Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.03.02 «Информационные системы и технологии»

Программа: «Информационные системы и технологии»

**ЛАБОРАТОРНАЯ РАБОТА**

по дисциплине

**«Интеллектуальные информационные системы и технологии»**

Тема: **«Проектирование и реализация оболочки экспертной системы, использующей продукционный способ представления знаний»**

Выполнил:

студент гр. ИСТ-19-1бзу

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

Неганов Д.А.

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

(Ф.И.О.)

Принял:

Старший преподаватель Истомин Д.А.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(должность, ФИО руководителя)*

**Пермь 2022**

**Цель:**

Сформировать у студентов способность построения учебного прототипа экспертной системы, основанной на знаниях, включающего основные компоненты ИИС.

**Описание:**

В рамках данной работы необходимо реализовать оболочку ЭС, которая была бы способна работать с продукционной БЗ. Работа позволит углубленно познакомиться со способами взаимодействия компонентов ЭС, а также реализовать отдельные функциональные блоки:

1. База знаний.

2. Механизм логического вывода.

3. Компонента объяснения

**Выполнение работы:**

Основные компоненты экспертной системы:

— Механизм логического вывода (*InferenceComponent*)

Отвечает за вывод результата. Алгоритм вывода выбирает одно из правил, готовых к активации (условия которых выполнены), добавляет правую часть этого правила к списку действующих фактов (при необходимости – задает вопрос пользователю), если один из добавленных фактов является результатом – возвращает его.

— Рабочая память (*MemoryComponent*)

В рабочей памяти хранятся: список правил БЗ, действующих фактов и активированных правил, и правил, которые готовы к активации.

Вспомогательные компоненты:

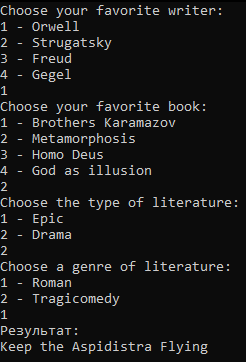
— Взаимодействие с пользователем (*UserInput*)

Определяет тип задаваемого вопроса, принимает введенное пользователем значение и валидирует его, а при необходимости задает вопрос повторно, возвращает полученное значение.

— JSON-конвертер (*ConditionConverter*)

Восстановление сериализованных классов-наследников от абстрактного класса Condition.

**Пример работы программы**



**Код программы**

using System;

using System.Collections.Generic;

using System.Linq;

namespace infsystem

{

enum ValueType

{

String, Bool, Number,

}

enum ExpressionType

{

And, Or,

}

enum ComparisonType

{

Equals, NotEquals, LessThan, GreaterThan, Defined, NotDefined, GreaterOrEquals, LessOrEquals,

}

struct Rule

{

public Condition Condition;

public Fact[] Assertions;

public int Salience;

public override string ToString() => $"([{Salience}] {Condition} ==> {string.Join<Fact>("; ",Assertions)})";

}

// Значение факта или значение, с которым сравнивается факт

class Value

{

public string? StringValue;

public double? NumberValue;

public ValueType Type

{

get

{

if (StringValue != null) return ValueType.String;

if (NumberValue != null) return ValueType.Number;

throw new Exception("Value is not defined");

}

}

public override string ToString()

{

if (StringValue != null) return StringValue;

if (NumberValue != null) return NumberValue.ToString();

throw new Exception("Value is not defined");

}

}

// Условие может быть либо простым условием сравнения факта со значением,

// либо конъюнкцией/дизъюнкцией набора других условий

abstract class Condition

{

// Выполняется ли условие при наличии данного набора фактов

// В том числе возвращает список конкретных фактов, которых достаточно для выполнения условия

public abstract (bool CanActivate, List<Fact> Requirements) Evaluate(IEnumerable<Fact> facts);

// Количество элементарных условий-сравнений, из которых состоит данное условие

// Используется для разрешения конфликтов

public abstract int Specificity { get; }

}

class ConditionExpression : Condition

{

public string Operator;

public Condition[] Conditions;

public ExpressionType Type

{

get

{

switch (Operator)

{

case "AND":

case "And":

case "and":

case "&&":

case "&":

return ExpressionType.And;

case "OR":

case "Or":

case "or":

case "||":

case "|":

return ExpressionType.Or;

default:

throw new FormatException($"Operator ({Operator}) is not supported. Supported operators: AND, OR, or equivalent.");

}

}

}

public override (bool CanActivate, List<Fact> Requirements) Evaluate(IEnumerable<Fact> facts)

{

var results = Conditions.Select(condition => condition.Evaluate(facts));

if (Type == ExpressionType.And)

{

var requirements = results

.SelectMany(result => result.Requirements)

.ToList();

return (results.All(result => result.CanActivate), requirements);

}

foreach (var result in results)

{

if (result.CanActivate) return result;

}

return (false, new());

}

public override int Specificity => Conditions.Select(condition => condition.Specificity).Sum();

public override string ToString() => $"({String.Join<Condition>(Type == ExpressionType.And ? " && " : " || ", Conditions)})";

}

// Проверка, определен или не определен факт с заданным именем, или сравнение факта с заданным значением

class ValueCondition : Condition

{

public string FactName;

public string Comparison;

public Value? Value;

public ComparisonType? Type =>

Comparison switch

{

"<" when Value is {Type: ValueType.Number} => ComparisonType.LessThan,

">" when Value is {Type: ValueType.Number} => ComparisonType.GreaterThan,

">=" when Value is {Type: ValueType.Number} => ComparisonType.GreaterOrEquals,

"<=" when Value is {Type: ValueType.Number} => ComparisonType.LessOrEquals,

"=" => ComparisonType.Equals,

"==" => ComparisonType.Equals,

"!=" => ComparisonType.NotEquals,

"<>" => ComparisonType.NotEquals,

"!" when Value is null => ComparisonType.Defined,

"?" when Value is null => ComparisonType.NotDefined,

\_ => throw new FormatException($"Comparison ({Comparison}) is invalid for given value ({Value}) or not supported")

};

public override (bool CanActivate, List<Fact> Requirements) Evaluate(IEnumerable<Fact> facts)

{

Fact? foundFact = facts.FirstOrDefault(fact => fact.Name == FactName);

if (foundFact is null)

{

// Если факт с заданным именем не найден и условие - "Факт не определен", то условие выполняется

return (Type == ComparisonType.NotDefined, new() );

}

else if (Type == ComparisonType.Defined)

{

// Если факт с заданным именем найден и условие - "Факт определен", то условие выполняется

return (true, new() { foundFact } );

}

var value = foundFact.Value;

if (Value is null) return (true, new() { foundFact } );

if (foundFact.Value.Type != Value.Type) throw new ArgumentException($"Fact ({value}) and condition ({Value}) types are incompatible");

var type = value.Type;

// Проверка, удовлетворяет ли найденный факт сравнению, описанному в данном условии

switch (Type)

{

case ComparisonType.Equals when value.ToString() == Value.ToString():

case ComparisonType.NotEquals when value.ToString() != Value.ToString():

case ComparisonType.LessThan when type == ValueType.Number && value.NumberValue < Value.NumberValue:

case ComparisonType.GreaterThan when type == ValueType.Number && value.NumberValue > Value.NumberValue:

case ComparisonType.LessOrEquals when type == ValueType.Number && value.NumberValue <= Value.NumberValue:

case ComparisonType.GreaterOrEquals when type == ValueType.Number && value.NumberValue >= Value.NumberValue:

return (true, new() { foundFact } );

case ComparisonType.Equals:

case ComparisonType.NotEquals:

case ComparisonType.LessThan when type == ValueType.Number:

case ComparisonType.GreaterThan when type == ValueType.Number:

case ComparisonType.LessOrEquals when type == ValueType.Number:

case ComparisonType.GreaterOrEquals when type == ValueType.Number:

return (false, new() { foundFact } );

default:

throw new ArgumentException($"Fact ({value}) and condition ({Value}) values cannot be compared");

}

}

public override int Specificity => 1;

public override string ToString() => $"'{FactName}' {Comparison} {Value}";

}

// Факт может иметь определенное значение, его значение может определяться вопросом к пользователю,

// либо (при отсутствии значения и вопроса) факт является искомым результатом

class Fact

{

public string Name;

public Value? Value;

public Input? Input;

public bool IsResult => Value == null && Input == null;

public override string ToString()

{

string value = $"'{Name}'";

if (Input != null) value += $" ?? {Input}";

if (Value != null) value += $" == {Value}";

return value;

}

}

// Вопрос к пользователю. Это может быть вопрос с выбором одного из предложенных значений,

struct Input

{

public string Question;

public string[]? Values;

public ValueType Type

{

get

{

if (Values != null) return ValueType.String;

return ValueType.Bool;

}

}

public override string ToString()

{

string value = $"'{Question}'";

if (Values != null) value += $" ({string.Join("/", Values)})";

return value;

}

}

}

using System;

using System.Text.Json;

using System.Text.Json.Serialization;

namespace infsystem

{

// JSON-конвертер условий. Нужен для правильной полиморфической десериализации

// классов-наследников от абстрактного класса Condition

internal class ConditionConverter : JsonConverter<Condition>

{

public override bool CanConvert(Type typeToConvert) => typeof(Condition).IsAssignableFrom(typeToConvert);

public override Condition Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)

{

if (reader.TokenType != JsonTokenType.StartObject)

{

throw new JsonException($"Cannot convert JSON to type {typeToConvert}: invalid JSON structure");

}

reader.Read();

if (reader.TokenType != JsonTokenType.PropertyName)

{

throw new JsonException($"Cannot convert JSON to type {typeToConvert}: invalid JSON structure");

}

var propertyName = reader.GetString();

Condition condition;

switch (propertyName)

{

case nameof(ConditionExpression.Operator):

case nameof(ConditionExpression.Conditions):

condition = new ConditionExpression();

break;

case nameof(ValueCondition.FactName):

case nameof(ValueCondition.Comparison):

case nameof(ValueCondition.Value):

condition = new ValueCondition();

break;

default:

throw new JsonException($"Cannot convert JSON to type {typeToConvert}: property name {propertyName} not recognized");

}

do

{

if (reader.TokenType == JsonTokenType.EndObject)

{

return condition;

}

if (reader.TokenType == JsonTokenType.PropertyName)

{

propertyName = reader.GetString();

reader.Read();

switch (propertyName)

{

case nameof(ConditionExpression.Operator)

when condition is ConditionExpression conditionExpression:

var operatorString = reader.GetString();

if (operatorString is null) throw new JsonException($"Cannot convert JSON to type {typeToConvert}: property {propertyName} is not nullable");

conditionExpression.Operator = operatorString;

break;

case nameof(ConditionExpression.Conditions)

when condition is ConditionExpression conditionExpression:

var conditions = JsonSerializer.Deserialize<Condition[]>(ref reader, options);

if (conditions is null) throw new JsonException($"Cannot convert JSON to type {typeToConvert}: property {propertyName} is not nullable");

conditionExpression.Conditions = conditions;

break;

case nameof(ValueCondition.FactName) when condition is ValueCondition valueCondition:

var factName = reader.GetString();

if (factName is null) throw new JsonException($"Cannot convert JSON to type {typeToConvert}: property {propertyName} is not nullable");

valueCondition.FactName = factName;

break;

case nameof(ValueCondition.Comparison) when condition is ValueCondition valueCondition:

var comparison = reader.GetString();

valueCondition.Comparison = comparison;

break;

case nameof(ValueCondition.Value) when condition is ValueCondition valueCondition:

var value = JsonSerializer.Deserialize<Value>(ref reader, options);

valueCondition.Value = value;

break;

default:

throw new JsonException($"Cannot convert JSON to type {typeToConvert}: property name {propertyName} not recognized");

}

}

} while (reader.Read());

throw new JsonException($"Cannot convert JSON to type {typeToConvert}: invalid JSON structure");

}

public override void Write(Utf8JsonWriter writer, Condition condition, JsonSerializerOptions options)

{

writer.WriteStartObject();

switch (condition)

{

case ConditionExpression expression:

writer.WriteString(nameof(expression.Operator), expression.Operator);

writer.WritePropertyName(nameof(expression.Conditions));

JsonSerializer.Serialize(writer, expression.Conditions, options);

break;

case ValueCondition value:

writer.WriteString(nameof(value.FactName), value.FactName);

if (value.Comparison is not null) writer.WriteString(nameof(value.Comparison), value.Comparison);

if (value.Value is not null)

{

writer.WritePropertyName(nameof(value.Value));

JsonSerializer.Serialize(writer, value.Value, options);

}

break;

default:

throw new JsonException($"Unknown condition type {condition.GetType()}");

}

writer.WriteEndObject();

}

}

}

using System;

using System.IO;

using System.Text.Json;

namespace infsystem

{

static class ExpertSystem

{

static void Main(string[] args)

{

var options = new JsonSerializerOptions

{

WriteIndented = true,

AllowTrailingCommas = true,

IncludeFields = true,

IgnoreReadOnlyFields = true,

IgnoreReadOnlyProperties = true,

IgnoreNullValues = true,

PropertyNameCaseInsensitive = true,

};

options.Converters.Add(new ConditionConverter());

var jsonString = File.ReadAllText("../../../rules.json");

var rules = JsonSerializer.Deserialize<Rule[]>(jsonString, options);

var memory = new MemoryComponent(rules);

var inferenceComponent = new InferenceComponent(memory);

var result = inferenceComponent.GetResult();

if (result is Fact resultFact)

{

Console.WriteLine($"Результат:\n{resultFact.Name}");

Console.ReadKey();

}

else

{

Console.WriteLine("Результат не определен");

}

}

}

}

using System.Collections.Generic;

using System.Linq;

namespace infsystem

{

class InferenceComponent

{

private MemoryComponent Memory;

public InferenceComponent(MemoryComponent memory)

{

Memory = memory;

}

// Алгоритм вывода выбирает одно из правил, готовых к активации (условия которых выполнены),

// добавляет правую часть этого правила к списку действующих фактов

// (при необходимости - задает вопрос пользователю),

// если один из добавленных фактов является результатом - возвращает его.

public Fact? GetResult()

{

UpdateAgenda();

while (Memory.Agenda.Any())

{

var (ruleToActivate, requirements) = GetOrderedAgenda().First();

Memory.ActivateRule(ruleToActivate, requirements);

foreach (var fact in ruleToActivate.Assertions)

{

if (fact.Input is Input input)

{

fact.Value = UserInput.AskQuestion(input);

}

Memory.AddFact(fact, ruleToActivate);

if (fact.IsResult) return fact;

}

UpdateAgenda();

}

return null;

}

// Алгоритм разрешения конфликтов

// Правила отсортированы сначала по заданному приоритету,

// затем по конкретности (сначала с наибольшим количеством условий),

// затем по порядку выполнения их условия (сначала самые новые)

private Dictionary<Rule, List<Fact>> GetOrderedAgenda() =>

Memory.Agenda

.Select((rule, i) => new

{

Rule = rule.Key,

OrderInAgenda = i,

rule.Key.Salience,

rule.Key.Condition.Specificity,

Requirements = rule.Value,

})

.OrderByDescending(ruleWithOrdering => ruleWithOrdering.Salience)

.ThenByDescending(ruleWithOrdering => ruleWithOrdering.Specificity)

.ThenByDescending(ruleWithOrdering => ruleWithOrdering.OrderInAgenda)

.ToDictionary(

ruleWithOrdering => ruleWithOrdering.Rule,

ruleWithOrdering => ruleWithOrdering.Requirements

);

private void UpdateAgenda()

{

foreach (var (rule, \_) in Memory.Agenda)

{

var result = rule.Condition.Evaluate(Memory.Facts.Keys);

if (!result.CanActivate)

{

Memory.RemoveFromAgenda(rule);

}

}

foreach (var rule in Memory.NotActivatedRules)

{

var result = rule.Condition.Evaluate(Memory.Facts.Keys);

if (result.CanActivate)

{

Memory.AddToAgenda(rule, result.Requirements);

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace infsystem

{

class MemoryComponent

{

// Список действующих фактов. Каждому факту в соответствие поставлено правило,

// в результате активации которого получен данный факт.

public readonly Dictionary<Fact, Rule?> Facts;

// Все правила базы знаний

public readonly Rule[] Rules;

// Список правил, условия которых уже выполнены, и которые могут быть активированы в данный момент

// (вместе со списком фактов, наличие которых обращает условие данного правила в истину)

public readonly Dictionary<Rule, List<Fact>> Agenda = new ();

public Rule[] NotActivatedRules =>

Rules.Where(rule => !ActivatedRules.ContainsKey(rule)).ToArray();

// Активированные правила, правая часть которых уже внесена в список действующих фактов

public Dictionary<Rule, List<Fact>> ActivatedRules = new ();

public MemoryComponent(IEnumerable<Rule> rules)

{

Rules = rules.ToArray();

Facts = new();

}

public MemoryComponent(IEnumerable<Rule> rules, IEnumerable<Fact> initialFacts)

{

Rules = rules.ToArray();

Facts = initialFacts.ToDictionary<Fact, Fact, Rule?>(

fact => fact,

\_ => null

);

}

public void AddFact(Fact fact, Rule reason) => Facts.Add(fact, reason);

public void ActivateRule(Rule rule, List<Fact> requirements)

{

if (ActivatedRules.ContainsKey(rule))

throw new InvalidOperationException($"Rule already activated:\n{rule}");

ActivatedRules.Add(rule, requirements);

RemoveFromAgenda(rule);

}

public void AddToAgenda(Rule rule, List<Fact> requirements)

{

if (Agenda.ContainsKey(rule)) RemoveFromAgenda(rule);

Agenda.Add(rule, requirements);

}

public void RemoveFromAgenda(Rule rule)

{

if (!Agenda.Remove(rule))

throw new InvalidOperationException($"Rule was not on agenda:\n{rule}");

}

}

}

using System;

using System.Linq;

namespace infsystem

{

static class UserInput

{

public static Value AskQuestion(Input input)

{

string question = input.Question.Trim();

if (question[^1] != '?') question += ':';

switch (input.Type)

{

case ValueType.String:

return new Value {StringValue = AskQuestionWithPossibleValues(question, input.Values)};

default:

throw new ArgumentException("Question doesn't have a valid type");

}

}

private static string AskQuestionWithPossibleValues(string question, string[] inputValues)

{

Console.WriteLine(question);

for (int i = 0; i < inputValues.Length; i++)

{

Console.WriteLine($"{i + 1} - {inputValues[i]}");

}

string? value = null;

while (value == null)

{

var input = Console.ReadLine();

if (input == null) continue;

if (ParseInt(input) is int index && index >= 1 && index <= inputValues.Length)

{

value = inputValues[index - 1];

}

else

{

input = input.ToLower();

var foundValue = inputValues.FirstOrDefault(value => value.ToLower() == input);

if (foundValue != null) value = foundValue;

}

}

return value;

}

private static int? ParseInt(string input)

{

try

{

return int.Parse(input);

}

catch (FormatException)

{

return null;

}

}

private static double? ParseDouble(string input)

{

try

{

return double.Parse(input);

}

catch (FormatException)

{

return null;

}

}

}

}