Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

ОТЧЁТ

по ознакомительной практике

Выполнил: Л. Р. Свита

Студент группы
321703
Проверил: В. Н. Тищенко

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель:

Закрепить практические навыки формализации информации в интеллектуальных системах с использованием семантических сетей.

Задачи:

- Построение формализованных фрагментов теории интеллектуальных компьтерных систем и технологий их разработки;
- Построение формальной семантической спецификации библиографических источников, соответствующих указанным выше фрагментам;
- Оформление конкретных предложений по развитию текущей версии Стандарта интеллектуальных компьтерных систем и технологий их разработки

1 ПОСТАНОВКА ЗАДАЧИ

Часть 3 Учебной дисциплины "Представление и обработка информации в интеллектуальных системах"

⇒ *библиографическая ссылка**:

- *Найханова Л.В.СистеИИ-2002лк*
  - ⇒ *URL**:
    [https://studfile.net/preview/986652/page:16/]
- *И. А. Кудрявцева КлассиППвТП-2010кн*
  - ⇒ *URL**:
    [https://lib.herzen.spb.ru/media/magazines/contents/1/173/kudryavtseva]
- *Студопедия.Орг ЭксперСJ-2022ст*
  - ⇒ *URL**:
    [https://studopedia.org/7-95623.html]
- *Джозеф Джарратано Эксперт:ПРиП-2007кн*
  - ⇒ *URL**:
    [https://ru.wikipedia.org/wiki/CLIPS#: :text=CLIPS%2C%20]
- *авриков, М. М. ТеореОРиРЯП-2015кн*
  - ⇒ *URL**:
    [https://clipsrules.net/]
- *Частиков А. П. РазраЭс.СС-2003кн*
  - ⇒ *URL**:
    [https://clipsrules.net/]
- *Ликин С.С. РазраПСЭСсПМ-2012кн*
  - ⇒ *URL**:
    [https://clipsrules.net/]

⇒ *аттестационные вопросы**:

⟨ • *Вопрос 5.1 по Части 3 Учебной дисциплины "Представление и обработка информации в интеллектуальных системах"*

⟩

Вопрос 5.2 по Части 3 Учебной дисциплины "Представление и обработка информации в интеллектуальных системах"

:= [Языки продукционного программирования,используемые ostis-системами]

⇒ *библиографическая ссылка**:

- *Джозеф Джарратано, Гари Райли ЭкспеС:ПРиП-2015кн*
  - := [Экспертные системы: принципы разработки и программирование]
- *Частиков А. П., Гаврилова Т. А. РазраЭС.СС-2003кн*
  - := [Разработка экспертных систем. Среда CLIPS.]
- *Гавриков М. М., Иванченко А. Н., Гринченков Д. В. ТеореОРиРЯЗ-2023кн*
  - := [Теоретические основы разработки и реализации языков программирования]
- *Рыбина Г.В. ИструСПДИЭС:РКА*
  - := [Инструментальные средства построения динамических интегрированных экспертных систем: развитие комплекса АТ-ТЕХНОЛОГИЯ-2011кн]
- *Аликин С.С., Жидаков В.П. РазраПСЭСсПМ*
  - :=

2 ФОРМАЛИЗОВАННЫЕ ФРАГМЕНТЫ ЯЗЫКОВ ПРОДУКЦИОННОГО ПРОГРАММИРОВАНИЯ,ИСПОЛЬЗУЕМЫХ OSTIS-СИСТЕМАМИ

Язык OPS5

:= [компьютерный язык, основанный на правилах или производственной системе, известный как первый подобный язык, который был использован в успешной экспертной системе, системе R1 / XCON, используемой для настройки компьютеров VAX. Семейство OPS (как говорят, сокращение от "Официальной производственной системы") было разработано в конце 1970-х Чарльзом Форги во время учебы в Университете Карнеги-Меллон.]

⇒ *примечание**:

[

Создано несколько универсальных языков продукционного программирования. Одним из наиболее известных является OPS5, разработанный в Университете Карнеги-Меллона.

]

⇒ *примечание**:

[

В OPS5 база данных называется рабочей памятью и состоит из нескольких сотен объектов с наборами атрибутов. Элемент рабочей памяти выглядит следующим образом:
Программа на OPS5 — это множество продукций, каждая из которых имеет следующие особенности. Левая часть продукции — это последовательность образцов, каждый из которых представляет собой частичное описание элементов рабочей памяти. Некоторые образцы начинаются символом "-". Такие образцы являются средством задания отрицательного контекста.

]

⇒ *примечание**:

[

Условия левой части правила выполнены, когда:
Для каждого образца есть соответствующий элемент рабочей памяти.
Для образцов с - нет подходящих элементов в рабочей памяти.

]

⇒ *примечание**:

[

Простейшие образцы в OPS5 содержат только константы и числа, но наибольший интерес представляют образцы с переменными и предикатами, выделяемые в угловых скобках.
Предикаты OPS5 включают стандартные операции сравнения: =, ≠, >, <.
Правая часть продукции состоит из последовательности безусловных действий, которые изменяют рабочую память: MAKE (создание нового элемента), MODIFY (изменение атрибутов существующего элемента) и REMOVE (удаление элемента).

]

⇒ *примечание**:

[

Основная задача разработчиков OPS5 - обеспечить высокую эффективность работы продукционной программы.
Для этого был предложен RETE-алгоритм, который позволяет эффективно строить конфликтное множество (без перебора элементов рабочей памяти и продукций).

]

⇒ *примечание**:

[

RETE-алгоритм связывает с каждым образцом список элементов рабочей памяти, которые ему соответствуют. Это позволяет избежать поиска по образцам при добавлении или удалении элементов.
Над продукционными правилами строится сеть, которая позволяет эффективно сортировать продукции.
RETE-алгоритм обладает высокой степенью внутреннего параллелизма и активно используется в современных интеллектуальных системах на основе OPS5.

]

Язык Рефал

:= [язык манипулирования символьными объектами, такими как текстами, формулами, программами и т. п.]

⇒ *пояснение**:

[

Рефал является универсальным метаязыком для описания преобразований языковых объектов. Он позволяет описывать сложные преобразования текста без ограничений. Основная операция в Рефале - конкретизация, переход от имени к значению. Минимальная семантическая единица - символ, заключенный в кавычки. Программа на Рефале - набор правил конкретизации, состоящих из левой и правой частей.

]

⇒ *структура интерпретатора Рефала**:

[

Включает три компонента: рабочую память, базу правил и интерпретатор. Интерпретатор просматривает правила в порядке их перечисления и применяет первое подходящее. Конкретизация происходит в соответствии с дисциплиной ведущего знака конкретизации.

]

⇒ *дополнительные особенности**:

[

Правила Рефала могут содержать переменные, что требует выполнения операции поиска по образцу. Правила используют только те переменные, которые были введены в левой части. К середине 70-х были разработаны эффективные программные версии Рефала. Большое количество приложений Рефала способствовало разработке общей метамодели систем продукций.

]

Язык Clips

:= [экспертная система и язык программирования, разработанный в 1980-х годах в Космическом центре имени Джонсона (NASA). Он предназначен для построения экспертных систем и систем искусственного интеллекта.]

⇒ *особенности**:

[

Поддерживает объектно-ориентированное программирование. Объекты определяются с помощью классов, слотов и сообщений.

Интегрирует декларативное (эвристические правила) и процедурное (функции) программирование.

Имеет встроенные средства для логического вывода и управления очередностью выполнения правил.

Написан на языке программирования C, что обеспечивает его портативность и производительность.

Бесплатный и открытый для использования и модификации.

Используется для построения экспертных систем в различных областях, таких как планирование, диагностика, обучение и т.д.

]

⇒ *вид главного файла в Clips**:

[

(clear) - очищает текущее состояние CLIPS, чтобы начать с чистого листа.

(load* "classes.clp") - загружает определение структуры классов из файла classes.clp.

(load-instances "instances.clp") - загружает экземпляры классов из файла instances.clp.

(load* "templates.clp") - загружает предопределенные факты из файла templates.clp.

(load* "functions.clp") - загружает пользовательские функции из файла functions.clp.

(load* "rules.clp") - загружает правила из файла rules.clp.

(reset) - сбрасывает состояние CLIPS, чтобы загруженные данные были готовы к использованию.

(run) - запускает интерпретацию загруженной программы.

]

⇒ *представления данных в CLIPS**:

[

Факты - это списки элементарных значений, которые могут быть упорядоченными или неупорядоченными (шаблонные факты).

Глобальные переменные доступны во всем окружении CLIPS и их значения сохраняются между вызовами reset.

Объекты в CLIPS могут быть различных типов: символьные, строковые, числовые, объекты пользовательских классов.

Пользовательские классы определяются с помощью конструкции defclass, а их экземпляры создаются с помощью make-instance.

]

⇒ *концепции языка**:

[

Правила в CLIPS состоят из двух частей - условия (LHS) и действия (RHS). Если условия LHS выполняются, то выполняются действия RHS. Правила создаются с помощью конструкции defrule.

CLIPS поддерживает процедурный подход с помощью функций, определяемых конструкцией deffunction. Функции могут вызываться как из LHS, так и из RHS правил. Взаимодействие с классами осуществляется через "сообщения определяемые с помощью defmessage-handler. Это позволяет инкапсулировать поведение объектов.
Использование правил (эвристический подход) и функций (процедурный подход) дает возможность сочетать декларативное и императивное программирование в CLIPS.
]

Язык Jess

:= [движок правил для платформы Java, разработанный Эрнестом Фридманом-Хиллом из Sandia National Labs. Это надмножество языка программирования CLIPS. Впервые он был написан в конце 1995 года. Язык обеспечивает программирование на основе правил для автоматизации экспертной системы и часто называется оболочкой экспертной системы. В последние годы также были разработаны интеллектуальные агентные системы, которые зависят от аналогичных возможностей.]

⇒ *особенности**:

[
Jess - это Java-реализация оболочки экспертных систем, которая развилась из языка CLIPS, но стала самостоятельной динамической средой.
Jess позволяет создавать Java-приложения, сочетающие процедурный код с обработкой данных на основе представленных в виде правил знаний.
Jess совместим с CLIPS, то есть многие скрипты CLIPS будут работать в Jess и наоборот.
В основе ядра Jess лежит эффективный Rete-алгоритм для сопоставления фактов с правилами, который избегает повторных вычислений.
Rete-алгоритм реализован в Jess с помощью иерархической системы узлов, отвечающих за проверку фактов и запуск соответствующих действий.
В отличие от CLIPS, исходный код Jess не является открытым.
]

⇒ *гибкость и расширяемость**:

[
Jess поддерживает пользовательские функции и классы Java, что обеспечивает высокую гибкость и возможность расширения.
Существует обширная библиотека встроенных функций, которые можно использовать в правилах Jess.
]

⇒ *модульность**:

[
Jess позволяет разделять программу на модули, что улучшает структурированность и модульность кода.
Модули могут импортировать и использовать правила, функции и классы из других модулей.
]

⇒ *объектно-ориентированный подход**:

[
Jess поддерживает объектно-ориентированный стиль программирования, позволяя определять классы и работать с объектами.
Объекты могут представлять факты в базе знаний и участвовать в сопоставлении с правилами.
]

⇒ *эффективность**:

[
Jess использует эффективный алгоритм Rete для сравнения правил с фактами, что делает его производительным при работе с большими базами знаний.

Jess также поддерживает параллельную обработку, что улучшает производительность на многоядерных системах.
]

⇒ *встроенная трассировка и отладка**:

[
Jess предоставляет встроенные средства для отладки и трассировки выполнения программы, что упрощает разработку и тестирование.
Есть возможность пошагового выполнения программы и просмотра состояния базы знаний.
]

⇒ *концепции языка**:

[
Продукционное программирование: Jess использует продукционную модель, в которой программа состоит из набора правил "если-то называемых продукциями. Эти правила сопоставляются с фактами в базе знаний, и применяются соответствующие действия.
Вывод на основе правил: Jess использует механизм логического вывода, чтобы применять правила и делать заключения на основе фактов в базе знаний. Он поддерживает прямой и обратный вывод.
Динамическая база знаний: База знаний Jess может динамически изменяться во время выполнения программы за счет добавления, изменения или удаления фактов.
Интеграция с Java: Jess тесно интегрирован с Java, позволяя использовать Java-классы и объекты в правилах Jess, а также вызывать Jess-команды из Java-кода.
Jess является кроссплатформенным и может работать на различных операционных системах, для которых доступна Java.
Jess используется для создания экспертных систем, систем поддержки принятия решений, систем автоматизации бизнес-процессов и других интеллектуальных приложений.
]

Язык ACT-R

:= [когнитивная архитектура, разработанная для моделирования и понимания человеческого мышления и поведения. Это символьно-подсимвольная система, которая использует модульный подход для представления различных когнитивных процессов, таких как восприятие, память, принятие решений и моторные действия. ACT-R основана на теории рационального действия и способна обучаться и адаптироваться на основе опыта]

⇒ *основы архитектуры ACT-R**:

[

ACT-R состоит из различных модулей, каждый из которых отвечает за определенный когнитивный процесс, такой как восприятие, память, принятие решений и т.д.

Модули взаимодействуют друг с другом через общий буфер рабочей памяти.

Декларативная память: хранит фактические знания и информацию в виде семантических единиц (концептов).

Процедурная память: хранит навыки и умения в виде правил продукции (если-то правила).

Память основана на теории распространения возбуждения, когда активация распространяется между связанными элементами памяти.

Процесс принятия решений основан на оценке полезности различных вариантов действий.

Выбирается действие, которое максимизирует ожидаемую полезность.

Полезность зависит от текущего контекста, цели и прошлого опыта.

Декларативная память обновляется через механизмы кодировки и консолидации информации.

Процедурная память обновляется за счет механизмов обобщения и специализации правил продукции.

Обучение происходит через взаимодействие с окружающей средой и накопление опыта.

Ядро ACT-R состоит из модулей восприятия, памяти, принятия решений и управления.

Дополнительные модули могут быть добавлены для моделирования специфических когнитивных процессов.

Модели ACT-R строятся с использованием правил продукции и математических функций.

]

⇒ *история разработки**:

[

ACT-R состоит из различных модулей, каждый из которых отвечает за определенный когнитивный процесс, такой как восприятие, память, принятие решений и т.д.

JACT-R был разработан Джоном Андерсоном в 1970-х годах, изначально назывался просто ACT, а приставка "R"была добавлена позже.

На протяжении многих лет ACT-R постоянно развивался и совершенствовался, выходили новые версии.

]

⇒ *применение**:

[

ACT-R широко используется для моделирования когнитивных процессов в областях обучения, психологии, человеко-машинного взаимодействия, ИИ и робототехники.

]

⇒ *ограничения**:

[
Сложность настройки и калибровки моделей под конкретные задачи
Необходимость глубоких знаний о когнитивных процессах для эффективного использования
Ограниченность в моделировании эмоциональных и социальных аспектов
]

⇒ *преимущества**:

[
Высокая объяснительная и предсказательная способность моделей
Тесная интеграция с экспериментальными данными и теориями когнитивной психологии
Возможность создания реалистичных моделей человеческого поведения
Гибкость и возможность расширения за счет модульной архитектуры
]

⇒ *перспективы**:

[
Дальнейшее развитие и интеграция с другими подходами
Применение в ИИ для создания более интеллектуальных систем
Использование как инструмент для изучения человеческого мышления и поведения
]

Язык drools

:= [открытый программный инструмент для правил управления бизнес-процессами (Business Rules Management System, BRMS), который основан на правилах принятия решений (Business Rules Engine, BRE). Он разработан сообществом JBoss, которое входит в состав Red Hat.]

⇒ *основные составляющие**:

[

Drools Compiler: отвечает за преобразование исходного кода правил в исполняемый формат. Поддерживает различные диалекты правил, такие как Drools Rule Language (DRL), Domain-Specific Language (DSL) и Decision Model and Notation (DMN).

Drools Engine:это ядро системы, отвечающее за выполнение правил. Использует эффективные алгоритмы, такие как Rete и Phreak, для быстрого сопоставления фактов с правилами. Управляет жизненным циклом сессий, в которых выполняются правила.

Drools Persistence:обеспечивает сохранение и загрузку состояния рабочей памяти, базы знаний и других данных. Позволяет восстановить работу системы после перезапуска или отказа.

Drools Workbench:интегрированная среда разработки (IDE) для создания, тестирования и управления правилами. Предоставляет визуальные инструменты для моделирования, редактирования и развертывания правил.

Drools Guvnor:репозиторий для хранения и управления бизнес-правилами и связанных с ними артефактов. Обеспечивает возможность коллективной разработки, версионирования и развертывания правил.

Drools Fusion:Расширение Drools для обработки потоковых данных и сложных событий. Позволяет определять правила для реагирования на события в режиме реального времени. **Drools Expert**:ядро движка правил Drools, отвечающее за выполнение основных операций с правилами.

Drools Planner:модуль оптимизации, который помогает решать задачи планирования и подбора, используя правила. **Drools jBPM**:интеграция Drools с фреймворком управления бизнес-процессами jBPM. Позволяет объединять правила и бизнес-процессы для комплексной автоматизации.

]

⇒ *основные концепции**:

[

Рабочая память - это хранилище фактов (объектов), которые представляют текущее состояние системы.

Факты могут быть добавлены, изменены или удалены из рабочей памяти во время выполнения.

База знаний содержит набор правил, которые определяют поведение системы.

Правила определяют условия (when) и действия (then), которые должны быть выполнены.

Движок правил - это сердце Drools, отвечающее за сопоставление фактов с правилами и выполнение соответствующих действий.Он использует эффективные алгоритмы, такие как Rete и Phreak, для быстрого выполнения правил.

Agenda - это список правил, готовых к исполнению на основе текущего состояния рабочей памяти.

Правила выполняются в определенном порядке, который можно настроить с помощью различных стратегий разрешения конфликтов.

Drools поддерживает несколько диалектов для определения правил, включая Drools

Rule Language (DRL), Domain-Specific Language (DSL) и Decision Model and Notation (DMN).
Эти языки предоставляют различные уровни абстракции и удобство использования для бизнес-пользователей.
Сессия - это контейнер, который объединяет рабочую память, базу знаний и движок правил.
Сессия отвечает за управление жизненным циклом выполнения правил, включая их активацию, отмену и выполнение.
Обработчики событий позволяют реагировать на изменения в рабочей памяти и запускать соответствующие правила.
Это помогает реализовать динамическое и событийно-ориентированное поведение системы.
]

⇒ *применение**:

[
Финансовый сектор: Drools используется для автоматизации процессов принятия решений, управления рисками, соблюдения нормативных требований и проведения финансовых операций.
Здравоохранение: Drools применяется для поддержки клинических решений, анализа медицинских данных, управления страховыми требованиями и оптимизации рабочих процессов.
Государственный сектор: Drools помогает автоматизировать административные процессы, управлять нормативно-правовой базой и принимать решения по государственным программам и политикам.
Производство и логистика: Drools используется для управления производственными процессами, оптимизации цепочек поставок, управления ресурсами и планирования.
Телекоммуникации: Drools применяется для управления тарифами, биллинга, обработки заявок клиентов и управления сетевыми ресурсами.
]

⇒ *ограничения**:

[
Сложность настройки и калибровки моделей под конкретные задачи
Необходимость глубоких знаний о когнитивных процессах для эффективного использования
Ограниченность в моделировании эмоциональных и социальных аспектов
]

⇒ *перспективы**:

[
Интеграция с искусственным интеллектом (ИИ) и машинным обучением (МО): Drools может быть интегрирован с передовыми ИИ-технологиями для создания более интеллектуальных систем принятия решений.
Поддержка больших данных и потоковой обработки: Drools продолжит развиваться в направлении обработки больших объемов данных и обеспечения высокопроизводительной обработки событий в режиме реального времени.
Расширение возможностей моделирования и визуализации: Дальнейшее развитие инструментов Drools Workbench и Guvnor для более удобного моделирования, тестирования и визуализации правил.

Облачные и распределенные решения: Drools будет все чаще использоваться в облачных средах и в распределенных архитектурах для обеспечения масштабируемости и отказоустойчивости.
Интеграция с другими технологиями управления бизнес-правилами: Drools может расширять свои возможности за счет интеграции с другими популярными системами управления правилами, таких как BRMS, BPMS и RPA.
]

Языки продукционного программирования

:= [класс языков программирования, основанных на концепции продукционных правил]

⇒ *основные составляющие**:

[

Продукционные правила (production rules): Основные элементы продукционной системы, состоящие из условия (если) и действия (то). Когда условие правила выполняется, выполняется соответствующее действие.

База знаний (knowledge base):Хранилище продукционных правил, фактов и других знаний, используемых продукционной системой.

Интерпретатор (inference engine): Механизм, который сопоставляет текущее состояние системы с условиями правил в базе знаний, выбирает применимые правила и управляет их выполнением.

Циклический процесс (recognize-act cycle):Последовательность шагов, выполняемых интерпретатором: распознавание применимых правил, разрешение конфликтов, выполнение действий.

Разрешение конфликтов (conflict resolution):Механизм, используемый интерпретатором для выбора наиболее подходящего правила для выполнения, когда несколько правил могут быть применены.

Рабочая память (working memory):Динамическое хранилище фактов и текущего состояния системы, используемое интерпретатором.

Вывод (inference):Процесс применения продукционных правил к текущему состоянию системы для вывода новых фактов или принятия решений.

Прямой вывод (forward chaining):Стратегия вывода, при которой интерпретатор применяет правила, условия которых соответствуют текущему состоянию системы.

Обратный вывод (backward chaining): Стратегия вывода, при которой интерпретатор применяет правила, чтобы доказать определенную цель или гипотезу.

Оболочка экспертной системы (expert system shell):Программная среда, предоставляющая инструменты и инфраструктуру для построения экспертных систем на основе продукционных правил.

]

База знаний

:= **[** централизованное хранилище, содержащее все факты, правила и другие знания, необходимые для функционирования продукционной системы. Она является основным компонентом, который отличает продукционные системы от других подходов к программированию.**]**

⇒ *основные характеристики**:

[

Структура: База знаний обычно организована в виде набора продукционных правил, фактов, объектов, отношений и других структур данных, зависящих от конкретной предметной области.

Декларативность:Знания в базе представлены в декларативной форме, описывающей "что"(факты и правила), а не "как"(процедурное решение).

Модульность: База знаний состоит из относительно независимых модулей (наборов правил), что упрощает ее поддержку и развитие.

Расширяемость:Новые знания легко добавляются в базу без необходимости перестраивать всю систему.

Логический вывод:Интерпретатор использует логические механизмы (прямой и обратный вывод) для применения правил и выведения новых фактов из текущего состояния базы знаний.

Объяснение:База знаний позволяет объяснять, как были получены выводы, показывая последовательность примененных правил.

]

⇒ *формы представления**:

[

Факты: Простые утверждения о состоянии мира, например, "Солнце - звезда".

Правила:Продукционные правила вида "ЕСЛИ-ТО описывающие связи между условиями и действиями.

Фреймы:Структуры для представления объектов, их атрибутов и отношений.

Семантические сети:Графы, отображающие концепты и отношения между ними.

Логические утверждения:Представление знаний в виде логических формул.

]

Продукционные правила

:= [структурно-лингвистические модели представления процедурных знаний предметной области (рекомендаций, указаний, стратегий или эвристических правил), которые формально записываются в виде следующих пар:
•Если <условие>, то <действие>
• Если <причина>, то <следствие>
• Если <посылка>, то <заключение>
• Если <ситуация>, то <действие>.]

⇒ *примечание**:

[
Условную (левую) часть правила также называют **антецедентом**, а часть действия (правую) — **консеквентом**
]

⇒ *классификация продукционных правил**:

[
Простое: это ПП, имеющее единственное условие и единственное действие. Например: «ЕСЛИ (основной аппарат насос), ТО (технологический блок —блок нагнетания)»
Составное : имеет множество условий и действий. Например: «ЕСЛИ (аппарат—ректификационная колонна) И (габариты аппарата—крупногабаритный), ТО (высота установки аппарата —на нулевом уровне) И (очередность размещения — в первую очередь)».
Фокусирующее :имеет множество условий и одно действие. Например: «ЕСЛИ (основной аппарат —абсорбер) И (узел вспомогательного назначения — узел теплообмена) И (узел вспомогательного назначения — узел перекачки), ТО (технологический блок —блок абсорбции)».
Разветвляющееся:имеет одно условие и множество действий. Например: «ЕСЛИ (технологический блок—блок перекачки), ТО (основной аппарат — емкость) И (узел вспомогательного назначения — узел нагнетания)».
]

Тенденции развития продукционных языков программирования

⇒ *основные направления**:

[

Интеграция с ИИ и МО: Продукционные языки все чаще интегрируются с технологиями искусственного интеллекта и машинного обучения, позволяя создавать более интеллектуальные системы, способные к самообучению и адаптации.

Обработка больших данных и потоковая обработка:Продукционные языки совершенствуются в направлении эффективной обработки и анализа больших объемов данных, а также обработки данных в режиме реального времени.

Улучшение declarative программирования:Продукционные языки продолжают развивать парадигму декларативного программирования, делая ее более выразительной и удобной для пользователей. **Повышение производительности**:Ведется работа над оптимизацией производительности продукционных языков, в том числе за счет параллельных вычислений и распределенных архитектур.

Многоплатформенность и кроссплатформенность:Продукционные языки становятся все более кроссплатформенными, позволяя разработчикам создавать приложения, которые могут работать на различных операционных системах и аппаратных платформах.

Упрощение разработки и повышение доступности:Интерфейсы и инструменты продукционных языков совершенствуются, становясь более интуитивными и доступными для пользователей, не имеющих глубоких технических знаний.

Расширение областей применения:Продукционные языки находят применение в новых областях, таких как Интернет вещей, робототехника, финансовые технологии и другие сферы, требующие интеллектуальной обработки данных и принятия решений.

]

Преимущества продукционных языков программирования

⇒ *пояснение**:

[

Декларативность: продукционные языки, основанные на декларативном подходе, позволяют описывать "что"должно быть сделано, а не "как"это сделать. Это делает код более читаемым, понятным и менее подверженным ошибкам.

Гибкость и адаптивность:продукционные системы легко адаптируются к изменениям в требованиях и данных, благодаря отделению логики от данных. Это упрощает поддержку и модификацию программ.

Интеграция с ИИ и МО:родукционные языки хорошо сочетаются с технологиями искусственного интеллекта и машинного обучения, позволяя создавать интеллектуальные системы, способные к самообучению и принятию решений.

Эффективная работа с данными:продукционные системы отлично подходят для обработки и анализа больших объемов данных, а также для работы с потоковыми данными в режиме реального времени.

Повышение производительности:Благодаря оптимизациям и использованию параллельных вычислений, продукционные языки демонстрируют высокую производительность при решении сложных задач.

Многоплатформенность:продукционные языки становятся все более кроссплатформенными, что позволяет разрабатывать универсальные приложения, работающие на различных операционных системах и аппаратных платформах.

Упрощение разработки:интуитивные интерфейсы и мощные инструменты для продукционных языков делают разработку более простой и доступной для широкого круга пользователей, включая непрограммистов.

Расширение областей применения:продукционные системы находят все более широкое применение в различных сферах, от Интернета вещей до финансовых технологий, что открывает новые возможности для их использования.

]

Недостатки продукционных языков программирования

⇒ *пояснение**:

[

Сложность понимания и освоения: продукционные системы, основанные на правилах и логическом выводе, могут быть более сложными для понимания и освоения, особенно для разработчиков, привыкших к императивному или объектно-ориентированному программированию.

Проблемы с производительностью:в некоторых случаях продукционные системы могут демонстрировать снижение производительности по сравнению с императивными подходами, особенно при решении задач, требующих высокой вычислительной мощности.

рудности отладки и тестирования:рПроцесс отладки и тестирования продукционных систем может быть более сложным из-за их динамического поведения и зависимости от порядка применения правил.

Ограничения масштабируемости:при работе с очень большими базами знаний или при необходимости обработки огромных объемов данных, продукционные системы могут столкнуться с проблемами масштабируемости.

Сложность интеграции с другими технологиями:встраивание продукционных систем в комплексные программные архитектуры может представлять определенные трудности из-за особенностей их реализации и взаимодействия с другими компонентами.

Ограниченная поддержка специфических областей:некоторые продукционные языки могут быть ограничены в поддержке специфических предметных областей или не иметь достаточного набора библиотек и инструментов для решения узкоспециализированных задач. **Проблемы с объяснением принятых решений**:в случае сложных продукционных систем, может быть трудно объяснить, как система пришла к определенному решению, что снижает прозрачность и доверие к ней.

]

ЗАКЛЮЧЕНИЕ

Во время ознакомительной практики научился процессу формализации текста на языке Scn: Выделению важной информации из статей, работе с монографией и стандартом. Также в рамках своей практической работы разобрался в предметной области формализованных мною понятий

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Аликин С.С., Жидаков В.П. Разработка платформы создания экспертных систем с применением метапрограммирования // Фундаментальные проблемы радиоэлектронного приборостроения / Жидаков В.П. Аликин С.С. — Ciberleninka, 2012.

[2] Гавриков, М. М. Теоретические основы разработки и реализации языков программирования / М. М. Гавриков, А. Н. Иванченко, Д. В. Гринченков. — Феникс, 2015.

[3] Рыбина2010,. Инструментальные средства построения динамических интегрированных экспертных систем: развитие комплекса АТ-ТЕХНОЛОГИЯ // Искусственный интеллект и принятие решений / Рыбина2010. — Ciberleninka, 2010.

[4] Частиков, А. П. Разработка экспертных систем. Среда CLIPS / А. П. Частиков, Т. А. Гаврилова, Д. Л. Белов. — БХВ-Петербург, 2003.

[5] Giarratano, Joseph. Expert Systems: Principles and Programming / Joseph Giarratano, Gary Riley. — 4th ed. — Course Technology, 2019.