**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования** **«Новгородский государственный университет имени Ярослава Мудрого»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра ИТиС

«Продукционная модель представления знаний»

Лабораторная работа №2

По дисциплине: Системы искусственного интеллекта

Отчет

Выполнил: студент гр. 8091

\_\_\_\_\_\_\_ \_\_\_\_Шаклеин В.В.

Проверил: Преподаватель

\_\_\_\_\_\_\_\_\_\_\_Михайлов Д.В.

Великий Новгород

2021

1. **Формулировка цели и задач**

Целью данной работы является приобретение студентами умений и навыков реализации пополняемой динамической базы знаний, не включаемой непосредственно в текст программы.

Необходимые для достижения поставленной цели задачи состоят в следующем:

· научиться оценивать возможности применения современных языков высокого уровня для реализации баз знаний экспертных систем (ЭС);

· изучение технических аспектов реализации продукционной модели представления знаний.

1. **Конфигурация системы продукций**

Предметная область для ЭС – Компьютерные вирусы

Компьютерные вирусы – вирусы, разбиты на группы, в зависимости от предназначения

Таблица 1. Описание предметной области

|  |  |  |
| --- | --- | --- |
| **Группа** | **Название** | **Характеристики** |
| **Вредительство** | **Morris Worm** | Denail\_of\_service |
| **ILOVEYOU** | Self\_reproduction  Data\_corruption |
| **SQL Slammer** | Self\_reproduction  Auto\_infection  Denail\_of\_service |
| **Кража данных** | **Stuxnet** | Self\_reproduction  Denail\_of\_service  Targeted |
| **CryptoLocker** | Self\_reproduction  Data\_encryption |
| **Conficker** | Self\_reproduction  Auto\_infection |
| **Получение выкупа** | **Tinba** | Data\_sniffing |
| **Nimda** | Self\_reproduction  Data\_corruption  Rootkit |

Для описания объекта или его отдельных свойств выбираются некоторые характеристики – величины, которые могут принимать либо количественные, либо качественные значения.

В свою очередь, совокупность всех характеристик некоторого объекта образует так называемый список разрешенных характеристик данного объекта. Списки разрешенных характеристик и разрешенных значений этих характеристик охватывают множество всех имеющихся фактов, подлежащих хранению в базе знаний экспертной системы.

В экспертных системах правила, по которым решаются проблемы в конкретной предметной области, хранятся в базе знаний. *Проблемная область ЭС* описывается посредством фактов и правил. Проблемы ставятся перед системой в виде совокупности фактов, описывающих некоторую ситуацию, и система с помощью базы знаний пытается вывести заключение из этих фактов.

Для данной лабораторной работы в качестве базы знаний был выбран текстовый файл.

В начале файла описываются все атрибуты и их значения. 1 атрибут на 1 строке.

Пример атрибутов:

Denial\_of\_service\* = y | n

Self\_reproduction\* = y | n

Data\_corruption\* = y | n

Auto\_infection\* = y | n

Targeted\* = y | n

Data\_encryption\* = y | n

Data\_sniffing\* = y | n

Rootkit\* = y | n

Virus\_class\* = DOS | Data\_Stealing | Ransom

Также в базе знаний хранятся правила, синтаксис которых:

IF A THEN S, где А – условие; S – заключение.

Пример правил:

IF Virus\_class = DOS & Denial\_of\_service = y & Self\_reproduction = n & Data\_corruption = n & Auto\_infection = n & Targeted = n & Data\_encryption = n & Data\_sniffing = n & Rootkit = n THEN Virus = Morris\_Worm

1. **Описание способа организации поиска на графе**

Поиск на графе организован с помощью последовательного отброса заведомо ложных путей и проверки доступных с помощью перебора.

1. **Описание конфликтного набора и алгоритм разрешения конфликта при логическом выводе**

В данной лабораторной работе был выбран обратный порядок вывода. В системах с обратным выводом вначале выдвигается гипотеза, а затем механизм вывода в процессе работы как бы возвращается назад, переходя от выдвинутой гипотезе к фактам и пытаясь найти среди них те, которые подтверждают эту гипотезу.

Конфликтный набор:

Система предлагает ответить на вопрос, ответами являются такие символы как y и n.

Чтобы вызвать конфликтный набор, можно на эти вопросы ответить, как приведено ниже:

DOS->y->y->y

Ответ системы: The knowledge base does not contain information about this phone

Механизм решения конфликта представляет собой алгоритм, который при возможности применения сразу нескольких правил запрашивает уточнение одного из параметров.

Когда пользователь опровергает определенный факт, т.е. отвечает отрицательно, алгоритм прерывает проверку текущего правила и переходит к следующему по списку в базе знаний.

1. **Тестовый набор правил**

DOS->y->n->n->n->n->n->n->n

Ответ системы: Morris\_Worm

Ransom->n->y->y->n->n->n->n->y

Ответ системы: Nimda

1. **Вывод**

Мною была изучена реализация пополняемой динамической базы знаний, не включаемой непосредственно в текст программы.

Также я изучил реализацию продукционной модели представления знаний.

Также была построена база знаний, предметной области, мобильные устройства.

1. **Текст программы**

Файл Eparser.py

import copy

def parse(path):

parameters = {}

rules = []

try:

with open(path, 'r') as f:

lines = f.readlines()

except:

print("Problem with open file " + path)

return

for line in lines:

if not line or line.startswith('-') or line.startswith('#'):

continue

elif line.startswith('IF'):

current = {}

sides = line.replace('IF', '').split('THEN')

current['LHS'] = {}

conditions = sides[0].split('&')

for condition in conditions:

HS = list(map(str.strip, condition.split('=', 1)))

current['LHS'][HS[0]] = [s.strip() for s in HS[1].split('|')]

action = sides[1]

HS = list(map(str.strip, action.split('=')))

current['RHS'] = {HS[0]: HS[1]}

rules.append(copy.deepcopy(current))

else:

splitLine = line.split('=', 1)

parameters[splitLine[0].strip()] = [s.strip()

for s in splitLine[1].split('|')]

return parameters, rules

Файл Shell.py

import Eparser

# Функция возвращает строку с правилом в удобном формате: ЕСЛИ условие, ТО действие

def rule\_repr(rule):

LHS = []

for attr, values in rule['LHS'].items():

LHS.append(attr + " = " + "|".join(values))

(RHSkey, RHSvalue) = list(rule['RHS'].items())[0]

return "IF " + " & ".join(LHS) + " THEN " + RHSkey + " = " + RHSvalue

# Функция печатает рабочую память

def printRM():

print("Memory:")

for r, v in RM.items():

print(r, " = ", v)

# Func returns a list of rules, pravaya ch kotorih = znach parametra

def getConflictRules(rules, goal):

ruleset = []

for rule in rules:

attribute = list(rule['RHS'].keys())[0]

if attribute == goal:

ruleset.append(rule)

return ruleset

# The function checks if there is at least one rule, the right side of which is the goal value.

def conflictRuleExists(rules, goal):

for rule in rules:

attribute = list(rule['RHS'].keys())[0]

if attribute == goal:

return True

return False

# The function checks if this rule works.

def ruleWorks(rule, RM):

conditions = rule['LHS']

for param in conditions:

if param in RM:

if RM[param] not in conditions[param]:

return False

else:

return False

return True

# The function for the user to enter the default parameter in the working memory

def parameterInput(param, RM):

value = input("Enter parameter value '" + param +

"' " + str(parameters[param+"\*"]) + ": ")

while(value not in parameters[param+"\*"]):

value = input()

RM[param] = value

# Функция печатает атрибуты / параметры, их значения и все правила, содержащиеся в БЗ

def printKnowledgeBase(parameters, rules):

print('-'\*105)

print('|' + '\t'\*6 + 'Knowledge base' + '\t'\*6 + '|')

print('-'\*105 + '\n')

print("Attributes:")

for attr, value in parameters.items():

print(attr + " = " + " | ".join(value))

print("\nRules:")

for i, rule in enumerate(rules):

print(str(i+1) + ") " + rule\_repr(rule))

print('-'\*105 + '\n')

# Получить атрибуты и правила из базы знаний

parameters, rules = Eparser.parse('./Virus.txt')

# Print KB

printKnowledgeBase(parameters, rules)

# Working memory, stack with targets and a list of already verified attributes

RM = {}

goals = []

checked\_goals = []

# Asks user to enter target hypothesis

goal = input('Please, enter "Virus": ')

goals.append(goal)

# Main loop

while(True):

# # If stack is empty, quit

if len(goals) == 0:

break

# Vspomog variab

new\_goal = False

new\_parameter = False

# Save current target

goal = goals[-1]

# Create a set of conflicting rules and keep them count

conflictRules = getConflictRules(rules, goal)

remainingRules = len(conflictRules)

# If no matching rule is found, stop the loop and notify the user.

if remainingRules == 0:

print('The knowledge base does not contain information about this virus.')

break

# Prints a set of rules and working memory.

print('Rules: ')

for cr in conflictRules:

print(rule\_repr(cr))

printRM()

# The cycle goes through a set of conflicting rules.

# If the rule works:

# 1) delete the current target from above

# 2) keeps its right side in working memory

# 3) set the variable new\_goal to True and exit the loop

for cr in conflictRules:

if ruleWorks(cr, RM):

(RHSkey, RHSvalue) = list(cr['RHS'].items())[0]

RM[RHSkey] = RHSvalue

curr\_goal = goals.pop()

print("Goal: " + curr\_goal + " = " + RHSvalue)

new\_goal = True

break

# If the target is updated

if new\_goal:

continue

# for each rule in the conflict rule set

for cr in conflictRules:

if new\_goal:

break

# уменьшить количество непроверенных правил

remainingRules -= 1

conditions = cr['LHS']

# for each parameter of checked rules

for param in conditions:

# if the parameter has already been checked and it was not possible to execute it,

# skip rule (do not check other parameters)

if param in checked\_goals:

break

# if current parameter is in working memory

if param in RM:

# parameter value does not match the value in working memory

if RM[param] not in conditions[param]:

break

else: # parameter is not in memory

# if one of the rules fulfills the current parameter, set it as a target

if conflictRuleExists(rules, param):

goals.append(param)

new\_goal = True

break

elif param + "\*" in parameters:

# None of the rules executes the parameter

# if possible (the parameter name ends with '\*'),

# asks user to enter parameter value

parameterInput(param, RM)

new\_parameter = True

break

else:

# parameter cannot be obtained from any rule

# user cannot login

checked\_goals.append(param)

if new\_parameter:

break

# If a new target has not been established and all the rules have been verified, delete the target from above.

# and save it to the list of goals that can not be achieved

if remainingRules == 0 and not new\_goal:

curr\_goal = goals.pop()

checked\_goals.append(curr\_goal)

# print ('The knowledge base does not contain information about this virus: ' + curr\_goal)

print('The knowledge base does not contain information about this virus: ')

print('-'\*100)

Файл Virus.txt

Denial\_of\_service\* = y | n

Self\_reproduction\* = y | n

Data\_corruption\* = y | n

Auto\_infection\* = y | n

Targeted\* = y | n

Data\_encryption\* = y | n

Data\_sniffing\* = y | n

Rootkit\* = y | n

Virus\_class\* = DOS | Data\_Stealing | Ransom

IF Virus\_class = DOS & Denial\_of\_service = y & Self\_reproduction = n & Data\_corruption = n & Auto\_infection = n & Targeted = n & Data\_encryption = n & Data\_sniffing = n & Rootkit = n THEN Virus = Morris\_Worm

IF Virus\_class = DOS & Denial\_of\_service = n & Self\_reproduction = y & Data\_corruption = y & Auto\_infection = n & Targeted = n & Data\_encryption = n & Data\_sniffing = n & Rootkit = n THEN Virus = ILOVEYOU

IF Virus\_class = DOS & Denial\_of\_service = y & Self\_reproduction = y & Data\_corruption = n & Auto\_infection = y & Targeted = n & Data\_encryption = n & Data\_sniffing = n & Rootkit = n THEN Virus = SQL\_Slammer

IF Virus\_class = Data\_Stealing & Denial\_of\_service = y & Self\_reproduction = y & Data\_corruption = n & Auto\_infection = n & Targeted = y & Data\_encryption = n & Data\_sniffing = n & Rootkit = n THEN Virus = Stuxnet

IF Virus\_class = Data\_Stealing & Denial\_of\_service = n & Self\_reproduction = y & Data\_corruption = n & Auto\_infection = n & Targeted = n & Data\_encryption = y & Data\_sniffing = n & Rootkit = n THEN Virus = CryptoLocker

IF Virus\_class = Data\_Stealing & Denial\_of\_service = n & Self\_reproduction = y & Data\_corruption = n & Auto\_infection = y & Targeted = n & Data\_encryption = n & Data\_sniffing = n & Rootkit = n THEN Virus = Conficker

IF Virus\_class = Ransom & Denial\_of\_service = n & Self\_reproduction = n & Data\_corruption = n & Auto\_infection = n & Targeted = n & Data\_encryption = n & Data\_sniffing = y & Rootkit = n THEN Virus = Tinba

IF Virus\_class = Ransom & Denial\_of\_service = n & Self\_reproduction = y & Data\_corruption = y & Auto\_infection = n & Targeted = n & Data\_encryption = n & Data\_sniffing = n & Rootkit = y THEN Virus = Nimda