Лекция 8. Логическое программирование.

Зафиксируем язык первого порядка L **с равенством** не более чем счетной сигнатуры $\Omega = <P_1,P_2,...; f_1,f_2,...; C_1,C_2,... >$, $C = \{C_{k\in K}\}$, $K \neq \varnothing$.

X – множество **переменных** x, y, z … ;

T - множество **термов** t, s, … - $C \subset T$, $X \subset T$, $f(t_1,\ldots,t_k) \in T$, $t_1,\ldots,t_k \in T$;

U – множество всех **основных термов** не содержащих свободных переменных;

At – множество всех **атомарных формул** A, B, C, … вида

$f(x_1,\ldots,x_k) = g(y_1,\ldots,y_l)$ или вида $P(x_1,\ldots,x_m)$;

правилом называется выражение $A \leftarrow B_1,\ldots,B_k$, $k \geq 0$ сигнатуры Ω, где $A, A_1,\ldots,A_k \in At$,

A – заголовок правила, а $B_1,\ldots,B_k$ – тело правила;

Целью (**запросом**) называется правило $\leftarrow A_1,\ldots,A_k$

Правило $A \leftarrow$, где $k = 0$ называется **фактом**.

Логическая программа Pr есть конечная совокупность правил.

Подстановкой называется отображение $\theta: X \rightarrow T$ в котором переменная X не входит в T.

Подстановка $\theta(x) = x$ называется **тождественной**.

Подстановки естественным образом распространяются на произвольные выражения.

Для терма $t = f(t_1,...,t_n)$ и атома $A = P(t_1,...,t_n)$ их подстановками являются соответственно $t\theta = f(t_1\theta,..., t_n\theta)$, $A\theta = P(t_1\theta,...,t_n\theta)$. Они называются **примерами** терма или атома. Если θ – **перестановка переменных** X, то терм $t\theta$ и атом $A\theta$ являются **вариантами** соответствующих термов и атомов.

Подстановка θ является **унификатором** выражений E_1, E_2, если $\theta E_1 = \theta E_2$. Унификатор θ выражений E_1, E_2 называется **наиболее общим унификатором**, если для любого другого унификатора θ' выражений E_1,E_2 найдется подстановка θ'' такая что $\theta' = \theta''\theta$.

Алгоритм унификации:

Вход: Два терма t_1 и t_2, которые надо унифицировать.

Результат: Подстановка θ - наиболее общий унификатор термов t_1 и t_2.

Алгоритм: начальное значение подстановки θ - пусто, стек содержит равенство $t_1 = t_2$

```
failure:= false
Пока стек не пуст и не failure выполнить:
   {считать равенство X=Y из стека
        если X – переменная, не входящая в Y, то
             {заменить все вхождения X в стеке и в θ на Y, добавить X=Y к θ}
        если Y – переменная, не входящая в X, то
             {заменить все вхождения Y в стеке и в θ на X, добавить Y=X к θ}
        если X и Y – одинаковые константы или переменные, то продолжить
        если X = f(X1,…,Xn) и Y = f(Y1,…,Yn), то
             {поместить Xi = Yi для всех i = 1,…,n в стек}
    в остальных случаях failure:= true
    }
если failure, то результат = отказ;
выдать результат = θ.
```

Вычисление логической программы. Фиксируется правило R вычисления, определяющее для каждого запроса $\leftarrow A_1,...A_i\ldots,A_k$ выделенный атом A_i .

Тогда для запроса $\leftarrow A_1,..., A_{i-1}, A_i, A_{i+1} \ldots,A_k$ и варианта некоторого правила программы Pr $A_0 \leftarrow B_1,...,B_n$ в котором все переменный отличны от переменных запроса **элементарный шаг вычисления (вывода)** состоит в нахождении наиболее общего унификатора θ для атомов A_i и A_0 (если таковой найдется) и переходом к новому запросу

$\leftarrow \theta A_1,..., \theta A_{i-1}, \theta B_1,..., \theta B_n , \theta A_{i+1}\ldots,\theta A_k$

Пространством вычислений программы Pr для заданного правила вычисления R называется множество всех возможных запросов с заданным на нем отношением выводимости.

Вычислением запроса Q называется путь $Q = Q_1, Q_2, Q_3, \ldots$ в пространстве вычислений начинающийся с Q и Q_{i+1} выводим из Q_i.

Вычисление может закончиться на запросе Q_n в двух случаях:

1. Q_n – пустой запрос. Тогда вычисление называется **успешным** и **результатом вычисления** является суперпозиция подстановок $\theta = \theta_{n-1}\theta_{n-2}\ldots\theta_1$, где θ_i – подстановка с помощью которой из запроса Q_i выводим запрос Q_{i+1}. Если $\langle x_1,\ldots,x_k\rangle$ - набор переменных запроса, то набор $\langle t_1= \theta x_1,\ldots, t_k= \theta x_k\rangle$ называется **ответом**, выдаваемым в результате вычислений. Если в запросе переменных нет, то результатом успешного вычисления является ответ «Да».

2. Q_n – не пусто и из него не выводим ни один запрос. В этом случае вычисление является тупиковым.

Каждому запросу соответствует часть пространства вычислений, содержащая все пути, ведущие из вершины этого запроса. Эти пути образуют **дерево вычислений запроса**.

Процесс вычисления (вывода) запроса $\leftarrow$ G можно представить в виде дерева рис. 1. В нём запрос G унифицируется с заключениями правил L1 и L2 некоторой подстановкой θ_1. Если среди правил программы Pr есть правила L3, L4, L5, которые унифицируются с некоторыми атомами A_i или A_j, то посылки этих правил $B^1{}_1\&...\&B^1{}_{n1}$ или $B^2{}_1\&...\&B^2{}_{n2}$ или $B^3{}_1\&...\&B^3{}_{n3}$ подставляются вместо соответствующих атомов A_i или A_j. Если какие-то правила L3, L4 или L5 являются фактами вида $A_i \leftarrow$, то соответствующий атом после унификации удаляется из посылки правила L1. Вывод (вычисление) заканчивается, когда найдена такая ветвь дерева вывода, которая содержит правило ($G \Leftarrow$).

Ответом программы Pr **на запрос** Q является ответ любого успешного вывода запроса Q.

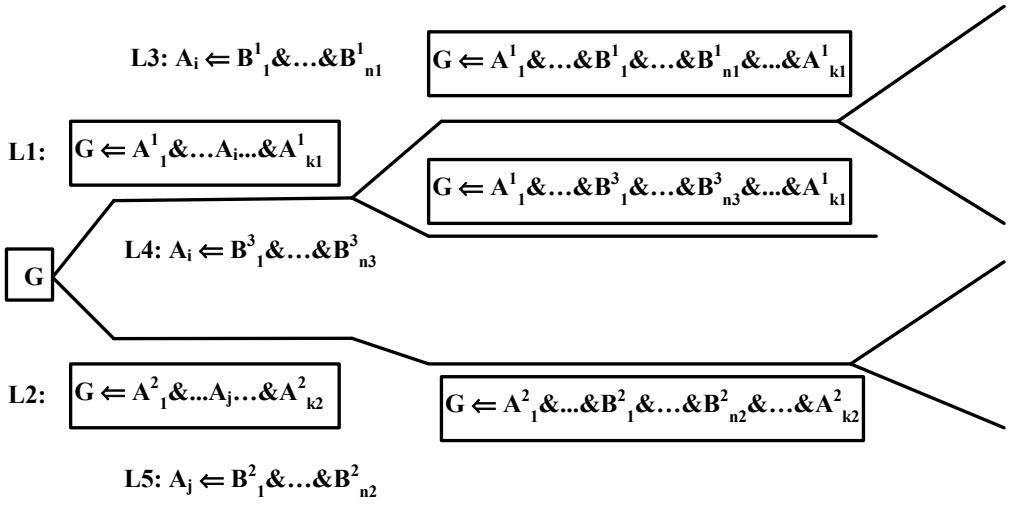

Рис. 1

Предсказания, решения, ответы на запрос в логическом программировании

Предсказания, решения, ответы на запрос в логическом программировании формулируется как запрос $\leftarrow G$ или $\leftarrow A_1,...,A_k$ ($G = A_1,...,A_k$) к логической программе.

В процессе вычисления ответа на запрос $G(x_1,\ldots,x_n)$ вычисляется:

1. вывод $\{L_1,\ldots, L_m, C_1,\ldots,C_n\} \vdash \exists x_1,\ldots,x_n G$;
2. ответ – набор термов $t_1,\ldots,t_n$ для которых $\{L_1,\ldots, L_m\ C_1,\ldots,C_n\} \vdash G[x_1/t_1,\ldots,x_n/t_n]$.

Представление реляционных операций.

Пять основных операций определяют реляционную алгебру:
объединение, симметрическая разность, декартово произведение, проекция и выборка.
Покажем, как каждая из них выражается в логической программе.

1. Объединение.
   $r\_ion\_s(X_1,...,X_n) \leftarrow r(X_1,...,X_n)$
   $r\_ion\_s(X_1,...,X_n) \leftarrow s(X_1,...,X_n)$
2. Пересечение
   $r\_meet\_s(X_1,...,X_n) \leftarrow r(X_1,...,X_n), s(X_1,...,X_n)$
3. Симметрическая разность
   $r\_diff\_s(X_1,...,X_n) \leftarrow r(X_1,...,X_n),$ not $s(X_1,...,X_n)$.
   $r\_diff\_s(X_1,...,X_n) \leftarrow s(X_1,...,X_n),$ not $r(X_1,...,X_n)$.
4. Декартово произведение.
   $r\_x\_s(X_1,\ldots,X_{m+n}) \leftarrow r(X_1,\ldots,X_m), s(X_{m+1},\ldots,X_{m+n})$
5. Проекция
   $r13(X_1,X_3) \leftarrow r(X_1,X_2,X_3)$.
6. Выборка
   $r1(X_1,X_2,X_3) \leftarrow r(X_1,X_2,X_3), X_2 > X_3$.
   $r2(X_1,X_2,X_3) \leftarrow r(X_1,X_2,X_3),$ *смит_или_джонс(X_1)*
   смит_или_джонс(смит).
   смит_или_джонс (джонс).

Объяснение и работа с оценками в экспертных системах.
Метаинтерпретаторы.

Программа, которая в случае нехватки информации, запрашивает информацию у пользователя.

$\leftarrow$ solve(Goal)

solve(true).
solve((A,B)) $\leftarrow$ solve(A), solve(B).
solve(A) $\leftarrow$ clause(A,B), solve(B).
solve(A) $\leftarrow$ askable(A), not known(A), ask(A, Answer), respond(Answer, A).
первоначально всегда not known(A) для нефактов.
askable – процедура, коротая в случае безуспешного решения цели интерпретатором может направить ее на рассмотрение пользователю.
ask(A, Answer) $\leftarrow$ write(A?), read(Answer).
respond(yes, A) $\leftarrow$ assert(A) – в программу вводится факт A.
respond(no, A) $\leftarrow$ assert(untrue(A)), fail. (предикат, который никогда не выполняется).
known(A) $\leftarrow$ A; known(A) $\leftarrow$ untrue(A).

Программа, объясняющая как доказывается цель.
← how(Goal)

how(Goal) ← solve(Goal, Proof), interpret(Proof).
solve(true, true).
solve((A,B),(ProofA, ProofB)) ← solve(A, ProofA), solve(B, ProofB).
solve(A, (A ← ProofB)) ← clause(A, B), solve(B, ProofB).

interpret((Proof1, Proof2)) ← interpret(Proof1), interpret(Proof2).
interpret(Proof) ← fact(Proof, Fact), nl, writeln([Fact,'факт в базе данных'])
fact((Fact ← true), Fact).
interpret(Proof) ← rule(Proof, Head, Body, Proof1),
nl, writeln([Head, 'доказывается с помощью правила']).
display_rule(rule(Head, Body)),
interpret(Proof1).

rule((Goal ← Proof), Goal, Body, Proof) ← Proof ≠ true, extract_body(Proof, Body).

extract_body((Proof1, Proof2), (Body1, Body2)) ←
extract_body(Proof1, Body1), extract_body(Proof2, Body2).
extract_body((Goal ← Proof), Goal).
display_rule(rule(A, B)) ← write('IF'), write_conjunction(B), writeln(['THEN', A])

Программа вычисления оценок утверждений.
← solve(Goal, Certainty)

solve(true, 1).
solve((A,B), C) ← solve(A, C1), solve(B, C2), minimum(C1,C2,C).
solve(A, C) ← clause_cf(A, B, C1), solve(B, C2), C := C1*C2.
clause_cf(A, B, C1) – предикат присваивающий оценку определенности правилу A ← B.