БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра интеллектуальных информационных технологий

**Отчёт**

по лабораторной работе №2

по дисциплине

«Логические основы интеллектуальных систем»

Выполнил:

студент группы 121702.

Шакин И.В.

Проверил:

Ивашенко В.П.

Минск 2023

**Вариант 9:** запрограммировать обратный нечеткий логический вывод на основе операции нечеткой композиции (min({1}∪{max({0}∪{xi+yi-1}|i}).

**Тема:** реализовать процедуру неклассического логического вывода и преобразования логических формул в базах знаний и онтологиях.

**Цель:** приобрести навыки разработки и программирования прикладных систем логического вывода, включая системы неклассического логического вывода.

**Состав бригады:** Шакин И.В., Промчук Д.В., Шершень К.А.

**Задачи:** разработать алгоритм обратного нечеткого логического вывода на основе заданной операции нечеткой композиции и программно реализовать разработанный алгоритм.

**Входные данные:** строка, содержащая следствие, вида ‘b1: 0.5, b2: 0.6, b3: 0.2’ и строка, содержащая отношение, вида ‘[[0.4, 0.9], [1, 0.5], [0.4, 0.9]]’.

**Выходные данные:** если система неравенств имеет решение, формат выходных данных представляет собой список решений, содержащих интервалы значений каждой переменной, если система неравенств не имеет решений, None - выходные данные.

**Теоретические сведения:**

***Задача обратного нечёткого логического вывода* -** задача, обратная задаче прямого нечёткого логического вывода. В качестве исходных данных данной задачи выступают два нечётких предиката - правило и заключение. Требуется найти множество посылок, которые могут при применении данного правила привести к указанному заключению [1].

***Задача прямого нечёткого логического вывода*** подразумевает известность некоторой пары нечётких предикатов, один из которых рассматривается как посылка, а второй как правило; обычно первый предикат является унарным, а второй бинарным. Тогда задача прямого нечёткого логического вывода сводится к нахождению композиции между двумя этими нечёткими предикатами [1].

***Нечёткое множество* определяется как (), где [0; 1] –** множество чисел от числа 0 до числа 1, элементы которого являются значениями степени нечёткой принадлежности, а σ – произвольное множество [1].

***Нечеткий предикат*** – нечёткое множество, значения которого интерпретируются как значения истинности [1].

***Т-норма (треугольная норма)* -** функция двух переменных **T: [0;1]×[0;1] → [0;1]** (т.е., бинарная операция на [0;1]), удовлетворяющая следующим условиям**:**

1. **T(x, 0) = T(0, x) = 0, T(x, 1) = T(1, x) = x;**
2. **T** не убывает в любой точке, то есть, **T(x1, y1) ≤ T(x2, y2),** когда **x1 ≤ x2, y1 ≤ y2;**
3. **T** коммутативна, т.е. для всех **x, y** из **[0;1], T(x, y) = T(y, x);**
4. **T** ассоциативна, т.е. для всех **x, y, z** из **[0;1] T(T(x, y), z) = T(x, T(y, z)) [2].**

**Описание алгоритма:**

**def read\_file(count)** - чтение файла по указанному индексу count. Ключевым условием функции является проверка (if len(lines) >= count\*3+2 and count>=0), проверяющая достаточное количество строк в файле для извлечения содержимого;

**def write\_file(consequence, relation, result)** - запись результата нечеткого логического вывода в файл “output”;

**def all\_combinations(\*arrays) -** принимает на вход произвольное количество списков, используя инструмент “product” из из модуля "itertools" для нахождения всех возможных комбинаций элементов из этих массивов. После этого она возвращает список, содержащий все эти комбинации;

**def consequence\_to\_variables(consequence) -** принимает принимает строку "consequence" в качестве аргумента и разбивает её на подстроки, используя запятые в качестве разделителей. Затем она проходит по каждой подстроке, разделяет её на две части по символу ":", преобразует вторую часть (после ":", которая предполагается числом) во float и создает кортеж из двух частей. В результате она возвращает список таких кортежей, представляющих различные переменные и их значения из входной строки "consequence";

**def check\_interval(x)** - принимает значение x в качестве аргумента и возвращает булево значение, указывающее, находится ли значение x в интервале от 0 до 1 включительно. Если x находится в этом интервале, функция возвращает True, в противном случае - False;

**def get\_unique(lists)** - принимает список списков в качестве аргумента. Затем она преобразует каждый вложенный список в кортежи и затем создает множество уникальных кортежей для каждого вложенного списка. После этого она преобразует каждый уникальный кортеж обратно в список и возвращает список уникальных списков. Таким образом, функция удаляет дубликаты списков из входного списка;

**def include\_each\_other(i, point)** - принимает два списка интервалов "i" и "point" в качестве аргументов. Затем она сравнивает каждый интервал из "i" с каждым интервалом из "point" и добавляет флаги в список "flags" в зависимости от результата сравнения. Как только установлены все флаги, функция выполняет различные проверки на наличие определенных флагов и возвращает соответствующее сообщение в зависимости от условий. Если обнаружены определенные сочетания флагов, функция возвращает определенную строку;

**def check\_inclusion(res, point)** - принимает список интервалов "res" и новый интервал "point" в качестве аргументов. Затем она проходит по каждому интервалу в списке "res" и вызывает функцию "include\_each\_other" для сравнения каждого интервала с "point". В зависимости от результата сравнения, функция выполняет определенные действия: если статус равен "Не добавлять", функция возвращает исходный список "res"; если статус равен "Заменить", функция заменяет соответствующий интервал в "res" на "point" и возвращает "res"; в противном случае функция добавляет "point" к "res" и возвращает "res";

**def get\_merged\_answer(answers)** - принимает список ответов "answers" в качестве аргумента. Затем она создает все возможные комбинации ответов из входного списка, затем проходит по каждой комбинации и вычисляет пересечения интервалов в каждой комбинации. Если найдены допустимые пересечения, то результат сравнивается с существующими интервалами (если они уже есть) и обновляется в соответствии с определенными условиями, используя функцию "check\_inclusion". После выполнения всех итераций функция возвращает сформированную строку результата с помощью функции "form\_result\_string", если результат не пустой, в противном случае возвращает "None";

**def form\_result\_string(result)** - принимает список результатов "result" в качестве аргумента. Затем она проходит по каждому элементу в списке результатов и формирует строковое представление каждого интервала. Строковое представление каждого интервала сохраняется в новом списке "result\_string". Если интервал представляет собой точку (т.е. левая и правая границы равны), то формируется строка вида "ai=value", в противном случае формируется строка вида "ai=[left\_bound;right\_bound]". Затем функция возвращает список строк, представляющих интервалы из входного списка "result";

**reverse\_fuzzy\_conclusion(consequence, relation\_matrix)** - основная функция, в которой заключен алгоритм решения задачи. Проверяется степень принадлежности каждого элемента вывода и на ее основании выбирается необходимый алгоритм дальнейших действий.   
Если степень принадлежности элемента вывода(y) равна единице, степень принадлежности соответствующего элемента матрицы(r) принадлежит интервалу [0,1], то степень принадлежности соответствующего элемента ответа принадлежит интервалу [2-r,1].

Если степень принадлежности элемента вывода(у) равна нулю, степень принадлежности соответствующего элемента матрицы(r) принадлежит интервалу [0,1], то степень принадлежности соответствующего элемента ответа принадлежит интервалу [0,1-r] либо является точкой <1,1>.

Если степень принадлежности элемента вывода(у) принадлежит интервалу (0, 1), степень принадлежности соответствующего элемента матрицы(r) принадлежит интервалу [0,1], то степень принадлежности соответствующего элемента ответа принадлежит интервалу [y+1-r,1] либо является точкой <y+1-r,y+1-r>

**Реализация:**

**Используемые сторонние функции и библиотеки:** в работе использовалась функция из библиотеки itertools – product(a, repeat = b) , которая возвращает список всевозможных комбинаций элементов из массива a длиной b; также в работе используется функция из библиотеки json - loads(string), которая используется для преобразования строки в формате JSON в объект Python.

**Тестирование и результат работы**

**Тест №1**

****

Рисунок 8 - Входные данные

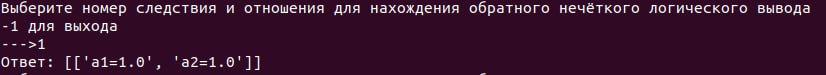


Рисунок 9 – Результат выполнения

**Тест №2**

****

Рисунок 10 - Входные данные

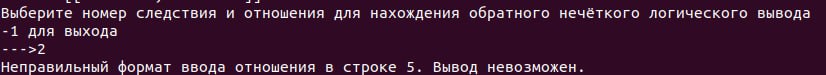


Рисунок 11 – Результат выполнения

**Тест №3**



Рисунок 12 - Входные данные

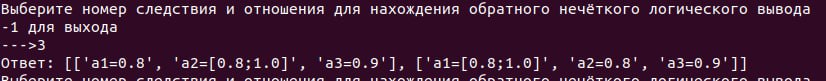


Рисунок 13 – Результат выполнения

**Тест №4**

****

Рисунок 14 - Входные данные

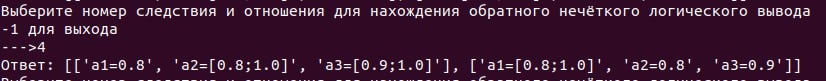


Рисунок 15 – Результат выполнения

**Тест №5**

****

Рисунок 8 - Входные данные

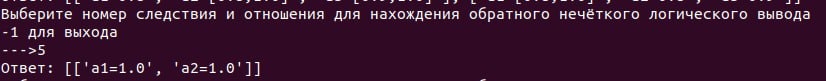


Рисунок 9 – Результат выполнения

**Личный вклад:** реализация функций, выполняющих сравнивание интервалов, функции реализующей преобразование преобразование вложенных списков в кортежи и последующее создание множества уникальных кортежей для каждого вложенного списка, реализация функций сравнения списков и формирования списка флагов для выбора определенной строки.

**Вывод:** в ходе лабораторной работы приобрели навыки разработки и программирования прикладных систем логического вывода, включая системы неклассического логического вывода.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Учебное пособие «Логические основы интеллектуальных систем. Практикум» – В. В. Голенков, В. П. Ивашенко, Д. Г. Колб, К. А. Уваров, Минск БГУИР, 2011, 70 стр. – Режим доступа: <https://libeldoc.bsuir.by/handle/123456789/992>