**Напоминаю:**

Факты представляют собой составные термы, с помощью которых фиксируется знание о наличии истинностного отношения между объектами предметной области — аргументами терма. Правило является обобщенной формулировкой **знания** о возможном истинностном отношении между объектами предметной области (обозначенными аргументами терма заголовка). Условие истинности этого отношения является телом правила (возможно – конъюнкция условий). Система активизируется при задании вопроса (цели доказательства), т.е. делает попытку доказать его истинность. Для доказательства система должна использовать некоторое (свое) знание из БЗ – система должна его **подобрать**, для этого и запускается алгоритм унификации вопроса и очередного заголовка. Но формулировки знаний и формулировка вопроса могут содержать переменные, которые не имеют значений. Чем больше переменных используется в формулировке знания, тем в более общей форме «звучит» знание и, в результате БЗ более компактна, а работа с ней более эффективна. Но, чтобы убедиться, что знание **подобрано** удачно, система должна быть уверена, что может быть построен наиболее общий пример вопроса и заголовка, т.е. суметь определить значения переменных – строит подстановку. Дальнейшее доказательство (доказательство истинности тела выбранного правила) должно продолжаться при найденных, при подборе знания, значениях переменных. В процессе работы система меняет состояние резольвенты (за два шага). Если для доказательства цели и последующей редукции из резольвенты было успешно выбрано правило из БЗ, то вместо цели, в резольвенту (в виде стека) помещается тело выбранного правила так, что левое правило оказывается в вершине стека. И к полученной резольвенте применяется подстановка.

Система работает **формально**, используя некоторые области памяти и информацию, хранящуюся в них. Система использует резольвенту (см. лекцию), алгоритм унификации и механизм отката. В каждый момент система должна помнить что (какие термы) ей еще предстоит доказать – цели доказательства. Первое состояние резольвенты – это вопрос. Для доказательства очередной цели используется алгоритм унификации, позволяющий проверить: подходит ли для доказательства вопроса конкретное знание (заголовок). Если «Да», то система должна проверить выполнимость тела. Т.е. вместо вопроса, фактически, надо доказывать тело – это очередные цели (они замещают вопрос). Если же тело не удастся доказать, то возникнет тупиковая ситуация, и система выполнит откат (куда?), и должна вновь пытаться доказать то, что не было доказано. Т.е. предыдущее состояние резольвенты восстанавливается. Для обозначения точки возврата, система помечает выбранное ранее правило, что позволяет ей перейти к данной метке, и использовать другое, следующее из ниже лежащих знаний в БЗ (не делать повторов). И т.к. БЗ просматривается сверху вниз, то если будет выбрано другое знание для доказательства исходной цели, то метка будет перенесена ниже по тексту БЗ. Т.о. метки ползут вниз. Во время работы системы, некоторое правило может быть использовано несколько раз, тогда оно будет помечено несколькими метками. Первой смещается последняя поставленная метка. Если все метки достигли конца БЗ, то использованы все знания БЗ – и следует останов работы системы.

Правила могут содержать системные предикаты, которые часто включают в тело: нуль-местные предикаты: ! и fail.

Системный предикат отсечения **!** (cat) включается в конъюнкцию целей так же, как и другие предикаты и отсекает (на основании особых правил его обработки), в определенном случае, бесперспективные пути доказательства.

Его обработка определяется следующими правилами.

Пусть есть фрагмент программы:

**p1 :– a, b, !, c, d.**

**p2 :– …** **.** где р1, р2 – заголовки правил, а, b, !, c, d – термы,

(как обычно, утверждения БЗ обрабатываются сверху вниз и **слева направо**)

1. Если отсечение еще не пройдено, то действует обычное правило отката;
2. При прямом ходе доказательства (прямой трассировке), отсечение истинно и не оказывает на вычисления никакого влияния;
3. Для целей, расположенных правее отсечения возможно пересогласование целей, если оно не требует перехода через отсечение;
4. **При обратной трассировке** (откате – попытке отменить в примере **с**) **отсечение** не только никогда не согласуется (его невозможно пересечь), но и запрещает использовать для согласования правила, лежащие ниже, т.е. **завершает использование** **всей** процедуры.

Существует еще один системный предикат – **fail,** который принудительно, при прямом ходе включает механизм отката.

**Рекурсия в Prolog**

Напомним: Рекурсия – это ссылка при описании объекта на описываемый объект. В логическом программировании мы описываем знание. Т.е. рекурсия в Prolog это: **при описании знания – ссылка на это же знание**. Как рассматривали ранее, для организации рекурсии необходимо обеспечить переход на новый шаг рекурсии (использование того же знания в очередной раз) с измененными аргументами. Т.к. выбор знания выполняется системой путем выбора заголовка правила, после чего обязательно проверяется истинность тела, то повторно использовать правило можно упомянув в теле терм, сопоставимый с заголовком того же правила (процедуры). Как обычно, для корректной работы рекурсии необходимо осуществить выход из рекурсии, что часто оформляют в виде отдельного предложения той же процедуры. Эффективный способ организации рекурсии – хвостовая рекурсия (ссылка на знание – эту же процедуру, последняя в теле правила). Как выход из рекурсии, так и переход на новый шаг может осуществляться несколькими способами. Напоминаю, что преобразование не хвостовой рекурсии в хвостовую можно обеспечить использованием дополнительных параметров. При этом, учитывая, что система использует механизм отката, требуется обеспечить, чтобы после выхода из рекурсии система не пробовала использовать вновь ниже лежащие правила (возможно используя отсечение).

Пример оформления хвостовой рекурсии:

p(ar1, …, argN) :- <выход из рекурсии>.

…

p :- t1, …,tk, p(arg11, …, argN1).