Абстрактные программы с процедурами и конечные автоматы с магазином

Р.И. Подловченко

Формализуется понятие программы с процедурами. Строится семейство моделей программ с процедурами; элементами каждой модели являются схемы программ с процедурами. Отбираются аппроксимирующие модели. Среди них - модель, обобщающая схемы Янова на случай использования в программе процедур; она названа максимальной. Устанавливается ряд свойств максимальной модели. В частности, по каждой схеме программ с процедурами построен детерминированный автомат с магазином, принимающий язык, порожденный схемой. Таким образом, найден класс детерминированных контекстно-свободных языков с разрешимой проблемой их равенства.

Статья относится к теории схем программ - научному направлению в теоретическом программировании, восходящему к работам А.А. Ляпунова [1] и Ю.И. Янова [2]. В этой теории используется та или иная формализация понятия программы и центральной считается задача разработки эквивалентных преобразований формализованных программ. Такие преобразования составляют базу для решения задач оптимизации программ по различным их характеристикам. Преобразования программ разрабатываются на их схемах.

Статья состоит из пяти разделов.

В разделе 1 описывается выбранная нами формализация программы; она была предложена в [3]. Согласно ей, в программе используются простые типы данных и допускаются все средства композиции операторов, включая оператор процедуры. В случае, когда отсутствуют процедуры, программа совпадает с введенной в [1]. В программе операторы и булевы выражения лишены своей структуры и заменены операторными символами и логическими переменными соответственно. Те и другие составляют конечный базис, над которым строятся программы. В связи с изложенным они названы абстрактными.

Функциональная трактовка абстрактной программы основана на семантике базиса, приписывающей операторным символам отображения

некоторого множества в себя, а логическим переменным - бинарные отношения в этом множестве. Как само множество, так и отображения и отношения берутся произвольными. Переход от одной семантики базиса к другой равнозначен переходу от одного класса программ к другому.

В разделе 2 на конкретном примере демонстрируется связь реальной программы, записанной на языке Паскаль, с абстрактной программой. Здесь выявляется следующее обстоятельство: если используемые реальной программой переменные рассматривать как ее память, а выполнение программы воспринимать как преобразование состояний памяти в себя, то мы придем к некоторой абстрактной программе. Таким образом, при изложенном взгляде на реальную программу их множество "утопает"в множестве абстрактных программ.

Раздел 3, следуя установившейся в теории схем традиции, вводит понятие схемы программы с процедурами.

По своей структуре схема берется совпадающей с программой. Этим обеспечивается следующая важная позиция: всякое преобразование схемы одновременно будет и преобразованием соответствующей ей, т.е. совпадающей с ней по структуре, программы.

На множестве схем некоторым "волевым"актом вводится параметрическое семейство эквивалентностей схем. Каждая эквивалентность, по определению, превращает множество схем программ в модель программ.

Возникает естественная задача: отобрать модели с аппроксимирующими эквивалентность схем. Под аппроксимирующей понимается эквивалентность, для которой существует хотя бы одна такая семантика базиса, что из эквивалентности схем следует эквивалентность соответствующих им программ, рассматриваемых при данной семантике базиса. Обнаруженный класс программ называется аппроксимируемым, а сама модель - аппроксимирующей. Цель отбора аппроксимирующих моделей очевидна: в такой модели всякое эквивалентное преобразование схемы является и эквивалентным преобразованием соответствующей ей программы из аппроксимируемого класса.

В разделе 3 решается задача отбора аппроксимирующих моделей (теорема 1). Решение опирается на результат, полученный в [3], устраняя вместе с тем имеющиеся там неточности.

Среди отобранных имеется модель обладающая свойством: если схемы эквивалентны в этой модели, то они эквивалентны в любой модели вообще. Такая модель названа максимальной.

Последние два раздела статьи посвящены изучению максимальной модели. К этому побуждает имеющийся в теории схем прецедент. Дело в том, что максимальная модель программ с процедурами содержит в себе максимальную модель программ без процедур, известную в теории под

названием схем Янова. А схемы Янова исследовались в многочисленных работах. В их числе работы [2], [4] - [6], в которых рассматриваются именно те задачи, что интересуют нас здесь. Заметим, что в [7] установлено, что в максимальной модели программ с процедурами разрешима эквивалентность схем, т.е. существует алгоритм, который, получив на свой вход две произвольные схемы, определяет, эквивалентны они в максимальной модели или нет.

В разделе 4 получены два следующих результата.

Во-первых, показано, что если схемы, эквивалентные в максимальной модели, воспринимать как программы, то они будут эквивалентны при любой семантике базиса. Верно и обратное: если программы эквивалентны при любой семантике базиса, то, воспринимая их как схемы, мы имеем эквивалентные схемы в максимальной модели. То и другое установлено теоремой 2.

Во-вторых. Всякой схеме приписывается порожденный ею язык и доказывается, что схемы эквивалентны в максимальной модели тогда и только тогда, когда совпадают порожденные ими языки (теорема 3).

Прокомментируем эти результаты. Задача, рассмотренная в теореме 2, для схем программ без процедур была поставлена Л.А. Калужниным в [4] и впервые решена в [5], правда, весьма громоздкими средствами. Что касается теоремы 3, то аналогичная ей теорема для схем программ без процедур была доказана в [6] и неоднократно переизлагалась в последующих работах.

Раздел 5 обнаруживает связь между схемами программ с процедурами и конечными автоматами с магазином. Установлено, что для всякой схемы можно построить детерминированный автомат с магазином, принимающий язык, порожденный этой схемой (теорема 5). В частности, если рассматривается схема программы без процедур, то построенный автомат является обычным конечным. Таким образом, теоремой 5 поглощается результат, полученный в [6] для схем Янова. Следствием теоремы 5 является утверждение: язык, порожденный схемой программ с процедурами, является детерминированным контекстно-свободным. Таким образом, со ссылкой на теорему 3 и разрешимость эквивалентности в максимальной модели, мы получаем новый класс детерминированных контекстно-свободных языков с разрешимым равенством языков.

В заключение отметим, что статья содержит определения всех используемых в ней понятий, т.е. допускает чтение без обращения к цитируемой в ней литературе.

1 Абстрактные программы с процедурами

В данном разделе вводится понятие абстрактной программы с процедурами - описываются ее структура и функционирование. Формальные определения сопровождаются содержательными пояснениями.

Абстрактная программа с процедурами (далее - просто программа) строится над конечным базисом, состоящим из элементов четырех непустых и непересекающихся алфавитов - Y, C, R и P. Элементы первых трех называются символами, элементы множества P - логическими параметрами; каждая логическая переменная принимает значения из множества 0, 1. Символами обозначаются операторы, логическими переменными - булевы выражения.

Программа представляет собой размеченный над базисом конечный ориентированный граф следующего строения. Граф распадается на подграфы с непересекающимися множествами вершин. Один из подграфов называется главным; в нем выделены вход - вершина без входящих в нее дуг и одной исходящей, и выход - вершина без исходящих из нее дуг. В любом неглавном подграфе тоже выделены две вершины - инициальная и финальная. Всякая вершина, кроме входа, выхода и финальных, принадлежит одному из четырех типов - преобразователь, распознаватель, вызов, возврат. Из распознавателя исходят две дуги, помеченные числами 0 и 1 соответственно; из преобразователя, вызова, возврата - по одной дуге. Распознавателю сопоставлена переменная из P, преобразователю, вызову, возврату - символ из Y, C, R соответственно. Вызовы и возвраты находятся во взаимно-однозначном соответствии. Вызов и соответствующий ему возврат составляют пару, принадлежащую одному и тому же подграфу. Всякой такой паре присвоен номер, отличный от номеров других пар. Дуга из вызова ведет в инициальную вершину своего или чужого подграфа, и тогда из финальной вершины этого подграфа исходит дуга, ведущая в соответствующий вызову возврат. Иных дуг, кроме упомянутых, из финальной вершины не исходит. Дуги, исходящие из вершин иного типа, ведут в вершины того же подграфа, которому принадлежит их начало.

На рис. 1 представлена программа, состоящая из двух подграфов. Главный подграф находится слева. Вход, выход и финальная вершина даны кружочками, преобразователь - прямоугольником, распознаватель - овалом, вызов - пятиугольником, возврат - ромбом; сопоставленные вершинам элементы базиса вписаны в фигуры, изображающие вершины. Схеме принадлежат три пары соответствующих друг другу вызовов и возвратов - две в г+авном подграфе, одна - в неглавном.

Рис. 1.

Содержательно: программа состоит из ведущей программы, носителем которого является главный подграф, и процедур, носителями которых являются другие подграфы (в частном случае их может не быть). Ведущая программа с процедурами, а также последние друг с другом связаны дугами, исходящими из вызовов. Вход в процедуру осуществляется только через ее инициальную вершину. Число дуг, приходящих из вызовов в инициальную вершину, совпадает с числом дуг, исходящих из финальной вершины того же подграфа.

Функциональная трактовка программы основана на семантике базиса. Так называется алгебраическая система σ, использующая произвольно выбранное множество Ξ_σ и сопоставляющая каждому элементу b базиса всюду определенное отображение σb, тип которого определяется следующим образом:

$$\begin{array}{llll} \text{если} & b \in Y \bigcup C, & \text{то} & \sigma b : \Xi_\sigma \to \Xi_\sigma, \\ \text{если} & b \in R, & \text{то} & \sigma b : \Xi_\sigma \times \Xi_\sigma \to \Xi_\sigma, \\ \text{если} & b \in P, & \text{то} & \sigma b : \Xi_\sigma \to \{0,1\}. \end{array}$$

Элементы множества Ξ_σ называются состояниями памяти (без предъявления таковой). Множество всех семантик базиса обозначается Σ.

Опишем теперь процедуру выполнения программы на паре (σ, ξ_0), где $\sigma \in \Sigma, \xi_0 \in \Xi_\sigma$. Процедура использует магазин, заполняемый парами (n, ξ), где n - натуральное число, ξ - состояние из Ξ_σ, и состоит в путешествии по программе, маршрут которого определяется последовательностью проходимых дуг. Путешествие сопровождается преобразованием состояния памяти и содержимого магазина.

Путешествие по программе начинается по дуге, исходящей из ее входа, при состоянии памяти ξ_0 и пустом магазине и подчиняется следующим правилам. Пусть в некоторый момент времени путешествие вывело на дугу, ведущую в вершину v, и ξ - текущее состояние памяти. Различаем случаи:

1) v - преобразователь с символом y; тогда состояние памяти ξ преобразуется в состояние $\sigma y(\xi)$, содержимое магазина остается прежним, и путешествие продолжается по дуге, исходящей из v;

2) v - распознаватель с символом y; тогда состояние памяти ξ и содержимое магазина не изменяются, и путешествие продолжается по дуге, исходящей из v и помеченной значением $\sigma p(\xi)$;

3) v - вызов с символом c и номером n; тогда в магазин заносится пара (n, ξ), состояние памяти ξ преобразуется в состояние $\sigma c(\xi)$, и путешествие

продолжается по дуге, исходящей из v;

4) v - финальная вершина; в этом случае магазин заведомо не пуст; тогда из него снимается верхний элемент, пусть это - (n', ξ'), состояние памяти ξ не изменяется, путешествие продолжается по дуге, ведущей к возврату с номером n';

5) v - возврат с символом r, и перед приходом в него была снята из магазина пара (n', ξ'); тогда состояние памяти ξ преобразуется в состояние $\sigma r(\xi', \xi)$, и путешествие продолжается по дуге, исходящей из v;

6) v - выход; тогда путешествие заканчивается; в этом случае говорим, что программа остановилась на паре (σ, ξ_0) с заключительным состоянием ξ; заметим, что магазин опустел.

Процедура выполнения программы описана.

При заданной семантике σ программой реализуется частичное отображение множества Ξ_σ в себя: оно определено для ξ_0 из Ξ_σ тогда и только тогда, когда программа останавливается на паре (σ, ξ_0), и в этом случае состоянию ξ_0 сопоставляет заключительное состояние.

Программы G_1, G_2 назовем σ -эквивалентными (обозначим: $\overset{G_1\sigma}{\sim G_2}$) в том и только в том случае, если они осуществляют одно и то же отображение Ξ_σ в себя.

2 Связь реальных программ с абстрактными

Здесь на примере конкретной программы, обозначаем ее G_1^0, записанной на языке Паскаль, демонстрируется, как построить соответствующую ей программу абстрактную; она обозначается G_2^0. Построение включает описание семантики базиса, на которой рассматривается G_2^0. В этой семантике состояниями памяти являются наборы значений всех переменных, участвующих в G_1^0. Если работу программы G_1^0 оценивать как преобразование наборов исходных значений переменных в наборы их окончательных значений, то G_1^0 эквивалентна программе G_2^0. Построения, проведенные для программы G_1^0, можно применить к любой программе на языке Паскаль, использующей простые типы данных и процедуры, в общем случае - рекурсивные.

В качестве G_1^0 берется программа, которая для заданных натуральных чисел n_1, n_2 строит полусумму их факториалов. Будучи записана на языке Паскаль, эта программа имеет следующий вид:

program Example (output);
const $n_1 = 7; n_2 = 15$;
var u,s: integer;
procedure Fact (n: integer; var t: integer);

```
    begin
    if   n=0   then   t:=1
      else
        begin
          Fact (n-1, t);
          t := n * t
        end
    end;
begin
      Fact (n1, u);
      Fact (n2, s);
      s := (s + u) div 2;
      write (n1, n2, s)
end.
```

Здесь при вычислении факториала мы отдали предпочтение именно процедуре, а не функции, что соответствует духу абстрактных программ, конструируемых на базе операторов.

Программа G_2^0 представлена на рис. 1. При этом символы y_1, y_2, c_1, c_2 соответствуют операторам

$$\begin{array}{ll} y_1: & s := (s+u) \quad \underline{\text{div}}2 \\ y_2: & t := 1 \\ c_1: & n := n1 \\ c_2: & n := n2 \\ c_3: & n := n-1; \end{array}$$

логическая переменная p соответствует булеву выражению $n = 0$; символы r_1, r_2, r_3 обозначают действия по использованию результатов процедуры после Fact обращения к ней посредством вызовов c_1, c_2, c_3. В G_2^0 нашли отражение лишь разделы операторов, как ведущей программы из G_1^0, так и входящей в нее процедуры. Отсутствие оператора write(n_1, n_2, s) объясняется тем, что абстрактная программа не заботится ни о вводе начальных данных, ни о выводе полученных результатов.

Построение семантики σ_0 начнем с упорядочивания используемых в G_1^0 переменных: n_1, n_2, n, t, u, s. Набор их значений, т.е. шестерки целых чисел, примем за состояния памяти. Таким образом, множество Ξ_{σ_0} определено. Отображения, приписываемые в σ_0 символам y_1, y_2, c_1, c_2, c_3, совпадают с отображениями, которые осуществляются соответствующими им операторами присваивания. Например, отображение $\sigma_0 y_2$ шестерку $(d_1, d_2, d_3, d_4, d_5, d_6)$ из Ξ_{σ_0} переводит в шестерку $(d_1, d_2, d_3, 1, d_5, d_6)$.

Отображение, приписываемое p, строится так:

$$\sigma_0 p(d_1, d_2, d_3, d_4, d_5, d_6) = \begin{cases} 1, & \text{если} \quad d_3 = 0; \\ 0, & \text{если} \quad d_3 \neq 0; \end{cases}$$

здесь $(d_1, d_2, d_3, d_4, d_5, d_6)$ - произвольный элемент из Ξ_{σ_0}.

Остановимся на трактовке отображений, сопоставляемых символам r_1, r_2, r_3. Сначала рассмотрим, как строится $\sigma_0 r_3$. Исходим из того, что переменная n в G_1^0 является параметром-значением, т.е. в произведении $n * t$ значение n берется из состояния памяти, предшествующему вызову процедуры Fact, а значение t - из текущего состояния памяти. На этом основании $\sigma_0 r_3$ определяется так. Пусть

$$\begin{array}{ll} \xi' = (d'_1, d'_2, d'_3, d'_4, d'_5, d'_6); & \xi' \in \Xi_{\sigma_0}; \\ \xi'' = (d''_1, d''_2, d''_3, d''_4, d''_5, d''_6); & \xi'' \in \Xi_{\sigma_0}; \end{array}$$

тогда $\sigma_0 r_3(\xi', \xi'')$ - это набор

$$(d''_1, d''_2, d''_3, d'_3 * d''_4, d''_5, d''_6).$$

В случае символов r_1 и r_2 дело обстоит проще, так как в осуществляемых ими действиях не используется параметр n. Поэтому

$$\begin{array}{l} \sigma_0 r_1(\xi', \xi'') = \xi'', \\ \sigma_0 r_2(\xi', \xi'') = \xi''. \end{array}$$

Итак, семантика σ_0 построена. Легко проверить, что программа G_2^0 осуществляет на семантике σ_0 такое преобразование множества Ξ_{σ_0} в себя, которое совпадает с преобразованием, реализуемым программой G_1^0, если отречься от того, что по выходе из блока программы все локализованные в нем переменные утрачивают свои значения.

3 Абстрактные модели программ с процедурами

В этом разделе определяется схема абстрактной программы и на множестве схем вводится семейство эквивалентностей. Каждая эквивалентность превращает множество схем программ в абстрактную модель программ. Вводится понятие аппроксимирующей модели как такой, в которой из эквивалентности схем программ следует эквивалентность самих программ, рассматриваемых для некоторых семантик базиса. Осуществляется отбор аппроксимирующих моделей.

Схема программы строится над базисом $y \bigcup C \bigcup R \bigcup P$ и по своей структуре совпадает с программой.

Функционирование схемы осуществляется на функциях разметки. Введем понятие последней.

Слово в алфавите $Y \bigcup C \bigcup R$ называется цепочкой; цепочка считается правильной, если в любом ее префиксе число вхождений символов из R не превосходит числа вхождений символов из C. Обозначим H множество всех правильных цепочек. Пусть

$$X = \{x \mid x : P \to \{0, 1\}\}.$$

Функцией разметки назовем отображение $\mu : H \to X$.. Множество всех функций разметки обозначим $\mathcal{L}$.

Пусть $\mu \in \mathcal{L}$. Выполнением схемы на функции μ назовем процесс, состоящий в путешествии по схеме и сопровождающийся построением правильной цепочки. Для однозначности выбора пути, кроме μ, используется магазин, в который загружаются номера вызовов в схеме. Путешествие начинается по дуге, исходящей из входа, при пустых магазине и цепочке. Переход через вершину с сопоставленным ей символом сопровождается приписыванием этого символа к текущей цепочке справа. Если переходимая вершина - вызов, то в магазин загружается его номер. При переходе через финальную вершину и распознаватель текущая цепочка не изменяется. В первом случае из макушки магазина, который заведомо не пуст, извлекается номер, и путешествие продолжается по дуге, ведущей к возврату с этим номером. При переходе через распознаватель используется функция μ: в качестве следующей берется дуга, несущая метку $\mu h(p)$, где p - сопоставленная распознавателю переменная, а h - текущая цепочка. При достижении выхода путешествие прекращается; говорим, что схема остановилась на μ, и построенную цепочку называем результатом ее выполнения на μ. Путь, пройденный в схеме при ее выполнении на μ, именуем прокладываемым в схеме функцией μ.

Пусть ν - эквивалентность в множестве H, и $L \subseteq \mathcal{L}$. Схемы G_1, G_2 назовем (ν, L) - эквивалентными (обозначим: $G_1 \overset{\nu,L}{\sim} G_2$) тогда и только тогда, когда, какой бы ни была функция μ из L, всякий раз, как на ней останавливается одна из схем, останавливается и другая, и в этом случае результатами их выполнения являются ν - эквивалентные цепочки.

Множество всех схем над выбранным базисом, рассматриваемое вместе с (ν, L) - эквивалентностью схем, называется (ν, L)-моделью.

Говорим, что (ν, L)-модель - аппроксимирующая, если существует такая семантика базиса $\sigma, \sigma \in \Sigma$, , что для любых схем G_1, G_2 выполняется

$$G_1 \overset{\nu,L}{\sim} G_2 \Longrightarrow G_1 \overset{\sigma}{\sim} G_2.$$

Отбором аппроксимирующих моделей занимается теорема 1. Введем используемые в ней понятия.

(ν, L)- модель назовем строго аппроксимирующей, если существует такое непустое множество S семантик базиса, $S \subseteq \Sigma$, что

$$G_1 \overset{\nu,L}{\sim} G_2 \Longleftrightarrow \forall \sigma \in S \left(G_1 \overset{\sigma}{\sim} G_2 \right); \qquad (1)$$

здесь G_1, G_2 - произвольные схемы.

Определим полугрупповую модель программ.

Факт ν - эквивалентности цепочек h_1, h_2 из H будем записывать в виде $h_1 \overset{\nu}{\sim} h_2$. Функцию разметки $\mu, \mu \in \mathcal{L}$, назовем ν - согласованной, если для любых цепочек h_1, h_2 из H справедливо

$$h_1 \overset{\nu}{\sim} h_2 \longrightarrow \mu h_1 = \mu h_2.$$

Сдвигом функции μ, на цепочку $h, h \in H$, назовем функцию $\mu', \mu' \in \mathcal{L}$, определенную следующим образом: какой бы ни была цепочка h' из H, $\mu' h' = \mu h h'$; функцию μ' будем обозначать μ_h. Множество $L, L \in \mathcal{L}$, по определению, замкнуто по сдвигу, если для любых μ из L и любых h из H функция μ_h принадлежит L.

(ν, L)-модель назовем полугрупповой, если

1) эквивалентность ν обладает свойством: какими бы ни были цепочки h_1, h_2, h_3, h_4 из H и символ r из R,

а) $(h_1 \overset{\nu}{\sim} h_2 \& (h_3 \overset{\nu}{\sim} h_4) \longrightarrow (h_1 h_3 \overset{\nu}{\sim} h_2 h_4)$;

б) $(h_1 \overset{\nu}{\sim} h_2) \longrightarrow (h_1 r \in H \Longleftrightarrow h_2 r \in H) \& ((h_1 r, h_2 r \in H) \longrightarrow (h_1 r \overset{\nu}{\sim} h_2 r))$;

2) множество L состоит из ν - согласованных функций разметки и замкнуто по сдвигу.

Теорема 1 *Полугрупповая модель программ является строго аппроксимирующей.*

Доказательство. Пусть задана полугрупповая (ν, L)-модель. Построим для нее множество S семантик базиса такое, что выполняется(1).

Каждой функции μ из L сопоставим свою семантику, обозначаемую как σ_μ, и положим $S = \{\sigma_\mu \mid \mu \in L\}$. По определению, все семантики из S используют одно и то же множество Ξ и одни и те же отображения, приписываемые символам из $Y \bigcup C \bigcup R$, т.е. отличаются друг от друга только отображениями, сопоставленными символам из P. Определим σ_μ.

Класс всех цепочек из H, ν - эквивалентных цепочке $h, h \in H$, обозначим $\hat{h}$. Полагаем $\Xi = \{\hat{h} \mid h \in H\}$. Если b - символ из Y или C, то для любого $\hat{h}$ из Ξ

$$\sigma_\mu b(\hat{h}) = \widehat{hb};$$

если b - символ из R, то для любых $\hat{h}_1$, $\hat{h}_2$ из Ξ

$$\sigma_\mu b(\hat{h}_1, \hat{h}_2) = \begin{cases} \widehat{h_2 b}, & \text{если} \quad h_2 b \in H; \\ \hat{h}_2, & \text{если} \quad h_2 b \notin H. \end{cases}$$

Однозначность отображений $\sigma_\mu b$, где $b \in Y \bigcup C \bigcup R$, обеспечивается свойством 1а и 1б эквивалентности ν.

Всякому символу p из P сопоставим отображение

$$\sigma_\mu p(\hat{h}) = \mu h(p), \quad \hat{h} \in \Xi$$

Отметим, что корректность определения $\sigma_\mu p$ следует из ν - согласованности функции разметки μ.

Справедливо следующее утверждение: какой бы ни была схема, путь, который в ней прокладывается при ее выполнении на паре $(\sigma_\mu, \hat{h}), h \in \Xi$, совпадает с путем, прокладываемым в ней функцией разметки μ_h. И так как в силу замкнутости L функция μ_h принадлежит L, то из этого утверждения легко извлекается (1).

Теорема 1 доказана.

4 Максимальная модель программ

В этом разделе рассматривается одна из аппроксимирующих моделей программ; она названа максимальной. Разрешимость в ней эквивалентности схем следует из результатов, полученных в [7]. Свойства этой эквивалентности устанавливаются теоремами 2 и 3.

Назовем максимальной (ν_0, L_0)-модель, где ν_0 - равенство цепочек в H, а L_0 совпадает с $\mathcal{L}$. Максимальной она называется потому, что из эквивалентности схем в этой модели следует их эквивалентность в любой модели программ.

Легко проверить, что максимальная модель является полугрупповой, следовательно, по теореме 1, - аппроксимирующей. Покажем, что имеет место

Теорема 2 *Какими бы ни были схемы* G_1, G_2,

$$G_1 \overset{\nu_0, L_0}{\sim} G_2 \Longleftrightarrow \forall(\sigma \in \Sigma) G_1 \overset{\sigma}{\sim} G_2.$$

Доказательство. Справедливость утверждения

$$\forall(\sigma \in \Sigma) G_1 \overset{\sigma}{\sim} G_2 \Longrightarrow G_1 \overset{\nu_0, L_0}{\sim} G_2$$

вытекает из теоремы 1. Установим, что верно и обратное.

Пусть $\sigma \in \Sigma$; покажем, что $G_1 \overset{\nu_0, L_0}{\sim} G_2 \longrightarrow G_1 \overset{\sigma}{\sim} G_2$.

Каждой цепочке h из H сопоставим отображение σh множества Ξ_σ в себя. Пусть $h = b_1, \ldots b_k, k \geq 0$, где $b_i \in Y \bigcup C \bigcup R, i = 1, \ldots, k$. Полагаем:

1) если $k = 0$, то σh - тождественное отображение;

2) предположим, что $\sigma(b_1 \ldots b_{k-1})$ определено; тогда, если $b_k \in Y \bigcup C$, то

$$\sigma h = \sigma b_k(\sigma(b_1 \ldots b_{k-1}));$$

если же $b_k \in R$, то найдем в h соответствующий ему символ из C (соответствие между символами из C и символами из R в правильной цепочке подобно соответствию между открывающимися и закрывающимися скобками в префиксе алгебраического выражения); если это - b_l, то

$$\sigma h = \sigma b_k(\sigma(b_1 \ldots b_{l-1}), \sigma(b_1 \ldots b_{k-1})).$$

Паре (σ, ξ), где $\xi \in \Xi_\sigma$, сопоставим функцию разметки $\mu(\sigma, \xi)$, определив ее равенством

$$\mu(\sigma, \xi)h(p) = \sigma p(\sigma h \xi), \quad p \in P, h \in H.$$

Легко установить, что, какой бы ни была схема, выполнение ее на паре (σ, ξ) идет по пути, совпадающему с путем ее выполнения на функции $\mu(\sigma, \xi)$. Отсюда и из (ν_0, L_0) - эквивалентности G_1, G_2 следует, что G_1, G_2 либо обе завершают работу на паре (σ, ξ), либо обе не завершают ее, и в первом случае завершают с одинаковым результатом.

Теорема 2 доказана.

Теореме 3 предпошлем ряд понятий и построений.

Вершину схемы, имеющую тип: вход, преобразователь, вызов, возврат, назовем основной.

Пусть $x \in X$. Определим x-путь из основной вершины v как максимальный путь в схеме, начинающийся в v и удовлетворяющий требованию: всякая его внутренняя вершина - это распознаватель, и из него путь идет по дуге с меткой $x(p)$, где p - сопоставленная распознавателю переменная.

x - преемником основной вершины v назовем вершину, в которой завершается x-путь из v, если он конечен, и тот распознаватель, который впервые в нем повторяется, если x-путь бесконечен.

Рассмотрим путь w в схеме, начинающийся во входе и, если w конечен, то завершающийся либо в основной вершине, либо в выходе схемы. Представим w в виде

$$v_0 w_0 v_1 w_1 \dots v_k w_k \dots,$$

где v_0 - вход схемы; $v_1, \dots, v_k, \dots$ - все основные вершины, встречающиеся на пути w. Назовем w маршрутом, если

1) для любого $i, i = 0, 1, \dots$, существует такой x_i из X, что w_i является либо x_i-путем из v_i, либо представляет собой его продолжение на дугу из финальной вершины;

2) всякому входящему в w возврату v_j предшествует в w вызов v_{i_j}, парный с возвратом v_j и такой, что в пути w между v_{i_j} и v_j имеется равное число вхождений вызовов и возвратов.

Маршрут, завершающийся в выходе схемы, назовем маршрутом через схему. Пусть

$$w = v_0 w_0 v_1 w_1 \dots v_k w_k - \tag{2}$$

маршрут через схему. Предположим, что w_i - это x_i-путь из вершины $v_i, i = 0, 1, \dots, k$ или его продолжение на дугу из финальной вершины, и b_i - символ, сопоставленный вершине v_i. Введем дополнительный символ y_0, не принадлежащий базису. Слово

$$(y_0, x_0)(b_1, x_1) \dots (b_k, x_k) \tag{3}$$

в алфавите $Y \bigcup C \bigcup R \bigcup \{y_0\}$ назовем конфигурацией, порожденной маршрутом w. Отметим, что всякий маршрут через схему порождает конечное множество конфигураций, не обязательно одноэлементное, и что никакие два различных маршрута не порождают общую для них конфигурацию.

Схеме G сопоставим множество $K9G)$ конфигураций, порожденных всеми маршрутами через G.

Теорема 3 *Какими бы ни были схемы G_1, G_2,*

$$G_1 \overset{\nu_0, L_0}{\sim} G_2 \Longleftrightarrow K(G_1) K(G_2).$$

Доказательство. Пусть G_1, G_2 эквивалентны в максимальной модели, и $z \in K(G_1)$; покажем, что $z \in K(G_2)$. Предположим, что z имеет вид (3). Построим функцию разметки μ, определив ее следующим образом:

$$\mu h = \begin{cases} x_i, & \text{если} \quad h = b_1 \dots b_i, \quad i = 0, 1, \dots, k; \\ x^*, & \text{в противном случае;} \end{cases} \quad h \in H;$$

здесь x^* - какой-либо элемент из X. Очевидно, что μ прокладывает в G_1 маршрут, который порождает конфигурацию z, следовательно G_1, останавливается на μ с цепочкой $b_1 \dots b_k$. Но тогда схема G_2, будучи эквивалентной схеме G_1, останавливается на μ с тем же результатом. Отсюда следует, что μ прокладывает в G_2 маршрут, порождающий конфигурацию z, т.е. $z \in K(G_2)$.

Предположим теперь, что $K(G_1) = K(G_2)$, и докажем, что G_1, G_2 эквивалентны в максимальной модели. Рассмотрим функцию разметки μ из L_0, на которой останавливается одна из схем G_1, G_2, скажем, схема G_1. Тогда μ прокладывает в G_1 маршрут w_1. Если w_1 имеет вид (2), и b_i - символ, сопоставленный вершине $v_i, i = 1, \dots, k$, то при любом $i, i = 0, 1, \dots, k$, путь w_i - это $\mu b_1 \dots b_i$ - путь из v_i, возможно, дополненный дугой из финальной вершины. Используя обозначение

$$x_i = \mu b_1 \dots b_i, \quad i = 0, 1, \dots, k,$$

получим, что (3) - конфигурация, порожденная маршрутом w_1. Так как она принадлежит и множеству $K(G_2)$, то порождается некоторым маршрутом w_2 через G_2. Очевидно, что w_2 прокладывается функцией разметки μ. Отсюда вытекает, что G_2 останавливается на μ и с той же цепочкой, что и G_1.

Теорема 3 доказана.

В заключение скажем, что верна

Теорема 4 *В максимальной модели разрешима эквивалентность схем.*

Она доказана в [7].

5 Конечные автоматы с магазином и максимальная модель программ

Основным результатом этого раздела является теорема 5, устанавливающая, что по любой схеме программы можно построить детерминированный автомат с магазином (ДМП-автомат), принимающий множество конфигураций, порожденное этой схемой. Таким образом, последнее множество является контекстно - свободным языком. Со ссылкой на теорему 4 мы получаем класс контекстно - свободных языков с разрешимым в нем равенством языков.

Дадим определение ДМП-автомата как частного случая конечного автомата с магазином, введенного в [8].

ДМП-автомат состоит из трех частей - входной ленты, управляющего устройства и магазинной памяти. Входную ленту можно рассматривать как линейную последовательность ячеек, причем каждая ячейка содержит точно один входной символ из некоторого конечного входного алфавита. Входная головка в каждый момент времени обозревает одну ячейку. За один шаг работы автомата входная головка сдвигается на одну ячейку вправо. Входная головка только читает символы входной ленты.

Магазинную память можно описать как цепочку символов памяти, каждый из которых принадлежит конечному алфавиту памяти. Верхним элементом будем считать самый левый символ магазина.

Управляющее устройство представляет собой конечное множество состояний вместе с отображением, которое описывает, как меняются состояния при считывании очередного символа с входной ленты и одновременном извлечении верхнего символа из магазина. Параллельно с этим отображение определяет, какую информацию поместить в магазинную память.

Строгое определение ДМП-автомата выглядит так: он задается семеркой

$$U = (\Sigma, Q, \Gamma, \gamma, q_0, s_0, F),$$

где Σ - конечный входной алфавит; Q - конечное множество символов состояний; Γ - конечный алфавит магазинных символов; γ - отображение множества $Q \times (\Sigma \bigcup \{e\}) \times \Gamma$ в множество $Q \times \Gamma^*$, здесь e - пустое слово; q_0 - начальное состояние, $q_0 \in Q$; s_0 - начальный символ магазина, $s_0 \in \Gamma$; F - множество заключительных состояний, $F \subseteq Q$.

Опишем работу ДМП-автомата U. Такт работы U записывается в виде

$$(q, a\omega, s\alpha) \longrightarrow (q', \omega, \delta\alpha), \tag{4}$$

если $\delta(q, a, s) = (q', \delta)$, где $q, q' \in Q$, $s \in \Gamma$, $\alpha, \delta \in \Gamma^*$, $a\omega \in \Sigma^*$, $a = e$ только при $\omega = e$ и в остальных случаях $a \in \Sigma$.

Полагаем, что такт невозможен, если магазин пуст.

Запись (4) говорит о том, что автомат U, находясь в состоянии q, обозревая символ a на входной ленте и имея s в качестве верхнего символа магазина, переходит в состояние q', сдвигает входную головку на одну ячейку вправо и заменяет верхний символ магазина цепочкой δ магазинных символов. Если $\delta = e$, то последнее действие сводится к тому, что верхний символ просто удаляется из магазина.

Конфигурацией автомата U называется тройка

$$(q, \omega, \alpha) \in Q \times \Sigma^* \times \Gamma^*,$$

где q - текущее состояние управляющего устройства; ω - непрочитанная часть входной цепочки; α - текущее содержимое магазина. Если $\omega = e$, то считается, что вся входная лента прочитана.

Начальной конфигурацией автомата U называется конфигурация вида (q_0, ω, s_0), где $\omega \in \Sigma^*$. Она описывает состояние автомата перед началом работы, когда управляющее устройство находится в начальном состоянии, входная лента содержит цепочку, которую нужно распознать, а в магазине только начальный символ.

Заключительной конфигурацией называется конфигурация вида (q, e, α), где $q \in F$ и $\alpha \in \Gamma^*$.

Говорим, что автомат U принимает цепочку ω, если, находясь в начальной конфигурации (q_0, ω, s_0), он выполняет конечную последовательность тактов, заканчивающуюся заключительной конфигурацией. Множество цепочек, принимаемых автоматом U, обозначим $\kappa(U)$.

Теорема 5 *По любой схеме программы G можно построить ДМП-автомат $U(G)$ такой, что*

$$\kappa(U(G)) = K(G).$$

Доказательство Сначала опишем, как по схеме G строится автомат $U(G)$, а затем покажем, что он - требуемый.

Определим компоненты автомата.

Полагаем $\Sigma = (Y \bigcup C \bigcup R \bigcup \{y_0\}) \times X$.

Построим множество Q. Для этого сопоставим входу схемы G состояние q_0, остальным основным вершинам схемы G - состояния $q_1, \ldots, q_n$ выходу схемы G - состояние $\overline{q}$. Полагаем при этом, что никаким двум из перечисленных вершин не сопоставлено одно и то же состояние. В качестве Q возьмем множество $\{q_0, q_1, \ldots, q_n, \overline{q}, q^*\}$, где q^* - добавочное состояние.

Начальным считаем состояние q_0, единственным финальным - состояние $\overline{q}$.

Множество Γ берем состоящим из трех натуральных чисел, которыми в G нумеруются пары вызов-возврат, и специального символа s_0, играющего роль начального символа магазина.

Отображение γ определим следующим образом. Пусть

$$(q, a, s) \in Q \times (\sigma \bigcup \{e\}) \times \Gamma$$

и $(q', \delta) = \gamma(q, a, s)$. Тогда, если q - это $\overline{q}$ или q^*, то

$$q' = q^*, \quad \delta = s, \tag{5}$$

какими бы ни были a и s. Если $a = e$, то при любых q и s тоже выполняются равенства (5). Рассмотрим случай, когда $a \in \Sigma$, $a = (u, x)$; q - это образ основной вершины v схемы G, и b - символ, сопоставленный v, причем, если v - вход схемы, то полагаем, что $b = y_0$. В этом случае

$$q' = \begin{cases} \text{образ } x\text{-преемника вершины } v, \text{ если } u = b \text{ и } x\text{-преемником} \\ \text{не является финальная вершина;} \\ \text{образ возврата с номером } s, \text{ если } u = b \text{ и } x\text{-преемник -} \\ \text{это финальная вершина;} \\ q^* - \text{в остальных случаях;} \end{cases}$$

$$\delta = \begin{cases} s, \text{если } v \text{ - вход, преобразователь;} \\ s's, \text{если } v \text{ - вызов и } s' \text{ - его номер;} \\ e, \text{если } v \text{ - возврат.} \end{cases}$$

Автомат $U(G)$ построен.

Покажем, что $K(G) \subseteq \kappa(U(G))$. Пусть $z \in K(G)$, z порождена маршрутом, имеющим вид (2), и сама z имеет вид (3). Обозначим q^i состояние автомата $U(G)$, являющееся образом вершины $v_i, i = 0, \ldots, k$. Из построения $U(G)$ видно, что на цепочке (3) он пройдет последовательность состояний

$$q^0, q^1, q^2, \ldots, q^k, \tag{6}$$

где $q^0 = q_0$, после чего придет в состояние $\overline{q}$. Таким образом, $U(G)$ принимает цепочку (3).

Верно и отношение $\kappa(U(G)) \subseteq K(G)$. Действительно, пусть $U(G)$ принимает цепочку (3) и проходит при этом последовательность состояний (6), после чего приходит в состояние $\overline{q}$. Тогда, если v_i - прообраз состояния $q^i, i = 0, 1, \ldots, k$ то (2) - это маршрут через схему G, порождающий конфигурацию (3).

Теорема 5 доказана.

Следствие 1 теоремы 5. Какой бы ни была схема программы G, KG - детерминированный контекстно-свободный язык.

Справедливость этого утверждения следует из известного факта: язык, принимаемый ДМП-автоматом, является детерминированным контекстно-свободным.

Следствие 2 теоремы 5. В классе ДМП-автоматов $U(G)$, где G - произвольная схема программы над заданным базисом, разрешима эквивалентность автоматов.

Здесь мы опираемся на теорему 4.

Список литературы

[1] А.А. Ляпунов. О логических схемах программ // В сб. Проблемы кибернетики, вып. 1, Физматгиз, 1958, с. 46-74.

[2] Ю.И. Янов. О логических схемах алгоритмов // В сб. Проблемы кибернетики, вып. 1, Физматгиз, 1958, 75-127.

[3] Р.И. Подловченко. Рекурсивные программы и иерархия их моделей // Программирование, N 6, (1991), 44-51.

[4] Л.А. Калужин. Об алгоритмизации математических задач // В сб. Проблемы кибернетики, вып. 2, Физматгиз, 1962, 5-44.

[5] Р.И. Подловченко., Г.Н. Петросян, В.Е. Хачатрян. Интерпретации схем алгоритмов и различные типы отношений эквивалентности между схемами // Известия АН Арм.ССР, т. VII, N 2, (1972), 140-151.

[6] Y.D. Rutledge. On Ianov's Program Schemata // Y. ACM, 1964, v. 11, N 1, 1-19.

[7] Р.И. Подловченко. Специальные перегородчато-автоматные модели рекурсивных программ // Программирование, N 3, (1994), 3-26.

[8] А. Ахо, Дж. Ульман. Теория синтаксического анализа, перевода и компиляции // т. 1, изд-во Мир, 1978.