Ф

УДК 681.3 **DOI** 10.24412/2413-7383-2025-4-39-241-253

Д.А. Филипишин, С.А. Зори
Федеральное государственное бюджетное образовательное учреждение высшего образования «Донецкий национальный технический университет», г. Донецк, 283001, г. Донецк, ул. Артема, 58

МЕТОД РЕЗОЛЮЦИИ КАК СРЕДСТВО РАСШИРЕНИЯ МАШИНЫ ВЫВОДА ОНТОЛОГИЙ*

D.A. Filipishin, S.A. Zori
Federal State Budgetary Educational Institution of Higher Education "Donetsk National Technical University", 283001, Donetsk, Artema str, 58

THE RESOLUTION METHOD AS A MEANS OF EXTENDING THE ONTOLOGY INFERENCE MACHINE

В статье рассматривается применение решателя языка Prolog как средства расширения машины вывода (ризонера) для построения онтологий предметных областей. Предложен новый подход к решению продукционных задач в рамках редактора онтологий, рассматриваемого как интегрированная среда разработки (IDE). Проанализирована перспектива использования решателя языка Prolog, основанного на языке предикатов математической логики дизъюнктов Хорна, семантическими связями и аксиоматикой бинарных отношений, свойственных онтологиям. Предложено решение имеющихся недостатков в обеих системах с целью создания новой онтологической системы, включающей возможности обоих подходов.
Ключевые слова: онтология, Prolog, IDE, reasoner, решатель, продукции, загадка Эйнштейна, игра Холмс.

The article discusses the use of the Prolog language solver as a means of extending the inference machine (reasoner) to build domain ontologies. A new approach to solving production tasks within the framework of the ontology editor, considered as an integrated development environment (IDE), is proposed. The perspective of using a Prolog language solver based on the predicate language of the mathematical logic of Horn disjuncts, semantic connections and axiomatics of binary relations inherent in ontologies is analyzed. A solution to the existing shortcomings in both systems is proposed in order to create a new ontological system that includes the capabilities of both approaches.
Key words: ontology, Prolog, IDE, reasoner, solver, products, Einstein's Riddle, the Holmes game.

* Работа выполнена в рамках молодежной научной лаборатории «Искусственный интеллект» (Код НИОКТР FRFR-2024-0010)

Введение

Любой язык программирования всегда ориентирован на определенный круг задач, при решении которых он наиболее эффективен. Для языка Prolog типичными являются проекты, связанные с разработкой систем искусственного интеллекта – это различные экспертные системы, системы планирования, программы-переводчики, интеллектуальные игры и т.п. Prolog обладает достаточно мощными средствами, позволяющими извлекать информацию из баз данных и знаний. При этом его методы поиска принципиально отличаются от «традиционных». Кроме этого, Prolog часто используют в задачах, связанных с манипулированием на естественном языке [1].

Существует не меньше десятка зарубежных инструментов онтологического инжиниринга, поддерживающих формализмы для описания знаний и использующих машины вывода из этих знаний. Для анализа возможностей инструмента онтологии был выбран редактор онтологий Protégé. Этот редактор имеет исчерпывающую документацию, поддерживается значительным сообществом, состоящим из разработчиков и ученых, правительственных и корпоративных пользователей, использующих его для решения задач, связанных со знаниями. Protégé доступен для свободного скачивания с официального сайта вместе с плагинами и онтологиями [2].

Protégé – локальная, свободно распространяемая Java-программа, предназначенная для создания, просмотра и редактирования онтологий явных моделей предметной области с целью включения этих моделей в программный код. Редактор онтологий Protégé позволяет разворачивать иерархическую структуру абстрактных или конкретных классов и слотов. Структура онтологии аналогична иерархической структуре каталога. На основе онтологии Protégé может генерировать формы получения знаний для введения экземпляров классов и подклассов. Имеет графический интерфейс. Поддерживает использование языка OWL. Генерирует HTML-документы, отображающие структуру онтологий. Использует фреймовую модель представления знаний OKBC – позволяет редактировать модели предметных областей, представленных в форматах UML, XML, SHOE, DAML+OIL, RDF / RDFS и т.п. (а не только в не в OWL) [3].

Таким образом, Prolog – это язык и система логического программирования, основанные на языке предикатов математической логики дизъюнктов Хорна [4]. Хорновский дизъюнкт [4] в контексте математической логики — это дизъюнктивный одночлен с не более чем одним положительным литералом. Литерал в теории булевых функций и логике высказываний — булева формула, имеющая вид x (истина) или $\bar{x}$ (ложь) для некоторой переменной *x*. Хорновские дизъюнкты могут быть пропозициональными формулами, либо формулами первого порядка, в зависимости от того, рассматриваются ли пропозициональные литералы или литералы первого порядка.

Резолюция цели с определённым дизъюнктом для получения новой цели является основной для правила вывода в SLD-резолюции (*Selective Linear Definite resolution*), используемого для реализации логического программирования и языка программирования Пролог. В логическом программировании определённый дизъюнкт используется как процедура редукции цели. Например, дизъюнкт $\neg p \vee \neg q \vee \cdots \vee \neg t \vee u$ работает как процедура: «*чтобы показать u, показать p, q, ⋯ и t*».

Решатель (программа для решения задач) в языке Prolog устроен на основе фактов, правил и запросов. Prolog автоматически управляет решением задачи, стараясь найти все возможные наборы значений, удовлетворяющие запросу. Для это используется метод поиска с возвратом. Рассмотрим особенности и недостатки языка Prolog.

Prolog не алгоритмический, программа в нем не содержит явных алгоритмических конструкций типа условных или циклических операторов. Не выполняет команды по порядку, а строит выводы на основе правил, условий или функций без состояния. Это требует другой модели мышления. Один и тот же код может вести себя по-разному из-за порядка строк, типов или связки с платформой, что усложняет отладку. Механизм вывода (решатель) пытается найти решение «в лоб», чем часто вызывает критические ошибки исполнения.

Онтологическая система представляет собой инструмент поиска скрытых знаний на основе семантических связей и множества аксиом, заданных на отношениях. Редактор онтологий Protégé, являющийся наиболее перспективным представителем программной системы, основанной на дескриптивном описании онтологии (OWL), не имеет в своем арсенале средств решения продукционных задач.

Таким редактором выступает система Ontorion Fluent Editor, от польских разработчиков, которая позволяет управлять онтологией скриптовым описанием с использованием реального английского языка, а также использовать «if… then…» для фактов, в соответствии с рисунком 1.

```
If an airplane has-value-mcr lower-or-equal-to 0.7 and a wing-profile has-
value-c equal-to 15 then the wing-profile useds-on the airplane .
```

Рисунок 1 - Пример использования «if then» и союза «and» в качестве конъюнкции [5]

Ризонер (машина вывода в онтологии) на текущий момент не умеет обрабатывать множественные решения и, несмотря на концепцию об открытости мира, требует значительного уточнения для своей работы. Таким образом, является актуальной задача разработки инструмента построения дескриптивных онтологий (OWL) с использованием решателя языка Prolog, методом резолюций.

Цель работы и постановка задачи

Целью работы является анализ практического расширения машины вывода ризонера в редакторе онтологий, представленного как интегрированная среда разработки (IDE), методом резолюции, т.е. средствами решателя Prolog. Пример решения продукционной задачи продемонстрирован разбором решения общеизвестной задачи-игры «HOLMES», в соответствии с рисунком 2, которая по сути является усложненной «загадкой Эйнштейна», в соответствии с рисунком 3, имеющей дополнительные ограничения на расстановку.

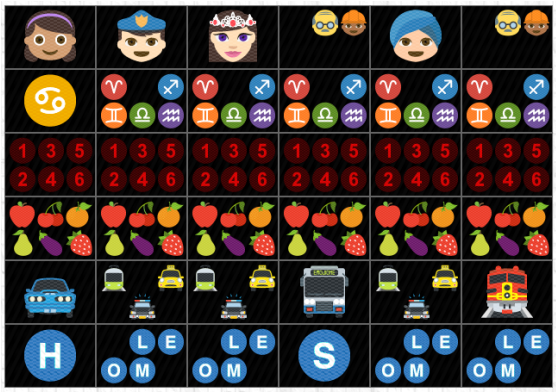
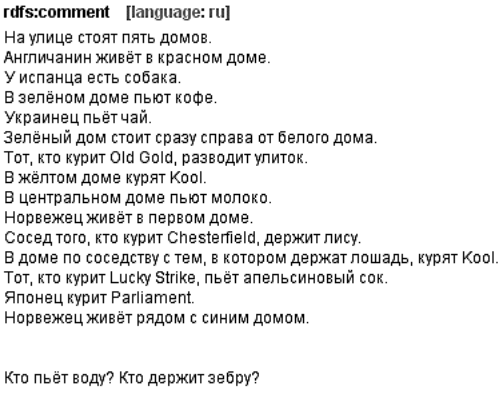


Рисунок 2 - Пример продукционной задачи - игра Шерлок
в режиме сложности «легко»

Задача Эйнштейна (загадка Эйнштейна, головоломка о зебре) – логическая задача, по легенде созданная Альбертом Эйнштейном в годы его детства. Также бытует мнение, что она использовалась Эйнштейном для проверки кандидатов в ассистенты на способность к логическому мышлению. Доказательств авторства задачи нет. Иногда автором головоломки называют Льюиса Кэрролла, но в условии упоминаются марки сигарет, которые не существовали при жизни Кэрролла и во времена детства Эйнштейна.


rdfs:comment [language: ru]
На улице стоят пять домов.
Англичанин живёт в красном доме.
У испанца есть собака.
В зелёном доме пьют кофе.
Украинец пьёт чай.
Зелёный дом стоит сразу справа от белого дома.
Тот, кто курит Old Gold, разводит улиток.
В жёлтом доме курят Kool.
В центральном доме пьют молоко.
Норвежец живёт в первом доме.
Сосед того, кто курит Chesterfield, держит лису.
В доме по соседству с тем, в котором держат лошадь, курят Kool.
Тот, кто курит Lucky Strike, пьёт апельсиновый сок.
Японец курит Parliament.
Норвежец живёт рядом с синим домом.

Кто пьёт воду? Кто держит зебру?

Рисунок 3 - Пример «загадки Эйнштейна»

Для решения «загадки Эйнштейна» необходимо создать 5 классов по 5 индивидов к каждому из них, задать 6 бинарных отношений со свойствами «functional» и «inverse functional», связывающих классы между собой, описать перечисления индивидов для каждого из класса, указать что сущности не пересекаются между собой («different» и «disjoint»), задать аксиоматику и указать факты какое «животное» у кого имеется и т.п., а также добавить посылки для исключения вариантов бездействия индивидов. Предлагаемая система модификаций редактора онтологий в интегрированную среду проектирования (IDE) позволяет устранить ряд недостатков: каждый индивид создается как экземпляр класса (не нужно назначать класс после его создания), исключает необходимость указывать перечисления индивидов для классов, вся основная логика семантических связей переносится на аксиоматику отношений (свойства domain и range), посылкой является сам запрос, но при наличии достаточных оснований для рассуждения (фактов). Такой эффект достигается методом «сужения» слишком открытого мира в Protégé, с доработкой существующих инструментов построения онтологий и машины вывода.

Формальная модель онтологии в рамках предлагаемой интегрированной среды разработки (IDE) представляет собой дополненный кортеж формальной модели онтологии в моделях, основанных на дескриптивной логике (OWL):

$$O =< C, R, H_c, H_R, I, A, H_A >, \quad (1)$$

где C – множество классов; R: CxC – множество свойств классов; H_c: CxC – иерархия классов; H_R: RxR – иерархия отношений; I – множество индивидов; A – множество аксиом над классами и ограничений отношений; H_A – множество ограничений аксиом отношений. Отличие формальных моделей OWL [6] и предлагаемой (в рамках IDE) в наличии (H_A) множества ограничений не только для отношений, а и аксиом, т.е. наличии множества условий для семантической связи одного отношения для разных пересечений классов, что на сегодняшний день нет ни в одном из редакторов онтологий.

Таким образом, при обработке запроса «*Аня имеет флешку*» или вопрос на естественном языке «*Какую флешку имеет девочка Аня?*», при описании онтологии

класс-«девочка»/индивид-«Света» → предикат-«имеет» (domain «девочка» - range «флешка» - свойство «not functional» или «один к одному») → *класс-«флешка»* и наличии фактов наличия флешек у девочек *«Аня» имеет НЕ-«красная», «Варя» имеет НЕ-«синяя» и «Света» имеет НЕ-«красная» и НЕ-«белая»*, обратится к имеющемуся множеству индивидов «*флешка*» и методом жадного алгоритма получит ответ «*белая*».

Индивиды для класса «девочка»: *Аня, Варя, Света*. Индивиды для класса «флешка»: *синяя, белая, красная*. Согласно аксиоматике отношения «*имеет*» в онтологии сказано: одна «*флешка*» принадлежит одной «*девочке*». Ответ получен так: у Светы «*синяя*», у Ани НЕ-«*красная*», а значит «*синяя*» или «*белая*», «*синяя*» у Светы, значит у Ани «*белая*». Из примера видно, что посылку (аксиому) «*девочка*» имеет «флешка» или «*ЛЮБАЯ девочка имеет хотя бы одну флешку*» более задавать не нужно. Также система работает с тем набором данных, который добавлен в множество классов-концептов.

Такой подход практически делает невозможным применение существующего инструмента онтологической системы, а именно хранение онтологии в домене и подключение метаонтологии, как например DBpedia [7]. Но значительно упрощает процесс построения онтологий и снижает порог вхождения для обычного пользователя. Рассмотрим краткое решение задачи-игры «HOLMES», для анализа необходимых средств при решении подобных продукционных задач (Рис.2).

Решатель языка Prolog и предлагаемое решение задачи-игры «HOLMES»

Рассмотрим формальный подход методу резолюции. Базовой конструкцией теории является схема отношения, выступающая некоторым аналогом классического предиката [8].

Выражение вида [8]:

$$f: a_1, \dots, a_n \rightarrow a_0, \qquad (2)$$

где a_i, $i = \overline{1,n}$ – имена величин, называется функциональной связью (ФС). В записи (2) f – это имя ФС, a_i, $i = \overline{1,n}$ – аргументы, a_0 – результат ФС.

При задании интерпретации [3] структура вида:

$$S(r) = (r)(S_{01}(a_{01}), \dots, S_{0N0}(a_{0N0}), \\ if\, p_1 \Rightarrow S_{11}(a_{11}), \dots, S_{1N1}(a_{1N1}) \dots, \\ p_k \Rightarrow S_{k1}(a_{k1}), \dots, S_{kNk}(a_{kNk}) fi|filter \qquad (3)$$

фиксирует некоторое подмножество размеченного декартова произведения [8], определяемое ФС из набора *filter*. Подразумевается, что в подмножество попадают только те кортежи (наборы) атрибутов схемы, значения которых удовлетворяют всем ФС этой схемы. Неформально, мы можем ставить два типа вопросов: (*i*) удовлетворяет ли набор конкретных значений атрибутов схеме, и (*ii*) какие наборы удовлетворяют схеме, если зафиксированы значения некоторых из атрибутов. Принимаются явные соглашения, связанные с использованием предопределенных отношений (порядка и т.д.) для первичных схем [4].

Таким образом, иерархия типов стоится в виде совокупности объявлений в форме [4], близкой к фиксации башни функционалов, когда каждый элемент явно дефинируется предками:

$$S_0(r_0) \leftarrow (r_0).\left(S_{01}(r_{01}), \dots, S_{0k}(r_{0k})\right); \qquad (4)$$
$$(r_0).S_{01}(r_{01}) \leftarrow (r_0.r_1).(S_{11}(r_{11}), \dots); (r_0).S_{0k}(r_{0k}) \leftarrow \dots$$

При этом полагают, что в терминальных конструкциях все $S_{ij} \in E$ («построение от базиса»); считается также, что в записи *S(r)*, синтаксически *r* представляет собой место индивидной подстановки, а семантически форма сообщает, что конкретизация *r* удовлетворяет отношению *S* (набор *filter* индуцирует функцию, принадлежности [9]).
Важно отметить, что применение для спецификации программы логического языка даже первого порядка неизбежно приводит нас к проблеме, связанной с результатом Геделя. На практике это вызывает необходимость искусственного торможения резолютивного вывода (управления последним) с помощью внелогических операций, подобных операции усечения («cut» или «!») в языке Пролог. Очевидная эклектичность вызывает не менее очевидное недоверие к инструменту.

Предлагается следующая трактовка [4] стандартной клаузы языка Пролог (интерпретация фактов и целевого утверждения остается стандартной). Пусть имеет место:

$$S(r) :- S_1(r_1), \dots, S_m(r_m), Q_1(t_1), Q_N(t_N), \quad (5)$$

где $S_i(r_i), i = \overline{1, M}$, удовлетворяет (3), т.е. выступают «аналогами» классических предикатов, $Q_j(t_j), j = \overline{1, N}$ имеют форму ФС, причем для любого $j = \overline{1, N}, t_j \subseteq \cup r_i (i = \overline{1, M})$. Приведенные соглашения практически полностью покрывают синтаксис любой стандартной Пролог-системы.

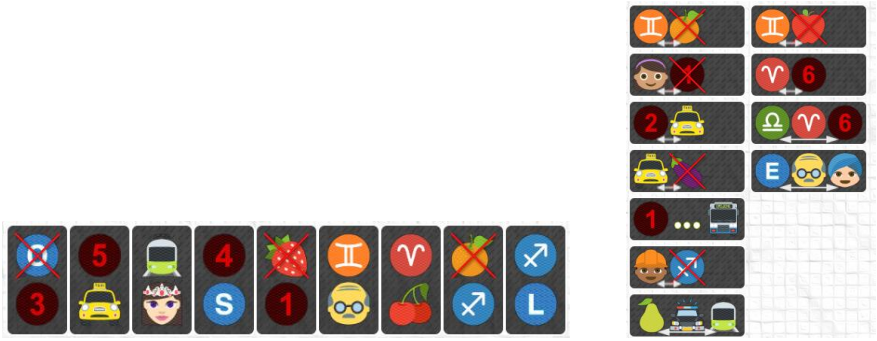
Таким образом, продемонстрирована возможность сведения стандартных предложений языка Пролог к конструкциям языка теории С-моделей. В [12] показана полнота этой теории, приведена схема алгоритма логического вывода, а также установлено, что сложность такого алгоритма не превосходит $O(|N|^3)$, где N – константа, соответствующая длине исходной спецификации. Более того, используемый алгоритм реализует обратный вывода в стиле С.Ю. Маслова (см. [12], Приложение 1).

Как уже говорилось выше, действия решателя языка Prolog напрямую зависят от порядка описания фактов, чего нет у ризонера, т.к. механизм вывода онтологий основан на аксиоматике отношений и порядок фактов не играет существенной роли, определяя лишь порядок загрузки фактов в базу знаний (справедливо только для Ontorion Fluent Editor, в Protégé нет скриптового управления и факты добавляются вручную каждому индивиду). В современных программных системах с элементами искусственного интеллекта [4] обязательной составляющей является подсистема объяснения полученного вывода, а отсутствие в алгоритме бэк-трекинга делает процесс объяснения более простым и естественным по сравнению с традиционным подходом.

Вернемся к продукционной задаче-игре «Холмс». Эта игра была создана по мотивам олдскульной игры "Шерлок", первоначально написанной Эвереттом Кейзером для MSDOS более 20 лет назад. Шерлок – это компьютеризированная версия логических головоломок, в которой вам предлагается ряд подсказок, которые помогут вам определить точное расположение всех изображений на игровом поле. Каждый ряд игрового поля содержит однотипные изображения (лица, дома, цифры, фрукты, уличные знаки, буквы и т.д.). Компьютер скремблирует расположение элементов в каждом ряду (не показывая вам их расположения), а затем представляет вам набор графических подсказок, описывающих расположение различных изображений. Вы используете подсказки, чтобы определить, где чего быть не может (и где что должно быть), пока не узнаете, где находятся все изображения.

Данная задача описана матрицей 6 на 6, в каждой ячейке которой по итогу должен находиться только 1 элемент. Расстановка элементов осуществляется с помощью продукционных правил сопоставления («в одном столбце» или НЕ-«в одном столбце») и расстановки (элемент «рядом» и НЕ-«рядом», в соответствии с рисунком 4б, «в одном»

или НЕ-«в одном» столбце (Рис.4а), а также описание смежности в соответствии с рисунком 4б, т.е. расстановки 3х элементов рядом друг с другом, которое может быть прямым и обратным, и правило следования «символ предшествует символу»).



а) расположение по столбцам б) описание смежности

Рисунок 4 - Продукционные правила для решения задачи-игры Шерлок

В «загадке Эйнштейна», отношение «*в одном столбце*» описано как «мужчина курит сигареты» или «мужчина держит животное», отношение смежности и «рядом» описано также, но конкретно для каждого индивида или класса. В целом можно считать, эти две загадки равноценными задачами построения онтологии, за исключением применения правил исключения вариантов и потребности разрешения противоречивых вариантов для игры «шерлок» даже в режиме сложности «легко».

Приведенные на рисунке 4 ограничения легче описать в виде предикатов или фактов, нежели разбирать на аксиомы и искусственно ограничивать пространство решения для ризонера в онтологии. Для решения задачи «Шерлок» в режиме «сложно», в котором изначально не заполнена ни одна из ячеек, жизненно необходимо «пробовать» подставлять условия в виду наличия достаточного количества противоречивых подстановок, перекладывая возможность «ошибаться» на машину вывода. Потому как описать множественное решение согласно приведенным в игре ограничениям имеющимися инструментами редактора онтологий Protégé или Ontorion Fluent Editor не является возможным. Для этого следует применять механизм обратного вывода в стиле С.Ю. Маслова (см. [12], Приложение 1), на основе решателя языка Prolog. Но не просто скопировать решатель у языка Prolog, а расширить имеющуюся машину вывода включающей возможности обеих систем (ризонер в Protégé и решатель в Prolog) продукционными правилами (if…then…) как в редакторе онтологий Ontorion Fluent Editor.

Таким образом, полученная система вывода позволит не просто перебирать множество предикатов согласно описанным аксиомам онтологии, а и реагировать на дополнительные ограничения применяя механизм обратного вывода, свойственно Prolog. И как уже было сказано ранее, вводимый механизм также потребует решения проблемы, связанной с результатом Геделя. А именно необходимостью продумать систему управления машиной перебора, например, аналогично языку Prolog, условием отсечения:

```
Name=diana,     # условие отсечения
!.              # отсечение
```

Рассмотрим одно из возможных решений задачи-игры «Шерлок», в соответствии с рисунком 2. Перед началом разбора, рассмотрим предложенную матрицу на предмет подготовленных «ответов» (единичных значений в представленных ячейках). Как

видим, в соответствии с рисунком 2, в нашем случае, уже единично определены: «девочка», «полицейский», «царевна», «индус», «рак», «машина», «автобус», «поезд», «H» и «S». Эти элементы будут служить опорными точками для применения указанных, в соответствии с рисунком 4 ограничений.

Далее, согласно правилу расположения «по середине», т.е. элементы, которые указаны в последовательностях в середине, можно убрать с краев матрицы. Для представленного примера это «Овен» и «Мужчина». Правила расположения, в соответствии с рисунком 4а, являются взаимозаменяемыми, например, «О» не может находиться в одном столбце с «3», что означает и обратное отношение: «3» не может находиться в одном столбце с «О». Правила расстановки в соответствии с рисунком 4б, также являются взаимозаменяемыми. Например, рядом с «Близнецами» НЕ-может находиться «апельсин», что также означает: рядом с «апельсином» НЕ-может находиться «Близнецы» (т.е. слева или справа).

Далее проверим соответствия по столбцам: «трамвай – царевна», «4 - S» и «Близнецы - Мужчина». После чего можно снова приступать к правилам расстановки. Рядом с «Близнецы» уберем «апельсины» и «яблоки». Также нам известно, где находится «девочка», поэтому уберем рядом «1».

Следуя правилам расстановки, предположим, что мы не знаем расположение «2», «такси», «патиссон». «Автобус» занимает свое единичное место, поэтому можно четко определить, что «1» справа от него быть не должно, в соответствии с рисунком 5. Рядом с «шахтер» не может быть «стрелец», но под ним «стрелец» находиться может. Правило «груша – полицейская машина - трамвай» также можно однозначно определить, в виду того, что «трамвай» единично определен и рядом с ним «полицейская машина» может находиться только слева. Отметим «груша» и «полицейская машина».

Рисунок 5 - Правило расстановки-следования: «1» предшествует «автобус»

Таким образом, методом исключения нам стало известное расположение «такси». Отметим «2» справа, т.к. слева находится «4» и уберем по бокам «патиссон». Также в столбце с «такси» отметим «5» согласно правилу расположения столбцов «5 - такси». Далее, согласно правилу «вишня - овен», можно удалить «вишня» там, где нет «Овен». А именно, там, где расположен «Близнецы». Также можно убрать «стрелец», там, где нет «L» и наоборот. Выполним правило расстановки «Е – мужчина - индус», т.к. мы знаем расположение «мужчины» и «индуса». Здесь же можно убрать «стрелец», т.к. в этом столбце больше нет «L».

Рисунок 6 - Расстановка оставшихся элементов

Далее, исходя из правила «Овен» рядом с «6», уберем «Овен», который рядом с «4» и «2». Также уберем «вишня» там, где нет «Овен». Согласно правилу «весы – овен - 6», можно убрать «Весы», которые находятся справа от «Близнецы», т.к. этот знак должен быть рядом с «Овен».

Теперь мы знаем расположение «Стрелец». Отметим «L» и уберем «апельсин». Далее необходимо совместить два правила «овен - 6» и «весы – овен - 6», чтобы оба правила подходили (Рис. 6). Согласно описанным правилам, мы можем расположить только в обратном порядке «6 – овен - весы», т.к. в прямом нет возможности отметить «6». После чего, отметим «вишня» в столбце с «Овен», уберем «клубника» в столбце с «1» и, по итогу, уберем «О» в столбце с «3». Загадка решена (Рис. 7). Буквы внизу составляют слово «HOLMES», т.е. название игры.

Также можно ускорить процесс решателя добавив правило исключения, а именно убирать первый символ, который должен быть «в том же столбце» в тех местах, где уже нет второго символа сверху или снизу, и наоборот. Правила можно объединять, для случаев наличия смежных расстановок, через какой-то общий символ, т.е. если в подмножестве невозможна какая-то пара, то следует исключить всю цепочку из этого столбца.

Сюда же относится правило исключения подстановок, когда система может не применять правила смежности по ряду или столбцу, а просматривать количество смежных ячеек, в которых можно применить рассматриваемое правило смежности. Если количество этих ячеек равно 1, то следует применить это правило для этой ячейки как единственно подходящий вариант.

Для правил расстановки (3х символов) можно применять правило, не задумываясь в том случае, если один из крайних символов является единично определенным и находится в крайнем столбце. Тогда остальные символы просто больше некуда записывать.

Решение продукционных задач в рамках предлагаемого подхода сводится к добавлению продукций «if…then…» для фактов в онтологии и дальнейшего перебора условий с возвратом. Также необходимо помечать примененные условия, с целью уменьшения множества оставшихся не выполненных условий. Дополнительно следует реализовать алгоритм, проверяющий достоверность полученных результатов, чтобы в каждой «ячейке» был единичным только тот символ, которого нет в других ячейках в этом же ряду.

Приемы решения логических задач базируются на выдвижении некоторой гипотезы и ее подтверждении (или опровержении) согласно некоторой методике. Таким приемы соответствуют вычислению ответа на основе связанной многократной работы двух модулей: построение некоторого состояния, проверка полученного состояния. Подход вычисления ответа, а точнее – построения ответа согласуется с решением задач на языке Prolog [17].

Машинный способ решения описанной задачи заключается в циклической проверке правил до тех пор, пока хотя бы одно правило было применено из перечисленных множеств, в нашем случае это 2 множества правил. Тип множества правил последовательно переключается с целью поиска подходящего решения. В случае, если в ходе перебора правил система не может найти решения ни в одном из множеств правил, следует применить принудительную подстановку из имеющихся (не выполненных) правил расстановки или ином доступном, если все правила расстановки уже были задействованы. Далее проверить правила сходимости относительно других правил и, в случае обнаружения несовместимости, отменить принятое ранее решение и ряд изменений, после чего применить следующее правило расстановки и повторить проверку совместимости до достижения желаемого результата. Такой способ не лишен недостатков и требует значительной доработки, но позволяет решить большую часть поставленных продукционных задач в рамках семантической модели.

Использование языка Пролог в качестве базы позволяет использовать его в качестве интеллектуального («разумного») инструмента решения задач, в виду доступности синтаксиса и простоты интерпретации семантических конструкций. Работы в этом направлении ведутся со времени основания логического языка, и на сегодняшний день разработано множество программ (баз знаний фактов и правил), которые развиваются вместе с прикладными научными направлениями [17], [19-22].

Чтобы закодировать в терминах языка Пролог процедуру проверки на соответствие (согласование), нужно описать правилами все требования, предъявляемые к искомому состоянию. Примеры таких процедур фактов и правил рассмотрены в различных прикладных работах по логическим языкам [18], [19].

Для перебора элементов внутри множеств был предложен несколько иной подход к использованию конструкции циклического перебора на базе существующей конструкции «*for..in*», записываемый ключевым словом «*loop*» в правилах семантики внутреннего скриптового языка.

Рассмотрим пример предлагаемой конструкции цикла:

```
bool k = false;
loop(rule in _rules; k == true; k = false){
        if( is_acceptable(rule) )  k = true;
}
```

где *rule* - имя переменной, которая является указателем на объект внутри массива или кортежа; *_rules* – массив или кортеж объектов для перебора; «*k==true*» - условие последующего выполнения, которое выполняется аналогично конструкции «*do while*»; «*k = false*» - инструкция, которая выполняется в конце каждой итерации.

Главным отличием от стандартной конструкции *for* является механизм перебора всех элементов внутри переданного массива (или кортежа) за каждую итерацию. Проверяется наличие элементов внутри массива (или кортежа), выполняется инструкция шага, выполняется тело циклической конструкции, проверяется условие продолжения и цикл повторяется до тех пор, пока выполняется условие продолжения или не будет вызван оператор «break». Данный тип циклической конструкции позволяет значительно эффективнее обрабатывать множество правил целиком, нежели существующие виды циклов. Эта особенность в значительной мере позволяет упростить логику работы с элементами множеств, но также добавляет ряд недостатков, которые могут быть связаны с необходимостью последовательной обработки каждого элемента, аналогично стандартному циклу «*for..in*» или «*for..of*», что вероятно потребует ввода ряда дополнительных переменных.

Таким образом, можно подвести итог, что рассмотренный пример решения задачи-игры «Холмс», которая является несколько усложненным вариантом «загадки Эйнштейна» (общеизвестным примером для обучения построения онтологий OWL), наглядно демонстрирует потребность внедрения в машину вывода онтологии (ризонер) механизм перебора с возвратом, аналогичный решателю языка Prolog. Это позволит вывести процесс построения онтологий предметных областей на новый уровень и решать более сложные задачи искусственного интеллекта, нежели на сегодняшний день способен OWL и OWL2.

Заключение

В статье рассмотрена проблема отсутствия необходимых инструментов в онтологии для решения продукционных задач. В качестве решения описанной проблемы предложено расширить машину вывода редактора онтологий, представленного как интегрированная среда разработки (IDE), с помощью метода резолюций, аналогичного решателю языка Prolog.

К перспективам развития работы можно отнести внедрение в онтологии не рассмотренные в данной статье инструменты языка Prolog, позволяющие проектировать экспертные системы, с целью реализации инструментов принятия решений и дополнения инструмента построения онтологий предметных областей с «собственным поведением».

Список литературы

1. Хабаров С.П. Интеллектуальные информационные системы. PROLOG-язык разработки интеллектуальных и экспертных систем: учебное пособие. СПб. СПбГЛТУ, 2013. 138 С.
2. Цуканова, Н.И. Онтологическая модель представления и организации знаний: учебное пособие для вузов. М.: Горячая линия – Телеком, 2015.272 с.
3. Гаркуша, Д.А. Функциональные особенности реализованных онтологических платформ // Проблемы искусственного интеллекта. 2023. № 4 (31). 4-11. http://search.rads-doi.org/project/14374/object/210537 doi: 10.34757/2413-7383.2023.31.4.001.
4. Новосельцев В.Б. Эффективный нерезолютивный вывод для ограниченного исчисления хорновских дизъюнктов // Известния Томского политехнического университета, 2008. Т. 312 №5. С. 94 – 97.
5. Боргест Н.М., Орлова А.А. Онтологический редактор Fluent Editor: учебно-методическое пособие к лабораторным работам / сост.: Н.М. Боргест, А.А. Орлова. Самара: Изд-во Самарского университета, 2017. 44 С.
6. Mendes P. N. DBpedia: a multilingual cross-domain knowledge base / P. N. Mendes, M. Jakob, and C. Bizer // LREC. 2012. Pp. 1813–1817.
7. Цуканова, Н.И. Онтологическая модель представления и организации знаний: учебное пособие для вузов. М.: Горячая линия – Телеком, 2015. 272 С.
8. Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. СПб.: БХВ-Петербург, 2003. 992 с., ил. ISBN 5-94157-156-9
9. Вирт Н. Структурное программирование. М.: Мир, 1975. 189 С.
10. Клини С. Математическая логика. М.: Мир, 1973. 480 С.
11. Новосельцев В.Б. Теория структурных функциональных моделей // Сибирский математический журнал. 2006. Т. 47. - № 5. С. 1014 1030.
12. Косарев, Н.И. Продукционная модель представления знаний в системах поддержки принятия решения // Вестник Сибирского юридического института ФСКН России №2 (13), 2013.
13. Дорохина Г. В. Требования к информационной технологии цифрового сбора, обработки и анализа данных. Проблемы искусственного интеллекта. 2020. № 4 (19). С. 4–9.
14. Bruno Borlini Duarte An Ontology-based Reference Model for the Software Systems Domain with a focus on Requirements Traceability/ Bruno Borlini Duarte. Vitória, ES, 2022- 149 p.: il.; 30 cm. (дата обращения: 04.12.2025).
15. Гаркуша, Д. А. Анализ онтологических платформ. Искусственный интеллект: теоретические аспекты, практическое применение: материалы Донецкого международного научного круглого стола. Донецк: ФГБНУ «ИПИИ», 2023. 252 с. С. 37–40.
16. Загорулько Ю.А. Современные средства формализации семантики областей знаний на основе онтологий. Информационные и математические технологии в науке и управлении. 2018. № 3 (11). С. 27 36. DOI:10.25729/2413-0133-2018-3-03.
17. Половикова О.Н. Особенности программной реализации логических задач на языке Prolog / О.Н. Половикова, В.В. Ширяев, Н.М. Оскорбин, Л.Л. Смолякова // Известия АлтГУ. Математика и механика, 2021, №1 (117).
18. Лорьер Ж.Л. Системы искусственного интеллекта. URL: http://lib.alnam.ru/book_sii.php (дата обращения: 04.12.2025).
19. Математическая логика и логическое программирование // Математический форум MathHelpPlanet. URL: http://mathhelpplanet.com/static.php?p=matematicheskaya-logika-i-logicheskoye-programmirovaniye. (дата обращения: 04.12.2025).
20. Решение логических задач на Prolog // Блог программиста: программирование и алгоритмы (версия от 28.05.2018). URL: https://pro-prof.com/archives/1299. (дата обращения: 04.12.2025).
21. Santos Costa V. On Just in Time Indexing of Dynamic Predicates in Prolog // Lecture Notes in Computer Science. Berlin, Heidelberg, 2009. DOI:10.1007/978-3-642-04686-5.
22. Повторение и рекурсия. Откат // Проект: project:prolog:povtorenie_i_rekursija. URL: http://verim.org/project/prolog/povtorenie_i_rekursija. (дата обращения: 04.12.2025).

References

1. Khabarov S.P. Intelligent information systems. PROLOG is a language for developing intelligent and expert systems: a textbook / S.P. Khabarov. - St. Petersburg State Technical University, 2013. – 138P.
2. Tsukanova, N.I. The ontological model of knowledge representation and organization: a textbook for universities / N.I. Tsukanova, Moscow: Hotline – Telecom, 2015. – 272P.
3. Garkusha, D.A. Functional features of implemented ontological platforms // D.A. Garkusha // Problems of artificial intelligence. - 2023. № 4 (31). - 4-11. - http://search.rads-doi.org/project/14374/object/210537 doi: 10.34757/2413-7383.2023.31.4.001.
4. Novoseltsev V.B. Effective nonresolutionary derivation for the limited calculus of Horn disjuncts // Izvestiya Tomsk Polytechnic University, 2008. – T. 312 №5. – Pp. 94 – 97.
5. Borgest N.M., Orlova A.A. Ontological editor Fluent Editor: an educational and methodological guide to laboratory work / comp.: N.M. Borgest, A.A. Orlova. Samara: Publishing House of Samara University, 2017. – 44P.
6. Mendes P. N. DBpedia: a multilingual cross-domain knowledge base [Текст] / P. N. Mendes, M. Jakob, and C. Bizer // LREC. – 2012. – Pp. 1813–1817.
7. Tsukanova, N.I. The ontological model of knowledge representation and organization: a textbook for universities / N.I. Tsukanova, Moscow: Hotline – Telecom, 2015. – 272P.
8. Adamenko A.N., Kuchukov A.M. Logical programming and Visual Prolog. - St. Petersburg: BHV-Petersburg, 2003. – 992P. ISBN 5-94157-156-9
9. Wirth N. Structural programming. Moscow: Mir, 1975. – 189P.
10. Klini S. Mathematical Logic, Moscow: Mir, 1973. – 480P.
11. Novoseltsev V.B. Theory of structural functional models // Siberian Mathematical Journal. – 2006. – T. 47. - № 5. – Pp. 1014 – 1030.
12. Kosarev, N.I. The production model of knowledge representation in decision support systems / N.I. Kosarev // Bulletin of the Siberian Law Institute of the Federal Drug Control Service of Russia №2 (13), 2013.
13. Dorokhina G. V. Information technology requirements for digital data collection, processing and analysis. Problems of artificial intelligence. 2020. № 4 (19). Pp. 4–9.
14. Bruno Borlini Duarte an Ontology-based Reference Model for the Software Systems Domain with a focus on Requirements Traceability/ Bruno Borlini Duarte. – Vitória, ES, 2022- 149 p.: il.; 30 cm. (date of request: 04.12.2025).
15. Garkusha, D. A. Analysis of ontological platforms. Artificial intelligence: theoretical aspects, practical application: proceedings of the Donetsk International Scientific Round Table. Donetsk: FGBNU "IPII", 2023. 252P. Pp. 37–40.
16. Zagorulko Yu.A. Modern means of formalizing the semantics of knowledge domains based on ontologies. Information and mathematical technologies in science and management. 2018. № 3 (11). C. 27 36. DOI:10.25729/2413-0133-2018-3-03.
17. Polovikova O.N. Features of software implementation of logical tasks in the Prolog language / O.N. Polovikova, V.V. Shiryaev, N.M. Offending, L.L. Smolyakova // News of AltSU. Mathematics and Mechanics, 2021, №1 (117).
18. Laurier J.L. Artificial intelligence systems. URL: http://lib.alnam.ru/book_sii.php (date of request: 04.12.2025).
19. Mathematical logic and logical programming // Mathematical Forum MathHelpPlanet. URL: http://mathhelpplanet.com/static.php?p=matematicheskaya-logika-i-logicheskoye-programmirovaniye. (date of request: 04.12.2025).
20. Solving logical problems in Prolog // Programmer's blog: programming and algorithms (version from 28.05.2018). URL: https://pro-prof.com/archives/1299. (Date of request: 04.12.2025).
21. Santos Costa V. On Just in Time Indexing of Dynamic Predicates in Prolog // Lecture Notes in Computer Science. Berlin, Heidelberg, 2009. DOI:10.1007/978-3-642-04686-5.
22. Repetition and recursion. Rollback // Project: project:prolog:povtorenie_i_rekursija. URL: http://verim.org/project/prolog/povtorenie_i_rekursija. (date of request: 04.12.2025).

RESUME

D.A. Filipishin, S.A. Zori

The resolution method as a means of extending the ontology inference machine

At the moment, the existing description of domain ontology construction approaches does not have the tools to solve production problems using the reverse inference method. Each task, the purpose of which is to create and implement an ontology in information systems, software complexes and other software products, requires analysis and identification of the purpose of building an ontology, as well as development or involvement of third-party tools using the ontology as a knowledge base.

To work effectively within the framework of the ethnographic approach (end-user orientation), it is necessary to use the developed ontology comprehensively and as limited as possible from third-party software products. The work involves a number of modifications of the ontological platform for solving programming problems.

РЕЗЮМЕ

Д.А. Филипишин, С.А. Зори

Метод резолюции как средство расширения машины вывода онтологий

На данный момент существующее описание подходов построения онтологий предметных областей не имеет инструментов решения продукционных задач методом обратного вывода. Каждая задача целью которой является создание и внедрение онтологии в информационные системы, программные комплексы и другие программные продукты, вызывает необходимость проводить анализ и выделять цель построения онтологии, а также разрабатывать или привлекать сторонние средства, использующие онтологию как базу знаний.

Для эффективной работы в рамках этнографического подхода (ориентирования на конечного пользователя) необходимо использовать разработанную онтологию комплексно и максимально ограничено от сторонних программных продуктов. Работа предполагает ряд модификаций онтологической платформы для решения задач программирования, в том числе ряд заимствований возможностей у инструментов языка логического программирования Пролог.

Филипишин Дмитрий Александрович – ассистент кафедры программной инженерии им. Л.П. Фельдмана ФГБОУ ВО «Донецкий национальный технический университет», научный сотрудник молодежной научной лаборатории «Искусственный интеллект». *Область научных интересов:* онтологии, системы искусственного интеллекта; эл. почта domaco@mail.ru, адрес: 283001, г. Донецк, ул. Артема, д. 58, телефон +7 949 334 91 49.

Зори Сергей Анатольевич – доктор технических наук, доцент, заведующий кафедрой программной инженерии им. Л.П. Фельдмана ФГБОУ ВО «Донецкий национальный технический университет». *Область научных интересов:* системы искусственного интеллекта; системный анализ; эл. почта ik.ivt.rec@mail.ru, адрес: 283001, г. Донецк, ул. Артема, д. 58, телефон +7 949 315 43 43.

Статья поступила в редакцию 11.10.2025