Практикум по языку Пролог, Второе задание

Дедлайн – **20 декабря**, первый показ – до **10 декабря**

Сдача второго задания – Большаковой Е.И., с предварительной высылкой по почте: **1)** программы **2)** файла с пояснением задачи (конкретно – см. вариант).

При программировании допускается использование любых встроенных предикатов языка, в том числе внелогических.

Во всех вариантах задания разрабатываемая программа должна быть организована как законченная система с понятным пользовательским интерфейсом, включая выдачу пользователю необходимых подсказок и пояснений. В вариантах **1, 2, 3, 5, 8, 9, 10** необходимо реализовать ввод исходных данных из файла, а также их редактирование и сохранение в файле после обработки, для неоднократного выполнения функций системы.

1. Система, отвечающая на вопросы о родственных отношениях

Основным модулем системы должна быть ***база знаний***, в которой хранится информация о членах семьи (не менее 20 человек) из нескольких (не менее четырех) поколений и родственных отношениях между ними (не менее 10-15 видов отношений).

Базовые отношения родства, к которым относятся 4 понятия: *родитель*, *супруги*, *мужчина*, *женщина*, должны быть представлены в базе явно, в виде соответствующих фактов Пролога. Все остальные отношения (внук, дядя, невестка и т.п.) определяются на основе базовых и для конкретных лиц должны выводиться системой исходя из базовых фактов. Таким образом, база знаний о родственных отношениях является дедуктивной. Отметим, что базовый набор отношений может быть выбран не единственным способом: например, вместо отношения *родитель* могут быть взяты отношения *мать* и *отец*.

Основная функция системы – ответы на запросы двух видов:

- Определить для двух конкретных членов семьи, в каком родственном отношении они находятся, например: *В каком родстве Елена и Петр?*
- Определить для заданного члена семьи, кто состоит с ним в конкретном родственном отношении: например: *Кто является сестрой Ольги?* или *Кто внуки Андрея?*

В случае вопросов о родственном отношении двух членов семьи, не находящихся в близком родстве, необходимо в ответе произвести «синтез» искомого отношения из нескольких известных системе отношений, например: *Елена – тётя внука Петра* или *Елена – тётя Николая, внука Петра*.

Система также должна допускать модификацию базы знаний в диалоге с пользователем: введение в неё новой информации о членах семьи и коррекцию старой информации. При этом в случае ввода новых утверждений о родственных отношениях, не являющихся базовыми, они должны быть «разложены» системой на более простые (базовые), которые и заносятся в базу знаний. Для проверки реализованной системы (при сдаче задания) должен быть файл со схемой (деревом) родственных связей.

Для корректной работы системы все члены семьи должны иметь разные имена. Чтобы исключить противоречия в базе знаний, желательно, чтобы при вводе в неё новой информации система проверяла её непротиворечивость по отношению к текущему состоянию базы и осуществляла только ввод допустимых фактов.

Интерфейс с пользователем может быть организован с помощью стандартных средств: меню, форм для ввода/вывода, или же простых фраз естественного языка (в этом случае необходим синтаксический анализ запросов пользователя путём выделения в них ключевых слов). Для корректного вывода русских имён в ответах системы можно встроить в неё список имён, с указанием их именительного и родительного падежей (именно эти падежи используются в запросах указанных выше видов и ответах на них).

Укажем некоторые отношения родства, которые могут быть представлены в системе: *зять* – муж дочери; *золовка* – сестра мужа; *свояченица* – сестра жены; *шурин* – брат жены; *деверь* – брат мужа; *сноха* – жена сына (для его отца) или жены двух братьев друг другу.

2. Программа синтаксического анализа предложений естественного языка

Предлагается рассмотреть подмножество грамматически правильных предложений одного из европейских языков (английский, французский или др., русский язык исключается). В общем случае предложения состоят из группы подлежащего (именной группы) и группы сказуемого (глагольной группы). Группа подлежащего в свою очередь состоит из нескольких прилагательных и/или числительных, существительного или местоимения, а также, возможно, артикля. Группа сказуемого включает глагол в одной из личных форм и нескольких дополнений или обстоятельств в виде именных групп с предлогом или без, например: *A little boy played with a big dog in the garden for a long time*.

Рассматриваемое подмножество языка должно включать:

- единственное и множественное число существительных и глаголов;
- возможность нескольких прилагательных в качестве определений существительных;
- употребление наречий при глаголах и прилагательных;
- согласование в лице и числе именной группы в роли подлежащего и глагольной группы в роли сказуемого;
- грамматическое согласование составных частей самой именной группы и правильное употребление артиклей (если они есть в языке).

Построенная на базе указанной грамматики пролог-программа должна выполнять ***синтаксический анализ (разбор)*** поступающего на вход предложения естественного языка методом рекурсивного спуска. В результате для грамматически правильного предложения должно быть построено и визуализировано его дерево разбора; а для грамматически неверных предложений выданы диагностические сообщения. Если согласно грамматике возможно несколько альтернативных вариантов разбора исходного предложения, все деревья разбора должны быть построены и визуализированы.

Встроенный в программу словарь для анализируемого подмножества естественного языка должен включать не менее 15-20 единиц для каждой части речи (существительное, прилагательное, глагол и т.п.) и допускать расширения. Синтаксический разбор следует реализовать достаточно эффективно, с использованием разностных списков, без использования предиката *append* и порождения лишних точек бектрекинга.

Структура предложений может быть описана с помощью расширенной контекстно-свободной грамматики. Для сдачи задания дополнительно (вместе с программой) должен быть выслан файл с кратким описанием грамматики языка.

Способ составления программы с указанными свойствами описан в книге [*Братко И.* Алгоритмы искусственного интеллекта на языке PROLOG. – М., Вильямс, 2004, стр. 510-518], а также в Методических указаниях к варианту (см. **стр. 11**, **обязательно** !).

Усложнение рассмотренного варианта задания предполагает создание *системы автоматического перевода* с одного естественного языка на другой (например, с английского на французский). В этом случае результатом работы программы является не дерево разбора исходного предложения, а эквивалентное по смыслу предложение целевого языка. Реализация такой программы проще, если брать пару родственных языков (русский – украинский, испанский – итальянский и т.п.). Кроме описывающей исходный язык грамматики потребуется также описание преобразований вида:

грамматическая конструкция исходного языка → конструкция целевого языка.

3. Программа расчета сопротивления электрической цепи

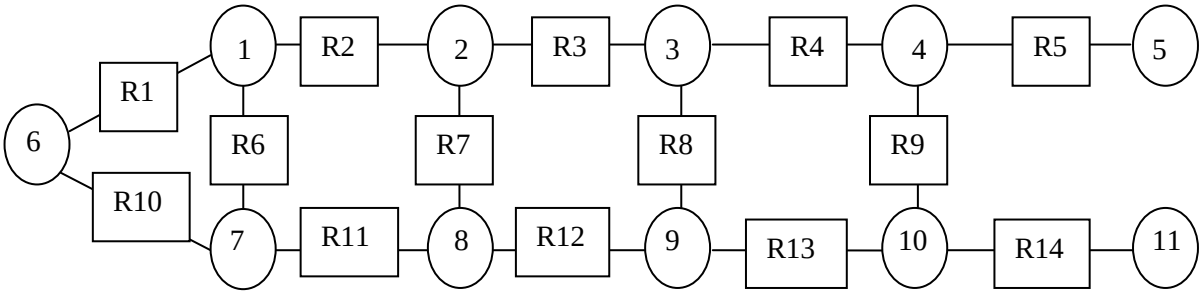
Задана конфигурация электрической цепи, состоящей из соединённых между собой сопротивлений (резисторов). Задаются также две точки цепи, к которым подключается источник тока (иные источники тока в сети отсутствуют), требуется рассчитать общее сопротивление заданной цепи.

Электрическая цепь задается как набор пролог-фактов вида:

connection (точка1, точка2, значение_сопротивления).

Каждый такой факт фиксирует две соседние точки сети и величину сопротивления между ними. Точки сети обозначаются латинскими буквами или же нумеруются, величина сопротивления задается целым или вещественным числом.

Пример цепи:

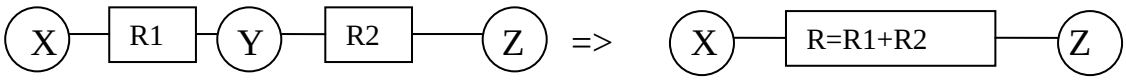


В качестве точек подключения источника тока могут быть взяты, например, точки 6 и 11.

Электрическая цепь может трактоваться как нагруженный неориентированный мультиграф, каждое ребро которого соответствует ровно одному сопротивлению с заданным значением. Предполагается вычислять общее сопротивление сети на базе законов Ома, путем последовательного ее преобразования и упрощения – до тех пор, пока в ней не останется одно единственное ребро, соединяющее две исходных точки подключения источника тока. Соответствующее этому ребру сопротивление и является результатом – величиной общего сопротивления.

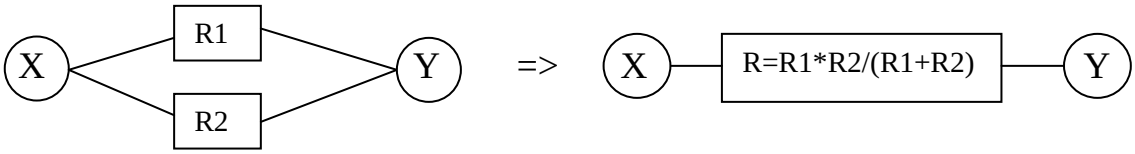
Правила преобразования графа электрической цепи:

1. Замена двух последовательно соединенных ребер одним

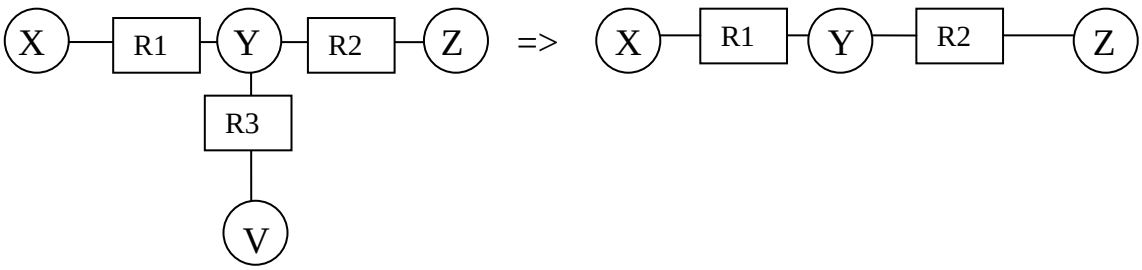


При этом вершина Y должна быть кратности 2 (т.е. из неё выходит ровно 2 ребра).

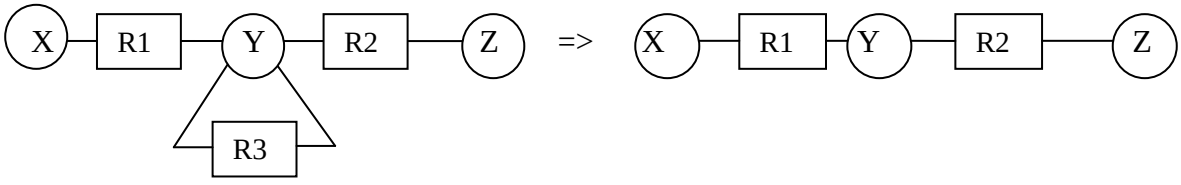
2. Замена двух параллельных ребер одним



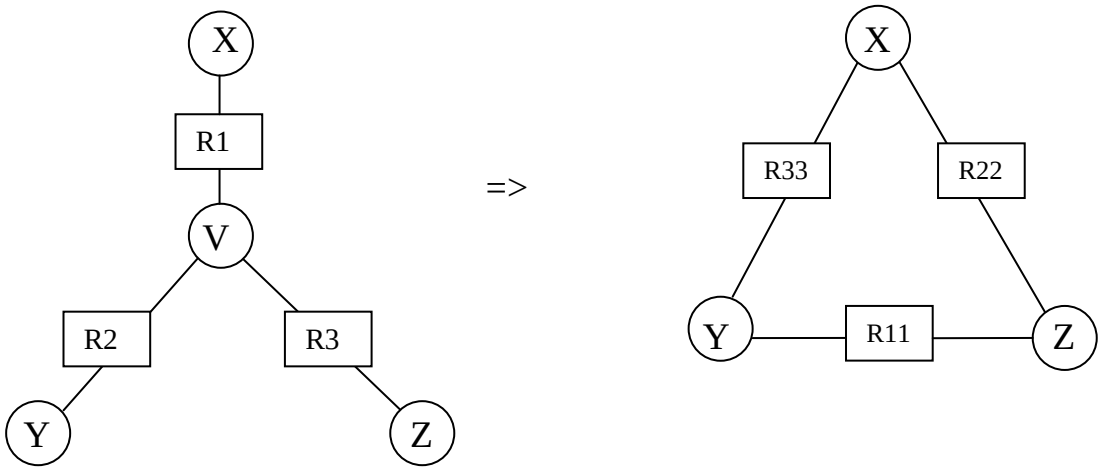
3. Удаление изолированной ветви – "висячего" ребра с сопротивлением R3, причем вершина V не может быть точкой подключения источника тока.



4. Удаление петли



5. Замена трехлучевой "звезды" сопротивлений на треугольник



при этом из вершины **V** выходит ровно 3 ребра и она не является точкой подключения источника тока. Сопротивления в треугольнике рассчитываются по формулам:

R11=(R1*R2+R1*R3+R2*R3)/R1
R22=(R1*R2+R1*R3+R2*R3)/R2
R33=(R1*R2+R1*R3+R2*R3)/R3

В общем случае преобразование электрической цепи по этим правилам включает несколько циклов, каждый цикл состоит из применения правил 1-4, затем преобразований по правилу 5, после чего снова могут оказаться применимыми правила 1-4 и 5.

В ходе преобразований может понадобиться возврат – в том случае, если для упрощения в какой-то момент была неудачно выбрана вершина кратности 3 для применения правила 5, после чего преобразования зашли в тупик. Тогда необходимо вернуться в точку выбора и выбрать другую вершину кратности 3. Поскольку при возврате необходимо восстановить конфигурацию точки выбора, а результаты предикатов `assert` и `retract` при возврате не отменяются, то преобразуемый граф цепи необходимо хранить в виде списка рёбер.

Также в ходе преобразований цепи необходимо отслеживать и исключать повторное возникновение одной и той же конфигурации сети (после нескольких преобразований).

Пользовательский интерфейс с программой расчета сопротивления цепи должен предусматривать ввод исходной конфигурации электрической цепи из входного файла, точки подключения источника тока должны запрашиваться у пользователя, после чего производятся расчеты и выдается ответ.

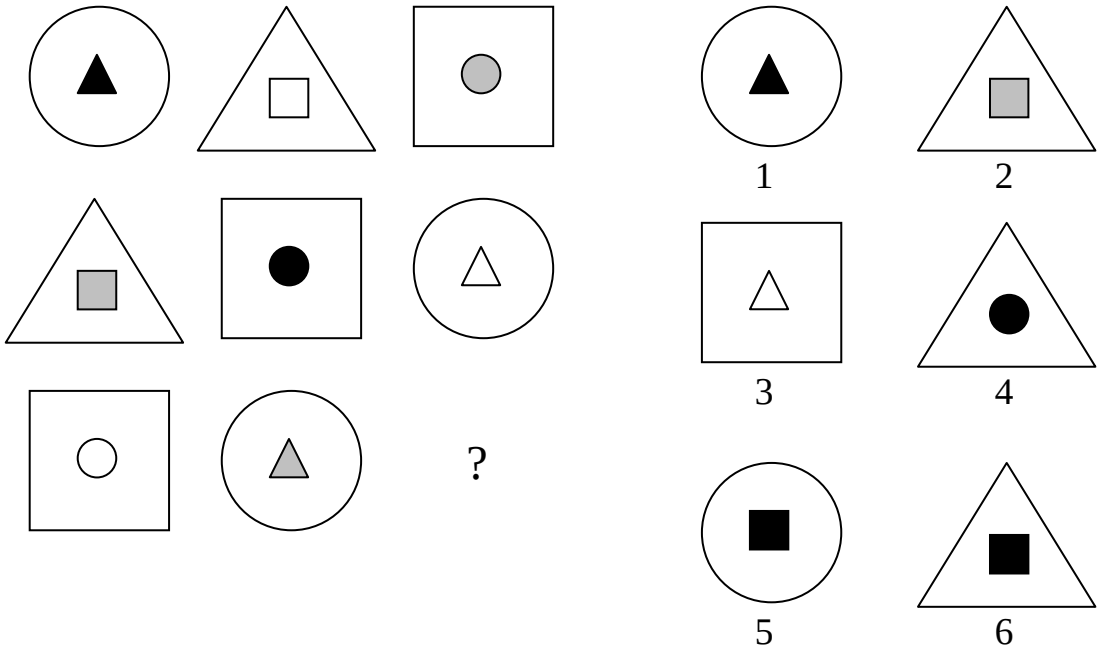
Следует произвести отладку программы на цепях разной конфигурации и с разным числом точек соединений/узлов графа (от 7 до 20), предусмотрев для отладки возможности удобного редактирования цепи. Набор примененных тестов должен быть выслан/показан вместе с программой.

Усложнение данного варианта задания предполагает визуализацию заданной изначально конфигурации электрической цепи.

4. Система генерации геометрических головоломок

В головоломках рассматриваемого вида требуется найти закономерность в наборе из нескольких *составных геометрических фигур*. Такие фигуры могут состоять из нескольких простых фигур и их частей, например: точка, квадрат, треугольник, прямоугольник, круг, дуга, прямые и ломаные отрезки линий и др. Составляющие фигуры могут быть соединены различным образом: могут быть вложены друг в друга, могут пересекаться или соприкасаться. Кроме того, их контуры могут быть разного цвета, а их внутренняя часть – заштрихованной или цветной.

Кроме обнаружения закономерности в заданном наборе фигур головоломка предполагает также определение некоторой недостающей фигуры. Искомая фигура содержится среди *вариантов ответа* – нескольких пронумерованных составных фигур, которые также включаются в головоломку. Ниже приведен пример головоломки, в которой требуется по закономерности построения двух первых (левых) рядов из трёх фигур найти фигуру, обозначенную вопросительным знаком и завершающую последний ряд фигур согласно найденной закономерности. Среди вариантов ответа (на рисунке показаны справа) всегда есть правильный; в рассматриваемом примере правильным ответом является фигура 6. В общем случае возможны несколько правильных ответов, тогда их все целесообразно включить в головоломку в качестве вариантов ответа.



Возможны другие виды геометрических головоломок с составными фигурами, например, с б'ольшим числом рядов и/или более сложными фигурами (схематическими изображениями человечков или животных и др.).

Система генерации головоломок должна строить набор составных фигур без некоторой недостающей фигуры, а также варианты ответа, включающие эту фигуру. Построенные наборы показываются пользователю, и он имеет возможность:

- либо запросить решение головоломки, если он затрудняется решить её сам, и тогда ему показывается её правильное решение (или несколько возможных решений);
- либо самостоятельно решить головоломку – при этом проверяется правильность его решения с последующей выдачей ему соответствующего сообщения; в случае верного решения могут быть показаны другие возможные решения, в случае же неверного решения показывается правильное решение (решения).

Важно, чтобы сгенерированные системой составные фигуры, предлагаемые как варианты ответа на головоломку, были в достаточной степени похожи на фигуру, представляющую правильный ответ (иначе решение головоломки существенно упрощается). Для этого можно брать в качестве вариантов ответа фигуры, отличающиеся всего одним-двумя элементами от фигуры, являющейся решением головоломки.

Сложность головоломки, как и возможное их число, зависит от количества параметров-элементов, образующих составные геометрические фигуры. Необходимо предусмотреть не менее 5-7 различных типов параметров-элементов: простых фигур, их взаимного расположения, цвета и др. Целесообразно ввести несколько уровней сложности генерируемых головоломок, в зависимости от количества этих параметров-элементов в фигурах – с тем, чтобы пользователь мог устанавливать нужный ему уровень.

Требуется предусмотреть возможность решения пользователем за сеанс работы с системой нескольких головоломок, при этом должны порождаться разные задачи, причём в разные сеансы последовательности генерируемых задач должны быть разными. Для этого следует в процессе генерации задачи производить случайный выбор параметров-элементов, определяющих составную фигуру, используя для этого датчик случайных величин и доступное программе значение, заранее неизвестное: дату и/или текущее время.

5. Вопросно-ответная система по онтологии предметной области

Центральным модулем системы должна быть ***база знаний***, в которой представлена онтологическая информация об объектах некоторой предметной области и их классах. ***Онтология предметной области*** представляет собой инвентарь сущностей (объектов и понятий) этой области, их описаний и описаний их связей. Объекты онтологии обычно объединяются в *классы*, по схожести их свойств. К числу основных связей, представленных в онтологиях, относятся связь *род-вид* (класс-подкласс), связь *часть-целое*, связь *класс-экземпляр класса* (класс-объект).

Онтологии, фиксирующие только отношение род-вид, называются *таксономиями*. Классической таксономией является система классов в биологии – растений и животных; в виде таксономии могут быть представлены знания о музыкальных инструментах, пищевых продуктах и напитках, оружии и т.п.

Объекты любого класса характеризуются определенным набором *атрибутов* (свойств) и их значений, причем атрибуты класса часто наследуются объектами подклассов, и потому такие наследуемые атрибуты, как правило, описываются в самом общем классе цепочки наследования. Наследование свойств происходит обычно по связям *род-вид* (класс-подкласс) и *класс-экземпляр*, и в ряде случаев – по связи *часть-целое*.

Многие атрибуты объектов задаются явно своими значениями (например, цвет животного), значения других атрибутов могут вычисляться по значениям других атрибутов. Для некоторых атрибутов в некоторых классах может быть установлено специальное *значение по умолчанию* – обычно это стандартное значение для объектов описываемого класса (например, по умолчанию, птицы умеют летать).

Функции разрабатываемой вопросно-ответной системы включают:

- Вычисление ответов на запросы следующих видов:
  – для конкретного объекта/класса онтологии указать его атрибуты (свойства);
  – определить значение нужного свойства у конкретного объекта/класса;
  – для двух заданных объектов/классов определить отличающие их свойства.
- Расширение в диалоге с пользователем базы знаний: введение в неё новых объектов и классов и/или коррекция старой информации.

Мощность итоговой базы знаний должна включать не менее 15 различных классов объектов, различающихся по не менее 5-7 признакам. Описание схемы классов и признаков отсылается/предъявляется вместе с реализованной системой.

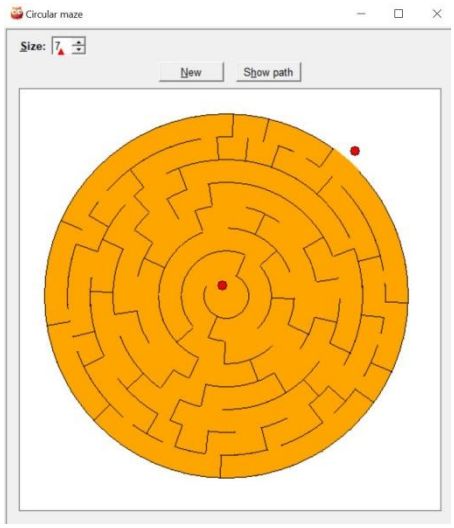
Вопросно-ответная система должна обладать возможностями *логического вывода*, который позволяет вычислять значения атрибутов (свойств) объектов и классов, которые не представлены явно в их описаниях. Необходимо реализовать две различные *стратегии наследования атрибутов*, различающиеся порядком движения по цепочке подкласс-класс и рассмотрения свойств по умолчанию. При первой стратегии сначала делается попытка найти или отнаследовать нужное свойство по цепочке классов наследования, без использования значений по умолчанию. При второй стратегии находится самое ближайшее, представленное в цепочке классов наследования значение нужного свойства, включая значение по умолчанию.

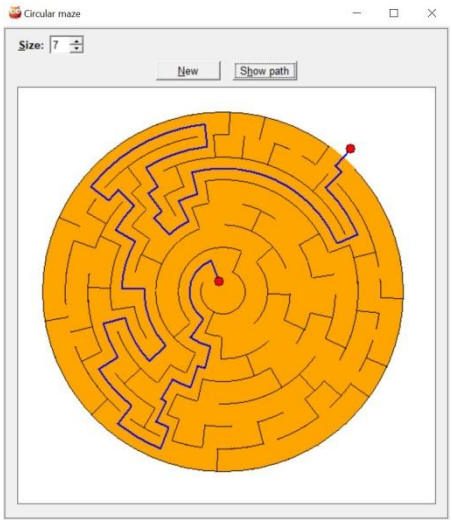
Усложнение рассмотренного варианта задания предполагает реализацию *множественного наследования*, которое характерно для ряда предметных областей и требует разработки дополнительных стратегий логического вывода.

Пользовательский интерфейс вопросно-ответной системы может быть организован с помощью меню, форматов для ввода/вывода, или же простых фраз естественного языка. В последнем случае требуется проведение частичного синтаксического анализа запросов пользователя путем выделения в них ключевых слов.

6. Программа построения кругового лабиринта

Рассматриваются лабиринты, расположенные в круговой области и состоящие из стенок вдоль концентрических окружностей и вдоль отрезков хорд, соединяющих эти окружности. Каждый лабиринт имеет один *вход* на внешней стороне круговой области. Размер лабиринта задается числом концентрических окружностей (или числом круговых проходов по лабиринту). Такой лабиринт может быть получен из сплошной сетки стенок из окружностей и хорд, путем удаления одной внешней стенки и некоторого количества внутренних стенок. Пример лабиринта и пути в нем:





Лабиринт проходим, если внутри него между стенками существует путь, ведущий от входа в его центр (или наоборот). Будем говорить в этом случае, что лабиринт имеет решение. *Решение единственно*, если среди всех таких путей есть минимальный по длине путь, являющийся частью всех остальных путей. Проход лабиринта *достижим*, если существует путь, соединяющий его с входом или центром. Тот факт, что все проходы лабиринта достижимы, означает, что в лабиринте отсутствуют замкнутые области.

Требуемая пролог-программа должна по заданному размеру R (числу круговых проходов, $4 \le R \le 20$) построить лабиринт, который имеет единственное решение и состоит только из достижимых клеток. Программа визуализирует построенный лабиринт, отмечает в нём вход и центр и по указанию пользователя высвечивает путь между ними. Желательно, чтобы построенный лабиринт был интересным – содержал достаточно большое число внутренних стенок и почти замкнутых комнат (областей), чтобы путь в нём был достаточно извилистым.

Программа должна уметь строить в одном сеансе несколько лабиринтов разных размеров, причем лабиринты одной и той же размерности должны быть разными. Для этого необходимо при генерации лабиринтов использовать случайные величины. Лабиринты больших размеров должны строиться за приемлемое время.

Заметим, что стратегии генерации лабиринта могут быть различны: удаление внутренних стенок исходной сплошной сетки или же их добавление в изначально пустой внешний круг; при этом можно сначала построить путь, а затем достроить оставшуюся часть лабиринта. При любой стратегии единственность решения (пути), как и достижимость клеток лабиринта, целесообразно проверять как можно раньше, либо же гарантировать выполнение этих требований самим методом построения лабиринта.

7. Система-планировщик вычислений

Система реализует планирование вычислений заданных N программ на нескольких вычислительных серверах, работающих одновременно и независимо, но отличающихся производительностью. Возможность выполнения некоторых программ зависит от результатов вычисления других (поскольку они используют их выходные результаты как свои входные данные). Производительность одних серверов отличается в 2 раза от производительности других: есть одноядерные и двухядерные сервера (все ядра одной мощности). В каждый момент на сервере выполняется только одна программа.

Исходной информацией для планирования является:

- количество программ, которые надо выполнить ($5 \le N \le 12$), вместе с указанием для каждой программы длительности ее выполнения на одном ядре;
- количество серверов ($3 \le M \le 7$) с указанием количества ядер для каждого из них;
- зависимости программ по результатам: список пар программ, для которых выходные результаты первой программы необходимы для вычисления второй.

Система-планировщик должна составлять расписание/план вычислений, начинающийся от общей точки запуска всех серверов, время можно отсчитывать в условных единицах. Требуется найти варианты загрузки серверов, при которых для вычисления всех программ тратится минимальное время и/или вычислительные серверы загружены максимально (минимально суммарное время простоя).

Пользовательский интерфейс должен быть удобен как для просмотра исходной для планирования информации (число программ и серверов, их характеристики, зависимости программ), так и результатов планирования.

Построенный план должен включать для каждого сервера время запуска очередной программы, ее идентификатор и время ее окончания, а также общую загрузку сервера. По запросу следует выводить общее время выполнения всех программ, суммарное время простоя серверов, среднюю загрузку серверов.

Для сдачи задания вместе с программой планировщика должен быть выслан набор тестов, использованных при ее проверке.

Возможные усложнения задачи:

- ✓ Наличие серверов с разным числом ядер (одной мощности); можно считать, что на k ядрах программа считается в k раз быстрее, чем на одном ядре.
- ✓ Возможность выполнять на сервере несколько программ (каждая может занимать несколько ядер).

8. Компьютерная игра

Предлагается выбрать для реализации игру из класса ***игр двух лиц*** (игроков) ***с полной информацией*** – к этому классу относятся, к примеру, шахматы, шашки, реверси.

В играх двух лиц с полной информацией выбор компьютером очередного хода осуществляется на основе просмотра фрагмента дерева игры, отображающего все возможные ходы игрока и ответы противника. При этом применяется *минимаксная процедура* поиска достаточно хорошего хода от заданной игровой позиции, а чаще – её более эффективная модификация, известная как ***альфа-бета процедура***. Для оценки концевых узлов (листьев) дерева игры используется так называемая *статическая оценочная функция*, оценивающая позицию игры как таковую без учёта её продолжений.

Реализуемая игра должна быть достаточно сложной, т.е. исключающая (или сильно затрудняющая) практическую возможность полного просмотра дерева игры и обнаружения выигрышной стратегии (если таковая существует). С другой стороны, игра не должна быть слишком сложной, с сильно ветвистым деревом игры (как, например, полная игра в шашки или шахматы). Рекомендуется выбирать шашки без дамок, калах, шахматный эндшпиль, реверси, крестики-нолики на неограниченной доске и т.п.

Альфа-бета процедура основана на частичном просмотре возможных продолжений игры на заданное количество *D* ходов игроков, т.е. просмотре дерева игры от заданной игровой позиции на глубину дерева *D*, и оценки возможных игровых позиций с помощью статической оценочной функции. В отличие от минимаксной процедуры, решающей ту же задачу, альфа-бета процедура выполняет одновременный просмотр дерева и оценку его вершин, что позволяет сократить часть работы.

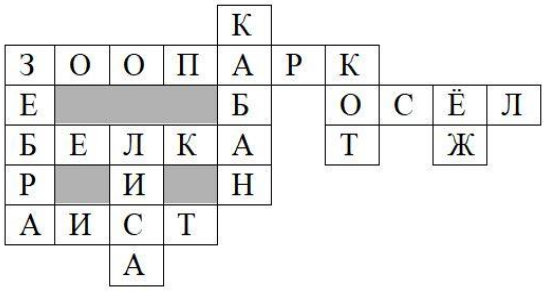
В реализуемой компьютерной игре необходимо предусмотреть возможность изменения (перед началом игры) *глубины просмотра дерева игры* альфа-бета процедурой: $2 \le D \le 5$, шаг глубины соответствует ходу одного игрока. Просмотр дерева игры на 2-3 хода вперед должен выполняться за приемлемое время. Целесообразно реализовать случайный выбор хода из нескольких равноценных (чтобы с программой было интересно играть).

Игровая программа должна уметь играть как против человека (пользователя), так и против самой себя или другой подобной игровой программы. Пользовательский интерфейс должен быть понятным и удобным и включать установку уровня сложности игры (глубины просмотра дерева игры), отображение текущей позиции игры, показ предыдущих позиций. Усложнение данного варианта задания предполагает графическую визуализацию хода игры.

9. Программа построения головоломок крисс-кросс

Головоломка крисс-кросс похожа на кроссворд, но предлагает задачу более простую: для заданного набора слов и заданной схемы, подобной сетке кроссворда, требуется вписать в схему все слова набора.

Схема состоит из пересекающихся, но не соприкасающихся линий (столбцов и строк) из клеток; схема должна быть *связной*, т.е. каждая линия должна иметь хотя бы одно пересечение с другой. Пример схемы со вписанными словами:



Программа составления головоломки, получая на вход набор из 7-12 различных слов одного из естественных языков (русского или английского) должна построить подходящую для них схему или несколько схем. *Схема построена правильно*, если она является связной и все слова набора могут быть вписаны в нее (после чего не остается пустых клеток), причем возможен лишь один вариант заполнения схемы (головоломка имеет единственное решение).

Если для входного набора слов не существует решения-схемы, то программа должна сообщить об этом, указав по возможности причину неудачи (например, наличие в наборе слова, не имеющего общих букв с другими словами). В остальных случаях построенная схема должна быть изображена на экране компьютера, и по запросу пользователя может быть показано решение головоломки, т.е. заполнение схемы исходными словами.

В случаях, когда для заданного набора слов возможно несколько схем, предпочтение следует отдавать более компактным и связным схемам. Связность схем зависит от количества пересечений в ней; компактность схемы определяется площадью наименьшего объемлющего ее прямоугольника. Эти критерии оценки схемы рекомендуется формализовать тем или иным способом (в виде эвристической функции) и использовать при построении правильных схем.

Отметим, что длина слова из набора и количество слов одинаковой длины могут быть ключом как к разгадке самой головоломки, так и к написанию программы их составления. Эффективность перебора вариантов схем при построении правильной схемы может зависеть

от порядка, в каком рассматриваются слова исходного набора. В любом случае при построении схемы возникает существенный перебор, и имеет смысл применять эвристическое отсечение некоторых вариантов частично построенных схем, поскольку в ином случае построение всей схемы может оказаться слишком длительным (программа должна строить схему для 10-12 слов за приемлемое время). По этой же причине проверку условия единственности решения головоломки целесообразно проводить как можно раньше в ходе построения схемы.

10. Система составления расписания курсов

Назначение системы – составление недельного расписания занятий для курсов иностранного языка (английского), занятия либо утренние, либо вечерние, проводятся 2 раза в неделю (по 2 часа) для каждого обучаемого. Занятия могут быть в любой день недели, включая субботу и воскресенье.

Исходной информацией для составления расписания являются:

- заявки на обучение от N человек ($7 \le N \le 20$), с указанием их фамилии и желательным временем обучения (утро, вечер);
- количество M преподавателей ($3 \le M \le 9$), для каждого известна его фамилия и предпочтительное ему время занятий (утреннее/вечернее/без предпочтений) и допустимая недельная загрузка.

При генерации расписания курсов должны быть организованы группы утреннего и вечернего обучения, согласно заявкам (в группе может быть от 3 до 7 обучаемых, каждой группе назначается номер) и должны быть учтены следующие требования:

- у группы не может быть 2 занятия в один день, а дни занятий идти друг за другом (т.е. между днями занятий должен быть хотя бы один день без занятий);
- у преподавателя не могут быть в один день утренние и вечерние занятия.

Пользовательский интерфейс системы должен быть удобен для просмотра исходных данных (заявки, информация о преподавателях) и сгенерированного системой расписания учебных групп. Построенное расписание должно выводиться либо для заданного дня недели или же для всей недели сразу, причем для каждого дня недели построчно указываются сначала утренние занятия групп, с указанием номера группы и преподавателя, а затем – аналогичная информация для вечерних занятий. Отдельно можно просмотреть распределение обучаемых по группам с указанием их преподавателя.

Поскольку в общем случае возможны несколько вариантов расписания занятий (отличающиеся размером групп, распределением по дням и др.), то должна быть предусмотрена возможность их упорядоченного вывода. В случае невозможности построения расписания следует выдать сообщение с указанием причины.

Возможные усложнения задачи:

1) В субботу и воскресенье кроме утренних и вечерних возможны дневные занятия.

2) В заявках на обучение кроме времени дня указываются предпочтительные (или же нежелательные) дни занятий.

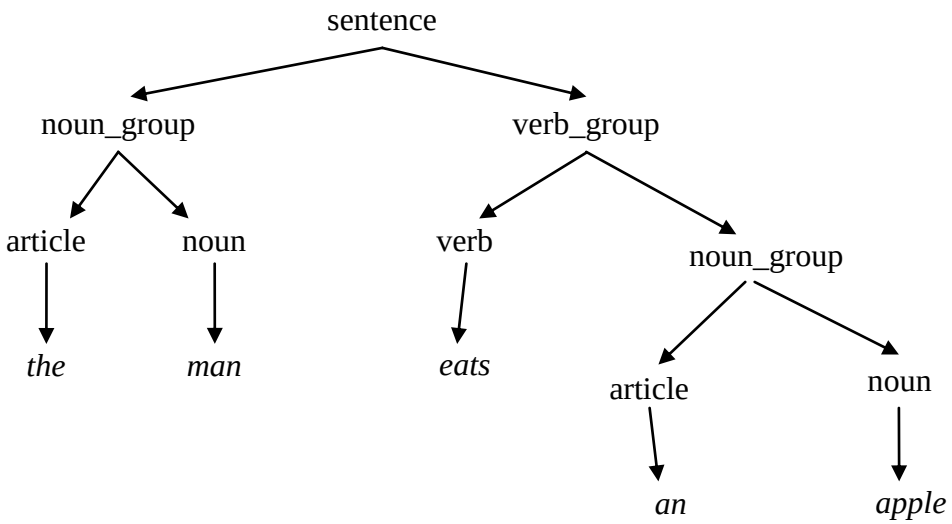
3) Курсы проводят обучение 2 или 3 разным иностранным языкам, соответственно в заявке на обучение указываются требуемые языки, а для каждого преподавателя – языки, которые он может преподавать.

Методические указания для Варианта 2 (синтаксический анализ ЕЯ)

Один из способов описания синтаксиса языка – формальные грамматики. Рассмотрим контекстно-свободную грамматику для простого подмножества английского языка. Грамматика определяет внутреннюю структуру предложений языка и состоит из правил, которые описывают группы слов (словосочетания), составляющие предложения языка. В отличие от общепринятого, нетерминальные символы грамматики обозначим именами, начинающимися со строчной буквы, а терминальные символы запишем в квадратных скобках. Нетерминалы в правых частях правил грамматики (составляющие) записываются через запятую, а в конце правил ставится точка.

```
sentence → noun_group, verb_group.
noun_group → article, noun.
verb_group → verb.
verb_group → verb, noun_group.
article → [the].
article → [a].
article → [an].
verb → [likes].
verb → [eats].
noun → [man].
noun → [cat].
noun → [apple].
```

Согласно этой КС-грамматике, дерево синтаксического разбора для английского предложения *the man eats an apple* выглядит так:



Это дерево можно представить в виде прологовского терма, отображающего структуру предложения (функторы-имена нетерминалов сокращены до двух букв):

```
st(ng(a(the), n(man)), vg(v(eats), ng(a(an), n(apple))))
```

Рассмотренная грамматика может быть преобразована в программу на языке Пролог. Пусть любому нетерминальному символу соответствует предикат (унарный), который выполняет проверку грамматической правильности соответствующей группы (составляющей) предложения, т.е. истинный, если его аргумент представляет такую группу, например:

`sentence(X)` истинно, если `X` – грамматически правильное предложение нашего языка:

```
?- sentence([the,man,likes,the,cat]) → true
```

`noun_group(X)` истинно, если `X` – грамматически правильная группа существительного:

```
?- noun_group([the, man]) → true
```

Таким образом, на вход программы (главный предикат `sentence`) поступает предложение языка в виде прологовского списка (слова предложения – прологовские атомы):

[the, man, likes, the, cat] . Аргументами остальных предикатов также будут списки.

Каждое правило исходной КС-грамматики необходимо записать на Прологе, определив пролог-правило для соответствующего предиката. Правило грамматики для предиката sentence должно выражать следующее: последовательность слов является предложением, если её можно разбить на две подпоследовательности Y и Z такие, что Y – группа существительного, а Z – группа глагола. Для разбиения можно использовать известный предикат append , тем самым :

```
sentence(X) :- append(X, Y, Z), noun_group(Y), verb_group(Z).
```

Аналогично для остальных нетерминалов:

```
noun_group(X) :- append(X, Y, Z), article(Y), noun_group(Z).
verb_group(X) :- verb(X).
verb_group(X) :- append(X, Y, Z), verb(Y), noun_group(Z).
```

Словарные правила грамматики (задающие слова языка) могут быть записаны на Прологе как факты:

```
article([the]).
article([a]).
article([an]).
verb([likes]).
verb([eats]).
noun([man]).
noun([cat]).
```

В итоге получается программа-анализатор на Прологе, который распознаёт (и только) предложения языка, порождаемого грамматикой. Этот синтаксический анализатор реализует нисходящий рекурсивный разбор слева направо, в процессе которого при необходимости производится возврат. Каждое пролог-правило анализатора выполняет сначала разбиение своего аргумента – исходного списка слов на два подсписка, которые затем проверяются на грамматическую правильность. При этом порождаются всевозможные варианты разбиения, большая часть которых бесполезна и отбрасывается в ходе дальнейших проверок.

Полученная программа структурно похожа на исходную грамматику: каждому правилу грамматики соответствует пролог-предикат и определяющее его пролог-правило схожей структуры, единственным отличием является применение append, но именно он есть источник ненужного перебора. Более эффективное решение (без append) основано на следующей ключевой идее, опишем ее на примере предиката noun_group. Поскольку предикат noun_group ответственен за группу существительного, то пусть он сам определяет, какая начальная часть предложения является такой группой, оставляя оставшуюся часть предложения для последующей обработки предикатом verb_group. Чтобы реализовать эту идею, предикат должен иметь два аргумента: noun_group(X,Y), первый аргумент – исходное предложение, второй – его остаток после выделения группы существительного:

```
?- noun_group([the,man,likes,the,cat], [likes,the,cat]). → true
```

Аналогично строятся и другие предикаты, тем самым получаем программу:

```
sentence(S, S2) :- noun_group(S, S1), verb_group(S1, S2).
noun_group(S, S2) :- article(S, S1), noun(S1, S2).
verb_group(S, S1) :- verb(S, S1).
verb_group(S, S2) :- verb(S, S1), noun_group(S1, S2).
article([the | S], S).
article([a | S], S).
article([an | S], S).
verb([likes | S], S).   и так далее.
```

Для главного предиката sentence(S, S2) полученной программы условие истинности формулируется так: если в начале заданной последовательности слов S есть грамматически правильное предложение, а S2 – остаток после него, например:

```
?- sentence([the, man, likes, the, cat], []) → true
?- sentence([the, man, likes, the, cat, apple], [apple]) → true
```

Интерпретация соответствующего пролог-правила для `sentence(S, S2)` такова: в списке слов `S` есть правильное предложение, если от начала `S` до начала списка `S1` расположена группа существительного, а между началом `S1` и `S2` стоит группа глагола.

Получившаяся программа более эффективна (по сути, она использует так называемые *разностные списки* Пролога), но она не строит дерево разбора для грамматически верных предложений. Для построения дерева требуется добавить во все предикаты программы еще один аргумент, в котором в ходе разбора будет формироваться это дерево. Соответствующее правило для главного предиката:

```
sentence(S,S2, st(NG, VG)) :- noun_group(S,S1, NG), verb_group(S1,S2, VG).
```

Интерпретация этого правила такова: для того, чтобы выполнить разбор предложения, необходимо найти в нем группу существительного, за которой следует группа глагола, а затем объединить деревья разбора этих двух групп (составляющих предложения), используя функтор `st`. Аналогичным образом в третьем аргументе остальных предикатов необходимо указать, каким образом дерево для группы конструируется из деревьев ее частей (составляющих):

```
noun_group(S, S2, ng(A, V)) :- article(S, S1, A), noun(S, S2, N).
verb_group(S, S1, vg(V)) :- verb(S, S1, V).
verb_group(S, S2, vg(V, NG)) :- verb(S, S1, V), noun_group(S1, S2, NG).
article([the | S], S, a(the)).
article([a | S], S, a(a)).
article([an | S], S, a(a)).
verb([likes | S], S, v(likes)).
verb([eats | S], S, v(eats)).
noun([man | S], S, n(man)).
```

и так далее.

Как видно, словарные правила (задающие слова языка) получаются достаточно громоздкими, но их можно упростить, если общую информацию о том, как строится дерево для каждой части речи, хранить один раз, например, для существительного:

```
noun([N | S], S, n(N)) :- is_noun(N).
is_noun(man).
is_noun(cat).
```

Важный вопрос при построении пролог-программы синтаксического разбора – реализация грамматического согласования: существительного с глаголом, артикля с существительным и др. Опять же, это возможно с помощью дополнительного аргумента, равенство значений которого для разных составляющих обеспечивается в ходе доказательства и унификации. В частности, для главного предиката предложения это реализуется так:

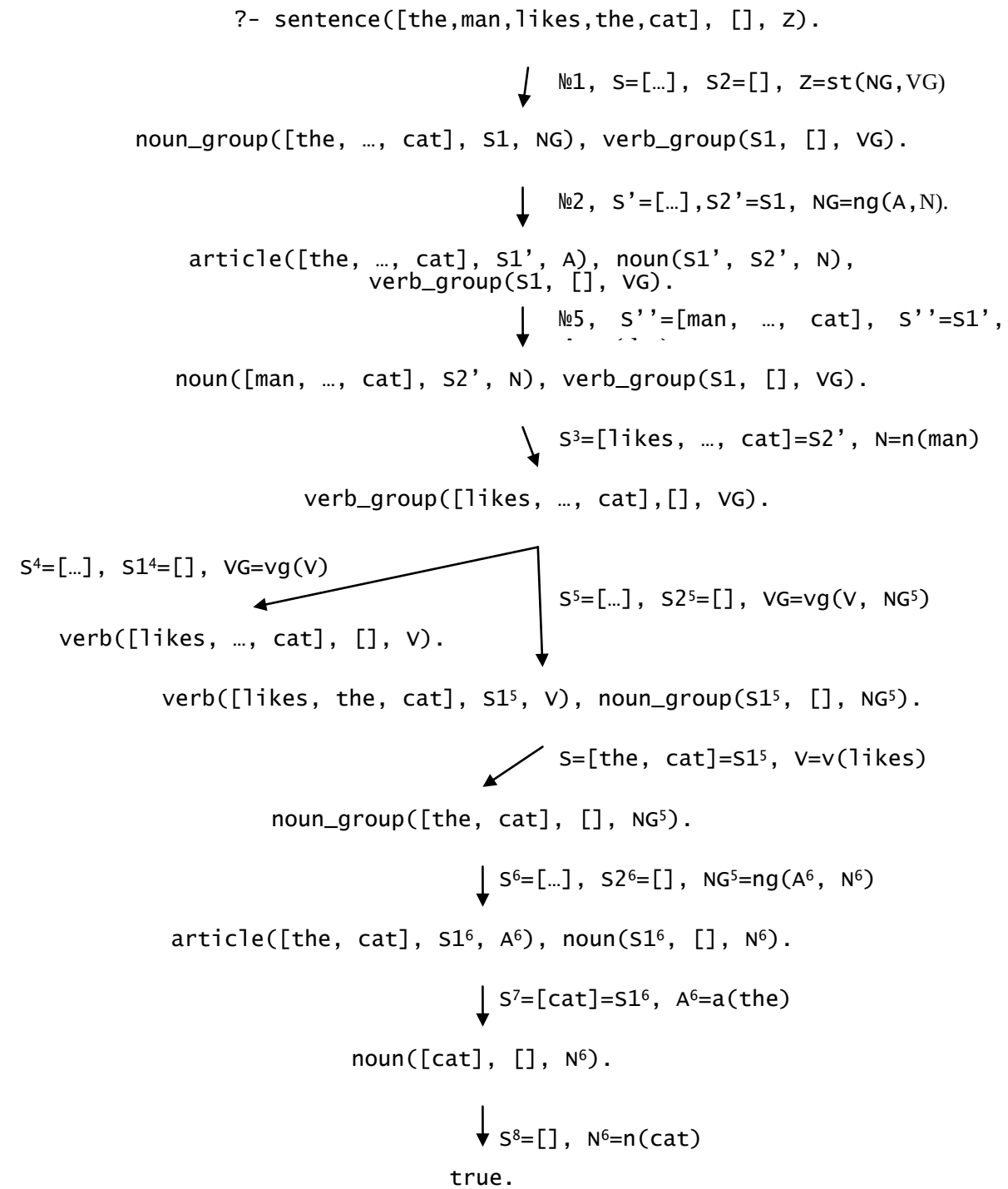
```
sentence(S,S2,st(NG,VG),N):- noun_group(S,S1,NG,N), verb_group(S1,S2,VG,N).
```

Аналогично реализуется согласование во всех остальных пролог-правилах для групп предложения, но для словарных правил грамматическое число (единственное, множественное задается явным образом, в частности::

```
is_noun(man,sing)      но   is_noun(men,plur).
is_verb(eats,sing)     но   is_verb(eat,plur).
```

и так далее.

В заключение приведем **пример дерева вывода** для задачи синтаксического анализа:



Итоговое значение для Z строится так:

Z = st(NG, VG) = st(ng(A, N), VG) = st(ng(a(the), n(man)), VG) =
st(ng(a(the), n(man)), vg(V, NG^5))

vg(V, NG^5) = vg(v(likes), ng(A^6, N^6)) = vg(v(likes), ng(a(the), n(cat)))