

1. Определение и свойства интеллектуальных систем.

ИС – это адаптивная система, позволяющая строить программы целесообразной деятельности по решению поставленных перед ними задач на основании конкретной ситуации, складывающейся на данный момент в окружающей их среде.

ИС решает следующие задачи:

- В которых неизвестен алгоритм решения
- В которых используется информация в виде изображений, рисунков, знаков, букв, слов, звуков
- В которых предполагается наличие выбора

- ИС должна уметь в наборе фактов распознать существенные,

- ИС способны из имеющихся фактов и знаний сделать выводы не только с использованием - дедукции, но и с помощью аналогии, индукции т.д.

- ИС должны быть способны к самооценке.

ИС должна уметь обобщать.

2. Ключевые особенности интеллектуальных систем.

Способность к обучению

- Системы могут адаптироваться и улучшать свои действия на основе опыта (данных).

Обработка неполной и нечёткой информации

- Умеют работать с неясными, неполными или противоречивыми данными.

Принятие решений и рассуждение

- Используют правила, логические выводы и знания для решения задач.

Самоорганизация и адаптация

- Системы могут изменять свою структуру или поведение без внешнего вмешательства.

Использование знаний (база знаний)

- Хранят и используют знания о предметной области.

Интерактивность и обратная связь

- Взаимодействуют с пользователями или другими системами.

Работа в реальном времени

- Некоторые ИС работают с быстрым откликом, анализируя поток данных.

3. Компоненты интеллектуальных систем.

Интерфейс пользователя – то, как человек взаимодействует с ИС (текст, голос, кнопки).

Механизм вывода – «мозг» системы: делает выводы, принимает решения.

База знаний – хранит правила.

База данных - хранит факты.

Механизм объяснений – объясняет, почему система выдала такой ответ.

Модуль обучения – позволяет системе обучаться на новых данных (в современных ИС).

Подсистема управления – координирует работу всех компонентов.

4. Современные подходы и технологии интеллектуальных систем.

Современные интеллектуальные системы используют широкий спектр технологий. Основное направление — **машинное обучение**, включая **глубокое обучение** на основе нейросетей. Это позволяет системам обучаться на больших объёмах данных и выявлять сложные зависимости.

Обработка естественного языка (NLP) — ещё одно ключевое направление. Она используется для понимания текста, речи, перевода, генерации ответов. Благодаря NLP стали возможны такие системы, как голосовые помощники, чат-боты и генеративные модели (например, ChatGPT).

Интеллектуальные агенты — системы, способные автономно действовать в среде, принимая решения. Они активно используются в робототехнике, навигации, автоматизации.

Гибридные системы объединяют экспертные правила, нейросети и другие подходы, получая преимущества каждого.

Также применяются **нейроморфные вычисления**, **генетические алгоритмы**, **нечёткая логика**.

5. Этические и социальные аспекты ИИ. Будущее интеллектуальных систем.

Этика:

- Проблема ответственности: кто виноват, если ИИ ошибся?
- Приватность: ИИ собирает и анализирует личные данные, что создаёт риски утечки и слежки.
- Справедливость: ИИ не должен дискриминировать по полу, расе, возрасту или другим признакам.
- Прозрачность: важно, чтобы решения ИИ были объяснимы и понятны людям.

Социальные аспекты:

- Влияние на рынок труда: ИИ может заменить некоторые профессии, но создаст новые рабочие места.
- Необходимость переобучения и адаптации работников.
- Возможность усиления социального неравенства из-за доступа к технологиям.

Будущее интеллектуальных систем:

- Умные помощники в медицине, юриспруденции, образовании.
- Автономный транспорт и роботы-доставщики.
- Гибкие системы, которые учатся и адаптируются в реальном времени.
- Вопрос замены человека ИИ: скорее частичная автоматизация, чем полная замена.

6. Исторический обзор исследований в области искусственного интеллекта

Основные этапы развития ИИ:

- 1950 г. — Алан Тьюринг предложил тест, который проверяет, может ли машина думать как человек.
- 1956 г. — на Дартмутской конференции появился термин «искусственный интеллект» и начались активные исследования.
- 1960–70-е — создавались первые экспертные системы, программы на основе логики и символов.
- 1980-е — «зима ИИ» из-за ограничения вычислительных мощностей и завышенных ожиданий от технологий.
- 1997 г. — суперкомпьютер Deep Blue победил чемпиона мира по шахматам Гарри Каспарова, что стало значимым прорывом.
- 2000-е — развитие методов машинного обучения и нейронных сетей, рост объёмов данных.
- 2010-е — появление глубокого обучения, революция в распознавании образов и речи.
- 2020-е — генеративные модели (например, GPT), применение ИИ в медицине, финансах, творчестве и быту.

7. Понятие экспертной системы

Экспертная система — это интеллектуальная компьютерная программа, которая имитирует процесс принятия решений и рассуждений человеческого эксперта в определённой области знаний. Основой экспертной системы служит база знаний, содержащая факты и правила, а также механизм вывода, который применяет эти правила для решения конкретных задач.

Экспертные системы используют формальные методы представления знаний, чаще всего в виде правил «если — то», что позволяет им делать логические выводы. Они способны объяснять свои решения пользователю, что повышает доверие и помогает понять логику работы системы.

Экспертные системы широко применяются в медицине для диагностики заболеваний, в инженерии для поддержки проектирования, в финансах для оценки рисков и принятия решений.

Их основная цель — автоматизировать экспертные знания и сделать их доступными там, где нет специалиста или нужна быстрая помощь.

8) Правила как метод представления знаний.

Правила — это один из основных методов представления знаний в интеллектуальных системах. Они имеют форму условных утверждений вида:

ЕСЛИ <условие>, ТО <действие/заключение>

Характеристики правил:

1. **Декларативность** — правила описывают зависимости между фактами, а не последовательность выполнения.
2. **Модульность** — каждое правило независимо, что упрощает добавление и модификацию знаний.
3. **Прозрачность** — логика вывода легко интерпретируется человеком.

Применение в CLIPS и FuzzyCLIPS:

- В **CLIPS** правила используются для логического вывода на основе точных данных.
- В **FuzzyCLIPS** правила могут содержать нечеткие условия и выводы, работая с вероятностными и приближенными знаниями.

Пример в CLIPS:

(defrule правило-пример

(температура высокая)

=>

(активировать охлаждение))

Преимущества:

- Простота понимания и реализации.
- Эффективность в экспертных системах.

Недостатки:

- При большом количестве правил возможны конфликты.
- Требуют четкой формализации знаний.

Таким образом, правила — это удобный и наглядный способ представления знаний, особенно в системах, основанных на логическом выводе.

9) Основные участники команды разработчиков экспертной системы.

В разработке экспертной системы (ЭС) участвуют несколько ключевых специалистов, каждый из которых выполняет свою роль:

1. Эксперт в предметной области

- Обладает глубокими знаниями в конкретной сфере (медицина, инженерия, финансы и т. д.).
- Формулирует правила, концепции и критерии принятия решений.

2. Инженер по знаниям (когнитолог)

- Анализирует и структурирует знания эксперта.
- Переводит их в формализованный вид (правила, онтологии, фреймы).
- Выступает посредником между экспертом и программистом.

3. Программист (разработчик ЭС)

- Реализует систему, используя инструменты (CLIPS, FuzzyCLIPS, Prolog и др.).
- Настраивает механизм логического вывода и интерфейсы.

4. Пользователь

- Взаимодействует с готовой ЭС, вводя данные и получая рекомендации.
- Может быть конечным специалистом (врач, инженер) или тестировщиком.

5. Системный аналитик/Проектный менеджер

- Координирует процесс разработки.
- Обеспечивает соответствие системы требованиям заказчика.

Дополнительные роли (в крупных проектах):

- **Тестировщик** – проверяет корректность работы ЭС.
- **UX/UI-дизайнер** – разрабатывает удобный интерфейс.

Итог:

Успех разработки ЭС зависит от слаженной работы всех участников, где ключевыми являются **эксперт, инженер по знаниям и программист**.

10) Структура экспертной системы на основе правил.

Экспертная система (ЭС) на правилах состоит из **четырёх основных компонентов**:

1. База знаний (БЗ)

- Содержит **правила** вида «ЕСЛИ-ТО» и **факты** о предметной области.
- Пример (в CLIPS):

(defrule диагностика-гриппа

(температура ?t)

(if (>= ?t 38) then (assert (симптом "грипп"))))

2. Механизм логического вывода (инференции)

- Анализирует правила и факты, принимает решения.
- **Стратегии вывода:**
 - **Прямая цепочка** (от данных к выводу).
 - **Обратная цепочка** (от гипотезы к подтверждению).

3. Рабочая память (база данных)

- Хранит **входные данные** и **промежуточные результаты**.
- Пример:
(assert (температура 39))

4. Пользовательский интерфейс

- **Интерфейс ввода/вывода:** запрашивает данные, выводит рекомендации.
- **Объяснительный механизм** (опционально): поясняет, как система пришла к выводу.

Дополнительные компоненты:

- **Модуль приобретения знаний** – помогает инженеру по знаниям обновлять БЗ.
- **Отладчик** – для тестирования правил.

Итог:

ЭС на правилах – это **БЗ + механизм вывода + рабочая память + интерфейс**. Примеры: MYCIN (медицина), CLIPS-системы.

Коротко:

1. **База знаний** (правила).
2. **Механизм вывода** (прямой/обратный).
3. **Рабочая память** (факты).
4. **Интерфейс** (ввод/вывод + объяснения).

11) Основные характеристики экспертной системы. Сравнение экспертных систем с обычными системами и экспертами-людьми.

1. Основные характеристики экспертных систем (ЭС):

- **Специализация** – работают в узкой предметной области (медицина, диагностика, финансы).
- **Использование знаний** – опираются на формализованные правила и базу знаний (а не только алгоритмы).
- **Способность к рассуждению** – применяют логический вывод (прямой/обратный) для решения задач.
- **Объяснимость** – могут аргументировать свои решения (в отличие от многих традиционных программ).
- **Интерактивность** – взаимодействуют с пользователем через запросы и ответы.

2. Сравнение экспертных систем с обычными системами:

Критерий	Экспертные системы (ЭС)	Обычные системы
Основа работы	База знаний + логический вывод	Жесткие алгоритмы + данные
Гибкость	Могут адаптироваться к новым правилам	Требуют перепрограммирования
Объяснение решений	Да (например, через трассировку)	Нет (черный ящик)
Применение	Сложные неформализованные задачи	Стандартные вычислительные задачи

3. Сравнение экспертных систем с экспертами-людьми:

Критерий	Экспертные системы (ЭС)	Эксперты-люди
Скорость обработки	Быстрая (мгновенный вывод)	Зависит от человека, может быть медленной
Усталость/ошибки	Не устают, ошибки только из-за некорректных правил	Подвержены усталости и субъективности
Масштабируемость	Легко тиражируются и обновляются	Опыт одного эксперта сложно передать
Творчество/интуиция	Нет (работает только по заданным правилам)	Есть (может находить нестандартные решения)

Итог:

- ЭС **превосходят обычные системы** в сложных экспертных задачах, требующих рассуждений.
- ЭС **уступают людям-экспертам** в гибкости и интуитивном принятии решений, но выигрывают в скорости и стабильности.

Коротко:

- **ЭС vs обычные системы:** гибкость, объяснимость, работа со знаниями.
- **ЭС vs люди:** скорость, отсутствие усталости, но нет интуиции.

12) Методы вывода с прямой и обратной цепочкой в системах основанных на правилах.

1. Прямая цепочка рассуждений (Forward Chaining)

- **Принцип работы:**
 - Система **отталкивается от имеющихся данных** и применяет подходящие правила, пока не достигнет конечного вывода.
 - Пример: диагностика неисправности по симптомам.
- **Алгоритм:**
 1. В рабочую память загружаются исходные факты.
 2. Механизм вывода ищет правила, условия которых выполняются.
 3. Если правило срабатывает, его действие добавляется в рабочую память.
 4. Процесс повторяется, пока не будут обработаны все возможные правила.
- **Плюсы:**
 - Хорошо подходит для задач с **большим объемом входных данных**.
 - Позволяет находить **все возможные решения**.
- **Минусы:**
 - Может быть **неэффективным**, если цель заранее известна.
- **Пример в CLIPS:**

```
(defrule diagnose-engine-issue
  (symptom "high-temperature")
  (symptom "low-oil-pressure")
=>
  (assert (problem "engine-failure")))
```

2. Обратная цепочка рассуждений (Backward Chaining)

- **Принцип работы:**
 - Система **стартует с гипотезы (цели)** и проверяет, можно ли ее доказать, опираясь на факты и правила.
 - Пример: подтверждение медицинского диагноза.
- **Алгоритм:**
 1. Задается целевое утверждение (гипотеза).
 2. Система ищет правила, в действии которых содержится эта цель.
 3. Для каждого такого правила проверяются его условия (если они неизвестны, они становятся новыми подцелями).
 4. Процесс рекурсивно повторяется, пока не будут подтверждены или опровергнуты все условия.
- **Плюсы:**
 - Эффективен, когда **цель заранее определена**.
 - Тратит ресурсы только на **релевантные правила**.
- **Минусы:**
 - Может **пропускать альтернативные решения**, если не искать все возможные пути.
- **Пример в Prolog (похожий принцип):**

```
has_flu(X) :- has_fever(X), has_cough(X).
```


Критерий	Прямая цепочка	Обратная цепочка
Стартовая точка	Данные (факты)	Цель (гипотеза)
Эффективность	Лучше при множестве входных данных	Лучше при конкретной цели
Использование	Диагностика, мониторинг	Планирование, доказательство теорем

Итог:

- **Прямая цепочка** – "от данных к выводу", подходит для открытых задач.
- **Обратная цепочка** – "от цели к данным", оптимальна для целевого поиска решений.

Коротко:

- **Forward Chaining:** данные → вывод, используется в CLIPS.
- **Backward Chaining:** гипотеза → подтверждение, применяется в Prolog.

13) Разрешение конфликтов в системах основанных на правилах.

Конфликт возникает, когда одновременно активируются несколько правил, но система должна выбрать только одно (или определить порядок выполнения). В CLIPS и подобных системах используются следующие методы разрешения конфликтов:

1. Основные стратегии разрешения конфликтов

1.1. Явный приоритет правил (Salience)

- Каждому правилу можно назначить числовой приоритет
- Правило с более высоким значением выполняется первым
- Пример в CLIPS:

```
(defrule Правило-1
  (salience 10) ; высокий приоритет
  (факт А)
=> ...)
```

```
(defrule Правило-2
  (salience 5) ; низкий приоритет
  (факт А)
=> ...)
```

1.2. Специфичность правила

- Правило с большим количеством условий имеет более высокий приоритет
- Пример:

```
(defrule Детальное-правило ; выполнится первым
```

```
(факт А)
(факт Б)
=> ...)
```

```
(defrule Общее-правило ; выполнится вторым
(факт А)
=> ...)
```

1.3. Время добавления фактов

- Правила, связанные с последними добавленными фактами, получают приоритет
- Динамически изменяет порядок активации

1.4. Порядок определения правил

- По умолчанию в CLIPS: первое загруженное правило имеет приоритет
- Важен порядок правил в исходном коде

2. Дополнительные методы

2.1. Метаправила

- Специальные правила, управляющие активацией других правил
- Позволяют реализовать сложные стратегии разрешения конфликтов

2.2. Отключение правил

- Конфликтующие правила можно временно отключать
- В CLIPS: (undefrule имя_правила)

3. Особенности FuzzyCLIPS

- Конфликты могут возникать из-за пересечения нечетких множеств
- Решение:
 - Использование приоритетов
 - Агрегация результатов (выбор максимального значения)
 - Комбинирование нечетких выводов

4. Пример конфликта и его разрешения

```
(defrule Высокий-приоритет
(salience 100)
(температура ?t)
(test (> ?t 38))
=>
(assert (диагноз "Грипп")))
```

```
(defrule Низкий-приоритет
(salience 50)
(температура ?t)
```

```
(test (> ?t 37))  
=>  
(assert (диагноз "Простуда")))
```

Итог:

1. Основные методы в CLIPS: приоритеты, специфичность, порядок загрузки
2. В сложных случаях применяются метаправила
3. В FuzzyCLIPS учитывается нечеткость данных
4. Выбор стратегии зависит от конкретной задачи

14) Преимущества и недостатки интеллектуальных систем на основе правил

1. Преимущества

1. Прозрачность и интерпретируемость

- Логика работы системы четко выражена в виде правил «ЕСЛИ-ТО».
- Легко анализировать и объяснять принятые решения (важно в медицине, юриспруденции).

2. Модульность и простота обновления

- Правила можно добавлять, удалять или изменять независимо друг от друга.
- Не требует перепрограммирования всей системы (в отличие от алгоритмических систем).

3. Эффективность в узких предметных областях

- Хорошо работают для задач с четкими формализуемыми знаниями (диагностика, классификация).

4. Совместимость с экспертами-людьми

- Инженер по знаниям может напрямую работать с экспертом, переводя его знания в правила.

5. Поддержка в инструментах (CLIPS, FuzzyCLIPS)

- Готовые механизмы логического вывода и отладки.

2. Недостатки

1. Сложность масштабирования

- При большом количестве правил (сотни/тысячи) возникают:
 - **Конфликты** (неоднозначность активации правил).
 - **Проблемы с производительностью** (медленный вывод).

2. Жесткость и отсутствие гибкости

- Не могут обрабатывать ситуации, не заложенные в правилах.
- Нет способности к обучению (в отличие от нейросетей).

3. Зависимость от экспертов

- Качество системы напрямую зависит от:
 - Глубины знаний эксперта.
 - Умения инженера по знаниям их формализовать.

4. Проблемы с неполными/противоречивыми данными

- Требуют строгой полноты и непротиворечивости базы знаний.
- В FuzzyCLIPS частично решаются через нечеткую логику, но это усложняет систему.

5. Ограниченная применимость

- Плохо подходят для:
 - Обработки естественного языка.
 - Распознавания образов (например, изображений).

Вывод:

Системы на правилах оправданы, когда:

Нужна прозрачность решений.

Предметная область четко формализуема.

Не подходят для сложных, плохо структурированных задач (анализ изображений, NLP).

Примеры удачного применения:

- Медицинская диагностика (MYCIN).
- Техническая диагностика оборудования.
- Бизнес-правила в банковской сфере.

15. Выделите наиболее существенные характеристики фреймовой модели представления знаний.

Фреймовая модель — это способ структурирования знаний об объектах, событиях или ситуациях в виде **фреймов** (структур данных, напоминающих таблицы).

Основные характеристики:

1. Иерархическая структура

- Фреймы объединяются в иерархии (родительские и дочерние), что позволяет наследовать свойства (например, фрейм "животное" → фрейм "кошка").

2. Слоты и значения

- Каждый фрейм состоит из **слотов** (атрибутов), которые заполняются значениями (например, фрейм "автомобиль" может иметь слоты: "марка", "цвет", "мощность").
- Слоты могут иметь **значения по умолчанию** (например, у фрейма "птица" слот "способ передвижения" = "летает").

3. Процедурные прикрепления

- Возможность привязки процедур (скриптов, правил) к слотам, которые активируются при изменении данных (например, проверка допустимости значения или автоматический расчет).

4. Наследование свойств

- Дочерние фреймы наследуют атрибуты родительских, что сокращает дублирование информации (например, фрейм "студент" наследует свойства фрейма "человек").

5. Гибкость и модульность

- Фреймы позволяют легко добавлять, изменять и удалять слоты, адаптируя модель под новые знания.

6. Семантическая связность

- Фреймы отражают **типизированные знания** о мире, что делает их удобными для моделирования предметных областей (например, медицина, технические системы).

7. Использование для сценариев и стереотипов

- Эффективны для описания стандартных ситуаций (например, фрейм "ресторан" включает слоты: "меню", "официант", "оплата").

16. Как работает механизм вывода в системе на основе фреймов

Фреймовые системы используют **иерархическую структуру знаний** и **наследование свойств** для логического вывода. Вывод в них может происходить несколькими способами, включая:

1. Наследование значений по умолчанию

- Если в дочернем фрейме отсутствует значение слота, система ищет его в родительском фрейме.
- *Пример:*
 - Родительский фрейм "Птица" имеет слот "Способ передвижения" = "Летает".
 - Дочерний фрейм "Пингвин" не имеет этого слота, но наследует его.
 - Однако если в "Пингвин" явно указано "Способ передвижения" = "Плавает", система использует это значение вместо наследуемого.

2. Активация процедурных прикреплений

- Некоторые слоты содержат **прикрепленные процедуры** (демоны), которые выполняются при:
 - **Если-добавлено** (при заполнении слота).
 - **Если-нужно** (при попытке получить значение).
 - **Если-удалено** (при очистке слота).
- *Пример:*
 - В фрейме "Банковский счет" слот "Баланс" может иметь процедуру "Если-нужно", которая вычисляет баланс на основе транзакций.

3. Сопоставление фреймов (фрейм-сценарии)

- Система может сравнивать текущую ситуацию с известными фреймами и выбирать наиболее подходящий.
- *Пример:*

- Если система видит объект с "**Крыльями**", "**Клювом**" и "**Перьями**", она активирует фрейм "**Птица**".

4. Логический вывод через связи между фреймами

- Фреймы могут ссылаться друг на друга, образуя семантическую сеть.
- *Пример:*
 - Фрейм "**Студент**" связан с фреймом "**Университет**", и система может вывести, где учится студент.

5. Обработка исключений и уточнение данных

- Если наследуемое значение противоречит явно заданному, система выбирает более конкретное.
- *Пример:*
 - Фрейм "**Киви**" (птица) наследует "**Летает**", но в слоте "**Особенности**" указано "**Не летает**" → система корректирует вывод.

17. Изложите основные положения нечеткой логики. Дайте определение лингвистической переменной.

Нечёткая логика — это математический аппарат для работы с неточными, размытыми понятиями, где истинность высказываний может принимать промежуточные значения между 0 и 1 (в отличие от классической булевой логики, где есть только 0 или 1).

1. Основные концепции

- Нечёткое множество (Fuzzy Set)
 - Объект может принадлежать множеству частично (степень принадлежности $\in [0, 1]$).
 - *Пример:* Температура 25°C может на 70% относиться к множеству "Тёплая".
- Функция принадлежности (Membership Function, μ)
 - Определяет степень принадлежности элемента к нечёткому множеству.
 - Примеры функций: треугольная, трапецевидная, гауссова.
- Лингвистические переменные
 - Переменные, значениями которых являются слова естественного языка (например, "высокий", "средний", "низкий").
- Нечёткие правила (Fuzzy Rules)
 - Правила вида "ЕСЛИ (условие), ТО (действие)", где условия и действия формулируются в нечётких терминах.
- Дефаззификация (Defuzzification)

- Преобразование нечёткого вывода в чёткое число (например, для управления системой).
- Методы: центроидный (центр тяжести), максимумы, среднее по максимумам.

2. Области применения

- Системы управления (кондиционеры, стиральные машины).
- Искусственный интеллект и принятие решений.
- Обработка естественного языка.
- Экономика и финансы (анализ рисков).

Лингвистическая переменная (Linguistic Variable)

Определение:

Лингвистическая переменная — это переменная, значения которой задаются словами или фразами естественного языка, а не числами. Каждое значение связано с нечётким множеством через функцию принадлежности.

Структура лингвистической переменной:

1. Имя переменной (например, "Скорость").
2. Терм-множество — набор её значений (например, {"Низкая", "Средняя", "Высокая"}).
3. База правил — нечёткие правила для вывода.
4. Функции принадлежности для каждого термина.

Пример:

- Переменная: "Температура"
- Термы: {"Холодная", "Прохладная", "Нормальная", "Тёплая", "Горячая"}
- Функции принадлежности:
 - "Холодная" $\rightarrow \mu(x) = 1$ при $x < 5^{\circ}\text{C}$, плавно убывает до 0 к 15°C .
 - "Тёплая" $\rightarrow \mu(x) = 1$ при 25°C , убывает к 15°C и 35°C .

18. Процесс извлечения знаний у экспертов. Методы извлечения знаний

Извлечение знаний (Knowledge Acquisition) — это процесс выявления, структурирования и формализации знаний экспертов для их дальнейшего использования в экспертных системах, базах знаний или системах искусственного интеллекта.

Основные этапы:

1. Идентификация проблемы – определение предметной области и целей извлечения знаний.
2. Выбор экспертов – подбор специалистов, обладающих нужными знаниями.

3. Сбор данных – применение методов извлечения (интервью, наблюдение и др.).
4. Анализ и структурирование – выявление ключевых понятий, правил, зависимостей.
5. Формализация – представление знаний в виде моделей (фреймы, правила, онтологии).
6. Верификация – проверка корректности извлечённых знаний.

Методы извлечения знаний

1. Интервьюирование

- Структурированное интервью – чёткий список вопросов.
- Неструктурированное интервью – свободная беседа с экспертом.
- Метод "мыслей вслух" – эксперт проговаривает свои рассуждения при решении задачи.

2. Наблюдение за работой эксперта

- Пассивное наблюдение – фиксация действий без вмешательства.
- Активное наблюдение – инженер по знаниям задаёт уточняющие вопросы.

3. Анализ протоколов (Case Studies)

- Разбор реальных кейсов, документов, отчётов, журналов принятия решений.

4. Метод репертуарных решёток (Kelly Grid)

- Эксперт сравнивает объекты по заданным параметрам, выявляя скрытые закономерности.

5. Мозговой штурм

- Групповая генерация идей с последующим анализом.

6. Метод сортировки карточек (Card Sorting)

- Эксперт группирует понятия по категориям, что помогает выявить структуру знаний.

19. Формализация извлечённых экспертных знаний

Формализация — это процесс преобразования неструктурированных знаний экспертов в строгую модель, пригодную для компьютерной обработки. Она включает структурирование, классификацию и запись знаний в соответствии с выбранным методом представления.

Основные этапы формализации

1. Структурирование знаний

- Выделение ключевых понятий, объектов, атрибутов и отношений.
- Разделение знаний на:
 - Декларативные (факты, описания).
 - Процедурные (алгоритмы, правила действий).

- Эвристические (опыт, нестрогие закономерности).

2. Выбор модели представления знаний

В зависимости от предметной области применяются:

- а) Логические модели - Знания записываются в виде логических формул.
- б) Продукционные - правила "ЕСЛИ-ТО"
- в) Фреймовые модели - Знания описываются иерархией объектов с атрибутами.
- д) Семантические сети - Знания представляются в виде графа (узлы — объекты, дуги — связи).

3. Запись знаний в формальном виде

- Использование специализированных языков:
 - Prolog (логические модели).
 - CLIPS, Drools (продукционные системы).
 - OWL, RDF/XML (онтологии).

4. Проверка на непротиворечивость

- Верификация (соответствие логике).
- Валидация (соответствие реальным знаниям эксперта).
- Методы:
 - Логический анализ (исключение конфликтующих правил).
 - Тестирование на контрольных примерах.

20. Методы измерения качества знаний

Качество знаний в экспертных системах, базах знаний и ИИ-моделях оценивается по **точности, полноте, непротиворечивости** и другим критериям. Рассмотрим ключевые методы измерения.

1. Критерии качества знаний

Перед оценкой необходимо определить **метрики**, по которым будут проверяться знания:

- **Точность** (Accuracy) – соответствие знаний реальным данным.
- **Полнота** (Completeness) – охват всех значимых случаев в предметной области.
- **Непротиворечивость** (Consistency) – отсутствие конфликтующих правил или фактов.
- **Понятность** (Interpretability) – ясность представления для пользователей.
- **Актуальность** (Relevance) – соответствие современным данным.
- **Надежность** (Reliability) – устойчивость к ошибкам в разных условиях.

2. Методы оценки качества

- 2.1. Экспертная проверка (ручная валидация) – Анализ знаний экспертом вручную.
- 2.2. Тестирование на контрольных примерах - Запуск системы на заранее подготовленных данных с известными ответами.
- 2.3. Логический анализ на непротиворечивость - Автоматическая проверка конфликтов в правилах.
- 2.4. Статистические методы - Анализ частоты использования правил и их вклада в результат.
- 2.5. Сравнение с эталонными системами (бенчмаркинг) - Сопоставление работы системы с признанными аналогами.
- 2.6. Оценка пользователями (User Testing) - Опросы, анкетирование, A/B-тестирование.
- 2.7. Методы машинного обучения (для динамических знаний) - Использование ML для выявления аномалий в знаниях.

21. Современные подходы к извлечению знаний, применяемые технологии

Современные подходы используют комбинацию классических методов и ИИ/ML.

Подходы:

- **Машинное обучение:** обучение на основе большого количества данных. Не требует прямого участия эксперта.
- **Data Mining (интеллектуальный анализ данных):** нахождение закономерностей в больших массивах данных.
- **Text Mining и NLP:** извлечение знаний из текстов с помощью обработки естественного языка.
- **Crowdsourcing:** сбор знаний от многих пользователей.
- **Обратное проектирование:** анализ поведения системы/человека и построение модели.

Технологии:

- **Языки онтологий:** OWL, RDF
- **Инструменты NLP:** SpaCy, NLTK
- **ML-фреймворки:** Scikit-learn, TensorFlow, PyTorch
- **Системы экспертного ввода:** Protégé (для онтологий), CLIPS
- **ChatGPT/LLM:** генерация и обобщение знаний из диалогов и текстов.

22. Технологии LIME / SHAP для объяснения моделей

Модели машинного обучения (такие как нейронные сети, машины опорных векторов, ансамбли решающих деревьев) являются "прозрачными" в том смысле, что все происходящие внутри них вычисления известны. Но тем не менее часто говорят, что модели машинного обучения плохо интерпретируемы. Здесь имеется в виду то, что процесс принятия решения не удается представить в понятной человеку форме, то есть:

1. Понять, какие признаки или свойства входных данных влияют на ответ

2. Разложить алгоритм принятия решения на понятные составные части
3. Объяснить смысл промежуточных результатов, если они есть
4. Описать в текстовом виде алгоритм принятия решения (возможно, с привлечением схем или графиков)

На помощь приходят такие технологии как LIME/SHAP.

LIME (Local Interpretable Model-agnostic Explanations):

- Объясняет локально, т.е. для одного конкретного случая.
- Берёт сложную модель и создаёт рядом с ней простую интерпретируемую модель на основе шумных данных.
- Например, объясняет, что доход и отсутствие просрочек повлияли на решение о выдаче кредита.

SHAP (SHapley Additive exPlanations):

- Использует теорию игр (значения Шепли) для оценки вклада каждого признака.
- Отключает/включает признаки, чтобы понять их влияние.
- Точные и строгие интерпретации. Например: Доход +0.3, История +0.4, Возраст -0.1.

Пример LIME (локальная интерпретация)

Задача:

Предсказать, одобряют ли кредит клиенту:

Признак	Значение клиента
Доход	80 000
Возраст	28 лет
Кредитная история	Без просрочек
Стаж работы	3 года
Сумма кредита	1 200 000

Модель предсказала: **Кредит ОДОБРЕН (вероятность = 0.82)**

Что делает LIME?

1. Берёт оригинальный пример (выше).
 2. Создаёт **похожие примеры**, слегка меняя признаки (например, доход = 75 000, стаж = 2 года).
 3. Прогоняет их через модель и получает ответы.
 4. Строит **простую линейную модель** по этим данным (вокруг оригинального примера).
 5. Говорит: **какие признаки повлияли и насколько**.
-

Интерпретация LIME:

Признак	Влияние на результат
---------	----------------------

Доход высокий	+0.4
---------------	------

Нет просрочек	+0.3
---------------	------

Стаж работы 3 года	+0.1
--------------------	------

Возраст < 30	-0.05
--------------	-------

Сумма кредита > 1 млн	-0.1
-----------------------	------

Вывод: модель решила одобрить кредит, потому что **высокий доход и отсутствие просрочек** сыграли наибольшую положительную роль.

Пример SHAP (значения Шепли)

SHAP объясняет результат на основе **вкладов каждого признака** в итоговое решение. Используется теория игр: сколько "очков" внёс каждый "игрок" (признак).

Те же данные:

- Доход: 80 000
- Возраст: 28
- Кредитная история: чистая
- Стаж: 3 года
- Сумма кредита: 1 200 000

SHAP разложит итоговую вероятность:

Базовая вероятность (до учёта признаков): 0.5

Финальный прогноз модели: 0.82

Признак	Вклад (SHAP value)
---------	--------------------

Доход высокий	+0.22
---------------	-------

Кредитная история чистая	+0.15
--------------------------	-------

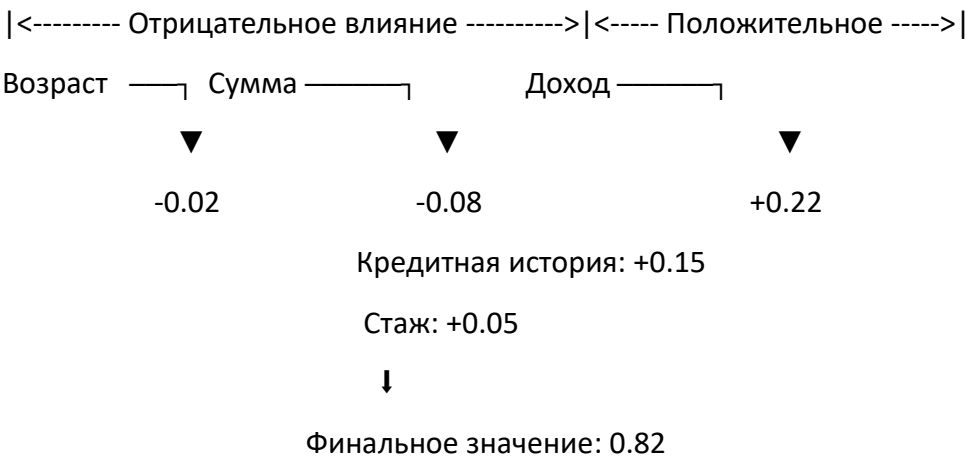
Стаж работы = 3 года	+0.05
----------------------	-------

Возраст < 30	-0.02
--------------	-------

Сумма кредита > 1 млн	-0.08
-----------------------	-------

Сумма вкладов: $+0.22 + 0.15 + 0.05 - 0.02 - 0.08 = +0.32$
Базовая вероятность 0.5 → финальная = $0.5 + 0.32 = \mathbf{0.82}$

График (примерно как выглядит в SHAP):



Сравнение LIME и SHAP:

Особенность	LIME	SHAP
Область действия	Локальная (вокруг одного примера)	Локальная и глобальная
Интерпретируемость	Высокая (простая модель)	Очень высокая (точный вклад)
Математическая строгость	Средняя	Высокая (основано на теории игр)
Скорость	Быстрая	Медленнее (особенно для сложных моделей)
Независимость от модели	Да	Да

23. Технология Decision Trees для генерации правил

Дерево решений — способ принятия решений, при котором задача разбивается на условия, ведущие к выводу.

Как используется:

- Каждый путь от корня до листа — правило ЕСЛИ-ТО.

Пример:

ЕСЛИ доход > 60 тыс. И возраст > 25 И нет просрочек ТО одобрить.

Преимущества:

- Легко читается и интерпретируется.
- Подходит для генерации базы знаний в CLIPS.

В **CLIPS** деревья решений могут быть реализованы через систему правил (production rules), которые автоматически строятся на основе данных.

Как Decision Trees интегрируются в CLIPS?

1. Построение дерева решений

- Используются алгоритмы (ID3, C4.5, CART), которые выбирают атрибуты для разбиения на основе энтропии или индекса Джини.
- **Пример:**

```
(deftemplate weather
```

```
  (slot outlook (type SYMBOL)) ; солнечно, дождливо, пасмурно
```

```
  (slot temperature (type NUMBER))
```

```
(deftemplate decision (slot go (type SYMBOL))) ; да/нет
```

Если outlook лучше всего разделяет данные, дерево начнётся с этого признака.

2. Генерация правил в CLIPS

- Каждое разбиение в дереве преобразуется в правило.

Пример:

```
(defrule sunny-weather
```

```
  (weather (outlook sunny) (temperature ?t > ?t 20)))
```

```
=>
```

```
  (assert (decision (go yes))))
```

3. Использование в экспертной системе

- Правила активируются при совпадении фактов в рабочей памяти.

24. Методы экспертного оценивания. Метод ранжирования

Экспертное оценивание — это процесс сбора и анализа мнений экспертов для принятия решений в условиях неполной информации. В интеллектуальных системах, таких как CLIPS, эти методы реализуются через правила, факты и алгоритмы обработки данных.

Метод ранжирования - это процедура упорядочивания альтернатив или критериев путем присвоения им порядковых номеров (рангов) в соответствии с их значимостью, где ранг 1 присваивается наиболее предпочтительному варианту.

Как это работает:

1. Задаются объекты (например, 5 факторов риска)

2. Каждый эксперт ранжирует их:
1 — самый важный,
5 — наименее важный
3. Ранжирования всех экспертов сводятся в таблицу
4. Считается среднее значение ранга — итоговое место

Пример:

5 факторов: A, B, C, D, E

Эксперт 1: A (1), B (2), C (3), D (4), E (5)

Эксперт 2: B (1), A (2), D (3), E (4), C (5)

И так далее...

Затем считаются средние ранги и делается вывод: какой фактор наиболее значимый.

Преимущества:

- Простота реализации.
- Позволяет быстро определить лидера.

Недостатки:

- Не учитывает степень различия между рангами.

25. Методы экспертного оценивания. Метод парных сравнений.

Экспертное оценивание — это процесс сбора и анализа мнений экспертов для принятия решений в условиях неполной информации. В интеллектуальных системах, таких как CLIPS, эти методы реализуются через правила, факты и алгоритмы обработки данных.

Метод парных сравнений - это процедура последовательного сравнения всех возможных пар альтернатив для определения их относительной важности с использованием стандартизированной шкалы оценок.

Пример:

$A > B$ (в 3 раза), $A < C$ (в 2 раза), $B > C$ (в 5 раз)

Создаётся матрица сравнений и вычисляются веса.

Преимущества:

- Точность и гибкость.

Недостатки:

- Количество сравнений растёт по формуле $n(n-1)/2$.

26. Методы экспертного оценивания. Метод АНР (Analytic Hierarchy Process)

Экспертное оценивание — это процесс сбора и анализа мнений экспертов для принятия решений в условиях неполной информации. В интеллектуальных системах, таких как CLIPS, эти методы реализуются через правила, факты и алгоритмы обработки данных.

АНР - это структурированная методика для организации и анализа сложных решений, основанная на построении иерархии критериев и последовательном применении метода парных сравнений на каждом уровне иерархии.

Структура АНР:

1. **Цель** — Например, выбрать поставщика
2. **Критерии** — Цена, Качество, Срок
3. **Альтернативы** — Поставщик А, В, С

Этапы:

1. Построить иерархию (Цель → Критерии → Альтернативы)
2. Провести парные сравнения между критериями → получить веса
3. Провести сравнение альтернатив по каждому критерию
4. Перемножить веса и получить итоговые оценки альтернатив

Пример:

- Цена важнее Качества в 2 раза → Вес "Цена" = 0.67, "Качество" = 0.33
- А лучше по цене, В — по качеству

→ А: итог = $0.67 \times (\text{оценка по цене}) + 0.33 \times (\text{оценка по качеству})$

27. Методы экспертного оценивания. Метод измерения степени влияния

Экспертное оценивание — это процесс сбора и анализа мнений экспертов для принятия решений в условиях неполной информации. В интеллектуальных системах, таких как CLIPS, эти методы реализуются через правила, факты и алгоритмы обработки данных.

Эксперты оценивают, насколько **один фактор влияет на другой**. Это позволяет строить **когнитивные карты** и выявлять ключевые звенья.

Как работает:

1. Список факторов (например, А, В, С, D)
2. Эксперты оценивают влияние **от А к В**, от В к С и т.д.
3. Оценки могут быть по шкале:
 - 0 — нет влияния
 - 1–10 — от слабого к сильному
4. Формируется **матрица влияний**
5. Анализируются:

- Какие факторы самые "влияющие"
- Какие — "зависимые"

Пример:

A B C

A - 7 3

B 0 - 9

C 2 1 -

→ B — самый влиятельный, так как сильно влияет на C

Применение:

- Системный анализ
- Построение стратегий
- Нахождение "узких мест"

28. Работа с неопределенностью в интеллектуальных системах

Интеллектуальные системы (ИС) часто работают в условиях, где:

- данные неполные или недостоверные;
- входная информация субъективна или размыта (например, «высокая температура», «молодой возраст»);
- эксперт не может точно сформулировать правила или значения.

Пример:

Доктор говорит: "Температура пациента — немного повышенная" → нельзя чётко сказать: 37.2 — это "болен" или "здоров"?

Для таких случаев нужны методы, способные **работать с неопределённостью**.

Основные подходы к работе с неопределённостью

1. Нечёткая логика (Fuzzy Logic)

Позволяет использовать **размытые (гибкие)** значения, а не бинарные (да/нет). Работает на основе **нечётких множеств**.

Определение: Нечёткое множество

Нечёткое множество — это множество, в котором **каждому элементу соответствует степень принадлежности** от 0 до 1.

Пример:

Множество "высокая температура":

Температура (°C)	Степень принадлежности к "высокая"
------------------	------------------------------------

36.5	0
------	---

37.0	0.3
------	-----

37.5	0.6
------	-----

38.0	1
------	---

— 36.5 — точно не высокая (0)

— 38.0 — точно высокая (1)

— 37.5 — **частично** высокая (0.6)

Это позволяет системе гибко рассуждать и делать выводы на основе размытых критериев.

Применение нечёткой логики:

- В экспертных системах для описания правил:

ЕСЛИ температура высокая ТО диагноз = лихорадка

- В CLIPS это реализуется через расширения (FuzzyCLIPS).
- Широко используется в бытовой технике: кондиционеры, стиральные машины, автомобили.

2. Байесовский подход

Использует **вероятности** для учёта неопределённости.

Например: «Вероятность болезни = 80%» — система принимает решения, учитывая не точный диагноз, а **степень уверенности**.

Поддерживает **обновление знаний** по мере получения новых данных (байесовское обновление).

3. Теория Демпстера — Шафера

Позволяет учитывать не только вероятность события, но и **меру неопределённости** (недостатка информации).

Пример:

- Уверенность, что пациент болен = 60%
- Уверенность, что здоров = 20%
- Остальные 20% — **неопределённость**

Эта теория более гибкая, чем вероятности, и полезна в ситуациях, когда **не всё известно**.

4. Интервалы и импутация

- Вместо конкретных чисел используются интервалы: "доход от 50 до 80 тыс".
- При отсутствии данных система делает приближённую оценку (импутацию) или работает с частично известной информацией.

Вывод

Подход	Что делает	Когда полезен
Нечёткая логика	Позволяет оперировать размытыми понятиями	Слова типа «высокий», «близкий», «малый»
Байесовские методы	Работа с вероятностями	При наличии статистики
Теория Шафера-Демпстера	Учитывает неполноту знания	Когда информации мало или она противоречива
Интервальные оценки	Уточняет диапазоны	Нет точных значений

29. Теория вероятностей в обработке неопределенностей

Теория вероятностей — один из базовых подходов к обработке неопределённости в инженерии интеллектуальных систем. Она используется, когда неопределённость связана с неполнотой или неточностью информации о событиях, которые можно охарактеризовать вероятностными зависимостями.

В интеллектуальных системах неопределённость может возникать по следующим причинам:

- Неполные или шумные данные
- Нечёткие или субъективные знания
- Стохастическое поведение внешней среды
- Неопределённые действия пользователя или системы

Применение теории вероятностей

Теория вероятностей позволяет моделировать и учитывать случайность при принятии решений. Основные формы её применения в интеллектуальных системах:

а) Байесовский подход

Использует априорные и апостериорные вероятности для обновления знаний при поступлении новых данных.

Основу составляет формула Байеса:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

где:

$P(H|E)$ — вероятность гипотезы H при наличии доказательства E (апостериорная вероятность),

$P(H)$ — априорная вероятность гипотезы (исходная вероятность события или гипотезы до получения новых данных (на основе предыдущих знаний или статистики)),

$P(E|H)$ — вероятность наблюдения E при условии гипотезы H ,

$P(E)$ — вероятность наблюдения E .

- Применяется, например, в диагностических системах, распознавании образов, интеллектуальных агентах.

б) Наивный байесовский классификатор

Простой и эффективный метод классификации, предполагающий условную независимость признаков.

Нужно классифицировать объект (например, письмо — спам или нет), имеющий набор признаков (слова, характеристики).

Для каждого возможного класса C считается вероятность $P(C|X)$, где X — вектор признаков объекта.

- Часто используется в задачах обработки естественного языка, спама.

в) Марковские модели

Используются, когда система развивается во времени, и текущее состояние зависит только от предыдущего — это свойство называется марковским.

Марковская цепь:

1. Последовательность состояний S_1, S_2, \dots, S_n

2. Вероятность перехода зависит только от текущего состояния:

$$P(S_{t+1}|S_t, S_{t-1}, \dots) = P(S_{t+1}|S_t)$$

- Применение: прогноз погоды, моделирование поведения пользователей и т.д.

г) Баесовские сети (графические модели вероятностей)

Баесовская сеть — это направленный ациклический граф (DAG), где:

1. Вершины представляют случайные величины (например, болезнь, симптом, температура),
2. Рёбра обозначают причинно-следственные или статистические зависимости между ними,
3. Каждая вершина имеет таблицу условных вероятностей (CPT), определяющую её поведение в зависимости от родителей.

- Применения: Медицина (диагностика по симптомам), экспертные системы (вывод в условиях неопределённости), робототехника (обработка сенсорных данных)

30. Интеллектуальная система на основе байесовских рассуждений. Ошибка байесовского метода

Интеллектуальная система на основе байесовских рассуждений

Это система, которая:

- Хранит знания в виде вероятностной модели (например, байесовской сети),
- Использует формулу Байеса для обновления вероятностей гипотез при поступлении новых данных,
- Принимает решения, выбирая наиболее вероятные гипотезы или действия.

Принцип работы:

1. Формируется множество возможных гипотез H_1, H_2, \dots, H_n
2. Получается новое наблюдение E
3. Для каждой гипотезы вычисляется апостериорная вероятность $P(H_i|E)$ (вероятность H_i при условии E). Система выбирает гипотезу с наибольшей вероятностью или действует в соответствии с вероятностным распределением.

Пример:

В медицине — по симптомам (наблюдениям, E) оцениваются вероятности заболеваний (гипотез, H), и предлагается наиболее вероятный диагноз.

Ошибка байесовского метода

Это наименьшая возможная ошибка классификации, которая может быть достигнута любым классификатором, если известны истинные априорные и условные вероятности.

Формально:

Байесовская ошибка — это вероятность того, что байесовский классификатор примет неверное решение, даже если всё задано идеально. Вычисляется по формуле:

$$\text{Bayes error} = 1 - \mathbb{E}_x \left[\max_i P(C_i | x) \right],$$

где:

- x — наблюдение (вектор признаков),
- C_i — классы (класс C — это множество объектов с общими свойствами, которым мы присваиваем одинаковую метку при классификации),
- $P(C_i|x)$ — апостериорная вероятность класса C при условии x

Пример:

Допустим, у нас есть два класса:

$$P(C1|x)=0.6$$

$$P(C2|x)=0.4$$

Мы выберем $C1$, но при этом в 40% случаев (в среднем) это будет ошибка — она и входит в байесовскую ошибку.

31. Опишите особенности правила вывода (modus ponens) в нечеткой логике

В классической логике правило Modus Ponens звучит так:

Если A истинно и $A \Rightarrow B$ (если A , то B ; операция называется импликация) истинно, то B истинно.

В нечеткой логике эта схема обобщается с учётом степеней принадлежности и неопределенности.

Вместо булевых истинных/ложных значений используются функции принадлежности $\mu_A(x)$, $\mu_B(y)$, которые показывают, насколько элементы x и y принадлежат нечетким множествам A и B .

Правило формулируется так:

Если $x \in A'$, то $y \in B'$, где A' и B' — нечеткие множества с функциями принадлежности $\mu_{A'}(x)$, $\mu_{B'}(y)$.

Вычисление вывода через функцию принадлежности импликации

Итоговая степень принадлежности $\mu_B(y)$ вычисляется по формуле:

$$\mu_B(y) = \sup_{x \in U} \min(\mu_A(x), \mu_{A' \Rightarrow B'}(x, y))$$

Здесь:

U — универсальное множество возможных x ,

$\mu_A(x)$ — степень принадлежности наблюдаемого факта x к множеству A (посылка),

$\mu_{A' \Rightarrow B'}(x, y)$ — это функция двух аргументов, дающая степень истинности импликации «если $x \in A$, то $y \in B$ ».

$\mu_B(y)$ — степень принадлежности y к множеству B (следствие)

На практике часто исследуется конкретное значение x_0 , тогда формула упрощается:

$$\mu_B(y) = \min(\mu_A(x_0), \mu_{A' \Rightarrow B'}(x_0, y))$$

В отличие от классической логики, где вывод либо есть, либо нет, нечеткий Modus Ponens позволяет получить степень уверенности в выводе. Это особенно важно при работе с неопределенностью и нечеткими данными.

32. Теория факторов определенности и доказательные рассуждения

Теория факторов определённости (CF)

Теория факторов определённости предназначена для количественной оценки степени уверенности в гипотезах при наличии неопределённой, неполной и противоречивой информации.

В отличие от классической вероятности, CF позволяет работать с частичной уверенностью, выраженной числом в диапазоне $[-1, 1]$.

Основные компоненты

- Фактор доверия (MB — Measure of Belief) — степень подтверждения гипотезы; $MB \in [0, 1]$.
- Фактор недоверия (MD — Measure of Disbelief) — степень опровержения гипотезы; $MD \in [0, 1]$.

Расчёт фактора определённости

$$CF(h; E) = \frac{MB(h; E) - MD(h; E)}{1 - \min(MB(h; E), MD(h; E))}$$

Значения CF варьируются от -1 (полное опровержение) до +1 (полное подтверждение).

Объединение (слияние) нескольких факторов определённости

Пусть есть два фактора определённости CF1 и CF2, полученные из разных правил или доказательств, поддерживающих или опровергающих одну гипотезу.

Если оба положительны ($CF_1 > 0, CF_2 > 0$):

$$CF_1 + CF_2 \times (1 - CF_1)$$

Если оба отрицательны ($CF_1 < 0, CF_2 < 0$):

$$CF_{combined} = CF_1 + CF_2 \times (1 + CF_1)$$

Если знаки разные (один положительный, другой отрицательный):

$$CF_{combined} = \frac{CF_1 + CF_2}{1 - \min(|CF_1|, |CF_2|)}$$

Доказательные рассуждения (Теория Дэмпстера-Шафера)

Представляет собой расширение классической теории вероятностей, позволяющее работать с неопределённостью и неполнотой информации.

Вместо точных вероятностей вводятся меры массы доверия, которые распределяются по множествам гипотез.

Позволяет объединять множественные источники информации, учитывая степень согласованности и конфликта.

Важны понятия belief (уверенность) и plausibility (правдоподобие), которые задают интервалы вероятности.

33. Сравнение байесовских рассуждений и коэффициентов определенности

Критерий	Байесовские рассуждения	Коэффициенты определённости (CF)
Основная идея	Вычисление апостериорной вероятности гипотезы через формулу Байеса, используя априорные вероятности и данные	Количественная оценка степени поддержки или опровержения гипотезы в диапазоне от -1 до 1
Представление неопределённости	Вероятности в диапазоне [0,1], где 1 — полная уверенность, 0 — полное опровержение	CF от -1 до 1, где положительные значения — поддержка, отрицательные — опровержение, 0 — неопределённость
Требования к исходным данным	Необходимы априорные вероятности и точные вероятности условных событий	Требуются меры доверия (MB) и недоверия (MD), которые могут быть экспертными оценками
Обработка конфликтов	Формально учитывается через вероятностные модели, но конфликтующая информация может усложнять расчёты	Специальные формулы для объединения CF учитывают конфликт и неопределённость
Интерпретируемость	Иногда сложна для понимания без статистического образования	Интуитивно понятна: положительные и отрицательные значения отражают поддержку или опровержение

Критерий	Байесовские рассуждения	Коэффициенты определённости (CF)
Область применения	Широко применяется в статистике, машинном обучении, прогнозировании	Используется в экспертных системах, где вероятности трудно получить напрямую
Комбинирование доказательств	Комбинируются через формулы вероятностей и правила условной независимости	Используются формулы объединения CF с учётом знаков и величин факторов
Вычислительная сложность	Может требовать сложных вычислений, особенно при больших пространствах событий	Относительно просты в вычислении и реализации

34. Байесовские сети. Основные определения

Байесовская сеть (Bayesian Network)

Это вероятностная графическая модель, которая представляет совместное распределение вероятностей множества случайных величин.

Формально — ориентированный ациклический граф (DAG), где:

- Узлы — переменные.
- Рёбра — условные зависимости между переменными.

1. Вершина (Node)

Каждый узел графа соответствует переменной (например, событию, признаку).

2. Дуги (Edges)

Направленные рёбра, показывающие прямое влияние или причинно-следственную связь от родительской переменной к дочерней.

3. Родители узла (Parents)

Набор узлов, у которых есть дуги, направленные в данный узел. Обозначают причины

4. Дочерний узел (Child)

Узел, в который входит дуга из другого узла (родителя). Обозначают следствия

5. Условная вероятность (Conditional Probability)

Для каждого узла задана таблица условных вероятностей (CPT — Conditional Probability Table), описывающая вероятность данного узла при фиксированных значениях родителей.

6. Марковское свойство

Нет дуги в сети - нет причинно-следственной связи. Это упрощает вычисление совместной вероятности.

7. Совместное распределение вероятностей (Joint Probability Distribution)

Совместная вероятность всех переменных равна произведению условных вероятностей каждого узла при его родителях:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i \mid \text{Parents}(X_i))$$

35. Рассуждения с помощью байесовских сетей

Байесовская сеть — это направленный ациклический граф, где:

- узлы — случайные переменные;
- дуги — вероятностные зависимости (причинно-следственные связи);
- каждая переменная имеет таблицу условных вероятностей (CPT), отражающую влияние родителей.

Сеть описывает совместное распределение всех переменных. Поддерживает рассуждение:

- от причины к следствию (прогнозирование),
- от следствия к причине (диагностика),
- комбинированное рассуждение.

Модель, описывающая возможные причины чихания:

Простуда Аллергия

\ /

\ /

Насморк

↓

Чихание

Переменные

C: Простуда (cold)

A: Аллергия (allergy)

R: Насморк (runny nose)

S: Чихание (sneeze)

Каждая переменная принимает два значения: true (1) или false (0).

Вероятности

Априорные вероятности

Переменная Значение Вероятность

Переменная Значение Вероятность

C 1 0.2

C 0 0.8

A 1 0.1

A 0 0.9

$P(R|C, A)$

C A $P(R = 1)$

0 0 0.05

0 1 0.8

1 0 0.9

1 1 0.95

$P(S|R)$

R $P(S = 1)$

0 0.1

1 0.7

Пример рассуждения

Дано: наблюдается чихание ($S = 1$).

Найти: вероятность простуды, то есть $P(C = 1 \mid S = 1)$

1. Применим формулу Байеса:

$$P(C = 1|S = 1) = \frac{P(C = 1, S = 1)}{P(S = 1)}$$

Вычислим числитель и знаменатель.

2. Расчёт $P(C, S)$

Вычислим по формуле:

$$P(C, S) = \sum_A \sum_R P(C) \cdot P(A) \cdot P(R|C, A) \cdot P(S|R)$$

Для $C = 1$

A	R	$P(C) \cdot P(A) \cdot P(R C,A) \cdot P(S R)$
1	1	$0.2 \times 0.1 \times 0.95 \times 0.7 = 0.0133$
1	0	$0.2 \times 0.1 \times 0.05 \times 0.1 = 0.0001$
0	1	$0.2 \times 0.9 \times 0.9 \times 0.7 = 0.1134$
0	0	$0.2 \times 0.9 \times 0.1 \times 0.1 = 0.0018$

Сумма:

$P(C=1, S=1) = 0.0133 + 0.0001 + 0.1134 + 0.0018 = 0.1286$ (нашли числитель, но еще понадобится для вычисления знаменателя)

3. Для вычисления безусловной вероятности в знаменателе ($P(S=1)$) нам нужно вычислить ее вероятность при простуде $P(C=1, S=1)$ (уже вычислили предыдущим шагом) и без нее $P(C=0, S=1)$, а затем сложить.

Для $C = 0$

A	R	$P(C) \cdot P(A) \cdot P(R C,A) \cdot P(S R)$
1	1	$0.8 \times 0.1 \times 0.8 \times 0.7 = 0.0448$
1	0	$0.8 \times 0.1 \times 0.2 \times 0.1 = 0.0016$
0	1	$0.8 \times 0.9 \times 0.05 \times 0.7 = 0.0252$
0	0	$0.8 \times 0.9 \times 0.95 \times 0.1 = 0.0684$

Сумма:

$P(C=0, S=1) = 0.0448 + 0.0016 + 0.0252 + 0.0684 = 0.14$

4. Финальный расчёт

$P(S=1) = P(C=1, S=1) + P(C=0, S=1) = 0.1286 + 0.14 = 0.2686$

$P(C=1|S=1) = 0.1286 / 0.2686 \approx 0.479$

Вывод:

Вероятность того, что у человека простуда, если он чихает: $\approx 47.9\%$.

36. Понимание байесовских сетей. Представление совместного распределения вероятностей

Байесовская сеть — это направленный ациклический граф (DAG), где:

- Узлы — это переменные
- Рёбра — зависимости (причины \rightarrow следствия, от узла к узлу)

- Каждая переменная описана **таблицей условных вероятностей (CPT - таблица, в которой указаны вероятности значений переменной, в зависимости от значений её родителей в байесовской сети)**

Сеть кодирует, как связаны вероятности разных переменных, и позволяет **вычислять любые вероятности по частичной информации** (по причинно-следственной связи).

Математически вероятность того, что записано в узле, пишется так: $P(A | \text{<parents>})$, где A – событие в узле, а <parents> - родители.

Совместное распределение вероятностей

Совместное распределение вероятностей — это полная таблица всех возможных комбинаций значений нескольких случайных величин и соответствующих им вероятностей (на понимание: есть переменные со значениями да/нет, поэтому строится таблица со всеми этими переменными, где перебираются абсолютно все исходы (все значения переменных перебираются)).

Отсюда проблема, например, 2 значения для 10 переменных это 1024 исхода (строки в таблице).

Поэтому ввели формулу совместной вероятности:

Формула:

Если у нас есть переменные X_1, X_2, \dots, X_n , то:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | \text{Parents}(X_i))$$

То есть:

совместная вероятность равна произведению локальных условных вероятностей, указанных в CPT (таблицах условных вероятностей).

37. Понимание байесовских сетей. Алгоритм Перла построения сети

(!!!Понимание байесовских сетей в предыдущем вопросе!!!)

Алгоритм Перла построения сети

Главная идея: разбить совместное распределение на условные вероятности и выразить их через локальные связи.

Перл разработал алгоритм причинной инференции, а также теорию, как определить структуру сети по зависимостям между переменными.

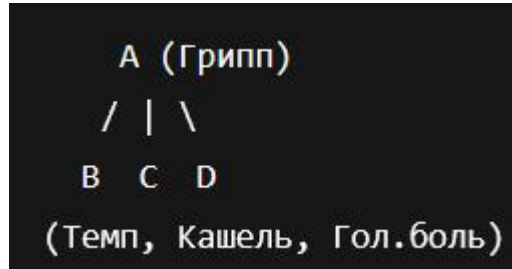
Этапы построения сети по Перлу:

- 1) **Определение переменных**
- 2) **Сбор знаний/данных**
- 3) **Определение зависимостей (определяется, какие переменные влияют друг на друга)**
- 4) **Создание направленного графа (DAG)**

5) Применение правила d-сепарации (проверка независимости между переменными при заданных условиях)

d-сепарация — ключевое правило Перла, определяющее, какие переменные **условно независимы**.

Пример:



- В и С зависят друг от друга через А, если мы не знаем А.
- Но если мы знаем А, то $B \perp C \mid A$ — т.е. температура и кашель **независимы**, если известна причина (грипп).

Это позволяет **меньше хранить** информации, упрощает таблицы вероятностей и вывод.

6) Построение таблиц СРТ

38. Условная независимость в байесовских сетях.

Переменные А и В **условно независимы при известной переменной С**, если знание В не добавляет никакой новой информации о А, когда уже известна С.

Обозначается:

$$A \perp B \mid C$$



То есть:

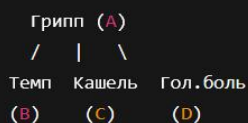
$$P(A \mid B, C) = P(A \mid C)$$

Вместо того чтобы хранить огромное совместное распределение, мы разбиваем его на более простые **локальные зависимости**, используя условную независимость.

 **Пример (тот же, что раньше):**

mathematica

 Копировать  Редактировать



В этой сети:

- В и С **условно независимы**, если известен А.
 $\rightarrow B \perp C \mid A$

То есть, если ты знаешь, есть ли грипп, то наличие температуры не даёт новой информации о кашле.

39. Вывод в байесовских сетях. Точные выводы в цепочках и в полидереве.

Вывод — это **вычисление апостериорной вероятности** интересующих нас переменных при известных доказательствах (короче, вычисление вероятности по формуле Байеса, как я понял).

Точный вывод в цепочке

Цепочка — это линейная структура: $A \rightarrow B \rightarrow C \rightarrow D$.

Стоит задача — найти $P(D \mid A)$.

Решение:

📌 Шаги (точный вывод по правилу Байеса + маргинализация):

1. Используем цепное правило Байеса:

$$P(A, B, C, D) = P(A) \cdot P(B|A) \cdot P(C|B) \cdot P(D|C)$$

2. Используем маргинализацию, чтобы вычислить нужную вероятность:

$$P(D \mid A) = \sum_{B, C} P(D \mid C) \cdot P(C \mid B) \cdot P(B \mid A)$$

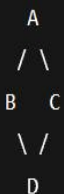
(Я хз как объяснить)

Точный вывод в полидереве

Polytree — это байесовская сеть, в которой между любой парой узлов есть только один путь.

Пример:

CSS



Здесь:

- D зависит от B и C ,
- B, C зависят от A .

Используется алгоритм распространения убеждений (belief propagation) или алгоритм сообщений (message passing).

Каждый узел отправляет сообщения своим соседям:

- **вниз по сети** (от родителей к потомкам),
- **вверх** (от потомков к родителям).

Эти сообщения содержат:

- вероятности,
- информацию о доказательствах,
- промежуточные вычисления.

40. Вывод в байесовских сетях. Алгоритм передачи сообщений Кима и Перла

(!!!Вывод в байесовских сетях в вопросе 39!!!)

Алгоритм передачи сообщений Кима и Перла

Чтобы **эффективно вычислять апостериорные вероятности**:

$P(X|evidence)$

где:

- X — интересующая переменная,
- $evidence$ — известные значения других переменных.

Каждый узел в сети обменивается сообщениями со своими соседями, чтобы на основе локальных зависимостей можно было вычислить глобальные вероятности.

Этот процесс состоит из двух направлений:

1. Вверх по сети (снизу вверх) — от листьев к корню → *сбор информации от потомков*
2. Вниз по сети (сверху вниз) — от корня к листьям → *распространение информации от родителей*

Сообщение	Обозначение	Куда направляется	Что означает
π-сообщение (π_i)	$\pi(X)$	от родителей к X	"Ожидания" от родителей (приорная инфо)
λ-сообщение (λ)	$\lambda(X)$	от потомков к X	"Доказательства" от потомков

41. Вывод в байесовских сетях. Точные выводы в многосвязных сетях

(!!!Вывод в байесовских сетях в вопросе 39!!!)

Многосвязные сети

Многосвязная сеть (multiply connected network) — это байесовская сеть, в которой существует более одного пути между некоторыми парами узлов.

Пример:

CSS

Копировать Редактировать

```
A → C ← B
  \      /
   → D ←
```

Тут C и D зависят как от A, так и от B. Между A и B есть два пути через C и D → сеть **многосвязная**.

В многосвязных сетях:

- Сообщения могут "ходить по кругу"
- Узлы зависят от **нескольких цепей** информации

42. Математика нечетких множеств

В классической теории множеств элемент либо принадлежит множеству, либо нет:

$\mu_A(x) \in \{0,1\}$

- $\mu_A(x)=1$ — элемент x принадлежит множеству A
- $\mu_A(x)=0$ — элемент x не принадлежит множеству A

В нечетких множествах элемент принадлежит множеству с некоторой степенью принадлежности:

$\mu_A(x) \in [0,1]$

- Значение — **функция принадлежности** (membership function)
- Отражает **насколько элемент x подходит под понятие A**

Один и тот же объект или значение одновременно принадлежит нескольким нечетким множествам с разной степенью принадлежности:

Например, температура 18°C может быть:

- **Холодной** с принадлежностью 0.65
- **Тёплой** с принадлежностью 0.30
- **Жаркой** с принадлежностью 0.05

Это не противоречит — наоборот, это **главная особенность нечетких множеств**. В отличие от классической логики, где температура либо холодная, либо нет, здесь может быть и то, и другое одновременно, но в разной степени.

43. Нечеткий логический вывод. Композиционное правило.

Нечеткий логический вывод — это процесс получения вывода из нечетких правил вида:

Если $X — A$, то $Y — B$,

где A и B — нечеткие множества.

Композиционное правило используется для объединения входных данных с базой правил. Оно формулируется как:

$$B' = A' \circ R,$$

где A' — степень принадлежности входа к нечеткому множеству A , R — нечеткое отношение (правило), а B' — выходной нечеткий результат.

Композиция (\circ) обычно выполняется через **максимум-минимум** или **максимум-произведение**.

44. Объясните возможность управления неопределенностью с помощью нечеткого управления.

Нечеткое управление позволяет работать с неточными, частично известными или лингвистическими данными (например, "температура высокая").

Вместо строгих границ (как в классической логике), используются **нечеткие множества** и **лингвистические переменные**, что позволяет системе реагировать гибко и устойчиво даже при неопределённости входов.

Это особенно полезно в системах, где точные математические модели трудно построить (например, климат-контроль, автопилоты).

45. Проиллюстрируйте упрощенный алгоритм нечеткого вывода.

Упрощённый алгоритм состоит из следующих шагов:

1. **Фаззификация** — преобразуем чёткие входные данные в степени принадлежности нечетким множествам.
2. **Применение базы правил** — вычисляем результат для каждого правила (обычно через \min или произведение).
3. **Агрегация** — объединяем результаты всех правил (чаще всего через \max).
4. **Дефаззификация** — преобразуем итоговый нечеткий вывод в чёткое значение (например, через метод центра тяжести).

46. Приведите методы приведения к четкости в алгоритмах нечеткого вывода.

Приведение к четкости (дефаззификация) — последний этап нечеткого вывода. Основные методы:

- **Метод центра тяжести (центроид)** — находит "центр массы" результирующей функции принадлежности. Наиболее распространён.
- **Метод максимума (max)** — выбирает значение с максимальной степенью принадлежности.
- **Среднее значение максимума (SOM)** — берётся среднее из всех значений, имеющих максимальную степень.
- **Метод взвешенного среднего** — используется, когда правила выдают конкретные числа.

47. Понятие Байесовских сетей доверия.

Байесовская сеть — это **граф**, в котором узлы — это случайные переменные, а дуги — условные зависимости между ними.

Каждая переменная зависит от своих "родителей" по графу. У каждой переменной есть **таблица вероятностей**, задающая распределение вероятностей при известных значениях родителей.

Сеть позволяет оценивать **априорные** и **апостериорные вероятности**, проводить анализ и делать выводы в условиях неопределенности.

48. Управление неопределенностью с помощью Байесовского подхода.

Байесовский подход использует **теорему Байеса** для обновления знаний при поступлении новой информации.

Он позволяет:

- учитывать **предыдущие знания** (априорные вероятности);
- корректировать их на основе **новых данных**;
- работать с **неполной** или **шумной информацией**.

Этот метод особенно полезен в диагностике, прогнозировании, распознавании образов и принятии решений под риском.

Теорема Байеса:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

49. Изложите основные положения нечеткой логики. Дайте определение лингвистической переменной.

Нечеткая логика — это расширение классической логики, в которой **истинность** может быть не только 0 или 1, но **любым значением от 0 до 1**.

Основные положения:

- Используются **нечеткие множества** с функциями принадлежности.
- Операции "и", "или", "не" заменены на **t-нормы и s-нормы**.
- Позволяет формулировать правила в виде **"если-то"** на естественном языке.

Лингвистическая переменная — это переменная, чьи значения описываются **словами**, а не числами. Например:

Температура = {низкая, средняя, высокая}, где каждое значение — нечеткое множество.

50. Опишите особенности правила вывода (modus ponens) в нечеткой логике (идентичен 31 вопросу)

В классической логике правило Modus Ponens звучит так:

Если А истинно и $A \Rightarrow B$ (если А, то В; операция называется импликация) истинно, то В истинно.

В нечеткой логике эта схема обобщается с учётом степеней принадлежности и неопределенности.

Вместо булевых истинных/ложных значений используются функции принадлежности $\mu_A(x)$, $\mu_B(y)$, которые показывают, насколько элементы x и y принадлежат нечетким множествам A и B .

Правило формулируется так:

Если $x \in A'$, то $y \in B'$, где A' и B' — нечеткие множества с функциями принадлежности $\mu_{A'}(x)$, $\mu_{B'}(y)$.

Вычисление вывода через функцию принадлежности импликации

Итоговая степень принадлежности $\mu_B(y)$ вычисляется по формуле:

$$\mu_B(y) = \sup_{x \in U} \min(\mu_A(x), \mu_{A' \Rightarrow B'}(x, y))$$

Здесь:

U — универсальное множество возможных x ,

$\mu_A(x)$ — степень принадлежности наблюдаемого факта x к множеству A (посылка),

$\mu_{A' \Rightarrow B'}(x, y)$ — это функция двух аргументов, дающая степень истинности импликации «если $x \in A$, то $y \in B$ ».

$\mu_B(y)$ — степень принадлежности y к множеству B (следствие)

На практике часто исследуется конкретное значение x_0 , тогда формула упрощается:

$$\mu_B(y) = \min(\mu_A(x_0), \mu_{A' \Rightarrow B'}(x_0, y))$$

В отличие от классической логики, где вывод либо есть, либо нет, нечеткий Modus Ponens позволяет получить степень уверенности в выводе. Это особенно важно при работе с неопределенностью и нечеткими данными.

51. Нечеткий логический вывод: композиционное правило

Нечеткий логический вывод — это процесс получения выводов из неточных или нечетких данных, и композиционное правило лежит в его основе. Оно связывает входные нечеткие множества (предпосылки) с выходными через нечеткие отношения. Формально, если есть правило "если A , то B " и вход A' (похожий, но не идентичный A), то вывод B' вычисляется как композиция A' с отношением, заданным правилом. Обычно используют операцию **max-min** или **max-prod**.

Как это работает?

- Представим нечеткое правило: "Если температура высокая, то вентилятор работает на полную".
- "Высокая температура" — это нечеткое множество с функцией принадлежности, например: $30^\circ\text{C} = 0.8$, $25^\circ\text{C} = 0.2$.
- Вход: "температура средняя" ($25^\circ\text{C} = 0.7$ в "средней", 0.3 в "высокой").

- Композиция: для каждого значения выхода (например, "вентилятор на полную") берём минимум степени совпадения входа с условием правила и максимум по всем возможным связям. Итог — нечеткое множество B' , показывающее, насколько сильно включается вентилятор.

Пример в цифрах:

- Правило: $\mu_{\text{высокая}}(25^{\circ}\text{C}) = 0.3$, $\mu_{\text{полная}}(100\%) = 1$.
- Вход: $\mu_{\text{средняя}}(25^{\circ}\text{C}) = 0.7$.
- Степень активации = $\min(0.3, 0.7) = 0.3$.
- Вывод: вентилятор работает на 30% от "полной" мощности.

Это делает нечеткую логику мощным инструментом для систем, где данные неоднозначны, а решения нужно принимать плавно.

52. Управление неопределенностью с помощью нечеткого управления

Нечеткое управление (fuzzy control) — это подход, который позволяет системам работать с неопределённостью, неточными данными и отсутствием строгих моделей. Оно переводит качественные, лингвистические знания (например, "слишком жарко") в количественные управляющие сигналы. Нечеткие контроллеры используют правила вида "если X, то Y", где X и Y — нечеткие множества, а не точные значения.

Почему это работает?

- Нечеткость моделирует реальный мир, где границы между состояниями размыты.
- Оно устойчиво к шуму и не требует точных измерений.
- Легко интегрирует экспертные знания.

Пример из жизни:

- В стиральной машине нечеткий контроллер оценивает "загрузку" (мало, средне, много) и "загрязнённость" (слабая, сильная). Если загрузка "средняя" (0.6) и загрязнённость "сильная" (0.8), правило "если загрузка средняя и загрязнённость сильная, то скорость барабана высокая" активируется с силой $\min(0.6, 0.8) = 0.6$. Итог — барабан крутится на 60% от максимума, экономя энергию и воду.

Где ещё?

- Кондиционеры: регулируют температуру плавно, а не скачками.
- Автомобили: адаптивный круиз-контроль с учетом "близости" препятствий.

Нечеткое управление — это мост между человеческим мышлением и машинной логикой, идеальный для сложных и неопределённых систем.

53. Упрощённый алгоритм нечеткого вывода

Упрощённый алгоритм нечеткого вывода — это практическая реализация нечеткой логики, вдохновлённая тем, как люди принимают решения на основе интуиции и приблизительных оценок. Он состоит из четырёх ключевых шагов:

1. Фаззификация

Точные входные данные преобразуются в нечеткие множества. Например, температура 27°C может быть "средней" с $\mu = 0.6$ и "высокой" с $\mu = 0.4$, в зависимости от заданных функций принадлежности (например, треугольных или гауссовых).

2. Оценка правил

Проверяются все правила базы знаний. Для правила "если температура высокая, то вентилятор на полную" степень активации = $\mu_{\text{высокая}}(27^\circ\text{C}) = 0.4$. Если правил несколько, каждое вносит вклад в итоговый вывод.

3. Агрегация

Выходы всех правил объединяются в одно нечеткое множество. Например, если одно правило говорит "вентилятор на полную" ($\mu = 0.4$), а другое "вентилятор на среднюю" ($\mu = 0.6$), агрегация (обычно \max) строит общую кривую возможных скоростей вентилятора.

4. Дефаззификация

Нечеткое множество преобразуется в точное значение (см. вопрос 54). Например, метод центраоида может дать "скорость вентилятора = 65%".

Почему это круто?

- Алгоритм гибкий: работает даже с частичными или шумными данными.
- Простой: не требует сложных уравнений, только правила и функции.
- Человечный: имитирует экспертное мышление, где точность вторична.

Пример: Контроль освещения в комнате — "если света мало, то лампа ярче". Алгоритм плавно регулирует яркость, даже если датчик света слегка врёт.

54. Методы приведения к четкости (дефаззификация)

Дефаззификация — это финальный шаг нечеткого вывода, превращающий нечеткое множество в конкретное число для управления системой. Вот основные методы с деталями:

• Центроид (центр тяжести) (COG)

Самый популярный метод. Вычисляет центр масс под кривой функции принадлежности.

Формула:

$$x^* = \frac{\int x \cdot \mu(x) dx}{\int \mu(x) dx} \text{ (для непрерывных)} \text{ или } x^* = \frac{\sum x_i \cdot \mu(x_i)}{\sum \mu(x_i)} \text{ (для дискретных)}.$$

Плюс: учитывает всю форму множества. Минус: вычислительно сложнее.

• Среднее максимумов (MOM)

Берёт среднее значение точек, где μ достигает максимума. Если $\mu_{\max} = 0.8$ на диапазоне $[60, 80]$, то $x^* = (60+80)/2 = 70$ $x^* = (60 + 80) / 2 = 70$ $x^* = (60+80)/2 = 70$.

Плюс: простой. Минус: игнорирует "хвосты" множества.

• Биссектриса

Делит площадь под кривой пополам. Если нечеткое множество — треугольник с базой $[0,$

100] и пиком $\mu = 1$ на 50, то $x^* = 50$ $x^{\wedge*} = 50$ $x^* = 50$.

Плюс: баланс между частями множества. Минус: сложнее вычислять для асимметричных форм.

- **Взвешенное среднее**

Для дискретных множеств: $x^* = \sum (x_i \cdot \mu(x_i)) / \sum \mu(x_i)$. Например, скорость 50 с $\mu = 0.5$ и 70 с $\mu = 0.7$ даёт $x^* = (50 \cdot 0.5 + 70 \cdot 0.7) / (0.5 + 0.7) = 63.3$.

Плюс: быстрота. Минус: только для дискретных данных.

Что выбрать? Центроид — для плавности, MOM — для скорости, биссектриса — для баланса. Зависит от задачи и формы множества.

55. Основные положения теории Демпстера-Шафера

Теория Демпстера-Шафера (D-S) — это математический аппарат для работы с неопределённостью, расширяющий классическую теорию вероятностей. Она особенно полезна, когда данных мало или они противоречивы.

Ключевые идеи:

- **Множественная неопределённость:** Вместо вероятностей событий используются *массовые функции* (m), которые распределяют доверие между подмножествами гипотез. Например, $m(\{A, B\}) = 0.6$ значит, что 60% доверия отдано комбинации A или B, но не конкретно A или B.

-

Правило комбинации Демпстера: Объединяет данные от разных источников. Если источник 1 даёт $m_1(A) = 0.7$, а источник 2 — $m_2(A) = 0.5$, то итоговое доверие корректируется с учётом конфликтов:

- $m(A) = \frac{m_1(A) \cdot m_2(A)}{1 - K}$, где K — мера конфликта.

Ра

деление неопределённости: Отличает "неизвестность" (нет данных) от "невозможности" (данные против). Например, $m(\emptyset) = 0$, а $m(\text{все гипотезы}) = 0.3$ — это неизвестность.

Пример:

- Диагностика: симптомы указывают на грипп (0.6) или простуду (0.4), но $m(\{\text{грипп, простуда}\}) = 0.2$ допускает, что это может быть что-то из двух.
- Комбинация с тестом (грипп = 0.8) усиливает доверие к гриппу.

D-S идеальна для экспертных систем, где нужно объединять мнения и учитывать неполноту знаний.

56. Команды обработки фактов в CLIPS

CLIPS — это язык для создания экспертных систем, где факты — это утверждения в базе знаний. Команды обработки фактов позволяют манипулировать ими динамически:

- **(assert <факт>)**
Добавляет факт. Пример: (assert (temperature 30)) — температура 30°C теперь в базе, и правила могут сработать.
- **(retract <индекс>)**
Удаляет факт по его номеру. Если (temperature 30) — факт №5, то (retract 5) убирает его, отключая связанные правила.
- **(modify <индекс> <слот> <значение>)**
Изменяет факт. (modify 5 (temperature 25)) обновляет температуру до 25°C, сохраняя структуру факта.
- **(duplicate <индекс> <слот> <значение>)**
Создаёт новый факт-копию с изменениями. (duplicate 5 (temperature 35)) добавляет новый факт, не трогая старый.

Зачем это нужно?

- Динамика: база знаний меняется в реальном времени (например, датчики обновляют данные).
- Гибкость: экспертная система адаптируется к новым условиям.

Пример: Система контроля климата добавляет (assert (humidity high)), а при изменении — (modify 3 (humidity low)).

57. Функциональные составные части правил в CLIPS

Правила в CLIPS — это ядро экспертных систем, работающее по принципу "если-то". Их структура:

- **LHS (Left-Hand Side) — условия**
Паттерны, которые должны совпасть с фактами. Пример: (temperature ?t&:(> ?t 25)) — срабатывает, если температура > 25°C. Может быть несколько условий с логическими связками (and, or, not).
- **RHS (Right-Hand Side) — действия**
Что делать при совпадении. (assert (fan on)) — включить вентилятор. Возможны и сложные действия: вызов функций, вывод сообщений.
- **Salience (приоритет)**
Число от -10000 до 10000, определяющее порядок срабатывания. (defrule high-temp (temperature ?t&:(> ?t 30)) => (assert (fan on)) (salience 10)) — приоритет 10.
- **Комментарий**
Текст в кавычках: "Включить вентилятор при жаре". Удобно для отладки.

Пример правила:

```
(defrule overheat
  (temperature ?t&:(> ?t 35))
  (humidity high)
  =>
```



```
(assert (fan full-speed))  
  
(printout t "Жара и влажность — вентилятор на максимум!" crlf)  
  
(salience 20))
```

Если температура > 35°C и влажность высокая, вентилятор включается на полную.

Особенность: Правила модульные, легко читаются и масштабируются, что делает CLIPS мощным инструментом.

58. Предложите доступные стратегии разрешения конфликтов при использовании правил в CLIPS

Во время выполнения система сопоставляет условия всех правил с текущими фактами в рабочей памяти. Если условия некоторого правила выполняются, такое правило считается активным и добавляется в конфликтный набор (conflict set).

Когда в конфликтном наборе находится несколько правил одновременно, система должна решить, какое правило применить первым. Эта ситуация называется конфликт правил.

Стратегии разрешения конфликтов

Для управления выбором правила из конфликтного набора CLIPS предоставляет встроенные стратегии разрешения конфликтов, которые можно задавать через:

```
(set-strategy <strategy-name>)
```

Каждая стратегия определяет, какое правило будет приоритетным, если сработало несколько.

Основные стратегии:

Стратегия	Суть
depth	По умолчанию. Выбирается последнее активированное правило (LIFO).
breadth	Выбирается первое активированное правило (FIFO).
lex	Лексикографический порядок: сначала предпочтение меньшему числу условий, затем — по структуре шаблонов.
mea	(Means-End Analysis) — выбирается правило, которое наиболее меняет рабочую память (вводит больше новых фактов).
complexity	Предпочтение более сложным правилам — с большим числом условий.
simplicity	Предпочтение более простым правилам — с меньшим числом условий.
random	Выбирается случайное правило из конфликтного набора.

Пример установки стратегии:

```
(set-strategy lex)
```

После этого система будет использовать лексикографическое сравнение для разрешения конфликтов.

59. Предложите варианты задания приоритета правил в CLIPS.

В CLIPS приоритет правил задаётся с помощью директивы `salience`. Это числовое значение, которое определяет порядок срабатывания правила в случае, если несколько правил одновременно активированы (то есть условия их срабатывания выполнены).

Синтаксис:

```
(defrule имя-правила
```

```
  (declare (salience N))
```

```
  (условия)
```

```
  =>
```

```
  (действия)
```

```
)
```

N — целое число (может быть как положительным, так и отрицательным). По умолчанию `salience` = 0.

Принцип работы:

- Правила с большим `salience` имеют больший приоритет и срабатывают раньше.
- Если несколько активных правил имеют одинаковый `salience`, тогда CLIPS применяет стратегию разрешения конфликтов (по умолчанию — `depth`).

60. Предложите варианты использования функций для ввода вывода информации в CLIPS.

CLIPS предоставляет ряд встроенных функций для ввода-вывода, которые позволяют взаимодействовать с пользователем и управлять выводом данных:

- `printout`: выводит текст в стандартный поток вывода или в файл.

- read: считывает данные из стандартного ввода.
- open: открывает файл для чтения или записи.
- close: закрывает файл, открытый ранее.

Эти функции могут быть использованы в правилах для диагностики или интерактивного управления ходом выполнения экспертной системы.

61. Варианты использования функций для работы со списками в CLIPS

CLIPS поддерживает функции для манипуляции со списками, что позволяет эффективно обрабатывать коллекции данных:

- a. create\$: создает новый список.
- b. insert\$: вставляет элемент в список на указанную позицию.
- c. delete\$: удаляет элемент из списка по его индексу.
- d. replace\$: заменяет элемент списка на новый элемент по индексу.
- e. nth\$: извлекает n-й элемент списка.
- f. member\$: проверяет наличие элемента в списке.
- g. length\$: возвращает длину списка.

Эти функции позволяют создавать и модифицировать списки данных, что полезно для управления сложными структурами данных в процессе выполнения правил.

62. Варианты использования функций для работы со строками в CLIPS

CLIPS включает функции для работы со строками, что обеспечивает манипулирование текстовой информацией:

- a. str-cat: конкатенация (слияние) двух и более строк.
- b. sub-string: извлечение подстроки из строки.
- c. str-length: определение длины строки.
- d. str-compare: сравнение строк.
- e. upcase: преобразование строки в верхний регистр.
- f. lowercase: преобразование строки в нижний регистр.
- g. str-index: поиск подстроки в строке.

Эти функции предоставляют мощные инструменты для обработки и анализа текстовых данных в экспертных системах.

63. Способы использования математических функций в CLIPS

Математические функции в CLIPS позволяют выполнять различные вычисления:

- a. Арифметические операции: +, -, *, /, mod.
- b. Функции округления: round, floor, ceil.

- c. Математические функции: sqrt, abs, exp, log, sin, cos, tan.

Эти функции могут быть использованы для вычислений в рамках правил, позволяя создавать более сложные и вычислительно зависимые системы.

64. Варианты применения циклических конструкций в CLIPS

В CLIPS циклы могут быть реализованы через конструкции loop и while:

- a. loop: выполняет вложенные команды до явного выхода из цикла с помощью return или break.
- b. while: аналогично языкам программирования общего назначения, выполняет вложенные команды до тех пор, пока заданное условие истинно.

Циклические конструкции полезны для реализации повторяющихся задач, таких как перебор элементов списка, выполнение повторяющихся расчетов или итеративное применение правил до достижения определенного условия.

65. Варианты реализации условных конструкций в CLIPS

CLIPS поддерживает условные конструкции, аналогичные тем, что используются в традиционных языках программирования:

- a. **if-then-else**: Стандартная конструкция, позволяющая выполнять различные действия в
 - b. (if (<variable> <condition>) then
 - (do-something) else
 - (do-something-else))
- c. **switch**: Позволяет выбрать действие из множества вариантов на основе значения переменной.
 - (switch <variable>
 - (case <value1> (do-action1))
 - (case <value2> (do-action2))
 - (default
 - (do-default-action)))

Эти конструкции упрощают написание сложной логики в правилах и функциях, позволяя системе адаптироваться к различным условиям и сценариям.

66. Способы применения абстрактных и конкретных классов в CLIPS

В CLIPS объектно-ориентированное программирование реализуется через систему классов и экземпляров. Вы можете определить как абстрактные, так и конкретные классы:

- a. **Абстрактные классы** используются как базовые шаблоны для других классов, не предполагая создания экземпляров этих классов напрямую.

```
(defclass Person (is-a USER) (role abstract)
(slot Name (type STRING)) (slot Age (type INTEGER)))
```

- b. **Конкретные классы** — это классы, из которых можно создавать экземпляры. Они могут наследовать свойства и методы от абстрактных классов.

```
(defclass Employee (is-a Person)
(role concrete)
(slot Salary (type INTEGER)) (slot Department (type STRING)))
```

Это позволяет строить сложные иерархии классов, поддерживающие переиспользование кода и инкапсуляцию.

67. Действия для управления объектами в CLIPS

Управление объектами в CLIPS включает создание, модификацию и удаление экземпляров классов:

- a. **Создание объекта:** используется функция `make-instance`.

```
(make-instance [JohnDoe] of Person (Name "John Doe") (Age 30))
```

- b. **Модификация объекта:** для изменения атрибутов существующих объектов используется функция `send`.

```
(send [JohnDoe] put-Age 31)
```

- c. **Удаление объекта:** для удаления экземпляра класса применяется функция `delete-instance`.

```
(delete-instance [JohnDoe])
```

Эти действия обеспечивают полный контроль над жизненным циклом объектов в вашей системе

