С.Н. ВАСИЛЬЕВ

ОСНОВАНИЯ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

О Г Л А В Л Е Н И Е

Введение

Автоматизация дедуктивных и других логических построений представляет собой одно из наиболее важных направлений того раздела кибернетики, за которым закрепилось наименование «искусственный интеллект» (ИИ) [1]. Многие другие направления исследований в области ИИ, нацеленные, например, на создание так называемых экспертных систем, «интеллектуальных» баз данных, систем поддержки рассуждений по умолчанию (default reasoning), систем объяснения наблюдений, в том числе в проблематике диагностики и т.д., также стимулировали появление и развитие логического программирования.

Оказалось, что традиционные языки программирования (типа ФОРТРАН, АЛГОЛ, ПЛ-1 и др.) недостаточно эффективны при создании экспериментальных систем ИИ, прежде всего, из-за необходимости исчерпывающего описания на этих языках алгоритма решения всей задачи. Этот недостаток был замечен на фоне результатов по автоматическому доказательству теорем, возбудивших надежду на возможность разработки языков программирования весьма высокого уровня, требующих от пользователя, в основном, описания того, ЧТО нужно сделать (т.е. требующих формулировки задачи: ДАНО …, ТРЕБУЕТСЯ …), а не того, КАК это делается.

В начале 70-х гг. XX в. Р. Ковальским и А. Колмерауром была высказана идея, что классическая логика (а именно язык исчисления предикатов) может быть использована как язык программирования. До этого логика использовалась лишь как язык утверждений (спецификаций) либо как дедуктивный аппарат, например, в вопросно-ответных системах. Явное указание возможности процедурной интерпретации языка логики привело к созданию языков программирования типа ПРОЛОГ (PROgramming in LOGic – программирование в логике), и первый PROLOG-интерпретатор бы создан в 1972 г.

ПРОЛОГ нашел ряд приложений в задачах ИИ, послужив в начале 80-х гг. XX в. [18] одной из причин сверхоптимистических прогнозов в отношении вычислительных систем 5-го поколения, а сегодня, как это часто бывает, «на волне популярности» его характеристики, в том числе и недостатки (см. Заключение), часто неоправданно распространяют на логический подход (к ИИ) в целом. Не следует путать ПРОЛОГ с вообще логическим программированием, а последнее – с логическим подходом к программированию. ПРОЛОГ основывается на логическом программировании примерно как ЛИСП – на λ-исчислениях, а логическое программирование – лишь одно направление в том широком фронте исследований, который называется логическим подходом к программированию.

Новое дыхание логическому подходу в целом придаёт разработка исчисления позитивно-образованных формул [19]. Наконец, в качестве настольной рекомендуется книга [22] – прекрасное учебное пособие, прежде всего, тем, кто, уже имея понятие о программировании (самостоятельно либо по школе), намерен стать профессионалом в области информационных технологий. Книга [22] по-настоящему будит мысль читателя.

Применение механизмов дедукции логического программирования разнообразны; например, оно является базой так называемого абдуктивного программирования, ориентированного на автоматическое программирование гипотез [23 – 28].

Указания рекомендуются студентам III – V курсов при изучении спецкурса «Методы искусственного интеллекта и принятия решений». Более подробное изложение ПРОЛОГа см. также в [20, 21].

§ 1. Логические программы

Определим синтаксис логических программ. Термы и атомы (т.е. атомарные или элементарные формулы) определяются как в языке исчисления предикатов (первого порядка) [2, 3]. Неформально говоря, тер-

мы – это переменные, константы и выражения вида $f(t_1,...,t_n)$, где каждое t_i – терм, а f – n-арный функциональный символ. Атом – это всякое выражение вида $P(t_1,...,t_m)$, где t_i – термы, P – m-арный предикатный символ.

Дизъюнктом называется всякое выражение вида $A_1,...,A_k \leftarrow B_1,...,B_n$, где A_i, B_j – атомы $\left(i=\overline{1,k},\quad j=\overline{1,n},\quad k.n\geq 0\right)$.

Декларативная семантика дизъюнктов (т.е. их содержательный, пока непроцедурный, смысл) совпадает с теоретико-модельной семантикой отвечающих им формул логики.

В языке исчисления предикатов дизъюнкту $A_1,...,A_k \leftarrow B_1,...,B_n$ отвечают, например, формулы вида

$$\forall x_1 ... \forall x_s \quad (B_1 \& ... B_n \rightarrow A_1 \vee ... \vee A_k), \tag{1}$$

или

$$\forall x_1 ... \forall x_s \quad (\neg B_1 \vee ... \neg B_n \vee A_1 \vee ... \vee A_k), \tag{2}$$

или

$$\neg\exists x_1 ... \exists x_s \quad (B_1 \& ... \& B_n \& \neg A_1 \& ... \& \neg A_k), \tag{3}$$

где $x_1,...,x_s$ – все переменные рассматриваемого дизъюнкта. Если, в частности, $k=0$, то имеем так называемую цель (целевой дизъюнкт) $\leftarrow B_1,...,B_n$, которой отвечают, например, формулы

$$\forall x_1 ... \forall x_s \quad (B_1 \& ... \& B_n \rightarrow Л),$$

или

$$\forall x_1 ... \forall x_s \quad (\neg B_1 \vee ... \vee \neg B_n),$$

или

$$\neg\exists x_1 ... \exists x_s \quad (B_1 \& ... \& B_n),$$

где Л – тождественно ложный предикат.

Если $k=1$, дизъюнкт называется **программным** (A – заголовок, $B_1 ... B_n$ – тело дизъюнкта). Если к тому же $n=0$, то программный дизъюнкт называется **фактом**. Если $k=1$ и $n\neq 0$, то программный дизъюнкт называется **правилом** (редуктором).

Хорновские дизъюнкты – это дизъюнкты, у которых $k=0$ или $k=1$. При $k=n=0$ имеем **пустой** дизъюнкт, обозначаемый □ (в логике ему отвечает предикат Л).

Справедливы включения понятий, представленные на рис. 1.

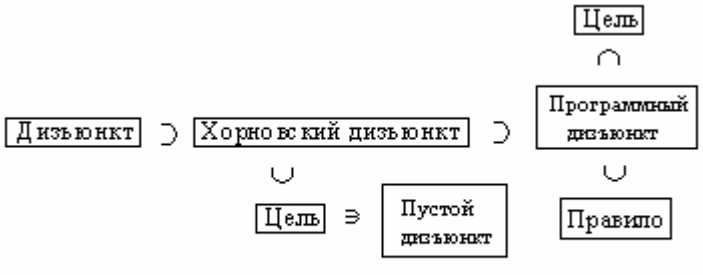

Рис. 1

Логической программой $\mathcal{P}$ (или просто программой, поскольку программы в другом смысле рассматриваться не будут) называется всякое конечное (непустое) множество программных дизъюнктов.

Программа $\mathcal{P}$ может мыслиться как набор нелогических (собственных) аксиом теории первого порядка, а цель – как отрицание формулы

$$\exists x_1 ... \exists x_s \ (B_1 \& ... \& B_n), \tag{4}$$

логическая выводимость которой из аксиом программы нас интересует. Эта выводимость, очевидно, будет иметь место тогда и только тогда, когда из аксиом программы $\mathcal{P}$ и формулы цели, т.е. $\neg(4)$, выводимо противоречие Л. Именно на опровержение (на доказательство «от противного») и ориентировано большинство известных систем автоматического доказательства теорем.

При решении задач нас может интересовать не просто то, что формула (4) является теоремой теории, отвечающей рассматриваемой программе, а чаще всего то, при каких именно значениях переменных $x_1 ... x_s$ справедливо $B_1 \& ... \& B_n$ (назначением баз данных обычно является выдача даже всех таких наборов значений $x_1 ... x_s$).

Пример 1. Пусть дана программа $\mathcal{P}$

$$\text{ОТЕЦ}(\text{ИВАН}, \text{ПЁТР}) \leftarrow , \tag{5}$$

$$\text{ОТЕЦ}(\text{ПЁТР}, \text{СЕМЁН}) \leftarrow , \tag{6}$$

$$\text{ДЕДУШКА}\ (X,Y) \leftarrow \text{ОТЕЦ}\ (X,Z), \text{ОТЕЦ}\ (Z,Y), \tag{7}$$

где ОТЕЦ, ДЕДУШКА – два предикатных символа, ИВАН, ПЕТР – константы, X, Y, Z – переменные. Задание программе цели

$$\leftarrow \text{ДЕДУШКА}\ (X, \text{СЕМЁН}) \tag{8}$$

означает желание получить ответ на вопрос «Кто является дедушкой Семёна?» на основе описания предметной области (отец отца всякого человека является дедушкой этого человека) и реального состояния объектов, функций и отношений, входящих в эту область ,отцом Петра является Иван, а отцом Семёна – Пётр).

Программе $\mathcal{P}$ из примера 1 отвечает теория первого порядка с тремя собственными аксиомами (5), (6) и

$$\forall x \forall y \forall z \ (\text{ОТЕЦ}(x, y) \ \& \ \text{ОТЕЦ}(x, y) \rightarrow \text{ДЕДУШКА}(x, y)). \tag{9}$$

Мы можем установить выводимость в этой теории формулы $\exists x \text{ДЕДУШКА}(x, \text{СЕМЁН})$ путем вывода противоречия в теории, которая отличается от упомянутой лишь дополнительно введённой аксиомой

$$\neg \exists x \text{ДЕДУШКА}(x, \text{СЕМЁН}), \tag{10}$$

отвечающей цели (8).

Упражнения:

1. Какой частный вид примут формулы (1 – 3) для фактов?

2. Постройте в теории первого порядка [2, 3] с собственными аксиомами (5), (6), (9) вывод формулы $\exists x \text{ДЕДУШКА}(x, \text{СЕМЁН})$, а в теории с собственными аксиомами (5), (6), (9), (10) – вывод противоречия.

§ 2. Теоретико-модельная семантика логических программ

Пусть пока S – произвольная формула исчисления предикатов S не обязательно «дизъюнктивная формула, т.е. не обязательно отвечает дизъюнкту [2, 3].

Для S понятия интерпретации, истинностной оценки, выполнимости, общезначимости, невыполнимости (противоречивости) модели понимаются как обычно [2, 3].

В этих терминах, задавая цель G программе $\mathcal{P}$, мы тем самым просим показать, что множество хорновских дизъюнктов $\mathcal{P} \cup \{G\}$, понимаемое как конъюнкция отвечающих этим дизъюнктам «хорновских» формул, невыполнимо, т.е. любая интерпретация множества $\mathcal{P} \cup \{G\}$ не является для него моделью.

Оказывается, существует подкласс интерпретаций (так называемые эрбрановские интерпретации), которыми можно ограничиться при выяснении вопроса о невыполнимости формул S.

Введем некоторые определения.

Литерой называется атом или отрицание атома.

Выражением (в строгом смысле слова) называется терм, литера или дизъюнкт. Выражение, не содержащее переменных, называется **основным** выражением. **Подвыражением** выражения E называется всякое выражение, встречающееся в E (может быть, просто совпадающее с E).

Пусть S – формула. **Эрбрановский универсум** (э-универсум) $\mathcal{U}(S)$ для S – это множество всех основных термов, которые можно построить из констант и функциональных символов, входящих в S (если S не содержит констант, то для образования термов э-универсума вводим вспомогательную константу, например, *a*).

Пример 2. Для программы $\mathcal{P}$ (5) – (7) (т.е. конъюнкции формул (5), (6), (9))
$\mathcal{U}(S) = \{\text{ИВАН}, \text{ПЁТР}, \text{СЕМЁН}\}$.

Пример 3. Для программы $\mathcal{P}$ вида

$P(x) \leftarrow Q(f(x), g(x)),$

$R(y) \leftarrow$

э-универсум $\mathcal{U}(S) = \{a, f(a), g(a), f(f(a)), f(g(a)), g(f(a)), ...\}$ – бесконечное множество.

Эрбрановским базисом $B(S)$ для формулы S называется множество всех основных атомов, которые можно образовать с помощью предикатных символов из S и термов из $\mathcal{U}(S)$.

Пример 4. Для программы $\mathcal{P}$ из примера 3 $B(\mathcal{P}) = \{P(a), Q(a, a), R(a), P(f(a)), Q(a, f(a)), ...\}$. В примере 2 $B(\mathcal{P})$ конечно.

Интерпретация I для формулы S [2, 3] называется **эрбрановской интерпретацией** (э-интерпретацией), если:

а) областью интерпретации I является $\mathcal{U}(S)$;

б) константам из S в силу I отвечают «они сами»;

в) каждому n-арному функциональному символу f из S в силу I отвечает операция, которая кортежам термов $(t_1, ..., t_n) \in \mathcal{U}(S)^n$ ставит в соответствие термы $f(t_1, ..., t_n) \in \mathcal{U}(S)$.

Для полного задания э-интерпретации достаточно указать подмножество из $B(S)$, которое состоит из всех основных атомов, объявляемых истинными в этой интерпретации. Поэтому э-интерпретации иногда отождествляются с подмножествами из $B(S)$.

Э-интерпретация, в которой истинна (выполнима) формула S, называется эрбановской моделью для S (э-моделью).

Теоретико-модельная (она же декларативная) семантика логических программ совпадает с теоретико-модельной семантикой соответствующих формул логики.

Пусть S – конечное множество дизъюнктов (конъюнкция дизъюнктивных формул).

Предложение 1. *Если S выполнимо, то S имеет э-модель.*

Следствие 1. *S невыполнимо тогда и только тогда, когда S не имеет э-модели.*

Упражнения:

3. Докажите, что любая логическая программа непротиворечива (имеет модель).

4. Для формулы вида

$$\left(\forall x\ P(x,x)\&\ \forall x\forall y\forall z\ \left(P(x,y)\&\ P(y,z)\to P(x,z)\right)\&\ \forall x\forall y\left(P(x,y)\vee P(y,x)\right)\right)\to$$
$$\to\exists y\forall x\ P(y,x)$$

а) покажите, что любая интерпретация с конечной областью является моделью;

б) укажите интерпретацию, не являющуюся моделью.

5. Проверьте, что основной дизъюнкт $A_1,...,A_k \leftarrow B_1,...,B_n$ истинен в э-интерпретации I тогда и только тогда, когда хотя бы при одном $i\in\overline{1,k}\ A_i\in I$, или хотя при одном $j\in\overline{1,n}\ B_j\notin I$.

6.На примере формулы S вида $\neg P(a)\&\exists x\ P(x)$ убедитесь, что для произвольных формул S предложение 1 и следствие 1 несправедливы.

§ 3. Алгоритм унификации

Для преобразования одних выражений в другие (как правило, более частные в содержательном смысле) служат подстановки.

Подстановка – это любое конечное множество $\{t_1/v_1,...,t_n/v_n\}$, где v_i – попарно различные переменные, t_i – термы и при каждом $i=\overline{1,n}$ t_i отлично от v_i. Если все t_i – основные термы, то подстановка называется **основной**. **Чистая** подстановка – это подстановка, в которой все t_i суть переменные.

Пусть $\theta=\{t_1/v_1,...,t_n/v_n\}$ – подстановка, E – выражение (или множество выражений). Через $E\theta$ обозначается результат одновременной замены в E всех вхождений $v_i\ i=\overline{1,n}$, на t_i. $E\theta$ называется **примером** выражения E.

Если все переменные из E встречаются в множестве $\{v_1,...,v_n\}$, то θ называется **подстановкой для всех переменных выражения** E. При этом если θ – основная подстановка, то $E\theta$ называется **основным примером** выражения E.

Если θ – чистая подстановка для всех переменных, выражения E и t_i попарно различны, то $E\theta$ называется **вариантом** выражения E.

Пусть $\theta=\{t_1/u_1,...,t_n/u_n,\ \ s_1/v_1,...,s_k/v_k\}$, $\sigma=\{r_1/v_1,...,r_k/v_k,\ \ q_1/w_1,...,q_m/w_m\}$ – подстановки, где $u_1,...,u_n,\ \ v_1,...,v_k,\ \ w_1,...,w_m$ попарно различны. Тогда **композицией** $\theta\sigma$ этих подстановок называется подстановка, получающаяся из множества

$$\{t_1\sigma/u_1,...,t_n\sigma/u_n,\ \ s_1\sigma/v_1,...,s_k\sigma/v_k,\ \ q_1/w_1,...,q_m/w_m\}$$

Вычеркиванием элементов $t_i\sigma/u_i,\ \ s_j\sigma/v_j$, у которых $t_i\sigma=u_i$ или $s_j\sigma=v_j$.

Пример 5. Пусть $\theta=\{f(y)/x,\ z/y\},\ \ \sigma=\{a/x,\ b/y,\ y/z\}$. Тогда $\theta\sigma=\{f(b)/x,\ y/z\}$.

Подстановка θ называется **унификатором** множества выражений $W=\{E_1,...,E_k\}$, если $E_1\theta=...=E_k\theta$. При этом W называется **унифицируемым**.

Пример 6. Множество $W=\{P(a,y),\ P(x,f(b))\}$ унифицируемо, так как подстановка $\theta=\{a/x,\ f(b)/y\}$ является его унификатором: $W\theta=\{P(a,f(b))\}$ (одноэлементное множество.

Унификатор σ называется **наиболее общим унификатором** (ноунификатором) для $W=\{E_1,...,E_k\}$, если для любого унификатора θ множества W существует подстановка λ такая, что $\theta=\sigma\lambda$.

Для непустого множества выражений $W=\{E_1,...,E_k\}$ определим **множество рассогласований** $D(W)$. Оно получается выявлением первой (слева) позиции, на которой не для всех выражений из W стоит один и тот же символ, и затем выписыванием из каждого выражения E_j того подвыражения, которое начинается с символа, занимающего эту позицию. Множество выписанных выражений и есть $D(W)$.

Пример 7. В множестве $W=\{P(x,f(y,z)),P(x,a),P(x,g(h(k(x))))\}$ рассогласование начинается с 5-й позиции и $D(W)=\{f(y,z),a,g(h(k(x)))\}$.

Алгоритм унификации множества выражений W :

1. Полагаем $k=0, W_0=W, \sigma_0=i,$ где i – пустая подстановка.

2. Если W_k – одноэлементное множество, то останавливаемся, и подстановка σ_k есть искомый но-унификатор для $W,$ иначе строим $D_k=D(W_k)$.

3. Если в D_k содержатся элементы $v_k, t_k,$ такие что v_k – переменная, t_k – терм и v_k не встречается в $t_k,$ то переходим на шаг 4. В противном случае останавливаемся, заключая, что W не унифицируемо.

4. Полагаем, что $\sigma_{k+1}=\sigma\{t_k/v_k\}, W_{k+1}=W_k\{t_k/v_k\}$.

5. Полагаем $k=k+1$ и переходим на шаг 2.

Пример 8. Если $W=\{P(a,x,f(g(y))),P(z,f(z),f(u))\},$ то по алгоритму получается но-унификатор

$$\sigma_k=\sigma_3=\{a/z,f(a)/x,g(y)/u\}.$$

Пример 9. Если $W=\{Q(f(a),g(x)),Q(y,y)\},$ то получается, что W не унифицируемо.

Теорема 1 [4]. *Для любого конечного* (*непустого*) *множества* W *алгоритм унификации всегда заканчивает работу, причем если* W *не унифицируемо, то* – на шаге 3, если же W *унифицируемо, то* – на шаге 2, и последнее σ_k *есть но-унификатор для* W.

Доказательство теоремы 1 можно найти в монографии [5].

Упражнения:

7. Постройте композицию подстановок

$$\theta=\{a/x,\ f(z)/y,\ y/z\},\quad \sigma=\{b/x,\ z/y,\ g(x)/z\}.$$

8. Пусть E – произвольное выражение, θ,σ,μ – любые подстановки. Докажите свойства подстановок:

а) $Ei=iE=E,$

б) $(E\theta)\sigma=E(\theta\sigma),$

в) $(\theta\sigma)\mu=\theta(\sigma\mu)$.

9. Пусть θ_1,θ_2 – подстановки и существуют подстановки σ_1,σ_2 такие, что $\theta_1=\theta_2\sigma_1,\ \theta_2=\theta_1\sigma_2$. Показать, что существует чистая подстановка γ такая, что $\theta_1=\theta_2\gamma$.

10. Доказать, что множество $W_{k+1},$ которое строится по алгоритму унификации, совпадает с множеством $W\sigma_{k+1}$.

11. Выяснить унифицируемость, а в случае унифицируемости найти но-унификатор множества W:

$$W=\{Q(a),Q(b)\},\ W=\{Q(a,x,f(x)),Q(a,y,y)\},$$

$$W=\{Q(x,y,z),Q(u,h(v,v),u)\},$$

$$W=\{P(x_1,g(x_1),x_2,h(x_1,x_2),x_1,k(x_1,x_2,x_3)),\ P(y_1,y_2,l(y_2),y_3,f(y_2,y_3),y_4)\}.$$

§ 4. Наименьшая эрбрановская модель

Пусть $\mathcal{P}$ – логическая программа. Обозначим через $M(\mathcal{P})$ множество всех э-моделей программы $\mathcal{P}$. Оно не пусто (см. упр. 3). Множество $\cap M(\mathcal{P})$ есть э-модель, называемая **наименьшей э-моделью**.

Предложение 2. $\cap M(\mathcal{P})=\{A\in B(\mathcal{P}):\ A$ *есть логическое следствие* $\mathcal{P}\}$.

Пример 10. Эрбрановский базис $B(\mathcal{P})$ в примере 1 состоит из 18 атомов (найдите их). При этом в силу предложения 2 наименьшей э-моделью для $\mathcal{P}$ будет I = {ОТЕЦ (ИВАН, ПЁТР), ОТЕЦ (ПЁТР, СЕМЁН), ДЕДУШКА (ИВАН, СЕМЁН)} (заметим, что отличными от неё э-моделями являются, например: $I\cup$ {ОТЕЦ (ИВАН, СЕМЁН)}, $I\cup$ {ОТЕЦ (СЕМЁН, ИВАН), ДЕДУШКА (СЕМЁН, ПЁТР), ДЕДУШКА (ПЁТР, ИВАН)}).

Множество $2^{B(\mathcal{P})}$ всех подмножеств эрбрановского базиса (т.е. множество всех э-интерпретаций программы $\mathcal{P}$) образует полную решётку [6]относительно частичного порядка, задаваемого теоретико-множественным включением $\subseteq$.

Определим отображение $T_{\mathcal{P}}$ из решётки э-интерпретаций в себя. Пусть I – э-интерпретация. Тогда $T_{\mathcal{P}}(I)=$ $\{A\in B(\mathcal{P}): A\leftarrow B_1,...,B_n$ есть основной пример некоторого дизъюнкта из $\mathcal{P}$ и $\{B_1,...,B_n\}\subseteq I\}$ $(n\geq 0)$.

Предложение 3. I *– э-модель для* $\mathcal{P}$ *тогда и только тогда, когда* $T_{\mathcal{P}}(I)\subseteq I$.

Введём обозначения: $T\uparrow 0=\varnothing,\ T\downarrow 0=B(\mathcal{P}),\ T\uparrow n=T(T\uparrow(n-1)),\ T\downarrow n=T(T\downarrow(n-1)),$, если n – непосредственно следующий ординал [3]; $T\uparrow n=\sup\{T\uparrow k: k<n\},\ T\downarrow n=\inf\{T\downarrow k: k<n\},$ если n – предельный ординал.

Скажем, что интерпретация I – **наименьшая неподвижная точка** отображения $T_{\mathcal{P}}: 2^{B(\mathcal{P})}\to 2^{B(\mathcal{P})}$, если I – неподвижная точка (т.е. $T_{\mathcal{P}}(I)=I$) и для всех неподвижных точек I^1 отображения $T_{\mathcal{P}}$ имеем $I\subseteq I^1$. Наименьшая неподвижная точка отображения $T_{\mathcal{P}}$ обозначается $l(T_{\mathcal{P}})$. Аналогично определяется **наибольшая неподвижная точка**: $g(T_{\mathcal{P}})$.

Теорема 2 [7]. $\cap M(\mathcal{P})=l(T_{\mathcal{P}})=T_{\mathcal{P}}\uparrow\omega=\cup\{T_{\mathcal{P}}^n(\varnothing): n<\omega\}$.

Пример 11. Наименьшая э-модель I из примера 10 является, как легко проверить, неподвижной точкой отображения $T_{\mathcal{P}}$, а по теореме 2 и наименьшей его неподвижной точкой.

Пусть $\mathcal{P}$ – программа, G – цель $\leftarrow B_1,...,B_n$. **Ответной подстановкой для** $\mathcal{P}\cup\{G\}$ называется подстановка только для переменных из G, но не обязательно для всех. При этом все константы и функции из подстановки считаются уже встретившимися в $\mathcal{P}\cup\{G\}$.

Корректная ответная подстановка для $\mathcal{P}\cup\{G\}$ – это такая ответная подстановка θ, что универсальное замыкание формулы $(B_1\,\&...\&\,B_n)\theta$ есть логическое следствие из $\mathcal{P}$.

Пример 12. Ответными подстановками в примере 1 могут быть: $\{$ИВАН$/X\}$, $\{$СЕМЁН$/X\}, i$ и др. Не являются ответными, например, подстановки: $\{$ИВАН$/X$, ПЁТР$/Y\}$, $\{$ИВАН$/Y\}$, $\{$АНТОН$/X\}$. Корректной ответной подстановкой является лишь подстановка $\{$ИВАН$/X\}$.

Предложение 4. *Пусть* θ *– ответная подстановка для* $\mathcal{P}\cup\{G\}$ *такая, что формула* $(B_1\&...\&B_n)\theta$ *не содержит переменных. Следующие утверждения эквивалентны друг другу:*

а) θ *– корректная ответная подстановка,*

б) $(B_1\&...\&B_n)\theta$ *истинно в любой э-модели для* $\mathcal{P}$*,*

в) $(B_1\&...\&B_n)\theta$ *истинно в наименьшей э-модели для* $\mathcal{P}$*.*

Упражнения:

12. Пусть L – непустое множество э-моделей для программы $\mathcal{P}$.

Доказать, что $\cap L$ есть э-модель.

13. Почему $\cap M(\mathcal{P})$ – непустое множество?

14. Пусть S – конечное множество дизъюнктов, L – непустое множество э-моделей для S. Показать, что $\cap L$ не обязательно модель.

15. Проверить, что множество всех э-интерпретаций для программы $\mathcal{P}$ является полной решёткой относительно теоретико-множественного отношения $\subseteq$.

16. Пусть $\mathcal{P}$ – программа вида

$P(a)\leftarrow P(x), Q(x),$

$P(f(x))\leftarrow P(x),$

$Q(b)\leftarrow ,$

$Q(f(x))\leftarrow Q(x),$

Показать, что $T_{\mathcal{P}}\downarrow\omega=\{P(f^n(a)), Q(f^n(b)): n<\omega\}$, где $f^n(x)=\underbrace{f(f(...(f}_{n\text{ раз}}(x)...)$, и поэтому $T_{\mathcal{P}}\downarrow\omega\neq g(T_{\mathcal{P}})$.

17. Проверить, что:

а) если цель G не содержит переменных (основной дизъюнкт), то единственной возможной ответной подстановкой является i;

б) предложение 4 в общем случае несправедливо, если не требовать условия отсутствия переменных в $(B_1\&...\&B_n)\theta$.

§ 5. Обоснование метода ВЛП-резолюции

В § 4 введено декларативное определение корректности ответной подстановки. Рассмотрим механизм отыскания корректных ответных подстановок [8]. В него заложен некоторый вариант резолюционного метода опровержения, развивающий метод резолюции из [4] и называемый ВЛП-резолюцией [9, 10] (SLD-resolution – Линейная резолюция с правилом Выбор для Программных дизъюнктов).

Правилом выбора называется функция из множества целей в множество атомов, имеющая в качестве своего значения на каждой цели некоторый атом этой цели, называемый **выбранным атомом**.

Пусть $\mathcal{P}$ – программа, G – цель, R – правило выбора. **ВЛП-вывод для** $\mathcal{P}\cup\{G\}$ посредством R состоит из (конечной или бесконечной) последовательности $G_0=G,G_1,G_2,...$ целей, последовательности $H_1,H_2...$ вариантов программных дизъюнктов (из $\mathcal{P}$ и последовательности $\theta_1,\theta_2,...$ но-унификаторов, причём каждое G_{i+1} выводимо из G_i и H_{i+1} посредством θ_{i+1} и R. Последнее означает: если G_i есть цель $\leftarrow A_1,...,A_m,...,A_n$ и H_{i+1} – вариант дизъюнкта из $\mathcal{P}$ вида $A\leftarrow B_1,...,B_q$, то G_{i+1} называется **выводимым** из G_i и H_{i+1} посредством но-унификатора θ_{i+1} и R, если выполняются следующие условия:

а) A_m – выбранный (посредством R) атом

б) $A_m\,\theta_{i+1} = A\theta_{i+1}$ (т.е. θ_{i+1} – но-унификатор для $\{A_m, A\}$),

в) G_{i+1} есть цель $\leftarrow (A_1,...,A_{m-1}, B_1,...,B_q, A_{m+1},...,A_n)\theta_{i+1}$.

В терминологии метода резолюции G_{i+1} есть *резольвента* для G_i и H_{i+1}. Каждое H_{i+1} есть **подходящий вариант** соответствующего программного дизъюнкта так что H_{i+1} и G_i не имеют общих переменных.

Опровержением (ВЛП-опровержением) для $\mathcal{P}\cup\{G\}$ посредством R называется вывод, содержащий в последовательности целей пустой дизъюнкт. Этот дизъюнкт по необходимости – последняя цель в этом выводе. Если $G_n = \square$, то говорят, что опровержение имеет длину n.

Пример 13. Для программы $\mathcal{P}$ и цели G из примера 1 примем в качестве R правило выбора из текущей цели первого (слева) атома. Набор следующих трёх последовательностей является ВЛП-опровержением для $\mathcal{P}\cup\{G\}$:

$G_0 = G,$

$G_1 = \leftarrow \text{ОТЕЦ}(X, Z), \text{ОТЕЦ}(Z, \text{СЕМЁН}),$

$G_2 = \leftarrow \text{ОТЕЦ}(\text{ПЁТР}, \text{СЕМЁН}),$

$G_3 = \square;$

$H_1 = (6)\{W / X\}$ (см. пример 1), $H_2 = (4),\ H_3 = (5);$

$\theta_1 = \{X / W, \text{СЕМЁН} / Y\},\ \theta_2 = \{\text{ИВАН} / X, \text{ПЁТР} / Z\},\ i.$

Только ВЛП-выводом (но не опровержением) является следующий текст:

$G'_0 = G,$

$G'_1 = \leftarrow \text{ОТЕЦ}(X / Z), \text{ОТЕЦ}(\text{Z}, \text{СЕМЁН}),$

$G'_2 = \leftarrow \text{ОТЕЦ}\,(\text{СЕМЁН}, \text{СЕМЁН});$

$H'_1 = (6)\{W / X\},\ H'_2 = (5),\ H'_3 = (4);$

$\theta'_1 = \{X / W, \text{СЕМЁН} / Y\},\ \theta'_2 = \{\text{ПЁТР} / X, \text{СЕМЁН} / Z\}.$

Структура опровержения иллюстрируется рис. 2.

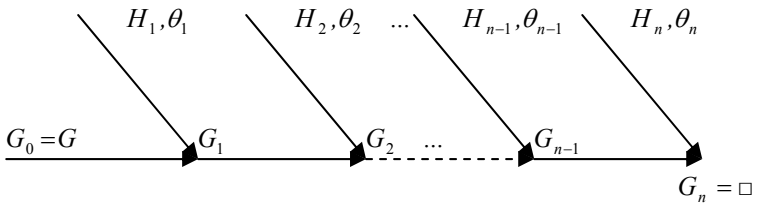

Рис. 2.

Множеством успехов программы $\mathcal{P}$ называется множество всех $A \in B(\mathcal{P})$ таких, что $\mathcal{P}\cup\{\leftarrow A\}$ имеет опровержение (с использованием правила выбора, зависящего от A).

R-ответной подстановкой θ для $\mathcal{P}\cup\{G\}$ называется всякая подстановка, которая может быть получена ограничением композиции $\theta_1\theta_2...\theta_n$ на переменные из G, где $\theta_1, \theta_2, ..., \theta_n$ – последовательность но-унификаторов, использованных в некотором опровержении для $\mathcal{P}\cup\{G\}$ посредством R.

Следующая теорема обосновывает метод ВЛП-резолюции.

Теорема 3. *Каждая R-ответная подстановка для $\mathcal{P}\cup\{G\}$ является корректной ответной подстановкой.*

Пример 14. Образуем композицию но-унификаторов, использованных в опровержении из примера 13. Получим $\theta_1\theta_2 i=\theta_1\theta_2=\{$ИВАН / W, СЕМЁН / Y, ИВАН / X, ПЁТР / Z$\}$. Ограничение этой композиции на переменные из G есть $\{$ИВАН / X$\}$. Это и есть R-ответная подстановка для $\mathcal{P}\cup\{G\}$. По теореме 3 она корректна.

Полнота метода ВЛП-резолюции устанавливается следующей теоремой.

Теорема 4. *Для каждой корректной ответной подстановки θ множества $\mathcal{P}\cup\{G\}$ существует правило выбора R, R-ответная подстановка σ для $\mathcal{P}\cup\{G\}$ и некоторая подстановка γ такие, что θ совпадает с композицией*

Пример 15. Пусть программа $\mathcal{P}$ состоит из одного факта и $D_1=P(x,a)\leftarrow$, а цель G есть $\leftarrow P(x,y)$. Для любого правила выбора R вариант H_1 дизъюнкта D_1, например, вида $H_1=P(\omega,a)\leftarrow$ в совокупности с $G=G_0$ потребует, например, подстановки $\theta_1=\{x/\omega, a/y\}$, после чего цель G_1 оказывается пустой. Рассматривая произвольно другие варианты H_1 и строя но-унификаторы θ_1 для G_0 и H_1, будем получать θ_1 либо вида $\{x/u, a/y\}$, либо $\{u/x, a/y\}$, где u – переменная из подстановки σ, образующей подходящий вариант H_1 на базе дизъюнкта D_1, т.е. $H_1=D_1\sigma$, $\sigma=\{u/x\}$. Не будет в качестве R - ответной подстановки получаться подстановка $\theta_2=\{a/x, a/y\}$, которая тем не менее является корректной ответной подстановкой, так как $P(x,y)\theta_2=P(a,a)$ есть логическое следствие программы $\mathcal{P}$ ($P(a,a)$ выводимо из $\forall xP(x,a)$). Подстановкой γ, о которой идет речь в теореме 4, будет либо $\gamma=\{a/x\}$, либо $\gamma=\{a/u\}$. Действительно, в первом случае $\theta_1=\{x/u, a/y\}$ и R-ответной подстановкой будет $\sigma=\{a/y\}$. При этом $\theta_2=\sigma\gamma$. Во втором случае $\theta_1=\{u/x, a/y\}=\sigma$, θ_2 совпадает с композицией $\sigma\gamma$ после её ограничения на переменные из G.

Из теорем 3 и 4 вытекают сформулированные ниже следствия.

Следствие 2. *Множество $\mathcal{P}\cup\{G\}$ невыполнимо тогда и только тогда, когда для него существует опровержение.*

Следствие 3. *Множество успехов программы $\mathcal{P}$ равно её наименьшей э-модели $\cap M(\mathcal{P})$. Более того, если $A\in B(\mathcal{P})$ и $\mathcal{P}\cup\{\leftarrow A\}$ имеет опровержение длины n, то $A\in T_{\mathcal{P}}\uparrow n$.*

Пример 16. Длина опровержения множества $\mathcal{P}\cup\{\leftarrow A\}$, где $\mathcal{P}$ – программа из примера 1, а $A=$ДЕДУШКА $($ИВАН, СЕМЁН$)\in B(\mathcal{P})$, равна 3 ($G_3=\square$, см. пример 13). По следствию 3 $A\in T_{\mathcal{P}}\uparrow 3$. Действитель-

но, $T_{\mathcal{P}}\uparrow 0=\varnothing$, $T_{\mathcal{P}}\uparrow 1=T_{\mathcal{P}}(\varnothing)=\{$ОТЕЦ $($ИВАН, ПЁТР$)$, ОТЕЦ $($ПЁТР, СЕМЁН$)\}$,

$T_{\mathcal{P}}\uparrow 2=T_{\mathcal{P}}(T_{\mathcal{P}}\uparrow 1)=(T_{\mathcal{P}}\uparrow 1)\cup\{$ДЕДУШКА $($ИВАН, СЕМЁН$)\}$, $A\in T_{\mathcal{P}}\uparrow 2\subseteq T_{\mathcal{P}}\uparrow 3$.

Этот пример, в частности, показывает, что из $A\in T_{\mathcal{P}}\uparrow n$ при некотором $n<\omega$ не следует с необходимостью существование для $\mathcal{P}\cup\{\leftarrow A\}$ ВЛП-опровержения длины n.

Упражнения:

18. Убедитесь непосредственно, что $A \in T_{\mathcal{P}} \uparrow 1$, где $\mathcal{P}$ – программа из примера 15, $A = P(a,a)$ (при длине опровержения $\mathcal{P} \cup \{ \leftarrow A \}$ тоже равной 1).

19. Пусть $\mathcal{P}$ – программа, G – цель. Приведите ещё один пример корректной ответной подстановки, которая не может быть получена как R-ответная подстановка для $\mathcal{P} \cup \{G\}$ ни при каком правиле выбора R.

20. Пусть $\mathcal{P}$ – программа, $A = A(x_1,...,x_n)$ – атом с переменными $x_1,...,x_n$. Доказать эквивалентность следующих утверждений:

а) $\mathcal{P} \to \forall x_1 ... \forall x_n \, A(x_1,...,x_n)$ общезначимо,

б) $\mathcal{P} \cup \{ \leftarrow A(a_1,...,a_n) \}$ невыполнимо, где $a_1,...,a_n$ – константы, не встречающиеся в $\mathcal{P}$ или A,

в) существует опровержение для $\mathcal{P} \cup \{ \leftarrow A(x_1,...,x_n) \}$ с подстановкой i в качестве R-ответной подстановки.

§ 6. Независимость правила выбора

Следствие 2 показывает, что если $\mathcal{P} \cup \{G\}$ невыполнимо, то существует опровержение для $\mathcal{P} \cup \{G\}$ с использованием некоторого правила выбора R. Оказывается [8–11], что R может быть заранее специфицировано (т.е. наперед задано) и тогда, если $\mathcal{P} \cup \{G\}$ невыполнимо, мы можем всегда найти опровержение, используя заданное правило R. Этот факт известен как «независимость правила выбора».

Теорема 5. *Если $\mathcal{P} \cup \{G\}$ имеет ВЛП-опровержение посредством R, а R' – любое правило выбора, то существует ВЛП-опровержение для $\mathcal{P} \cup \{G\}$ посредством R'.*

Эта теорема позволяет усилить следствие 2 и теорему 4.

Пусть R – произвольное фиксированное правило выбора.

Следствие 4. *Множество $\mathcal{P} \cup \{G\}$ невыполнимо тогда и только тогда, когда для него существует опровержение посредством R.*

Следствие 5. *Для каждой корректной ответной подстановки θ множества $\mathcal{P} \cup \{G\}$ существует R-ответная подстановка σ для $\mathcal{P} \cup \{G\}$ и некоторая подстановка γ такие, что $\theta = \sigma\gamma$ (с точностью до ограничения $\sigma\gamma$ на переменные из G).*

Упражнение:

21. Дайте усиленную формулировку следствия 3, применив теорему 5.

§ 7. О процедурах ВЛП-опровержения

Всякое опровержение по определению есть некоторая синтаксическая конструкция. Как находить эти конструкции?

Пространством поиска опровержений является дерево некоторого, определяемого далее, вида. При построении этого дерева для сокращения его размерности правило выбора можно считать наперед фиксированным в соответствии с § 6.

Поисковое дерево. ВЛП-дерево для $\mathcal{P} \cup \{G\}$ и R определяется так:

а) каждая вершина дерева есть цель (возможно, пустая);

б) корневая вершина есть G;

в) пусть $\leftarrow A_1,...,A_m,...,A_n \ (n \geq 1)$ есть вершина в этом дереве, а A_m – атом, выбранный посредством R. Тогда эта вершина имеет потомка для каждого программного дизъюнкта $D = A \leftarrow B_1,...,B_q$ такого, что A_m и A' унифицируемы, A' – заголовок подходящего варианта $A' \leftarrow B'_1,...,B'_q$ дизъюнкта D. Это – потомок вида

$$\leftarrow (A_1,...,A_{m-1},B'_1,...,B'_q,A_{m+1},...,A_n)\theta,$$

где θ – но-унификатор для A_m и A';

г) вершины, которые являются пустыми дизъюнктами, не имеют потомков.

Каждая ветвь этого дерева есть вывод из $\mathcal{P} \cup \{G\}$. Те ветви, которые завершаются пустым дизъюнктом, являются ВЛП-опровержениями для $\mathcal{P} \cup \{G\}$ – (**успешные ветви**). В ВЛП-дереве могут быть бесконечные ветви. Некоторые непустые цели могут не иметь потомков. Это – цели, выбранные атомы которых не унифицируемы с заголовком ни одного программного дизъюнкта. Этим **тупиковым целям** отвечают и **тупиковые ветви**, в них ведущие.

Если задано ВЛП-дерево, то **правилом поиска** называется стратегия поискана дереве успешных ветвей. **Процедура ВЛП-опровержения** определяется своими правилами выбора и поиска.

Способ фиксации правила выбора R влияет на размерность дерева, которое для одного R может быть конечным, а для другого R – бесконечным. Тем не менее из следствий 4 и 5 вытекают следующие утверждения.

Пусть R – любое фиксированное правило выбора.

Следствие 6. *Если $\mathcal{P} \cup \{G\}$ невыполнимо, то ВЛП-дерево для $\mathcal{P} \cup \{G\}$ и R имеет по крайней мере одну успешную ветвь.*

Следствие 7. *Каждая корректная ответная подстановка θ для $\mathcal{P} \cup \{G\}$ ображается» на ВЛП-дереве для $\mathcal{P} \cup \{G\}$ и R, т.е. для θ имеется успешная ветвь, которой соответствует R-ответная подстановка, более общая чем θ.*

В связи с применением логического программирования в базах данных нас могут интересовать не одна, а все R-ответные подстановки. Поэтому желательно, чтобы правило поиска удовлетворяло такому требованию, что любая успешная ветвь им будет в конечном счёте найдена (требование полноты относительно выводимости).

При поиске на деревьях используются разные стратегии: «вначале вглубь», «вначале вширь» и др. (об этом читайте в монографии [12]).

Если же ВЛП-дерево бесконечно и используется стратегия «вначале вглубь», то такое правило поиска может не удовлетворять выставленному требованию полноты. С другой стороны, стратегия «вначале вширь» иногда оказывается неэффективной из-за сложной комбинаторики.

Само логическое программирование обеспечивает удобные возможности для двух основных видов параллелизма: так называемый И-параллелизм, который возможен, если одновременно могут преследоваться разные подцели

$$\leftarrow A_1,...,A_n,$$

и ИЛИ-параллелизм, который возможен, если в связи с рассматриваемой целью могут в одно и то же время использоваться разные альтернативные правила:

$$A \leftarrow B_1,...,B_m,$$
$$.....$$
$$A \leftarrow B'_1,...,B'_k.$$

Вместе с тем пока большинство созданных ПРОЛОГ-систем логического программирования (исключая LOGLISP [13] и некоторые другие) используют стратегию «вначале вглубь» и механизм возврата (backtracking) при попадании в тупиковую вершину. Функционирование таких систем с механизмом возврата – строго последовательное. В них правило поиска сводится к **правилу упорядочения**, т.е. к правилу, указывающему, в каком порядке должны быть испытаны программные дизъюнкты. Упорядочение дизъюнктов в $\mathcal{P}$ и их испытание всегда в соответствии с этим порядком – простое и часто достаточно эффективное правило поиска, хотя и имеет недостаток, состоящий в том, что для каждой цели порядок испытания фактов и правил жёстко фиксирован, т.е. один и тот же.

Рассмотрим полноту ПРОЛОГ-систем, в которых используется механизм возврата и фиксированный порядок испытания программных дизъюнктов.

При этом большинство ПРОЛОГ-систем (см., например, [14]) используют правило выбора R, которое всегда выбирает первый (слева) атом цели. Однако имеются системы и с более сложными правилами R, например IC-PROLOG [15], MU-PROLOG [16] и др. По следствию 6, если $\mathcal{P} \cup \{G\}$ невыполнимо, независимо от R соответствующее ВЛП-дерево всегда содержит успешную ветвь. Тем не менее система с механизмом возврата, фиксированным порядком испытания программных дизъюнктов и произвольным R не гарантирует отыскания успешной ветви. Это – плата за отказ, например, от полной стратегии поиска «вначале вширь» (которую более обоснованно использовать в базах данных).

Рассмотрим примеры.

Пример 17. Пусть $\mathcal{P}$ и G из примера 1 (см. также обозначения примера 13). Выбрав в качестве R правило, упомянутое выше, а в качестве порядка на $\mathcal{P}$ порядок ((4), (5), (6)), получим дерево просмотра целей

$$G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow \square.$$

Поменяв порядок на ((5), (4), (6)), при том же R получим дерево

$$G_0 \rightarrow G_1 \rightarrow G'_2,$$

где G'_2 – тупиковая вершина. Возвращаясь к цели G_1 и используя следующий, ещё не испытанный, дизъюнкт в $\mathcal{P}$, т.е. факт (4), получим G_2 и далее $\square$. В целом, дерево просмотра целей примет вид, представленный на рис. 3.

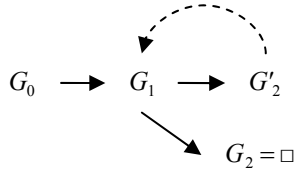

Рис. 3.

Если в качестве R для той же программы $\mathcal{P}$ и цели G примем правило выбора самого последнего (слева) атома цели, то при любом порядке на $\mathcal{P}$ получается дерево

$$G_0 \rightarrow G_1 \rightarrow G''_2 \rightarrow \square,$$

где G''_2 есть цель $\leftarrow$ОТЕЦ $(X,$ ПЁТР).

Пример 18. Пусть $\mathcal{P}$ – программа вида:

$$P(a,b)\leftarrow\ , \tag{11}$$

$$P(c,b)\leftarrow\ , \tag{12}$$

$$P(x,z)\leftarrow\ P(x,y),P(y,z), \tag{13}$$

$$P(x,y)\leftarrow P(y,x), \tag{14}$$

а G – цель $\leftarrow P(a,c)$. Используя произвольный, но фиксированный порядок в $\mathcal{P}$ и любое правило выбора R, система с механизмом возврата (и стратегией «вначале вглубь») никогда не найдет опровержения. Действительно, дизъюнкты (13) и (14) имеют заголовки, не содержащие констант и функциональных символов. Поэтому их заголовки унифицируемы с любой (непустой) целью, которая может возникнуть. Если дизъюнкт (13) предшествует по порядку испытаний дизъюнкту (14), то система никогда не дойдет до дизъюнкта (14) (и наоборот). Однако все дизъюнкты (11)–(14), как нетрудно проверить, необходимы для опровержения.

Пример 19 (более полный пример и его модификации см. в [14]). Рассмотрим задачу выбора механика для включения его в состав формируемой экспедиции. Программа $\mathcal{P}$ содержит набор фактов, устанавливающих специальности предполагаемых участников экспедиции и состояние их здоровья, а также правило включения в экспедицию:

КТО-ЕСТЬ-КТО (БОТАНИК, ВАСЯН) ← ,
КТО-ЕСТЬ-КТО (СИНОПТИК, ФЕДЯКОВ) ← ,
КТО-ЕСТЬ-КТО (МЕХАНИК, КАЛЯЕВ) ← ,
КТО-ЕСТЬ-КТО (МЕХАНИК, САЖИН) ← ,
ЗДОРОВ (ВАСЯН) ← ,
ЗДОРОВ (ФЕДЯКОВ) ← ,
ЗДОРОВ (САЖИН) ← ,
ВКЛЮЧИТЬ (СПЕЦИАЛЬНОСТЬ, ФАМИЛИЯ) ← КТО-ЕСТЬ-КТО (СПЕЦИАЛЬНОСТЬ, ФАМИЛИЯ), ЗДОРОВ (ФАМИЛИЯ)
(«специальность» и «фамилия» объявляются переменными).

В качестве цели решения задачи выберем цель G вида

← ВКЛЮЧИТЬ (МЕХАНИК, ФАМИЛИЯ).

Считая программу упорядоченной так, как она записана, правилом выбора R – выбор первого (слева) атома и используя возврат, получим дерево просмотра целей, представленное на рис. 4.

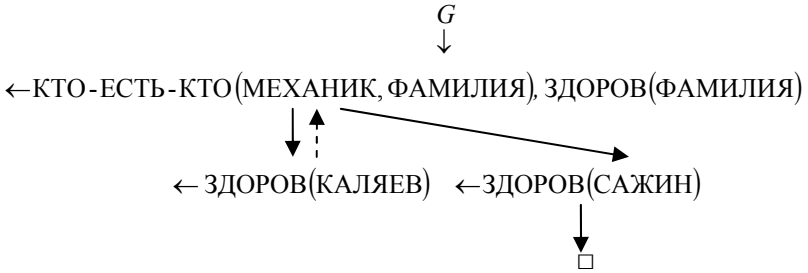

Рис. 4.

При этом на успешной ветви образуются

$\theta_1 =$ {ФАМИЛИЯ / X, МЕХАНИК / СПЕЦИАЛЬНОСТЬ},

$\theta_2 =$ {САЖИН / ФАМИЛИЯ},

$\theta_3 = i$,

где X – переменная из подстановки, образующей подходящий вариант дизъюнкта (15). Композиция подстановок $\theta_1 \theta_2 i$ имеет вид {САЖИН / X, МЕХАНИК /СПЕЦИАЛЬНОСТЬ, САЖИН / ФАМИЛИЯ}. R - ответной подстановкой явится подстановка {САЖИН /ФАМИЛИЯ}.

Упражнения:

22. Приведите пример программы и цели, для которых ВЛП-дерево конечно в случае одного правила выбора и бесконечно при другом.

23. Приведите пример, раскрывающий существенность требования предварительного преобразования программного дизъюнкта в подходящий вариант при построении потомков в поисковом дереве.

24. Чем отличается поисковое дерево от дерева просмотра целей (см. примеры 17 и 19)?

25. Проверить, что всякое опровержение для множества $\mathcal{P} \cup \{G\}$ из примера 18 использует каждый из четырёх дизъюнктов программы.

26. Для выработки навыков формализации и освоения изложенного материала поупражняйтесь в составлении логических программ и решении Ваших задач.

Заключение

Основной тезис логического программирования утверждает, алгоритмизация решения задач состоит в указании двух компонент: «логики» и «управления». Логика определяет, ЧТО за задача должна быть решена. Управление определяет, КАК она должна быть решена. Идеалом логического программирования является то, чтобы программист (пользователь) указывал бы (специфицировал)только логическую компоненту задачи. Управление должно осуществляться исключительно системой логического программирования (обычно некоторым интерпретатором).

Однако этот идеал, как мы видели в § 7, ещё не достигнут.

Достижение идеала логического программирования предполагает решение двух основных проблем. Первая из них – это проблема **управления**. В настоящее время программистам приходится указывать разную управляющую информацию частично упорядочением дизъюнктов и атомов в дизъюнктах, а частично введением в дизъюнкты как бы на правах атомов таких нелогических управляющих признаков, как признак отсечения (который подавляет повторный просмотр поддеревьев, нежелательный в некоторых случаях). Однако эти управляющие признаки недостаточно удовлетворительны по ряду причин [8]. Первой задачей в решении проблемы управления является предоставление программистам более удовлетворительных управляющих средств. Второй задачей является передача ответственности за управление от программиста самой системе.

Вторая проблема логического программирования – это проблема работы с **отрицаниями**. Хорновские дизъюнкты не имеют достаточной выразительной силы, и поэтому для вывода негативной информации применяются дополнительные специальные правила, например правило неуспеха [17]. В логические программы, а именно в тела дизъюнктов, вводят отрицания атомов (на практике многие программы более естественно записываются именно так). Продолжаются исследования по разработке таких программ и в целом по реализации отрицания в системах логического программирования.

Ряд практических задач сводится к проверке выполнимости логических формул (т.е. существованию модели [2, 3]). Соответствующий подход реализован в системах логического программирования CLP (Constraint Logic Programming). Задача с неполной информацией или с ограниченными ресурсами, в том числе задачи диагностики, объяснения наблюдений, автоматизации построения теорий, нуждаются в третьем под-

ходе к логическому программированию – абдуктивном подходе, в рамках которого могут отыскиваться недостающие условия и конструктивные средства для разрешения поставленной задачи.

Исчисление позитивно-образованных формул [19] в сравнении с методом резолюций обладает рядом преимуществ (более выразительный язык, меньшая комбинаторность поиска выводов, совместимость с эвристиками предметной области, модифицируемость семантики и др.). Является актуальным использование этой теоретической базы для создания более эффективных практических систем логического программирования во всех указанных выше классах задач.

Использованная литература

1. *Глушков В.М.* Кибернетика // Математическая энциклопедия. Т. 2. С. 850–855. М.: Советская энциклопедия, 1979.

2. *Ершов Ю.Л., Палютин Е.А.* Математическая логика. М.: Наука, 1979.

3. *Мендельсон Э.* Введение в математическую логику. М.: Мир, 1971.

4. *Робинсон Дж.* Машинно-ориентированная логика, основанная на принципе резолюции // Киберн. сб. Нов. сер. М.: Мир, 1970. Вып. 7. С. 194–218.

5. *Чень Ч., Ли Р.* Математическая логика и автоматическое доказательство теорем. М.: Наука, 1983.

6. *Мальцев А.И.* Алгебраические системы. М.: Наука, 1970.

7. *Emden M., Kowalski R.* The Semantics of Predicate Logic as a Programming Language // JACM. 23. 4. P. 733–742.

8. *Lloyd J.* Foundation of Logic Programming // Tech. Report. 82/7. Univ. Melbourne, 1983.

9. *Apt K., Emden M.* Contributions to the Theory of Logic Programming // JACM. 29. 3. P. 841–862.

10. *Clark K.* Predicate Logic as a Computational Formalism // Res. Report. 79/59. Imper. College, 1980.

11. *Hill R.* LUSH-Resolution and its Completeness // DGL Memo 78. Univ. Edinburgh, 1974.

12. *Уинстон П.* Искусственный интеллект. М.: Мир, 1960.

13. *Robinson J., Sibert E.* Logic Programming in LISP / School of Computer and Information Science. Syracuse Univ., 1980.

14. Система «ПРОЛОГ-ЕС» Введение в ПРОЛОГ (инструкция для пользователя). Киев: ИК АН УССР, 1979.

15. *Clark K., McCabe F.* The Control Facilities of IC-PROLOG, in Expert Systems in the Micro Electronic Age. Edinburg Univ. Press. P. 122–149.

16. *Hayes P.* Computation and Deduction // Proc. MFCS Conf., Czechoslovakian Acad. Sci., 1973.

17. *Clark K.* Negation as Failure, in Logic and Data-bases // Plenum Press. N.Y.,1978. P. 293–322.

18. Fifth Generation Computer Systems // Proc. Internat. Conf. on Fifth Generation Computer Systems / T. Moto-Oka ed. North-Holland Publ. Comp. Amsterdam; N.Y.; Oxford, 1982.

19. *Васильев С.Н., Жерлов А.Л., Федосов Е.А., Федунов Б.Е.* Интеллектное управление динамическими системами. М.: Наука, ФИЗМАТЛИТ, 2000.

20. *Клоксин У., Мелиш К.* Программирование на языке Пролог. М.: Мир, 1987 (F.W. Clocksin, C.S. Mellish. Programming in Prolog. Springer-Verlag, 1981).

21. *Братко И.* Программирование на языке Пролог для искусственного интеллекта. М.: Мир, 1990 (I. Bratko. Prolog Programming for Artificial Intelligence. Addison-Wesley Publ. Comp. Inc. , Wokingham, 1986).

22. *Непейвода Н.Н.* Прикладная логика. Уч. пос. 2-е изд., исп. и доп. Новосибирск: Изд-во Новосиб. ун-та, 2000.

23. *Kakas A.C., Kowalski R.A., Toni F.* The Role of Abduction in Logic Programming . Handbook of Logic in Artificial Intelligence and Logic Programming / Eds. D.M. Gabbay, C.J. Hoger, J.A. Robinson. Oxford Univ. Press, 1998.

24. *Вагин В.Н., Головина Е.Ю., Загорянская А.А., Фомина М.В.* Достоверный и правдоподобный вывод в интеллектуальных системах / Под ред. В.Н. Вагина, Д.А. Поспелова. М.: ФИЗМАТЛИТ, 2004.

25. *Вагин В.Н., Хотимчук К.Ю.* Методы абдуктивного вывода в задачах планирования работы в сложных объектах // Изв. РАН. Теория и системы управления. 2010. № 5. С. 95–113.

26. Автоматическое порождение гипотез в интеллектуальных системах / Сост. Е.С. Панкратова, В.К. Финн; под ред. В.К. Финна. М.: Книжный дом «ЛИБРОКОМ», 2009.

27. *Cox P.T., Pietrzykowski T.* General Diagnosis by Abductive Inference // Proc. of the IEEE Symposium on Logic Programming. 1987. P. 183–189.

28. *Matyasiki P., Nalepai G.J., Zieciki P*. Prolog-Based Real-Time Intelligent Control of the Hexor Mobile Robot // http: // ai. ia. agh. edu. pl / wiki/_media / hekate: bib:ptm-ki 2007.pdf