|  |  |  |  |
| --- | --- | --- | --- |
| ***1.Поняття «штучний інтелект». Тест Тьюрінга.***  Искусственный интеллект – область компьютерной науки, которая занимается автоматизацией разумного поведения.  Искусственный интеллект – область исследований, направленных на то чтобы заставить ЭВМ выполнять функции, которые трудны для них в настоящее время, а особенно те, которые способен выполнить человек.  Тест Тьюринга — эмпирический тест, идея которого была предложена Аланом Тьюрингом для того, чтобы определить, может ли машина мыслить. Тест Тьюринга сравнивает способности предположительно разумной машины со способностями человека.  В этом тесте машину и ее человеческого соперника(следователя) помещают в разные комнаты, отдельные от комнаты, в которой находится человек-имитатор. Следователь общается с ними исключительно с помощью текстововго устройсва (например, компьютерный терминал). Следователь должен отличить компьютер от человеа исключительно на основе их ответов на вопросы, задаваемые через это устройство. Если же следователь не может отличить машину от человека тогда машину можна считать разумной.  В тесте исключается предвзятость в пользу живых существ, заставляя опрашивающего сфокусироваться исключительно на содержании ответов на вопросы. | ***2. Залежності, які можуть бути виявлені методами Data Mining. Їх характеристика.***  Data-mining – добыча данных, совокупность методов обнаружения в данных ранее известных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности.  Выделяют пять стандартных типов закономерностей, которые позволяют выявлять методы Data Mining: ассоциация, последовательность, классификация, кластеризация и прогнозирование.  1)Ассоциация – имеет место в том случае, если несколько событий связаны друг с другом.  2)Последовательность – цепочка связанных во времени событий.  3)Классификация – с ее помощью определяются признаки, характеризующие группу, к которой принадлежит тот или иной обьект.  4)Кластеризация – отличается от классификации тем, что сами группы заранее не заданы. С помощью кластеризации средства Data Mining самостоятельно выделяют различные однородные группы данных.  5)Прогнозирование – на основе исторической информации можно предсказать как будет вести себя система в будущем. | ***3.Методи (засоби) Data Mining. (Стисло опишіть сутність кожного метода).***  Data-mining – добыча данных, совокупность методов обнаружения в данных ранее известных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности.  Методы Data-mining:  1)Статистический анализ – к базовым методам Data Mining традиционно причисляют все подходы, использующие элементы  теории статистики. Последние версии почти всех известных статистических пакетов включают  наряду с традиционными статистическими методами также элементы Data Mining.  2)Метод аналогичных сущностей – поиск подходящего случая в исторической базе данных текущему случаю. Если похожи – значит и результат должен быть аналогичным.  3)Метод «деревья решений» - они являются одним из наиболее популярных подходов к решению задачи классификации.Они создают иерархическую структуру классифицирующих правил типа "если-то" (if-then), имеющую вид дерева. Для принятия решения, к какому классу отнести некоторый объект или ситуацию, требуется ответить на вопросы, стоящие в узлах этого дерева, начиная с его корня.  4)Нейронные сети - это класс моделей, основанных на биологической аналогии с мозгом человека и предназначенных для решения разнообразных задач анализа данных после прохождения этапа, так  называемого обучения на имеющихся данных. Обычно решают задачу классификации.  5)Нечеткая логика - Основным способом исследования задач анализа данных является их отображение на формализованный язык и последующий анализ полученной модели. Неопределенность по объему отсутствующей информации у системного аналитика можно разделить на три большие группы: неизвестность, неполнота (недостаточность, неадекватность), недостоверность.  Основной сферой применения нечеткой логики и во многом остается управление.  6)Генетические алгоритмы – имитация биологических процессов. | |
| ***4.Основні відмінності баз даних (БД) від баз знань (БЗ).***  ***1)***Внутренняя интерпретация. Данные, хранящиеся в базе данных, имеют семантическую интерпретацию лишь после обработки их соответствующими программами. Знания, в отличие от данных, всегда несут в себе содержательную информацию.  ***2)***Наличие ситуативных связей. Знания связаны не только структурно. Они выражают закономерности относительно фактов, процессов, явлений и причинно-следственных отношений между ними. Ситуационные связи помогают строить процедуры анализа знаний на совместимость, противоречивость, одновременность и т.п.  ***3)***Активность знаний. Это принципиальное отличие знаний от данных. Знание всегда активные. Например, противоречивость в знаниях является стимулом к появлению новых знаний. Таким самым стимулом активности является неполнота знаний, которая и предопределяет необходимость их пополнения.  ***4)***Знания отличаются от данных формой представления. Данные выражают количественные характеристики и большей частью выдаются в цифровом виде. Знание - это преимущественно качественные характеристики, которые набирают в виде текстовой информации. | ***5.Етапи розв’язку задачі людиною.***  Процесс решения задачи состоит из таких основных подпроцессов, как:   * Обнаружение проблемной ситуации; * Постановка задачи: выявление и более или менее строгое определение исходного (данного) — его элементов и отношений между ними — и требуемого (цели); * Нахождение решения задачи. | ***6.Числення висловлювань. Символи та речення в численні висловлювань.***  Символы высказываний(пропозициональные символы) составляют высказывания или утверждения относительно некоторого мира.  Символы исчисления высказываний:   * Большие буквы латинского алфавита * Константы истинности (true,false) * Логические связки: → ˅ ˄ ¬ ≡   Предложения исчисления высказываний:   * Каждый логический символ и символ истинности являются предложением; * Отрицание предложения есть предложение; * Коньюнкция(логическое умножение), или операция И, двух предложений есть предложение. Члены коньюнктивных предложений называются коньюнктами; * Дизьюнкция(логическое сложение), или операция ИЛИ, двух предложений есть предложение. Члены коньюнктивных предложений называются дизьюнктами; * Импликация(включение) одного предложения в другое есть предложение. В импликации P→Q, P - предпосылка или антецедент, а Q – заключение, или логическое следствие. * Эквивалентность двух предложений есть предложение.   В предложениях исчисления высказываний знаки () и [] используются для группировки символов в подвыражениях и, таким образом, дают возможность управлять порядком их оценки и и присваивания значений.  Выражение является предложением или правильно построенной формулой исчисления высказываний тогда и только тогда, когда оно может быть сформулировано в виде некоторой последовательности допустимых символов согласно установленным правилам. | |
| ***7.Числення висловлювань. Визначення значення істинності речення числення висловлювань. Відмінності від числення предикатів.***  Интерпретация набора высказываний –это присвоение значения истинности Т или F, каждому пропозициональному символу.  Символу true всегда присваивается T, а символу false – F.  Интерпретации или значения истинности предложений определены следующим образом:   * Определение истинности отрицания: высказывание ¬ P, где Р – любой пропозициональный символ, есть F, если Р имеет значение Т; и высказывание ¬ P есть Т, если Р имеет значение F. * Определение истинности коньюнкции ˄: высказывание имеет значение Т, только если оба коньюнкта имеют значение Т; иначе выражение имеет значение F. * Определение истинности дизьюнкции ˅: высказывание имеет значение F, только если оба дизьюнкта имеют значение F; иначе выражение имеет значение Т. * Определение истинности импликации →: высказывание имеет значение F только тогда, когда предпосылка есть Т и значение истинности следствия есть F; иначе выражение имеет значение Т. * Определение истинности эквивалентности ≡: высказывание имеет значение Т только тогда, когда оба выражения имеют одинаковое значение истинности для всех возможных интерпретаций; иначе выражение имеет значение F.   Значения истинности сложных выражений часто описываются таблицами истинности. В них перечисляются все возможные варианты интерпретации данного выражения.  Отличия от исчисления предикатов:   * В исчислении высказываний каждый атомарный(элементарный) символ обозначает высказывание некоторой сложности. При этом не существует способа получить доступ к компонентам отдельного суждения. Исчисление предикатов предоставляет эту возможность. Посредством правил вывода можно изменять выражения исчислений предикатов, непосредственно обращаясь к их компонентам и выводя новые предложения. * Кроме того, исчисление предикатов позволяет включать в выражения переменные. | ***8.Числення предикатів. Відмінності від числення висловлювань. Символи числення предикатів. Логічні операції, які використовуються в численні предикатів.***  Отличия от исчисления высказываний:   * В исчислении высказываний каждый атомарный(элементарный) символ обозначает высказывание некоторой сложности. При этом не существует способа получить доступ к компонентам отдельного суждения. Исчисление предикатов предоставляет эту возможность. Посредством правил вывода можно изменять выражения исчислений предикатов, непосредственно обращаясь к их компонентам и выводя новые предложения. * Кроме того, исчисление предикатов позволяет включать в выражения переменные.   Алфавит исчисления предикатов:   * Набор букв английского алфавита верхнего и нижнего регистра * Набор цифр 0…9 * Символ подчеркивания   Символы в исчислении предикатов начинаются с буквы, за которой следует последовательность вышеперечисленных знаков.  К символам исчисления предикатов относят следующие:   * Символы истинности true или false. Это зарезервированные символы. * Символы констант – это символьные выражения, начинающиеся с символа нижнего регистра. * Символы переменных – это символьные выражения, начинающиеся с символа верхнего регистра. * Функциональные символы – это символьные выражения, начинающиеся с символа нижнего регистра. С функциями связывается арность, указывающая на число их аргументов.   Логические операции, которые используются в исчислении предикатов:  - ¬ – отрицание;  - ∧ – логическое И (конъюнкция, логическое умножение);  - ∨ – логическое ИЛИ (дизъюнкция, логическое сложение);  - → – импликация (если - то);  - ≡ – эквивалентность; | ***9. Числення предикатів. Змінні, константи, предикати та функції у численні предикатів.***  Символы исчисления предикатов могут представлять переменные, константы, функции или предикаты.  Константами называют определенные обьекты или свойства из области рассуждения. Символьные константы должны начинаться со строчной буквы (буквы нижнего регистра). Константы true и false зарезервированы как символы истинности.  Идентификаторы переменных используются для обозначения общих классов обьектов или свойств из области рассуждения. Переменные представляются идентификаторами, начинающимися с прописной буквы (буквы верхнего регистра).  Функции обозначают отображение одного или нескольких элементов множества (называемое областью определения функции) в однозначно определяемый элемент другого множества (множество значений функции). Элементы определения и множества значений – это обьекты из области рассмотрения.  Функциональное выражение – символьное выражение, начинающееся со строчной буквы, за которым следует список аргументов (термов), заключенных в скобки. Функциональное выражение возвращает результаты любого типа (строку, число, дату и т.д.).  Предикат – символьное выражение, начинающееся со строчной буквы, за которым следует список аргументов (термов), заключенных в скобки. Предикат возвращает результат логического типа (истину или ложь). Предикат указывает на отношение между несколькими обьектами в мире. | |
| ***10.Числення предикатів. Квантифікація змінних у численні предикатів. Необхідність використання кванторів. Числення предикатів 1-ого порядку.***  Квантификация переменных — это важная часть семантики исчисления предикатов. Если переменная используется в предложении, то она выполняет роль заполнителя и обозначает знакоместо. На это место в выражении может быть подставлена любая константа, допускаемая интерпретацией. Таким образом, переменная является шаблоном для подстановки.  В исчислении предикатов переменные должны быть связаны одним из двух кванторов: всеобщности(универсальности) или существования. Переменная считается свободной, если она не связана квантором универсальности или существования. Выражение считается замкнутым (closed), если все его переменные связаны кванторами. Основное выражение (ground expression) вообще не имеет никаких переменных. В исчислении предикатов все переменные должны быть связаны кванторами.  Квантор всеобщности означает, что предложение истинно для всех значений переменной. Для указания области действия квантора, т.е. выделения имен переменных, на которые распространяется действие квантора, часто используются круглые скобки.  Квантор существования указывает, что предложение истинно, по крайней мере для одного значения в области определения.  На определенном нами языке переменные, связанные квантором всеобщности и квантором существования, могут ссылаться только на объекты (или константы) из рассматриваемой области определения. Имена предикатов и имена функций не могут быть заменены именами переменных, стоящих под знаком квантора. Этот язык называется исчислением предикатов первого порядка (first-order predicate calculus).  Исчисление предикатов первого порядка позволяет связывать знаком квантора переменные, соответствующие объектам из предметной области, но не предикаты или функции. | ***11.Числення предикатів. Атомарні речення числення предикатів.***  Атомарное предложение – это предикатная константа арности n, за которой следуют n термов t1, t2,…,tn заключенных в круглые скобки и отделенных запятыми. Значения истинности true и false тоже являются атомарными предложениями.  Можно комбинировать атомарные предложения и формировать предложения в исчислении предикатов, используя логические операторы: ˅, ˄, ¬, →, ≡.  Каждое атомарное предложение есть предложение:   * Если s – предложение, то его отрицание ¬s тоже является предложением. * Если s1 и s2 – предложения, то их коньюнкция s1˄s2 тоже является предложением. * Если s1 и s2 – предложения, то их дизьюнкция s1˅s2 тоже является предложением. * Если s1 и s2 – предложения, то их импликация s1→s2 тоже является предложением. * Если s1 и s2 – предложения, то их эквивалентность s1≡s2 тоже является предложением. * Если Х – переменная и s – предложение, то есть предложение. * Если Х – переменная и s – предложение, то есть предложение. | ***12.Числення предикатів. Визначення інтерпретація I в області визначення D.***  Пусть область определения D – некоторое непустое множество.  Интерпретация на D – это связывание логических обьектов из D с каждой константой, переменной, предикатом и функциональным символом в выражении исчисления предикатов на основе следующих правил:   1. Каждой константе ставится в соответствие элемент из D. 2. Каждой переменной ставится в соответсвие непустое подмножество из D; оно является областью допустимых значений для этой переменной. 3. Каждая функция f арности m определяется для m параметров из D и задает отображение из Dm в D. 4. Каждый предикат p арности n определяется для n параметров из D и задает отображение из Dn в (T,F).   При таком определении интерпретации , чтобы получить значение выражения, следует присвоить выражению значение истинности на этой интерпретации. | |
| ***13.Числення предикатів. Визначення значення істинності виразу числення предикатів.***  Пусть существует выражение Е и интерпретация I для Е на непустой области определения D. Значение истинности для Е определяется так:   1. Значение константы – это элемент из D, которому соответствует данная константа в интерпретации I. 2. Значение переменной – это множество элементов из D, которые соответствуют данной переменной в интепретации I. 3. Значение функционального выражения - это такой элемент из D, который получается в результате оценивания функции для значений параметров, соответствующих интерпретации. 4. Значение символа истинности true – это Т, а false - это F. 5. Значение атомарного предложения равно либо Т, либо F, и определяется интерпретацией I. 6. Значение отрицания предложения равно T, если значение предложения равно F; и значение отрицания предложения равно F, если значение предложения равно Т. 7. Значение конъюнкции двух предложений равно T, если оба предложения принимают значение T, иначе оно равно F. 8. Значение дизьюнкции двух предложений имеет значение F, только если оба предложения имеют значение F; иначе оно равно Т. 9. Значение импликации двух предложений имеет значение F только тогда, когда предпосылка есть Т и значение истинности следствия есть F; иначе оно имеет значение Т. 10. Значение эквивалентности двух предложений имеет значение Т только тогда, когда оба предложения имеют одинаковое значение истинности для всех возможных интерпретаций; иначе оно имеет значение F.   Наконец, для переменной X и предложения S, содержащего X, выполняются следующие соотношения:   1. Значение выражения XS равно Т, если S равно Г для всех значений X из I, иначе оно равно F. 2. Значение XS равно T, если в интерпретации существует значение X, для которого S равно T; иначе оно равно F. | ***14.Числення предикатів. Поняття “логічно слідує”, “задовольняє”, “модель”, “адекватність”.***  Для выражения X исчисления предикатов и интерпретации I имеют место следующие определения:   * Если X имеет значение T на I при конкретных значениях переменных, то говорят, что I ***удовлетворяет*** X. * Если I удовлетворяет X при всех значениях переменных, то I является ***моделью*** X. * X выполнимо (satisfiable) тогда и только тогда, когда существуют такая интерпретация и значение переменной, которые ему удовлетворяют; в противном случае X невыполнимо (unsatisfiable). * Набор выражений выполним тогда и только тогда, когда существуют интерпретация и значения переменных, которые удовлетворяют каждому элементу. * Если набор выражений невыполним, то говорят, что он противоречив (inconsistent). * Если X имеет значение Т для всех возможных интерпретаций, то говорят, что X имеет силу, или ***адекватно*** (valid). * Выражение исчисления предикатов X ***логически следует*** из набора S выражений исчисления предикатов, если каждая интерпретация и значения переменных, которые удовлетворяет S, удовлетворяют и X. * Правило вывода обосновано (sound), если каждое выражение исчисления предикатов, полученное в соответствии с правилом из множества S выражений исчисления предикатов, также логически следует из S. * Правило вывода полно (complete), если на данном множестве S выражений исчисления предикатов правило позволяет вывести любое выражение, которое логически следует из S. | ***15.Числення предикатів. Правила виводу модус поненс, модус толленс, виключення І, введення І, універсальне інстанціювання.***  Если известно, что предложения Р и Р→Q истинны, то ***модус поненс*** позволяет вывести Q.  Согласно правилу вывода ***модус толленс*** (modus tollens), если известно, что P→Q является истинным и Q ложно, можно вывести ¬Р.  ***Исключение "И"*** - правило, позволяющее вывести истинность обоих конъюнктов на основе истинности конъюнктивного предложения. Например, если Р˄Q истинно, можно сделать вывод, что Р и Q истинны.  ***Введение "И"*** позволяет вывести истинность конъюнкции из истинности ее конъюнктов. Например, если Р и Q истинны, то конъюнкция P˄Q истинна.  ***Универсальное инстанцирование*** сводится к следующему: если любую переменную, стоящую под квантором всеобщности в истинном предложении, заменить любым соответствующим термом из области определения, то результирующее выражение - истинно. Таким образом, если *а* принадлежит той же области определения, что и X и Xр(X), то можно вывести р(*а*). | |
| ***16.Числення предикатів. Поняття уніфікації. Необхідність використання уніфікації.***  Чтобы применять правила вывода типа *модус поненс,* система вывода должна уметь определять, когда два выражения являются эквивалентными, или *равносильными.* В исчислении высказываний это тривиально: два выражения равносильны тогда и только тогда, когда они синтаксически идентичны. В исчислении предикатов определение равносильности двух предложений усложняется наличием переменных. Правила универсального инстанцирования позволяет заменять переменные под знаком квантора всеобщности термами из области определения. Необходимо определить процесс замены переменных, при котором несколько выражений могут стать идентичными (обычно дня того, чтобы можно было применять правила вывода).  Унификация — это алгоритм определения необходимых подстановок с целью *приведение* в соответствиедвух выражений исчислении предикатов. Унификация и такие правила вывода, как модус поненс, позволяют делать выводы на множестве логических утверждений. Для этого логическая база данных должна быть выражена в соответствующей форме. | ***17.Числення предикатів. Застосування підстановки для змінних, зв’язаних кванторами, в процесі уніфікації.***  Весьма важный аспект этой формы заключается в требовании, чтобы все переменные стояли под знаком квантора всеобщности. Эго обеспечивает полную свободу в выполнении подстановок. Переменные, стоящие под квантором существования, можно устранять из предложений в базе данных, заменив их константами, обеспечивающими истинность предложения. Например, ⱻ*Х parent(Х, tот*) может быть заменено выражением *parent(bob, tот)* или *parent(mary, tom),* принимая во внимание, что Боб (bob) и Мери *(таrу)* являются родителями Тома (tот) в этой интерпретации.  Процесс удаления переменных, связанных квантором существования, усложнен тем фактом, что значение этих подстановок может зависеть от значения других переменных в выражении. Например, о высказывании ⱯХ ⱻУ mother(Х,У) значение переменной У под квантором существования зависит от значения X. *Сколемизация* — это замена каждой переменной, связанной квантором существования, функцией нескольких или всех имеющихся в предложении переменных, которая возвращает соответствующую константу. В вышеупомянутом примере, поскольку значение У зависит от X, У можно заменить *сколемовской функцией f* от Х, Это порождает предикат ⱯХ *mother(X,f(Х)). Сколемизация*— это процесс, который также позволяет связывать переменные, стоящие под квантором всеобщности, с константами.  После удаления из логической базы данных переменных, связанных квантором существования, можно применить унификацию и привести предложения в форму, необходимую для применения таких правил вывода, как *модус поненс.*  Процесс унификации осложняется тем фактом, что переменная может быть заменена любым термом, включая другие переменные и функциональные выражения произвольной сложности. Эти выражения могут тоже содержать переменные. Например, father(jаск) можно использовать в качестве подстановки дня X в выражении *тап(Х)* для получения вывода, что отец Джека смертен. | ***18.Числення предикатів. Обмеження при застосуванні підстановок під час уніфікації.***  При создании алгоритма унификации, который вычисляет подстановки, необходимые для соответствия двух выражений, возникают некоторые проблемы.  Хотя константу можно систематически использовать в качестве подстановки для переменной, любая константа рассматривается как “базовый экземпляр” и не может быть заменена. Нельзя также два различных “базовых экземпляра” использовать в качестве подстановки для одной и той же переменной.  Переменная *не может* быть унифицирована с термом, содержащим ее. Поэтому переменная *X* не может быть заменена на р(Х), поскольку это порождает бесконечное выражение: р(р(р(р(...Х)...). Тест для этой ситуации называется *проверкой вхождения.*  Вообще, процесс решения задачи требует ряда выводов и, следовательно, ряда последовательных унификаций. Логические решающие устройства задач должны поддерживать согласованность подстановок для переменных. Важно, чтобы любая унифицирующая подстановка была сделана согласованно по всем вхождениям этой переменной во все выражения. А выражения должны быть приведены в соответствие друг другу.  Если переменная связана, вес последующие унификации и процедуры вывода должны учитывать это. Если переменная связана с константой, её уже нельзя связывать с другим термом при последующих унификациях. Если переменная X1 использовалась в качестве подстановки для другой переменной Х2, а затем была заменена константой, то в Х2 тоже необходимо отразить это связывание. Множество замен, используемых в последовательности выводов, играет важную роль, потому что *оно* может содержать ответ на первоначальный вопрос. | |
| ***19.Числення предикатів. Композиція підстановок. Приклад.***  Если S и S' являются двумя множествами подстановок, то композиция S и S' (пишется SS’) получается после применения S’ к элементам S и добавления результата к S. Рассмотрим пример композиции последовательности подстановок (X/Y,W/Z), (V/Х), (а/V, f(b)/W)*.*  Они эквивалентны единственной подстановке  (а/Y, *f(b)/Z).*  Последняя подстановка была выведена путем компоновки (Х/У, W/Z) с (V/Х) для получения (V/У, W/Z*)* и компоновки результата с (а/V, f(b)/W)дня получения *(а/V, f(b)/Z).*  Композиция подстановок — это метод, с помощью которого объединяются подстановки унификации. Его можно реализовать, используя рекурсивную функцию unify. Можно показать, что композиция является ассоциативной, но не коммутативной. | **20.Числення предикатів. Найбільш загальний уніфікатор. Необхідність пошуку найбільш загального уніфікатору.**  Последнее требование алгоритма унификации — унификатор должен быть максимально общим, т.е. для любых двух выражений должен быть найден наиболее общий унификатор. Это очень важно, поскольку при потере общности в процессе решения уменьшается вероятность достижения окончательного решения или такая возможность исчезает полностью.  Например, предложения р(Х) и р(У) можно унифицировать любым константным выражением вида (fred /Х, fred/У). Однако fread не является наиболее общим унификатором. Используя в качестве унификатора любую переменную, можно получить более общее выражение: (Z/Х, Z/У). Решения, полученные при использовании первой подстановки, всегда будут ограничены содержащейся в них константой fred,. лимитирующей логические выводы. Следовательно, fred можно использовать в качестве унификатора, но это снижает универсальность результата.  ОПРЕДЕЛЕНИЕ  НАИБОЛЕЕ ОБЩИЙ УНИФИКАТОР  Если s — произвольный унификатор выражения Е, а g— наиболее общий унификатор этого набора выражений, то в случае применения s к Е будет существовать еще один унификатор s' такой, что Еs=Еgs', где Еs и Еgs''— композиции унификаций, примененные к выражению Е.  Наиболее общий унификатор для набора выражений определяется с точностью до обозначения. В конечном счете, не имеет никакого значения, как называется переменная — X или У, поскольку это не снижает общности для результирующей унификации. | **21.Числення предикатів. Приклад уніфікації**  Представим функцию unify.  function unify(Е1, Е2);  begin  case  Е1 и Е2 — константы или пустые списки: %рекурсия завершается  if Е1=Е2 then return ()  е1зе return fail;  Е1 - переменная:  if Е1 входит в Е2 then return fail  еlsе return (Е2/Е1);  Е2 — переменная:  if Е2 входит в Е1 then return fail  еlsе return (Е1/Е2)  Е1 или Е2 - пуст then return fail %списки различных размеров  otherwise: %Е1 и Е2 списки  begin  НЕ1:=первый элемент Е1;  НЕ2:= первый элемент Е2;  SUBS1: =unify (НЕ1, НЕ2);  if SUBS1:=fail then return fail;  ТЕ1:=аррlу(SUBS1, хвост Е1);  ТЕ2:=аррlу(SUBS1, ХВОСТ Е2);  SUBS2: =unify (ТЕ1, ТЕ2) ;  if SUBS2= fail then return fail;  еlse return composition(SUBS1, SUBS2)  end  end  %конец case  end  Поведение вышеописанного алгоритма можно понять, проследив обработку запроса *unify((parents X(father X)(mother bill)), (parents bill (father bill)Y))*  Поскольку ни один параметр не является атомарным символом, при первом вызове функции unify она будет пытаться рекурсивно унифицировать первые элементы каждого выражения с помощью вызова *unify (parents, parents).* Эта унификация успешно выполнится и возвратит пустую подстановку (). Применение ее к остальным выражениям не вызовет никаких изменений; затем вызывается *unify(X,(father X)(mother bill)), (bill (father bill)Y)).* Во втором вызове unify ни одно выражение не является атомарным, поэтому алгоритм разделяет каждое выражение на его первый компонент и остальную часть. Затем следует вызов *unify(X,bill).* Этот вызов завершается успешно, поскольку оба выражения атомарные, и одно из них — переменная. В результате вызова возвращается подстановка (*bill/Х)*. Она применяется к “хвосту" каждого выражения, а функция unify — к результатам подстановки *unify((father bill)(mother bill)),((father bill)Y)).*  В результате этого вызова должна быть получена унификация (*father bill)* с (father bill). Это порождает вызовы  *unify(father,father)*  *unify(bill,bill)*  *unify((),())*  Все они успешно завершаются, возвращая пустое множество подстановок.Затем функция unify вызывается для оставшихся выражений *unify(((mother bill)),(Y))* Это, в свою очередь, приводит к вызовам  *unify((mother bill),Y)*  *unify((),())*  В первом из них (mother bill) унифицируется с У. В процессе унификации вместо переменной У подставляется целая структура *(mother bill).* Таким образом, унификация завершается успешно и возвращает *((mother bill)/Y).* Вызов *unify((),())* возвращает (). В результате комбинации с полученной ранее подстановкой (*bill/X)* приходим к ответу *(bill/X, (mother bill)/Y)).* | |
| ***22.Семантичні мережі. Приклад семантичної мережі.***  *Семантическая сеть* – представляет знания в виде графа, узлы которого соответствуют фактам или понятиям, а дуги — отношениям или ассоциациям между понятиями. Как узлы, так и связи обычно имеют метки.  Семантические сети также реализуют наследование.  Термин “семантическая сеть” обозначает семейство представлений, основанных на графах. Эти представления отличаются главным образом именами узлов, связен и выводами, которые можно делать в этих структурах. Однако общее множество предположений и отношений содержится во всех языках представления сетей. | ***23.Поняття “концептуальний граф”. Відмінності концептуальних графів від семантичних мереж. Приклад концептуального графу.***  *Концептуальный граф* — это конечный, связанный, двудольный граф. Узлы графа представляют *понятия,* или *концептуальные отношения.* В концептуальных графах метки дуг не используются. Отношения между понятиями представляются узлами концептуальных отношений.  В концептуальных графах узлы понятий представляют либо конкретные, либо абстрактные объекты в мире рассуждении. Конкретные понятия, такие как кошка, телефон или ресторан, характеризуются нашей способностью сформировать их образ. Отметим, что к конкретным относятся обобщенные понятия, например, кошка или ресторан, а также понятия конкретных кошек и ресторанов. Мы можем также сформировать образ обобщенной кошки. Абстрактные понятия включают такие абстракции как любовь, красота, верность, для которых не существует образов в нашем воображении. Узлы концептуальных отношений описывают отношения, включающие одно или несколько понятий. Одним из преимуществ концептуальных графов без использования помеченных дуг является простота представления отношений любой арности.  Каждый концептуальный граф представляет одно высказывание. Типовая база знании будет состоять из ряда таких графов. Графы могут быть произвольной сложности, но они должны быть конечными. Пример  Разработчики первых семантических сетей в основном не заботились об определении отношений класс-элемент и класс-подкласс, в результате чего возникала семантическая неоднозначность. В концептуальных графах каждое понятие является уникальным экземпляром конкретного типа. Каждый прямоугольник понятия снабжается меткой типа, определяющей класс или тип экземпляра, представленного этим узлом. | ***24.Концептуальні графи. Відмінності імені екземпляру концептуального графу від його маркеру. Позначення маркерів в концептуальних графах.***  Концептуальные графы позволяют описывать конкретные, но неименованные экземпляры. Для обозначения каждого экземпляра в мире рассуждений используется уникальный дескриптор, называемый *маркером*. Он представляет собой число, перед которым расположен символ #. Маркеры отличаются от имен тем. что они являются уникальными: экземпляр может иметь одно имя, несколько имен или вовсе быть безымянным, но он имеет ровно один маркер. Подобным образом различные экземпляры могут быть тезками, но не могут иметь один и тот же маркер. Это различие дает возможность преодолевать семантические неоднозначности, возникающие при именовании объектов.  Маркеры позволяют отделить экземпляры от их имен. Если собака #1352 имеет имя Эмма, для его описания можно использовать концептуальное отношение name (имя). Там, где не возникает опасность неоднозначности, мы можем упростить граф и ссылаться на экземпляр прямо по имени. | |
| ***25.Концептуальні графи. Ієрархія типів в концептуальному графі.***  Иерархия типов— это частичное упорядочение на множестве типов, которое можно обозначить символом ≤*.* Если s и t — есть типы и t≤ з, то говорят, что t — это *подтип* з, а з — *супертип* t. Поскольку эта систематизация частична, тип может иметь один или несколько супертипов, а также один или несколько подтипов. Если з, t. и u — типы, причем t ≤ з и t ≤ u, то говорят, что t — это *общий подтип* для з и u. Подобным же образом, если s ≤ v и u ≤ v, то v — *общий супертип* для з и u,  Иерархия типов концептуальных графов представляет собой решетку, описывающую общий вид системы множественного наследования. В этой решетке типы могут иметь множество родителей и детей. Однако каждая пара типов должна иметь *минимальный общий супертип* и *максимальный общий подтип.* Тип v является минимальным общим супертипом для типов з и u, если s ≤ v и u ≤ v и для любого общего супертипа w типов з и u v≤w. Максимальный общий подтип имеет аналогичное определение. Минимальный общий супертип совокупности типов служит для определения свойств, присущих только этим типам. Поскольку многие типы не имеют общих супертипов или подтипов, необходимо добавлять типы, которые выполняют эти роли. Чтобы иерархия типов действительно стала решеткой, в концептуальные графы включают два специальных типа. *Универсальный* обозначается символом ┬ является супертипом всех типов. *Абсурдный тип* обозначается символом ┴ и является подтипом всех типов. | ***26.Концептуальні графи. Операції по створенню нових концептуальних графів. Приклади використання операцій.***  Теория концептуальных *графов* включает ряд операций для создания новых графов на основе существующих. Они позволяют генерировать новый граф путем либо специализации, либо *обобщения* существующего графа, и очень важны для представления семантики естественного языка.  Предположим, что g1и g2— концептуальные графы.  Правило *копирования* позволяет сформировать новый граф *g,* который является точной копией g1.  *Ограничение* позволяет заменить узлы понятий графа узлами, представляющим их специализацию. Возможны две ситуации.  1. Если понятие *помечено* общим маркером, то общий маркер *может быть* заменен индивидуальным.  *2.* Метка типа может быть заменена одной из *меток его* подтипов, если это соответствует объекту ссылки понятия.  Правило *объединения* позволяет интегрировать два графа в один. Если узел понятия с1 графа s, идентичен узлу понятия с2 графа *s2,* то можно сформировать новый граф, вычеркивая *с2* и связывая все его отношения с с1. Объединение — это правило *специализации,* так как результирующий граф является менее общим, чем любой из его компонентов.  Если граф содержит два одинаковых отношения, то одно из них может быть вычеркнуто вместе со всеми его дугами. В этом заключается правило *упрощения.* Дублирующиеся отношения часто возникают в результате операции объединения.  Объединение и ограничение являются правилами специализации. Они определяют частичное упорядочение на множестве полученных графов. Если граф *g1* является специализацией g2, то мы можем говорить, что g1— обобщение g2. | ***27.Концептуальні графи. Пропозиційні вузли. Приклад.***  Графы можно использовать для определения отношении между объектами мира, а также между высказываниями. Рассмотрим, например, утверждение «Tom believes that Jane likes pizza» (“Том верит, что Джейн любит пиццу"). Здесь believes — это отношение, аргументом которого является высказывание. Концептуальные графы включают тип понятий рrорозithion (высказывание), объектом ссылки которого является множество концептуальных графов. Это позволяет определить отношения, включающие высказывания. Понятия высказываний, или пропозициональные понятия, изображаются прямоугольником, содержащим другой концептуальный граф. Эти понятия с соответствующими отношениями могут использоваться для представления знаний о высказываниях. На рис. показан концептуальный граф для приведенного выше утверждения. Отношение ехреriеnсеr (чувствующий) аналогично отношению аgеnt (агент) — оно тоже связывает субъект с глаголом. Связь ехреriеnсеr используется для состояний доверия, которые основываются скорее на интуиции, чем на действиях. | |
| ***28.Концептуальні графи. Алгоритм подання концептуального графу за допомогою числення предикатів.***  алгоритм преобразования концептуального графа *д* в выражение исчисления предикатов.  1. Каждое из n родовых понятий графа *д* связать с отдельной переменной х1, х2, … , xn  *2.* Каждое конкретное понятие в *д связать с* отдельной константой. Эта константа может быть просто именем или маркером, используемым для указания объекта ссылки для данного понятия.  3. Представить каждый узел понятия унарным предикатом, имя которого соответствует типу этого узла, и аргументом которого является переменная или константа данного узла.  4. Представить каждое n-арное концептуальное отношение на графе *д* n-арным предикатом с тем же именем. Каждый аргумент предиката является переменной или константой, соответствующей узлу понятия, связанного с этим отношением.  5. Сформировать тело выражения исчисления предикатов в виде конъюнкции всех атомарных выражений, построенных в пп.З и 4. Все переменные в выражении считаются связанными квантором существования. | ***29.Семантичні мережі. Приклад подання семантичної мережі мовою Пролог.***  В этой сети реализованы канонические формы представления данных. Предикат isa (Type, Parent) указывает на то, что объект Type является подтипом Parent, а предикат hasprop(Object, Property, Value) свойство объектов. Предикат hasprop указывает на то, что объект Object обладает свойством со значением Value. При этом Object и Value – это узлы сети, а Property – имя объединяющей их связи.  Фрагмент списка предикатов, описывающих показанную на рис. иерархию птиц имеет вид.  isa(canary, bird). isa(robin, bird).  isa(ostrich, bird). isa(penguin, bird).  isa(bird, animal). isa(fish, animal).  isa(opus, penguin). isa(tweety, canary).  hasprop (tweety, color, white). hasprop(robin, color, red).  hasprop(canary, color, yellow). hasprop(penguin, color, brown).  hasprop(bird, travel, fly). hasprop(fish, travel, swim).  hasprop(ostrich, travel, walk). hasprop(penguin, travel, walk).  hasprop(robin, sound, sing). hasprop(canary, sound, sing).  hasprop(bird, cover, feathers). hasprop(animal, cover, skin).  Предикат hasproperty начинает поиск с конкретного объекта. Если информация напрямую не связана с этим объектом, он переходит по связи isa к суперклассам. Если суперкласса больше не существует и предикат hasproperty не нашел нужного свойства, поиск завершиться неудачей. Рекурсивный алгоритм поиска, позволяющий определить, обладает ли некий объект семантической сети указанным свойством: hasproperty(Object, Property, Value):-  hasproperty(Object, Property, Value).  hasproperty(Object, Property, Value):-  isa(Object, Parent),  hasproperty(Parent, Property, Value). | ***30.Сценарії. Компоненти сценарію.***  *Сценарий* — это структурированное представление, описывающее стереотипную последовательность событий в частном контексте.  Сценарий включает следующие компоненты.  • *Начальные условия,* которые должны быть истинными при вызове сценария.  • *Результаты* или факты, которые являются истинными, когда сценарий завершается.  • *Предположения,* которые поддерживают контекст сценария. Множество предположений описывают принятые по умолчанию условия реализации сценария.  • *Роли* являются действиями, которые совершают отдельные участники.  • *Сцены.* Шенк разбивает сценарий на последовательность сцен, каждая из которых представляет временные аспекты сценария.  Элементы сценария — основные “части” семантического значения — представляются отношениями концептуальной зависимости. Собранные вместе во фреймоподобной структуре они представляют последовательность значений или событий. | |
| ***31.Сценарії. Приклад опису ситуації за допомогою сценарію.***  Сценарий: РЕСТОРАН  Аналог: Кафе  Реквизиты: Столы, Меню, F = Пицца, Чек, Деньги  Роли: S = Заказчик, W = Официант, C = Повар, M = Кассир, O = Владелец  Начальные условия: S голоден, у S есть деньги  Результаты: у S стало меньше денег, у O стало больше денег, S не голоден и (возможно) доволен  **Сцена 1 Вход**  S PTRANS S в ресторан; S ATTEND смотрит на столы; S MBUILD куда сесть; S PTRANS S за стол; S MOVE S в сидячее положение  **Сцена 2 Заказ**    **Сцена 3 Употребление пищи**    **Actions :** | ***32.Фрейми. Вміст полів фрейму.***  Семантические сети можно разбивать на части, добавляя к описаниям узлов дополнительную информацию и обеспечивая тем самым фреймовую структуру. **Фреймы** — это схема представления, во многом подобная сценариям и ориентированная на включение в строго организованные структуры данных неявных информационных связей, существующих в предметной области. Это представление поддерживает организацию знаний в более сложные единицы, которые отображают структуру объектов этой области. Возможно, наше собственное знание о мире тоже организовано во фреймоподобные структуры. Мы приспосабливаемся к каждой новой ситуации, вызывая информацию, основанную на опыте. Затем мы адаптируем эти знания в соответствии с новой ситуацией. Фреймы представляют сущности как структурированные обьекты с поименованными ячейками и связанными с ними значениями.  Фреймы позволяют организовать иерархию знаний. В сети каждое понятие представляется узлами и связями на одном и том же уровне дегализация. Однако очень часто для одних целей объект необходимо рассматривать как единую сущность, а для других — учитывать детали его внутренней структуры.  Ячейки фрейма содержат такую информацию:  1. Данные для идентификации фрейма.  2. Взаимосвязь этого фрейма с другими фреймами. «Гостиничный телефон» может служить специальным экземпляром «телефона», который, в свою очередь, может служить примером «механизма связи».  3. Дескрипторы требований для фрейма. Стул, например, имеет высоту сидения от 20 до 40 см, его задняя часть выше 60 см… Эти требования могут быть использованы для определения соответствия новых объектов стереотипу фрейма.  4. Процедурная информация об использовании описанной структуры. Важной особенностью фреймов является возможность присоединить к ячейке процедурный код.  5. Информация по умолчанию. Это значения ячейки, которые должны быть истинными, если не найдены противоположные. Например, стулья имеют 4 ножки, на телефонах есть кнопки, гостиничные кровати заправляются персоналом.  6. Информация для нового экземпляра. Многие ячейки фрейма могут оставаться незаполненными, пока не указано значение для отдельного экземпляра или пока они не понадобятся для некоторого аспекта решения задачи. Например, цвет кровати может оставаться неопределенным.  Процедурные вложения являются важным свойством фреймов, так как они позволяют связать фрагменты программного кода с соответствующими сущностями фреймового представления. С помощью процедурных вложении можно создавать демоны.  Демон — это процедура, которая является побочным эффектом некоторого другого действия в базе знаний. Например, при каждом изменении определенной ячейки в системе можно запускать процедуру контроля соответствия типов или тест непротиворечивости.  Когда создается экземпляр фрейма класса, система будет пытаться заполнить его ячейки либо заданными пользователем значениями, либо значениями по умолчанию, взятыми из фрейма класса, либо выполнит некоторую процедуру или демон для получения значения экземпляра. Как и в семантических сетях, ячейки и значения по умолчанию наследуются через иерархию класс/подкласс. Конечно, используемая по умолчанию информация может привести к немонотонности описания данных, поскольку корректность такой информации не всегда можно доказать. | | ***33. Фрейми. Приклад використання фрейму.***  Фрейм, представляющий гостиничный номер:    Фреймы для базы знаний о птицах:    В первой ячейке каждого фрейма содержится имя узла. Во второй ячейке определяется отношение наследования между данным узлом и его родителями. В третьей ячейке содержится список свойств, описывающих этот узел. В последней ячейке фрейма находится список исключений и принимаемых по умолчанию значений для данного узла. |
| ***34.Фрейми. Приклад наведення фрейму мовою Пролог.***      Предикат frame связывает имена ячеек, списки свойств и принимаемые по умолчанию значения. Это позволяет различать типы знаний и приписывать им разное поведение в иерархии наследования. Некоторые процедуры для извлечения свойств из этого представления: | ***35.Продукційні системи. Визначення та структура.***  **Продукционная система** — это модель вычислений, играющая особо важную роль в задачах искусственного интеллекта как для создания алгоритмов поиска, так и для моделирования решения задач человеком. Продукционная система обеспечивает управление процессом решения задачи по образцу и состоит из :  1. ***Набор продукционных правил***. Их часто просто называют продукциями. Продукция — это пара «условие-действие», которая определяет одну порцию знаний, необходимых для решения задачи. Условная часть правила — это образец (шаблон), который определяет, когда это правило может быть применено для решения какого-либо этапа задачи. Часть действия определяет соответствующий шаг в решении задачи.  2. ***Рабочая память*** содержит описание текущего состояния мира в процессе рассуждений. Это описание является образцом, который сопоставляется с условной частью продукции с целью выбора соответствующих действий при решении задачи. Если условие некоторого правила соответствует содержимому рабочей памяти, то может выполняться действие, связанное с этим условием. Действия продукционных правил предназначены для изменения содержания рабочей памяти.  3. ***Цикл «распознавание-действие***». Управляющая структура продукционной системы проста: рабочая память инициализируется начальным описанием задачи. Текущее состояние решения задачи представляется набором образцов в рабочей памяти. Эти образцы сопоставляются с условиями продукционных правил; что порождает подмножество правил вывода, называемое конфликтным множеством. Условия этих правил согласованы с образцами в рабочей памяти. Продукции, содержащиеся в конфликтном множестве, называют допустимыми. Выбирается и активизируется одна из продукций конфликтного множества (разрешение конфликта). Активизация правила означает выполнение его действия. При этом изменяется содержание рабочей памяти. После того как выбранное правило сработало, цикл управления повторяется для модифицированной рабочей памяти. Процесс заканчивается, если содержимое рабочей памяти не соответствует никаким условиям.    Продукционные системы обеспечивают модель представления человеческого опыта в форме правил и позволяют разрабатывать алгоритмы поиска по образцу — центральный элемент основанных на правилах ***экспертных систем***. | | ***36.Продукційні системи. Принцип їхньої дії.***  Принцип работы продукционной системы: рабочая память инициализируется начальным описанием задачи. Текущее состояние решения задачи представляется набором образцов в рабочей памяти. Эти образцы сопоставляются с условиями продукционных правил; что порождает подмножество правил вывода, называемое конфликтным множеством. Условия этих правил согласованы с образцами в рабочей памяти. Продукции, содержащиеся в конфликтном множестве, называют допустимыми. Выбирается и активизируется одна из продукций конфликтного множества (разрешение конфликта). Активизация правила означает выполнение его действия. При этом изменяется содержание рабочей памяти. После того как выбранное правило сработало, цикл управления повторяется для модифицированной рабочей памяти. Процесс заканчивается, если содержимое рабочей памяти не соответствует никаким условиям.  В процессе разрешения конфликтов выбирается для выполнения правило из конфликтного множества. Стратегии разрешения конфликтов могут быть достаточно простыми, например, выбор первого правила, условие которого соответствует состоянию мира. Можно для такого выбора использовать сложную эвристику. Следует подчеркнуть, что продукционная система допускает использование дополнительных эвристик для управления алгоритмом поиска.  Чистая продукционная модель не имеет никакого механизма выхода из тупиковых состояний в процессе поиска; она просто продолжает работать до тех пор, пока не будут исчерпаны все допустимые продукции. Многие практические реализации продукционных систем содержат механизмы возврата в предыдущее состояние рабочей памяти. |
| ***37.Продукційні системи. Комутативні продукційні системи. Їх властивості.***  При определенных условиях порядок, в котором множество применимых правил, использующихся в базе данных, не существенен. В этом случае эффективность работы системы можно существенно повысить, отказавшись от исследования путей решения эквивалентных во всем, кроме очередности, правил.  Продукционная система называется коммутативной, если она обладает следующими **свойствами**:  1. Каждое из множества правил, применимое к базе данных D, применимо к любой базе данных, полученной при использовании любого применимого правила к D.  ∀D ( r1.p (D) & r2.p (D) → r2.p (r1.f (D)) ) (***если правило применимо, то навсегда***)  2. Если целевое условие выполняется по отношению к базе данных D, то оно также выполняется для любой базы данных, полученной при использовании любого применимого правила к D.  ∀D ( r.p(D) & t(D) → t(r.f(D)) ) (***если цель достигнута, то насовсем***)  3. База данных, полученная в результате применения к D любой последовательности применимых правил, инвариантна относительно перестановок в этой последовательности.  ∀D r1.p(D) & r2.p(D) → r2.f(r1.f(D)) = r1.f(r2.f(D)) (***если правила применимы, то они коммутируют***)  Коммутативность системы продукций не означает, что всю последовательность правил, используемых для преобразования исходной базы данных в целевую, можно переупорядочить. Дело в том, что после того, как некоторое правило было применено к глобальной базе данных, применимыми могут стать дополнительные правила. Только те правила, которые изначально применимы к базе данных, можно объединить в произвольную последовательность, чтобы получить результат, независящий от порядка.  **Преимущества**:  Первое преимущество состоит в том, что всегда можно использовать безвозвратный режим поиска, поскольку применение правила никогда не ведет к необходимости его отмены.  Второе преимущество состоит в том, что нет необходимости вводить механизм применения различных последовательностей правил.  Третье преимущество состоит в том, что применение неудачного правила откладывает, но не делает невозможным завершение процесса. Кроме того, после решения можно удалить неудачные правила из решающей последовательности.  Есть простой способ преобразования любой системы продукций в коммутативную. Допустим, мы решили задачу представления, и пусть в составленной системе продукций имеется стратегия управления с поиском на графе, которая порождает дерево поиска для глобальной базы данных.  Рассмотрим другую систему продукций, глобальной базой данных которой является дерево поиска. правила новой системы представляют собой различные способы преобразования дерева поиска с помощью стратегии управления первой системы продукции. Ясно, что любые правила второй системы, применимые на любом этапе, применимы в дальнейшем. Коммутативные свойства второй системы продукций явным образом реализует недетерминированную возможность проб, которой мы наделили первую систему продукций. Использование этого преобразования приводит к более сложной базе данных и множеству правил продукций, но допускает безвозвратный режим управления. | ***38.Продукційні системи. Некомутативні продукційні системи. Стратегії керування послідовністю вибору правил.***  Если исходная ПС может быть разложена в сумму ПС, которые могут рассматриваться независимо, то такую систему называют разложимой (некоммутативной) ПС. Чтобы разложить базу данных, необходимо уметь разложить соответствующее терминальное условие, т.е. суметь выразить глобальное терминальное условие через терминальные условия каждой составляющей. Наиболее важным и часто встречающимся случаем является ситуация, когда глобальное терминальное условие выражается как конъюнкция составляющих.  Есть два основных вида переупорядочения баз данных в процессе обработки:  - их можно расположить в каком-то жестком порядке, непосредственно в момент их порождения;  - динамически переупорядочить в ходе обработки.  ***Структура правил*** в продукционной системе, включая различия между условием и действием, а также порядок проверки условий, определяет метод исследования пространства. Выражения исчисления предикатов всего лишь определяют истинные отношения в области формулировки задачи и не делают никаких утверждений относительно порядка и интерпретации их компонентов. Следовательно, некоторое частное правило может иметь вид . Согласно правилам исчисления предикатов альтернативная форма того же правила может быть такой . Хотя эти формулировки логически эквивалентны, они не ведут к одинаковым результатам, если интерпретируются как продукции (продукционные правила), потому что реализация продукционной системы обеспечивает определенный порядок проверки соответствия и активизации правил.  Продукционная система налагает на декларативный язык описания правил процедурную семантику. Поскольку продукционная система проверяет правила в определенном порядке, программист может управлять поиском через структуру и порядок следования правил в продукционном наборе. Подобные стратегии:  1. **Рефракция**. Рефракция означает, что после активизации правила оно не может быть запущено снова, пока не изменятся элементы рабочей памяти, соответствующие его условиям. Рефракция препятствует зацикливанию.  2. **Новизна**. Стратегия новизны отдает предпочтение правилам, условия которых соответствуют образцам, добавленным в рабочую память последними. Это позволяет сосредоточить поиск на одной линии рассуждения.  3. **Специфичность**. Согласно этой стратегии целесообразнее использовать более конкретное, а не более общее правило. Одно правило более специфично (конкретно) чем другое, если оно содержит больше условий, а значит, соответствует меньшему количеству образцов в рабочей памяти. | | ***39.Продукційні системи. Переваги та недоліки продукційних систем.***  **Преимущества**:  1. Разделение знания и управления. Продукционная система— изящная модель разделения знания и управления в компьютерной программе. Управление обеспечивается циклом «распознание-действие». При этом знания о методах решения задач сосредоточены непосредственно в правилах. Преимущество такого разделения заключается в простоте изменения базы знаний (не требуется изменять код программы управления). И наоборот.  2. Естественное соответствие поиску в пространстве состояний. Компоненты продукционной системы естественно отображаются в логическую структуру поиска в пространстве состояний. Последовательные состояния рабочей памяти составляют вершины графа пространства состояний. Правила вывода — набор возможных переходов между состояниями. Разрешение конфликтов обеспечивает выбор перехода (ветви) в пространстве состояний. Эти правила упрощают выполнение, отладку и документирование алгоритмов поиска.  3. Модульность продукционных правил. Важный аспект в моделировании продукционных систем - это отсутствие синтаксического взаимодействия между продукционными правилами. Правила могут только влиять на активизацию других правил, изменяя образец в рабочей памяти. Правила не могут «вызывать» другие правила непосредственно, как подпрограммы. При этом они не могут устанавливать значения переменных в других продукционных правилах. Область действия переменных этих правил ограничена отдельным правилом. Эта синтаксическая независимость способствует инкрементальной разработке экспертных систем.  4. Управление иа основе образцов. Задачи, решаемые с помощью программ ИИ, требуют особой гибкости при выполнении программы. Это вызвано еще и тем фактом, что правила в продукционной системе могут запускаться в любой последовательности. Описание задачи, представляющее текущее состояние мира, определяет конфликтное множество и, следовательно, конкретный путь поиска и решения.  5. Трассировка и трактовка. Модульность правил и итерационный характер их выполнения облегчают контроль за работой продукционной системы.  6. Независимость от выбора языка. Модель управления продукционной системой не зависит от представления правил и рабочей памяти, если это представление поддерживает сравнение с образцами.  **Недостатки**:  1. Продукционные системы могут порождать бесконечные циклы при поиске на графе пространства состояний. Эти циклы особенно трудно определить в продукционной системе, потому что правила могут активизироваться в любом порядке. Следовательно, зацикливание может появиться при работе системы, но его не легко обнаружить путем проверки синтаксиса набора правил.  2. Сложность оценки целостности образа знаний, заключенных в продукционную систему. |
| ***40.Алгоритми пошуку на графі. Пошук в глибину та в ширину. Алгоритми їх виконання. Переваги та недоліки використання евристик при пошуку.***  Поиск на графе относится к группе информированных стратегий, когда хранится вся доступная полученная ранее информация. В случае поиска на графе создается так называемый граф поиска. **Граф поиска** – это ориентированный граф G (V, E), в котором множество узлов V – это множество состояний базы данных, а множество дуг E – это множество  примененных правил.  По своей сути узлы – это состояния БД. Кроме этого, есть множество дуг. Дуги – это правила. В общем случае в графе поиска могут быть циклы, петли, мультидуги. Он может быть не связан и т.д.  Существуют два принципиальных способа поиска на графе – поиск в ширину и поиск в глубину. Если список O – это стек, то есть для раскрытия выбирается последний положенный узел, то в этом случае говорят о поиске в глубину. Если список O – это очередь, то есть для раскрытия выбирается первый положенный узел, то в этом случае говорят о поиске в ширину.  **Boolean DFS (v,u,check)**, Сv – мн-во непосещенных вершин, смежных с v, u – искомый элемент, check = false.  1) push(v,Q) – добавить элем. в начало очереди.  2) пометить v как посещенный.  3) Проверить Сv != { Ø } (не пустое)? Если «да», то взять w ∈ Сv, вызвать DFS (w,u,check) и перейти на п.3.  4) взять первый элемент из Q, положить в tmp: tmp ← pop(Q).  5) Проверить tmp == u? Если «да», положить check = true.  6) Вывести check. 7) Закончить.  **Boolean BFS (v,u)**, где u – искомый элемент, Сv – мн-во потомков вершины v. FIFO  1) Q ← {v}. 2) Пометить v, как посещенную.  3) Проверить Q != { Ø } Если «нет», то на п. 10.  4) v ← pop(Q) — взять первый элемент.  5) Проверить v == u? Если «да», закончить и вернуть true.  6) Взять w ∈ Сv. Если Сv пустое, то перейти на п.3.  7) Проверить w – не посещенная? Если «нет», то на п. 6.  8) push(w, Q). 9) Пометить w как посещенную, перейти на п.6.  10) Вернуть false. 11) Закончить.  При решении любой задачи всегда существует дилемма – проверять или не проверять дублирование узлов? Если дублирование проверять, то поиск в ширину может построить меньшее количество узлов, чем поиск в глубину. Если граф конечный, то поиск в ширину и поиск в глубины полны, то есть найдут путь, если он есть. Но в случае бесконечного графа, поиск в ширину найдет путь, если он есть, но искать его будет достаточно долго; поиск в глубину найдет быстрее, но при этом может пройти мимо нужного пути и уйти в бесконечность.  Поиск в ширину / глубину - это все неинформированные процедуры поиска. Без использования дополнительной информации о задаче оба метода безнадежно плохи. **Эвристики** уменьшают вычислительные затраты на применение правил за счет того, что сокращают общее количество применений правил. Но применение самих эвристик также требует каких-то вычислений и привносит свои затраты. Кроме того, манипуляции с графом поиска, особенно если он велик, также требуют затрат.  **Преимущества:** - эвристика помогает при отсутствии общего решения задачи;- дает достаточно хорошее решение в большинстве случаев (В действительности может быть даже известно (то есть доказано) то, что эвристический алгоритм формально неверен. Его всё равно можно применять, если при этом он даёт неверный результат только в отдельных, достаточно редких и хорошо выделяемых случаях или же даёт неточный, но всё же приемлемый результат)  **Недостатки:** - не гарантирует нахождение лучшего решения;- не гарантирует нахождение решения, даже если оно заведомо существует (возможен «пропуск цели»)**;** - может дать неверное решение в некоторых случаях. | ***41.Мова Пролог. Використання опису типів предикатів у розділі DOMAINS мови Пролог. Приклад.***  DOMAINS - секция описания доменов (типов). Секция применяется в том случае, если в программе используются нестандартные домены. Для большинства программ раздел DOMAINS, как правило, необходим для объявления списков, составных структур и пользовательских доменов. При объявлении нового предиката определяется, к каким доменам (типам) будут принадлежать аргументы этого. Доменом параметра является стандартный (встроенный) домен языка, либо домен, объявленный программистом в разделе domains. Даже если аргумент принадлежит стандартному домену, его имя будет более информативно, например предикат личность(имя, пол, возраст) более понятен, чем личность(symbol, symbol, integer).    Следует отметить, что два пользовательских домена, которые описаны с использованием одного и того же системного домена, явно друг с другом не совместимы.  **Составной объект данных (структура)** представляет другой объект или совокупность объектов и позволяет системе обрабатывать несколько фрагментов данных как один. Составной объект состоит из имени предиката (функтора) и одного или нескольких принадлежащих функтору аргументов, заключенных в круглые скобки: **функтор(объект1, объект2, ..., объектN).**    В данном случае дата\_cтp – имя домена структуры, дата – функтор, который является идентификатором (именем) составного объекта, а месяц, день, год – простые объекты структуры. Если объекты структуры принадлежат к одному домену, такая структура называется однодоменной, в противном случае структура является многодоменной.    Точка с запятой читается как **или**. В этом случае возможны 4 варианта идентификации.  При объявлении доменов многоуровневых составных объектов следует помнить, что одна строка описывает только один уровень, а не весь объект. Например, объект книга не может быть определен таким образом: **книга = книга(название, автор(имя, фамилия))**  **Список** – это объект, который содержит внутри себя упорядоченный набор других объектов, следующих друг за другом. | | ***42.Мова Пролог. Типи предикатів на Пролозі, різниця між ними.***  Предикаты в Прологе определяют отношения между объектами, то есть используются для представления как фактов, так и правил. При трансляции утверждений Пролог контролирует правильность написания имени предиката, количество объектов и типы их доменов. Термин арность в Прологе обозначает число объектов утверждения. Так утверждение likes(mary,apples) имеет арность 2.    Предикаты **нулевой арности** в основном используются для построения правил:  ехать\_домой :- погода (слякоть), транспорт (автобус).  Предикаты **арности 1** в Прологе (Prolog) полезны при сортировке объектов программы по доменам. В приведенном в таблице примере предикат female указывает, что имя betty относится к домену женских имен. Предикаты **арности 2** используются для установления отношения между двумя объектами. Например, предикат: father(person,person) и соответствующее ему утверждение: father(john,kathy). Предикаты **арностей выше 2** пригодны для установления связи нескольких объектов по какому-либо признаку. В утверждении:  europe(“France”,”Germany”,”Spain”,”Italy”)  все используемые значения: France, Germany, Spain и Italy принадлежат домену country. Общим для них является то, что все они обозначают европейские страны. |
| ***43.Мова Пролог. Введення виведення даних. Приклад.***  В системе VP, существует набор встроенных предикатов для осуществления ввода данных с клавиатуры: **readchar(Ch)** – чтение символа с текущего устройства ввода. Переменная хранит код введенного символа. Предикат реагирует на нажатие *Enter*, *Tab*, *Esc*. Предикат терпит неудачу, если в процессе ввода достигнут признак конца файла (симв. с кодом 26). Ch должна быть не связана. **readint(Val)** – чтение целочисленного значения с текущего устройства ввода, причем введенное значение должно принадлежать отрезку –2147483648..2147483647, иначе False. Если при вводе цч значения будут введен символ, не являющийся цифрой или знаками + или –; или будет нажата клавиша Esc; или будет достигнут признак конца файла, то False. **readln(St) –** чтение строкового значения с текущего устройства ввода. Ввод символов строки продолжается до тех пор, пока не достигнут конец файла или не нажата клавиша Enter.  **readreal(Val)** **–** чтение вещественного значения с текущего устройства ввода. Если при вводе вещественного значения будут введен недопустимый символ или будет нажата клавиша Esc, или будет достигнут признак конца файла, то предикат вернет False.  В системе VP существует предикат форматного вывода **writef** в текущее устройство вывода (экран по умолчанию). Он используется следующим образом: writef(Форматная\_строка, Арг1, Арг2,…,Аргn), причем в форматной строке формат вывода задается с помощью последовательности такого вида ***%-m.pf***, где:  символ ***-*** показывает, что значение будет выровнено по левому краю поля вывода, ***m*** – десятичное число, которое описывает минимальный размер поля вывода, ***p*** – указывает либо количество знаков после запятой у вещественных чисел, либо максимальное количество выводимых символов строковых значений, ***f*** – формат выводимого значения. Формат, в котором будет выводиться соответствующий аргумент в вызове предиката **writef**, может быть представлен следующими символами: ***f*** – вещественное значение с фиксированной точкой, **e** – вещественное значение с плавающей точкой, **g** – представляет вещественное число в наиболее коротком для него формате (f или e), т.е. в формате, который требует меньше символов для вывода, **D** – целочисленное знаковое значение, **U** – Целочисленное беззнаковое значение, **X** – Шестнадцатеричное значение, **O** – Восьмеричное значение, **c** – Символ, **s** – Строка.  **Нельзя:** целочисленные значения представлять как вещественные и наоборот; строковые значения представлять как числовые и наоборот; представлять значения в формате, который не может хранить представляемое значение из-за меньшего диапазона возможных значений.   |  |  | | --- | --- | | **PREDICATES**  **example**  **CLAUSES**  **example:- D = 65, E1 = 5.567, E2 = 0.00003456,**  **St = "This is an example",**  **writef("D\_INTEGER = %\nD\_INTEGER = %10\nD\_CHAR = %c\n", D, D,D),**  **writef("E1\_REAL = %\nE1\_REAL = %10.2\nE1\_REAL\_F = %f\nE1\_REAL\_E =**  *продолж ->* | **%e\nE1\_REAL\_G = %g\n",**  **E1, E1, E1, E1, E1),**  **writef("E2\_REAL = %\nE2\_REAL\_F = %f\nE2\_REAL\_E = %e\nE2\_REAL\_G =**  **%g\n",**  **E2, E2, E2, E2),**  **writef("St = %\nSt\_10\_5 = %10.5\n",**  **St, St),**  **nl.**  **GOAL**  **example.** | | ***44.Мова Пролог. Робота з файлами. Приклад.***  При осуществлении операций ввода/вывода при работе с файлами общая схема выполнения действий выглядит следующим образом:  1. Открытие файла для чтения(openread)/записи(openwrite)/ добавления в конец(openappend) / чтения и записи(openmodify).  2. Изменение текущего устройства ввода(readdevice)/ вывода(writedevice) на указанный файл. 3. Используя предикаты для ввода с текущего устройства вв/выв на текущее устройство вывода осуществление операций вв/выв. 4. Закрытие файла (closefile). 5. Изменение при необходимости текущего устройства вв/выв на стандартные устройства (клавиатура/монитор).  Для открытия файла в зависимости от режима работы используются следующие предикаты:  **openread(симв\_имя\_файла, имя\_файла\_ОС)** – открывает файл ОС, заданный именем имя\_файла\_ОС для чтения и связывает этот файл с именем файла в Прологе, заданным с помощью 1-ого параметра. Если файл уже открыт, то происходит ошибка открытия файла. Если файл не найден, то происходит ошибка выполнения программы.  **openwrite(симв\_имя\_файла, имя\_файла\_ОС)** – создает новый файл ОС и открывает этот файл для записи и связывает этот файл с именем файла в Прологе, заданным с помощью 1-ого параметра. Если файл уже открыт, то происходит ошибка открытия файла (ошибка выполнения программы). Если файл существует, то он уничтожается и создается заново.   |  |  | | --- | --- | | **DOMAINS**  **FILE = inp\_file**  **GOAL**  **F\_Name = “c:\\test.txt”,**  **openread(inp\_file, F\_Name),**  **readdevice(inp\_file),**  % работа с файлом  **closefile(inp\_file).** | **DOMAINS**  **FILE = outp\_file**  **GOAL**  **F\_Name = “c:\\test.txt”,**  **openwrite(outp\_file, F\_Name),**  **writedevice(outp\_file),**  % работа с файлом  **closefile(outp\_file).** |   **openappend(симв\_имя\_файла, имя\_файла\_ОС)** – открывает файл (или создает новый файл, если указанный файл не существует) для записи в конец и связывает этот файл с именем файла в Прологе, заданным с помощью 1-ого параметра. Если файл уже открыт, то происходит ошибка открытия файла (ошибка выполнения программы). Использование этого предиката аналогично использованию **openread/openwrite**.  **openmodify(симв\_имя\_файла, имя\_файла\_ОС)** – открывает файл для его модификации (записи данных). Использование этого предиката аналогично использованию **openread/openwrite**. | | ***45.Експертні системи. Список задач, які можуть бути розв’язані за допомогою експертних систем.***  *Экспертные системы* используют знания специфичной предметной области. Обычно разработчики экспертной системы приобретают эти знания с помошью экспертов, методологию и деятельность которых затем эмулирует система. Экспертные системы строятся для решения широкого круга проблем в таких облас­тях. как медицина, математика, машиностроение, химия, геология, вычислительная тех­ника. бизнес, законодательство, оборона и образование. Эти программы решают разно­образные проблемы. ***Общие задачи экспертных систем:***  *Интерпретация*— формирование высокоуровневых выводов из наборов строк данных. *Прогнозирование* — проектирование возможных последствий данной ситуации. *Диагностика* — определение причин неисправностей в сложных ситуациях на основе наблюдаемых симптомов. *Проектирование* — нахождение конфигурации компонентов системы, которая удовлетворяет целевым условиям и множеству проектных ограничений. *Планирование* — разработка последовательности действий для достижения множества целей при данных начальных условиях н временных ограничениях*.*  *Мониторинг* — сравнение наблюдаемого поведения системы с ее ожидаемым поведением. *Инструктирование* — помощь в образовательном процессе по изучению технической области. *Управление* — управление поведением сложной среды. |
| ***46.Експертні системи. Архітектура типової експертної системи. Призначення компонентів.***  Пользователь взаимодействует с системой через пользовательский *интерфейс.*При разработке интерфейса экспертной системы используют разнообразие стилей, включая “вопрос-ответ”, меню управления или графич. интерфейс. Ядром экспертной системы является ***база знаний****,* которая содержит знание из частной прикладной области. База знаний содержит как общие знания, так и информацию о частных случаях. Механизм вывода применяет знания при решении реальных задач. По существу, он является интерпретатором базы знаний. В продукционной системе механизм вывода совершает цикл распознавание-действие. Процедуры, которые выполняют этот управляющий цикл, отделены от самих продукционных правил. Такое разделение механизма вывода и базы знаний дает возможность представить знания более естественным образом. Например, правила если..., то... точнее описывают навыки человека по решению задач, чем компьютерный код более низкого уровня. В связи с тем, что база знаний отделяется от программных управляющих структур более низкого уровня, разработчики экспертной системы могут сосредоточиться на накоплении и организации знаний, а не на деталях их компьютерной реализации. В идеале разделение знаний и управления позволяет вносить изменения в одну часть базы знаний без создания побочных эффектов в других. Разделение знаний и управляющих элементов программы позволяет использовать в различных системах одни и те же модули управления и интерфейса пользователя. Оболочка экспертной системы включает все компоненты, показанные на рис, а база знаний и данные о частных случаях могут пополняться для новых приложений. Штриховые линии на рис. ограничивают модули оболочки. Экспертная система должна сохранять информацию о частных случаях, в т.ч. факты и выводы. Сюда включаются данные, полученные в отдельном случае решения задачи, частные заключения, степени доверия к заключениям н тупики в процессе поиска. Подсистема объяснений позволяет программе пояснить свое рассуждение пользователю. Она обеспечивает обоснования заключений системы в ответ на вопросы "как?", объяснение необходимости конкретных данных, а также ответы на вопросы "почему". Многие системы включают также редактор базы знаний. Редакторы баз знаний помогают программистам локализовать и откорректировать сбои в действиях программы, используя информацию, обеспечиваемую подсистемой объяснений. Кроме того, они могут помочь в пополнении новых знаний, поддержке корректности синтаксиса правил и в проверке на непротиворечивость при изменениях в базе знаний. | ***47.Експертні системи. Критерії виправданості застосування експертної системи.***  Экспертные системы привлекают значительные денежные инвестиции и человеческие усилия. Попытки решить слишком сложную, малопонятную или, другими словами, не соответствующую имеющейся технологии проблему могут привести к дорогостоящим и постыдным неудачам. Поэтому были выработаны критерии оправданности решения данной задачи с помощью экспертной системы: **1.** Необходимость решения оправдывает стоимость и усилия по разработке экспертной системы. Многие экспертные системы были построены в таких областях, как разведка минералов, бизнес, оборона н медицина, где существует большой потенциал для экономии денег, времени и защиты человеческой жизни. **2.** Отсутствие человеческого опыта в ситуациях, где он необходим. В геологии, например, существует необходимость удаленной экспертизы минирования и бурения. Геологи и инженеры часто сами проходят большие расстояния, что связано с большими затратами и потерями времени. При использовании экспертной системы многие проблемы удаленных районов могут быть решены без их посещения. **3.** Проблема может быть решена с использованием символьных рассуждений. Решение проблемы не требует физической ловкости или конкретных навыков. Современным роботам и системам зрения не хватает ума и гибкости, присущих человеку. **4.** Проблемная область является хорошо структурированной и не требует рассуждений на основе здравого смысла. Высокотехнологичные области очень удобны для изучения и формализации: для них четко определена терминология и построены ясные и конкретные концептуальные модели. Рассуждения на основе здравого смысла трудно автоматизировать. **5.** Проблема не может быть решена традиционными вычислительными методами. Технология экспертных систем не должна использоваться там, где это не является необходимостью. Если проблема может быть удовлетворительно решена более традиционными методами, она не должна рассматриваться как кандидат для решения экспертными системами. **6.** Известны эксперты, способные взаимодействовать между собой и четко выражать свои мысли. Знания, используемые экспертными системами, формируются из опыта и суждений людей, работающих в данной области. Важно, чтобы эти эксперты не только хотели, но и были способны поделиться знаниями. **7.** Проблема имеет приемлемые размеры и границы. Например, программа, пытающаяся воплотить весь опыт врача, нереальна. Программа, подсказывающая медицинские решения по использованию конкретного медицинского оборудования или конкретного множества диагнозов, более практична. | | ***48.Експертні системи. Цикл розробки експертної системи.***  Так же, как и большинство разработок ИИ, экспертные системы требуют нетрадиционного жизненного цикла разработки, основанного на раннем прототипировании и постепенной модификации кода. Обычно работа с системой начинается с попытки инженера по знаниям описать процессы, происходящие в данной предметной области. Далее инженер по знаниям н эксперт начинают процесс извлечения знаний по решению проблем. Когда инженер по знаниям получит общее представление о предметной области и проведет несколько сеансов решения задач с экспертом, он сможет приступить к разработке системы: выбрать способ представления знаний (например, правила или фреймы), определить стратегию поиска (прямой, обратный, в глубину, в ширину н т.п.), разработать пользовательский интерфейс. После выполнения этих обязательных этапов инженер по знаниям строит *прототип*. Этот прототип должен быть способен решить проблемы из данной предметной области и обеспечить испытательный стенд для проверки предварительных проектных решений. Когда прототип готов, инженер по знаниям и эксперт в предметной области испытывают и уточняют знания путем решения конкретных задач и устранения дефектов. Если предположения, сделанные при проектировании прототипа, оказываются корректными. прототип можно поступательно расширять до тех пор, пока он не превратится в окончательную систему. |
| ***49.Експертні системи. Методи отримання знань.***  В выполнении всех задач, возникающих в процессе приобретения знаний, могут принимать участие эксперт, инженер знаний и экспертная система. Человек-эксперт, работая в прикладной области, оперирует знанием, мастерством и практическими навыками. Это знание часто является неопределенным н неточным. Инженер по знаниям должен транслировать этот неформальный опыт в формальный язык вычислительной системы. Инженерия знаний является сложной задачей, связанной с жизненным циклом любой экспертной системы. Чтобы упростить эту задачу, полезно создать концептуальную модель — прослойку между человеческим опытом и реализованной программой. Под концептуальной моделью понимается концепция знаний о данной области, построенная инженером по знаниям. Инженеры по знаниям должны задокументировать и передать свое представление о предметной области с помощью общих методов проектирования программ. При разработке экспертной системы необходимо описать требования к системе. Однако из-за специфики жизненного цикла разработки требования к экспертной системе должны развиваться вместе с прототипом. Словари терминов, графические представления пространства состояний и комментарии в коде сами являются частью этой модели. Инженеры по знаниям должны сохранять записи своих интервью с экспертами по предметной области. Часто по мере улучшения понимания проблемной области инженеры по знаниям формируют новые интерпретации или открывают новую информацию. Записи наряду с документированием данных и интерпретаций играют важную роль при рассмотрении проектных решений и тестировании прототипов. Наконец, эта модель играет промежуточную роль при формализации знаний. Выбор языка представления оказывает значительное влияние на модель предметной области, создаваемую инженером по знаниям. Концептуальная модель –– это промежуточное проектное решение, шаблон для начала процесса кодирования человеческого опыта. Если инженер по знаниям использует модель исчисления предикатов, она может быть представлена в виде набора простых сетей, определяющих состояния рассуждений посредством типичных способов решения проблем. Лишь после дальнейших уточнении эта сеть превратится в набор правил “если..., то...". | ***50.Експертні системи на основі правил. Переваги та недоліки.***  В экспертных системах, основанных на правилах, знания о решении задач представляют в виде правил "если..., то...". Этот подход применяется в системах, архитектура которых показана на рис.    Как один из наиболее естественных он широко используется в коммерческих и экспериментальных экспертных системах.  **Преимущества этого подхода**: **1.** Возможность использовать знания экспертов в очень простой форме. Это важно в областях, зависящих от эвристик управления сложной и/или отсутствующей информацией. **2.** Правила отображаются в пространство состояний поиска. Возможности объяснения облегчают отладку программ. **3.** Отделение знаний от управления упрощает разработку экспертной системы, обеспечивая итеративный процесс разработки, в котором инженер накапливает, реализует и тестирует отдельные правила. **4.** В ограниченных предметных областях возможны хорошие результаты. Поскольку для интеллектуального решения задачи требуется большой объем знаний, экспертные системы ограничиваются узкими областями. Однако существует много областей, в которых разработка подобных систем оказалась чрезвычайно полезной. **5.** Хорошие возможности объяснения. Хотя системы на основе правил поддерживают гибкие, проблемно-зависимые объяснения, необходимо отметить, что максимальное качество этих объяснений зависит от структуры и содержания правил. Возможности объяснения в системах, основанных на данных и на целях, сильно различаются.  **Недостатки: 1.** Правила, полученные от экспертов, часто являются эвристическими по природе и не охватывают функциональных знаний или знании, основанных на моделях. **2.** Эвристические правила обычно не позволяют управлять отсутствующей информацией или неожиданными значениями данных. **3.** Их неприменимость на границе области определения. В отличие от людей, системы на основе правил при столкновении с новыми проблемами обычно не способны возвратиться назад к первоначальным принципам рассуждения.  **4.** Объяснения применимы лишь на описательном уровне. Теоретические обоснования отсутствуют. Поскольку эвристические правила не обеспечивают более глубоких рассуждений. **5.** Знания являются проблемно-зависимыми. Формализованные знания о некоторой предметной области, как правило, очень специфичны и применимы только к данной области. Современные языки представления знаний не обеспечивают гибкости, присушен человеку. | | ***51.Експертні системи на основі моделей. Переваги та недоліки.***  Качественные рассуждения на основе моделей включают следующие аспекты: 1. Описание каждого компонента прибора, позволяющее моделировать их поведение. 2. Описание внутренней структуры прибора. Оно обычно содержит представление компонентов и их взаимосвязей наряду с возможностью моделировать нх взаимодействие, Требуемая степень знаний внутренней структуры зависит от применяемого уровня абстракции и ожидаемого диагноза.  3. Диагностика частной проблемы. Это требует наблюдений реального поведения прибора, обычно — измерения его входов и выходов. Входные и выходные измерения получить легче всего.  **Преимущества: 1.** Возможность использовать при решении задач функциональные и структурные знания о предметной области. Это увеличивает эффективность механизма рассуждений при решении различных задач, включая те, которые не были предусмотрены при разработке системы. **2.** Механизмы рассуждений на основе моделей обычно очень эффективны. Они являются мощными и гибкими средствами решения задач, поскольку, как и люди, часто возвращаются к исходным данным при столкновении с новой проблемой. **3.** Некоторые знания можно использовать в разных задачах. Системы рассуждений на основе моделей зачастую базируются на теоретических научных знаниях. Поскольку наука обычно оперирует общими теориями, такое обобщение часто расширяет возможности механизма рассуждений на основе моделей. **4.** Обычно системы рассуждений, основанные на моделях, обеспечивают причинные объяснения. Таким образом пользователям можно передать более глубокое понимание причин неисправности, которое может сыграть важную образовательную роль.  **Недостатки: 1.** Отсутствие экспериментального (описательного) знания предметной области. Эвристические методы, используемые при рассуждениях на основе правил, отражают важный класс экспертных оценок. **2.** Необходимость точной модели предметной области. Знания из многих областей имеют строгую научную основу, которую можно использовать в рассуждениях на основе моделей. Однако во многих сферах, например, в некоторых медицинских направлениях, большинстве проблем проектирования или финансовых приложениях, хорошо определенная научная теория отсутствует. В таких случаях подходы, основанные на моделях, не могут быть использованы. **3.** Высокая сложность. Рассуждения, основанные иа моделях, обычно ведутся на детализированном уровне, что приводит к значительным усложнениям. Именно по этой причине эксперты в первую очередь разрабатывают эвристики. **4.** Исключительные ситуации. Необычные обстоятельства, например, замыкание или взаимодействие множества неисправностей электронных компонентов, могут изменить функциональность системы таким образом, что ее трудно будет предсказать. |
| ***52.Експертні системи на основі досвіду. Переваги та недоліки.***  В рассуждениях на основе опыта для решения новых проблем используется детальная база данных, содержащая известные решения задач. Рассуждения на основе опыта обеспечивают ряд преимуществ для разработки экспертных систем. Процесс извлечения знаний может быть упрощен, если механизм рассуждения будет использовать записанные решения экспертов по ряду проблем. Это могло бы освободить инженера по знаниям от необходимости разрабатывать общие правила на основе примеров, полученных от экспертов. Механизм рассуждения сможет обобщать правила автоматически, применяя их к новым ситуациям.  Подходы к таким рассуждениям позволяют экспертной системе учиться на своем опыте. После решения задачи путем поиска система может сохранить это решение, чтобы воспользоваться им при возникновении подобной ситуации (без привлечения поиска). Важно запоминать информацию как об успешных, так н о неудачных попытках решения. Таким образом, рассуждения на основе опыта являются мощным средством для обучения. Одним из первых примеров таких систем является программа для игры в шашки, где важные позиции на доске, найденные посредством поиска или эксперимента, запоминались для последующего использования.  **Преимущества:** **1.** Возможность напрямую использовать исторический опыт. Во многих областях необходимую информацию можно получить из существующих исторических материалов, записей или других источников. При этом устраняется необходимость интенсивного приобретения знаний с помощью человека-эксперта. **2.** Возможность сокращения рассуждений. Если известен аналогичный случай, новые задачи решаются гораздо быстрее, чем при использовании правил или моделей. **3.** Рассуждения на основе опыта позволяют избежать прошлых ошибок и использовать прошлые успехи. Они обеспечивают модель обучения, как интересную с теоретическом точки зрения, так и достаточно практичную для применения в сложных ситуациях. **4.** Не требуется всестороннего анализа знаний о данной области. В отличие от систем, основанных на правилах, где инженер по знаниям должен описать взаимодействие правил, рассуждения на основе опыта обеспечивают простую аддитивную модель приобретения знаний. Для использования этого подхода необходимо обеспечить соответствующее представление случаев, индексирование данных и стратегию адаптации. **5.** Соответствующие стратегии индексирования повышают эффективность решения задач. Мощность механизма рассуждений на основе опыта определяется возможностью выявлять отличительные особенности целевой проблемы и выбирать соответствующий случай из базы данных. Алгоритмы индексации часто обеспечивают эту функциональность автоматически,  **Недостатки:** **1.** Описания случаев обычно не учитывают более глубокие знания о предметной области. Это снижает качество объяснения и во многих ситуациях приводит к ошибочному применению опыта, а значит, к неправильным или низкокачественным советам. **2.** Большая база данных может привести к снижению производительности системы. **3.** Трудно определить хороший критерий для индексирования и сравнения случаев. Словари поиска и алгоритмы определения подобия необходимо тщательно отлаживать вручную. Это может нейтрализовать многие из преимуществ, присущих рассуждениям на основе опыта. | ***53.Експертні системи реального часу. Їх властивості.***  Экспертные системы реального времени решают следующие классы задач: мониторинг в реальном масштабе времени, обнаружения неисправностей, диагностика, оперативное планирование, системы – советчики оператора. Статические экспертные системы не способны решать подобные задачи, так как они не выполняют требования, предъявляемые к системам, работающим в реальном времени:  **1.** Представлять изменяющиеся во времени данные, поступающие от внешних источников, обеспечивать хранение и анализ изменяющихся данных. **2.** Выполнять временные рассуждения о нескольких различных асинхронных процессах одновременно (т.е. планировать в соответствии с приоритетами обработку поступивших в систему процессов). **3.** Обеспечивать механизм рассуждения при ограниченных ресурсах (время, память). Реализация этого механизма предъявляет требования к высокой скорости работы системы, способности одновременно решать несколько задач (т.е. операционные системы UNIX, VMS, Windows NT). **4.** Осуществлять постоянный мониторинг процесса, и при необходимости автоматически запускать механизм логического вывода решений по устранению критических ситуаций с одновременным информированием ЛПР. **5.** Моделировать "окружающий мир", рассматриваемый в данном приложении, обеспечивать создание различных его состояний. **6.** Протоколировать свои действия и действия персонала, обеспечивать восстановление после сбоя. **7.** Обеспечивать наполнение базы знаний для приложений реальной степени сложности с минимальными затратами времени и труда (необходимо использование объектно-ориентированной технологии, общих правил, модульности и т.п.). **8.** Обеспечивать настройку системы на решаемые задачи (проблемная/предметная ориентированность). **9.** Обеспечивать создание и поддержку пользовательских интерфейсов для различных категорий пользователей. **10.** Обеспечивать уровень защиты информации (по категориям пользователей) и предотвращать несанкционированный доступ. | | ***54.Експертні системи реального часу. Їх архітектура.***  Специфические требования, предъявляемые к экспертной системе реального времени (динамической ЭС), приводят к тому, что их архитектура отличается от архитектуры статических систем. Появляются две новые подсистемы: подсистема моделирования внешнего мира и подсистема связи с внешним окружением. Последняя осуществляет связи с внешним миром через систему датчиков и контроллеров. Кроме того, традиционные компоненты статической ЭС (база знаний и машина вывода) претерпевают существенные изменения, чтобы отразить временную логику происходящих в реальном мире событий.  ***компоненты, подвергающиеся изменениям, заштрихованы*** |