# ОСОБЕННОСТИ ПОСТРОЕНИЯ ВЫЧИСЛИТЕЛЬНОГО МЕХАНИЗМА ЯЗЫКОВ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

**Сергей ПЕЛИН,** Университет прикладных знаний Молдовы (UŞAM),
**Николай ПЕЛИН,** Тираспольский Государственный Университет (UST),

**Rezumat.** În articol sunt analizate elementele logicii, care sunt puse la baza unui deductor automat de interpretare a programelor computaționale scrisă în logică, adica deductor cum în realitate şi este interpretorului limbajului de programare Prolog. Sunt descrisă pas cu pas: restricțiile față de disjuncții a formei conjuctiv normale cărei poate fi aplicat efectiv principiu de rezoluție; rezoluția şi unificația gestionate în conformitate cu o strategie fixă de cautare a soluției; condiția stabilită apriori pentru alegerea disjuncților din care să retrage rezolventa în prim pas şi în pașii ce urmează în continuare pînă la interpretarea deplină a formulei logice (program scris în Prolog) şi obținerii unui rezultat în formă acceptabilă pentru utizator. Se face analiza unor funcţii a produsului SPprolog – sistem pentru proiectări în Prolog cu suport inteligent pentru instruire şi consultare ce ţine de lucrul cu sistemul şi limbajul de programare în logică.

**Abstract.** In this article are analyzed logic theory elements, which formed the basis of logic programming interpreter, i.e. interpreter of Prolog logic programming language. Step by step are described: requirements to the disjuncts of conjunctive normal form, where a resolution principle can be effectively applied; resolution and unification controlled in accordance with a specific decisions search strategy; established a priori condition of disjuncts selection for resolvents extraction, on the first and succeeding steps the resolution process, up to the full interpretation of logical formula (program written in Prolog language) and obtaining a result in the form acceptable for the user. Are described some core functions of SPprolog – a developing environment that represents means for developing applications in Prolog supported by the intellectual training system which provides training for working with the system and logic programming language.

**Резюме.** В статье проводится анализ элементов логики, которые легли в основу интерпретатора логических программ, т.е. интерпретатора языка логического программирования Пролог. Шаг за шагом описаны: требования предъявляемые к дизъюнктам конъюктивно-нормальной формы к которой эффективно можно применить принцип резолюции; резолюция и унификация управляемые в соответствии с конкретной стратегией поиска решений; априорно установленным условием для выбора дизъюнктов для извлечения резольвент на первом и последующих шагах резолютивного процесса вплоть до полной интерпретации логической формулы (программы написанной на языке Пролог) и получения некоторого результата в приемлемой для пользователя форме.

**Ключевые слова:** интерпретатор, резолюция, дизъюнкты Хорна, язык Пролог

## Элементы теории

**Вводные замечания.** Известно [1-7], что логическая программа, представляющая собой совокупность дизъюнктов Хорна, описывающая ситуацию реального мира, может быть проинтерпретирована с помощью логического интерпретатора, обеспечивающего работу соответствующего механизма вывода. Элементы теории, на основе которых строятся такого рода механизмы, рассмотрим ниже.

В статье [9], опубликованной выше в этом журнале, как и в [14], мы довольно много внимания уделили теории и практике логического программирования. Мы ввели соответствующие представления о конъюктивно-нормальной форме (КНФ). Проверить КНФ на выполнимость (т.е., когда формула, по крайней мере, один раз может быть интерпретирована со значением *ИСТИНА*) можно, но это не всегда эффективное занятие. Все же есть удобный метод для выявления невыполнимости множества дизъюнктов. Мы знаем, что множество дизъюнктов невыполнимо тогда и только тогда, когда пустой дизъюнкт *Л* является логическим следствием из него. Невыполнимость множества *S* можно проверить, порождая логические следствия из *S* до тех пор, пока не получим пустой дизъюнкт. Этот механизм эффективно реализуется с помощью, так называемого, правила резолюции, одного из наиважнейших аспектов механизма вывода в логическом программировании.

**Правило резолюций.** Рассмотрим следующую схему рассуждений. Имеем формулы

$$(p \vee q) \; и \; (\neg q \vee r).$$

Допустим, что они истинны. Тогда, если $q$ истинно, то $r$ тоже истинно. Если $q$ ложно, то $p$ – истинно. При любом значении $q$ формула $(p \vee r)$ – истинна. Представим изложенное правило в виде тавтологии, обозначаемой |=, а приведенные выше формулы в теоретико-множественном представлении. Тогда:

$$\{ p \vee q , \neg q \vee r \} \models p \vee r .$$

В случае, когда $q$ высказывание (атомарная формула), а $p$ и $r$ дизъюнкты, это правило называется правилом резолюции. Выражение $p \vee r$ называют резольвентой формул $(p \vee q)$ и $(\neg q \vee r)$. Способы доказательства невыполнимости формул на основе метода резолюций дают возможность использовать средства автоматического доказательства, применяемые в логическом программировании.

**Свойство завершаемости метода резолюций.** Конечное множество $S$ невыполнимо тогда и только тогда, когда пустой дизъюнкт может быть выведен из $S$ с помощью резолюций. Пустой дизъюнкт невыполним. Он не может быть логическим следствием из выполнимого множества дизъюнктов. Если множество дизъюнктов $S$ невыполнимо и содержит резольвенты своих элементов, то оно обязательно содержит пустой дизъюнкт. Факт невыполнимости формулы всегда можно установить методом резолюций.

Нет достаточно эффективного критерия для проверки выполнимости КНФ. Однако возможность констатации того, что множество дизъюнктов $S$ невыполнимо тогда и только тогда, когда пустой дизъюнкт *Л* является логическим следствием из этого множества, все же дает возможность проверки невыполнимости $S$, порождая

логические следствия из ***S*** до тех пор, пока не будет получен пустой дизъюнкт. Так мы пришли к методу резолюций.

Далеко не во всех случаях этот метод эффективен. Однако есть специальное множество дизъюнктов, для которых применение резолюции весьма эффективно. В общем случае, это когда в каждом из дизъюнктов множества содержится не более одной литеры одного знака и этот знак сохраняется одним и тем же для всех дизъюнктов множества, а все остальные литеры в каждом из дизъюнктов имеют противоположный знак. Такой дизъюнкт назван хорновским.

**Алгоритм проверки выполнимости множества хорновских дизъюнктов.** Если множество хорновских дизъюнктов ***S*** не содержит тавтологий, то алгоритм проверки аналогичен алгоритму проверки для КНФ. Если ***Л*** $\notin$ ***S*** и ***p*** – унитарный позитивный дизъюнкт из ***S***, а ***c*** – дизъюнкт из ***S,*** содержащий $\neg p$, то вычисляем резольвенту ***r*** и заменяем ***S*** на $(S \setminus \{c\}) \cup \{r\}$.

Но отличие все же есть. Здесь на каждом этапе некая литера удаляется из одного дизъюнкта [3, с. 46-47]. Дизъюнкт ***r*** – не что иное, как дизъюнкт ***c***, из которого удалена литера $\neg p$. Следовательно, выполнение алгоритма завершается всегда, независимо от принятой стратегии при выборе ***p*** и ***c***. Если ***N*** – число литер, первоначально присутствующих в ***S*** (с учетом повторений), то цикл будет выполняться не более ***N*** раз.

Алгоритм может закончиться либо порождением пустого дизъюнкта, либо получением множества ***S***, уже не содержащее дизъюнктов, равных ***p*** и ***c***. Первый исход означает невыполнимость множества ***S***, второй - множество ***S*** обязательно выполнимо. Любое конечное выполнимое множество хорновских дизъюнктов допускает одну и только одну минимальную модель [3, с. 48-49]. Как известно моделью формулы называют случай, когда эта формула может быть интерпретирована со значением ***Истина***. Нахождение минимальной модели означает, что истинными считаются только явно сформулированные факты (высказывания) и логические следствия, получаемые из них согласно правилам. Это положение соответствует гипотезе «замкнутого мира», широко используемой в области баз данных.

**Правило резолюции в логике предикатов.** Правило резолюции в логике предикатов работает следующим образом [1, с. 55-56]. Две фразы могут быть резольвированы друг с другом, если одна из них содержит позитивную литеру, а другая – соответствующую негативную литеру, с одним и тем же обозначением предиката (предикатной константой) и одинаковым количеством аргументов (термов), и, если аргументы у обеих литер могут быть согласованы (унифицированы) друг с другом.

Рассмотрим две фразы:

$$P(a) \vee \neg Q(b,c)$$
$$Q(b,c) \vee R(b,c)$$

Здесь в первой фразе содержится негативная литера $\neg Q(b,c)$, а во второй фразе позитивная литера $Q(b,c)$, аргументы обоих литералов могут быть унифицированы, т.е. $b$ унифицируется с $b$, а $c$ унифицируется с $c$. Следовательно, эти фразы могут быть резольвированы друг с другом. В результате получается третья фраза: $P(a) \vee R(b,c)$.

Эта фраза называется резольвентой и включается в множество фраз, в которое до этого момента входили только первые две фразы. При выполнении последующих резолюций можно воспользоваться любой из фраз этих фраз.

**Унификация.** Суть унификации в следующем. Два атома унифицируемы, если:

- они построены из одной предикатной константы, примененной к попарно унифицируемым термам (здесь достаточно рассмотреть унификацию двух термов);
- один из них – переменная, в этом случае получается мгновенная унификация;
- два терма построены из одной и той же функциональной константы, примененной к попарно унифицируемым термам.

Получается (рекурсивный) алгоритм унификации – очень простой и линейной сложности относительно числа термов.

Если один из термов переменная $x$, а другой терм содержит $x$, но не сводится к $x$, то в этом конкретном случае унификация невозможна. Систематическая проверка не вхождения некоторой переменной в терм, подлежащий унификации с этой переменной, приводит к неэффективному алгоритму. Все же средство для решения проблемы можно найти. В логическом программировании проверку не вхождения часто опускают.

Используя унификацию, можно распространить алгоритм резолюций на исчисление предикатов.

Унификация переменной с константой. Если в одной из фраз в качестве аргумента выступает переменная, то она унифицируема с соответствующей константой другой фразы, а резольвента будет содержать эту константу на том месте, где рассматриваемая переменная располагалась в исходной фразе. Например, фразы:

$$P(a,b) \vee \neg Q(b,c)$$
$$Q(x,y) \vee R(x,y)$$

резольвируемы, т.к. предикатная константа $Q$ связана с одинаковым числом аргументов в обеих фразах. При этом переменная $x$ унифицируется с константой $b$, а переменная $y$ унифицируется с константой $c$. В полученной резольвенте

$$P(a,b) \vee R(b,c)$$

переменные, служившие аргументами для $R$, заменены на константы.

**Принцип логического программирования.**

**Концепция**. Если использовать подходящую стратегию выбора, то метод резолюций становится хорошим средством для проверки выполнимости множества хорновских дизъюнктов в логике предикатов.

Принцип логического программирования заключается в том, что алгоритмические свойства некоторой функции можно представить множеством дизъюнктов и использовать метод резолюций для вычисления значений этой функции.

Автоматический решатель задач, базирующийся на принципе резолюций, может выполнить алгоритм, описанный набором хорновских дизъюнктов.

В языке программирования Пролог принцип резолюции - основа его работы.

**Нисходящая стратегии решения задач при использовании правила резолюции.**

При использовании правила резолюции выбираются разные стратегии решения задач. Рассмотрим одну из них, называемую нисходящей или обратной стратегией [1,с.57-59]. В этой стратегии ставится цель обнаружить, является ли единственная фраза ***P*** следствием существующего множества фраз ***Q***. Название нисходящей эта стратегия получила из-за того, что процесс решения задачи (первая резолюция) начинается с отрицания заключения (дизъюнкта цели) и затем это отрицание унифицируется (сопоставляется) с фактом или правилом (унитарным или точным хорновским дизъюнктом). Полученная резольвента должна участвовать в следующей резолюции и так продолжается до тех пор, пока не будет выведен пустой дизъюнкт. Такую стратегию называют «поиском в глубину» из-за того, что результат последней резолюции всегда используется в следующей за ней резолюции.

Пример резолюции нисходящим методом. Пусть имеется множество фраз (дизъюнктов) [1, с. 58-59]:

***1. p (a)∨ ¬q (a, b).***

***2. q (x, y)∨ ¬r (x, y).***

***3. s (b).***

***4. r (a, b).***

Требуется выяснить, является ли фраза ***p(a)*** следствием существующего множества фраз. Для начала добавим к первым четырем фразам отрицание фразы ***p(a),*** которая в контектно-свободной грамматике записывается в следующем виде:

***5. ¬p(a).***

Далее выполняем первую резолюцию с обязательным использованием только что добавленной фразы под номером ***5.*** Унифицируется первая же фраза. Получаем резольвенту:

***6. ¬ q (a, b).***

Далее унифицируем резольвенту *6* с фразой под номером *2.*

Получаем следующую резольвенту:

***7. ¬ r (a, b).***

Затем 7 с 4 (фраза 3 не унифицируется). Получаем пустую фразу, а это означает, что обнаружено противоречие. Так как добавленная к множеству фраз фраза ***¬p(a)*** приводит к противоречию, то ***p(a)*** является следствием множества фраз.

Об эффективности стратегий решения. Принцип резолюции представляет большой интерес так как непосредственно находится во взаимосвязи с вопросами "автоматического доказательства". Однако в поиске эффективной стратегии решения для всех областей приложения приводит к выводу: для заданного набора фраз доказательство на компьютере получается либо очень быстро, либо оно не получается вовсе [8, с. 173-174]. Эффективность программ падает с увеличением числа фраз. Проблема заключается в том, чтобы определить тот момент, когда надо остановить программу, потому что при этом возникают две опасности:

1. лавинообразное увеличение числа предложений;
2. риск преждевременного прекращения работы перед самым моментом получения решения.

## Интерпретация логических программ

**Абстрактный интерпретатор логических программ.** Абстрактный интерпретатор, описанный в [7, с. 21-23], выполняет вычисления с ответами ***Истина/Ложь***. Он получает программу ***P*** и основной вопрос *Q* и дает ответ ***Истина,*** если ***Q*** выводимо из ***P*** и ответ ***Ложь*** в противном случае. Если цель не выводима, то интерпретатор вообще не завершает работу и не выдает никакого ответа.

Демонстрация работы интерпретатора:

**Вход:** ***Основной вопрос Q и программа P.***

**Результат:** ***Истина, если найден вывод Q из программы P,***
***Ложь в противном случае.***

**Алгоритм:** ***Положить резольвенту равной Q.***
***Если Q не пуста то она резольвируется с***
***элементом из P.***
***Полученная резольвента $Q_1$ если не пуста,***
***то она резольвируется с следующим***
***элементом из P до их полного перебора.***
***Если резольвента $Q_k$-пуста, то результат –И,***
***иначе результат - Л.***

**Механизмы управления, заложенные в интерпретатор логических программ.** Руководство исполнением логических программ осуществляется механизмами управления, заложенного в интерпретатор. Программист же может оказывать влияние на общий ход исполнения программы, устанавливая, например, последовательность вычислений.

Логические программы недетерминированы, т.е. в них отсутствуют факты, точно определяющие ход исполнения программы. Это обстоятельство вынуждает интерпретатор осуществлять поиск для того, чтобы не пропустить ни одного решения. Нужна стандартная стратегия поиска (дерево поиска), которое интерпретатор строит для того, чтобы, имея программу, произвести конкретные вычисления. Дерево поиска просматривается по методу «поиска в глубину» [2, с.95]. Механизм поиска представлен на рис.4.

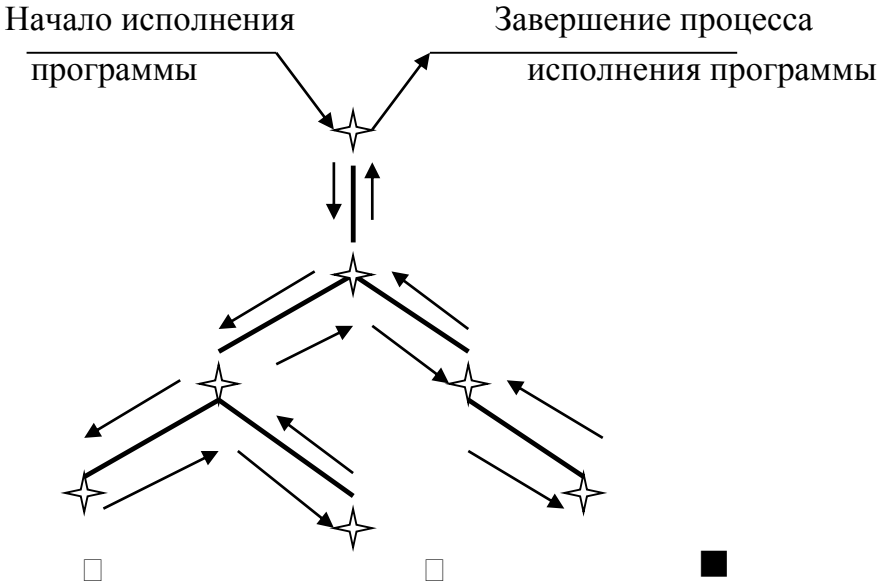


Рис. 4. Поиск в глубину.

Интерпретатор строит вычисления, начиная с корня дерева вычислений, затем продвигается вниз, по выходящей из корня ветви до тех пор, пока не достигает очередной конечной вершины (до получения пустого дизъюнкта □) или последней вершины ■. Если достигается некоторая вершина □, то интерпретатор может извлечь ответ и сообщить о нем устройству вывода. Об интерпретаторах, управляемых такой стандартной стратегией, иногда говорят, что они осуществляют поиск «слева направо в глубину с возвратом».

Оператор отсечения дополняет стандартную стратегию для удаления нежелательных вычислений, и отчасти и поэтому интерпретаторы не могут использовать в полной мере резолюцию [4, с. 73] .

В некоторых интерпретаторах имеются средства для «контроля зацикливания», которые пытаются распознать бесконечные вычисления.

Исполнение программ логическим интерпретатором. Интерпретатор представляет собой программу, способную строить резолютивные выводы, как правило, методом «сверху вниз, слева направо». Логическая программа начинает исполняться после ее подачи на вход логическому интерпретатору [4, с.55]. Получив входную логическую программу, интерпретатор предпринимает обычные шаги, необходимые для исполнения программы в режиме интерпретации:

1. Осуществляет синтаксический контроль входных утверждений;
2. Хранит их в центральной памяти в соответствующей упрощенной, компактной и доступной форме;
3. Переводит входное целевое утверждение во внутреннее представление и затем начинает процесс построения вывода путем последовательного применения правила резолюции к текущему целевому утверждению (вопросу) и некоторой родительской процедуре (к факту или правилу), выбираемой из хранящейся в памяти версии входной программы.

Если интерпретатору удается вывести пустое отрицание □, означающее, что решение получено, то он выдает какое-либо сообщение об этом вместе с найденными значениями целевых переменных.

**Исполнение программ в режиме интерпретации.** Исполнение программ в режиме интерпретации инициируются и управляются интерпретатором [4, с. 227]. Программа – интерпретатор осуществляет доступ к двум важным областям данных, расположенных в памяти компьютера:

1. статическая (или входной массив данных) область данных, в которой содержится закодированная версия входной логической программы, не подвергается изменениям в течение всего периода исполнения;

2. Динамическая (или стек исполнения) область, которая используется интерпретатором для ведения протокола собственных действий.

В стеке представлены состояние управления исполнением программы интерпретатором и состояние данных ( каким переменным, какие значения присвоены в текущий момент времени).

**Предположение о замкнутости мира.** Предположение о замкнутости мира заложено в механизм интерпретации входной программы, написанной на Прологе [1, с.70-71]. Пролог действует так, как будто бы множество фраз текущей программы является единственным источником знаний. Если запрос терпит

неудачу, то это означает, что интерпретатор не смог выполнить его доказательство по множеству фраз, входящих в текущую программу.

Интерпретатор Пролога исходит из предположения, что если соблюдение некоторого конкретного случая отношения доказать нельзя, то это случай отношения является ложным. Здесь не делается отличие между неизвестным отношением и отношением, неистинность которого доказуема.

**«Чистый Пролог».** Программа на «чистом Прологе» – это логическая программа, в которой задан порядок фраз и целевых утверждений. Абстрактный интерпретатор построен так, чтобы использовать информацию о заданных порядках [7, с.80-81]. Логическая программа, реализованная в виде фраз (дизъюнктов) Хорна с использованием заданного алфавита и синтаксиса совместно с интерпретатором этой программы описанным выше вполне можно считать формальной системой в смысле [6, с. 81].

Логические программы, исполняемые с помощью вычислительной модели Пролога, называются программами на «чистом Прологе». «Чистый Пролог» представляет собой приближенную реализацию вычислительной модели логического программирования на последовательной машине.

## Методологические аспекты изучения логического программирования.

**Некоторые наблюдения.** Как видим из вышеизложенного логическое программирование как теория достаточно проста. Однако, как показывает практика преподавания логического программирования в ряде университетов Республики Молдова и Румынии, а также преподавание его на соответствующих курсах вне университетской программы [13], осмысление учебного материала приходит медленно, приобретаемые навыки для практического программирования на языке Пролог не достаточно высоки. По нашему убеждению связано это, прежде всего, с неправильным подходом к преподаванию курса информатики еще в школе. Как отмечается в [5], лучший результат получается при изучении вначале основ и техники логического (декларативного) программирования и уж затем переходить к изучению алгоритмического (процедурного) программирования. Однако исторически сложилось иная ситуация и сейчас приходится искать эффективные методы преподавания и варианты технических решений, которые способствовали уменьшению «алгоритмического эффекта» в мышлении обучаемого, повысило бы результативность для начинающих, предоставили бы новые возможности как для традиционного, так и для дистанционного обучения через Интернет, а также для самообучения.

Нами разработаны ряд методов и средств, которые призваны улучшить методику преподавания прежде всего курсов связанных с теорией и практикой логического

программирования. Это прежде всего система SPprolog, более подробно описанная одним из авторов в [9], метод структуризации знаний с помощью так называемой «матрицы элементов знаний» [11, 12] и так называемая система UŞAM SOFT «QUEST» предназначенная для самообучения и обучения, автоконтроля и контроля знаний. Из-за наличия ограничений по объему статьи, дадим лишь некоторые представления о системе SPprolog.

**SPprolog** - это комплексная система, позволяющая разрабатывать программные приложения на языке логического программирования Пролог, в сопровождении интеллектуальной обучающей системы, обеспечивающей обучение, как работе с системой так и, языку логического программирования.

Под интеллектуальной обучающей системой подразумевается комплекс средств и мер который включает:

- интеллектуальный, мультимедийно-интерактивный интерпретатор языка логического программирования, который позволяет доступно и наглядно провести детальную визуальную интерпретацию программы, графически выделить соответствующие фрагменты программы, обеспечивающих реализацию конкретных процедур и механизмов (бэктрэкинг, рекурсивные правила, арифметические, логические и иные операции языка логического программирования), снадбить выделенные фрагменты необходимыми текстуальными разъяснениями по сути протекаемых в них процессов, применяя аудиовизуальные эффекты для повышения уровня восприятия материала;
- автоматизированный режим «Совет Эксперта», основанный на экспертных знаниях о «культурном», эффективном и оптимизированном программировании, а также знаниях о языке и теории, на которых основывается этот язык логического программирования, об основных проблемах, возникающих при изучении языка. Данный режим по ходу написания программы «подскажет» программисту о появляющихся ошибках или о вариантах написания более эффективного кода; «предложит» список доступных предикатов, которые можно применить в соответствующем месте программы; «предложит» типовой вариант решения некоторой задачи и «разъяснит» принцип работы программы, реализованной по этому варианту. В случае каких-либо затруднений режим «Совет Эксперта» идентифицирует соответствующую процедуру или механизм, по которым у программиста нет полной ясности и «предложит» для консультаций нужный участок в электронном учебном пособии.
- электронный курс языка программирования Пролог, представляемого в виде «матрицы элементов знаний» [12] и снабженный системой последовательного (запрограммированного преподавателем) воспроизведения знаний для самообучения и контроля.

- ускоренный курс обучения языку и системе SPprolog, основанный на запрограммированном, последовательном наборе действий (actions), позволяющий используя предыдущие два механизма системы наглядно и точно разъяснить основные принципы работы системы и языка пролог.

Также одной из основных особенностей системы SPprolog относится функционально расширенная (по сравнению с существующими системами) среду разработки программ на языке Пролог, с многофункциональным редактором кода, автоматизированной системой организации персональных элементов программирования (toolbox).

Система SPprolog может быть использована теми кто проявляет интерес к логическому программированию: при обучении и самообучении, при традиционном и дистанционном обучении, а также при повседневном занятии программированием как видом профессиональной деятельности. Она может использоваться в разных образовательных учреждениях (школах, вузах, на специализированных курсах) обучаемыми - как среда для разработки программ и наглядного понимания результатов составленной программы, как экспериментальная среда при изучении того или иного механизма языка, как техническое средство обучения - преподавателями в процессе объяснения различных механизмов и приемов или в процессе объяснения принципа работы некоторой программы.

Однако предполагается, что основной успех данной системы должен быть именно в процессе самообучения языку логического программирования, так как система сама «расскажет» и «объяснит» новую тему по заложенному плану (по запрограммированным действием); предложит необходимые примеры, подробно и наглядно объяснит принцип их работы, а в процессе самостоятельного составления программ заменит «опеку» преподавателя – ценными и своевременным советами из заложенной экспертной системы.

## Заключение

В статье дан обзор элементов теории, на основе которых построен механизм логического вывода заложенного в интерпретаторах языка логического программирования Пролог. На основе проведенного анализа литературы, результатов проведенных исследований высказывается мнение о необходимости пересмотра программы учебного курса по информатике в лицеях и высших учебных заведениях с целью введения соответствующего раздела, посвященного декларативному программированию и программированию в логике, в частности. Проводится описание основных характеристик SPprolog, разработанной одним из авторов [10] комплексной системы, позволяющей проектировать программные приложения на языке логического программирования Пролог, в сопровождении

интеллектуальной обучающей системы, обеспечивающей обучение и самообучение как работе с системой так и, языку логического программирования.

**Литература**


1. Малпас Дж. Реляционный язык Пролог и его применение: Пер. с англ. Под. Ред. В.Н. Соболева – М.: Наука. Гл. ред. Физ.- мат.лит., 1990-464 с.
2. Ковальски Р. Логика в решении проблем: Пер. с англ. – М.: Наука. Гл. ред. физ. мат. лит., 1990. – (Пробл. искусств. интеллекта) - 280 с.
3. Тей А., Грибомон П., Луи Ж. и др. Логический подход к искусственному интеллекту: от классической логики к логическому программированию. Пер. с. франц. Пермякова П.П. Под. ред. Гаврилова Г.П. – М., «Мир», 1990 – 432 с.
4. Хоггер К. Введение в логическое программирование: Пер. с англ. М: -Мир, 1998-348 с.
5. Колмероэ А., Кануи А., М. ван Канегем. Пролог- теоретические основы и современное развитие. В сб.: Логическое программирование. – М.: Мир, 1988, с.27-133.
6. Лорьер Ж.-Л. Системы искусственного интеллекта: Пер.с франц.- М.: Мир, 1991-568 с.
7. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог: М., Мир, 1987.
8. Алексеев М. Совершенствование методики построения образовательного ВЕБ-сайта. Автореферат диссертации на соискание ученой степени кандидата педагогических наук, Москва, 2001 г.
9. Пелин Н., Пелин С. Вопросы теории и практики логического программирования. Статья в этом же номере журнала.
10. Pelin S. Work optimisator and intellectual assistant for Prolog development environment. Proceedings of the International Conference "Knowledge management: Projects, Systems and Tehnologies". Volume I, november 9-10, 2006, Bucharest.The 2nd supliment of the review INFORMATICA ECONOMICA. Volume X, ISSN 1453-1305, pag. 185-190.
11. Pelin N., Pelin S., Pelin A. "The matrix of knowledge elements" effective tehnology for knowledge management. Proceedings of the International Conference "Knowledge management: Projects, Systems and Tehnologies". Volume I, nofember 9-10, 2006, Bucharest.The 2nd supliment of the review INFORMATICA ECONOMICA. Volume X, ISSN 1453-1305, pag. 63-71.
12. Pelin N., Pelin S., Pelin A. The problems of knowledge structure defining, activation an usage. The Proceedings of the EICHT International Conference on Informatics in

Economy. Informatics in Knowledge Society. Bucharest, may 17-18, 2007, ISBN 978-973-594-921-1, pag. 749-755.
13. Pelin N. Studierea programarii logice in Moldova. Anuale stiintifice ale doctoranzilor si competitorilor «Probleme actuale ale stiintelor umanistice». Universitatea Pedagogica de stat "Ion Creanga" Vol. I. Chisinau, 1996, pag. 69-73.
14. Pelin S., Pelin N. Deducția logică – mecanism de calcul al limbajului de programare logică. Studia universitas. Revistă științifică. Științe exacte. Nr. 8. CEP USM. Ch.: 2008.- pp. 110 -116.
15. www.prolog.md