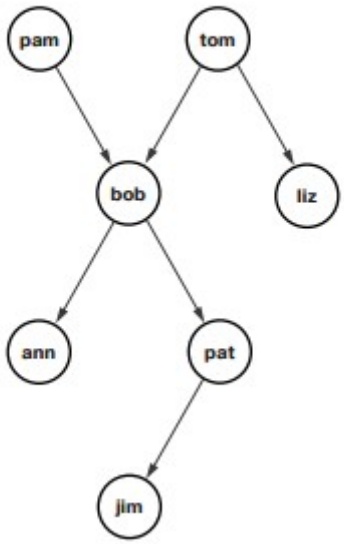
1. Основы языка Prolog

- **Основная идея**
  Prolog (“PROgramming in LOGic”) — декларативный язык программирования общего назначения. Prolog был создан с целью сочетать использование логики с представлением знаний. Он используется в системах обработки естественных языков, исследованиях искусственного интеллекта, экспертных системах, онтологиях и других предметных областях, для которых естественно использование логической парадигмы. Главной парадигмой, реализованной в языке Prolog, является логическое программирование.
- **Способ представления проблемы в языке**
  Пролог — это ЯП программирования для символических, нечисловых вычислений. Он особенно хорошо приспособлен для решения проблем которые касаются объектов и отношений между ними. Пример семейные отношения:

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
```



2. Элементы языка

Программа на прологе состоит из предложений. Предложения трех видов: факты, правила, вопросы. Все предложения строятся из термов.
Термы в свою очередь делятся на простые термы и термы структуры.
Простые в свою очередь на константы и переменные.
Константы делятся на атомы и числа (константы - это поименованные конкретные объекты или отношения)
Терм является синтаксической единицей.

Факты в языке Пролог описываются логическими предикатами с конкретными значениями. Факты в базах знаний на языке Пролог представляют конкретные сведения (знания). Обобщённые сведения и знания в языке Пролог задаются правилами логического вывода (определениями) и наборами таких правил вывода (определений) над конкретными фактами и обобщёнными сведениями. Предложения с пустым телом называются фактами.
Пример факта: `Кот(Иван).`
Оно эквивалентно правилу: `Кот(Иван) :- ИСТИНА.`

- **Атомы**
  Атом есть любая последовательность символов, заключенная в одинарные кавычки. Кавычки опускаются, если и без них атом можно отличить от символов, используемых для обозначения переменных.
- **Структуры**
  Структура - это единый объект состоящий из совокупности других объектов, называемых компонентами.
  Компоненты в свою очередь могут быть также структурами. Название структуры стоит перед скобками,а компоненты внутри скобок, через запятую. Пример: `data(27,april,1992)`. Структуру можно представить в виде дерева.
- **Переменные**
  Понятие переменной в Прологе отличается от принятого во многих языках программирования. Переменная не рассматривается как выделенный участок памяти. Она служит для обозначения объекта, на который нельзя сослаться по имени. Переменную можно считать локальным именем для некоторого объекта.

  Переменные служат для обозначения объектов, значения которых меняются в ходе выполнения программы.
  Имена переменных могут начинаться:
  - или с прописной буквы
  - или с символа подчеркивания
  Если значение переменной не интересует, то можно использовать анонимные переменные в виде символа подчеркивания '_'. Значение анонимной переменной не выводится на печать. Если несколько анонимных переменных, то они все разные. Использование анонимных переменных позволяет не выдумывать имена переменных, когда не надо.

- **Правила**
  Правила позволяют вам вывести один факт из других фактов. Другими словами, можно сказать, что правило - это заключение, для которого известно, что оно истинно, если одно или несколько других найденных заключений или фактов являются истинными.
  Пример: `Вывод :- Условие.`
  Читаются как: `Заголовок ИСТИНА, если Тело ИСТИНА`. Тело правила содержит ссылки на предикаты, которые называются целями правила.

  Левую часть правила называют головой предложения, а правую – телом предложения. В теле предложения перечисляют условия, определяющие вывод заключения. Если условия имеют между собой конъюнктивную связь, то между ними ставится запятая “,”. Если условия в правиле имеют между собой дизъюнктивную связь, то между ними ставится точка с запятой (“;”). Голова предложения всегда сдвинута влево относительно перечня условий. Каждое условие начинается с новой строки.

  Пример: `offspring(Y, X) :- parent(X, Y)` (если X является родителем Y, то Y является сыном или дочерью Y)

3. Принцип поиска доказательства

- **Понятие конкретизации переменных**
- **Понятие задачи и подзадачи**

4. Процедурные и декларативное значение

Различие между декларативными и процедурными прочтениями состоит в том, что последнее определяет не только логические соотношения между головой предложения и целями в его теле, но и задает порядок, в котором должны обрабатываться цели.

- **Идея декларативного значения**
  Декларативное значение программ позволяет установить, является ли заданная цель истинной, а в случае положительного ответа — При каких значениях переменных она является истинной. Чтобы точно определить декларативное значение, необходимо ввести понятие экземпляра предложения. Экземпляр предложения *C* называется предложение *C*, в котором вместо каждой из его переменных подставлен некоторый терм. Вариант предложения *C* называется такой экземпляр предложения *C*, где вместо каждой переменной подставлена другая переменная.
- **Процедурное значение как «необходимое зло»**
  Процедурное значение определяет, каким образом Prolog отвечает на вопросы. Получить ответ на вопрос означает попытаться достичь целей, заданных в списке. Их можно достичь, если переменные, которые встречаются в целях, могут быть конкретизированы таким образом, что цели логически следуют из программы. Поэтому процедурное значение Prolog определяет процедуру выполнения целей в списке применительно к заданной программе. «Выполнить цель» - означает попытаться ее достичь.
- **Предикат «!»**
  В процессе вычисления цели пролог проводит перебор вариантов, в соответствии с установленным порядком. Цели выполняются последовательно, одна за другой, а в случае неудачи происходит откат к предыдущей цели.
  Однако для повышения эффективности выполнения программы, часто требуется вмешаться в перебор, ограничить его, исключить некоторые цели.
  Для этой цели введена конструкция отсечения (cut), обозначаемая как «!».
- **Опасности «процедурного» программирования в среде Prolog**

5. Понятие отрицания цели

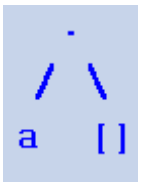
Встроенный предикат `not` имеет один аргумент. Этим аргументом является отношение, значение истинности которого (после обработки этого отношения) заменяется на противоположное.

- **Способ доказательства отрицания цели**
- **Предикаты true, fail**

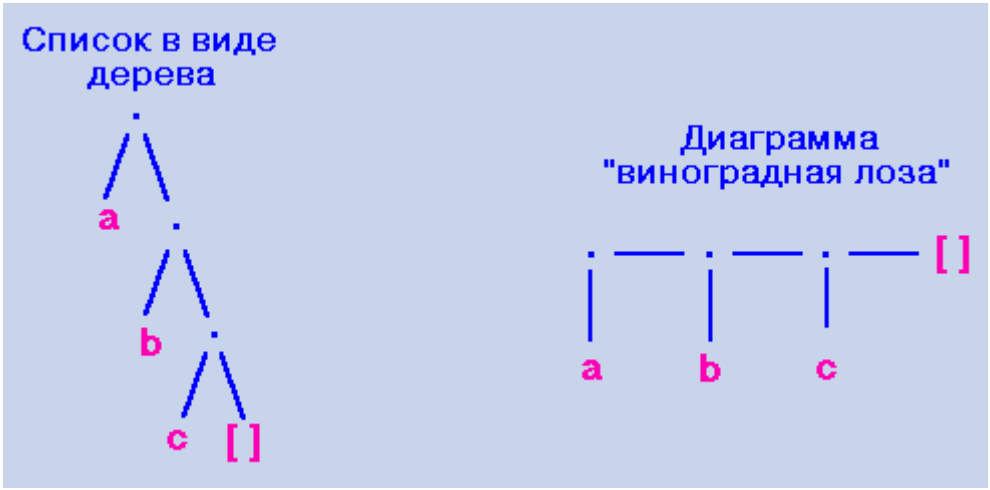
6. Работа со списками

Списки - такая же важная структура данных в Прологе. Список: [a,b,c,d], [1,2,[3]]. Элементами списка могут быть любые термы! Пустой список не null, а [].

- **Представление списка**
  .(a,[]) соответствует [a] или



  Сответственно список [a,b,c] представляется как структура .(a ,.(b,.(c,[]) или в виде дерева, или диаграммой "виноградная лоза".



- **Обращение к каждому элементу списка**
  Главной операцией при работе со списками является расщепление списка на голову и хвост. Первый элемент списка - голова. Список без головы - хвост. В Прологе имеется специальная форма представления списка: [Head|Tail] или [H|T].
  Пример: [a|[]] = [a].
  При конкретизации формы списком H сопоставляется с головой списка, а T - с хвостом.
  Пример:
  p([a,b,c]).
  ?-p([X|Y])
  X=a,
  Y=[b,c]
- **Стандартные операции со списками**
  member(X,L) - если X принадлежит L, то истина и ложь в противном случае.
  member(X, [X|T]).
  member(X, [H|T]) :-member(X, T).
  Пример:
  member(a, [a,b,c]) - yes
  member(X, [a,b,c]) - X = a, X = b, X = c

append(L1,L2,L3) - используется для соединения двух списков. L1 и L2 - списки, а L3 - их соединение.
Пример:
append([a,b],[c],X). - X = [a, b, c]

reverse(L1,L2) - обращает список.
Пример: reverse([a,b,c],X). X = [c,b,a]

lenght[L1, N] - длина списка.

Добавление: записать как факт add(X, L, [X|L]).

Удаление:

```
del(X, [X | Tail], Tail),
del(X, [Y | Tail], [Y | Tail]) :-
  del(X, Tail, Taill).
```

7. Виды рекурсии

- **Прямая рекурсия**
- **Обратная рекурсия**
- **Особенности языка Prolog при работе с рекурентными изменением данных**

8. Ввод и вывод

- **Понятие потока**
- **Простые операции вводы и вывода**
- **Операции загрузки файла с предикатами (база знаний)**

9. Дополнительные предикаты и ключевые слова

- **Name**
  Name(слово, список) — разбивает слово на буквы и присваивает им числа, записывая их в список.
- **bagof, setof**
  При помощи механизма автоматического перебора можно получить один за другим все объекты, удовлетворяющие некоторой цели. Всякий раз, как порождается новое решение, предыдущее пропадает и становится с этого момента недоступным.
  Однако есть возможность собрать все решения в список. Встроенные предикаты *bagof* (набор) и *setof* (множество) обеспечивают такую возможность. Вместо них иногда используют *findall* (найти все).
- **assert, retract**
  Используются для работы с БД (запись, удаление)
- **Is**
  Работа с числами и присваивание переменным. *X=5, Y is X*2*