличных от них самих в данной аппроксимации. Если по этому отношению построить граф, вершинами которого яв-

[1] В этой статье будут различаться отношения и предикаты. Так, *type* – это отношение (подмножество в множестве пар из элементов и классов). Выражение *type(e, A)* задает елемент *(e, A)* в отношении *type.* С другой стороны, *hasType* – предикат. Выражение *hasType(e, A)* принимает логическое значение.

ляются дескрипторы классов данной аппроксимации, то с учетом аксиомы для предиката *isSubclassOf* построенный граф будет ацикличным.

В отношении *elementOf* связывается идентификатор дескриптора элемента аппроксимации с дескриптором минимального класса, содержащего этот элемент в аппроксимации.

В отношении *notEqualDescriptor* фиксируются пары дескрипторов, которые в онтологии объявлены неравными.

Используя эти отношения, можно эффективно вычислять ответы на следующие вопросы к текущей аппроксимации:

- Чему равно данное выражение в текущей аппроксимации?
- Равны ли два выражения?
- Является ли данное выражение элементом класса, заданного другим выражением?
- Какие элементы являются элементами данного класса?
- Какие элементы связаны с данным элементом определенным свойством?
- Какие элементы имеют определенное значение для данного свойства?
- Какие свойства имеет данный элемент?

Заметим, что в перечисленных вопросах элементы, классы и свойства могут задаваться не только именами, но и сложными выражениями, задающими элементы, классы и свойства в аппроксимации.

Используя программные модули, строящие ответы на перечисленные вопросы, можно стандартным образом реализовать язык *SPARQL* для построения ответов на более сложные запросы к аппроксимации. Теоретически можно представить, что по аппроксимации строится граф из дескрипторов аппроксимации. Вершинами этого графа будут дескрипторы элементов аппроксимации, а ребрами – дескрипторы элементов бинарных отношений (свойств) в этой аппроксимации. В результате получаем расширение языка *SPARQL* за счет того, что теперь вершинами и свойствами в тройках могут быть сложные алгебраические выражения, а не только имена и элементы типов данных.

Правила переписывания и продукции в алгебрах и онтологиях

В алгебраическом подходе так же, как и в онтологиях, кроме равенств большое значение имеют правила переписывания и продукции. Для работы с онтологиями в Веб разработаны стандарты языков *Semantic Web Rule Language* (SWRL) [13] и *Rule Interchange Format* (RIF) [14]. Мы будем различать правила переписывания для вычислений выражений в аппроксимации алгебры и правила, изменяющие аппроксимации алгебры.

Правило переписывания имеет вид:

$$Condition(x_1,\ldots,x_k) =>$$
$$rewrite\ L(x_1,\ldots,x_k)\ to\ R(x_1,\ldots,x_k)\ [имя\ правила],$$

где $L(x_1,\ldots,x_k)$ и $R(x_1,\ldots,x_k)$ – термы алгебры одного типа с переменными $x_1,\ldots,x_k$.

С помощью таких правил могут задаваться аксиомы и способы вычисления в алгебре. Правило действует в аппроксимации алгебры при построении ответа на вопрос: «Чему равен терм *t?*», по следующему принципу. Если в терме *t* есть подтерм $L(t_1,\ldots,t_k)$, и в аппроксимации выполняется условие $Condition(t_1,\ldots,t_k)$, то в терме *t* подтерм $L(t_1,\ldots,t_k)$ заменяется на $R(t_1,\ldots,t_k)$. Полученный терм *t'* является результатом переписывания тема *t* за один шаг по этому правилу. Терм *t'* считается (полагается) равным терму *t*. В общем случае при спецификации алгебр (или онтологий) правил переписывания может быть много, и они при вычислениях применяются к терму *t* и результатам его переписывания до тех пор, пока не получится терм, к которому не может быть применено ни одно из правил. Этот терм называется результатом вычисления терма *t* по системе правил переписывания (каноническим термом).

Частными случаями вычислений по правилам переписывания являются вычисления в аппроксимации с помощью отношения *resultOp* на дескрипторах. В более общем случае правила переписывания могут быть построены по равенствам – аксиомам алгебры (левая часть равенства переписывается в правую часть). В теории вычислений по правилам переписывания известны алгоритмически проверяемые условия на систему правил переписывания – условия Черча-Россера [15], при выполнении которых каждый терм алгебры переписывается к единственному своему каноническому виду, независимо от последовательности применяемых правил. Однако во многих системах символьного вычисления последовательность применения правил может фиксироваться и управляться пользователями при описании алгебры.

Следующий вид правил, который важен в онтологиях, – это правила, которые меняют аппроксимацию алгебры онтологии. Основные из них те, которые по каждому предложению текста, описывающим онтологию, достраивают внутреннее представление онтологии (ее аппроксимацию). При этом будет предполагаться, что каждая онтология строится в среде другой онтологии, которая предоставляет свой словарь и операции для построения новой онтологии. На первых этапах построения библиотеки онтологий такой средой (ядром) построения новых онтологий может быть внутреннее представление, построенное по стандартам RDF, RDFS или OWL. К этой аппроксимации ядерной онтологии нужно отнести и основные правила изменения аппроксимаций.

К основным правилам изменения аппроксимации онтологии нужно отнести правило создания в базе данных аппроксимации нового дескриптора вида *create(NameElement, Type)*. Это правило создает по имени *NameElement* новый дескриптор элемента в уже существующем в аппроксимации типе (классе) *Type*. Тип *Type* может быть задан не только именем, но и сложным выражением. Условием создания такого элемента в этом типе является уникальность имени элемента в этом типе и возможность добавления новых элементов в *Type* (например, *Type* не должен быть равен пустому классу).

Правило *makeElement(El,Cl)* делает уже существующий в аппроксимации дескриптор, соответ-

ствующий *El*, элементом класса, заданного дескриптором *Cl*. Естественно, перед выполнением этого правила нужно проверить, является ли *Cl* классом в аппроксимации. При выполнении этого условия нужно найти тип дескриптора *El* и в базе данных аппроксимации сделать этот дескриптор элементом пересечения типа дескриптора *El* и дескриптора класса *Cl*.

Другим правилом изменения аппроксимации онтологии может быть правило, которое по выражению <*Субъект Свойство Объект*> добавляет к базе данных аппроксимации факт, отражающий то, что дескриптор *Субъекта* связывается с дескриптором *Объекта* свойством, которое задается дескриптором *Свойства*. Условием применения этого правила является то, что в уже построенной аппроксимации дескриптор *Свойства* принадлежит классу *Property*, дескриптор *Субъекта* принадлежит области определения (*domain*) *Свойства*, а дескриптор *Объект* принадлежит области значений (*range*) *Свойства*.

Особым образом обрабатывается выражение <*Субъект subclassOf Объект*>. По этому выражению запускается правило *makeSubclass(Субъект, Объект)*. Это правило выполняется почти так же, как и правило, запускаемое выражением <*Субъект Свойство Объект*>, но предварительно в аппроксимации проверяется условие, является ли дескриптор *Объекта* подклассом дескриптора *Субъекта*. Если это условие выполняется, то запускается правило *makeEqual(Субъект, Объект)*, которое в аппроксимации делает дескриптор *Субъекта* равным дескриптору *Объекта*.

Правило *makeEqual(El1, El2)* может быть применено, если дескрипторы выражений *El1* и *El2* одного типа. На множестве дескрипторов используется линейный порядок, связанный с лексикографическим порядком на термах и длиной терма – весом терма. Переменные имеют вес больший, чем вес терма без переменных. Для некоторых термов вес может быть установлен пользователем. При приравнивании двух дескрипторов одного типа, дескриптором в новой аппроксимации оставляется дескриптор с меньшим весом, дескриптор с большим весом удаляется из дескрипторов, а пара: старый дескриптор и новый дескриптор, помещается в отношение *resultOp*. Затем ссылки на удаленный дескриптор во всех отношениях аппроксимации заменяются ссылками на новый дескриптор. В отношениях аппроксимации, в которых дескрипторы не должны связываться с собой (например, не должно быть строк *resultOp(?X,?X))*, такие появившиеся строки удаляются. Если выражения *El1* и *El2* в исходной аппроксимации являются классами, то, кроме того, находят все классы, лежащие между *El1* и *El2* и приравнивают их к новому дескриптору по тому же правилу. Процесс приравнивания заканчивается (ввиду конечности множества дескрипторов в исходной аппроксимации) новой аппроксимацией онтологии.

Правило *createVar(NameVar, Type)* аналогично правилу *create(NameElement, Type)*, но создает в аппроксимации не элемент типа *Type*, а переменную, дескриптор которой имеет большой вес.

Правило *createTerm(NameOperation, ListTermsArgument, Type)* проверяет, есть ли такой терм в аппроксимации и, если его нет, то создает новый терм в аппроксимации типа *Type* по дескриптору выражения *NameOperation*, задающего имя операции, и списку дескрипторов *ListTermsArgument* – аргументов операции. Условием применения правила является существование дескрипторов *ListTermsArgument* в аппроксимации, соответствие их типов типам аргументов операции и выполнение для этого списка дескрипторов условия применения операции с именем, заданным термом *NameOperation*. Правило помещает построенный терм с именем операции и аргументами-дескрипторами в отношение аппроксимации *descriptor*.

Правило *makeNotEqual(El1,El2)* фиксирует в отношении аппроксимации *notEqual* неравенство двух дескрипторов, соответствующих выражениям *El1* и *El2*, которое задается в спецификации онтологии.

Правило *createRewriteRule* создает в аппроксимации правило переписывания, указывая его идентификатор, имя, список переменных с их типами, левый и правый термы правила переписывания, предусловия и постусловия выполнения правила. Условием применения этого правила является также правильность построения входящих в него термов, вхождение всех переменных из списка в левый терм и вхождение переменных правого терма и условий переписывания в список переменных правила.

Важным правилом для алгебраического моделирования онтологий является правило *calculateOnApprox (Term)*, возвращающее дескриптор, равный выражению *Term* в текущей аппроксимации. Это правило используется в других правилах для получения дескрипторов выражений, входящих в эти правила, и для проверки условий применения правил. Правило *calculateOnApprox(Term)* для сложного терма вида $Term=NameOperation(t_1,\ldots,t_k)$ находит сначала дескрипторы $d_1,\ldots,d_k$ для аргументов $t_1,\ldots,t_k$ по тому же правилу *calculateOnAapprox*, а потом проверяет, находится ли терм $NameOperation(d_1,\ldots,d_k)$ в отношении *descriptor* или *resultOp* текущей аппроксимации. Если это выражение будет найдено там, то правило *calculateOnApprox* выдает дескриптор выражения.

Для проверки условий могут использоваться также правила (предикаты) *isSubclassOf (subCl, Cl), isElementOf(El, Cl), has(Субъект, Свойство, Объект)*, которые вычисляются по аппроксимации и возвращают булевы значения *True* или *False*. Правило *isSubclassOf(subCl, Cl)* вычисляется исходя из отношения аппроксимации *subclassOf* и свойств рефлексивности и транзитивности отношения *isSubclassOf*. Кроме того, *isSubclassOf(owl:Nothing, Cl)* является истиной для любого класса *Cl*. При вычислении правила *isElementOf(El, Cl)* используется отношение аппроксимации *elementOf(El, Cl1)* и следующее свойство: если *elementOf(El, Cl1)* входит в отношение и верно *isSubclassOf(Cl1, Cl)*, то верно *isElementOf(El, Cl)*.

Правило *calculate* по входному выражению выдает результат. В отличие от правила *calculateOnApprox*, правило *calculate* действует с расширением аппроксимации во время вычислений и может использовать любые правила, включая правила переписывания. В частности, правило *calculate*, примененное к терму $NameOperation(t_1,\ldots,t_k)$, должно проверить сначала – не является ли *NameOperation* именем правила, для

которого заданы действия в аппроксимации, если это имя правила, то применяется правило с этим именем к термам t_1 ,..., t_k. В противном случае, находятся дескрипторы d_1 ,..., d_k термов t_1 ,..., t_k по тому же правилу *calculate* и проверяется, находится ли терм *NameOperation*(d_1 ,..., d_k) в отношении *descriptor* или *resultOp* аппроксимации на этом шаге вычисления. Если находится, то выдается дескриптор – результат вычисления. Если не находится, то терм *NameOperation* (d_1 ,..., d_k) помещается в отношение *descriptor*, расширяя аппроксимацию, а в результате вычислений выдается терм *NameOperation*(d_1 ,..., d_k).

Среди входных выражений для правила *calculate* смогут быть сложные вопросы, состоящие из последовательности предварительных действий, на которые *calculate* выдает ответ.

Если на некоторый вопрос к онтологии система не может вывести ответ, а пользователь знает стратегию получения ответа на вопрос, то он может, задавая последовательность наводящих вопросов к онтологии, по которым система, отвечая на вопросы, достраивает аппроксимацию онтологии и приводит к ситуации, в которой получается ответ на исходный вопрос. Таким образом происходит обучение аппроксимации онтологии по наводящим вопросам пользователей.

Помимо вопросов к онтологиям, меняющим аппроксимации онтологий, может быть разработан язык запросов, подобный *SPARQL*, с использованием правил, расширяющих аппроксимацию во внутреннем представлении онтологии запроса, но не меняющим исходную аппроксимацию онтологии.

Система правил в алгебраическом подходе должна быть открытой и предоставлять возможность пользователям конструировать новые правила из уже существующих.

Естественно, все правила должны фиксировать в журнале, какие изменения в аппроксимации они проделали, а в случае невозможности их выполнения выдавать соответствующие сообщения. Эти сообщения необходимы для отладки конструирования онтологий, аппроксимаций, запросов и правил. Каждое правило должно быть оформлено в виде транзакции: оно либо выполняется до конца, либо не выполняется совсем.

МОДУЛЬНАЯ ОРГАНИЗАЦИЯ ОНТОЛОГИЙ

Модульная организация программ и использование библиотек программ является основой современного стиля программирования. Это относится и к алгебраическому моделированию. В системе алгебраического представления знаний *CASL* и алгебраических языков программирования *Haskell* модульность лежит в основе.

В разработках онтологий модульность находится только в начале развития. В стандарте *OWL* есть конструкция *owl:Imports*, которая позволяет использовать внутри одной онтологии другие онтологии. Интересен способ использования *insert*- конструкций языка *SPARQL* с подзапросами к внешним онтологиям. С помощью этой конструкции фрагмент одной онтологии может быть добавлен к другой онтологии. Особое место в модуляризации онтологий занимает создание и использование шаблонов проектирования онтологий *Ontology Design Patterns* (ODPs) [16, 17]. Каждый такой шаблон представляет собой небольшую документированную *OWL*-онтологию с классами, свойствами, элементами и аксиомами, которая описывает фрагмент ситуации данной области знания. Предварительно создается и отлаживается набор таких шаблонов. Возможно использование внешних устоявшихся шаблонов. Потом из таких шаблонов формируется онтология предметной области как из модулей. О современном состоянии модуляризации онтологий и проблемах модульного проектирования онтологий хорошо написано в работе [18]. По теме модульного построения онтологий проводятся конференции [19].

В предлагаемом нами алгебраическом подходе к построению онтологий в Веб модульность имеет важное значение. Как уже отмечалось ранее, каждая онтология создается в среде некоторой другой онтологии, которая предоставляет свой словарь, начальную аппроксимацию онтологии и правила среды для формирования новой онтологии. Другие онтологии, разработанные в этой среде, могут быть также использованы как шаблонные внутри новой онтологии и могут образовывать фрагмент библиотеки онтологий предметной области, разрабатываемой группой специалистов, объединившихся с целью создания такой библиотеки по некоторой проблемной области знания.

Естественно, что некоторые онтологии библиотек разных предметных областей не могут быть объединены и могут противоречить друг другу. Если такое объединение все же требуется, то это может представить не только для онтологий, но и для самой предметной области проблему, которую должны решить специалисты проблемных областей, знания о которых представляются в онтологиях.

Тексты (спецификации) онтологий в библиотеках должны быть не очень большими и организовываться наподобие вики-страниц со ссылками на другие страницы онтологий, которые в них используются.

Каждому тексту онтологии соответствует внутреннее представление онтологии, которое хранится отдельно. К онтологии в библиотеке можно обратиться с запросом на языке, подобном *SPARQL*, расширенном правилами онтологии. Ответ на запрос строится по аппроксимации онтологии с использованием правил онтологии. Ответ может быть представлен в табличном или графическом виде, отображая, например, по фрагментам, структуру аппроксимации онтологии.

Онтологии, находящиеся в библиотеке, не могут меняться. Каждая версия онтологии является новой. Новые онтологии могут добавляться в библиотеку. Удаляться из библиотеки могут только такие онтологии, которые не используются.

Одна онтология может использоваться в другой онтологии двумя способами. При одном способе добавление одной онтологии к другой может состоять в процедуре слияния двух внутренних представлений (аппроксимаций) онтологий и добавления правил одной онтологии к правилам другой. Если при этом возникают противоречия, то возможен другой способ добавления онтологии. Для этого первая онтология достраивается по предложениям текста второй онтологии. В случае обнаружения противоречия (не-

выполнение условия) на некотором предложении, противоречие устраняется добавлением дополнительных условий.

Для удобства чтения текстов онтологий их модульная организация может быть устроена с использованием всех принципов объектно-ориентированного программирования: иерархии, наследования правил, инкапсуляции, полиморфизма правил. Возможно также использование параметрических онтологий по аналогии с параметрическими модулями в алгебраическом подходе к программированию [9, 10].

В библиотеке онтологий проблемной области могут храниться не только шаблонные онтологии, описывающие фрагменты знаний проблемной области, но шаблонные ситуации для формирования вопросов к знаниям проблемной области. Используя библиотечные онтологии, пользователи могут сформировать онтологии задач, описывающих конкретные ситуации предметной области и обратиться к ним с запросом. Таким образом, модульность позволит моделировать ситуации проблемной области и отвечать на вопросы типа: «Что будет, если …?».

О ЯЗЫКЕ ПРЕДСТАВЛЕНИЙ ОНТОЛОГИЙ В АЛГЕБРАИЧЕСКОМ ПОДХОДЕ

Представлять онтологии в библиотеках онтологий по различным областям знания должны специалисты в этих областях. В каждой области знания складывается собственный язык для описания знаний этой области. Это может быть связано как с традициями в этой области знания, так и с необходимостью иметь язык, адекватный миру представляемых знаний. Было бы странно и неудобно представлять знания, например, по математике на естественном языке, не пользуясь формулами. Это относится и к другим областям знания.

В учебниках, когда описываются понятия и закономерности предметной области, обычно вводятся и конструкции языка, с помощью которых об этих понятиях можно говорить. Естественно, эту возможно нужно обеспечить и при построении онтологий, в которые помимо имен классов, свойств, элементов, переменных, операций пользователи могут вводить языковые конструкции для применения правил. Простейшими из таких конструкций могут быть могут быть шаблоны типа *If_Then_Else_*, в котором вместо подчеркиваний вставляются выражения соответствующих типов (в языках *CASL*, *Haskell*, *Mathematica*, *Maude* обеспечивается возможность ввода таких конструкций для именования и вызова любых функций, определяемых в программе). Помимо этого, в онтологиях должна быть обеспечена возможность определения формальных языков (например конструкциями Бэкуса-Наура) и привязка выражений языка к действиям в аппроксимации онтологии. В системах, работающих с онтологиями, естественно, должны быть средства отладки введенных языков.

Таким образом, язык в алгебраическом подходе становится открытым и контекстным. Правильность языкового выражения (синтаксическая и семантическая) проверяется в контексте определенной онтологии, с учетом используемых в ней онтологий, и ее среды на основании аппроксимации онтологии. Формируя библиотеку онтологий предметной области, пользователи одновременно формируют и согласовывают язык своей предметной области, на котором они будут работать с построенными онтологиями.

Среди языковых конструкций онтологии могут быть конструкции, которые видны в контексте другой онтологии (например, если эти две онтологии разработаны в одной онтологии-среде). Их использование в другой онтологии приводит к автоматической подгрузке первой онтологии с параметрами, указанными в этой языковой конструкции. Такой способ является, иногда, более естественным для формирования новых онтологий и онтологий задач, чем предварительная загрузка всех модулей с указанием функций, которые надо использовать.

БЛИЗКИЕ РАБОТЫ

Несколько работ по алгебраическому представлению знаний и алгебраическому программированию уже упоминалось в тексте. На меня большое влияние оказывали работы профессора Жозефа Гогуена (Joseph Goguen**)** [20], начиная с работ по алгебраическому моделированию абстрактных типов данных и до приложения этого подхода к онтологиям [21].

Продолжением работ Ж. Гогуена и работ группы *The Common Framework Initiative for algebraic specification and development of software* (CoFI) [22] стала разработка языка спецификаций и моделирования для распределенной онтологии (*The distributed ontology, modelling and specification language – DOL*) [23, 24]. В основе этого подхода лежит идея модульности представления знаний и интеграция знаний, представленных на разных языках *OWL, UML, CASL* …, в основе которых может быть лежат разные логики. В языке *DOL* предоставляются средства для интерпретации одних языков в других с использованием онтологий. Рассматриваются библиотеки и сети онтологий.

Автора настоящей статьи приводят в восторг работы профессора Стефена Вольфрама (Stephen Wolfram) [25] и его фирмы [26], в частности, системы *Mathematica* [27] и *Wolfram Alpha* [28], в которых накоплен громадный объем знаний и их возможностей. К недостаткам этих систем следует отнести некоторую их закрытость в пополнении знаний и ограниченность коллективного творчества сторонними пользователями.

Достижения проекта *Wikidata* [29, 30] также поражают накопленным объемом фактов и размерами онтологий, а также большим количеством групп, которые в этом проекте активно участвуют и представляют данные по своим областям знания.

В 2022 г. в рамках проекта *MediaWiki* открывается новый проект: *Abstract Wikipedia* [31]. Принципы расширения возможностей систем *Wikipedia* и *Wikidata* изложены в статье [32] . В проекте предлагается создание многоязычной *Wikipedia*, в которой содержание вводится только один раз на одном из языков, но отображается на многих языках за счет использования представленных в *Wikidata* знаний о лексике языков и представления содержания страниц в независимом от естественных языков виде. Существенной частью проекта является подпроект *Wikifunctions* – создания самими пользователями *Wiki*-страниц биб-

лиотеки программных функций. По словам авторов проекта он позволит каждому создавать, поддерживать, запускать и ссылаться на функции библиотеки *Wikifunctions*, а также пользователям, незнакомыми с языками программирования, пользоваться возможностями библиотеки для обработки данных из *Wikidata* и получать ответы на естественном языке. Авторы [32, с. 40] утверждают: «Ожидается, что Викифункции и Абстрактная Википедия будут стимулировать ряд направлений исследований в области представления знаний, генерации естественного языка, систем совместной работы и автоматизированной разработки программного обеспечения. Наличие большого каталога функций будет полезно для многих задач» . Слабо обсужденными в этом проекте остаются вопросы модульности и создания среды для формирования и использования формальных языков (а не только естественных), хотя в проекте *Wikifunctions* без модульности будет трудно обойтись, так как программные функции обычно контекстны.

ЗАКЛЮЧЕНИЕ

В настоящей статье обсуждается развитие графовых средств представления онтологий методами алгебраического подхода к представлению знаний и программированию. В качестве вершин и ребер графа предлагается использовать не только уникальные идентификаторы объектов в Веб с именами, но и термы – правильно построенные выражения из таких имен и имен операций, которые из классов, свойств, элементов классов и типов данных строят новые классы, свойства и элементы классов.

В алгебраическом подходе к моделированию онтологий в Веб предлагается использование следующих принципов:

- модульность при построении онтологий и создание библиотек онтологий способами, аналогичными созданию *Wikipedia*;
- каждая онтология разрабатывается в среде другой онтологии, которая предоставляет свои правила построения онтологий и языковые средства;
- язык представления онтологий и вопросов к онтологиям открытый и контекстный: в каждой онтологии пользователи могут вводить свои конструкции языка для функций и правил и определять их действия через функции и правила, доступные из текущей онтологии и ее среды;
- с каждой онтологией связывается ее алгебра и аппроксимация алгебры онтологии (внутреннее представление онтологии), по которой выполняются эффективные вычисления проверки синтаксической и семантической правильности выражений, заданий онтологии и запросов к ней, а также условий применимости операций и правил;
- в графе, соответствующем спецификации онтологии, вершины и ребра являются термами – выражениями, задающими классы, элементы классов и свойства операциями из других классов, элементов и свойств;
- вычисления в онтологии выполняются с использованием ее внутреннего представления по правилам переписывания и правилам изменения внутреннего представления, заданным или доступным в контексте данной онтологии, но описанным в других онтологиях.

Предлагаемые принципы могли бы лечь в основу разработки стандарта алгебраического языка представления онтологий в Веб: *Algebraic Web Ontology Language* (*Algebraic OWL*).

СПИСОК ЛИТЕРАТУРЫ

1. RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014. – URL: https://www.w3.org/ TR/2014/REC-rdf11-concepts-20140225/Overview.html (дата обращения: 30.07.2022).
2. RDF Schema 1.1, W3C Recommendation 25 February 2014. – URL: https://www.w3.org/TR/rdf-schema/ (дата обращения: 30.07.2022).
3. OWL 2 Web Ontology Language, Structural Specification and Functional-Style Syntax (Second Edition), W3C Recommendation 11 December 2012. – URL: https://www.w3.org/TR/owl-syntax/ (дата обращения: 30.07.2022).
4. OWL 2 Web Ontology Language, New Features and Rationale (Second Edition), W3C Recommendation 11 December 2012. - URL: https://www.w3.org/TR/2012/REC-owl2-new-features-20121211/ (дата обращения: 30.07.2022).
5. RDF-star and SPARQL-star, Final Community Group Report 17 December 2021. – URL: https://www.w3.org/2021/12/rdf-star.html (дата обращения: 30.07.2022).
6. Wikidata: SPARQL tutorial. – URL: https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial (дата обращения: 30.07.2022).
7. Bidoit M., Mosses P.D. CASL User Manual: Introduction to Using the Common Algebraic Specification Language // Lecture Notes in Computer Science.2900 (IFIP Series). – Springer, 2004.
8. Мена А. Изучаем Haskell. Библиотека программиста. – Санкт-Петербург: Питер, 2015.
9. Durán F., Eker S., Escobar S., Martí-Oliet N., Meseguer J., Rubio R., Talcott C.L. Programming and symbolic computation in Maude // Journal of Logical and Algebraic Methods in Programming. – 2020. – Vol. 110. – P. 1-100.
10. Clavel M., Duran F., Eker S., Escobar S., Lincoln P., Martı-Oliet N., Meseguer J., Rubio R., Talcott C. Maude Manual (Version 3.2.1). – 2022. – URL: http://maude.cs.illinois.edu/w/images/6/65/Maude-3.2.1-manual.pdf (дата обращения: 30.07.2022).
11. Бениаминов Е.М., Лапшин В.А. Уровни представлений онтологий, языки, математические модели и проект Веб-сервера онтологий в стиле Веб 2.0 // Научно-техническая информация. Сер 2. – 2012. – № 3. – С. 1-10.
12. Бениаминов Е.М. Работа в системе коллективного формирования библиотек онтологий ЭЗОП (Руководство пользователя). – Москва: РГГУ (препринт), 2015. – URL: http://beniaminov.rsuh.ru/User_guideEzop.pdf (дата обращения: 30.07.2022).
13. SWRL: A Semantic Web Rule Language, Combining OWL and RuleML. W3C Member Submission

21 May 2004. – URL: https://www.w3.org/Submission/SWRL/ (дата обращения: 30.07.2022).
14. RIF Overview (Second Edition). W3C Working Group Note 5 February 2013. – URL: https://www.w3.org/TR/rif-overview/ (дата обращения: 30.07.2022).
15. Church A., Rosser J.B. Some properties of conversion // Transactions of the American Mathematical Society. – 1936. – Vol. 39, № 3. – P. 472–482.
16. Ontology Engineering with Ontology Design Patterns: Foundations and Applications (Studies on the Semantic Web) / eds. P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, V. Presutti // IOS Press. – 2016. – Vol. 25.
17. Ontology Design Pattern. – URL: http://ontologydesignpatterns.org/wiki/ (дата обращения: 30.07.2022).
18. Shimizu C., Hammar K., Hitzler P. Modular Ontology Modeling // Semantic Web. – 2022. – Vol. Pre-press, № Pre-press. – P. 1-31. – URL: https://www.semantic-web-journal.net/system/files/swj2886.pdf (дата обращения: 30.07.2022).
19. Modular Knowledge 2022, 1st Workshop on Modular Knowledg, @ ESWC 2022, Hersonissos, Greece, May 29, 2022. – URL: https://mk2022.fbk.eu/home (дата обращения: 30.07.2022).
20. Goguen J. (home page). – URL: https://cseweb.ucsd.edu/~goguen/ (дата обращения: 30.07.2022).
21. Goguen J. Information Integration, Databases and Ontologies. – URL: https://cseweb.ucsd.edu/~goguen/projs/data.html (дата обращения: 30.07.2022).
22. CoFI: The Common Framework Initiative for algebraic specification and development of software. – URL: https://homepages.inf.ed.ac.uk/dts/cofi/ (дата обращения: 30.07.2022).
23. Mossakowski T., Codescu M., Neuhaus F., Kutz O. The distributed ontology, modelling and specification language – DOL // The Road to Universal Logic–Festschrift for 50th birthday of Jean-Yves Beziau, Vol. II / eds. A. Koslow, A. Buchsbaum. – Birkhäuser: Studies in Universal Logic, 2015.
24. Object Management Group. The distributed ontology, modeling, and specification language (DOL). – 2016. – OMG standard. – URL: http://www.omg.org/spec/DOL (https://www.omg.org/spec/DOL/1.0/PDF) (дата обращения: 30.07.2022).
25. Stephen Wolfram. – URL: https://www.stephenwolfram.com/ (дата обращения: 30.07.2022).
26. Wolfram (страница компании). – URL: https://www.wolfram.com/ (дата обращения: 30.07.2022).
27. Система Mathematica. – URL: https://www.wolfram.com/mathematica/ (дата обращения: 30.07.2022).
28. Система WolframAlpha. – URL: https://www.wolframalpha.com/ (дата обращения: 30.07.2022).
29. Система Wikidata. – URL: https://www.wikidata.org/wiki/Wikidata:Main_Page (дата обращения: 30.07.2022).
30. Hernández D., Hogan A., Riveros C., Rojas C., Zerega E. Querying Wikidata: Comparing SPARQL, Relational and Graph Databases / eds P. Groth, et al. // ISWC 2016, Part II. – LNCS. – Vol. 9982. – P. 88-103. – Springer, Charm, 2016. – URL: https://aidan-hogan.com/docs/wikidata-sparql-relational-graph.pdf (дата обращения: 30.07.2022).
31. Проект: Abstract Wikipedia. – URL: https://meta.wikimedia.org/wiki/Special:MyLanguage/ Abstract_ Wikipedia (дата обращения: 30.07.2022).
32. Vrandečić D. Viewpoint: Building a multilingual Wikipedia // Communications of the ACM. – 2021. – Vol. 64, № 4. – P. 38-41. – URL:https://cacm.acm.org/magazines/2021/4/251343-building-a-multilingual-wikipedia/fulltext (дата обращения: 30.07.2022).

Материал поступил в редакцию 01.08.22.

Сведения об авторе

БЕНИАМИНОВ Евгений Михайлович – доктор физико-математических наук, профессор, заведующий кафедрой Российского государственного гуманитарного университета, Москва.
e-mail: ebeniamin@yandex.ru