УДК 519.687

В. А. ГРЕБИННИК

ИСПОЛЬЗОВАНИЕ ПРОЦЕДУРНЫХ ЗНАНИЙ В ЛОГИЧЕСКОЙ МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ. УНИФИКАЦИЯ ФУНКЦИОНАЛОВ

Логическая модель представления знаний, основанная на логике предикатов первого порядка, является одной из самых старых и самой математически строгой среди всех известных. В ее основе лежит формальная система, задаваемая четверкой вида: $M = <T, P, A, B>$ [2, 4]. Множество T является множеством базовых элементов – алфавитом. Множество P – множество синтаксических правил – правил построения правильных формул (ППФ). С их помощью из элементов T образуют синтаксически правильные совокупности. В множестве синтаксически правильных совокупностей выделяют некоторое подмножество A. Элементы A называют аксиомами. Множество B является множество правил вывода. Применяя их к элементам A можно получать новые синтаксически правильные совокупности, к которым снова можно применять правила из множества B. Так формируется множество выводимых в данной формальной системе совокупностей. Это показывает, что именно правило вывода является наиболее сложной составляющей формальной системы. Для знаний, входящих в базу знаний, можно считать, что множество A образуют все информационные единицы, которые введены в базу знаний извне, а с помощью правил вывода из них выводятся новые производные знания. Другими словами формальная система представляет собой генератор порождения новых знаний, образующих множество выводимых в данной системе знаний. Средства данной модели являются наиболее подходящим инструментом при описании предметной области, в которой будет происходить работа проектируемой экспертной системы. Предикатная форма описания отношений между сущностями предметной области отлично справляется с задачей формализации данного типа знаний. Механизм вывода в данной модели, основанный на методе резолюций, позволяет получать заключения, подтвержденные строгим математическим доказательством. На основании логической модели представления знаний построен язык программирования Пролог [3].

Однако при всех своих положительных качествах логическая модель представления знаний имеет один существенный недостаток, связанный с отсутствием необходимых средств для представления процедурных знаний в рамках данной модели. Как известно знания делятся на процедурные и декларативные. В логической модели набор предикатов является декларативными знаниями, а процедурные в виде механизма логического вывода – методе резолюций. Механизм логического вывода в конкретной модели представления знаний изменен быть не может. Поэтому логическая модель, хорошо описывая знания декларативные, является неэффективной для решения задачи представления процедурных знаний. В языке Пролог отсутствие необходимости программировать конкретную последовательность действий для решения задачи провозглашалось как основное преимущество перед процедурными языками. Однако, полная замена процедурного программирования декларативным определениями отношений между сущностями не всегда выгодна с точки зрения эффективности решаемой задачи и приводит к большим вычислительным затратам. К примеру, рассмотрим тривиальные арифметические операции, которые являются "родными" для компьютера. С точки зрения процедурного программирования нет ничего проще, чем описать функцию возведения в целую степень в виде подпрограммы.

```
S:=1;
for I:=1 to n do  S:=S*k; (1)
```

Рассмотрим как ту же самую задачу приходится решать, используя декларативные средства

Пусть Stepen (*X, Y, Z*) – отношение степени, где *X* – основание, *Y* – степень, *Z* – результат возведения *X* в степень *Y*.

Тогда

Stepen(X, 0, 1) – факт
Stepen(X, Y, Z) : Stepen(X, Y1, T), Z = T * X (2)

А этио примере приходится прибегать к использованию рекурсивного определения понятия степень. Очевидна неэффективность применения Пролога для таких задач с точки зрения эффективности выполнения программы. Рекурсивное решение данной задачи отнимет намного большее время, и объем памяти, а для больших чисел объем затрачиваемой стековой памяти при рекурсии может оказаться недопустимо большим. Гораздо естественней выглядела бы возможность представления решения задачи вычисления степени в виде функционала, также, как в Прологе реализованы операции сложения, умножения, деления и вычитания. Но в Прологе отсутствуют средства для описания функций пользователя в виде процедурных подпрограмм [3].

Рассматривая все сказанное выше можно сделать вывод о том, что логическая модель и Пролог, как практическая реализация данной модели, нуждаются в дополнении простыми средствами представления процедурных знаний. Необходимо встроить данные средства, оставляя всю математическую строгость получаемых логических выводов в рамки формальной системы. Решение данной проблемы предлагается ниже.

Наиболее простым средством для представления процедурных знаний в логической модели является расширение применения функторов и их смысловой нагрузки. Первым простейшим шагом может быть введение в Пролог возможность описывать пользовательские функции на каком-нибудь процедурном языке программирования и использовать их в предикатах наравне со стандартными арифметическими функциями. Применительно к нашему примеру последовательность (1) представляется в виде функтора *f(X, Y)* и используется далее в предикатах, как терм для вычисления степени. Однако даже на применение стандартных функторов Пролог накладывает существенное ограничение на их использование. Напомним, что в процессе вывода при использовании стандартных функционалов необходимо, чтобы в момент вычисления значения функционала все его аргументы были известны, т. е. программа рассматривает его как константу, получающуюся после вычисления функционалов [3]. Это необходимо для правильной работы механизма унификации. Рассмотрим подробнее правила унификации предикатов, используемые Прологом во время логического вывода.

Унификация термов является одной из главных процедур при выполнении Пролог-программы в ходе которой переменные получают значения из Эрбрановского универсума [1, 2], после чего выполняется операция резолюции и появляется новый дизъюнкт. Если термы не унифицируемы, то процесс терпит неудачу. Если же термы унифицируемы, тогда процесс находит подстановку переменных, делающих эти термы тождественными, и завершается удачей. Рассмотрим правила унификации термов в Прологе.

1. Если *S* и *T* константы , то *S* и *T* сопоставимы (унифицируемы) только в том случае, если они являются одним и тем же объектом.

2. Если *S* переменная, а *T* произвольный объект, то ни сопоставимы и *S* приписывается значение *T*. Наоборот, если *T* переменная, а *S* произвольный объект, то *T* приписывается значение *S*. Говорят, что *T* конкретизируется значением *S*.

3. Если *S* и *T* – структуры, то они сопоставимы, если: а) *S* и *T* имеют одинаковый главный функтор, и б) все их соответствующие компоненты сопоставимы [1].

Результирующая конкретизация определяется сопоставлением компонент.

Исходя из пункта (3) термы $f+(3,5)$ и $f-(18,10)$ были бы неунифицируемы, если бы Пролог предварительно не вычислял эти значения, пользуясь семантикой функторов "+" и "–" и не заменял бы эти термы результатом вычисления – константой. В исчислении предикатов при удалении свободных переменных возникает необходимость применения функции Сколема [1], которая носит декларативный характер, так как просто устанавливают факт существования некоторой зависимости. Поэтому возникала необходимость при выводе пользоваться правилами унификации функторов, не используя их семантику.

Однако правило (3) существенно ограничивает возможности унификации функторов, семантика которых нам известна. Пусть $f(x)$ и $g(x)$ функции которые принимают значение, а при $x = b$. Тогда замена $x|b, f(x)|a, g(x)|a$ позволяет унифицировать термы $f(x)$, $g(x)$. Пролог способен выполнить такие действия только в том случае, если к началу процесса унификации переменная x уже приобрела значение b в результате предыдущих вычислений и семантика $f(x)$, $g(x)$ ему известна, т. е. они являются стандартными арифметическими функциями. Как было сказано выше, второе ограничение снимается, если разрешить описывать семантику функторов на процедурном языке и использовать в дальнейшем как стандартные функции, т. е. при необходимости унификации пользовательского функтора, аргументы которого известны, необходимо вычислить его значение и заменить на полученную в результате вычисления константу. Дальнейший процесс унификации выполняют по приведенным выше стандартным правилам. Если же аргумент к моменту вычисления не известен, то и в таком случае можно попытаться произвести унификацию пользуясь измененным правилом (3). В таком виде как есть оно должно использоваться только для тех функторов, семантика которых нам неизвестна (к примеру функции Сколема). Для функций, семантика которых нам известна предлагается другой алгоритм. Пусть $y = f(x)$ – функция, тогда при заданном x мы можем вычислить y, $g(x)$ – обратная ей функция, т. е. $g(f(x)) = x$. Тогда правило унификации функторов будет иметь следующий вид:

а) Если S константа, а T функция вида $f(x)$, то S и T сопоставимы, если существует такое значение $x = a = g(S)$ при котором $f(a) = S$. В результате унификации T меняется на S, а x принимает значение a, вычисленное на основании $a = g(S)$.

б) Если S функция $f_1(x)$, а T функция вида $f_2(x)$, то S и T сопоставимы, если существует решение уравнения $f_1(x) = f_2(x)$, т. е. существует $x = a$ при котором $f_1(a) = f_2(a)$.

Если к моменту решения переменная x еще не получила никакого значения, то поиск значения a вести в общем случае неэффективно. Поэтому решение данного вопроса можно перенести на более поздний момент вычислений, когда значение x определится в ходе дальнейших унификаций, считая что данные функторы унифицируемы. Тот факт, что вопрос унификации $f_1(x)$ и $f_2(x)$ не решен, можно заменив вхождение данных функторов условным символом унификации $V_1(f_1(x), f_2(x))$, и как только переменная x получит некоторое значение a, необходимо тут же выполнить проверку $f_1(a) = f_2(a)$. Если данное условие не выполняется, значит функторы не унифицируемы. Точно также мы можем поступить и в случае (a), если вид обратной функции $g(x)$ нам не известен. При этом условный символ унификации примет вид $V_1(S, f(x))$, а проверочное условие $f(x) = S$. Легко заметить, что при необходимости унификации двух символов $V_1(f_1(x), f_2(x))$ условной унификации $V_1(f_1(\mathrm{x}), f_2(\mathrm{x}))$ и $V_2(f_3(y), f_4(y))$ мы получим третий символ условной унификации по правилу $V_3(f_1(x), f_2(x), f_3(y), f_4(y))$ проверочным условием выполнения которого будет $f_1(x) = f_2(x) = f_3(y) = f_4(y)$.

Таким образом, вводя возможность использования в Прологе вычисляемых функционалов мы получаем следующие преимущества:

1. Естественный и простой способ представления процедурных знаний с сохранением всех преимуществ логической модели;

2. Ускорение процедуры логического вывода в следствие отбрасывания заведомо неправильных значений переменных;

3. Расширение области поиска возможных значений при решении задач, в результате расширения возможности унифицирования функционалов.

К недостаткам следует отнести значительно возросшую сложность механизма унификации, что требует значительных усилий при программировании. Однако, вследствие того, что данный механизм программируется лишь один раз при реализации механизма логического вывода, получаемые преимущества с лихвой компенсируют этот недостаток.

Список литературы: 1. *Дюбко. Г.Ф.* Введение в формальные системы. К.: УМК ВО, 1992. 171 с. 2. *Логический подход к искусственному интелекту: от классической логики к логическому программированию* / А. Тейз, П. Гриббомон, Ж. Луи и др. М.: Мир, 1990. 432 с. 3. *Ин Ц., Соломон Д.* Использование ТурбоПролога. М.: Мир, 1993.608 с. 4. *Логическое программирование. Курс лекций* / Морозов М.Н. 1999. URL: http://www. marstu.mari.su/mmlab/home/prolog/study_1.html

Поступила в редколлегию 19.07.2000