

Анализ мотивации, целей и подходов проекта унификации языков на правилах

Л.А.Калиниченко¹, С.А.Ступников¹

¹Институт проблем информатики РАН
Россия, г. Москва, 117333, ул. Вавилова, 44/2
{leonidk, ssa}@ipi.ac.ru

Аннотация. Работа посвящена анализу стандарта W3C RIF (Rule Interchange Format), ориентированного на обеспечение интероперабельности разнообразных систем на правилах введением расширяемого семейства унифицированных языков (диалектов) на правилах, позволяющих создавать сохраняющие семантику отображения в диалекты языков различных систем на правилах. Для определения мотивации проекта RIF в работе дан краткий обзор развития и применения языков и систем на правила в областях представления знаний, дедуктивных баз данных, логических моделей рассуждений. В работу включен анализ различных семантик логических языков на правилах, оказавших влияние на конструкцию RIF. Рассмотрены основные классы применений интероперабельных программ на правилах, на основе которых были выработаны требования к RIF. Рассмотрены основные решения, принятые в проекте RIF.

Ключевые слова: унификация языков, расширяемость языков, сохраняющие семантику отображения языков, системы логического программирования, системы на активных правилах, производственные системы, представление знаний, дедуктивные базы данных, логические модели рассуждений, стратифицированная семантика, стабильная модель логической программы, хорошо обоснованная семантика, диалекты RIF, каркас RIF.

1 Введение

Многообразие информационных технологий (ИТ) и их воплощений в конкретных ИТ продуктах проявляется в многообразии языков, предназначенных для спецификации предметных областей, программ, схем баз данных, онтологий, интерфейсов информационных ресурсов (ИР), реализованных в рамках определенной ИТ, и пр. Число таких

языков, их разнообразие со временем быстро растет, порождая сложные проблемы интеграции и интероперабельности разноязыких ИР. Стандартизация языков несколько ограничивает разнообразие, однако число стандартов остается большим, а конкретные реализации одного и того же стандарта зачастую остаются несовместимыми

Примерами классов подобных языков являются языки реляционных и объектных баз данных, языки онтологического моделирования, языки представления слабо структурированных, графовых, мультимедийных данных, языки представления баз знаний, языки логического программирования, дедуктивные языки запросов к базам данных, языки спецификации процессов (потоков работ), языки спецификации интерфейсов программных ИР для обеспечения их (ИР) интероперабельности, языки со специализированной семантикой (например, для выражения темпоральных, пространственных моделей), языки для определения нечетких, вероятностных представлений, языки концептуального моделирования и метамоделей, и многие другие.

Проблемы унификации языков с целью нивелирования различий их синтаксиса и семантики исследуются давно. Так, например, язык IDL был разработан OMG в качестве стандарта языка спецификации интерфейсов объектов или программных компонентов, полученных в результате применения различных языков и систем программирования. Язык YAWL, основанный на сетях Петри, ориентирован на унифицированное представление разнообразных образцов поведения, представимых в различных языках описания процессов. Язык TSQL ориентирован на представление темпоральных реляционных баз данных.

Среди подходов к унификации языков особое место занимает концепция расширяемых языков, в которых фиксируется некоторое ядро, позволяющее унифицировать некоторую совокупность простых языков в определенном классе, и над таким ядром надстраиваются расширения, каждое из которых вместе с ядром является результатом сохраняющего семантику отображения некоторого исходного языка (языков) [1]. Развитым проектом расширяемого языка является язык СИНТЕЗ [2, 3], сопровождаемый методами и средствами построения его расширений. Основным средством поддержки процесса построения расширений языка-ядра является Унификатор информационных моделей [4], позволяющий конструировать расширения языка СИНТЕЗ и отображения конкретных исходных языков в такие расширения. При этом сохранение семантики операторов трансформации состояний и поведения в исходном языке в их отображении в язык СИНТЕЗ основано на принципе уточнения¹ [5]. Таким образом, можно конструировать доказательно правильные расширения языка и отображения конкретных языков в язык СИНТЕЗ.

¹ Говорят, что спецификация *A* уточняет спецификацию *B*, если систему, удовлетворяющую *A*, можно использовать вместо системы, удовлетворяющей *B*, и при этом пользователь не замечает этой замены.

Эти методы применялись к отображению в язык СИНТЕЗ языков баз данных, процессных языков, онтологических языков, языков спецификации объектных интероперабельных компонентов, и др.

Вместе с тем, за исключением языков манипулирования данными в базах данных, вопросы унификации языков программирования не рассматривались вообще. Поэтому инициатива W3C, известная как RIF (Rule Interchange Format), заслуживает в контексте работ по унификации языков особого внимания [6, 7]. Использование и развитие языков на правилах для логического программирования, дедуктивных баз данных, представления знаний, создания интеллектуальных информационных систем продолжается уже более тридцати лет. Технология языков на правилах созревала в этот период в результате теоретических исследований, практического и коммерческого применения таких языков. В частности, накопленный в этой области опыт существенно превосходит опыт в области дескриптивных логик, активно развиваемых для онтологического моделирования. Для эффективного использования потенциала подходов, основанных на правилах, сообщества применения языков на правилах в области искусственного интеллекта, в бизнесе, в Семантическом Вебе организовали проект по выработке решений по унифицированному, интероперабельному использованию спецификаций, представленных на различных языках на правилах (таких языках, как [8]). Применение результатов этого проекта должно быть общеупотребительным и не ограничиваться Семантическим Вебом. Идея RIF состоит в создании расширяемого унифицированного языка на правилах, обеспечивающего возможность создания сохраняющих семантику отображений в него различных языков на правилах. Такой унифицированный язык представляется как семейство диалектов, которые имеют общее ядро (корневой диалект) и совокупность расширяющих его диалектов, образующих ориентированный граф без циклов. Каждое ребро графа (отношение расширения диалектов) отправляется от более простого к расширенному диалекту. Для сравнения, при построении аксиоматических расширений языка СИНТЕЗ отношение расширения интерпретировалось как включение множества аксиом более простого языка во множество аксиом более сложного языка [1]. Итак, в RIF унифицированный язык на правилах представляется как семейство унифицированных языков (диалектов).

В RIF по отношению к диалекту каждая система программирования (система вывода) на правилах может выступать в двух независимых ролях – роли поставщика и роли потребителя. Первая роль означает, что система на правилах обеспечивает преобразование собственных программ в программы на унифицированном диалекте. Вторая роль означает, что система на правилах может воспринимать программы на диалекте и преобразовывать их в программы на собственном языке системы. Таким образом, для полновесного включения каждого языка на правилах в

совокупность интероперабельных языков достаточно снабдить соответствующую систему программирования двумя сохраняющими семантику преобразователями – из собственного языка в диалект (роль поставщика) и из диалекта в собственный язык (роль потребителя).

Для работы по проекту RIF в 2005 г. была образована рабочая группа W3C RIF WG [9] с целью выработки «обменного формата» правил – так кратко именуется семейство определяемых RIF унифицированных диалектов языка на правилах. Изначально было декларировано, что целью RIF не является обеспечение единственного языка, охватывающего черты всех известных языков на правилах. Различные средства различных языков и систем на правилах зачастую не совместимы. Поэтому и была принята концепция диалектов RIF, обеспечивающая возможность обмена модулями правил между различными системами на правилах. По замыслу проекта, для правил, созданных в рамках некоторого приложения, должна быть обеспечена возможность их публикации, использования совместно с другими правилами, повторного использования в других приложениях и других системах на правилах.

Кроме того, важной задачей RIF WG было обеспечение возможности совместной работы диалектов RIF и существующих стандартов Семантического Веба – таких как RDF и OWL, не совместимых с большинством существующих языков на правилах. В частности, важно показать, что существует возможность совместной работы дескриптивных логик и систем на правилах:

- сформировать общий базис для онтологических языков на правилах и на дескриптивной логике с целью достижения их интероперабельности;
- проанализировать пересечение этих двух формализмов представления знаний для понимания того, какой выразительности можно достичь при их объединении (комбинации);
- обеспечить использование машин на правилах как масштабируемых служб рассуждений в онтологиях, определяемых на таком пересечении.

В октябре 2009 г. был завершен важный этап RIF – публикация документов спецификации RIF [7] в качестве кандидатов для стандартизации W3C:

1. RIF Overview
2. RIF Core Dialect
3. RIF Basic Logic Dialect
4. RIF Framework for Logic Dialects
5. RIF RDF and OWL Compatibility
6. RIF Datatypes and Built-Ins 1.0
7. RIF Production Rule Dialect
8. RIF Test Cases
9. RIF Combination with XML data
10. OWL 2 RL in RIF

В июне 2010 г. документы 2–6 были приняты в качестве стандарта W3C. Создание общепринятой спецификации RIF – сложная задача. Выполненная RIF WG работа впечатляет глубиной замысла, масштабом, точностью разработанных, тщательно формализованных спецификаций. Предложенная концепция окажет существенное влияние на развитие декларативных языков и их применение в различных областях, расширяя тем самым существующие пределы использования систем на правилах. Результаты, полученные RIF WG, заслуживают детального анализа и изучения. Настоящая работа посвящена именно такому анализу. Статья организована следующим образом. В разделе 2 дан краткий анализ истории развития и применения языков и систем на правилах, включая области представления знаний, дедуктивных баз данных, логических моделей рассуждений. В разделе 3 приведен обзор различных семантик логических программ на правилах, таких как стратифицированная семантика, семантика стабильной модели логической программы, хорошо обоснованная семантика. В разделе 4 рассмотрены основные классы применений интероперабельных программ на правилах, на основе которых были выработаны требования к RIF. В разделе 5 рассмотрены основные решения, принятые в проекте RIF.

2 Языки на правилах и их применения: краткий исторический экскурс

Настоящий раздел является введением в контекст, в рамках которого формировался проект RIF. При написании раздела использовались материалы известных обзоров симбиотического развития логики и программирования, логики и баз данных, а также применения полученных результатов для представления знаний и программирования различных моделей рассуждений в приложениях [10, 11, 12, 13].

2.1 Программы на правилах, классы применений

Логическое программирование характеризуется тремя основными классами применений: в качестве универсального языка программирования, языка баз данных, языка представления знаний. В качестве языка программирования оно позволяет представить и вычислить любую вычислимую функцию. Как язык баз данных, оно обобщает реляционные базы данных, позволяя наряду с фактами представлять правила общего вида. Наконец, как язык представления знаний, оно является немонотонной логикой, которую можно использовать для рассуждений по умолчанию (подробнее о моделях рассуждений см. раздел 2.4).

Логические программы представляют собой множества условных высказываний вида:

if B_1 and ... and B_n then H

в которых *следствие H* представляет собой атомарную формулу, а *условия B_i* являются *литералами*, представляющими собой атомарные формулы или их отрицания. Все переменные неявно связаны квантором всеобщности, располагаемым перед условным высказыванием. Условные высказывания в логических программах называются *клаузами*. *Факты* являются клаузами специального вида, в которых $n = 0$ (нет условий) и нет переменных. Клаузы, не являющиеся фактами, называются *правилами*. *Цели* (или *запросы*) представляют собой конъюнкции литералов подобно условиям в клаузах. Однако все переменные в них неявно связаны квантором существования и задачей цели является отыскание означиваний переменных цели, при которых цель приобретает истинное значение.

При *обратных рассуждениях* (от следствий к условиям) условные высказывания рассматриваются как процедуры редуцирования цели: чтобы показать, что H , нужно разрешить *B_1 and ... and B_n* .

Поскольку условные высказывания в логических программах рассматриваются в таком обратном порядке, обычно они так и записываются:

H if B_1 and ... and B_n

так что обратные рассуждения становятся эквивалентными «прямому связыванию» или «прямому сцеплению» в направлении, в котором записывается условное высказывание. В синтаксисе языка Prolog клауза выглядит так:

$H :- B_1, \dots, B_n$

В таком виде клаузы можно трактовать либо декларативно, как условные высказывания в обратной записи, либо процедурно, как процедуры редуцирования цели, исполняемые в прямом направлении.

Позитивные, атомарные цели и подцели разрешаются в процессе обратных рассуждений, цели и подцели с отрицанием вида *not G* , где G – атомарное предложение, разрешаются при трактовке отрицания как неудачи, согласно которой *not G* удовлетворяется тогда и только тогда, когда обратные рассуждения с подцелью G терпят неудачу. Отрицание как неудача превращает логическое программирование в немонотонную логику.

Цели и условия клауз могут быть обобщены, так что вместо конъюнкций литералов могут быть использованы произвольные формулы логики первого порядка.

2.2 Применение логики для представления знаний и решения задач

В этом разделе кратко рассмотрено развитие подходов, основанных на применении языков на правилах для представления знаний и решения задач [10]. Рассматриваемые подходы важны для понимания мотивации проекта RIF.

2.2.1 Развитие эвристических и формальных подходов

В 1957 г. Logic Theorist Ньюела стал первой программой, с помощью которой были доказаны 38 теорем из 52, приведенных в Главе 2 Principia Mathematica. При этом применялись три правила вывода: обратные рассуждения, прямое и обратное связывание. Logic Theorist явился предтечей GPS, универсального решателя задач.

МакКарти в 1958 в Advice Taker использовал формальную логику для представления знаний и доказательства теорем в процессе решения задач. Разработанный подход к доказательству теорем привел к созданию вопросно-ответных систем на основании полных методов доказательства математической логики.

Принцип резолюции был разработан Робинсоном в 1965 г., следуя формально-логической традиции с ориентацией сугубо на математические применения.

К концу 60-х годов 20 века в подходах к искусственному интеллекту наметились две основных тенденции: эвристический подход, связываемый главным образом с продукционными системами и стремлением к представлению специальных знаний конкретных предметных областей, и формальный подход, применявшийся в системах, основанных на принципе резолюции, в рамках которого подчеркивалась важность универсальных методов решения задач, не зависящих от предметной области.

2.2.2 Процедурные представления знаний и логическое программирование

Процедурный подход как альтернатива логическому находит выражение в системах Planner и micro-Planner (MIT). В 1971 г. Виноградом на основе системы micro-Planner был реализован диалог на естественном языке для простой предметной области.

Р. Ковальски в сотрудничестве с А.Кольмероз предпринял попытку повторить в логике реализацию системы Винограда на основе принципа

резолуции. Это привело к реализации процедурной интерпретации Хорновских клауз в 1974 г. и к разработке языка программирования Prolog в начале 70-х в Марсельском университете А. Кольмероз и его сотрудниками на основе теоретических работ Р.Ковальски (программа доказательства теорем, которая включала интерпретатор Хорновских клауз Ковальски). Выполнение логической программы состоит в попытке решить задачу, т.е. доказать целевое утверждение G , используя предположения A , заданные в логической программе.

2.2.3 Немонотонные рассуждения

В середине 70х годов прошлого века Марвин Минский (MIT) предпринял очередное наступление против логического подхода, предложив фреймы для представления знаний по умолчанию без необходимости строгой и точной спецификации исключений.

Логическое сообщество ответило немонотонными логиками, включая очерчивание МакКарти (1980 г.), логику умолчания Рейтера (1980 г.), аутоэпистемическую логику Мура (1985 г.), отрицание как неудачу в логическом программировании, обоснованное Кларком (1978 г.).

В этот же период было замечено, что рассуждение по умолчанию можно интерпретировать как форму абдуктивных рассуждений. Основываясь на этих результатах, Ковальски показал, что отрицание как неудача в логическом программировании можно также интерпретировать в таких терминах. Дунг (2006 г.) показал, что большинство немонотонных логик может быть интерпретировано в терминах логики аргументации на основе абдуктивных гипотез (более подробно об этом – в разделе 2.4).

2.2.4 Интеллигентные агенты

Ранние работы по созданию логически базированных агентных архитектур относятся к началу 90-х годов 20 века. Характерным для этих работ является их формулирование в терминах убеждений, желаний, намерений. Не смотря на то, что их формальные спецификации были сформулированы в модальной логике, их реализации осуществлялись в процедурных языках, являющихся расширением продукционных систем. Агентные модели логического программирования были разработаны Ковальски в период 1999 – 2006 г. для согласования возможности использования логики для спецификации агентов с ее реализацией. Вместо использования модальностей для установления различий между убеждениями и желаниями, абдуктивные модели логического программирования (ALP) используют специальную трактовку различий между данными и ограничениями целостности в базах данных для интерпретации убеждений как данных и желаний (или целей) как ограничений целостности.

В традиционных базах данных ограничения целостности используются пассивно, чтобы воспрепятствовать неправильным изменениям данных. В абдуктивном логическом программировании они играют роль, подобную ограничениям целостности в активных базах данных, в которых они, с одной стороны, препятствуют неправильным изменениям, а с другой – выполняют корректирующие действия для поддержания целостности базы данных. Продукционные правила, используемые как ассоциации стимул – реакция, могут рассматриваться как ограничения целостности такого вида.

2.2.5 Продукционные системы

Продукционные системы первоначально были разработаны логиком Э. Постом в качестве математической модели вычислений и позднее были использованы А. Ньюелом как модель решения задач человеком. Продукционные системы ограничивают представление знаний совокупностью фактов как атомарных предложений и правил условие – действие в синтаксисе условных высказываний, не имеющих логической семантики. Они позволяют использовать при рассуждениях разновидность *modus ponens*, называемую прямым связыванием.

Продукционные правила выполняются в цикле, реализующем чтение входного факта, применение прямого связывания (от условий к действиям) для определения совпадения факта с одним из условий продукционного правила, проверку остальных условий правила, и затем порождение действия правила в качестве кандидата действий для исполнения. Если подобным образом возбуждается более одного правила, вступает в действие процедура разрешения конфликтов для принятия решения о том, какие действия подлежат выполнению.

По сравнению с классической логикой, которой соответствует как декларативная теоретико-модельная семантика так и различные процедуры доказательства, продукционные системы не имеют декларативной семантики вообще. Продукционные правила имеют вид логических импликаций, не обладая семантикой импликаций.

Различие между продукционными системами и логическим программированием имеет две стороны. Существуют технические отличия, такие как то, что продукционные правила выполняются в прямом направлении, тот факт, что следствием продукционных правил являются действия, а также то, что продукционные правила встраиваются в цикл, в котором производится разрешение конфликтов для выбора из имеющихся действий-кандидатов. Существуют также культурные различия, отражающие традиции в отношении к вопросам точности и полноты, а также различия в акцентах, отдающих предпочтение знаниям о конкретных предметных областях в противовес универсальным методам решения задач.

2.3 Дедуктивные базы данных

При написании настоящего раздела использованы работы [14, 12]. Зарождение дедуктивных баз данных можно обнаружить еще в работах по автоматическому доказательству теорем, и позднее – в работах по логическому программированию. Грин и Рафаэль (1968 г.) были первыми, кто распознал связь между доказательством теорем и дедукцией в базах данных. Ими был разработан ряд вопросно-ответных систем, демонстрирующих, что дедукция может быть систематически реализована в контексте баз данных. Другие ранние системы включали MRPPS, DEDUCE-2 и DADM. Одной из первых статей по обработке рекурсивных запросов была работа Минкера и Николаса, в которой впервые было описано как связанные рекурсивные запросы могут быть преобразованы в их нерекурсивные эквиваленты. В системе DADM в 1981 г. была показана важность установления различия между экстенциональной и интенциональной базами данных (EDB и IDB). В 1978 г. появились важные и в контексте дедуктивных баз данных работы Рейтера относительно предположения о замкнутости мира, статья Кларка об отрицании как неудаче, работа Минкера и Галлера по компиляции правил, работа Николаса и Язданьяна по проверке ограничений целостности.

В 1976 г. Ван Эмденом и Ковальски было показано, что наименьшая неподвижная точка логической программы на клаузах Хорна совпадает с ее наименьшей моделью Эрбрана. Тем самым были заложены строгие основания семантики логических программ и, в особенности, дедуктивных баз данных, в которых вычисление неподвижной точки служило операционной семантикой (по крайней мере, при вычислении снизу вверх). Хеншен и Накви в 1984 г. предложили один из первых эффективных методов вычисления рекурсивных запросов в контексте баз данных, что было существенным прогрессом по сравнению с использовавшимися до тех пор стратегиями, основанными на принципе резолюции (неудовлетворительном для больших баз данных) или простых методах вычисления (подобных наивному методу вычисления неподвижной точки Чанга).

Область дедуктивных баз данных, в особенности, область обработки рекурсивных запросов, становится быстро развивающейся во второй половине 80х – начале 90х годов 20 века под влиянием исследований по компьютерам пятого поколения. Особенно интересными являются проекты Nail! в Стенфорде, LDL (1984 г.), проект LDL++ (1990 г.) в MCC в Остине, а также проект по дедуктивным базам данных в ECRC, внесшие существенный теоретический вклад и завершившиеся созданием экспериментальных систем. Дальнейший прогресс в этой области был отмечен рядом работ и специальных конференций, включая конференции по дедуктивным и объектно-ориентированным базам данных (DOOD) в

1989 г. (Киото), 1991 г. (Мюнхен), 1993 (Феникс), 1995 (Сингапур), 1997 (Монтре), 2000 (Лондон).

Дедуктивные базы данных заимствуют значительное число концепций от логического программирования, так, правила и факты, представляемые в языке дедуктивных баз данных Datalog, похожи на представления программ на языке Prolog. Вместе с тем, существует ряд существенных различий между дедуктивными базами данных и логическим программированием. Для последних характерно следующее:

- зависимость от порядка правил в программах и процедурность²: в языке Prolog выполнение программы зависит от порядка правил в программе и даже от порядка частей правила; этими свойствами языка программисты пользуются для получения более эффективных программ. В языках баз данных (таких как SQL или Datalog) выполнение программ не зависит от порядка правил и фактов в программе;
- специальные предикаты: в языке Prolog программисты могут влиять на процедурное выполнение программы непосредственно с помощью специальных предикатов, таких как *cut*. В дедуктивных базах данных нет аналогов этому;
- ориентация на покортежную обработку, в то время как дедуктивные базы данных оперируют множествами.

В работе [11] дан обзор семантик различных языков логического программирования, включая стратифицированные программы, хорошо обоснованную и стабильную модели логических программ. Эти семантики обсуждаются в разделе 3, поскольку они в значительной степени оказали влияние на решения, принятые в проекте RIF.

В таблице 1 [12] сопоставлены различные системы дедуктивных баз данных по следующим характеристикам:

1. *Рекурсия*. В большинстве систем поддерживается рекурсия общего вида. Однако, в ряде систем возможности ограничиваются линейной рекурсией, или даже узкими возможностями поиска в графах, таких как построение транзитивного замыкания.
2. *Отрицание*. Большинство систем допускает в правилах подцели с отрицанием. Таким программам может соответствовать множество минимальных неподвижных точек, каждую из которых можно было бы считать моделью программы. Системы должны выбирать из них наиболее предпочтительную модель, которая и считается ответом на запрос.
3. *Ограничения целостности*. Некоторые дедуктивные системы позволяют использовать правила, трактуемые как ограничения целостности. В отличие от IDB предикатов, правила, выступающие в

² Следует заметить, что языки и системы логического программирования, имеющие хорошо обоснованную семантику, или семантику стабильной модели, являются чисто декларативными, так что это различие в дальнейшем было устранено.

роли ограничений целостности, не могут быть нарушены при изменении данных.

4. *Интерфейсы.* Большинство систем допускают взаимодействие с несколькими другими языками и системами. Некоторые соединения допускают встраивание обращений к дедуктивной базе данных из других языков, тогда как другие соединения допускают вызов других программ из дедуктивной системы. В таблице не различается направление вызова. Некоторые системы используют внешние языковые интерфейсы, которые могут быть применены для специализации системы в различных приложениях (например, путем добавления новых типов данных, определения способа реализации отношений, и т.д.). В таблице такие возможности обозначаются как расширяемость (extensibility).

Таблица 1 Дедуктивные системы баз данных [RU 93]

Название	Разработчик	Рекурсия	Отрицание	Ограничения	Интерфейсы
Aditi	U. Melbourne	Общая	Стратифицированное	Нет	Prolog
COL	INRIA	?	Стратифицированное	Нет	ML
Concept-Base	U. Aachen	Общая	Локально стратифицированное	Есть	C, Prolog
CORAL	U. Wisconsin	Общая	Модульно стратифицированное	Нет	C, C++,
EKS-VI	ECRC	Общая	Стратифицированное	Есть	Persistent Prolog
LogicBase	Simon Fraser U.	Линейная	Стратифицированное	Нет	C, C++, SQL
DECLARE	MAD Intelligent Systems	Общая	Локально стратифицированное	Нет	C, Lisp
X4	U. Karlsruhe	Общая, только бинарные предикаты	Нет	Есть	Lisp
Hu+	U. Toronto	Запросы - пути	Стратифицированное	Нет	Prolog, LDL, CORAL, Smalltalk
LDL LDL++	MCC	Общая	Локально стратифицированное	Нет	C, C++, SQL
LOGRES	Polytechnic of	Линейная	Инфляцион	Есть	INFORMIX

	Milan		ная семантика		
LOLA	Technical U. Munich	Общая	Стратифицированное	Есть	TransBase (SQL)
Glue-Nail	Stanford U.	Общая	Хорошо обоснованное	Нет	None
Starburst	IBM Almaden	Общая	Стратифицированное	Нет	Extensible
XSB	SUNY Stony Brook	Общая	Хорошо обоснованное	Нет	C, Prolog

2.4 Логические модели рассуждений

Если первоначально в работах по языкам логического программирования преобладало стремление достижения декларативности программ, то в дальнейшем доминирующим было намерение поддержать средствами языков различные модели логических рассуждений. В настоящем разделе будут перечислены модели логических рассуждений, оказавшие и продолжающие оказывать существенное влияние на развитие языков логического программирования.

Прежде всего напомним понятие логической монотонности. Говорят, что система представления знаний *логически монотонна*, если при расширении множества посылок множество следствий также расширяется (не происходит отказа от уже полученных заключений).

С другой стороны, немонотонные рассуждения означают, что следствия, выведенные из посылок, могут стать более не выводимыми при расширении множества посылок. Конечно, на практике рассуждения, вообще говоря, являются немонотонными. Монотонные рассуждения составляют лишь частный случай. Большинство систем поддержки логических рассуждений являются немонотонными (например, в области принятия решений, обработки исключительных ситуаций, аргументации, Байесовских стратегий, статистического вывода, и пр.).

Далее перечисляются различные виды (модели) немонотонных рассуждений.

Абдуктивные рассуждения [15] составляют процесс получения наиболее вероятных объяснений известных фактов. Абдуктивная логика должна быть немонотонной, поскольку наиболее вероятные объяснения не обязательно являются правильными. Абдукция - познавательная процедура принятия гипотез. Согласно Пирсу, методология науки должна пониматься как взаимодействие *абдукции*, осуществляющей принятие объяснительных правдоподобных гипотез; *индукции*, реализующей

эмпирическое тестирование выдвинутых гипотез; и *дедукции*, посредством которой из принятых гипотез выводятся следствия. Абдуктивный вывод представляется как вид автоматизированного *правдоподобного рассуждения*, при котором следствие необязательно истинно для каждой интерпретации, в которой истинны посылки.

Ковальски определяет подход к абдуктивному логическому программированию (ALP), целью которого является согласование продукционных правил и логики в унифицирующем агентно-базированном каркасе. В этом каркасе логические программы выглядят как совокупности условных высказываний (предложений), используемых для представления убеждений агента и имеющих описательный характер. Абдукция используется как для представления гипотез, объясняющих наблюдения агента, так и для действий, с помощью которых можно обеспечить достижение целей агента. Абдуктивные гипотезы могут иметь связанные с ними вероятности. ALP-базированный каркас агента вводит цели и убеждения в цикл наблюдений-размышлений-решений-действий, подобный циклу в продукционных системах.

Рассуждения на основе умолчания [16, 17, 18]. Примером предположения, основанного на умолчании, является соображение, что обычно птица может летать. В результате, если известно, что конкретный представитель животного мира является птицей, и больше ничего не известно, можно заключить, что он может летать. Однако, нужно иметь возможность отказаться от такого предположения на основе умолчания, если впоследствии выяснится, что этот представитель является пингвином. Этот пример показывает, что логика, моделирующая рассуждения по умолчанию, должна быть немонотонной. Логики, формализующие рассуждения по умолчанию, могут быть грубо разделены на две категории. К первой категории относятся логики, позволяющие выразить произвольные предположения на основе умолчания (логика умолчания [17], логика отмены заключений, логика аргументации, логика программирования множества ответов (ASP)). Ко второй категории относятся логики, формализующие конкретное предположение умолчания, согласно которому факты, о которых неизвестно, что они истинны, должны считаться ложными по умолчанию (предположение о замкнутости мира, очерчивание).

Предположение о замкнутости мира (CWA) есть допущение о том, что все то, о чем в данный момент неизвестно, что оно истинно, считается ложным. Логическая формализация этого допущения, предложенная Рейтером, имеет такое же название. Обратным этому допущению является предположение об открытости мира (OWA), утверждающее, что недостаток знания не имплицитно выводит о ложности факта.

Отрицание как неудача (NAF) связано с предположением о замкнутости мира, поскольку согласно ему, каждый предикат считается ложным, если нельзя доказать его истинность. Понимание необходимости рассуждения о неполной информации приводит к выводу, что в логических формулах наряду с отрицанием как неудачей (далее эта логическая связка будет обозначаться как *not*), требуется использование классического (или сильного) отрицания, обозначаемого далее как \neg . Следующий пример [19], иллюстрирующий различие между этими двумя видами отрицания, принадлежит Дж. МакКарти. Автобус может пересечь железную дорогу при условии, что нет приближающегося поезда. Правило

cross :- not train

не является адекватным воплощением этой идеи. Оно утверждает, что можно пересекать дорогу в отсутствии информации о приближающемся поезде. Предпочтительнее использовать: правила с сильным отрицанием вместо отрицания как неудачи:

cross :- ¬train

Это правило утверждает, что можно пересечь дорогу, если известно, что нет приближающегося поезда.

Комбинируя оба вида отрицания в одном правиле, можно выразить CWA. Например, правило

$\neg q(X, Y) :- not\ q(X, Y), p(X), p(Y)$

утверждает, что бинарное отношение q не связывает пару элементов p , если нет явной информации, что это имеет место.

Другой пример связан с формулированием закона инерции на основе здравого смысла. Всякое тело сохраняет состояние, в котором оно находится (Лейбниц, “An Introduction to a Secret Encyclopaedia”, 1679). Рейтер выразил это высказывание в логике умолчания ([20]):

$p(T+1) :- p(T), not\ \neg p(T+1), time(T)$

Подобно формализации CWA, это правило использует оба вида отрицания – отрицание как неудачу и сильное отрицание. Оно утверждает, что если p в момент времени T является истинным и нет информации о том, что p становится ложным в момент $T+1$, то p сохраняет истинное значение.

Таким образом, следуя [21], CWA для предиката p выражается правилом:

$$\neg p(X_1, \dots, X_n) :- \text{not } p(X_1, \dots, X_n)$$

Отношение p не выполняется для кортежа X_1, \dots, X_n , если нет явной информации об обратном.

Логическая программа с сильным отрицанием может включать правила с CWA для некоторых предикатов и оставлять другие предикаты в предположении OWA: На примере о том, является ли пингвин птицей, это выглядит следующим образом [22]:

$$\text{flies}(X) :- \text{bird}(X), \text{not } \text{excl}(X).$$

$$\text{bird}(X) :- \text{penguin}(X).$$

$$\text{excl}(X) :- \text{penguin}(X).$$

$$\text{bird}(\text{tweety}) :-.$$

$$\text{penguin}(\text{sam}) :-.$$

$$\neg \text{bird}(X) :- \text{not } \text{bird}(X).$$

$$\neg \text{penguin}(X) :- \text{not } \text{penguin}(X).$$

$$\neg \text{excl}(X) :- \text{not } \text{excl}(X).$$

$$\neg \text{flies}(X) :- \text{penguin}(X).$$

$$\neg \text{flies}(X) :- \neg \text{bird}(X).$$

Преимущества такого представления заключаются в том, что для удаления CWA о некотором предикате достаточно удалить явное CWA правило для этого предиката. Для введения информации о том, что *John* является раненой (wounded) птицей, и что раненые птицы являются исключением относительно их способности летать, обеспечивается следующими правилами:

$$\text{wounded } \text{bird}(\text{john}) :-.$$

$$\neg \text{wounded } \text{bird}(X) :- \text{not } \text{wounded } \text{bird}(X).$$

$$\text{bird}(X) :- \text{wounded } \text{bird}(X).$$

$$\text{excl}(X) :- \text{wounded } \text{bird}(X).$$

Теперь предположим, что наша информация о птицах, пингвинах, исключениях и раненых птицах неполна. Для выражения этого удалим явные CWA о птицах, пингвинах, исключениях и раненых птицах:

$$\text{flies}(X) :- \text{bird}(X), \text{not } \text{excl}(X).$$

$$\text{bird}(X) :- \text{penguin}(X).$$

$$\text{excl}(X) :- \text{penguin}(X).$$

$$\text{bird}(\text{tweety}) :-.$$

$$\text{penguin}(\text{sam}) :-.$$

$$\neg \text{flies}(X) :- \text{penguin}(X).$$

$$\neg \text{flies}(X) :- \neg \text{bird}(X).$$

$$\text{wounded } \text{bird}(\text{john}) :-.$$

$bird(X) :- wounded\ bird(X).$

$excl(X) :- wounded\ bird(X).$

На вопрос «если q – это птица, летает ли она?» предыдущая программа даст ответ «да». Но теперь у нас нет более CWA о том что q может быть пингином. Теперь требуются более консервативные рассуждения.

$flies(X) :- bird(X), not\ excl(X).$

$bird(X) :- penguin(X).$

$excl(X) :- not\ \neg penguin(X).$

$bird(tweety) :-$

$penguin(sam) :-$

$\neg penguin(X) :- \neg bird(X).$

$\neg flies(X) :- penguin(X).$

$\neg flies(X) :- \neg bird(X).$

$wounded\ bird(john) :-$

$\neg wounded\ bird(X) :- \neg bird(X).$

$bird(X) :- wounded\ bird(X).$

$excl(X) :- not\ \neg wounded\ bird(X).$

$bird(q).$

Логика очерчивания (circumscription) – это немонотонная логика, созданная Дж. МакКарти для формализации предположений здравого смысла, которые должны действовать по отношению к сущностям реального мира, если нет других указаний. В своей первоначальной формализации в логике первого порядка очерчивание сводилось к сужению экстенсиналов некоторых предикатов. Такое сужение подобно CWA, согласно которому, все то, о чем нельзя сказать, что оно истинно, ложно.

Аутоэпистемическая логика – это формальная логика для представления и рассуждения о знаниях [23]. В то время как логика высказываний может представлять только факты, аутоэпистемическая логика может выражать знания и их недостаточность относительно фактов. Аутоэпистемическая логика находит важные применения в логическом программировании. Одним из принципиальных вопросов при этом является определение семантики отрицания. Если в клаузе $p :- q, \neg r$ отрицание $\neg r$ интерпретируется как неудача программы доказать r , то отрицательный результат отражает свойство программы, ее неспособность доказать r , а вовсе не тот факт, что r в действительности ложно [GEL 87].

Логика отмены заключений (defeasible reasoning) – это разновидность рассуждений, основанных на суждениях, которые являются отменяемыми,

в отличие от неотменяемых суждений, характерных для дедуктивной логики. Рассуждения на основе отмены заключений не производят полной или окончательной трактовки утверждений, при которой устанавливается ошибочность или возможность пересмотра заключений. Способность к отмене заключений является важной в риторике, этике, теории решений, лингвистике.

Логика рассуждений о действиях и изменениях. Большая часть работ в этой области посвящена применениям ситуационного исчисления – формализма, предложенного Дж. МакКарти в 1968 году для описания действий, рассуждений о них и эффектов действий. Для преобразования плана поведения робота в исполняемую программу, достигающую с некоторой вероятностью фиксированной цели, вводится специальное логическое исчисление, основанное на ситуационной логике. Активно исследуются логики действий, применение модальных логик для рассуждений о знаниях и действиях.

Логика рассуждений с неопределенностью. Сюда относятся статистические методы обнаружения закономерностей в данных. Для этого используется первичная информация, содержащаяся либо в сетевых структурах, либо в базах данных. Под сетевыми структурами понимается в этом случае множество вершин и отношений на них, задаваемое с помощью ребер. Содержательно, ребра интерпретируются как причинные связи.

Следует заметить, что каждая из перечисленных в настоящем разделе моделей логических рассуждений является предметом серьезных исследований, результаты которых находят применение в различных областях. Эти исследования тесно связаны с развитием языков логического программирования с адекватной семантикой.

3 Разнообразие семантик логических программ

3.1 Минимальная модель программы. Стратифицированная семантика

Клауза Хорна представляет собой конъюнкцию литералов – атомарных формул (атомов) или их отрицаний с не более чем одним атомом в голове. Клауза Хорна с одним атомом в голове называется *определенной* (definite) клаузой. Клаузы Хорна играют основополагающую роль в логическом программировании. Клауза Хорна, не содержащая атома в голове, называется целевой клаузой. Резолюция

целевой клаузы с определенной клаузой, при которой образуется новая целевая клауза, составляет основу SLD резолюции (Selective Linear resolution with Definite clauses) – одного из способов интерпретации логических программ. Теоретико-модельная семантика подразумевается при реализации вывода в прямом направлении (снизу – вверх), или в обратном направлении (сверху – вниз)³.

Семантика логической программы определяется ее минимальной моделью. Программа, содержащая только атомы без отрицаний, называется позитивной. Каждой позитивной программе соответствует единственная минимальная модель, называемая наименьшей моделью. Определение наименьшей модели достигается оператором вычисления наименьшей неподвижной точки логической программы.

Позитивные логические программы позволяют реализовать декларативное моделирование при решении разнообразных задач. Во многих случаях, однако, требуется использование отрицания, нуждающегося в адекватной семантической интерпретации. Семантика языка Prolog представляет собой развитие от SLD к SLDNF резолюции на основе идеи отрицания как неудачи (NAF).

Реализация отрицания как неудачи в языке Prolog является проблематичной [25]: использование NAF в теле клауз приводит к сложностям в случае рекурсии при наличии циклических зависимостей предикатов (атомов) в правилах; поэтому современные системы логического программирования используют либо хорошо обоснованное отрицание по умолчанию [26], либо отрицание с семантикой, определяемой стабильной моделью [27].

Важным классом программ с отрицанием являются *стратифицированные программы*. Они обладают тем свойством, что можно установить порядок вычисления правил программы, при котором значения атомов с отрицанием могут быть предопределены. Иными словами, для вычисления тела правила, содержащего *not* $r(t)$, значение атома с отрицанием $r(t)$ должно быть определено. С этой целью предикаты вычисляются по слоям (стратам) программы снизу вверх. Этот подход работает, если в программе не возникает циклов с предикатами, содержащими отрицание.

Стратификация заключается в любом непротиворечивом присваивании номеров символам предикатов, гарантирующем, что существует однозначная формальная интерпретация логической программы. Говорят, что набор клауз вида $Q_1 \wedge \dots \wedge Q_n \wedge \neg Q_{n+1} \wedge \dots \wedge \neg Q_{n+m} \rightarrow P$ стратифицирован тогда и только тогда, когда существует стратификационная нумерация, удовлетворяющая следующим условиям:

³ При обратном рассуждении атомарный запрос унифицируется с головой правила и заменяется соответствующим экземпляром тела. При прямом рассуждении голова означенного экземпляра правила включается в множество следствий после того, как все атомы тела этого правила уже были включены в множество следствий.

1. Если предикат P позитивно выводим из предиката Q (т.е., P находится в голове правила, а Q позитивно входит в тело этого же правила), то стратификационный номер P должен быть большим или равным стратификационному номеру Q : $S(P) \geq S(Q)$.
2. Если предикат P выводим из предиката с отрицанием Q (т.е., P находится в голове правила, а Q входит с отрицанием в тело этого же правила), то стратификационный номер P должен быть большим стратификационного номера Q : $S(P) > S(Q)$.

Понятие стратифицированного отрицания позволяет получить эффективную операционную семантику стратифицированной программы на основе стратифицированной наименьшей неподвижной точки, вычисляемой итеративно. Оператор получения неподвижной точки применяется к каждой страте программы, двигаясь от страт с меньшими номерами к стратам с большими номерами.

В контексте IDB и EDB, EDB, представляющая собой набор фактов, получает номер (ранг) 0. Предикаты IDB, правила определения которых не включают отрицания, также имеют ранг 0. Предикаты IDB, чьи единственные отрицательные зависимости выражаются посредством предикатов ранга 0, получают ранг 1, и т.д. Стратифицируемость легко установить синтаксически анализом одной лишь IDB. Определение стратифицированной семантики реализуется индуктивно. После того, как все атомы с рангом меньшим k были классифицированы как позитивные или атомы с отрицанием, эти литералы используются для получения значений позитивных атомов ранга k и определения $\neg q$ для всех атомов q ранга k , которые не были выведены. Полученный результат называется стратифицированной моделью.

Стратифицированная семантика согласуется с семантикой хорошо обоснованной модели для всех рангов. Нестратифицируемым программам соответствует хорошо обоснованная семантика, или семантика стабильной модели.

3.2 Семантика стабильной модели логической программы

В логическом программировании существуют два принципиально разных взгляда на семантику программ, отражающих две философски различные точки зрения [28].

Первый подход отражает стремление сохранить единственную модель программы, даже для проблемных классов программ с отрицанием. Этого можно достичь, определяя надлежащим образом выбор единственной модели среди всех возможных моделей программы. Наиболее популярная семантика в этом подходе основана на *хорошо обоснованной модели*.

Второй подход характеризует противоположная точка зрения – соотнесение программе множества моделей, отбрасывая «догматическое»

требование единственности модели. В общем случае, считается, что одной программе может соответствовать множество совместимых с ней сценариев получения модели. В рамках этого подхода говорят о генерации множества *стабильных моделей*. Этот подход выходит за рамки простого ответа на запрос, речь идет о получении решения задач.

Далее оба подхода рассматриваются более подробно, начиная с семантики стабильной модели. Интуитивно, семантика стабильной модели основана на особой трактовке атомов с отрицанием, являющихся источником «противоречий» или «нестабильности». «Стабильность» при этом заключается в следующем. Если интерпретация M программы P непротиворечива, то она стабильна.

Простой пример программы, которой соответствует множество моделей, следует далее:

$$\begin{aligned} &man(petrov). \\ &single(X) :- man(X), not husband(X). \\ &husband(X) :- man(X), not single(X). \end{aligned}$$

Здесь утверждения $single(petrov)$ и $husband(petrov)$ взаимозависимы при использовании отрицания. Алгоритм SLD резолюции заикнулся бы здесь при попытке ответить на запрос $single(X)$. Вместе с тем у этой программы есть две минимальных модели Эрбрана, являющиеся стабильными:

$$\begin{aligned} M_1 &= \{man(petrov), single(petrov)\} \\ M_2 &= \{man(petrov), husband(petrov)\} \end{aligned}$$

Подход к логическому программированию на основе семантики стабильной модели [27], выражающий применение идеи аутоэпистемической логики [29] и логики умолчания [20] к анализу отрицания как неудачи, называется *Программированием множества ответов* (Answer Set Programming, ASP) [28]. Возможность использования средств вывода множества ответов как новой парадигмы программирования была обоснована в [30] (название «программирование множества ответов» было впервые употреблено в заголовке соответствующей части сборника, включающего эту статью) и в [31].

ASP отличается от многих языков представления знаний способностью выражать утверждения, основанные на умолчании, вида «обычно экземпляры класса C удовлетворяют свойству P ». Выражение умолчаний, исключений из таких умолчаний, а также способов использования этой информации для вывода адекватных заключений может быть поддержано ASP. ASP позволяет также выражать причинный эффект действий («утверждение F становится истинным в результате выполнения действия A »), утверждений, выражающих недостаток информации («не известно,

является ли утверждение P истинным, или ложным»), различные предположения общего вида, например, «утверждения, не следующие из базы знаний, являются ложными».

В ASP как языке программирования синтаксически добавляются дизъюнкции, сильные отрицания (наряду со слабым отрицанием, использующем семантику отрицания как неудачи), ограничения.

В дизъюнктивном правиле голова может представлять собой дизъюнкцию нескольких атомов:

$$A_1 \vee \dots \vee A_k :- B_1, \dots, B_m, \text{not } C_1, \dots, \text{not } C_n$$

Например, можно использовать правило вида:

$$\text{female}(X) \vee \text{male}(X) :- \text{person}(X)$$

Пример записи дизъюнктивного факта:

$$\text{broken}(\text{left_hand}, \text{tom}) \vee \text{broken}(\text{right_hand}, \text{tom})$$

Правило

$$\text{ok}(C) \vee \neg \text{ok}(C) :- \text{component}(C)$$

утверждает, что компонент может находиться в рабочем состоянии, или не работать.

Для выражения правил с умолчанием используется сильное отрицание в комбинации со слабым. Например, выражение того, что «по умолчанию птица летает», обеспечивается правилом

$$\text{flies}(X) :- \text{bird}(X), \text{not } \neg \text{flies}(X)$$

где \neg - сильное отрицание. Предикат $\text{not } p$ в теле правила можно интерпретировать (согласно В. Лифшицу) как «не верится, что p ».

Программы в ASP могут включать также правило выбора, такое как

$$\{s, t\} :- p.$$

Это правило означает следующее. Если p включается в стабильную модель, то следует выбрать произвольным образом, какой из атомов – s или t нужно включить.

Простое расширение позволяет использовать в программах ограничения – правила, в которых отсутствует голова:

$$:- B_1, \dots, B_m, \text{not } C_1, \dots, \text{not } C_n.$$

Здесь $B_1, \dots, B_m, C_1, \dots, C_n$ – атомы. Ограничению соответствует отрицание формулы, эквивалентной его телу:

$$\neg(B_1 \wedge \dots \wedge B_m \wedge \neg C_1 \wedge \dots \wedge \neg C_n)$$

В ASP ограничения играют важную роль. Включение ограничения в логическую программу P оказывает влияние на набор стабильных моделей P : стабильные модели, нарушающие ограничение, исключаются из такого набора. Иными словами, для любой логической программы P с ограничениями и любого ограничения C стабильные модели $P \cup \{C\}$ включают те стабильные модели P , которые удовлетворяют C .

Далее следует пример спецификации задачи трехцветной (b, r, g) раскраски графа $G = (V, E)$, представленного узлами $node(n)$ для каждого $n \in V$ и ребрами $edge(n, n')$ для каждой пары $(n, n') \in E$, задаваемой следующими правилами:

$$b(X) :- node(X), \text{not } r(X), \text{not } g(X)$$

$$r(X) :- node(X), \text{not } b(X), \text{not } g(X)$$

$$g(X) :- node(X), \text{not } r(X), \text{not } b(X)$$

и ограничениями

$$:- b(X), b(Y), edge(X, Y)$$

$$:- r(X), r(Y), edge(X, Y)$$

$$:- g(X), g(Y), edge(X, Y)$$

Логическое программирование в парадигме ASP успешно применяется для решения задач в широком классе проблем, включая диагностику в разных областях, интеграцию информации, поиск решения при задании набора ограничений, планирование действий, прокладку маршрутов, проблемы биомедицины и биологии, извлечение информации из текстов, классификацию.

Пусть P представляет собой множество правил вида:

$$A :- B_1, B_2, \dots, B_m, C_1, C_2, \dots, C_n$$

где $A, B_1, B_2, \dots, B_m, C_1, C_2, \dots, C_n$ – базовые (ground) атомы. Если P не содержит отрицаний ($n = 0$) в каждом правиле программы, то, по определению, единственной стабильной моделью P является ее минимальная модель. Чтобы расширить это определение на случай

программ с отрицанием, понадобится вспомогательное понятие редукта, определяемого следующим образом.

Для любого множество I базовых атомов *редуктом* P по отношению к I называется множество правил без отрицания, полученных из P исключением каждого правила, такого, что по крайней мере один из атомов C_i в теле этого правила принадлежит I , а затем исключением частей $not C_1, not C_2, \dots, not C_n$ из тел оставшихся правил.

Это *преобразование Гельфонда-Лифшица* [32], иногда называемое *стабильным* преобразованием. Целью такого преобразования является представление моделей как множества базовых атомов, в котором отсутствующие атомы представляют атомы с отрицанием. В этом контексте «минимальная модель» - это модель, содержащая минимальное множество позитивных атомов, а «монотонность преобразования» полных интерпретаций заключается в том, что оно монотонно при рассмотрении одних лишь позитивных атомов. Стабильные модели представляются в двузначной логике.

Говорят, что I – это *стабильная модель* P , если I – стабильная модель редукта P по отношению к I . Каждая стабильная модель P является моделью P . Полная модель логической программы P является стабильной, если она является неподвижной точкой трансформации Гельфонда-Лифшица. Если программа P имеет в точности одну стабильную модель, то она называется уникальной стабильной моделью P .

3.3 Хорошо обоснованная семантика и ее соотношение с ASP

Хорошо обоснованной семантикой (WFS) программы P , согласно определению в [26], является ее значение, представленное наименьшей неподвижной точкой трансформации множества литералов, атомы которых входят в базу Эрбрана данной программы P ; каждый позитивный литерал означает, что его атом является истинным, каждый литерал с отрицанием означает, что его атом является ложным, а атомы не имеют присвоенного им значения истинности. Таким образом, эта модель является моделью трехзначной логики.

Например, если известно, что

Объект A – это ночная бабочка, если A не летает днем.

но не известно, летает ли A днем, в хорошо обоснованной семантике высказывание «объект A – это ночная бабочка» получает значение *unknown*, то есть, его значением не является ни истина, ни ложь.

Если атом является истинным в хорошо обоснованной модели программ P , то он принадлежит каждой стабильной модели P . Обратное утверждение, вообще говоря, не выполняется. Например, программе

$p :- \text{not } q$ $q :- \text{not } p$ $r :- p$ $r :- q$

имеет две стабильные модели $\{p, r\}$ и $\{q, r\}$. Не смотря на то, что r принадлежит обеим моделям, значением r в хорошо обоснованной модели является *unknown*.

Более того, если атом является ложным в хорошо обоснованной модели программы, то он не принадлежит к любой ее стабильной модели. Таким образом, хорошо обоснованная модель логической программы является нижней гранью пересечения ее стабильных моделей и верхней гранью их объединения.

Одним из основных отличий ASP от хорошо обоснованной семантики является ориентация ASP на решение переборных задач, что невозможно в WFS. Из-за различия выразительной способности двух парадигм они используются в разных целях. ASP идеально подходит для решения сложных комбинаторных задач, соответствующие этой парадигме системы обычно применяются как компоненты поддержки баз знаний, встраиваемые в императивные системы программирования. Например, система LPARSE первоначально была создана как фронтальный процессор для решателя множества ответов SMOODELS, а впоследствии использовалась аналогичным образом с большинством других решателей множества ответов. Система DLV [33] является исключением: синтаксис ASP программ в DLV отличается от синтаксиса, используемого в других системах.

Диалекты логического программирования, такие как Datalog с немонотонным отрицанием, над которыми надстраивается ASP (как расширение Datalog'a), часто рассматриваются в качестве естественного основания для слоя правил Семантического Веба. Современные системы ASP содержат расширения для извлечения данных в RDF и задания запросов к OWL. В архитектуре Семантического Веба изучаются проблемы, возникающие в связи с добавлением правил с немонотонным отрицанием, основанным на предположении о замкнутости мира, над RDF и OWL, основанным на предположении об открытости мира⁴.

В противоположность ASP, WFS-базированные системы являются вычислительно полными и используются как полновесные системы программирования. Хорошо обоснованная семантика является основой реализации многих систем (например, XSB, Ontobroker, Intellidimension, SweetRules, SILK, FLORA).

⁴ Эта проблема подробно изучается в проекте RIF.

3.4 Фреймовая логика и язык метапрограммирования

В этом разделе кратко рассматриваются элементы языков F-Logic и HiLog (в их воплощении в языке системы на правилах FLORA-2 [34]), которые оказали существенное влияние на формирование каркаса RIF.

F-logic (фреймовая логика) [35, 36] рассматривается как язык представления знаний, и как онтологический язык. F-logic соединяет возможности концептуального моделирования с объектно-ориентированными, фреймово-базированными языковыми средствами и предлагает компактное, декларативное представление программ в хорошо обоснованной семантике логического языка программирования. Язык поддерживает такие средства, как например, уникальная идентифицируемость, сложно структурированных объектов, наследование, полиморфизм, методы, поддерживающие запросы, инкапсуляция. Первоначально F-logic как язык был разработан в ориентации на дедуктивные базы данных, но в последнее время он используется все чаще для поддержки семантических технологий. При этом F-logic обеспечивает логические основания фреймово-базированного и объектно-ориентированного языка представления данных и знаний.

HiLog – это логическое расширение языка Prolog для поддержки средств мета-программирования и программирования в логике высоких порядков при сохранения вычислимости в логике первого порядка.

Язык FLORA-2 – это диалект F-logic с многочисленными расширениями, включая мета-программирование в стиле языка HiLog и возможности изменения баз данных логическими средствами, представленными в виде *транзакционной логики*. FLORA-2 обладает также развитыми средствами проектирования модульного программного обеспечения на основе динамических модулей. Области применения FLORA-2 включают область интеллигентных агентов, Семантический Веб, сети баз знаний, онтологическую инженерию, интеграцию информации. В основе FLORA-2 используется машина вывода XSB.

Колорит языка логического программирования F-logic [34] придает ему немонотонную семантику при интерпретации операции отрицания как неудачи, поддержке множественного наследования с перекрытием, что выходит за рамки традиционного логического программирования.

В синтаксисе F-logic выражение принадлежности экземпляра классу выглядит как *John:student*, а выражение отношения подкласса реализуется как *student::person*. Сами классы интерпретируются как объекты, так что один и тот же объект может играть роль класса в одной формуле, и роль объекта в другой. Например, в формуле *student:class*, символ *student* играет роль объекта, тогда как в формуле *student::person* он вступает в роли класса.

F-logic также обеспечивает возможность задания информации о схеме при помощи сигнатурных формул. Например, *person/spouse {0:1} => person, name {0:1} => string, child => person* – это сигнатурная формула, которая выражает тот факт, что класс *person* имеет три атрибута – однозначные атрибуты *spouse* и *name* (однозначность выражается при помощи ограничения кардинальности 0:1) и множественного атрибута *child*. При этом также сообщается, что значением первого атрибута являются объекты типа *person*, второй атрибут принимает значения типа *string*, а последний имеет значением множество, экземплярами которого являются объекты типа *person*.

Ниже представлен пример программы о базе данных публикаций на языке Flora-2 [37]. Схема выглядит следующим образом:

```
paper[authors => person, title => string].
journal_p :: paper[in_vol => volume].
conf_p :: paper[at_conf => conf_proc].
journal_vol[of => journal, volume => integer, number => integer, year => integer].
journal[name => string, publisher => string, editors => person].
conf_proc[of_conf => conf_series, year => integer, editors => person].
conf_series[name => string].
publisher[name => string].
person[name => string, affil(integer) => institution].
institution[name => string, address => string].
```

Частью программы являются также определения объектов:

```
o_j1: journal_p[title -> 'Records, Relations, Sets, Entities, and Things',
  authors -> {o_mes}, in_vol -> o_i11].
o_di : conf_p[title -> 'DIAM II and Levels of Abstraction', authors -> {o_mes, o_eba},
  at_conf -> o_v76].
o_i11 : journal_vol[of -> o_is, number -> 1, volume -> 1, year -> 1975].
o_is : journal[name -> 'Information Systems', editors -> {o_mj}].
o_v76 : conf_proc[of -> vldb, year -> 1976, editors -> {o_pcl, o_ejn}].
o_vldb : conf_series[name -> 'Very Large Databases'].
o_mes : person[name -> 'Michael E. Senko'].
o_mj : person[name -> 'Matthias Jarke', affil(1976) -> o_rwt].
o_rwt : institution[name -> 'RWTH Aachen'].
```

Запросы можно задавать как относительно информации в схеме классов, так и относительно содержания структуры отдельных объектов. Это достигается помещением переменных в надлежащие синтаксические позиции. Следующий запрос:

```
?- student[?M=> person]
```

находит методы, определенные на множествах в классе *student* и возвращает объекты типа *person*. Следующий запрос возвращает все суперклассы класса *student*:

?- student::?C and student[name=> ?T].

Для повышения гибкости таких мета-запросов в HiLog введены синтаксические конструкции второго порядка, что позволяет использовать переменные на месте функциональных и предикатных символов. Например, допускаются запросы, подобные следующему:

?- person[?M(?Arg)=> integer].

в котором переменная *?M* связывается с функциями. Тем не менее семантика таких символов второго порядка остается первопорядковой. Это значит, что переменные связываются не с экстенционалами символов (т.е., с отношениями, интерпретирующими символы предикатов или функций), а с самими такими символами.

HiLog допускает также использование переменных над атомарными формулами. Например, рассмотрим следующий запрос, который завершается связыванием переменной *?X* с атомом *p(a)*.

p(a).
q(p(a)).
?- q(?X), ?X.

Здесь высказывание *p(a)* материализуется (is reified) в виде объекта, в результате чего оно может быть связано с переменной. Материализация атомарных формул в HiLog'e может быть расширена до произвольных бескванторных формул языков HiLog и F-logic. Например, можно сказать, что *John* верит в то, что *Mary* нравится *Sally*:

John[believes -> \${Mary[likes -> Sally]}].

Обозначение *\${...}* в языке FLORA-2 используется для обозначения операторов материализации. Примером более сложного оператора материализации является следующий:

John[believes -> \${Bob[likes -> ?X] :- Mary[likes -> ?X]}].

Это предложение материализует правило (не просто факт) и утверждает, что *John* также верит, что *Bob*'у нравятся все, кто нравится *Mary*. Соединяя с предыдущим оператором, что *John* верит в то, что *Mary* нравится *Sally*, можно было бы ожидать заключения, что *John* также

поверит в то, что *Bob* у нравится *Sally*. Однако, такое заключение не может быть получено, поскольку неизвестно, является ли *John* способным к рассуждению существом, которое может употреблять *modus ponens* в повседневной жизни. Но эту информацию можно сообщить простым образом:

John[believes -> ?A] :- John[believes -> {\${?Head :- ?Body}; ?Body}].

FLORA-2 включает также транзакционную логику с некоторыми уточнениями, позволяющими различать запросы от транзакций и обеспечить возможность реализации контроля в период компиляции. В транзакционной логике как действия так и запросы представляются предикатами. В языке FLORA-2 транзакции представляются методами объектов с префиксом «%».

При условии, что пост-условие транзакции выполняется, изменения вносятся в базу данных. Если это условие не выполняется, текущая попытка изменения базы данных считается неприемлемой, и предпринимается следующая попытка. Если не находится ни одного приемлемого выполнения, транзакция не выполняет никаких изменений в базе данных. Таким образом обеспечивается атомарность транзакций.

В языке FLORA-2 поддерживается концепция областей действия для отрицания по умолчанию. Модуль в языке FLORA-2 – это контейнер для отдельной базы знаний (или ее части). Модули позволяют изолировать отдельные части базы знаний и обеспечивают интерфейс для взаимодействия таких частей. Модули могут образовываться динамически, связываться с нужными частями базы знаний, обеспечивая эффективный вид инкапсуляции. Область действия запроса ограничивается модулем, или всеми модулями, зарегистрированными в приложении. Это решение позволяет считать предположение о замкнутости мира (CWA) приемлемым даже в среде Веба, если область действия программы явно и точно определена. Тем самым удается устранить возражения, направленные против идеи предположения о замкнутости мира, основанные на том, что Веб практически бесконечен и неудача в выводе некоторого факта из имеющейся информации не дает гарантии правильности заключения о том, что этот факт является ложным.

4 Примеры использования и требования к RIF

4.1 Примеры использования RIF и системы на правилах в области интересов группы RIF WG

При создании средств унификации неоднородных языков и обеспечения интероперабельности соответствующих систем [38] для уточнения требований к методам и средствам унификации и проверки их адекватности, с самого начала проекта следует позаботиться об установлении взаимодействия с группами создания и поддержки конкретных разнородных языков и систем. Группой RIF WG изначально было определено, что система на правилах представляет интерес для RIF WG если для нее будут взяты обязательства разработать применения, в которых потребовался бы обмен правилами с другими представляющими интерес для RIF WG системами на правилах, и реализовать соответствующий пример использования. Более 50 предложений примеров использования были получены RIF WG в 2005 г. [39]. Эти примеры представляли более 20 различных систем на правилах [RIFLRS 05] (Таблица 2⁵).

Таблица 2 Системы на правилах, отобранные для анализа и выработки требований к RIF

Название системы	Статус	Категория использования	Язык правил
FLORA-2	A/D	1, 2, 4, 8	Based on F-logic, HiLog, and Transaction Logic
ILOG BRMS	C/D	3	Proprietary production rule language (IRL)
Hoolet	A	8	SWRL, Vampire theorem prover
KAON2	C/A	3, 8	DL-safe Subset of SWRL
TRIPLE	A	6	TRIPLE (F-Logic inspired language for RDF)
Bossam	A	3	Proprietary Production Rule Language
JenaRules	D	1, 3	RDF production rule language
Euler	D	2	Backward-chaining reasoner
OntoBroker / OntoStudio	C/D	1	F-logic variant Supports well-founded semantic
Fair Isaac Blaze Advisor	C/D	1, 2, 5	Structured Production Rules Language
Oracle Database	C	1	SQL, XQuery, some SPARQL operators

⁵ В приведенной таблице столбец *Статус* означает *Commercial/Academic/Deployed* (Deployed means used in a deployed application).

Oracle Rules Manager	C	2, 5, 7	ECA rules in the Oracle Database
Oracle Business Rules	C	2, 5, 7	Rule engine (based on Jess) and tools in Fusion M
TextGarden	A/D	8	Predictive Model Markup Language
Xcerpt	A	5	Views with a logical syntax
XChange	A	1, 2, 3, 4	ECA rules, logical event query language
PeerTrust	A	4	Policy and trust negotiation language
MINS	A/D	1	Reasoner for Datalog programs with negation and function symbols
DLV	C/A/D	1, 5	Disjunctive Datalog with classical and default negation and aggregates
Prova	A/D	1, 4, 7,	Based on Java, reaction rules, distributed agents, workflows
TIBCO BusinessEvents	C/D	2, 3	Production and ECA rules
IRIS	A/D	1	Stratified datalog or WSML, Open source (LGPL)
R2ML	A/D	4, 5	XML Schema based rule markup

Категоризация примеров использования, приведенная первоначально в [40], продолжает эволюционировать до сих пор. По мере развития проекта RIF требования к примерам использования уточнялись, в частности, с тем, чтобы они в наибольшей мере соответствовали уже определенным диалектам. Результаты этой деятельности определены в уточняющих документах RIF WG [41].

Список категорий примеров использования согласно [40], выглядит так:

1. Интеграция информации
2. Принятие решений
3. Кросс-платформенная разработка и развертывание правил
4. Стратегии авторизации транзакций и управления доступом
5. Обмен бизнес-правилами, ориентированными на взаимодействие с пользователем
6. Публикация программ на правилах
7. Сервисы обмена правилами для третьей стороны
8. Развитое представление знаний

В Таблице 2 показаны категории примеров использования, предоставленных в рамках различных систем на правилах.

Анализ полученных примеров использования показывает, что правила применяются для реализации разнообразных задач, и поэтому системы на правилах нельзя считать монолитными. Правила использовались для

контроля качества вывода, реализации вычислений, управления информационными потоками, проверки ограничений целостности в базах данных, представление стратегий и управление ими, управления устройствами и процессами в реальном масштабе времени, определения потребности вмешательства человека в процессы управления, и др.

4.2 Требования к RIF

RIF должен быть определен таким образом, чтобы обеспечивалась возможность создания новых диалектов (требование расширяемости), следуя основным целям и общим требованиям RIF, равно как и возможность изменения существующих диалектов (при соблюдении требования совместимости снизу вверх).

Достижение междиалектной интероперабельности само по себе является плохо определенной задачей, поскольку известно, что трансляция диалектов с различной семантикой при полном сохранении смысла едва ли реализуема в большинстве случаев. Это не означает, что междиалектная трансляция вообще невозможна, однако, требуются дополнительные критерии для формулирования точного понимания того, что считать удовлетворительной трансляцией (посредством обмена RIF-правилами). До сих пор определение критериев качества междиалектной трансляции не входило в задачи RIF WG.

Цели проекта и примеры использования предопределили требования к RIF Требования, определенные как общие, предусматривают набор фундаментальных свойств, которыми должны обладать создаваемые диалекты.

4.2.1 Общие требования

- *Реализуемость.* RIF должен быть реализуем на основе устоявшихся методов и не должен требовать дополнительных исследований, например, алгоритмических или семантических проблем разработки трансляторов.
- *Семантическая точность.* Ядро RIF должно иметь ясно и точно определенные синтаксис и семантику. Каждый стандартный диалект RIF должен иметь ясно и точно определенные синтаксис и семантику, расширяющие ядро RIF.
- *Расширяемость формата.* Должна быть обеспечена возможность создания новых диалектов RIF, расширяющих существующие диалекты (совместимость в обратном направлении с уже созданными диалектами) и допускающих их постепенное использование в системах, которые используют уже существующие диалекты (совместимость в прямом направлении, в направлении развития).

- *Трансляторы.* Для каждого стандартного диалекта RIF должна существовать возможность создания трансляторов между языками правил, определяемыми данным диалектом и RIF, без изменения языка правил.
- *Стандартные компоненты.* Реализации RIF должны обеспечивать возможность использования стандартных вспомогательных технологий, таких как парсеры XML, генераторы парсеров, и не должны требовать специализированных реализаций при возможности повторного использования существующих решений.
- *Охват спектра языков на правилах.* Из-за большого разнообразия языков на правилах, едва ли найдется единственный язык обмена, который мог бы служить мостом для всех языков на правилах. Поэтому RIF предлагает диалекты, каждый из которых ориентирован на кластер подобных друг другу языков на правилах. RIF должен обеспечивать внутрдиалектную интероперабельность, т.е. интероперабельность между семантически подобными языками на правилах (при помощи обмена правилами RIF) в рамках одного диалекта, а также он должен поддерживать междиалектную интероперабельность, т.е. интероперабельность между диалектами на максимально возможном их перекрытии.

4.2.2 Требования, мотивированные примерами использования

- *Модель совместимости.* Спецификации RIF должны определять четкие критерии конформности для идентификации реализаций RIF, которые являются конформными RIF.
- *Поведение по умолчанию.* RIF должен специфицировать на соответствующем уровне детализации поведение по умолчанию, которого можно ожидать от приложения, совместимого с RIF, но не обладающего способностью обработки всех или части правил, определенных в RIF документе, либо он должен обеспечивать способ описания подобного поведения по умолчанию.
- *Семантические различия.* RIF должен охватывать языки на правилах, имеющие различную семантику.
- *Ограниченное число диалектов.* RIF должен иметь стандартное ядро и ограниченное число стандартных диалектов, базирующихся на этом ядре.
- *Данные OWL.* RIF должен охватывать базы знаний на OWL как данные настолько, насколько они оказываются совместимыми с семантикой RIF.
- *Данные RDF.* RIF должен охватывать триплеты RDF как данные настолько, насколько они оказываются совместимыми с семантикой RIF.

- *Идентификация диалекта.* Семантика RIF документа должна однозначно определяться его содержимым без привлечения данных извне.
- *Синтаксис XML.* RIF должен поддерживать синтаксис XML в качестве основного нормативного синтаксического представления.
- *Типы данных XML.* RIF должен поддерживать надлежащий набор скалярных типов данных с ассоциированными операциями в соответствии с их определениями в XML Schema part 2 и связанных с этим документом спецификациях.
- *Слияние наборов правил.* RIF должен поддерживать возможность слияния наборов правил.
- *Идентификация наборов правил.* RIF должен поддерживать возможность идентификации наборов правил. RIF должен охватывать базы знаний на OWL как данные настолько, насколько они оказываются совместимыми с семантикой RIF.

Подробные описания примеров использования даны в [39, 40, 41].

5 Краткий обзор основных решений RIF

Согласно [7], RIF изначально был ориентирован на обмен правилами, а не на создание одного всеобъемлющего языка на правилах. В противоположность другим стандартам Семантического Веба, таким как RDF, OWL и SPARQL, с самого начала было ясно, что единственным языком не удастся охватить все парадигмы на правилах, активно используемые для представления знаний и моделирования в сфере бизнеса. Было осознано, что даже обмен правилами сам по себе является сверхзадачей. Известные системы на правилах попадают в одну из трех широких категорий: системы на языках первого порядка, системы логического программирования, системы на активных правилах. Эти системы синтаксически и семантически имеют мало общего. Более того, даже в пределах одной парадигмы имеются большие различия между системами.

Такое разнообразие предопределило подход RIF WG, заключающийся в создании семейства языков, называемых диалектами, со строго определенными синтаксисом и семантикой. Семейство диалектов RIF должно быть однородным и расширяемым. Однородность означает, что диалекты должны впитать в себя как можно больше из существующего разнообразия синтаксического и семантического аппарата. Расширяемость означает, что при наличии обоснованной мотивации, должна быть обеспечена возможность определения новых диалектов RIF как синтаксических расширений существующих диалектов, содержащих

новые элементы с требуемой дополнительной функциональностью. Такие новые диалекты RIF, не являясь стандартами в начале, в дальнейшем могут быть стандартизованы.

Ввиду требования строгости определений, слово «формат» в названии RIF звучит несколько уничижительно. Фактически, RIF предоставляет больше, чем просто формат. Однако, понятие формата является существенным для понимания способа предполагаемого использования RIF. В конечном счете, в качестве среды обмена при помощи RIF между различными системами на правилах предлагается XML – формат обмена данными. Основная идея обмена правилами при помощи RIF заключается в том, что для различных систем будут определены отображения их собственных языков в диалекты RIF и обратно. Такие отображения должны сохранять семантику, так что полученные наборы правил могут передаваться от одной системы к другой при условии, что эти системы могут общаться друг с другом при помощи подходящего диалекта, поддерживаемого обеими системами.

5.1 Диалекты RIF

Работа RIF WG была сосредоточена на двух видах диалектов – диалектах, основанных на логике, и диалектах на правилах с действиями [7]. Вообще говоря, диалекты, основанные на логике, включают языки, использующие логику некоторого вида, такую как логика первого порядка (часто ограниченная логикой Хорна) или непервопорядковые логики, лежащие в основе различных языков логического программирования (например, логического программирования, следующего хорошо обоснованной или стабильной семантике). Диалекты на правилах с действиями включают системы на продукционных правилах, такие как Jess, Drools и JRules, и системы с реактивными правилами (или правилами «событие – условие – действие»), такие как Reaction RuleML и XChange. Из-за ограниченности ресурсов RIF WG, были определены только два логических диалекта – Базовый логический диалект (RIF-BLD) и его подмножество Диалект-ядро RIF, расширяемое также Диалектом продукционных правил (RIF-PRD). RIF-PRD - это единственный диалект на правилах с действиями, определенный RIF WG.

Диалект RIF-BLD соответствует логике Хорна с различными синтаксическими и семантическими расширениями⁶. Основные

⁶ Расширения языка трактуются следующим образом [KBWF05]. Пусть $L_1 \subseteq L_2$ - два логических языка, семантика которых определена на основе отношений следования (entailment) \models_1 и \models_2 . Говорят, что L_2 является расширением L_1 , если для любой пары формул $\varphi, \psi \in L_1$ следование $\varphi \models_1 \psi$ выполняется тогда и только тогда, когда выполняется $\varphi \models_2 \psi$. Для языков на правилах, множество правил, которые можно использовать в роли посылок (premises), не совпадает с множеством правил, которые можно использовать в качестве следствий (consequents). Поэтому нужно предположить, что $L_1 = \text{Premises}_1 \cup \text{Consequents}_1$ и

синтаксические расширения включают синтаксис фреймов и предикаты с поименованными аргументами. Основные семантические расширения включают типы данных и внешне определенные предикаты. Хотя этот диалект не является достаточно выразительным для многих применений правил, он охватывает большое число существующих систем на правилах, что позволяет считать его необходимой предпосылкой создания более выразительных диалектов в будущем. Эту деятельность предполагается ввести в рамках каркаса расширения RIF, называемого RIF-FLD.

RIF-PRD – это другой крупный диалект, разработанный RIF WG, охватывающий основные аспекты различных систем на производственных правилах. Серьезный интерес к технологиям на производственных правилах был проявлен крупными компаниями. Производственные правила в соответствии с существующей практикой в системах, подобных Jess или JRules, определяются, используя неформальные вычислительные схемы, не основанные на логике. Поэтому RIF-PRD не является частью набора логических диалектов RIF. Вместе с тем, значительное внимание было уделено тому, чтобы выделить максимально возможную общую часть производственного и логических диалектов. Выделение такой общей части послужило основанием для создания Диалекта-ядра RIF.

Предполагается, что существующие и будущие диалекты RIF будут использовать один и тот же набор типов данных, встроенных функций и предикатов в соответствии с их определением в документе RIF Datatypes and Built-Ins (RIF-DTB).

5.2 Каркас RIF для логических диалектов

Для упрощения создания новых диалектов и сокращения времени их разработки RIF WG определила каркас расширения RIF, называемый Каркасом логических диалектов (RIF-FLD) [42]. Не исключено появление в будущем аналогичного каркаса правил с действиями. RIF-FLD не является собственно диалектом, он определен как универсальный каркас для создания новых логических диалектов, расширяющих существующие. Он был разработан для существенного упрощения процесса определения и верификации новых логических диалектов, расширяющих возможности RIF-BLD.

Разработка каркаса RIF-FLD оказалась возможной, поскольку, не смотря на различия логических теорий, составляющих основу различных

$L_2 = Premises_2 \cup Consequents_2$. Более того, L_1 не обязательно может быть подмножеством L_2 . Скажем, он может включаться в L_2 на основе 1-1 трансформации i . В используемых обозначениях это можно выразить как $i(Premises_1) \subseteq Premises_2$ и $i(Consequents_1) \subseteq Consequents_2$. Теперь можно определить, что L_2 расширяет L_1 относительно трансформации i , если для каждой пары формул $\phi \in Premises_1$ и $\psi \in Consequents_1$ следование $\phi \vDash_1 \psi$ выполняется тогда и только тогда, когда выполняется $i(\phi) \vDash_2 i(\psi)$.

логических систем на правилах, в них используется много общих синтаксических и семантических конструкций. Более того, способы образования комбинаций различных подобных конструкций для образования соответствующих систем хорошо изучены. Однако, спецификация RIF-FLD уникальна, поскольку является результатом тщательного разбора этих знаний и представляет их в согласованном виде, причем XML используется даже на уровне каркаса.

RIF-FLD представляет собой весьма общий логический язык, использующий значительную часть широко используемых синтаксических и семантических конструкций: при этом, однако, намеренно ряд параметров оставлен неопределенным для того, чтобы конструкторы конкретных диалектов могли добавить необходимые детали. Например, RIF-FLD предоставляет способ представления правил синтаксиса при помощи понятия сигнатур. В нем также можно специфицировать некоторые семантические понятия, такие как модели и логическое следование, оставляя при этом открытым выбор конкретных вариантов (например, какие конкретно модели следует использовать при рассмотрении следования). Конструктор конкретного диалекта может определить его синтаксис и семантику заданием их специализации на основе синтаксиса и семантики RIF-FLD. При этом конструктор осуществляет возможность выбора параметров из вариантов, предоставляемых RIF-FLD, не требуя при этом определения формул, типов данных, моделей, следования, и пр. Этот подход продемонстрирован на определении RIF-BLD. Этот диалект специфицирован двумя способами: прямым перечислением всех определений, что потребовало около 40 страниц текста, и заданием спецификации на основе RIF-FLD, что потребовало всего пяти страниц. Любое различие между двумя спецификациями следует рассматривать как ошибку, подлежащую исправлению. Таким образом, на примере двойной спецификации RIF-BLD показаны преимущества определения диалектов с помощью каркаса RIF-FLD.

Каркас RIF включает гибкий набор конструкций, достаточных для:

- формирования диалектов с различной семантикой, включая хорошо обоснованную и стабильную семантику;
- введения правил, имеющих широкий диапазон представлений – от Хорновских правил до универсальных правил с произвольными формулами в голове (таким образом, например, становятся представимыми как правила дизъюнктивного Datalog'a, так и GLAV взгляды);
- выбора трактовки предположения об уникальности имен (в логическом программировании синтаксически разные базовые термы обозначают разные объекты, тогда как в дескриптивных логиках это не так);
- обеспечения возможности использования функций в правилах;
- использования развитых структур данных на основе фреймов;

- определения типов данных, включающих методы;
- определения иерархии классов и отношений наследования;
- введения разнообразных скалярных типов данных (XML Schema).

Разработчики каркаса не считают его спецификацию незыблемой. Напротив, при необходимости в будущем он может быть изменен и расширен.

К моменту написания настоящей работы было известно, что RIF FLD был использован для определения трех диалектов: Базового логического диалекта (RIF BLD), Диалекта-ядра средств программирования множества ответов (RIF CASPD), Диалекта-ядра логического программирования, базирующегося на хорошо обособленной семантике (RIF CLPWD). Известно также, что RIF FLD используется для определения Диалекта правил с неопределенностью (ULD) и Диалекта логики умолчания (DLD) как расширения BLD. Ведутся также работы по средствам синтаксического контроля текстов программ на соответствие различным диалектам, например, на соответствие Диалекту-ядру RIF.

5.3 Вопросы совместимости с RDF и OWL

Признавая необходимость сопряжения правил RIF с RDF и онтологиями OWL, RIF WG определила также концепции обеспечения совместимости RIF с RDF и OWL. Учитывая, что RIF, RDF и OWL – языки с различным синтаксисом и семантикой, следовало определить, как из правил RIF ссылаться на факты RDF и OWL и выяснить, может ли иметь логический смысл всеобъемлющий язык. Специальный документ RIF, посвященный совместимости RIF с RDF и OWL, дает ответ на эти вопросы. Основная идея заключается в использовании синтаксиса фреймов для взаимодействия с RDF/OWL. Фреймы отображаются в триплеты RDF и определяется объединенная семантика для такой комбинированной конструкции. Обмен правилами посредством RIF может находиться в зависимости от данных RDF, RDF Schema или онтологий OWL, или использоваться вместе с ними. Таким образом, требуется определить возможность обеспечения интероперабельности RIF с другими стандартами Семантического Веба.

Гибридный подход к обеспечению интероперабельности правил и онтологий [25]. Требуется создание логической надстройки над стеками OWL и правил, которая позволяла бы использовать вывод, реализуемый в OWL в правилах, и, наоборот, вывод на основе правил использовать в OWL. Основная идея такого взаимодействия заключается в том, что системы на правилах и OWL будут рассматривать друг друга как черные ящики с интерфейсом, реализуемым на основе экспортируемых предикатов. Онтологии OWL будут экспортировать свои классы и

свойства, а базы знаний на правилах будут экспортировать некоторые из определенных в них предикатов. Каждая из баз знаний будет иметь возможность ссылаться на предикаты в другой базе знаний, причем одной из возможных трактовок таких предикатов является их экстенциональная интерпретация как множеств фактов. В частности, онтологии можно использовать как разделяемые многими приложениями концептуализации предметных областей. В этом случае различным приложениям в этой области могут соответствовать специфические программы на правилах.

Такой подход, называемый гибридным, обеспечивает возможность интегрированного использования существующих машин вывода в системах на правилах и в онтологических системах для реализации рассуждений в гибридном языке вместо того, чтобы создавать совершенно новую машину вывода [43].

Можно представить себе два языка – язык логического программирования R и онтологический язык S (последний можно считать базированным на дескриптивной логике). Гибридное правило над R и S может иметь следующий вид:

$$H :- B_1, \dots, B_m, Q_1, \dots, Q_n$$

где $m, n \geq 0$; H, B_i – литералы, а Q_j – запросы, представленные в языке запросов Q_S .

Гибридная база знаний $K = (D, P)$ представляет собой конечное множество D аксиом в дескриптивной логике на онтологическом языке S и конечное множество гибридных правил P над R и S , включающих атомы не в дескриптивной логике. Оба языка поддерживаются машинами вывода, отвечающими на запросы в соответствующем языке запросов Q_R и Q_S . Язык запросов в гибридной парадигме синтаксически совпадает с с языком запросов Q_R . Ответом на запрос является совокупность фактов, которая следует из гибридной базы знаний.

Однородный подход к обеспечению интероперабельности правил и онтологий [ADGH 05]. В гибридном подходе обычные предикаты в языке на правилах и онтологические предикаты (которые чаще всего выступают в роли ограничений в посылках правил), строго разделены. Рассуждения осуществляются на основе сопряжения существующих машин вывода системы на правилах и онтологии.

В однородном подходе как онтологии, так и правила выражаются в одном языке L без необходимости установления априорных различий между предикатами на правилах и онтологическими предикатами.

5.4 Сценарии использования RIF совместно с данными RDF или онтологиями RDFS/OWL

Такие сценарии определены следующим образом [45]. Партнер по взаимодействию *A* использует язык правил, в котором имеются возможности для работы с RDF/OWL. Подобные возможности могут включать поддержку доступа к данным RDF, использование онтологий RDFS или OWL, либо расширение RDF(S)/OWL средствами языка правил *A*. Используя RIF, *A* посылает партнеру *B* свои правила, возможно, содержащие ссылки на нужные графы RDF. *B* получает правила и извлекает требуемые графы RDF. Правила транслируются во внутренний язык *B* и обрабатываются вместе с RDF графами, используя возможности для работы с RDF/OWL машины вывода *B* (подобные названным выше для партнера *A*).

Специализацией этого сценария является публикации правил RIF, ссылающихся на RDF графы (публикация рассматривается как специальный вид взаимодействия «один ко многим»). Когда партнер *A* публикует свои правила в Вебе, может быть несколько потребителей, получающих RIF правила и RDF графы из Веба, транслирующих RIF правила в соответствующие языки правил, и обрабатывает их вместе с RDF-графами в их собственных машинах вывода.

Другая возможная специализация сценария обмена правилами опирается на намерение расширения онтологии OWL правилами партнером, осуществляющим публикацию. Партнер по взаимодействию *A* использует язык правил, расширяющий OWL. *A* расщепляет свое определение онтологии и правил в отдельные онтологию OWL и документ RIF, публикует онтологию OWL и посылает (или публикует) документ RIF, который включает ссылки на онтологию OWL. Потребитель правил извлекает и транслирует онтологию OWL и документ в комбинированное описание, содержащее онтологию и правила в собственном языке правил, расширяющем OWL.

5.5 Состояние реализации RIF

RIF WG на основании отчетов по реализации RIF представила информацию в виде таблицы 3 [46].

Таблица 3 Состояние реализации RIF

Система на правилах	Организация	Диалект RIF	Поддержка функций
SILK	Vulcan, BBN, Stony Brook University	BLD, Default Logic Dialect (DLD)	producer, consumer
OntoBroker 5.3	ontoprise	BLD (partial)	producer, consumer

Fuxi	Chimezie Ogbuji	RIF Core and OWL 2 RL in RIF	producer
IBM Websphere ILOG JRules	IBM/ILOG	PRD+Core	producer (PRD), consumer (PRD+Core)
Eye	Jos De Roo	BLD+DTB	consumer
VampirePrime	Alexandre Riazanov	BLD	consumer
RIFle	José María Álvarez	Core, PRD, DTB	validator
OBR	Oracle	PRD without Import	producer, consumer
IRIS	STI Innsbruck	BLD+DTB	producer, consumer
N/A	Stijn Heymans, MichaelKifer	FLD	Core Answer Set Programming Dialect (RIF-CASPD)
N/A	Jidi Zhao, HaroldBoley	FLD	RIF Uncertainty Rule Dialect (RIF-URD)
riftr	Sandro Hawke	Core, BLD, DTB, RDF Import	producer, consumer, validator

6 Заключение

Работа по стандарту унификации языков на правилах, выполненная RIF WG, является значительным событием. Предложен тщательно обоснованный и формализованный подход к созданию семейства диалектов языков на правилах, позволяющих однородно представить многообразие существующих языков на правилах. Для логических языков на правилах определен каркас, позволяющий конструировать различные диалекты, позволяющие создавать взаимные сохраняющие семантику отображения различных логических языков существующих систем на правилах в такие диалекты. Такие отображения являются необходимой предпосылкой обеспечения интероперабельности и повторного использования логических программ. Новый диалект создается как расширение существующих диалектов, управляемое каркасом.

Впервые работа по унификации языков выполняется как задача крупного консорциума. Само по себе осознание необходимости подобной работы таким разнородным сообществом как W3C уже является знаменательным событием.

Важным достижением является также выделение общего ядра для языков на правилах с совершенно различной парадигмой – логических и продукционных языков. Определения логического и продукционного диалектов как расширения ядра (первое из которых удовлетворяет каркасу) служат необходимой предпосылкой начала реализации RIF. Создание каркаса для совокупности продукционных языков – одна из задач развития RIF.

В результате выполненной RIF WG работы становится возможным рассмотрение программ на правилах как особого вида информационных ресурсов, которые могут быть использованы в различных средах интеграции и интероперабельного использования ресурсов (например, в средах поддержки предметных посредников). Благодаря высокому уровню абстракции и декларативности программ на правилах, становится практически достижимой точная спецификация поведения предметных посредников (тогда как до сих пор при спецификации поведения на этом сугубо декларативном уровне приходилось ограничиваться спецификацией пред и постусловий). В частности, применение различных моделей логических рассуждений становится реально возможным при решении задач с использованием множества неоднородных распределенных информационных ресурсов.

Литература

1. Kalinichenko L.A. Methods and tools for equivalent data model mapping construction // *Advances in Database Technology: Proc. of the International Conference on Extending Database Technology EDBT'90, LNCS 416*. - Berlin-Heidelberg: Springer-Verlag, 1990. - P. 92-119.
2. Kalinichenko L.A. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment. Report of the Institute for Problems of Informatics of the Russian Academy of Sciences: Second edition of the SYNTHESIS language description. - Moscow: IPI RAN, 1993. - 110 p.
3. Kalinichenko L.A., Stupnikov S.A., Martynov D.O. SYNTHESIS: a Language for Canonical Information Modeling and Mediator Definition for Problem Solving in Heterogeneous Information Resource Environments. Fourth edition of the SYNTHESIS language description Moscow: IPI RAN, 2007. - 171 p.
4. В.Н. Захаров, Л.А. Калиниченко, И.А. Соколов, С.А. Ступников Конструирование канонических информационных моделей для интегрированных информационных систем // *Информатика и ее применения*. – М.: ИИИ РАН, 2007. – Т. 1, №. 2. – С. 13 – 38.
5. Abrial J.-R. *The B-Book - Assigning Programs to Meanings*. - Cambridge: Cambridge University Press, 1996.
6. Kifer M. Rule Interchange Format: the Framework // *Web Reasoning and Rule Systems: Proc. of the Second International Conference, LNCS 5348*. – Berlin-Heidelberg: Springer Verlag, 2008.p. 1-11
7. RIF Overview // W3C Working Draft. – <http://www.w3.org/TR/2010/WD-rif-overview-20100511> - 2010.
8. List of Rule Systems // http://www.w3.org/2005/rules/wg/wiki/List_of_Rule_Systems.html
9. RIF Working Group Web Page // http://www.w3.org/2005/rules/wiki/RIF_Working_Group

10. Kowalski R. Reasoning with Conditionals in Artificial Intelligence // <http://www.doc.ic.ac.uk/~rak/papers/conditionals.pdf> - Department of Computing, Imperial College London, 2009.
11. Minker J. Logic and databases: a 20 year retrospective // LNCS 1154 – Springer-Verlag, 1996.
12. Ramakrishnan R., Ullman J. A Survey of research on Deductive Database Systems // Technical Report. - Stanford University, 1993.
13. Gallaire H., Minker J., and Nicolas J. Logic and databases: A deductive approach // ACM Computing Surveys. – 1984. - P. 153-185.
14. K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge // Foundations of Deductive Databases and Logic Programming. - Los Altos, CA: Morgan Kaufmann Publishers, 1988. – P. 89-148.
15. Abduction // <http://iph.ras.ru/page54852159.htm>
16. N. Bidoit and C. Froidevaux. Minimalism subsumes default logic and circumscription // In Proc. of the LICS-87. – 1987. – P. 89-97.
17. Default logic // http://en.wikipedia.org/wiki/Default_logic
18. Wan1 H., Grosfod B., Kifer M., Fodor P., Liang S. Logic Programming with Defaults and Argumentation Theories // Logic Programming, LNCS 564. – 2009. - P. 432-448.
19. Stable model semantics // http://en.wikipedia.org/wiki/Stable_model_semantics
20. Reiter R. A logic for default reasoning // Artificial Intelligence.- 1980. - V. 13. – P. 81-132.
21. Gelfond M., Lifschitz V. Classical negation in logic programs and disjunctive databases // New Generation Computing. – 1991. - V. 9. – P. 365-385.
22. Chitta Baral. Answer set programming: knowledge representation, reasoning and declarative problem solving using AnsProlog // <http://www.public.asu.edu/~cbaral/tutorial-dallas-june-04.pdf> – 2004.
23. Marek W., Truszczyński M. Autoepistemic logic // Journal of the ACM. – 1991. - V. 38, N. 3. – P. 587 – 618.
24. Gelfond M. On stratified autoepistemic theories // Proc. of AAAI-87. – 1987. – P. 207-211.
25. Michael Kifer, Jos de Bruijn, Harold Boley, Dieter Fensel. A realistic architecture for the semantic web // Proc. of the 1st International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML2005), LNCS 3791. - Springer-Verlag, 2005. - P. 17–29.
26. A. Van Gelder, K.A. Ross and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs // Journal of the ACM. – 1991. – V.38, N 3. - P. 620-650.
27. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming // Logic Programming: Proceedings of the Fifth Conference and Symposium. - 1988. - P. 1070–1080.
28. Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner. Answer Set Programming: A Primer // Proc. of the 5th International ReasoningWeb Summer School 2009, LNCS 5689. – Springer-Verlag.
29. R. C. Moore. Semantical considerations on nonmonotonic logic // Artificial Intelligence. – 1985. – V. 25. - P. 75-94.
30. Marek W., Truszczyński M. Stable models and an alternative logic programming paradigm // The Logic Programming Paradigm: 25 Year Perspective. - Springer Verlag, 1999. – P. 375 – 398.

31. I. Niemelä, P. Simons and T. Soininen. Stable Model Semantics of Weight Constraint Rules // Proc. of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning. - Springer-Verlag, 1999.
32. Lifschitz V. What Is Answer Set Programming? // Proceedings of the 23rd national conference on Artificial intelligence. – AAAI Press, 2008. - V. 3. - P. 1594-1597.
33. System DLV // <http://www.dbai.tuwien.ac.at/proj/dlv/>
34. Kifer M. Nonmonotonic Reasoning in FLORA-2 // Logic Programming and Nonmonotonic Reasoning, LNCS 3662. - Springer-Verlag, 2005. - P. 1-12.
35. F-logic (frame logic) // <http://en.wikipedia.org/wiki/F-logic>
36. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages // Journal of ACM. - 1995. - V. 42. - P. 741–843.
37. Guizhen Yang, Michael Kifer, Hui Wan and Chang Zhao, FLORA-2: User's Manual // <http://flora.sourceforge.net/docs/floraManual.pdf> - 2008.
38. Harold Boley, Michael Kifer, Paula-Lavinia Patranjan, Axel Polleres. Rule Interchange on the Web Reasoning // Web 2007, LNCS 4636. - Springer-Verlag, 2007. - P. 269–309.
39. RIF Use Cases // http://www.w3.org/2005/rules/wg/wiki/Use_Cases.html
40. General Use Case Categories // http://www.w3.org/2005/rules/wg/wiki/General_Use_Case_Categories
41. RIF Use Cases and Requirements // W3C Working Draft. - <http://www.w3.org/TR/rif-ucr> - 2008.
42. RIF Framework for Logic Dialects // W3C Proposed Recommendation. Eds. H. Boley, M. Kifer. – 2010. – <http://www.w3.org/TR/2010/PR-rif-fld-20100511/>
43. Benjamin N. Grosf, Ian Horrocks, Raphael Volz, Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic // Proc. of WWW2003. – 2003.
44. G. Antoniou, C. V. Damasio, B. Grosf, I. Horrocks, M. Kifer, J. Maluszynski, P. F. Patel-Schneider. Combining Rules and Ontologies. A survey // Project REVERSE Report. – 2005.
45. Implementations – RIF // <http://www.w3.org/2005/rules/wiki/Implementations>
46. RIF RDF and OWL Compatibility // W3C Candidate Recommendation. - <http://www.w3.org/TR/2009/CR-rif-rdf-owl-20091001/> - 2009.

Abstract

Analysis of motivation, objectives and approaches of the project for the unification of the rule based languages

Kalinichenko L.A., Stupnikov S.A.

The paper is devoted to analysis of the W3C standard RIF (Rule Interchange Format) that is provided for the interoperability of various rule based systems by the introduction of the extensible family of unified languages (dialects) oriented on creation of semantic preserving mappings of rule based languages of various systems into the dialects. To characterize the motivation for the RIF project the paper contains a short survey of development and application of rule based languages and systems in the areas of knowledge representation, deductive databases, logical reasoning. An analysis of various

semantics of logical rule based languages that influenced the RIF decisions is included into the paper. Main classes of application cases of the interoperable rule based programs used for development of the requirements for RIF are considered. Finally the main decisions of the RIF project are overviewed.