компьютер в учебном процессе

Компьютерные инструменты в образовании, 2019
№ 1: 54–67
УДК: 519.682.5; 519.688+004.43
http://ipo.spb.ru/journal
doi:10.32603/2071-2340-2019-1-54-67

РЕШЕНИЕ НЕКОТОРОГО КЛАССА ЛОГИЧЕСКИХ ЗАДАЧ НА ЯЗЫКЕ PROLOG ДЕКЛАРИРОВАНИЕМ ГЕНЕРАТОРОВ СОСТОЯНИЙ

Половикова О. Н.[1], кандидат физ.-мат. наук, доцент, ponOlgap@gmail.com
Зенков А. В.[2], кандидат физ.-мат. наук, доцент, alexey_zenkov@yahoo.com

[1]Алтайский государственный университет,
пр. Ленина, 61, 656049, Алтайский край, Барнаул, Россия
[2]Алтайский государственный аграрный университет,
пр. Красноармейский, 98, 656049, Алтайский край, Барнаул, Россия

Аннотация

В данном исследовании обозначен и проанализирован способ решения логических задач декларативным языком Prolog на основе метода поиска в пространстве состояний. Суть метода — задекларировать генератор состояний, формирующий пространство поиска, и процедуру отбора для просеивания построенных состояний по определенному принципу. В исследовании предложена классификация способов построения генераторов данным методом. Описаны формальные модели следующих решений: обобщенная задача о переправе, задача на переливание и задача построения кодового слова. Для описания объектов и их состояний предложено использовать битовые цепочки, а для генерации очередного состояния — побитовые операции. Описанные подходы построения искомых вариантов позволяют найти все возможные решения заданной размерности. Обозначены перспективы генерации правил базы знаний. Полученные в ходе исследования примеры декларативных моделей используются в преподавании дисциплины «функциональное и логическое программирование» в Алтайском госуниверситете.

Ключевые слова: *поиск решений, пространство состояний, база знаний, логические задачи, Prolog-система.*

Цитирование: Половикова О. Н., Зенков А. В. Решение некоторого класса логических задач на языке Prolog декларированием генераторов состояний // Компьютерные инструменты в образовании. 2019. № 1. С. 54–67. doi:10.32603/2071-2340-2019-1-54-67

1. ВВЕДЕНИЕ

Проблематика машинного решения задач, в том числе и логических, обозначилась в ранних исследованиях по искусственному интеллекту [1]. Если рассмотреть методы нахождения решений, изучаемые в исследованиях по искусственному интеллекту, то можно обнаружить, что большинство из них используют понятие *поиска путём проб и ошибок* [2, с. 11]. Такой подход послужил основанием для формализации переборного

метода решения — поиск в пространстве состояний. Как и для любого способа автоматического решения задач, данный метод определяется двумя аспектами: представление и поиск. Рассматриваемый метод зарекомендовал себя как достаточно универсальный инструмент решения математических (логических) задач. В исследовании Люгера [3], Рассела и Норвига [4] представлены алгоритмы и практические примеры решения задач данным методом: шахматные ситуации, головоломки, поиск путей в графах, обработка подстановок, построение деревьев и т. д.

Несмотря на широкое распространение данного метода и его практическую универсальность, следует обозначить ряд вопросов, которые остаются открытыми и требуют дальнейших исследований в этом направлении. Остаются открытыми вопросы формализации самого пространства состояний решаемой задачи. Как построить решение, если отсутствует возможность хранения и обработки полного графа состояний (в том числе графа И//ИЛИ)? Эвристические приёмы носят вероятностный характер, но даже они не всегда способны сузить пространство поиска. Взаимодействие поисковых модулей с полным графом «слепого» комбинаторного перебора могут повлиять на эффективность самих поисковых алгоритмов. Вопрос о выборе представления – общий для любого способа решения задач, но, к сожалению, в исследованиях по искусственному интеллекту ещё не выработано универсального автоматического метода для нахождения искусных формулировок задачи [2, с. 19].

Поэтому особого внимания заслуживают работы, в которых поиск решения выстраивается с использованием компактной рекурсивной конструкции, включающей динамическое построение искомого решения и его проверку. На каждом шаге рекурсии происходит формирование (генерация) нового проверяемого в дальнейшем варианта решения задачи или подзадачи. В качестве искомого варианта может быть состояние объектов, их свойств, связей и т. д. Инструментом реализации такого подхода можно выбрать язык логического программирования, в котором используются приемы логического вывода для манипулирования знаниями, представленными в декларативной форме. Тогда достаточно определить и формализовать (задекларировать) базу знаний, которая регламентирует работу генератора и отвечает за проверку сгенерированных решений. Такой подход избавляет от необходимости представления и хранения в памяти компьютера всех вариантов состояний, что зачастую приводит к переполнению стековой памяти программы.

База знаний логического языке Prolog представляет собой предикатную модель фрагмента предметной области — совокупность правил и фактов, которыми декларируется решение задачи. Предикатные модели языка Prolog позволяют хранить знания о способах динамического построения элементов пространства состояний, о правилах их проверки. Декларативные логические языки обеспечивают логический вывод на их основе. На сегодняшний день, высокоуровневые языки логического программирования зарекомендовали себя как инструмент поиска решения нетривиальных задач различного уровня сложности и не только в области искусственного интеллекта [5–8].

Подходы и способы генерации новых элементов востребованы для различных прикладных областей: процедурная генерация контентов для компьютерной игры [9, 10], генерация изоморфного подграфа для абстрактной машины Уоррена [11], генераторы гипотез на логическом языке для реализации гипотетико-дедуктивного метода рассуждения [12, с. 33], генерация узлов концептуального графа для моделирования семантики естественного языка [3, с. 246]. Поэтому исследования вопросов построения генераторов состояний являются актуальными и требуют формализации. В рамках данного исследования ограничимся построением генераторов для некоторого класса логических задач декларативным языком Prolog.

Результатом данного исследования является построение и анализ предикатных моделей решения для некоторого класса логических задач методом поиска в пространстве состояний. Сам метод не является новым, но требует расширения практической базы программными шаблонами в условиях, когда невозможно представить или хранить полный переборный граф всех состояний. В работе предложена классификация подходов построения элементов пространства состояний, а также представлены способы декларирования самих генераторов.

2. КЛАССИФИКАЦИИ РЕШЕНИЙ НА ОСНОВЕ МЕТОДА ПОИСКА В ПРОСТРАНСТВЕ СОСТОЯНИЙ

Сама идея декларирования механизма (генератора состояний) для построения приемлемых вариантов, чтобы Prolog-система могла эти решения оценивать и проверять на согласованность, базируется на принципах декларативного логического программирования. В базу знаний закладываются правила, а система находит допустимое решение или сообщает, что решений нет.

Декларируемая система собирается по частям (см. рис. 1):

1. Генератор состояний, формирующий пространство поиска.
2. Процедура отбора просеивает элементы из пространства состояний сквозь базу знаний для обнаружения такого набора состояний объектов, который является согласованным.

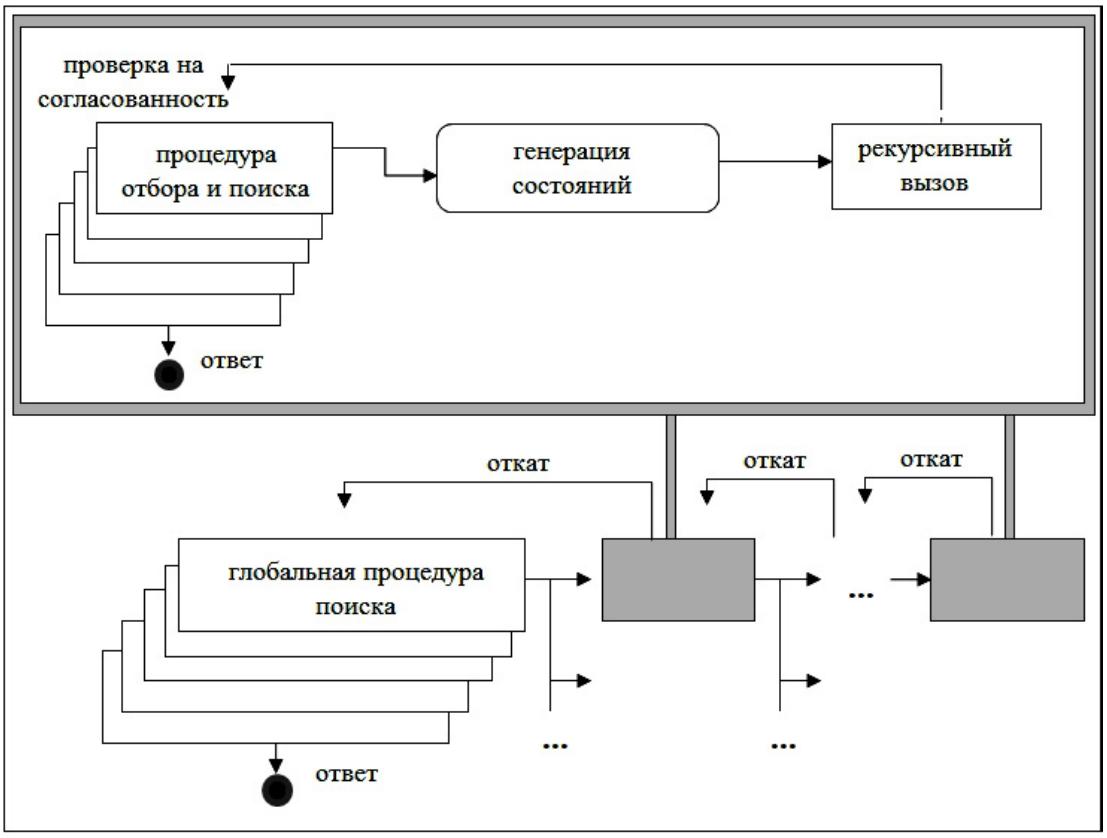

Рис. 1. Схема решения задачи на основе генератора состояний

Проверяемые условия и правила процедуры отбора могут корректироваться по ходу поиска Prolog-системой решения, например учитывая специфику ответов, получаемых генератором состояний. Генератор состояний, в свою очередь, может динамически подстраиваться под работу процедуры отбора.

В рамках данного исследования ограничимся анализом построения только вариантов генератора состояний. При этом все предлагаемые в качестве примеров-шаблонов решения основаны на рекурсивном вычислительном процессе, а не на итерационном. Такое акцентирование на рекурсии обуславливается спецификой рассматриваемых задач и логикой декларирования поиска решения.

Предлагается следующая классификация решений на основе метода поиска в пространстве состояний:

— генерация с сохранением истории,
— генерация без истории.

К первому виду относятся генерации с поэтапным построением итогового решения. При таких подходах на каждом шаге происходит формирование нового элемента для некоторой искомой системы, которая является ответом программы. В качестве примера можно привести генерации данных различных структур: деревьев, стеков, очередей, графов и т. д. Например, в решении задачи 3 генерируется бинарное дерево, где на каждом шаге определяется его очередная ветвь и добавляется в итоговое решение. В работе [11] по шагам генерируется изоморфный подграф: начиная со случайной вершины, пускается волна, которая *накрывает* вершины графа, используя его входящие и исходящие дуги. При построении подграфа необходимого размера набор его вершин сохраняется в памяти для дальнейшего исследования. Ко второму виду относятся решения, которые не сохраняют состояния всех прошедших проверку шагов генерации. Например, если ответом является первое допустимое решение (см. построение предикатной модели для задачи 1).

Нужно подчеркнуть и обособить отличие между хранением подходящих сгенерированных решений для выстраивания общего ответа и использованием данных с предшествующих шагов для получения очередного состояния. Как раз потребностью генераторов состояний основываться в своей работе на свойствах и данных выполнения предыдущих шагов определяется второй способ классификации решений на основе метода поиска в пространстве состояний:

— генерация, определяемая предыдущими решениями,
— генерация без предыдущих решений.

Для первого вида характерным признаком является использование в генерирующих процедурах переменных, которые описывают и передают состояние объектов от одной итерации (или рекурсии) к другой. Очередной генерируемый вариант решения может корректироваться на основе данных, полученных от отклика процедуры отбора. Пример такого решения представлен в данном исследовании (см. решение задачи 1).

В работе [13, с. 112–115] представлены примеры двух вариантов генерации. Предложен способ построения бинарного справочника с уникальными значениями (генерация, определяемая предыдущими решениями). На каждом шаге генерации происходит проверка «на уникальность» очередного бинарного значения, полученного случайным способом. В этой же работе есть примеры предикатов, реализующие очередное построение без учёта предыдущих значений справочника.

Способ генерации определяется логикой поиска решения, которая в программе оформляется последовательностью предикатов. Одной из ключевых проблем декларирования процедуры-генератора выступает построение вычислительного процесса,

который на очередном шаге вызова будет формировать новый ответ, без учета или с учетом предшествующих шагов, не сохраняя/сохраняя историю решения. На следующем шаге исследования рассмотрим предлагаемые способы генерации.

3. ШАБЛОНЫ МОДЕЛЕЙ РЕШЕНИЯ ЛОГИЧЕСКИХ ЗАДАЧ НА ЯЗЫКЕ PROLOG

Процесс генерации можно задекларировать и заставить работать, например, на основе рекурсивного построения цепочек битов, которые генерируются нужного размера добавлением 0 и 1 в начало или в конец некоторого начального решения. Битовую цепочку следует отождествить с состоянием объекта или с набором его свойств. Для генерации очередного состояния можно использовать побитовые операции над уже полученными ранее решениями.

Рассмотрим примеры построения моделей решения математических задач на основе генерации состояний.

Задача 1. Обобщенная задача о переправе. Требуется описать способ перевозки груза с левого берега на правый перевозчиком, используя средство перевозки. Груз — это совокупность из N объектов (включая и самого перевозчика). Объекты, которые участвуют в перевозке, могут находиться в одном из трёх состояний *оставаться на левом берегу*, *перевозиться*, *оставаться на правом берегу*. Объекты, находясь в определенных состояниях, должны подчиняться некоторым ограничениям. Описание способа перевозки груза предполагает поиск и вывод всех промежуточных состояний системы при переходе из начального состояния (*весь груз на левом берегу*) в конечное (*груз на правом берегу*) не более чем за T шагов.

В декларируемой модели решения определим три сущности: левый берег (L), средство перевозки (B), правый берег (R). Перевозимый груз можно пронумеровать и представить упорядоченным конечным набором элементов $O = (O_1, O_2, ...O_N)$. Тогда состояние каждой сущности можно закодировать бинарным упорядоченным конечным набором элементов (цепочкой битов) по следующему принципу: 1 — присутствие, 0 — отсутствие соответствующего объекта на берегу или в средстве перевозки. Определим состояние сущности B в каждый конкретный j-тый момент работы программы цепочками битов $B_j = (B_{j,1}, B_{j,2}, ...B_{j,N})$, где $B_{j,i} \in \{0, 1\}$, то есть перевозятся только те объекты O_i, которым в цепочке соответствует значение 1. Задекларируем бинарную цепочку для B_j соответствующим списком. Подобным образом зафиксируем состояние других сущностей модели решения в каждый конкретный j-й момент (количество необходимых переходов из одного состояния в другое по каждой сущности для перевозки всего набора объектов O как раз и определяет диапазон изменений j, например, $0 < j \leq T$, где T — ограничение на количество перемещений перевозчиком).

Состояние сущностей: левый берег (L), средство перевозки (B), правый берег (R) в каждый j-й момент поиска решения определяется списками:

$$[L_{j,1}, L_{j,2}, ...L_{j,N}], [B_{j,1}, B_{j,2}, ...B_{j,N}], [R_{j,1}, R_{j,2}, ...R_{j,N}], \tag{3.1}$$

где N — это количество объектов перевозки.

Пусть имеется K ограничений для состояний сущностей R и L (без ограничения общности считаем, что у этих сущностей одинаковый набор ограничений, подобный набор векторов можно предусмотреть и для сущности B), определим их также бинарными

упорядоченными конечными наборами: $Y_1, Y_2, \ldots Y_K$. Каждый i-набор характеризует одно ограничение:
$Y_i = (Y_{i,1}, Y_{i,2}, \ldots Y_{i,N}), \quad i = 1..K,$
которое трактуется как недопустимое состояние для сущности. Задекларируем сформированный набор ограничений соответствующими списками:

$$[Y_{i,1}, Y_{i,2}, \ldots Y_{i,N}], \quad i = 1..K \tag{3.2}$$

и унарными предикатами (для Prolog-программы): $y_i([Y_{i,1}, Y_{i,2}, \ldots Y_{i,N}])$.

Для адекватного просеивания (фильтрации) приемлемых состояний сущностей следует определить логику сравнения двух бинарных цепочек, задекларированных списками (см. выражение 3.1 и выражение 3.2). Успешным сравнением ($true$) следует считать исход, при котором состояние сущности удовлетворяет необходимым ограничениям из набора (в нашем случае — всем ограничениям), если хотя бы одно из ограничений не выполнено, тогда результат сравнения — $false$.

Для этих целей определим для класса побитовых цепочек следующие операторы: «побитовое И» (&), «побитовое РАВНО» (=) «побитовое НЕРАВНО» (≠), «побитовое ИЛИ» (|), «побитовое НЕ» (~) и «циклический побитовый сдвиг вправо» (>>) (часть операторов понадобятся для следующих этапов построения модели решения). Распишем логику оператора «побитовое И»: в результирующем списке на i-м месте 1, если на i-м месте 1 у первого и второго операнда, в остальных случаях в результирующем списке на i-м месте0.

Используя перегрузку оператора «побитовое И», можно выразить удачный исход сравнения, например для сущности L (для других сущностей аналогично) следующим выражением:

$$Y_i \& L_j \neq Y_i, \quad i = 1..K. \tag{3.3}$$

Список списков $S_j = [L_j, B_j, R_j]$ определяет состояние модели решения в каждый j-тый момент работы программы. Причем
$S_0 = [[1, 1, \ldots 1], \_, [0, 0, \ldots 0]]$ — начальное состояние системы,
$S_P = [[0, 0, \ldots 0], \_, [1, 1, \ldots 1]]$ — итоговое состояние, где $0 < p \leq T$. Переход в следующее состояние для сущностей R и L (R_j, L_j) определяется их предыдущим состоянием (R_{j-1}, L_{j-1}) и текущим состоянием сущности B (B_j).

Определим логику перехода сущностей из одного состояния в другое, используя перегруженные побитовые операторы. Если перемещение осуществляется с правого берега на левый, то: $R_j = R_{j-1} \& (\sim B_j)$ (забрали пассажиров), $L_j = L_{j-1} | B_j$ (привезли пассажиров); если перемещение с левого берега на правый: $R_j = R_{j-1} | B_j$ (привезли пассажиров), $L_j = L_{j-1} \& (\sim B_j)$ (забрали пассажиров).

Определим логику генератора состояний (без ограничений общности будем считать, что перевозчик перевозит одного пассажира). Каждое очередное состояние сущности B, которое определяется цепочкой битов, генерируется по следующему правилу:

1) самый старший бит (соответствует присутствию перевозчика) следует закрепить в положение 1 (этот бит в работе не участвует),
2) остальные биты циклически сдвигаются вправо на один разряд: $B_j = B_{j-1} >> 1$.

Если после генерации B_j состояния для сущности B, очередные состояния объектов L и R допустимы (выполнены ограничения, представленные выражением 3.3), происходит переход к следующему шагу, в противном случае генерируем очередное состояние для сущности B.

Ниже представлен блок предикатной модели (полная модель по запросу через email) генератора состояний для сущности B. Предикаты $bj/2$ и $cycle\_shift/3$, обеспечивающие выполнение циклического сдвига вправо битовой цепочки любой размерности с учетом фиксации бита в старшем разряде.

```
cycle_shift([X|[]],[],X).
cycle_shift([X|Y],[X|Y1],A):-cycle_shift(Y,Y1,A).
bj(Li,NewL):-Li=[_|L],cycle_shift(L, L1, X), NewL=[1,X|L1].
```

Задача 2. Задача на переливание. Требуется получить заданный объём жидкости X в безмерном сосуде с помощью сосудов объемом A, B. При этом есть источник жидкости для выполнения экспериментов, то есть жидкость в сосуды можно в неограниченном количестве наливать и из них выливать. Требуется получить (найти) решение за наименьшее число ходов (переливаний из одного сосуда в другой). Не будем считать шагами (steps) следующие манипуляции: наполнение сосудов A и B, выливание из сосудов A или B (если при этом не переливаем в сосуд объёмом X).

Построим решение для случая, $X = 7$, $A = 5$, $B = 3$. Предлагается использовать следующий алгоритм:

$X: +5 -3 +5 = 7\ (3\ steps)$,

где обозначение $+N$ следует интерпретировать: перелить в сосуд X из сосуда объема N, $-N$ следует интерпретировать: вылить из сосуда X в сосуд объёма N.
Очевидно, что данное решение не является единственным. Подберём другое решение для данной задачи:

$X: +3 +3 -5 +3 +3 -5 +5 = 7\ (7\ steps)$

Попытаемся проанализировать и упростить данное выражение, например, можно сократить элементы: +5 и −5. После преобразования получаем следующий алгоритм:

$X: +3 +3 -5 +3 + 3 = 7\ (5\ steps)$

Сгруппируем равные элементы и перепишем полученное выражение:

$X: +3\ (4\ раза) + -5\ (1\ раз) = 7$, или
$X: +3 * 4 + -5 * 1 = 7$, или
$X: +4 * 3 + -1 * 5 = 7$

Если объемы сосудов записать через параметры A и B, то модель решения данной задачи можно представить выражением: $a * A + b * B = X$, где a и b искомые значения (целые числа), $a, b \in Z$. A и B — параметры (переменные), значение которых определяется перед поиском решения (обоснование представленной модели описано одним из авторов в работе [14]).

Если условием задачи является получение необходимого объёма за **минимальное число шагов (переливаний)**, тогда искомые значения следует связать формулой: $|a| + |b| \rightarrow min$ или $sum(|a|, |b|) \rightarrow min$. Предлагается следующая предикатная модель решения данной задачи, где предикат $generate/2$, формирует новое значение для переменной a, двигаясь в сторону увеличения абсолютного значения. Предикат $seach/3$ для очередного значения переменной a по формуле

$$b = \frac{X - a * A}{B}, \quad B \neq 0$$

вычисляет значение для переменной b. Получение **целочисленного значения для переменной** b интерпретируется как «решение найдено».

```
% Поиск решения для A=27, B=11, X=3

generate(Old,New):- Old>0, New:=(-1)*Old.
generate(Old,New):- not(Old>0), New:=(-1)*Old+1.

seach(X,[A,B,C],[X,Y]):-(C-A*X) mod B =:=0,Y:=(C-A*X)//B.
seach(X,[A,B,C],L):-generate(X,X1),seach(X1,[A,B,C],L).

?-seach(0,[27,11,3],[X,Y]),write([X,Y]),
seach(0,[11,27,3],[X1,Y1]),write([X1,Y1]).

% Вывод программы: [5,-12] и [-12,5]
```

Таким образом, рассматриваемая задача может быть формализована и задекларирована для поиска решения Prolog-системой, если задать Prolog-процедуру генерации параметров a и b. Предикаты $generate/2$ и $seach/3$ формируют варианты пар в сторону увеличения их абсолютных значений. Тогда, очевидно, что первое найденное решение и будет гарантированно иметь минимальное количество шагов (переливаний). Следует заметить, что предлагаемый подход к решению можно обобщить и на другие *задачи на переливание*.

Задача 3. Построение кодового слова. По каналу связи передаются сообщения, содержащие только буквы. Для передачи используется двоичный код, допускающий однозначное декодирование. За некоторыми буквами уже закреплены их кодовые слова. Укажите кратчайшее кодовое слово для ещё одной буквы, при котором код будет допускать однозначное декодирование.

При декларировании модели решения данной задачи следует учесть, что:

1) буквенные коды строятся на основе комбинаций 0 и 1, например 01, 10;
2) для однозначного декодирования сообщения должно выполняться одно из *условий Фано* (например для префиксного кода): никакое кодовое слово не является началом другого кодового слова;
3) процесс поиска должен искать варианты кода по возрастанию их длин: 0, 00, 01, 10, 11, 011, 101, 110, 111 и т. д.

Моделью решения является неполное бинарное дерево с возможно разной степенью узлов. Из каждой вершины формируются две вершины, которые обозначаются 0 и 1. Если в некоторых узлах такого дерева расположить буквы, тогда коды для букв можно определить по следующему принципу: обозначение полного пути из корня дерева к данному узлу, используя введенные обозначения. Поиск решения базируется на определении количества незанятых узлов, а это определяется глубиной разветвления дерева.

Для решения данной задачи актуальным является определение правила, которое отвечает за построения очередного кода буквы. Такое правило должно определенным образом генерировать последовательности из 0 и 1 (цепочку битов). Если сгенерированный код пройдёт проверку «условием Фано», тогда процесс построения кода следует считать выполненным, решение найдено. Если проверка не пройдена, тогда генерируется следующий код, который также подвергается проверке. Таким образом, база знаний будет включать:

— факты, декларирующие уже занятые бинарные коды;
— правила, для проверки *условия Фано*;
— правило генерации бинарного кода.

Ниже представлен вариант построения решения данной задачи (задекларирована предикатная модель), как и в случае предикатной формализации задачи 3, для хранения данных используются списки, для построения логики генерации и фильтрации решений реализованы необходимые процедуры манипуляции над списками. Предположим также, что следующие коды уже заняты буквами: 111, 0, 100.

```
% Блок описания фактов
kod(_, [1, 1, 1]).
kod(_, [0]).
kod(_, [1, 0, 0]).
% Блок вспомогательных процедур правил
concat([],L,L).
concat([X|L],M,[X|L1]):- concat(L,M,L1).
% Блок проверки нарушений условая Фано
cteck(M):- kod(_,K),(concat(K,_,M); concat(M,_,K)).
checklist([],_).
checklist([H|T],Rezult):- nl,(check(H)->(write(no),write(H));
(write(yes),write(H),Rezult:=[H|Rezult])),
checklist(T,Rezult).
% Блок генерации нового кода
generate([],[]).
generate([H|T],[X,Y|T1]):- X=[0|H], Y=[1|H], generate(T,T1).
search(L,5,Rezult):- nl, write(Rezult).
search(L,N,Rezult):- not(N>4), N:=N+1, generate(L,L1),
checklist(L1,Rezult), search(L1,N,Rezult).
% Блок описания целевого запроса
?- search([[]],1,[]).
```

В Блоке генерации на очередном шаге рекурсии обновляется список вариантов для нового кода, за это отвечает предикат *generate*/2. Предикат *search*/3 собирает результаты генерации, передает их на проверку и следит за длиной списков, которые формируются правилами. Для решения данной задачи ввели ограничение на длину искомого кода до 5 элементов. Такое *ограничение* не влияет на построенную базу фактов и правил, может формироваться динамически, исходя из длины уже известных кодов букв или длины результирующего списка с уже проверенными вариантами кодов.

В решении отсутствует процедура правил для поиска именно кратчайшего кодового слова (согласно условию задачи). Поиск всех возможных вариантов (с учётом максимальной длины кода) выполнен намеренно, чтобы подчеркнуть возможности процедуры генерации состояний объектов (кодовых слов) при решении подобных задач. Например, если требуется построить несколько кодовых слов, или всевозможные кодовые слова определенной длины, или кодовое слово с наименьшим значением в числовом бинарном представлении.

Предикаты генерации создают коды от самых коротких ([[0], [1]]) до слов с пороговой длиной (см. рис. 2). Искомый кратчайший код добавляется в результирующий список самым первым. Поэтому решение данной задачи можно завершить при построении первого кода, который прошёл проверку.

Построение решения средствами генерации кода с проверкой позволяет не хранить в базе знаний бинарное дерево всех возможных кодов. Без процедуры генерации кода процесс решения можно свести к описанию бинарного дерева всех кодов и к поиску незанятых в нём узлов. Но такой альтернативный подход ставит встречную задачу *по*

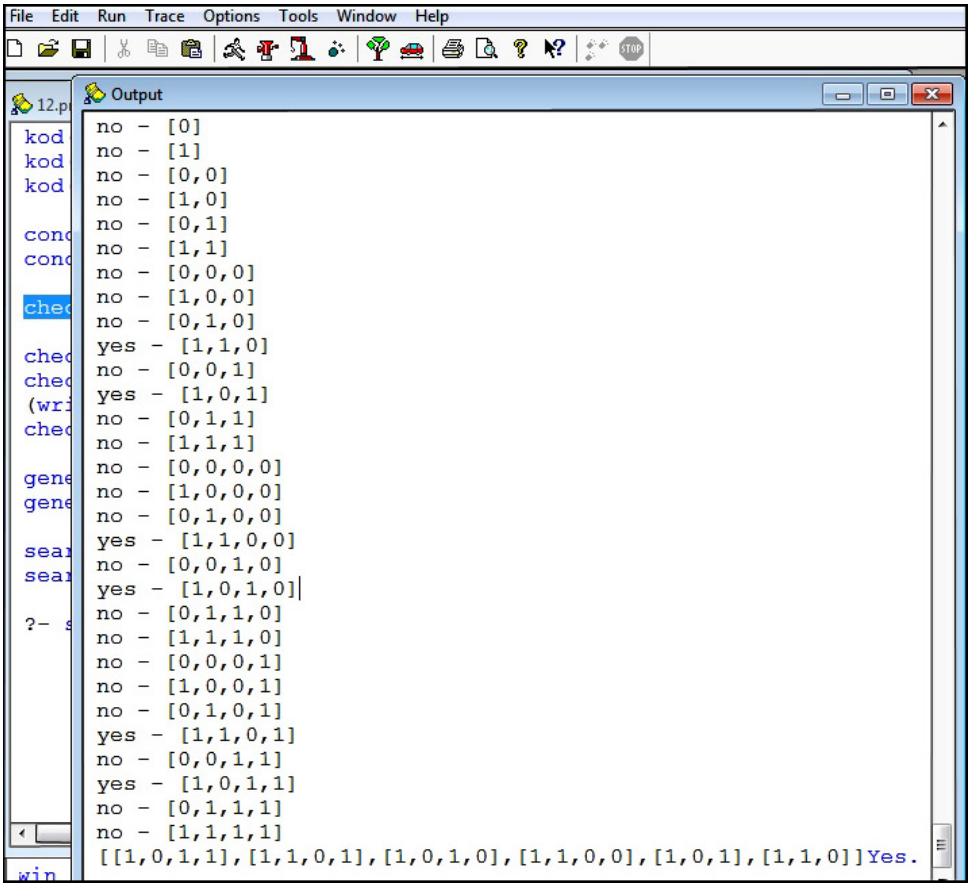

Рис. 2. Результаты решения задачи 3

определению уровня хранимого программой бинарного дерева (так как глубина его разветвления должна гарантировать необходимый поиск по основной задаче).

Немаловажным фактором для оценки выбранного подхода к построению предикатной модели решения является корректный отклик декларативной программы (базы знаний и процедуры вывода) на ситуацию, когда целевой предикат не может быть согласован при заданных условиях. Prolog-система должна выдать отрицательный ответ для случая, когда нельзя построить кодовое слово для искомой буквы (см. условие задачи 3), например, если все вершины дерева заняты другими буквами (нет свободных вариантов построения кода с учётом *условия Фано*). В этом случае соответствующие предикаты построят и проверят варианты для искомого бинарного кода в рамках заданных ограничений по размерности и сформируют пустой результирующий список.

4. ГЕНЕРАЦИЯ САМИХ ПРЕДИКАТОВ В МОДЕЛЯХ РЕШЕНИЯ

Представленный в работе классификатор решений на основе генератора пространства состояний, а также полученные в ходе проведенного исследования примеры кодов программ используются в преподавании дисциплины «Функциональное и логическое программирование» для студентов факультета математики и информационных технологий в Алтайском государственном университете.

Представленные выше подходы генерации вариантов методом поиска в пространстве состояний позволяют найти все возможные решения заданной размерности. Но де-

кларировать предикаты на языке Prolog можно не только для генерации самих вариантов решений. Процедурами правил можно генерировать и базу знаний, то есть обучать поисковые предикаты по ходу выполнения программы. Примером динамического построения самой предикатной модели является программа формирования семантической сети на языке Prolog, которая генерирует модуль prolog-кода, основываясь на базах начальных знаний [15, с. 96].

Продемонстрировать работу механизма генерации фактов с целью обучения программы можно на примере решения элементарных вычислительных математических задач: поиск *факториала* или *чисел Фибоначчи*. Процедура правил и фактов для вычисления *чисел Фибоначчи* основана на рекурсивном вызове данных процедур и декларируется базовыми средствами Prolog-системы. Полученное программой очередное *число Фибоначчи* (решение) добавляется в начало базы знаний в виде факта, учитывается при вычислении следующих чисел. При последующем запуске программы поиск решения осуществляется среди уже построенных фактов, а только потом среди правил – так количество рекурсивных вычислений с каждым запуском программы сокращается. За счет сокращения рекурсивных переходов происходит экономия времени вычислений и стековой памяти для передачи значений переменных.

Совместное использование процедур генерации для фактов и правил, которые создают новые варианты для пространства состояний и самих вариантов состояний, позволяет построить решение даже в условиях неполного начального декларирования модели знаний. Такая неопределенность должна разрешаться в процессе работы программы. Динамическое достраивание процедур блока генерации с учетом результатов проверки построенных ранее вариантов позволяет управлять поиском решения, по крайней мере, добиться ограничения пространства состояний «слепого» комбинаторного перебора.

Несмотря на случайный характер реализованного переборного метода (см. модели решения задачи 1-3, задачи 2, задачи 3), последовательное перемещение по всему пространству состояний позволяет найти требуемый вариант, если такой можно построить. Движение генератора по элементам пространства состояний ничем не координируется, последовательно в битовые цепочки добавляются 0 и 1 (задача 3), побитовое смещение вправо (задача 1), увеличение абсолютного значения параметра (задача 2). Отсутствуют знания для алгоритмического формирования нового элемента пространства, с каждым шагом приближаясь к решению, нет возможности оценить, насколько очередной вариант ближе к требуемым условиям, чем предыдущее построение. Но даже в отсутствие эвристических надстроек универсальность метода поиска в пространстве состояний гарантирует построение решения. «Слепой» («случайный») перебор элементов в процессе поиска пересекается с подходами к решению задачи человеком в условиях невозможности применения других *непереборных* методов.

Список литературы

1. *Бенерджи Р.* Теория решения задач. Подход к созданию искусственного интеллекта. М.: Мир, 1972.
2. *Нильсон Н.* Искусственный интеллект. Методы поиска решений. М.: Мир, 1973.
3. *Люгер Джордж Ф.* Искусственный интеллект: Стратегии и методы решения сложных проблем, 4-е изд.: Пер. с англ. М.: Вильямс, 2003.
4. *Рассел С., Норвиг П.* Искусственный интеллект. Современный подход: Пер. с англ. М.: Вильямс, 2006.
5. *Котеленко С. А.* Язык Prolog и реализация концепции Semantic Web // Известия Южного федерального университета. Технические науки. 2002. Т. 27, № 4. С. 121–128.

6. *Девятков В. В., Мьё Тхет Наунг.* Мультиагентный анализ правильности спецификаций протоколов инициирования сеансов // Вестник Московского государственного технического университета им. Н. Э. Баумана. Серия «Приборостроение». 2015. № 2(101). С. 107–116.
7. *Морозов А. А., Сушкова О. С.* Анализ видеоизображений в реальном времени средствами языка Акторный Пролог // Компьютерная оптика. 2016. Т. 40, № 6. С. 947–957. doi: 10.18287/2412-6179-2016-40-6-947-957
8. *Половикова О. Н.* Формализация процесса построения решения с использованием списков для класса логических задач в программах на языке Пролог // Известия Алтайского государственного университета. 2011. Т. 69. № 1-1. С. 117–120. doi: 10.14529/cmse150101
9. *Меженин М. Г.* Обзор систем процедурной генерации игр // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 1. С. 5–20. doi: 10.14529/cmse150101.
10. *Shaker N., Togelius J., Nelson M.* Procedural Content Generation in Games. Computational Synthesis and Creative Systems: A Textbook and an Overview of Current Research. Springer. 2016.
11. *Ильяшенко М. Б., Голдобин А. А.* Решение задачи поиска изоморфизма графов для проектирования специализированных вычислителей // Радиоэлектроника, информатика, управление. 2012. № 1(26). С. 31–36. doi: 10.14357/19922264150104
12. *Kalinichenko L., Kovalev D., Kovaleva D., Malkov O.* Methods and tools for hypothesis-driven research support: a survey // Informatics and applications. 2015. Vol. 9. Iss. 1. P. 28–54. doi: 10.14357/19922264150104
13. *Шрайнер П. А.* Основы программирования на языке Пролог: курс лекций. М.: Интернет – Ун-т Информ. Технологий. 2005.
14. *Половикова О. Н.* Генератор параметров для построения выражения к математической задаче на языке Prolog // Ломоносовские чтения на Алтае: фундомент. проблемы науки и техн., сборник статей международной конференции (state. reg. num. 0321804205). 2018. С. 724–725.
15. *Брусакова И. А., Мамаева С. О.* Система управления базами измерительных знаний // Прикладная информатика. 2006. № 5. С. 93–97.

Поступила в редакцию 18.02.2019, окончательный вариант — 21.03.2019.

Computer tools in education, 2019
№ 1: 54–67
http://ipo.spb.ru/journal
doi:10.32603/2071-2340-2019-1-54-67

Solving a Certain Class of Logical Tasks on the Prolog Language by Declaring State Generators

Polovikova O. N.[1], PhD, associate professor, ponOlgap@gmail.com
Zenkov A. V.[2], PhD, associate professor, alexey_zenkov@yahoo.com

[1]Altai State University, 61, pr. Lenina, 656049, Barnaul, Altai Krai, Russia
[2]Altai State Agricultural University, 98, pr. Krasnoarmeyskiy, 656049, Barnaul, Altai Krai, Russia

Abstract

In this study, a method for solving Logical Tasks in the declarative language Prolog is identified and analyzed using the State Space Search Method. The essence of this method is to declare a State Generator, which forms the Search Space, and a Selection Procedure for sifting the constructed states according to a certain principle. In this study

a classification of techniques is proposed for constructing generators by this method. The formal models of the following solutions are described: The Generalized Crossing Task, The Transfusion Task and The Code Word Construction Task. To describe objects and their states, it was proposed to use Bit Chains, and to generate the next state — Bitwise Operations. The described approaches for constructing the desired variants make it possible to find all possible solutions of a given dimension. The prospects for generating knowledge base rules are indicated. Examples of declarative models, obtained in the course of this study, are used in the course "Functional and Logic Programming" in the Altai State University.

Keywords: *search for solutions, state space, knowledge base, logical tasks, Prolog-system.*

Citation: O. N. Polovikova and A. V. Zenkov, "Solving a Certain Class of Logical Tasks on the Prolog Language by Declaring State Generators," *Computer tools in education*, no. 1, pp. 54–67, 2019 (in Russian); doi: 10.32603/2071-2340-2019-1-54-67

References

1. R. Benerdzhi, *Teoriya resheniya zadach. Podkhod k sozdaniyu iskusstvennogo intellekta* [Theory of problem solving. Approach to the creation of artificial intelligence], Moscow, USSR: Mir, 1972 (in Russian).
2. N. Nilson, *Iskusstvennyi intellekt. Metody poiska reshenii* [Artificial Intelligence. Solution Search Methods], Moscow, USSR: Mir, 1972 (in Russian).
3. D. F. Luger, *Iskusstvennyi intellekt: Strategii i metody resheniya slozhnykh problem* [Artificial Intelligence: Strategies and Methods for Solving Difficult Problems], 4th ed., Moscow, Russia: Williams, 2003 (in Russian).
4. S. Rassel and P. Norvig, *Iskusstvennyi intellekt. Sovremennyi podkhod* [Artificial Intelligence. Modern approach], Moscow, Russia: Williams, 2006 (in Russian).
5. S. A. Kotelenko, "Yazyk Prolog i realizatsiya kontseptsii SemanticWeb" [Prolog language and implementation of the concept of SemanticWeb], *Izvestiya Yuzhnogo federal'nogo universiteta. Tekhnicheskie nauki*, vol. 27, no. 4, pp. 121–128, 2002 (in Russian).
6. V. V. Devyatkov and M'e Tkhet Naung, "Mul'tiagentnyi analiz pravil'nosti spetsifikatsii protokolov initsiirovaniya seansov" [Multi-agent analysis of the correctness of the session initiation protocols], *Vestnik Moskovskogo gosudarstvennogo tekhnicheskogo universiteta im. N.E. Baumana. Seriya "Priborostroenie"*, no. 2, pp. 107–116, 2015 (in Russian).
7. A. A. Morozov and O. S. Sushkova, "Analiz videoizobrazhenii v real'nom vremeni sredstvami yazyka Aktornyi Prolog" [Real-time video image analysis using Actor Prolog language], *Computer optics*, vol. 40, no. 6, pp. 947—957, 2016 (in Russian); doi: 10.18287/2412-6179-2016-40-6-947-957
8. O. N. Poloikova, "Formalizatsiya protsessa postroeniya resheniya s ispol'zovaniem spiskov dlya klassa logicheskikh zadach v programmakh na yazyke Prolog" [Formalization of the decision building process using lists for a class of logical problems in Prolog programs], *Izvestiya Altaiskogo gosudarstvennogo universiteta*, vol. 69, no. 1, pp. 117–120, 2011 (in Russian).
9. M. G. Mezhenin, "Obzor sistem protsedurnoi generatsii igr" [Review of procedural game generation systems], *Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta. Seriya: Vychislitel'naya matematika i informatika*, vol. 4, no. 1, pp. 5–20, 2015 (in Russian); doi: 10.14529/cmse150101
10. N. Shaker, J. Togelius and M. Nelson, *Procedural Content Generation in Games. Computational Synthesis and Creative Systems: A Textbook and an Overview of Current Research*, Springer, 2016.
11. M. B. Il'yashenko and A. A. Goldobin, "Reshenie zadachi poiska izomorfizma grafov dlya proektirovaniya spetsializirovannykh vychislitelei" [The solution of the problem of searching for isomorphism of graphs for designing specialized calculators], *Radioelektronika, informatika, upravlenie*, no. 1, pp. 31–36, 2012 (in Russian).
12. L. Kalinichenko, D. Kovalev, D. Kovaleva, and O. Malkov, "Methods and tools for hypothesisdri-

ven research support: a survey," *Informatics and applications*, vol. 9, no. 1, pp. 28–54, 2015; doi: 10.14357/19922264150104
13. P. A. Shrainer, *Osnovy programmirovaniya na yazyke Prolog: kurs lektsii*, Moscow, Russia: Internet – Un-t Inform. Tekhnologii, 2005 (in Russian).
14. O. N. Polovikova, "Generator parametrov dlya postroeniya vyrazheniya k matematicheskoi zadache na yazyke Prolog" [Parameter generator for constructing an expression for a mathematical problem in the Prolog language], *Lomonosovskie chteniya na Altae: fundoment. problemy nauki i tekhn., sbornik statei mezhdunarodnoi konferentsii*, pp. 724–725, 2018 (in Russian).
15. I. A. Brusakova and S. O. Mamaeva, "Sistema upravleniya bazami izmeritel'nykh znanii," *Prikladnaya informatika*, no. 5, pp. 93–97, 2006 (in Russian).

Received 18.02.2019, the final version — 21.03.2019.