DOI 10.37882/2223–2966.2022.10.23

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ РЕШАТЕЛЕЙ ДЛЯ ЛОКАЛЬНЫХ ГЕОМЕТРИЧЕСКИХ ЗАДАЧ[1]

LOGICAL PROGRAMMING OF SOLVERS FOR LOCAL GEOMETRIC PROBLEMS

P. Novikov

Summary. The article introduces the concept of a local geometric problem from the position of the clausal form of logic. The abilities of logical programming of local geometric problem are investigated. The features of programming symmetric binary relations and transitive geometric properties in the PROLOG language are shown. Domains types and predicates have been created to implement the basic concepts of geometry. The functioning of various solvers of local geometric problems is investigated.

Keywords: local geometric problem, logical program, binary relations, transitive geometric properties, problem solver.

Новиков Павел Владимирович
К.т.н, Московский авиационный институт (национальный исследовательский университет)
novikov.mai@mail.ru

Аннотация. В статье вводится понятие локальной геометрической задачи с позиции клаузальной формы логики. Исследуются возможности логического программирования решателей этих локальных геометрических задач. Показаны особенности программирования симметричных бинарных отношений и транзитивных геометрических свойств на ПРОЛОГе. Созданы доменные типы и предикаты для реализации основных понятий геометрии. Изучается работа различных решателей локальных геометрических задач.

Ключевые слова: локальная геометрическая задача, логическая программа, бинарные отношения, транзитивные геометрические свойства, решатель задачи.

Со второй половины 50-х годов XX века по настоящее время учёные приложили немало усилий для создания решателей задач, основанных на логическом выводе [1], от универсальных решателей задач (GPS) [2] до автоматических доказателей теорем (ATP) типа VAMPIRE [3]. Все такие системы отличаются построением на максимально общих теоретических основах, реализуя стремление охватить возможно более широкий круг задач с позиции фундаментальной аксиоматики. В противоположность такому методу в настоящей статье предлагается подход, основанный на средствах и широких возможностях логического программирования, предлагающий решать конкретные геометрические задачи на доказательство, исходя лишь из их конкретики и опираясь, по возможности, только на те условия, что указаны в задаче. Это, в первую очередь, задачи на применение одной теоремы (одной аксиомы, одной леммы, одного свойства, одного следствия из теоремы), то есть *локальные* геометрические задачи.

Пропедевтическая цель изучения геометрии состоит в подготовке аппарата, необходимо для изучения смежных дисциплин [4]. Развивающая цель геометрии есть, помимо формирования образного и пространственного мышления, формирование логического мышления [5]. Дисциплина **логика** в её первоначальном, классическом виде, то есть логика умозаключений, рассуждений и строгих доказательств, отсутствует в системе современного среднего образования, а в системе высшего образования явно присутствует только в программах обучения философов и юристов [6]. *Математическая логика*, входящая в школьную дисциплину *Информатика*, представлена, в основном, исчислением высказываний (*Булевой алгеброй)* и направлена на приложения в схемотехнике. Хотя сто с лишним лет назад и ранее логика была в учебных программах лицеев и гимназий (см. [7] и [8], Таблица 5), ныне доказательным рассуждениям учащиеся овладевают именно в курсе геометрии, сверхзадача которой и есть развитие логического мышления [9].

Студентам вузов при изучении логического программирования также весьма полезно разрабатывать решатели локальных геометрических задач.

Это позволяет по-новому взглянуть и на геометрические отношения, и на логические программы, оставаясь в рамках небольших задач.

[1] Статья поступила в редакцию 10.08.2022

Для решения названных *локальных* задач предлагается аппарат клауз (хорновских дизъюнктов) [10], на котором основан язык ПРОЛОГ [8, 11].

Задача — это требование или вопрос, на который надо найти ответ. Программы на языке ПОЛОГ иллюстрируют поговорку математиков «Правильная постановка задачи есть половина её решения», так как сама логическая программа есть полная и точная формулировка задачи на языке исчисления предикатов первого порядка в его клаузальной форме [10, 12]. Поиск решения логически сформулированной задачи (или поиск ответа на вопрос) в логической программе выполняется автоматически. Поисковый метод в языке ПРОЛОГ основан на алгоритме резолюции (Робинсон, 1965), встроенном в ПРОЛОГ-машину вместе с алгоритмом унификации [13].

Клауза есть утверждение, имеющее посылку и заключение, связанные импликацией if (или:-) [8, 10, 12]. Клауза записывается *справа-налево*:

Заключение <- посылка.

Заключение называют головой (заголовком) клаузы, а посылку называют телом клаузы: *голова* if *тело*. Или *голова*:- *тело*. Посылка и заключение клаузы (голова и тело) состоят из атомарных суждений, каждое из которых выражено предикатом. Тело клаузы есть конъюнкция (and) атомарных суждений (предикатов). Голова клаузы всегда представлена строго одним предикатом (клаузы Хорна). Клауза с головой и телом называется **правило**. Клауза с головой без тела называется **факт**. Факты и правила составляют логическую программу. Клауза с телом без головы есть **вопрос** к программе. Успешный поиск решения завершается **пустой клаузой** без головы и тела. Два правила с одинаковой головой могут быть объединены в одно правило, где два тела объединяются дизъюнкцией or. Можно менять конъюнкцию and на символ «запятая —, » и дизъюнкцию or на «точку с запятой —; » [10–11].

Алгоритм резолюции просматривает всю программу сверху вниз по головам клауз, начиная с предикатов, заданных в вопросе. Если найдена голова с таким же предикатом, то просмотр идёт по телу этой клаузы слева направо. Пройденный путь запоминается и поиск продолжается сначала, но по новому пути, пока не будет обследовано всё дерево решений.

Если алгоритм резолюции находит голову клаузы, совпадающую с искомым предикатом, он может «снять» этот предикат, но лишь тогда, когда аргументы искомого предиката и найденной головы клаузы унифицируемы.

Для разработки логических программ поставленную вербально задачу следует переформулировать на языке клаузальной формы логики [10]. На первый взгляд, переделка словесных утверждений в клаузы весьма проста.

Фраза «*Всякий квадрат является прямоугольником*» есть утверждение: «Для всех F: если F — квадрат, то F — прямоугольник», выраженное клаузой:

rectangle (F) if square (F).

Высказывание «*Всякий квадрат является ромбом*» есть утверждение:

«Для всех F: если F — квадрат, то F — ромб», представленное клаузой

rhomb (F) if square (F).

«*Всякая фигура, являющаяся и ромбом, и прямоугольником, есть квадрат*» иначе «Для всех F: если F — ромб и F — прямоугольник, то F — квадрат»:

square (F) if rectangle (F) and rhomb (F).

«*Всякая фигура, являющаяся ромбом, или являющаяся прямоугольником, является параллелограммом*» переделывается в: «Для всех F: если F — ромб или F — прямоугольник, то F — параллелограмм», выраженное клаузой:

parallelogram (F) if rhomb (F) or rectangle (F).

Аналогично фраза «Окружность есть частный случай эллипса» (когда фокусы совпадают) есть «Для всех F: если F — окружность, то F — эллипс»:

ellipse (F) if circle (F).

Достаточно добавить к этим правилам факты вроде:

circle (“C1”). rhomb (“SPQR”). rectangle (“EFGH”). square (“ABCD”).

После этого можно задавать программе любые вопросы из числа тех, на которые сможет ответить эта логическая программа (пример 1).

В примере 1 показана программа, состоящая из созданных клауз. Программа написана на языке Visual Prolog 5.2, для которого характерно предварительное объявление пользовательских типов в разделе **domains** и пользовательских предикатов в разделе **predicates**. Основная программа находится в разделе

```
domains s = string
database p(s)
predicates
 circle (s) ellipse (s)
 nondeterm rhomb (s)
 nondeterm rectangle (s)
 nondeterm square (s)
 nondeterm parallelogram (s)
clauses
 rectangle ("EFGH").  rectangle ("QRST").
 rectangle (F) if square (F).

 rhomb ("SPQR").  rhomb ("QRST").
 rhomb (F) if square (F).

 square ("ABCD").
 square ("MNOP")  :-  !.
 square (F) if rectangle (F) and rhomb (F).

 parallelogram (F) if rhomb (F)
                   or rectangle (F).

 ellipse (F) if circle (F).
 circle ("C1").

goal square ("QRST").
Yes
goal parallelogram (S), not(p(S)), assert(p(S)).
S=QRST
S=SPQR
S=ABCD
S=MNOP
S=EFGH
5 Solutions
```

Пример 1. «Геометрические фигуры».

clauses, а вопросы к логической программе можно задавать в разделе **goal**. Порядок разделов обязателен. Только раздел **goal** может располагаться как после раздела **clauses**, так и до. Это деление на строго заданные разделы в программе есть особенность ПРОЛОГов фирмы PDC и фирмы Borland. Клаузы с одинаковой головой должны быть собраны вместе для ускорения поиска в языках Visual Prolog и Turbo Prolog.

Если в программе несколько клауз с одинаковым предикатом в голове, то в разделе **predicates** такие предикаты объявляются как nondeterm.

Программа из примера 1 правильно отвечает, например, на вопрос «QRST — ромб?», заданный как **goal** rhomb ("QRST"). И, например, на вопрос «Какие фигуры — эллипсы?», заданный как **goal** ellipse (F). Однако же этот элементарный пример содержит тонкие места, важные не с позиции геометрии, а с точки зрения логической программы. Учёт таких тонких мест расширяет круг решаемых локальных задач. Для этого следует рассмотреть рекурсивные клаузы. Рекурсивные клаузы нужны, чтобы организовывать повторяющиеся вычисления. Клауза является *рекурсивной*, если предикат головы клаузы имеется также в теле клаузы. В тексте изучаемой программы формально рекурсив-

ных клауз нет. Однако есть скрытая рекурсия. Клауза square (F) if rectangle (F) and rhomb (F). в теле содержит предикаты rectangle и rhomb, а клаузы rectangle (F) if square (F). и rhomb (F) if square (F). в теле содержат предикат square. Алгоритм резолюции при переходе к клаузе с головой square снова придёт к предикату square. Также и при переходе к клаузам с головами rectangle и rhomb алгоритм резолюции вернётся в теле этих клауз к предикатам rectangle и rhomb. Это рекурсивные вызовы. Если не предпринять мер, то при вопросе к программе вроде square (X), когда требуется определить все квадраты, программа выдаст или *Stack overflow*, или **PROGRAM ERROR. 1010**, в зависимости от версии языка ПРОЛОГ. Эта ошибка означает переполнение стека из-за бесконечной рекурсии. Для преодоления бесконечной рекурсии необходимо сделать точку останова рекурсии. Для этого факт square ("MNOP"). следует превратить в правило square ("MNOP"):-!. Предикат «отсечение» (предикат cut —!, см. [8, 11, 13]) останавливает поиск алгоритма резолюции среди клауз с головой square, прерывает рекурсию. В результате вопрос **goal** square ("QRST"). не создаст бесконечной рекурсии, а получит ответ Yes от ПРОЛОГ-машины (*хотя в программе есть лишь факты rhomb("QRST"). и rectangle ("QRST").*). Это иллюстрация очень важного приёма логического программирования.

Другой очень важный приём логического программирования также представлен в примере 1. Ответ на вопрос **goal** parallelogram(S). Будет:

goal parallelogram(S).
S = QRST
S = SPQR
S = ABCD
S = MNOP
S = EFGH
S = QRST
S = ABCD
S = MNOP
8 Solutions

Налицо повтор в ответах, происходящий из-за того, что одни и те же ответы могут были получены алгоритмом резолюции разными путями. Так как алгоритм фиксирует только пути решения, но не фиксирует ответы на вопросы, то повторы в ответах в подобных задачах весьма вероятны. Это не всегда удобно, а в ряде задач приводит к ошибочным решениям. Например, когда следует точно подсчитать количество правильных ответов.

Чтобы устранить повторы в ответах следует использовать приём, который предложен в [8]. Нужно объявить динамический предикат p(s) в разделе **database** (см. пример 1). Эти динамические предикаты позволяют создавать факты и добавлять их в программу во время работы программы, а не до начала её работы с помощью встроенного предиката *второго* порядка assert(*Fact(…)*). При добавлении каждого нового факта алгоритм резолюции запоминает все текущие полученные ответы как часть пути поиска решения, что позволяет избежать повторов в ответах. Раздел **database** должен быть между разделом **domains** и разделом **predicates**, так как здесь тоже объявляются предикаты: **database** p(s). s — строковый тип string, значащий всё, что заключено в кавычки "". Вопрос к логической программе с ответами без повторов выглядит так: **goal** parallelogram(S), not(p(S)), assert(p(S)). Его можно вербально сформулировать так: «Цель истинна (успешна), если S — параллелограмм, и если ещё нет динамического факта p(S), то добавить факт p(S)». Предикат assert(*Fact(.)*) истинен (true) после добавления в программу динамического факта *Fact(…)*. Для удаления используется retract(*Fact(.)*).

Оба показанных приёма широко используются при решении многих других локальных геометрических задач. К этим приёмам следует добавить новые приёмы, необходимые при использовании более сложных ситуаций в геометрических задачах, когда недостаточно унарных предикатов.

Над основными понятиями геометрии (точка, прямая, отрезок, угол, луч и т.д.) устанавливают определённые отношения: *принадлежит, имеет, пересекаются, параллельны, перпендикулярны, совпадают, скрещиваются, равны, подобны,* и т.п. Говорят: «Точка принадлежит прямой», «Плоскости параллельны», «Прямые перпендикулярны», «Треугольник имеет прямой угол», «Отрезки равны», «Треугольники подобны», «Фигуры совпадают», «Прямые скрещиваются», «Прямая пересекается с плоскостью», и пр. и пр. Так как в этих примерах речь идёт об отношениях над двумя однородными аргументами (две плоскости, две прямые, два отрезка, два треугольника и т.п.) или над двумя разнородными аргументами (треугольник и угол, прямая и плоскость, точка и прямая и т.д.), то это *бинарные* отношения [8].

Бинарные отношения в исчислении предикатов 1-го порядка могут быть выражены предикатом от двух аргументов: p (A, B). Принимают, что p — предикат, A — субъект, B — объект [6–7]. Предикат mama (A, B), например, означает, что *субъект* A является матерью *объекта* B, а не наоборот. Так с позиции формальной логики отношение параллельности parallel(a, b) не то же, что parallel(b, a), хотя с позиции геометрии это одно и то же отношение.

Бинарное отношение R на множестве M может обладать различными свойствами [8], среди которых весьма

важно выделить симметричность, рефлексию и транзитивность, встречающиеся у геометрических объектов:

симметричность □a, b ∈ M aRb => bRa,
рефлексия □a ∈ M aRa,
транзитивность □a, b, c ∈ M aRb и bRc => aRc.

На практике свойством *симметричности* обладают геометрические отношения равенства, совпадения, подобия, параллельности, пересечения, скрещивания и перпендикулярности. Действительно, верны утверждения:

«Если фигура X равна фигуре Y,
то фигура Y равна фигуре X».
«Если фигура X совпадает с фигурой Y,
то фигура Y совпадает с фигурой X».
«Если фигура X подобна фигуре Y,
то фигура Y подобна фигуре X».

«Если прямая X параллельна прямой Y,
то прямая Y параллельна прямой X».
«Если прямая X пересекается с прямой Y,
то прямая Y пересекается прямой X».
«Если прямая X скрещивается с прямой Y,
то прямая Y скрещивается с прямой X».
«Если прямая X перпендикулярна прямой Y,
то прямая Y перпендикулярна прямой X».

«Если плоскость X параллельна плоскости Y,
то плоскость Y параллельна плоскости X».
«Если плоскость X пересекается с плоскостью Y,
то плоскость Y пересекается с плоскостью X».
«Если плоскость X перпендикулярна плоскости Y,
то плоскость Y перпендикулярна плоскости X».

Для двух произвольных несовпадающих точек A и B может быть введены понятия: «прямая (AB), проходящая через точки A и B» и «отрезок [AB], соединяющий эти точки». Тогда бинарные отношения «прямая» и «отрезок» также будут симметричны, ведь прямая (AB) есть то же самое, что и прямая (BA), а отрезок [AB] есть то же самое, что и отрезок [BA].

Отношения равенства, совпадения, подобия, параллельности, отношения «иметь» и «принадлежать» являются *рефлексивными*. Принято, что:

«Фигура равна самой себе.»
«Фигура совпадает сама с собой.»
«Фигура подобна самой себе.»
«Прямая параллельна самой себе.»
«Плоскость параллельна самой себе.»
«Фигура принадлежит самой себе.»
«Фигура имеет саму себя.»

Те же отношения равенства, совпадения, подобия, параллельности, «иметь» и «принадлежать» являются ещё и *транзитивными*. Верны утверждения:

«Если фигура A равна фигуре B,
а фигура B равна фигуре C,
то фигура A равна фигуре C.»

«Если фигура A совпадает с фигурой B,
а фигура B совпадает с фигурой C,
то фигура A совпадает с фигурой C.»

«Если фигура A подобна фигуре B,
а фигура B подобна фигуре C,
то фигура A подобна фигуре C.»

«Если прямая A параллельна прямой B,
а прямая B параллельна прямой C,
то прямая A параллельна прямой C.»

«Если плоскость A параллельна плоскости B,
а плоскость B параллельна плоскости C,
то плоскость A параллельна плоскости C.»

«Если фигура A принадлежит фигуре B,
а фигура B принадлежит фигуре C,
то фигура A принадлежит фигуре C.»

«Если фигура A имеет фигуру B,
а фигура B имеет фигуру C,
то фигура A имеет фигуру C.»

Термин «равно» здесь является синонимом термина «конгруэнтно», когда две фигуры равны не только численно (длина, площадь, объём и т.п.), а совпадают при наложении [14]. Близкие по смыслу отношения «равно» и «совпадает» различны с той точки зрения, что отношение «совпадает» не предполагает предварительного движения фигур для наложения (пример 2).

В примере 2 приведено решение локальной геометрической задачи об определении точки пере-

```
domains c=char s=symbol st=string
predicates intersect (s, st)
 nondeterm belong (c, s)
 nondeterm belong (c, st)
 nondeterm intersection (s, st, c)
clauses intersect (a, "Alpha").
 belong ('A', a). belong ('A', "Alpha").
 intersection(A, B, C):-intersect(A, B),
          belong (C, A), belong (C, B).
goal belong (Subj, Obj).
Subj=A Obj=a
Subj=A Obj=Alpha
goal intersection (X, Y, Z).
X=a Y=Alpha Z=A
```

Пример 2. Пересечение прямой и плоскости в точке: intersection(A, B, C)

```
domains s = symbol c = char
predicates
        nondeterm par (s, s)
        nondeterm belong (c, s)
        nondeterm math (s, s)
clauses
 belong ('A', b). belong ('A', c).
 par (b, a). par (c, a).
 par (a, b). par (a, c).
 math (B, C):-belong (D, B), belong (D, C),
    par(B, A), par(C, A), not(belong(D, A)).
goal math (b, c).
Yes
goal math (Subj, Obj).
Subj = b Obj = b
Subj = b Obj = c
Subj = c Obj = b
Subj = c Obj = c
4 Solutions
```

Пример 3. Локальная задача на аксиому параллельных.

```
domains s=symbol
database pd(s)
predicates
 nondeterm par (s)
 nondeterm par (s, s)
 nondeterm parallel0 (s, s)
 nondeterm parallel1 (s, s)
 nondeterm parallel2 (s, s)
 nondeterm parallel (s, s)
clauses
 par (a, b). par (b, c). par (c, d).
 par (A) :- par (A, _)
          ; par (_, A).
 parallel0 (A, A) :-
    par(A), not(pd(A)), assert(pd(A)).
 parallel1 (A, B) :- par (A, B), A<>B.
 parallel1 (A, B) :- par (A, C),
          parallel1 (C, B), A<>B.
 parallel2 (A, B) :- par (B, A), A<>B.
 parallel2 (A, B) :- par (B, C),
          parallel2 (A, C), A<>B.
 parallel (X, Y) :- parallel0(X, Y);
          parallel1(X, Y);
          parallel2(X, Y).
goal parallel (d, a).
Yes
```

```
goal parallel (X, Y).
X=a Y=b
X=b Y=c
X=c Y=d
X=a Y=c
X=a Y=d
X=b Y=d
X=a Y=a
X=b Y=b
X=c Y=c
X=d Y=d
X=b Y=a
X=c Y=b
X=d Y=c
X=c Y=a
X=d Y=a
X=d Y=b
16 Solutions
```

Пример 4. Свойство «параллельность».

```
domains c = char s = string
database d (s, s)
predicates nondeterm match (s, s)
           nondeterm triangle (s, c, c, c)
clauses triangle ("T1",'A', 'B', 'C').
        triangle ("T2",'A', 'B', 'C') :- !.
        triangle ("T1",'C', 'B', 'A').
        triangle ("T2",'C', 'B', 'A').
 triangle(N, C, A, B):-triangle (N, A, B, C).
 triangle(N, B, C, A):-triangle (N, A, B, C).
 match(N, M) :- triangle (N, A, B, C),
                triangle (M, A, B, C).
goal triangle ("T2", 'B', 'A', 'C').
Yes
goal match(N, M), N<>M, not(d(N, M)),
     not(d(M, N)), assert (d(M, N)).
N=T1 M=T2
1 Solution
```

Пример 5. Совпадающие треугольники.

сечения прямой и плоскости. При создании программы выбраны встроенные типы: **char** (*литерный*: точки 'A','B', …) **symbol** (*символьный*: прямые a, b, c, …), **string** (*строковый*: плоскости "Alpha", "Beta", "Gamma", …). Созданы предикаты: intersect (symbol, string) означает «пересекаются прямая и плоскость», intersection(symbol, string, char) означает «пересекаются прямая и плоскость в точке», belong (char, symbol) означает «принадлежит точка прямой», а предикат belong (char, string) означает «принадлежит точка плоскости». Единственное правило в этой программе отражает связь между геометрическими фигурами A, B и C: «Прямая A пересекает плоскость B в точке C, если и прямая A пересекается с плоскостью B, и точка C принадлежит прямой A, и точка C принадлежит плоскости B.»: intersection(A, B, C):- intersect(A, B), belong(C, A), belong(C, B).

Факты intersect (a, "Alpha"). belong ('A', a). belong ('A', "Alpha"). означают: «прямая a пересекается с плоскостью Alpha», «точка A принадлежит прямой a» и «точка a принадлежит плоскости Alpha», соответственно. Вопрос к программе «Что чему принадлежит?» **goal** belong (Subj, Obj). даёт верные ответы: «A принадлежит a» и «A принадлежит Alpha». А вопрос «Какая прямая с какой плоскостью и в какой точке пересекается?», заданный в виде **goal** intersection (X, Y, Z). Даёт верный ответ X = a, Y = Alpha, Z = C. Это означает «прямая a, плоскость Alpha, точка C».

В примере 3 запрограммирована локальная геометрическая задача на пятый постулат Евклида: «*Через точку, не лежащую на прямой, можно провести только одну прямую, параллельную данной прямой*», который можно переформулировать так: «*Если через точку, не лежащую на прямой, проходят две прямые, параллельные данной, то эти прямые совпадают*».

Выбраны тип s = symbol для прямых и тип c = char для точек. Предикат принадлежности точки прямой belong такой же, как в примере 2. Предикат math(s, s) означает совпадение (mathes) первого аргумента со вторым, а par(s, s) означает параллельность (parallel) первого аргумента второму.

В примере 3 решается задача: «Точка A принадлежит прямым b и c. Прямые b и c параллельны прямой a. Точка A не принадлежит прямой a. Верно ли, что прямые b и c совпадают?» Запрограммированное правило math(B, C) позволяет не только получить утвердительный ответ на вопрос, но и ответить на более общий вопрос «Какие прямые совпадают?». Согласно ответам на такой вопрос отношение math(B, C) симметрично и рефлексивно. Если нужно сделать это отношение нерефлексивным, следует к правилу math(B, C) добавить конъюнктивно A<>B.

Пример 4 посвящён определению параллельных прямых parallel(X, Y). Предикат par (s, s) нужен для соз-

дания фактов вроде par (a, b). Реализованы свойства рефлексии, транзитивности и симметрии правилами parallel0(X, Y), parallel1(X, Y) и parallel2(X, Y), соответственно. Факты о прямых par(A, B) не могут быть избыточны, чтобы не было бесконечной рекурсии. Правильно, если каждая прямая не более одного раза будет субъектом и/или объектом отношения par(A, B), а все параллельности прямых можно будет выразить орграфом [8]. Каждая дуга такого орграфа должна начинаться в субъекте и заканчиваться в объекте, а орграф должен состоять из цепочек без циклов.

Программа в примере 4 позволяет проверить наличие или отсутствие параллельности между двумя конкретными прямыми и определить все пары параллельных прямых. В ней реализованы логические свойства отношений, общие для различных геометрических объектов. Программа может быть легко переделана для исследования других геометрических отношений, обладающих такими же логическими свойствами. Чтобы программа исследовала параллельность плоскостей, достаточно изменить тип symbol на string для правильного именования плоскостей. Отношения равенства, совпадения и подобия также имеют свойства симметрии, рефлексии и транзитивности. Вместо par и parallel надо создать предикаты: math и matching (совпадение), equal и equality (равенство), similar и similarity (подобие), с аргументами соответствующих типов.

В примере 5 решена задача распознавания одинаковых треугольников.

У треугольников разный порядок перечисления вершин, либо разные имена.

Показанные решения локальных геометрических задач с помощью логического программирования важны для понимания геометрии с точки зрения исчисления предикатов и для создания программ на доказательство и поиск. Соединение локальных задач в глобальные — в новых публикациях.

ЛИТЕРАТУРА

1. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. — М.: Наука, 1983, 36 с.
2. Newell A., Shaw J.C., Simon H.A. "Report on a generic problem-solving program". Proc. Inter. Conf. on Information Processing, 1959, pp. 256–264.
3. Riazanov A., Voronkov A. The design and implementation VAMPIRE. — AI Communications, August 2002, p. 92–110.
4. Ситаров В.А. Дидактика. Уч. пос. для высшего пед. учеб. зав. / Под ред. В.А. Сластенина, — М.: «Академия», 2004, 368 с.
5. Щербакова М.А. «О реализации развивающей цели обучения математике студентов гуманитарных факультетов». — Матер. Всерос. НПК «Современные проблемы развития и методика преподавания естественных и точных наук». — Уссурийск, декабрь 2008 \ УГПИ, 2009, С. 127–131.
6. Кириллов В.И., Старченко А.А. Логика. — М.: Проспект, 2008, 240 с.
7. Царскосельский лицей. Наставники и питомцы. 1811–1817 / Сост. Д.А. Рутгайзер — г. Пушкин, издание Царскосельского Лицея, 1991.
8. Новиков П.В. Логическое программирование: Учебное пособие к лабораторным работам. — М.: Изд-во МАИ, 2007. — 100 с.
9. Далингер В.А. Методика обучения математике. Обучение учащихся доказательству теорем. — М.: Издательство Юрайт, 2019, 338 с.
10. Ковальский Р. Логика в решении проблем. — М.: Наука, 1990, 280 с.
11. Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. — СПб: БХВ-Петербург, 2003, 992 с.
12. Новиков Ф.А. Дискретная математика для программистов. 3-е изд. — СПб: Питер, 2009. — 384 с.
13. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG. — М.: Вильямс, 2004, 640 с.
14. Математический энциклопедический словарь. — М.: БРЭ, 1995, 847 с.
15. Стерлинг Л., Шапиро Э. Искусство программирования на языке ПРОЛОГ. — М.: Мир, 1990, 326 с.

Журнал «Современная наука: актуальные проблемы теории и практики»