ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

УДК 519.6
DOI: 10.18101/2304-5728-2019-2-28-43

ОБ АЛГЕБРАИЧЕСКОЙ МОДЕЛИ КОМПЬЮТЕРНОЙ ПРОГРАММЫ

© Николаева Дарима Доржиевна
научный сотрудник,
Бурятский государственный университет имени Доржи Банзарова
Россия, 670000, г. Улан-Удэ, ул. Смолина, 24а
E-mail: darisha89@yandex.ru

© Ширапов Дашадондок Шагдарович
доктор физико-математических наук, профессор,
Бурятский государственный университет имени Доржи Банзарова
Россия, 670000, г. Улан-Удэ, ул. Смолина, 24а
E-mail: shir48@mail.ru

В статье рассматривается функциональная грамматика с точки зрения алгоритмических алгебр. В результате выявлен простейший инвариантный базис — пара (имя, значение), который в дальнейшем может служит унифицированным базисом для построения более сложных структур в памяти компьютера при реализации объявлений сложных объектов в компьютерных программах. Получена лемма о коммутативности преобразования операции присваивания и допустимого состояния памяти программы, возвращающей значение по имени переменной. Логическая связь между гомоморфизмом многоосновных алгебр данных и гомоморфизмом допустимых состояний памяти, выраженная операцией присваивания, устанавливается теоремой о коммутативности диаграммы в декартово-замкнутых категориях. Пример компьютерного моделирования классической динамической системы с использованием функциональных грамматик представлен в виде системы обыкновенных дифференциальных уравнений с заданными начальными условиями. Для демонстрации использования функциональных грамматик представлен алгоритм, порождающий решение обыкновенного дифференциального уравнения.
***Ключевые слова**:* динамические системы; моделирование; математическая модель языка; универсальная алгебра; контекстно-свободная грамматика; оператор присваивания; допустимое состояние памяти; семантика.

Для цитирования:
Николаева Д. Д., Ширапов Д. Ш. Об алгебраической модели компьютерной программы // Вестник Бурятского государственного университета. Математика, информатика. 2019. № 2. С. 28–43.

Введение

В настоящее время имеется большое количество работ по системному программированию, которые посвящены применению и развитию понятий формальных грамматик и многоосновных универсальных алгоритмических алгебр [2; 4]. Последние, в свою очередь, образуют теоретические основания системного программирования и глубоко проработаны. Многоосновные универсальные алгоритмические алгебры и формальные грамматики нашли широкое применение в практических работах по реализации приложений системного программирования и тем самым накопили достаточно большой практический опыт в разработках больших комплексов программ. Однако существует в задачах компьютерного моделирования динамических систем свободная ниша, где можно было бы применить логически мощный аппарат формальных грамматик, а также гибкий выразительный аппарат многоосновных универсальных алгоритмических алгебр. Любую компьютерную программу, которая порождает вычислительный процесс, можно рассматривать как пример динамической системы. Для этого в качестве пространства состояний динамической системы можно выбрать номер оператора программы и состояние памяти:

$$S = \{(m, b) \mid m - \text{номер оператора программы}, b - \text{состояние памяти}\},$$

где S — пространство состояний.

С другой стороны, существуют теоретические исследования по математическому моделированию динамических систем, где особенно выделяются работы по позитивно-образованным формулам [1] своей универсальностью при построении математических моделей динамических систем. В статье делается попытка поиска логических связей между алгоритмическими алгебрами, формальными грамматиками (основаниями компьютерных наук) и теорией динамических систем, основанной на позитивно образованных формулах.

1 Постановка задачи

Рассмотрим задачу обобщения оператора присваивания в языках программирования для компьютерного моделирования динамических систем. Для решения этой задачи мы рассмотрим структуры, которые возникают в памяти компьютера при объявлениях объектов (переменных) в программе с использованием концепции косвенного именования. Пусть в сигнатуре типов Ξ выделено подмножество Ξ_0 и задано вложение $\alpha : \Xi_0 \to \Xi$. Подмножество Ξ_0 является сигнатурой типов без именования. Содержательно отображение α соответствует операции разыменования или является операцией получения значений, используя имя переменной или константы. Тогда α^{-1} обратное отображение соответствует операции именования или операции адресования. Отметим, что семанти-

ку языка программирования, в частности языка, предназначенного для компьютерного моделирования динамических систем, можно представить в виде многоосновной алгебры A [2; 4], где любая программа x семантически представляет собой терм этой алгебры, построенной в соответствии с синтаксической грамматикой (или синтаксической алгеброй). При пошаговом исполнении на компьютере программы x происходит преобразование состояния памяти компьютера. Если обозначить состояние памяти через b_i, то это преобразование означает переход из состояния b_i в новое состояние b_j, которое осуществляется при выполнении одного оператора программы x. Требуется найти логическую связь между гомоморфизмом алгебр данных и преобразованием допустимых состояний памяти, выраженную через алгебру операторов $Y(V, \Omega, \Pi, \Xi)$ (операции присваивания).

2 Область допустимых состояний памяти

Точное определение понятия допустимого состояния памяти основывается на формальной грамматике G_{-1}, которая является точной моделью памяти. Любое допустимое состояние памяти принадлежит множеству слов, которое порождается грамматикой G_{-1}. Такой подход позволяет более точно определить границы между семантикой языка и прагматикой, последняя порождается программой x, которая решает какую-либо прикладную задачу. Для выяснения содержания отображений α и α^{-1}, а также множества типов Ξ_0 приведем два примера: если в тексте программы объявлена переменная «цел z», то фактический тип переменной z равен «имя цел», так как в памяти компьютера зарезервировано место для хранения целого значения переменной, упомянутое место должно адресоваться (указываться) или именоваться. Схематически это можно изобразить в виде следующей диаграммы:

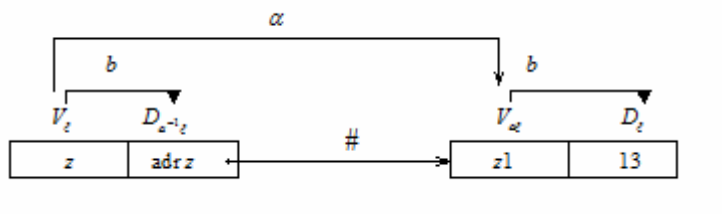


Рис. 1

На диаграмме (рис. 1) через V_ξ обозначено множество имен всех переменных, имеющих тип $\xi = \underline{\text{имя}}\ \underline{\text{цел}}$, где $\underline{\text{цел}} \in \Xi_0$. Через $D_{\alpha^{-1}\xi}$ обозначено множество всех значений, которые соответствуют именам из V_ξ, где сопоставление именам их значений представляется через отображение $b : V_\xi \to D_{\alpha^{-1}\xi}$. Отображение b, с одной стороны, выражает операцию чтения значения по имени или возвращение значения $b(z)$, с другой стороны, b описывает состояние памяти. В нашем случае отображение b является сужением более общего отображения $b \in D^V$, где V — множество всех имен объектов, имеющих значения во множестве всех возможных значений. Необходимо обратить внимание на нижние индексы V_ξ и $D_{\alpha^{-1}\xi}$, где $\alpha^{-1}\xi$ при $D_{\alpha^{-1}\xi}$ означает множество разыменованных значений, имеющих тип ξ. Другими словами, если сказанное интерпретировать на нашем примере, то адрес выделенного места для переменной z обозначен через $adr\,z$, которое принадлежит множеству $D_{\alpha^{-1}\xi}$ и является непосредственным значением имени z. С этой точки зрения объект z по своим свойствам совпадает с константой, так как адрес $adr\,z$ неизменно связан с именем z. Отметим, что мы рассматриваем статическое объявление переменной z. При исполнении объявления записи «$\underline{\text{цел}}\ \ z$» в некоторой программе происходит выделение места m по адресу $adr\ z$ для хранения динамического значения d_z переменной z. Однако для указанного динамического значения переменной z генерируется условное имя $z1$. Значением $z1$ будет динамическое значение d_z переменной z. Таким образом, выделенное место m представляет собой пару $(z1, d_z)$. Статическое значение переменной z будет $adr\,z$. Для рис. 1 $d_z = 13$. Отметим, что операция извлечения содержимого по адресу «$adr\,z$» возвращает нам пару, где именующую часть $z1$ вполне естественным образом можно отождествить с адресом «$adr\,z$». Для этой пары ($z1$, <значение>), принадлежащей декартовому произведению $V_{\alpha\xi} \times D_\xi$, отметим отображение $b : V_{\alpha\xi} \to D_\xi$, которое описывает состояние памяти. Здесь $V_{\alpha\xi}$ — множество всех имен объектов, имеющих тип $\alpha\xi$, D_ξ — множество значений, имеющих тип ξ. На диаграмме (рис. 1) операция α обозначает операцию разыменования, а α^{-1} обозначает операцию именования, $\alpha^0 = 1$ является тождественным отображением.

Перейдем к рассмотрению объявлений «имя цел *u*» и «имя^3 цел *v*». Для объектов *u* и *v* диаграммы представляются аналогичным образом, в связи с этим мы представим только основные обозначения, которые могут применяться на диаграмме без приведения рисунка. На самом верхнем четвертом уровне переменная v принадлежит множеству $V_{\alpha^{-3}\xi}$, которое обозначает множество всех переменных уровня «имя^3 ξ», где ξ — простейший тип без косвенного именования. Через $D_{\alpha^{-4}\xi}$ обозначено множество всех адресов, которые являются значениями переменных из множества $V_{\alpha^{-3}\xi}$. Между множествами $V_{\alpha^{-3}\xi}$ и $D_{\alpha^{-4}\xi}$ существует отображение-вложение $b : V_{\alpha^{-3}} \to D_{\alpha^{-4}\xi}$, которое отождествляет имя переменной с его значением. В функциональном программировании аналогом b служит отображение-вложение $* : V_{\alpha^{-3}\xi} \to D_{\alpha^{-4}\xi}$. Отметим, что свойство тройки $(b, V_{\alpha^{-3}\xi}, D_{\alpha^{-4}\xi})$ для переменной v во многом совпадает со свойством тройки $(b, V_{\alpha\xi}, D_{\xi})$, относящейся к константе. Однако отличие заключается в том, что исполнение отображения b для константы происходит при статической фазе работы компилятора, тогда как для тройки $(b, V_{\alpha^{-n}\xi}, D_{\alpha^{-n-1}})$ при $n \geq 0$ исполнение функции b происходит в процессе исполнения программы моделирования (динамическая фаза работы программы). Через v обозначено имя переменной, где $v \in V_{\alpha^{-3}\xi}$. В функциональной грамматике переменной v соответствует нетерминальный символ X, из которого порождается множество имен $\{v\}$, то есть $X \overset{*}{\Rightarrow} v$. Через $adr\, v1$ обозначено значение переменной v на рассматриваемом уровне, мы получаем тождественную пару $(v, adr\, v1)$, где v — имя и $adr\, v1 = b(v)$ — его значение. С точки зрения функциональных грамматик $*(v) =< адрес\ v > 1 \equiv adr\, v1$ есть адрес выделенного для переменной v места. Содержимое выделенного места должно содержать пару $(v1, adr\, v1)$, где $v1$ — новое имя переменной v и $v1 \in V_{\alpha^{-2}\xi}$ на уровне ниже рассматриваемого, $adr\, v1$ есть значение переменной $v1$. Таким образом, исходя из концепции косвенного именования «имя^n тип X», можно построить многоуровневые алгебраические модели прикладных задач, что существенно облегчает проектирование программных комплексов. На верхнем уровне происходит разработка математической модели проектируемого комплекса программ в виде многоосновной алгебраической системы, которая освобождена от рассмотрения мелких деталей задачи. Такой подход значительно облегчает логическое представление прикладной задачи. Связь между смежными уровнями математических моделей мож-

но выразить через структурно согласованные грамматики видов и памяти, в терминах которых можно описать логическую модель проектируемой программы. Приведем пример структурно согласованных грамматик видов G_1, $G_2^{(1)}$, которые логически связывают алгебраические модели смежных уровней [3]:

Грамматика $G_1 = (V_{T1}, V_{N1}, P_1,$ ПРАВИД) − структура видов.

ПРАВИД : : = ВИД | ничто ; ВИД : := АВИД | АБСТР ВИД ;
АВИД : := лог | цел ; АБСТР : := ε | АБСТР имя ;

Упорядоченность на M_1: $g_1^{(1)}$ = (имя ВИД) ВИД .

Пара структурно согласованных нетерминалов АБСТР из грамматики G_1 и ЦЕПЬ из грамматики $G_2^{(1)}$ моделируют концепцию косвенного именования. ЦЕПЬ является конкретизацией АБСТР. $G_2^{(1)}$ Грамматика $G_2^{(1)} = (V_{T2}, V_{N2}, P_2,$ ПРАЗНАЧ) – структура значений.

ПРАЗНАЧ : := ЗНАЧ | неопр ; ЗНАЧ : := АЗНАЧ | ЦЕПЬ АЗНАЧ ;
АЗНАЧ : := ЛОГ | ЦЕЛ ; ЦЕПЬ : : = ε | ЦЕПЬ *XH # XH* * ;
ЦЕЛ : := ЦЦЦ | неопр ; ЛОГ : := истина | ложь | неопр ;
H : := ε | Ц | ЦЦ | ЦЕЛ ; Ц : := 0 | 1 | 2 . . . | 9 .

Упорядоченность на M_{-1}:

$g^{(2)}$ = (конт *t*, знач *x*) знач : (*M# XH*P M*$^{(1)}$ ω *F*, *XH*) *P*.

Из представленных грамматик видно, что существует квазиизоморфизм, то есть некоторое подобие изоморфизма, а именно:

ПРАВИД $\rightarrow$ ПРАЗНАЧ ; ВИД $\rightarrow$ ЗНАЧ; АВИД $\rightarrow$ АЗНАЧ;
АБСТР $\rightarrow$ ЦЕПЬ; имя $\rightarrow$ *H # XH* * ;
лог $\rightarrow$ ЛОГ : : = истина | ложь | неопр ; цел $\rightarrow$ ЦЕЛ : : = ЦЦЦ .

Из вышесказанного видно, что грамматика $G_2^{(1)}$ является продолжением грамматики G_1 : терминальные символы грамматики G_1 находят дальнейшую конкретизацию. Грамматику $G_2^{(1)}$ назовем продолжением грамматики G_1 по терминалам имя, лог, цел.

Из приведенного примера структурно согласованных грамматик G_{-1}, G_1 и G_2 следует идея выразительного построения логической связи между алгебраическими системами смежных уровней. Под структурной согласованностью понимается логические и смысловые связи между нетерминальными символами и терминальными строками, которые задают математические структуры семантико-прагматических отображений

$\{\Phi_i^{(1)}\},\{\Phi_i^{(2)}\},\{g_j^1\},\{g_j^2\}$. Характерным частичным свойством межуровневой структурной согласованности является $V_{N-1}\cap V_{N1}\cap V_{N2}\neq\varnothing$, $V_{T-1}\cap V_{T1}\cap V_{T2}\neq\varnothing$. Где V_{N-1},V_{N1},V_{N2} — множества нетерминальных символов и V_{T-1},V_{T1},V_{T2} множества терминальных символов грамматик G_{-1},G_1,G_2. Следующим важным достижением формального описания структуры типов, значений и состояния памяти (грамматика G_{-1}) является точное формальное описание множества допустимых состояний памяти. Множество слов, порождаемых этими грамматиками, в точности описывает структуру приведенных диаграмм.

Перейдем к рассмотрению операции присваивания $v:=u$. Тип объекта u равен $\alpha^{-2}\xi$, он задает контекстные условия для определения значения переменной v. Контекстный тип ζ переменной v должен быть равным $\zeta=\alpha^{-3}\xi$, то есть иметь вид имя $\alpha^{-2}\xi$. Формула $val_\zeta(t,b)$ для вычисления значения v при операции присваивания должна быть следующей:

$$val_\zeta(v,b)=b^m(v),$$

где m вычисляется из соотношения

$$\begin{cases}\alpha^{m-1}\circ\eta=\zeta\\ \alpha^{m-1}(\alpha^{-4}\xi)=\alpha^{-3}\xi\end{cases}\Rightarrow\ m-1=1;m=2.\ val_\zeta(v,b)=b^m(v)=b^2(v)=adr\,v2.$$

Дадим определение терма:

1. Любой символ переменной типа ξ или символ нульарной операции типа (ξ) есть терм типа ξ.

2. Если ω есть символ операции со схемой $(\xi_1,...,\xi_n,\xi)$, а $t_1,...,t_n$ термы типов $\xi_1,...,\xi_n$ соответственно, то $\omega(t_1,...,t_n)$ есть терм типа ξ.

3. Других термов нет.

Допустимыми состояниями памяти на интуитивном уровне будем понимать те состояния, структура которых изоморфна приведенным выше диаграммам, которые описывают структуру объектов (переменных) различных типов. Другое формальное определение допустимого состояния памяти дается ниже при помощи контекстно-свободной грамматики G_{-1}. Если $t(v_1,...,v_n)$ есть терм из переменных $v_1,...,v_n\in V$, а $b(v_1),...,b(v_n)\in D$ значения этих переменных соответственно, где b является допустимым состоянием памяти из множества допустимых со-

стояний B, то определено значение терма t, которое равно выражению $t(b(v_1),...,b(v_n))$. То есть имеем равенство:

$$b(t(v_1,...,v_n)) = t(b(v_1),...,b(v_n)).$$

Отметим, что в некоторых случаях могут оказаться неопределенными некоторые значения $b(v_i)$ и, несмотря на это, можно будет вычислить значение терма $t(b(v_1),...,b(v_n))$ (например, случай с термом «если-то-иначе»). Новое состояние $b(t(v_1,...,v_n)) = t(b(v_1),...,b(v_n))$ будет допустимым состоянием памяти, если исходные значения принадлежали множеству допустимых состояний.

Сформулируем обобщенное правило вычисления значения терма t в условиях косвенной адресации.

Для любого допустимого состояния памяти b терма t, имеющего тип η, определим функцию $val_\zeta(t,\eta,b)$ — значение терма t относительно типа ζ при состоянии памяти b. Для этого необходимо рассмотреть два случая:

1. Если $t = v \in V_\eta$ и существует $m \geq 0$, такое, что $\alpha^{m-1} \circ \eta = \zeta$, то $val_\zeta(t,\eta,b) = b^m(t)$, где m — минимально.

2. Пусть $t = \omega(t_1,...,t_n)$, операция ω имеет тип $(\zeta_1,...,\zeta_n,\eta)$ и существует $m \geq 0$, такое, что $\alpha^m\eta = \zeta$. Тогда $val_\zeta(t,\eta,b) = b^m(\omega(d_1,...,d_m))$, где $d_i = val_{\zeta_i}(t_i,\eta_i,b)$ $(i = \overline{1,n})$, m берется минимальным.

Пусть $v_1,...,v_n$ — переменные, $t_1,...,t_n \in T(\Omega)$ — термы, V — множество всех переменных, $T(\Omega)$ — множество всех термов сигнатуры Ω, B — множество всех допустимых состояний памяти и $y = (v_1 := t_1,...,v_n := t_n)$ является операцией присваивания. Естественным является рассмотрение вопроса о преобразовании операцией присваивания у множества допустимых состояний памяти B, то есть $y : B \to B$, и вопроса о преобразовании термов $y : T(\Omega) \to T(\Omega)$. Приступим к рассмотрению преобразования $y : B \to B$. Если $b \in B$, то, обозначив через b_1 преобразованное состояние памяти $y(b)$, мы получим следующие соотношения для b_1

$$b_1(v) = y(b)(v) = \begin{cases} b(t_i), \ \textit{если} \ v{=}v_i, \ i{\in}\overline{1,n}, \\ b(v), \ \textit{если} \ v{\neq}v_i, \ i{\in}\overline{1,n}. \end{cases} \qquad (1)$$

Если подробно рассмотреть соотношение (1), то

$$b_1(v_i) = (y(b))(v_i) = b(t_i) = b(\omega_i(s_1,..,s_k)) = \omega_i(b(s_1),...,b(s_k)),$$

где $t_i = \omega_i(s_1,...,s_k)$ и $s_1,...,s_k \in T(\Omega)$.

Пример 1. Пусть состояние памяти b равно $b = ((v_1,5); (v_2,6); (v_3,7); (v_4,13); (v_5,11))$ и $y = (v_2 := v_1 * v_3; v_4 := 9)$. Вычислим новое состояние

$$b_1 = y(b) = ((v_1,5); (v_2,35); (v_3,7); (v_4,9); (v_5,11)).$$

При переходе состояния памяти из b в новое состояние $b_1 = y(b)$ изменяются значения переменных v_2 (с 6 на 35) и v_4 (с 13 на 9). Остальные переменные v_1, v_3, v_5 не изменяют своего значения.

Каждый оператор присваивания $y = (v_1 := t_1, v_2 := t_2)$ определяет преобразование термов $T(\Omega)$, то есть $y : T(\Omega) \to T(\Omega)$, если положить

$$y(t) = \begin{cases} t_i, & \text{если } t = v_i,\ i = \overline{1,n}, \\ v, & \text{если } t = v \neq v_i,\ i = \overline{1,n}, \\ \omega(y(s_1),...,y(s_m)), & \text{если } t = \omega(s_1,...,s_m),\ s_1,...,s_m \in T(\Omega). \end{cases} \quad (2)$$

Из формул (1) и (2) вытекает следующая лемма.

Лемма 1. Справедливо следующее соотношение $(y(b))(v) = b(y(v))$.

Пример 2. Пусть $y = (v_1 := t_1, v_2 := t_2, v_3 := t_3)$, $t = \omega(s_1, s_2)$, где $s_1 = v_1$ и $s_2 = v$, $\omega \in \Omega$, $t_1, t_2, t_3, t, s_1, s_2 \in T(\Omega)$.

Вычислим $y(t) = y(\omega(s_1, s_2)) = y(\omega(v_1, v)) = \omega(y(v_1), y(v)) = \omega(t_1, v)$.

Таким образом, $\omega(t_1, v) \in T(\Omega)$.

Пример 3. Пусть $y = (v_1 := t_1, v_2 := t_2)$, $t = \omega_1(s_1, \omega_2(s_1, s_3))$, где $s_1 = v_1, s_2 = v_2, s_3 = v$, t_1, t_2, t все являются термами, принадлежащими множеству $T(\Omega)$. Вычислим $y(t)$ и убедимся, что он принадлежит $T(\Omega)$.

$$y(t) = y(\omega_1(s_1, \omega_2(s_2, s_3))) = \omega_1(y(s_1), y(\omega_2(s_2, s_3))) = \omega_1(y(v_1), \omega_2(y(s_2), y(s_3))) =$$

$$\omega_1(t_1, \omega_2(y(v_2), y(v))) = \omega_1(t_1, \omega_2(t_2, v)),\ y(t) = \omega_1(t_1, \omega_2(t_2, v)) \in T(\Omega).$$

Пример 4. Пусть $y = (v_1 := t_1, v_2 := t_2)$, $t = \omega_1(v_1, \omega_2(v_2, v, s)) \in T(\Omega)$, где $s \in T(\Omega)$.

Вычислим

$$y(t) = y(\omega_1(v_1, \omega_2(v_2, v, s))) = \omega_1(y(v_1), \omega_2(y(v_2), y(v), y(s))) =$$

$$\omega_1(t_1, \omega_2(t_2, v, y(s))),$$

которое также принадлежит $T(\Omega)$.

Рассмотрим и докажем теорему, которая является полезной при разработке структур данных при проектировании комплексов программ. Если имеется типизированный базис $(U(V,\Omega,\Pi,\Xi),Y(V,\Omega,\Pi,\Xi))$ для схем программ над типизированной памятью, то можно рассматривать стандартные схемы программ над памятью и стандартные интерпретации. Поясним обозначения: V — множество переменных, Ω — сигнатура операций, Π — сигнатура предикатов, Ξ — типы или сорта несущих множеств многоосновной алгебры, $U(V,\Omega,\Pi,\Xi)$ есть *алгебра условий*, а $Y(V,\Omega,\Pi,\Xi)$ есть *алгебра операторов*. Допустим, $D=\bigcup\limits_{\xi\in\Xi}D_\xi$ и $D_1=\bigcup\limits_{\xi\in\Xi}D_{1\xi}$ — две однотипные (Ω,Π,Ξ)-алгебры данных с основами несущих множеств $(D_\xi)_{\xi\in\Xi}$ и $(D_{1\xi})_{\xi\in\Xi}$. Тогда гомоморфизмом $\gamma':D\to D_1$ называется семейство $\gamma=(\gamma_\xi)_{\xi\in\Xi}$ отображений, такое, что для любых $\xi\in\Xi$, $\gamma_\xi:D_\xi\to D_{1\xi}$ для любой операции $\omega\in\Omega$, который имеет схему $(\xi_1,...,\xi_n,\xi)$, и любого предиката π со схемой $(\xi_1,...,\xi_n)$ выполняются следующие соотношения:

$$\gamma_\xi(\omega(d_1,...,d_n))=\omega(\gamma_{\xi_1}(d_1),...,\gamma_{\xi n}(d_n)),$$
$$\pi(d_1,...,d_n)=1\Leftrightarrow\pi(\gamma_{\xi_1}(d_1),...,\gamma_{\xi_n}(d_n)).$$

Для строгого гомоморфизма выполняется необходимое и достаточное условие, а для обычного гомоморфизма выполняется только условие необходимости. Пусть D и D_1 — две многоосновные однотипные $\Omega-\Pi$-алгебры данных, $B=D^V$ и $B_1=D_1^V$ — непустые множества допустимых состояний памяти. Тогда гомоморфизм $\gamma:D\to D_1$ порождает отображение $\gamma':B\to B_1$, определенное равенством $(\gamma'(b))(v)=\gamma_\xi(b(v))$ для любого $v\in V_\xi$. В связи с этим сформулируем и докажем теорему.

Теорема 1. Для многоосновных $\Omega-\Pi-\Xi$-алгебр данных $D=(D_\xi)_{\xi\in\Xi}$ и $D_1=(D_{1\xi})_{\xi\in\Xi}$ коммутативна следующая диаграмма (рис. 2) из декартово-замкнутой категории и справедливо следующее соотношение

$$(\gamma'(b))(y(v))=\gamma_\xi((y(b))(v)).$$

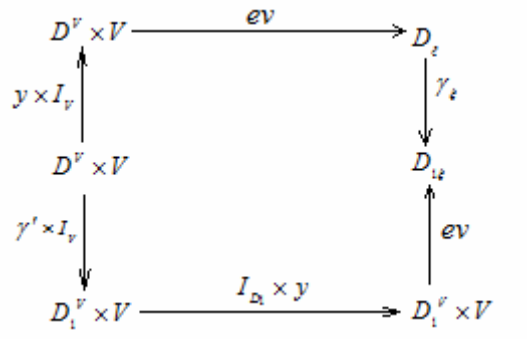


Рис. 2

Доказательство. Разъясним обозначения: $I_V : V \to V$ и $I_{D_1} : D_1^V \to D_1^V$ — тождественные отображения, $y \times I_V$, $\gamma' \times I_V$ и $I_{D_1} \times y$ — декартовые произведения двух отображений. Пусть $(b,v) \in D^V \times V$. Тогда суперпозиция отображений $y \times I_V$, ev и γ_ξ, начинающая с элемента (b,v), дает результат $\gamma_\xi((y(b)(v))$. Опираясь на лемму 1 и на определение отображения γ', получаем $\gamma_\xi((y(b)(v)) = \gamma_\xi(b(y(v))) = (\gamma'(b))(y(v)) \in D_{1\xi}$. Суперпозиция отображений $\gamma' \times I_V$, $I_{D_1} \times y$ и ev, начинающаяся с элемента $(b,v) \in D^V \times V$, дает результат $(\gamma'(b))(y(v)) \in D_{1\xi}$. Что и требовалось доказать.

3 Пример компьютерного моделирования динамической системы с использованием функциональных грамматик

Пусть задана динамическая система, где заданными являются: f, x_0, t_0.

$$\dot{x} = f(x,t); \quad x(t_0) = x_0; \; f(x_0,t_0) = f_0. \tag{3}$$

Для моделирования данной классической динамической системы рассмотрим общеизвестные конечно-разностные приближения для уравнения (3).

$$\begin{gathered} x(t_1) - x(t_0) = f(x(t_0),t_0) \cdot \Delta t; \\ x(t_{i+1}) = x(t_i) + f(x(t_i),t_i) \cdot \Delta t; \\ x(t_{i+1}) - x(t_i) = f(x(t_i),t_i) \cdot \Delta t, \quad i = 0,1,... \end{gathered} \tag{4}$$

Для наглядности применения функциональных грамматик к моделированию динамических систем специально выбрали простейшее уравнение первого порядка.

Из (4) получим программу, которая моделирует динамическую систему.

t_0 – starting moment; x_0 – initial state;

x = preceding state; x_n – next state;
t – current tine; t_n – finish time.

```
1. Program DinSystem;
2. var   x_n, x, x, dt, x_0, t_0, t : real;        n: integer;
3. begin
4.    read (x_0, t_0, deltat, t_n);
5.   t:=t_0;  x:=x_0;  x_n:=x_0
6.        while    t<t_n    do   begin
7.                x_n  :=  x  +  f(x,t)*deltat ;
8.                 t  :=  t+deltat;
9.                 writeln (“ t=    “,   t, “  x_n=     “, x_n );
10.                x := x_n
11.          end
12.    end.
```

Заметим, что заданная программа порождает процесс, она сама же является динамической системой, но не является математической моделью. Таким образом, перед нами стоит задача представить данную программу в виде математической модели. Имея математическую модель данной программы, мы имеем возможность исследовать динамическую систему, представленную в виде компьютерной программы.

Вычислительный процесс, моделирующий динамическую систему (3), начинается со строки 5 программы DimSystem: $t = t_0$, $x_n = x_0$. Функция цикла Ф10 (строка 6 – строка 11) включена в процесс моделирования Ф10 (<сравнение>, <операторы>) или Ф10 (L,Q), где L − это сравнение, а Q − операторы.

<сравнение> : := “ $t < t_n$ ” t t_n Ф16

<операторы> : :=

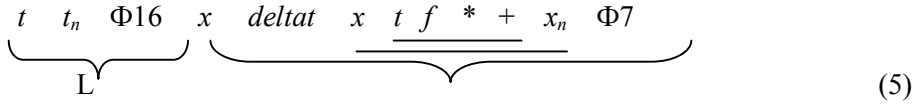
“ $x_n := x + deltat * f(x,t)$ Ф7

$t := t + deltat$ Ф7

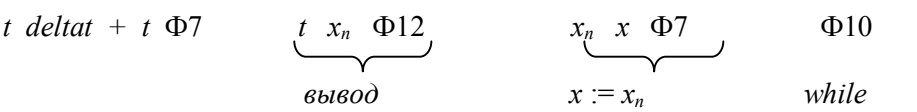
вывод (t, x_n) Ф12

$x := x_n$ ” Ф7

В постфиксной записи приведенная программа представляется в виде следующего алгебраического терма от базисных функций



(5)



Приведенную запись следует читать слева направо, так как это постфиксная запись функции. При этом переменные t, t_0, x, *deltat*, t — одноместные функции, где аргументом является имя переменной, а результатом будет значение переменной. Ф16 — базисная функция <сравнение>, результат возвращает логический *true* или *false*. Если обозначить логическое выражение $t\ t_n$ Ф16 (<сравнение>) через L, а остальную часть последовательности (5) — через Q (без Ф10), то имеем терм $L\ Q$ Ф10. Функция Ф10 представляет собой цикл, в нотации алгоритмических алгебр записывается в виде $\{Q\}$ и называется L-итерацией. С точки зрения алгоритмических алгебр $L\ Q$ Ф10 = $\{Q\}$ L-итерация дает в качестве результата оператор R, такой, что для любого состояния памяти b, bR принимается равным первому из состояний ряда b, bQ, $(bQ)Q = bQ^2$, bQ^3, … , для которого истинно условие bQ^mL. Если же такого состояния в указанном ряду не окажется, то результат применения оператора R к состоянию b считается неопределенным. Рассмотрим терм $L\ Q$ Ф10 с точки зрения функциональных грамматик. Для этого необходимо привести из синтаксиса языка программирования две нотации:

1. <операторы> : := <операторы> <оператор> Ф4
2. <цикл> : := пока <сравнение> выполнить <операторы > конец Ф10

Из синтаксиса видно, что у функции цикла Ф10 два аргумента: <сравнение> обозначим через L и <операторы> обозначим через Q. Приведем алгоритм работы базисных функций Ф10:

Ф10 = (функ x, y) знач: (x, y) r_1 (x, x, y)

r_1 = (знач x, функ y, z) знач: ((истина, y, z) $f_4(z, r_1\ (y, y, z))$ | (ложь, $y, z,$) ε);

В алгоритм r_1 входит базисная функция Ф4, которая представляется в виде

Ф4 = (знач x, y) знач: (ЗНАЧ, ЗНАЧ$^{(1)}$) ЗНАЧ$^{(1)}$.

Заметим, что аргументы x и y у функции Ф4 вызываются по значениям.

Сформулируем следующую теорему об эквивалентности операции L-итерации алгоритмических алгебр и базисной функции цикла Ф10.

Теорема 2. Результаты *L*-итерации $\{Q\}$ и терма $L\ Q$ Ф10 совпадают.

Доказательство. Пусть состояние памяти находится в *b*. Подставляем вместо аргументов *x* и *y*, *L* и Q соответственно в базисную функцию Ф10 и получим следущее:

Ф10 $(L, Q) = (L, Q)\ r_1(L, L, Q)$;

$r_1(L, L, Q) = (\ (\ bL =$ истина, $L, Q)$ Ф4$(bQ, r_1(L, L, Q))\ \ |\ (bL =$ ложь, $L, Q)$ $\varepsilon\ )$,

ε означает, что память находится в состоянии bQ^m. Из рекурсии видно, что функция Ф10 порождает ряд $b,\ bQ,\ (bQ)Q = bQ^2,\ bQ^3, \ldots$. Указанный ряд обрывается на каком-то члене ряда, скажем, на номере = 0, 1, 2,… при условии $bQ^mL=$ложь, и дает в качестве результата состояние памяти bQ^m. Если $m = \infty$, то функция Ф10 дает неопределенность (происходит зацикливание). **Теорема доказана**.

Замечание. Из теоремы видно, что моделирование динамической системы (3) является адекватной.

Введем обозначения: обыкновенное дифференциальное уравнение (ОДУ) обозначим через U_0; множество конечно-разностных уравнений (КРУ) через U_k ; множество алгоритмов на Паскале через P ; множество суперпозиций семантических функций через F ; гладкое решение ОДУ через R ; множество ломаных кривых через L ; обозначим через T множество малых вещественных чисел. Итак, имеем шестерку $< U_0, U_k, P, F, R, L >$. В данной шестерке множество U_k определяется как $U_k = \{U_k[\Delta t] \,|\, \Delta t \in T\}$, для каждого зафиксированного числа Δt имеем конкретное конечно-разностное уравнение $U_k[\Delta t]$. Также определяются в этой шестерке и другие множества: $P = \{P[\Delta t] | \Delta t \in T\}$ — множество программ, где $P[\triangle t] \in P$; $F = \{F[\Delta t] | \Delta t \in T\}$ — множество суперпозиций функций, где $F[\triangle t] \in F$; $L = \{L[\Delta t] | \Delta t \in T\}$ — множество аппроксимирующихся ломаных к гладкому решению R обыкновенно-дифференциального уравнения, где $L[\triangle t] \in L$. В данном случае величина $\triangle t$ не является параметром программы, а считается составной частью программы $P[\triangle t]$, которая входит в тело программы $P[\triangle t]$.

Целью всех построений $< U_0, U_k, P, F, R, L >$ и рассуждений является следующее: при $\triangle t \to 0$ имеется фундаментальная сходимость последовательности $P[\triangle t] \to P_k$ (P_k — конечная), где P_k порождает ломаную L_k (L_k -конечная), такую, что для $\forall \varepsilon > 0\ \ | L[\triangle t_i] - L[\triangle t_{i+1}] | < \varepsilon$. В данном случае имеем практическую (прагматическую) цель, которая является построением конечной программы, дающей заданную точность.

Каждая конкретная программа $P[\triangle t]$ принадлежит некоторому классу эквивалентных программ, каждый из которых может порождать одну и ту же ломаную $L[\triangle t]$. Среди этих программ могут быть и «хорошие» программы, и «плохие». Вопрос выбора хорошей программы P_x из класса эквивалентности относится к задаче оптимизации программы $P[\triangle t]$. В данной работе этот вопрос не затрагивается.

Заключение

В статье рассмотрен логический анализ условно генерируемого имени, которое является результатом исполнения объявления переменной в программе и наследуемого от понятия константы статической части переменной. Данный анализ выявил простейший инвариантный базис — пару (имя, значение). Данный инвариантный базис в дальнейшем может служит унифицированным базисом для построения более сложных структур в памяти компьютера при реализации объявлений сложных объектов в компьютерных программах. В связи с этим рассмотрены различные примеры о преобразовании множеств состояний памяти, а также множеств термов операцией присваивания. Также установлена логическая связь между гомоморфизмом многоосновных алгебр данных и гомоморфизмом допустимых состояний памяти, выраженная операцией присваивания. Получены и доказаны лемма 1 и теорема 1 о коммутативности диаграммы в декартово-замкнутых категориях.

Литература

1. Интеллектное управление динамическими системами / С. Н. Васильев [и др.]. М.: Физико-математическая литература, 2000. 352 с.

2. Капитонова Ю. В., Летичевский А. А. Математическая теория проектирования вычислительных систем. М.: Наука, 1988. 296 с.

3. Николаева Д. Д., Ширапов Д. Ш., Антонов В. И. Об одном подходе к моделированию динамических систем // Вестник Бурятского государственного университета. Математика, информатика. 2018. № 2. С. 95–109.

4. Тузов В. А. Математическая модель языка. Л.: ЛГУ, 1984. 176 с.

ON ALGEBRAIC MODELING OF COMPUTER PROGRAMS

Darima D. Nikolaeva
Researcher,
Dorzhi Banzarov Buryat State University
24a Smolina St., Ulan-Ude 670000, Russia
E-mail: darisha89@yandex.ru

Dashadondok Sh. Shirapov
Dr. Sci. (Phys. and Math.), Prof.,
Dorzhi Banzarov Buryat State University
24a Smolina St., Ulan-Ude 670000, Russia
E-mail: shir48@mail.ru

The article considers functional grammar in terms of algorithmic algebras. As a result, we have identified the simplest invariant basis (name — value pair), which can later serve as a unified basis for building more complex structures in computer memory when implementing declarations of complex objects in computer programs. It is obtained a lemma on commutativity of assignment operator transfer and valid state of program memory, which returns a value by the name of variable.

The logical connection between the homomorphism of multibase data algebras and the homomorphism of valid memory states is established by the theorem of diagrams commutativity in Cartesian closed categories and expressed by an assignment operator. An example of computer modeling of classical dynamic system using functional grammars is presented as a system of ordinary differential equations with given initial conditions. To demonstrate the application of functional grammars we have used an algorithm that generates a solution to an ordinary differential equation.

Keywords: dynamic systems; modeling; mathematical model of language; universal algebra, context-free grammar; assignment operator; valid memory state; semantics.

References

1. Vasilyev S. N., Zherlov A. K., Fedosov E. A., Fedunov B. E. *Intellektnoe upravlenie dinamicheskimi sistemami* [Intellectual Control of Dynamic Systems]. Moscow: Fiziko-matematicheskaya literatura Publ., 2000. 352 p.

2. Kapitonova Yu. V., Letichevskii A. A. *Matematicheskaya teoriya proektirovaniya vychislitelnykh sistem* [Mathematical Theory of Computer Systems Design]. Moscow: Nauka Publ., 1988. 296 p.

3. Nikolaeva D. D., Shirapov D. Sh., Antonov V. I. Ob odnom podkhode k modelirovaniyu dinamicheskikh system [On One Approach to Modeling Dynamic Systems]. *Vestnik Buryatskogo gosudarstvennogo universiteta. Matematika, informatika.* 2018. No. 2. Pp. 95–109.

4. Tuzov V. A. *Matematicheskaya model yazyka* [A Mathematical Model of Language]. Leningrad: Leningrad State University Publ., 1984. 176 p.