Декларативная и процедурная семантика логической программы

В естественном языке каждое слово имеет некоторое значение (смысл). В логической программе смысл каждого имени (идентификатора) определяется набором утверждений, которые описывают его свойства и связи.

В императивных (процедурных) языках программирования **под семантикой** некоторой конструкции языка понимается поведение вычислительной системы при обработке этой конструкции. **Семантика** формулы в логике предикатов связана с приписыванием этой формуле истинностного значения.

Prolog одновременно является: и логическим языком, и языком программирования, реализуемым на императивно работающей технике. Поэтому к нему применимы обе трактовки понятия семантики. В **декларативной** модели – формулировки программы рассматриваются как знания – отношения, взаимосвязи между объектами, сформулированные в виде правил. Для этой модели порядок следования предложений в программе и условий в правиле не важен. Действительно, важна общая совокупность знаний, а не их порядок. **Процедурная** модель рассматривает правила как последовательность шагов, которые необходимо успешно выполнить для того, чтобы соблюдалось отношение, приведенное в заголовке правила. В процедурной трактовке логической программы, т.е. при ее выполнении становится **важен порядок**, в котором записаны предложения в процедурах и условия в предложениях (в теле правил). В идеале стиль программирования должен быть полностью декларативным. Реально порядок следования предложений в программе крайне существенно сказывается на **эффективности** работы системы при выполнении программы. Порядок до такой степени существен, что некоторые утверждения, верные в декларативном смысле, с процедурной точки зрения, могут оказаться неработающей программой!

Как было ранее сказано, Prolog одновременно является: и логическим языком, и языком программирования, реализуемым на императивно работающей технике. Поэтому в «чистом» логическом программировании порядок следования предложений в программе может быть произволен, но, в реальной программе на Prolog оказывается важен порядок процедур, предложений внутри процедуры, а также порядок условий в теле предложений, т.е. вообще порядок любых составных частей текста программы. От порядка предложений и условий в теле правил зависит порядок подбора знаний и порядок, в котором будут находиться ответы на вопросы. Порядок условий влияет на количество проверок, выполняемых программой при решении, объемы памяти, используемой системой Prolog во время работы и др.

Процедурная семантика программы (базы знаний) на Prolog (с декларативной точки зрения нет разницы в каком порядке знания попали в базу, или в Вашу голову– Вы или система обязаны дать ответ на вопрос) состоит в том, что совокупность знаний должна быть использована техникой в некотором порядке. А в каком? А Вы знаете, в каком порядке в Вашей голове анализируются знания? Вы об этом не думаете. Какая разница, в каком – важен вывод. Но, формально, системе надо установить порядок, в котором она должна работать. И такой порядок установлен – один для всех программ, т.е. для системы (хотя это далеко не единственно возможный порядок).

Напомним, что унификация двух термов – это основной шаг доказательства, назначение которого – подобрать нужное в данный момент правило. В процессе работы система выполняет большое число унификаций. **Унификация –** операция, которая позволяет формализовать процесс логического вывода (наряду с правилом резолюции). С практической точки зрения – это отдельный вычислительный шаг работы системы, который может завершиться успехом или тупиковой ситуацией (неудачей). Процесс унификации запускается автоматически, если есть что доказывать, то надо запускать алгоритм унификации. Пользователь имеет право запустить этот процкссс вручную, с помощью утверждения T1 = T2, включенного в текст программы. Остаются вопросы: для

каких термов запускать алгоритм унификации, и что делать дальше в том или другом случае, и как понять это на формальном уровне. Принят следующий порядок.

Общая схема согласования целевых утверждений

Инициация работы системы выполняется заданием вопроса – цели доказательства. Общая схема доказательства вопроса подчиняется следующему принятому жесткому порядку.

Пусть есть некоторая **программа P** и **вопрос G**. Решение задачи с помощью логической программы P начинается с задания вопроса G и завершается получением одного из двух результатов:

* успех согласования программы (базы знаний) и вопроса; в качестве побочного эффекта формируется подстановка, которая содержит значения переменных, при которых вопрос является примером программы (примеров может быть несколько);

* неудача – тупиковая ситуация (вывод делается исходя только из знаний заданной БЗ).

Порядок работы:

Вычисления с помощью конечной логической программы представляют собой пошаговое преобразование исходного **вопроса**. На каждом шаге имеется некоторая совокупность целей - утверждений, истинность (выводимость) которых надо доказать. Эта совокупность называется **резольвентой -** её состояние меняется в процессе доказательства (Для хранения резольвенты система использует стек). Успешное завершение работы программы достигается тогда, когда резольвента пуста. Преобразования резольвенты выполняются с помощью **редукции**.

Редукцией цели G с помощью программы P называется замена цели G телом того правила из P, заголовок которого унифицируется с целью (в заголовке правила зафиксировано знания). Такие правила будем называть сопоставимыми с целью, и система подбирает нужные с помощью алгоритма унификации.

Новая резольвента образуется в два этапа

1. в текущей резольвенте выбирается одна из подцелей (по стековому принципу - верхняя) и для неё выполняется редукция - замена подцели на тело найденного (подобранного, если удалось) правила (а как подбирается правило?),
2. затем, к полученной конъюнкции целей применяется подстановка, полученная как наибольший общий унификатор цели (выбранной) и заголовка сопоставленного с ней правила.

Если для редукции цели из резольвенты был выбран факт из БЗ, то *новая* резольвента будет содержать в конъюнкции на одну цель меньше, т.к. факт – частный случай правила с пустым телом. И, если задан простой вопрос (на первом шаге он попадает в резольвенту) и подобран для редукции факт, то произойдет немедленное его согласование. А если для простого вопроса подобрано правило, то число целей в резольвенте не уменьшится, т.к. цель будет заменена телом подобранного правила.

Механизм отката (backtracking) и дерево поиска решений

При использовании принципа не детерминизма при поиске решения, возможны тупиковые ситуации, для разрешения которых используется механизм отката, кроме этого мы хотим получить все возможные ответы, значит, получив один ответ, надо начать заново, чтобы получить другой ответ.

Для ответа на поставленный вопрос система должна подобрать нужное знание в БЗ, каждое из которых, зафиксировано в заголовке правила. И таких знаний может быть несколько. Причем, неудача при использовании одного такого правила, вовсе не означает неудачу при использовании другого. Поэтому предусмотрена возможность отказа от

сделанного выбора – **backtracking.** Т.е. недетерминизм обеспечивается пошаговым алгоритмом с возвратами – перебором методом «проб и ошибок».

На каждом шаге работы системы – поиске способа (нового) доказательства подцели, система начинает поиск знания с начала базы. И на каждом таком шаге может сложиться одна из трех ситуаций:

1. решение найдено полностью и окончательно, и алгоритм завершен,

2. имеется некоторое число дальнейших альтернативных возможностей, но какая приведет к решению – неизвестно, а система должна выбрать следующее действие формально, т.е. автоматически,

3. решение не найдено, и из данного состояния невозможен переход в новое состояние. В этом случае автоматически включается бэктрэкниг («обратная трассировка»). Происходит возврат к моменту (состоянию), где еще можно сделать другой альтернативный выбор, т.е. к предыдущему состоянию резольвенты и попытке ее преобразовать – пере- согласовать. Альтернативный выбор заключается в использовании другого знания для доказательства цели, а, чтобы не было повторов, система каждый раз помечает выбранное знание, т.е. ранее выбранное правило было уже отмечено. Т.о., при возврате отменяется последняя уже выполненная редукция (восстанавливается предыдущее состояние резольвенты) и система выполняет ре- конкретизацию переменных, которые были конкретизированы на предыдущем шаге. Поэтому, система не должна забывать: какое состояние резольвенты и переменных было ранее. Для этого система использует дополнительную память (а как там организована информация, как бы Вы организовали?), и, хотелось бы, чтобы этой памяти был минимум, а сколько будет возвратов, а, значит, сколько понадобится памяти – зависит от порядка утверждений в БЗ.

Для того чтобы представить себе наглядно порядок работы системы, строят дерево поиска решения. **Дерево поиска решения** – формализм для исследования всех возможных путей вычисления – схема, которую мы можем изобразить для представления себе порядка выбора знаний и последующих действий системы.

Корень дерева - вопрос G. *Вершины дерева* образуют резольвенты, которые в общем случае являются конъюнктивными. Для каждого утверждения программы (правила или факта), заголовок которого унифицируется с выделенной подцелью в резольвенте, имеется ребро. *На ребрах дерева записываются подстановки*, которые формируются в результате унификации выделенной подцели и заголовка сопоставленного ей правила. *Лист* дерева называется успешной вершиной, если резольвента ему соответствующая – пуста и безуспешной вершиной, если резольвента не пуста и нет утверждений в базе знаний, которые удается сопоставить с выделенной в этой вершине подцелью.

Система хранит резольвенту в виде стека, содержащего цели, которые надо будет доказать. Говорят, что при обработке резольвенты, стек растёт влево – т.к. в вершине стека находится левое условие тела выбранного на предыдущем шаге правила. Если резольвента не пуста – запускается алгоритм унификации, а если – пуста, это значит, что получен один, однократный ответ «Да» на поставленный вопрос, после чего включается механизм отката, в попытке найти другое решение с помощью другого знания. При этом, БЗ просматривается сверху вниз.

Управление порядком работы системы

В императивных языках программирования, программист полностью контролирует последовательность вычислений и управляет использованием аппаратных ресурсов. В логических программах система выполняет полный перебор всех возможных вариантов

решения, а возможности управления вычислениями – минимальны. У программиста есть право только изменить порядок следования правил в процедурах и порядок следования условий в теле правила. И этим надо максимально пользоваться.

Но существуют постановки задач, в которых есть бесперспективные пути поиска решений, в чем программист совершенно уверен. Для исключения соответствующих действий из рассмотрения, в Prolog включены два системных предиката: предикат отсечения и предикат fail.

Предикат отсечения **!** (cat) включается в конъюнкцию целей так же, как и другие предикаты и отсекает в определенном случае бесперспективные пути доказательства.

Предикат fail принудительно включает механизм отката.

В современных версиях языка существуют и другие системные предикаты, позволяющие создавать интерфейс программы, но они не влияют на логику работы программы, которую мы изучаем.