Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 6

по курсу «Экспертные системы»

на тему: «Обратный дедуктивный вывод в обобщенных правилах продукции»

Студент ИУ7-31М (Группа) _____ (Подпись, дата) Клименко А. К. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Русакова З. Н. (И. О. Фамилия)

2024 г.

Введение

Цель лабораторной работы — приобретение практических навыков реализации алгоритма обратного дедуктивного вывода на обобщенных правилах продукции.

Задачи работы:

1. Формализовать алгоритм обратного дедуктивного вывода.
2. Разработать программу, реализующую алгоритм обратного дедуктивного вывода.
3. Протестировать программу на различных логических программах.

1 Теоретический раздел

1.1 Обобщенное правило продукции

Пусть x — задан на конкретном множестве, а c — объект этого множества, тогда

$$\frac{A(c), A(x) \to B(x)}{B(c)}.$$

В общем случае, если в посылке конъюнкция и заданы подстановки для каждого P_i, то мы можем получить композицию подстановок.

Отсутствие направления вывода в методе резолюции называется **потерей импликативности**. Импликативный вывод используется для простых случаев.

1.2 Система дедукции на основе обобщенных правил продукции

Определенное выражение является либо атомарным (атомом), либо представляет собой импликацию антицидентом (предпосылкой) которой является конъюнкция положительных атомов, а консеквент — единственный положительный атом.

В определенных выражениях в базе знаний не используются функциональные символы, но определенные выражения содержат направление вывода, что намного повышает эффективность систем вывода и упрощает его. Метод резолюции позволяет обработать более сложные представления знаний, что определенные выражения не всегда позволяют, но процедура резолюции влечет потерю импликативности, то есть нет направления вывода.

Пример:

$$\begin{aligned} (\neg A \& \neg B) &\to C, \\ (\neg B \& \neg C) &\to A \end{aligned}$$

дают одинаковые дизъюнкты, но видим, что в исходных формулах разное направление вывода.

1.3 Алгоритм обратного дедуктивного вывода

Обратная дедукция для базы знаний из определенных выражений производится методом поиска в глубину. Структуры данных: переменные, константы, атом. База правил представлена списком правил. Правило содержит список входных атомов, которые соответствуют входным вершинам, один выходной атом (выходная вершина), номер правила, флаг доказано/недоказано и метку (выбрано или нет). В атоме тоже вводится флаг (помимо списка переменных) доказан атом или не доказан. Факты — это атомы, в которых стоят константы. Они соответствуют закрытым вершинам в графах И-ИЛИ (мы их задаем) и еще есть целевой атом (целевая вершина). Возможно что целевой атом имеет константы, а может не имеет.

Целевой атом записываем в голову стека. Факты записываем в список закрытых вершин. В процессе поиска формируем стек открытых вершин (атомов); список подстановок для текущего шага; список закрытых атомов / закрытых вершин, которые добавляются к исходному списку фактов; список закрытых правил, которые содержат дерево решения.

Метод поиска остается без изменений. Пока оба флага выставлены в единицу, вызываем метод потомков на каждом шаге поиска. Изменяется метод потомков и метод разметки. В этих методах необходимо проводить унификацию и формировать подстановки. В методе потомков выполняется унификация атома подцели (из стека) и выходной атом рассматриваемого правила. Унифицируются подцель и выходной атом правила. Если унификация успешна, то формируем подстановки. Причем, константа может стоять как в подцели так и в выходном атоме правила. Если выходной атом получает константу, то её необходимо распространить на все атомы рассматриваемого правила, в которое входит переменная, получившая константу. Если константу получают переменные подцели, то она ставится в подцель. При этом можно рассмотреть 2 пути: если подцель получила константу, то распространить эту константу на оставшиеся в стеке атомы последнего правила. Альтернатива — можно не распространять константу из подцели, а оставить их с переменными, — они получат значения потом при их доказательстве.

Итог: выбираем подцель, находим первое правило, у которого подцель унифицируется с выходной вершиной. Нам нужно проверить, сколько атомов у этого правила уже закрыто (входные вершины доказаны). Для этого мы

каждый атом входной проверяем, не находится ли он в базе. Если он есть, то его переменные получают значения. Эти значения распространяем в другие атомы. В стек записываем только те атомы, которые не закрываются фактами. Если мы записали подцели, то номер правила помещаем в список открытых правил.

Подстановка необходима на одном шаге раскрытия подцели и формирования новых подцелей. Потом она уже не нужна, но можно использовать.

Следующий шаг — если у найденного правила все атомы доказаны. Это правило добавляем в список закрытых и начинаем разметку. В разметке мы должны выходную вершину правила добавить к фактам. В результате подстановки, мы должны полученные значения переменных предыдущих правил распространить на эти же переменные в оставшихся недоказанных атомах (если они есть). Идет обратное распространение переменных (снизу вверх в подцели). Соответственно, из головы стека подцель удаляем. Если недоказанные атомы для последнего правила были, то вызываем потомков. Если недоказанных атомов больше нет, то продолжаем цикл разметки.

Рассмотрим построение для наших правил.

Факты:

1. $O(N, M1)$
2. $M(M1)$
3. $A(W)$
4. $E(N, A1)$

Правила:

1. $A(x)\&W(y)\&S(x, y, z)\&H(z) \to C(x)$
2. $M(x_2)\&O(N, x_2) \to S(W, x_2, N)$
3. $M(x_1) \to W(x_1)$
4. $E(x_3, A1) \to H(x_3)$

Цель: $C(x_0)$

Шаги:

1. стек: $[C(x_0)]$

   $C(x_0)$ унифицируется с выходным атомом правила 1. Подстановка: $\{x_0/x\}$ Мы должны проверить, какие атомы этого правила закрыты. Берем $A(x)$ ищем в базе фактов, находим $A(W)$ — x заменяется на константу W. Распространяем переменную на остальные факты. $A(W)$ в стек не пишем, рассматриваем следующую подцель. $W(y)$ в базе фактов нет, поэтому кладем в стек:

   стек: $[W(y), S(W, y, z), H(z), C(W)]$

2. второй шаг метода потомков — $W(y)$ унифицируется с выходом правила 3. Подстановка: $\{y/x_1\}$. Нам нужно проверить у этого правила, какие вершины не доказаны и что писать в стек, в нашем случае при доказательстве он сопоставляется с фактом $M(M1)$, когда мы будем доказывать входную вершину получим подстановку $\{M1/y\}$. Правило 3 закрываем. Добавляем факт $W(M1)$ в базу фактов. Распространяем подстановку $\{M1/y\}$ на оставшиеся атомы.

   стек: $[S(W, M1, z), H(z), C(W)]$, а в базе фактов новый факт: $W(M1)$. Из разметки вышли — $W(y)$ убираем, а недоказанные подцели есть. Теперь будем доказывать $S(W, M1, z)$.

3. В результате унификации $S(W, M1, z)$ с выходом правила $S(W, x_2, N)$ надо распространять в оба направления — x_2 вниз (в подцели для S), z вправо (в $H(z)$). Подцели S будут доказаны из фактов, из этого следует, что вершина $S(W, M1, N)$ — новый факт, который мы добавим в базу фактов и уберем из стека.

4. Подцель $H(N)$. Правило 4. Входной модуль $E(x_3, A1)$. В результате распространения получаем $E(N, A1)$. Такой факт у нас есть. $H(N)$ в новые факты, убираем из стека.

5. Вызываем разметку для $C(W)$ — удаляем из стека. Стек пуст — исходное утверждение доказано.

2 Практический раздел

База правил хранит список правил. Факты представлены структурой правила с пустым списком входных атомов.

2.1 Формализация алгоритма обратного дедуктивного вывода

Алгоритм обратного дедуктивного вывода **Backward-ASK** работает на основе генераторов для обеспечения поиска всех возможных решений для цели с возможностью приостановки поиска. Главная идея алгоритма — обратный поиск в глубину в графе И-ИЛИ с использованием двух процедур — **Backward-OR** и **Backward-AND**.

Алгоритм 2.1. Backward-ASK. Алгоритм обратного дедуктивного вывода

Вход: KB — база правил, *цель* — атом.
Выход: генератор подстановок

1: **Вернуть** Backward-OR (KB, *цель*, {})

Процедура **Backward-OR** производит перебор правил, которые *разрешают* текущую подцель (то есть выходная вершина которых унифицируется с текущей подцелью), и после переименования переменных в найденном правиле вызывает процедуру **Backward-AND** для доказательства входных атомов текущего правила с применением подстановки, полученной в процессе унификации заключения текущего правила и текущей подцели. Если текущее правило оказалось фактом, его посылка будет пуста. Эту ситуацию обработает процедура **Backward-AND**.

Алгоритм 2.2. Backward-OR. Алгоритм перебора правил ИЛИ

Вход: KB — база правил, *подцель* — атом, θ — базовая подстановка.
Выход: генератор подстановок

1: **Цикл** по всем *правилам* $\leftarrow KB$ разрешающим *подцель* **выполнять**
2: ($посылка \Rightarrow заключение$) $\leftarrow$ **STD-VAR**(*правило*)
3: **Цикл** $\theta' \in$ **Backward-AND**(KB, *посылка*, **UNIFY**(*заключение*, *подцель*, θ)) **выполнять**
4: **Выбросить** подстановку θ'
5: **Конец цикла**
6: **Конец цикла**

Функция **STD-VAR** выполняет переименование переменных входного правила и возвращает новое правило.

Функция **UNIFY** производит унификацию двух атомов с заданной подстановкой и возвращает новую подстановку. В случае невозможности унификации, функция возвращает ошибку.

Процедура **Backward-AND** выполняет доказательство входного списка подцелей. Для этого выполняется перебор всех возможных подстановок для первой подцели с использованием процедуры **Backward-OR**. Затем выполняется рекурсивный вызов для доказательства остальных подцелей с полученной подстановкой.

Алгоритм 2.3. Backward-AND. Алгоритм перебора правил И

Вход: KB — база правил, *подцели* — список атомов, θ — базовая подстановка.

Выход: генератор подстановок

Если θ — ошибка **тогда Вернуть**
иначе Если список *подцелей* пуст **тогда**
    Выбросить подстановку θ
иначе
    голова, *хвост* $\leftarrow$ список *подцелей*
    Цикл $\theta' \in$ **Backward-OR**(KB, **SUBST**(*голова*, θ), θ) **выполнять**
        Цикл $\theta'' \in$ **Backward-AND**(KB, *хвост*, θ') **выполнять**
            Выбросить подстановку θ''
        Конец цикла
    Конец цикла
Конец условия

Функция **SUBST** применяет подстановку к входному атому и возвращает новый атом.

Рассмотрим пример работы описанного алгоритма обратного дедуктивного вывода. Пусть база знаний содержит следующие правила (факты являются частными случаями правил):

$$B(y, W)\ \&\ C(x, K, y) \rightarrow A(x, y), \tag{2.1}$$

$$D(y)\ \&\ E(x, K) \rightarrow B(x, y), \tag{2.2}$$

$$B(z, x)\ \&\ E(y, z) \rightarrow C(x, y, z), \tag{2.3}$$

$$D(W), \tag{2.4}$$

$$E(K, K). \tag{2.5}$$

Цель: $A(r, s)$.

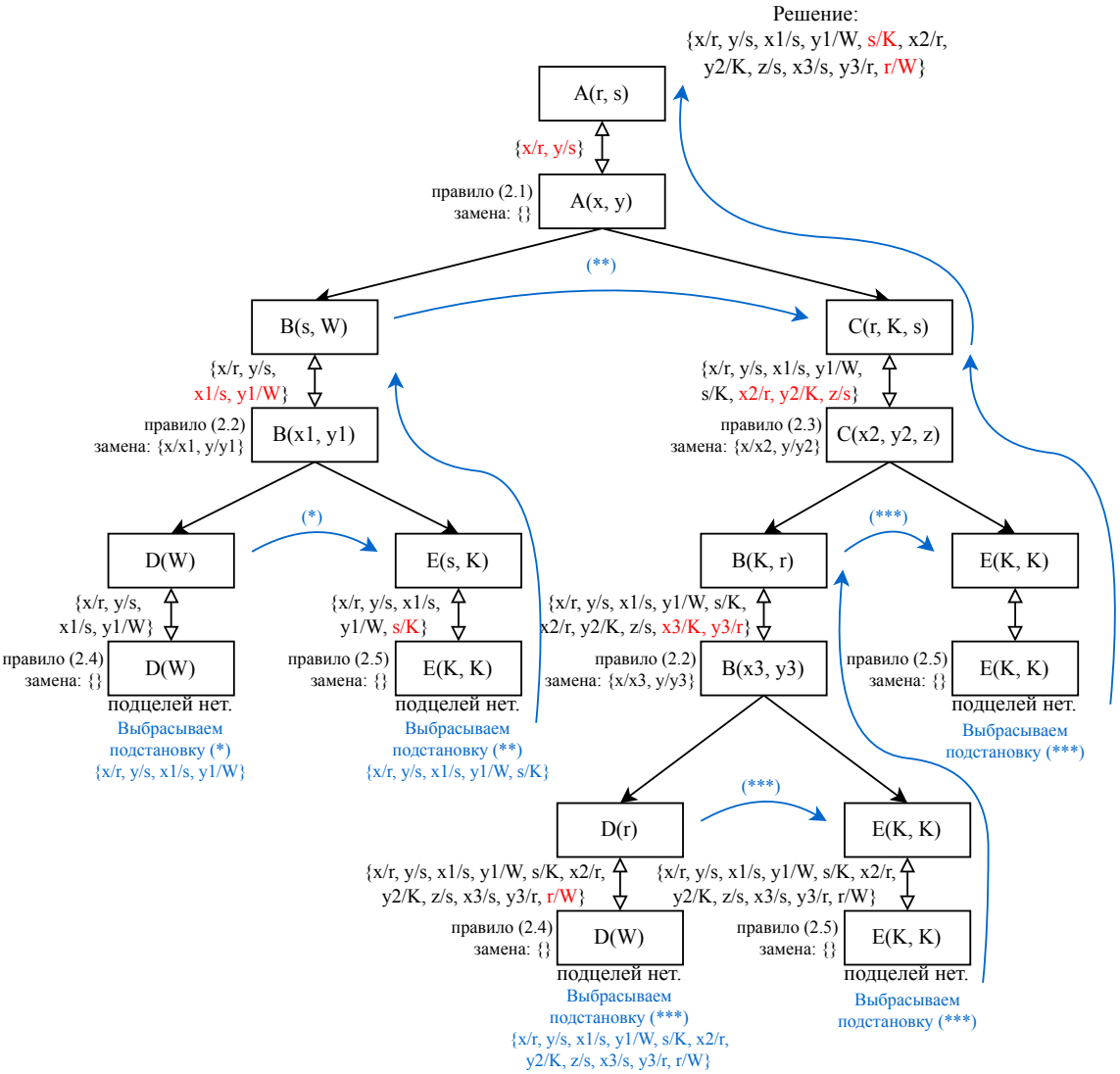

Рисунок 2.1 – Пример простой схемы вывода

Схема вывода показана на рисунке 2.1. Синим цветом указано направление передачи выбрасываемых подстановок из генераторов.

2.2 Поиск всех допустимых подстановок

Рассмотрим другой пример, иллюстрирующий возврат. Пусть база знаний состоит из следующих правил:

$$Mark(st, sub, 5)\ \&\ Mark(st, sub, 4) \rightarrow GoodAt(st, sub) \tag{2.6}$$

$$Mark(Alice, Maths, 4) \tag{2.7}$$

$$Mark(Alice, Maths, 5) \tag{2.8}$$

$$Mark(Bob, Maths, 3) \tag{2.9}$$

$$Mark(Bob, Maths, 4) \tag{2.10}$$

$$Mark(Carl, Phys, 5) \tag{2.11}$$

$$Mark(Carl, Phys, 4) \tag{2.12}$$

$$Mark(Alice, Phys, 4) \tag{2.13}$$

$$Mark(Bob, Phys, 4) \tag{2.14}$$

$$Mark(Bob, Phys, 3) \tag{2.15}$$

Цель — найти все такие пары «студент–предмет», что студент *st* имеет оценки 4 и 5 по предмету *sub*. То есть найти все возможные подстановки для предиката $GoodAt(st, sub)$. Схема перебора допустимых подстановок представлена на рисунке 2.2.

2.3 Отсечение дерева решений

Отсечение дерева решений реализуется введением дополнительного флага, передаваемого через очередь подстановок. Так как оператор отсечения может встретиться только в антиценденте некоторого правила, то флаг отсечения должен генерироваться в процедуре **Backward-AND**. Как только первый атом из списка входных подцелей является оператором отсечения, процедура **Backward-AND** выбрасывает вместо подстановки специальный флаг, после чего продолжает свое выполнение пропустив первый элемент в списке подцелей (оператор отсечения).

Обработать флаг отсечения вместо подстановки может только процедура **Backward-OR**. При получении данного флага в процедуре запрещается дальнейший перебор правил базы знаний.

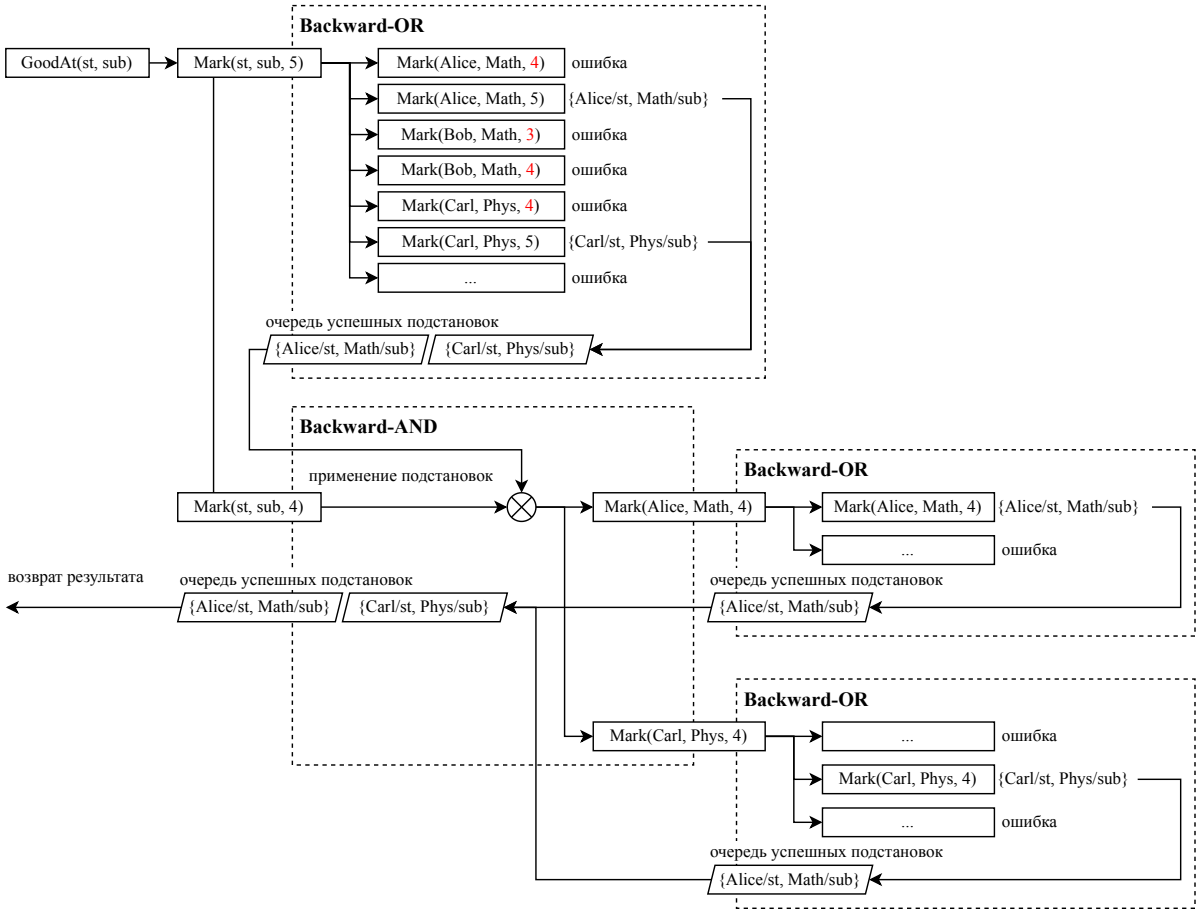

Рисунок 2.2 – Схема перебора всех допустимых подстановок

2.4 Механизм специальных процедур

Иногда значение истинности некоторого предиката не может быть выражено с использованием только лишь определенных выражений, или их количество может быть бесконечно большим. В таком случае можно ввсети таблицу, определяющую соответствие между именем предиката и некоторым объектом, которому будет делегирована задача по доказательству этого предиката. Описанная проверка производится в начале процедуры **Backward-OR**.

В данной работе реализованы следующие специальные процедуры:

1. `write(...)` — выводит значения аргументов в консоль, всегда истиннен.
2. `add(x, y, z)` $\Leftrightarrow x + y = z$. Может вычислить одну переменную.
3. `mul(x, y, z)` $\Leftrightarrow x * y = z$. Может вычислить одну переменную.
4. `leq(x, y)` $\Leftrightarrow x$ и y связаны с целыми числами и $x \leq y$.

5. `in_range(var, start, end)` — если все агрументы связаны с целыми числами, то истинность определяется условием $start \leq var < end$; если var — свободная переменная, а $start$ и end связаны с целыми числами, то генерирует последовательность подстановок $\{i/var\}_{i=start}^{end-1}$.

Используя последний предикат `in_range` можно организовать последовательный перебор чисел. Например, следующая программа выведет в столбик все уникальные пары чисел от 1 до 10 не включительно:

```
nums(x,y) :- in_range(v,x,y), in_range(u,v,y), write(v,u), fail
?- nums(1, 10)
```

2.5 Сборка и запуск ПО

Для сборки программы используется `CMake`. Стандартная последовательность команд для сборки и запуска программы:

```
mkdir build && cd build && cmake ..
make app && ./app # сборка и запуск программы
make unittests && ./unittests # опциональный запуск юнит-тестов
```

Альтернативно, можно собрать и запустить программу с помощью `makefile` скрипта в корневой папке используя цель «`app`».

При запуске основной программы (app) первым аргументом можно указать путь к файлу, содержащему правила базы знаний. По-умолчанию программа работает с пустой базой знаний.

Основная программа работает в интерактивном режиме. Для добавления правила в базу знаний необходимо написать символ «`+`» и текст самого правила. Для запуска алгоритма дедуктивного вывода достаточно написать атом. При этом запустится алгоритм прямого дедуктивного вывода. Для запуска алгоритма обратного дедуктивного вывода необходимо ввести символ «`!`» перед атомом.

Пример работы программы:

```
?- +P(x, y) :- Q(x), R(y)        # добавление правила
>> P(x1, y1) :- Q(x1), R(y1)
?- +R(A)                         # добавление фактов
```

```
>> R(A)
?- +Q(B)
>> Q(B)
?- P(a, b)                          # прямой вывод
{a=B, b=A}                          # результат - подстановка
next?- y                            # запрос продолжения поиска
end                                 # больше решений нет
?- !P(r, s)                         # обратный вывод
{r=B, s=A}
next?-
```

2.6 Тестирование ПО

Для тестирования разработанного ПО были разработаны логические тесты, представленные в таблицах 2.1–2.2.

Таблица 2.1 – Тестовые программы (часть 1)

№	Правила теста
1	Поиск максимума из трех чисел

	`max(a, b, c, a) :- less(b, a), less(c, a)`
	`max(a, b, c, b) :- less(a, b), less(c, b)`
	`max(a, b, c, c) :- less(a, c), less(b, c)`
	`less(1, 2)`
	`less(2, 3)`
	`less(1, 3)`

	Запрос: `max(1, 3, 2, x)` Результат: `{x=3}`
2	Определение длины списка
	`len(Nil, 0) :- !` `len(cons(_, x), succ(n)) :- len(x, n)`
	Запрос: `len(cons(A, cons(B, cons(C, Nil))), x)` Результат: `{x=succ(succ(succ(0)))}`
3	Генерация списка заданной длины
	`len(Nil, 0) :- !` `len(cons(_, x), succ(n)) :- len(x, n)`
	Запрос: `len(x, succ(succ(succ(0))))` Результат: `{x=cons(_, cons(_, cons(_, Nil)))}`

Таблица 2.2 – Тестовые программы (часть 2)

№	**Правила теста**
4	Реверс списка

	`reverse(Nil, Nil) :- !`
	`reverse(x, y) :- reverse_helper(x, y, Nil)`
	`reverse_helper(Nil, x, x)`
	`reverse_helper(cons(h, r), x, y) :-`
	` reverse_helper(r, x, cons(h, y))`

	Запрос: `reverse(cons(A, cons(B, cons(C, Nil))), x)` Результат: `{x=cons(C, cons(B, cons(A, Nil)))}`
5	Лабиринт

	`# +---+---+---+`
	`# | 1 |X2X|X3X|`
	`# +---+---+---+`
	`# | 4 | 5 | 6 |`
	`# +---+---+---+`
	`# | 7 |X8X| 9 |`
	`# +---+---+---+`
	`step(1, 4)`
	`step(4, 5)`
	`step(4, 7)`
	`step(5, 6)`
	`step(6, 9)`
	`way(x, y) :- step(x, y)`
	`way(x, y) :- step(x, z), way(z, y)`

	Запрос: `way(1, w)` Результат: `{w=4}, {w=5}, {w=7}, {w=6}, {w=9}`
6	Различность
	`neq(x, x) :- !, fail` `neq(_, _)`
	Запрос: `neq(A, A)` Результат: `no`
	Запрос: `neq(A, B)` Результат: `{}`

В таблице 2.3 представлены тесты, включающие в себя работу со специальными процедурами, описанными в разделе 2.4.

Таблица 2.3 – Тестовые программы (часть 3)

№	**Правила теста**
7	Отображение пути вывода

	`# leq(x, y)` $\Leftrightarrow x \leq y$ — спец. процедура
	`# write(...)` — спец. процедура
	`min(a, b, a) :- write(First, Route), leq(a, b), !`
	`min(_, b, b) :- write(Second, Route)`

	Запрос: `min(14, 50, x)` Результат: `{x=14}` Побочный эффект (вывод на экран): `First Route`

	Запрос: `min(3, 2, x)`
	Результат: `{x=2}`
	Побочный эффект (вывод на экран):
	`First Route`
	`Second Route`

8	Определение простоты числа

	`# mul(x, y, z)` $\Leftrightarrow x * y = z$ — спец. процедура
	`# in_range(a, b, c)` — спец. процедура
	`divisible(x, y) :- mul(z, y, x), mul(z, y, x), !`
	`composite(x) :- in_range(v, 2, x), divisible(x, v), !`
	`prime(1) :- !, fail`
	`prime(x) :- composite(x), !, fail`
	`prime(_)`

	Запрос: `prime(5)` Результат: `{}`
	Запрос: `prime(12)` Результат: `no`

Выводы

В ходе лабораторной работы была разработана программа реализующая алгоритм обратного дедуктивного вывода с возможностью полного перебора допустимых решений. Добавлена поддержка оператора отсечения, а также реализован механизм вызова специальных процедур.

В качестве направлений для дальнейшего развития алгоритма и программы можно выделить следующие:

— анализ зависимостей по переменным и параллелизация доказательства независимых предикатов;

— реализация вычислимых функциональных символов;

— добавление специальных процедур для работы с вводом-выводом;

— добавление CLP-решателя для поиска значений числовых переменных симплекс-методом.