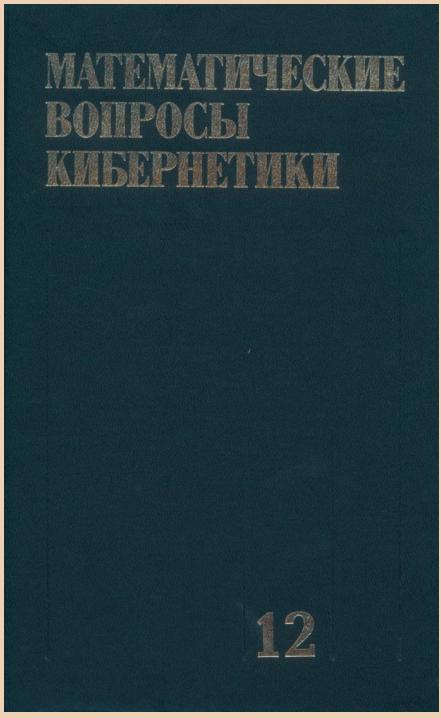
ИПМ им. М.В. Келдыша РАН • Электронная библиотека
Математические вопросы кибернетики • Выпуск 12



**Р. И. Подловченко,
Б. А. Долгих**

Двухступенчатое моделирование программ с процедурами

Рекомендуемая форма библиографической ссылки: Подловченко Р. И., Долгих Б. А. Двухступенчатое моделирование программ с процедурами // Математические вопросы кибернетики. Вып. 12. — М.: Физматлит, 2003. — С. 283–300. URL: http://library.keldysh.ru/mvk.asp?id=2003-283

ДВУХСТУПЕНЧАТОЕ МОДЕЛИРОВАНИЕ ПРОГРАММ С ПРОЦЕДУРАМИ *)

Р. И. ПОДЛОВЧЕНКО, Б. А. ДОЛГИХ

(МОСКВА)

Статья относится к теории моделей программ — одному из разделов теоретического программирования.

Исходными в этой теории являются выбор той или иной формализации программы и задача построения эквивалентных преобразований (э.п.) программ. В разработанной части теории (см. [2]) в качестве формализации программы берется ее представление на алгоритмическом языке программирования высокого уровня, и все построения следуют плану:

1. формализованная программа преобразуется в форму, когда в явном виде задается ее управляющий граф (назовем эту форму графовой формализацией программы);
2. осуществляется переход от программы к схеме, при котором сохраняется управляющий граф программы;
3. в множестве схем программ вводятся отношения эквивалентности, составляющие параметрическое семейство; рассмотрение отдельного отношения эквивалентности квалифицируется как выбор конкретной модели программ (она называется алгебраической);
4. принимается решение исследовать только такие модели, которые обладают свойством: *из эквивалентности схем в модели следует эквивалентность программ, для которых построены схемы* (такие модели и присущие им эквивалентности схем называются аппроксимирующими); это решение гарантирует автоматическое использование э.п. схем в качестве э.п. программ;
5. для аппроксимирующих моделей главной считается задача построения системы э.п. схем, полной либо во всей модели, либо в некотором принадлежащем ей классе схем; проблематика теории складывается вокруг этой главной задачи.

Этот план осуществлен в случае, когда при построении программы на алгоритмическом языке используются все средства композиции базовых операторов и базовых логических условий, кроме аппарата процедур.

Вместе с тем, в теории рассматриваются так называемые алгебраические модели программ с процедурами. Они определены в [1] и интуитивно

*) Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 03–01–00312).

ориентированы на случай, когда в программе, записанной на алгоритмическом языке, используются и процедуры. Однако для них отсутствует исходная графовая формализация; следовательно, открыт вопрос: аппроксимируют ли они классы реальных программ с процедурами?

Данная статья ликвидирует этот пробел в теории. В ней предлагается графовая формализация программы с процедурами и решается задача аппроксимации классов таких программ введенными в [1] алгебраическими моделями программ с процедурами.

Перейдем к описанию материала статьи.

Графовая формализация программы с процедурами рассматривается нами как обобщение графовой формализации программы без процедур. В связи с этим в § 1 напоминается, как осуществляется последняя (описание берется из [3]), после чего проводится анализ проблем, возникающих при требуемом ее обобщении. В этих целях описывается, каким образом в реальной программе действует аппарат процедур, и на этой основе принимаются соглашения, сопровождающие решение двух задач: а) заменить практикуемое аппаратом процедур копирование тела процедуры запоминанием состояния памяти в момент обращения к процедуре; б) запомнить место в программе, где происходит обращение к процедуре с тем, чтобы вернуться туда по ее выполнении.

Сама графовая формализация программы с процедурами осуществляется в § 2.

При отыскании алгебраических моделей программ с процедурами, аппроксимирующих классы введенных нами программ с процедурами, мы следуем пути, проложенному в [3] для программ без процедур. Там при решении этой задачи в качестве промежуточной ступени использовались стандартные схемы программ; их определение имеется в [3]. Они удобны тем, что эквивалентность таких схем вводится сразу как аппроксимирующая эквивалентность самих программ.

В случае программы с процедурами потребовалось создать объект, выполняющий функции промежуточной ступени. Это сделано в § 3 под названием обобщенной стандартной схемы. Эквивалентность этих схем тоже аппроксимирует эквивалентность программ с процедурами. Корректность термина «обобщенные стандартные схемы» подтверждается тем, что для них доказана теорема 1 — аналог известной теоремы о свободных интерпретациях, справедливой для обычных стандартных схем.

Наконец, § 4 посвящен второй ступени моделирования. В нем воспроизводится определение алгебраической модели программ, данное в [1]. Схемы, являющиеся объектами этой модели, называются алгебраическими. Затем по обобщенным стандартным схемам строится класс алгебраических схем. В этом классе вводится такое отношение эквивалентности схем, что последние эквивалентны тогда и только тогда, когда эквивалентны породившие их стандартные схемы (теорема 2). Отсюда следует, что построенный класс аппроксимирует исходные программы с процедурами, а содержащая этот класс алгебраическая модель программ с процедурами является аппроксимирующей.

В заключении используется достаточный признак аппроксимации одной алгебраической модели другой алгебраической моделью, описанный в [2]. При наличии уже построенной аппроксимирующей модели этот признак дает семейство аппроксимирующих моделей.

Отметим, что определенные нами обобщенные стандартные схемы оказались родственными конструкциям, введенным в [4] и активно используемым в исследовании различных функциональных свойств реальных программ с процедурами.

§ 1. О соглашениях при формализации программ с процедурами

Запланированная нами графовая формализация программ с процедурами является обобщением графовой формализации программ без процедур, выполненной, например, в [3]. В связи с этим напомним, как проводится последняя.

Условимся в дальнейшем, краткости ради, программы, записанные на алгоритмическом языке высокого уровня, называть *реальными*, а графовые их формализации — просто *программами*, в одном случае — *без процедур*, а в другом — *с процедурами*.

Рассматриваемые нами реальные программы строятся над конечным базисом Σ операторов присваивания и булевых выражений. В тех и других используются переменные одного и того же типа. Обозначим D область их значений, и пусть R — множество всех используемых переменных, называемое *памятью*.

Базисный оператор присваивания — это конструкция вида

$$r_0 := \mathrm{op}(r_1, \ldots, r_k) \tag{1}$$

где op — k-местная всюду определенная операция в D, т. е. отображение вида: $D^k \to D$, а $r_0, \ldots, r_k$ — переменные из R. Не исключается случай, когда op — одноместная тождественная операция, т. е. (1) имеет вид

$$r_0 := r_1 \tag{1'}$$

и называется *оператором пересылки*.

Базисное булево выражение — это конструкция вида

$$\mathrm{rel}(r_1, \ldots, r_l) \tag{2}$$

где rel — l-местное всюду определенное отношение в D, т. е. отображение вида $D^l \to \{0, 1\}$, а $r_1, \ldots, r_l$ — переменные из R.

Программа без процедур представляет собой конечный ориентированный граф с размеченными вершинами и дугами. Структура графа такова. В нем выделены две вершины: *вход* — вершина без заходящих в нее дуг и с единственной исходящей дугой, и *выход* — вершина без исходящих из нее дуг. Остальные вершины графа, если они имеются, по своему типу делятся на *преобразователи* и *распознаватели*. Из каждого преобразователя исходит одна дуга, из распознавателя — две. Разметка вершин подчиняется правилу: всякому преобразователю сопоставляется оператор из Σ, всякому распознавателю — булево выражение из Σ. Дуги, исходящие из распознавателя, помечаются символами 0 и 1 соответственно, другие дуги меток не имеют.

Всякой программе без процедур (далее она именуется просто программой) сопоставляется реализуемое ею отображение Ξ в себя, где

$$\Xi = \{\xi \mid \xi : R \to D\};$$

элементы множества Ξ называются *состояниями памяти* R (обычно без упоминания самого R). Реализуемое программой отображение является частичным и определяется посредством процедуры ее выполнения на некотором состоянии памяти ξ_0, рассматриваемом как исходное.

При описании этой процедуры со всяким базисным оператором ассоциируется всюду определенное отображение Ξ в себя, а со всяким базисным булевым выражением — всюду определенное отображение Ξ в множество $\{0, 1\}$. Определим их.

Пусть A — оператор вида (1); результат применения A к состоянию ξ, где $\xi \in \Xi$, обозначим ξ'. Состояние ξ' описывается так: пусть $r \in R$, тогда

$$\xi'(r) = \begin{cases} \xi(r), & \text{если } r \neq r_0, \\ \mathrm{op}(\xi(r_1), \ldots, \xi(r_k)), & \text{если } r = r_0. \end{cases} \tag{3}$$

Пусть B — булево выражение вида (2), тогда для $\xi \in \Xi$

$$B(\xi) = \mathrm{rel}(\xi(r_1), \ldots, \xi(r_l)). \tag{4}$$

Процедура выполнения программы на состоянии ξ_0, где $\xi_0 \in \Xi$, состоит в путешествии по ней, которое начинается в ее входе с начальным состоянием памяти ξ_0 и проходит в направлении дуг программы. При прохождении через преобразователь, которому сопоставлен оператор A вида (1), текущее состояние памяти ξ преобразуется согласно (3), и путешествие продолжается по единственной дуге, исходящей из преобразователя. При прохождении через распознаватель, которому сопоставлено булево выражение B вида (2), текущее состояние памяти ξ не изменяется, путешествие продолжается по дуге, помеченной $B(\xi)$. В случае достижения выхода путешествие прекращается, и мы говорим, что программа *остановилась при выполнении ее на* ξ_0, а состояние ξ объявляем *результатом* этого выполнения.

Таким образом, программа реализует частичное отображение Ξ в себя. Именно оно и объявляется *функцией, реализуемой программой.* Две программы, по определению, *эквивалентны,* если они реализуют равные функции.

Заметим, что из функционального описания программы видно, что в ней применяются как оператор безусловного перехода (go to), так и композиции типа: условный оператор, составной оператор, оператор цикла (с предусловием и с постусловием).

Предположим теперь, что реальные программы, подлежащие графовой формализации, используют процедуры. По-прежнему будем считать, что они строятся над конечным базисом операторов присваивания и булевых выражений, имеющих вид (1), (1′) и (2) и реализующих отображения Ξ в себя и Ξ в $\{0, 1\}$.

Исходим из того, что в реальной программе обращение к процедуре влечет за собой следующие действия: копируется тело процедуры, копия обрабатывается по определенным правилам и выполняется в пункте обращения к процедуре. Правила обработки тела процедуры опираются на классификацию переменных, используемых в процедуре.

Переменные классифицируются на

- формальные параметры, подставляемые значениями фактических параметров (те и другие будем называть параметрами-значениями),
- формальные параметры, подставляемые наименованиями фактических параметров (те и другие будем называть параметрами-наименованиями),
- переменные, локализуемые в теле процедуры,
- переменные, глобальные по отношению к процедуре.

Мы будем рассматривать тот частный случай, когда фактическим параметром-значением является просто переменная.

Сама обработка тела процедуры проходит по правилам:

- формальные параметры-наименования замещаются соответствующими им фактическими параметрами-наименованиями;
- формальные параметры-значения локализуются (т. е. им выделяется место в памяти), после чего им присваиваются значения фактических параметров.

По выполнении тела процедуры все локализованные в ней переменные теряют свои значения, так как у этих переменных отбирается хранившая их память.

Наша задача заключается в том, чтобы заменять копирование тела процедуры запоминанием состояния памяти в момент обращения к процедуре и чтобы запоминать место, где происходит обращение, с тем, чтобы вернуться туда по выполнении процедуры. Кроме того, должна быть учтена специфика обработки тела процедуры.

Решение этой задачи осуществляется структурными и функциональными соглашениями.

Они состоят в следующем.

1. Наряду с памятью R рассматривать ее дубликат R', называемый *регистром.*
2. Тело процедуры представлять конечным ориентированным графом подобно тому, как это делается для программы без процедур; вход в этот граф выделять под именем инициальной вершины; выход из него — под именем финальной вершины. Ассоциировать с этим графом три списка переменных, а именно: формальных параметров-значений, формальных параметров-наименований и переменных, локализуемых в данной процедуре.
3. Обращение к процедуре осуществлять посредством специальной пары вершин, называемых вызовом и возвратом; всякую такую пару идентифицировать своим номером, отличным от номеров других пар и воспринимаемым и как номер вызова, и как номер возврата.

   Полагать, что

   а) из вызова исходит единственная дуга, ведущая в инициальную вершину графа, задающего вызываемую процедуру; с вызовом ассоциировать две группы пересылок, передающих значения фактических параметров соответствующим им формальным параметрам;

   б) из финальной вершины вызываемой процедуры исходит дуга, ведущая в возврат, парный вызову; с возвратом ассоциировать три группы пересылок: первая передает значения формальных параметров-наименований соответствующим им фактическим параметрам; вторая «восстанавливает» значения формальных параметров-значений, беря их у расположенных в R' дублеров фактических параметров, соответствующих этим формальным параметрам; третья «восстанавливает» значения переменных, локализованных в теле процедуры, беря их из R' у дублеров этих переменных.
4. При функционировании программы использовать магазин, загружаемый парами вида ⟨состояние памяти R, номер вызова⟩. Выполнение вызова трактовать как загрузку магазина парой ⟨текущее состояние памяти R, номер этого вызова⟩ с последующей реализацией ассоциируемых с вызовом пересылок; выполнение финальной вершины трактовать как выгрузку из магазина верхней содержащейся в нем пары; при этом состояние памяти, принадлежащее паре, помещать в регистр R', а номер вызова использовать для перехода к возврату с этим же номером; выполнение возврата трактовать как реализацию всех трех групп ассоциируемых с ним пересылок.

Отметим, что действия, приписываемые возврату, продиктованы тем, что

- в теле процедуры формальные параметры-наименования «работают» вместо фактических параметров-наименований,
- локализуемые в процедуре переменные могут быть локализуемыми и в процедуре, охватывающей данную,
- если процедура рекурсивная, то необходимо восстанавливать значения, несомые параметрами-значениями.

§ 2. Графовая формализация программ с процедурами

На основе изложенных соглашений определим программу с процедурами в графовой форме.

Полагаем заданными память R и базис Σ. Введем множество R' такое, что между элементами R и R' имеется взаимно однозначное соответствие; назовем его *регистром.*

Наряду с пересылками типа (1′), где $r_0, r_1 \in R$, будем рассматривать пересылки типа

$$z := z', \tag{1″}$$

где $z \in R$, $z' \in R'$ и z' является образом переменной z.

Обозначим $\overline{\Sigma}$ расширение базиса Σ путем добавления отсутствующих в нем пересылок типа (1′) и всевозможных пересылок типа (1″).

Введем конструкции, называемые групповыми пересылками. Каждая из них представляет собой последовательность

$$z_1 := u_1; \ \ldots; z_k := u_k, \quad k \geqslant 1, \tag{5}$$

состоящую из пересылок, принадлежащих базису $\overline{\Sigma}$, и удовлетворяющую требованиям:

a) $i \neq j \longrightarrow z_i \neq z_j$;

b) все пересылки в (5) имеют общий тип.

Если общим является тип (1′), то групповая пересылка (5) называется *внутренней* и обозначается $[Z \longleftarrow U]$, где $Z = (z_1, \ldots, z_k)$, $U = (u_1, \ldots, u_k)$. В противном случае (5) называется *внешней* и обозначается $[Z \longleftarrow Z']$, где Z' является образом упорядоченного множества Z в отображении R в R'.

Если для внутренней пересылки (5) выполнено требование

$$i \neq j \longrightarrow u_i \neq u_j,$$

то обратной к ней, по определению, является групповая пересылка

$$u_1 := z_1; \ \ldots; \ u_k := z_k,$$

которая обозначается $[U \longleftarrow Z]$.

Программа с процедурами строится над базисом $\overline{\Sigma}$. Она представляет собой конечный ориентированный граф с размеченными вершинами и дугами. Граф распадается на подграфы с непересекающимися множествами вершин. Один из подграфов называется *главным*; в нем выделены две вершины: вход — вершина без заходящих в нее дуг и с 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, другие дуги пометок не имеют. Из преобразователя, вызова и возврата исходит по одной дуге. Вызовы и возвраты находятся во взаимно однозначном соответствии. Вызов и соответствующий ему возврат составляют пару, принадлежащую одному и тому же подграфу. Такая пара связана с некоторым неглавным подграфом следующим образом. Дуга из вызова ведет в инициальную вершину этого подграфа, а из его финальной вершины исходит дуга, ведущая в соответствующий вызову возврат, и это — единственная заходящая в него дуга. Из финальной вершины иных дуг, кроме описанных, т. е. ведущих в возвраты, не исходит. Пара вершин вызов-возврат и связанный с ним подграф называются *инцидентными* друг другу. Всякой паре вершин вызов-возврат присвоен номер, отличный от номеров других пар. Дуги, исходящие из вершин, отличных от вызова, ведут в вершины того же

подграфа, которому принадлежит их начало. Каждому неглавному подграфу сопоставляется кортеж $\langle Z_1, Z_2, Z_3\rangle$ непересекающихся упорядоченных множеств, где $Z_i \subseteq R$, $i = 1, 2, 3$. Любое из этих множеств может быть пустым. Переменные из Z_1 называются формальными параметрами-значениями, переменные из Z_2 — формальными параметрами-переменными, переменные из Z_3 — локализуемыми в данном подграфе переменными.

Разметка вершин графа подчиняется следующим правилам.

Пусть V — обсуждаемая вершина. Если V — преобразователь или распознаватель, то ей сопоставляется оператор или соответственно булево выражение из $\overline{\Sigma}$. Пусть V — вызов, и инцидентный этому вызову подграф специфицируется кортежем $\langle Z_1, Z_2, Z_3\rangle$. Тогда V сопоставляется кортеж

$$\langle [Z_1 \longleftarrow U_1], [Z_2 \longleftarrow U_2]\rangle \tag{6}$$

внутренних групповых пересылок, причем $[Z_2 \longleftarrow U_2]$ — обратимая. Переменные из U_1 — это фактические параметры-значения; переменные из U_2 — фактические параметры-переменные.

Пусть V — возврат, инцидентный подграфу с сопоставленным ему кортежем $\langle Z_1, Z_2, Z_3\rangle$ и парный вызову, которому сопоставлен кортеж (6). Тогда вершине V сопоставляется кортеж

$$\langle [Z_1 \longleftarrow Z_1'], [U_2 \longleftarrow Z_2], [Z_3 \longleftarrow Z_3']\rangle, \tag{7}$$

где $[Z_1 \longleftarrow Z_1']$ и $[Z_3 \longleftarrow Z_3']$ — внешние групповые пересылки, а $[U_2 \longleftarrow Z_2]$ — обратная к $[Z_2 \longleftarrow U_2]$.

Структура программы с процедурами описана.

Приведем пример того, как по реальной программе с процедурами строится соответствующая ей графовая форма.

Возьмем программу, записанную на языке Паскаль и вычисляющую полусумму факториалов, построенных для заданных значений переменных $n1$ и $n2$, если эти значения неотрицательны.

```
program Example (input, output);
    var n1, n2, 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
    read (n1,n2);
        if (n1>=0) ∧ (n2>=0) then
                                 begin
                                     Fact (n1,u);
                                     Fact (n2,s);
                                     s := (s+u) div 2
                                 end
                             else s := -1;
    write (n1,n2,s)
end.
```

При переходе к графовой форме опускаются описания переменных, ввод начальных данных и вывод результатов. Учитываемые части программы заключены нами в прямоугольники.

Графовая форма программы представлена в двух вариантах на рис. 1 и рис. 2. На них преобразователи, распознаватели, вызовы и возвраты изображены соответственно вытянутыми прямоугольниками, овалами, вытянутыми пятиугольниками и ромбами; внутрь фигур вписаны сопоставленные им конструкции; каждая пара вызов-возврат снабжена своим номером; подграфу, построенному для процедуры Fact, сопоставлен кортеж $\langle\{n\},\{t\},\varnothing\rangle$.

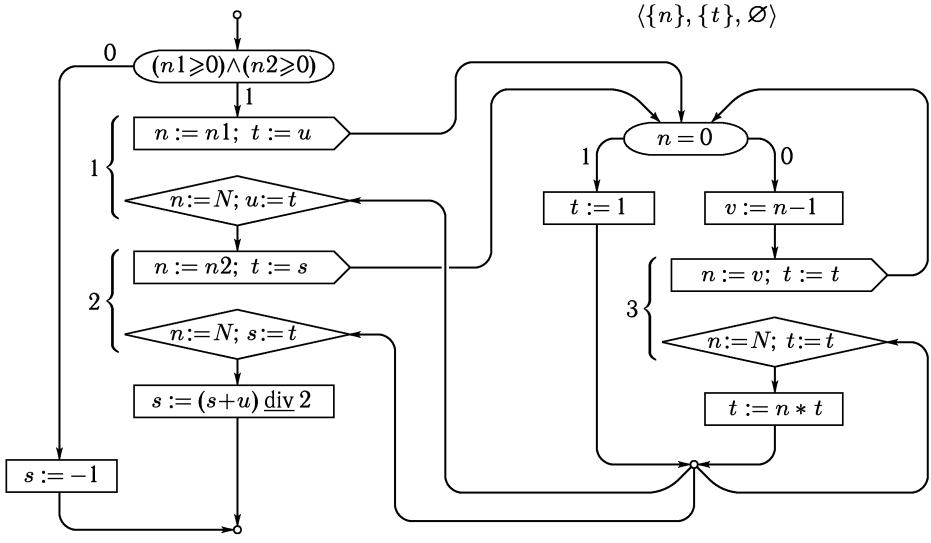


Рис. 1

Рис. 1 отражает пунктуальное следование соглашениям, изложенным выше. Символ N обозначает регистровый двойник переменной n. Оператор

$$v := n - 1$$

введен в связи с тем, что, по соглашению, фактическим параметром-значением может быть только переменная.

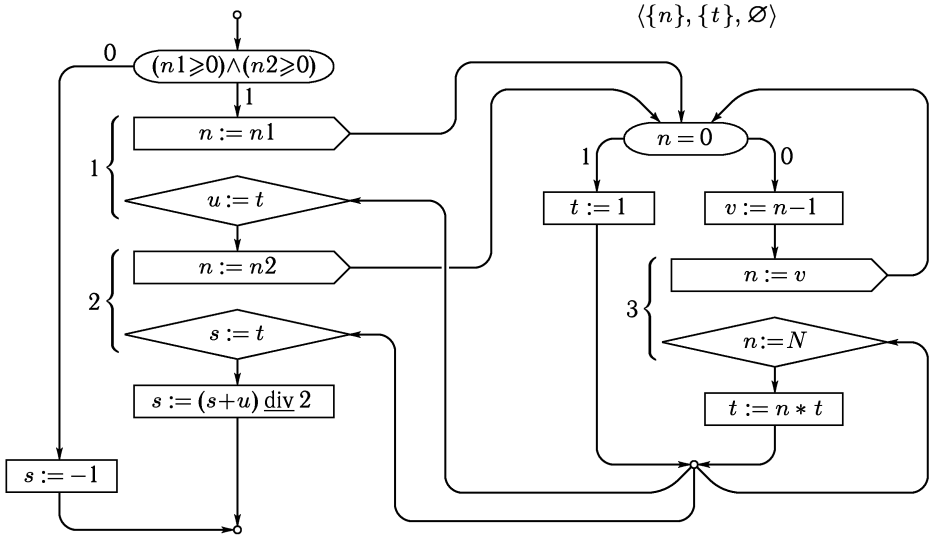


Рис. 2

Рис. 2 фиксирует окончательный вариант программы. Нами опущены пересылки $t := t$, что не требует пояснений; кроме того, опущены «лиш-

ние» пересылки, т. е. не влияющие на вычисление реализуемой программой функции. Заметим, что можно строго сформулировать, какие из пересылок являются «лишними», но это не входит в задачу данной статьи.

Наряду с множеством

$$\Xi = \{\xi \mid \xi\colon R \to D\},$$

называемым множеством состояний памяти R, введем множество

$$\overline{\Xi} = \{\overline{\xi} \mid \overline{\xi}\colon R' \to D\};$$

его элементы будем называть состояниями регистра R'. Здесь D — используемая в § 1 область значений переменных из R.

Функционирование программы с процедурами определяется на паре состояний $(\overline{\xi}_0, \xi_0)$, где $\overline{\xi}_0 \in \overline{\Xi}$, $\xi_0 \in \Xi$. При этом используется магазин, хранящий пары (ξ, i), где ξ — состояние памяти R, i — номер пары вызов-возврат. В начальный момент магазин пуст. Функционирование программы — это путешествие по графу, начинающееся во входе с состоянием памяти ξ_0 и состоянием регистра $\overline{\xi}_0$, проходящее в направлении дуг и сопровождающееся преобразованием текущих состояний памяти R и регистра R', а также преобразованием содержимого магазина. Пусть V — вершина, в которую мы пришли при путешествии, $\overline{\xi}$ — текущее состояние R', ξ — текущее состояние R; магазин хранит текущую информацию. Если V — преобразователь, и ему сопоставлен оператор вида (1) (в частном случае вида (1′)), то при переходе через V состояние $\overline{\xi}$ и содержимое магазина остаются неизменными, а состояние ξ преобразуется в состояние ξ' согласно равенству (3). Если V — распознаватель, которому сопоставлено булево выражение B вида (2), то при переходе через V состояния $\overline{\xi}$, ξ, а также содержимое магазина остаются неизменными, и путешествие продолжается по дуге, исходящей из V и помеченной числом $B(\xi)$, определяемым равенством (4).

Выполнение вызовов, возвратов и финальных вершин основано на функциональных и структурных соглашениях, изложенных в § 1, и состоит в следующем. Пусть V — вызов, которому сопоставлен кортеж (6); тогда при переходе через V в магазин загружается пара (ξ, i), где i — номер этого вызова, состояние $\overline{\xi}$ не меняется, а состояние ξ преобразуется в состояние ξ' согласно выполнению групповых пересылок из (6); путешествие продолжается по единственной дуге, ведущей из вызова в инициальную вершину инцидентного ему подграфа. Пусть V — финальная вершина подграфа; тогда при переходе через V снимается верхняя пара из магазина; если это — пара $(\widetilde{\xi}, i)$, то состояние $\widetilde{\xi}$ помещается в регистр R', а именно: то, что в $\widetilde{\xi}$ является содержимым переменной из R становится содержимым ее образа в R'; это будет новым состоянием регистра R'; текущее состояние памяти ξ не меняется; путешествие продолжается по дуге, ведущей к возврату с номером i. Пусть V — возврат, которому сопоставлен кортеж (7). Тогда состояние $\overline{\xi}$ и состояние магазина остаются неизменными, а новое состояние памяти R формируется из состояний $\overline{\xi}$, ξ путем выполнения групповых пересылок из (7). Наконец, если V — выход программы, то путешествие по ней прекращается. В этом случае говорим, что программа остановилась на заданной паре $\overline{\xi}_0$, ξ_0, и состояния $\overline{\xi}$, ξ называем *заключительными*.

Обратимся к программе, изображенной на рис. 2. Легко проверить, что она вычисляет в переменной s значение $((n1)! + (n2)!) \underline{\operatorname{div}} 2$, если $n1 \geqslant 0$ и $n2 \geqslant 0$, и значение -1, если это условие не выполняется.

Пусть $V_0, V_1, \ldots$ — последовательность вершин программы, составляющая маршрут путешествия по ней на паре состояний $(\overline{\xi}_0, \xi_0)$.

Обозначим ξ_i состояние памяти R при подходе к вершине V_i, $i=0, 1, \ldots$. Последовательность пар

$$(\xi_0, V_0)(\xi_1, V_1)\ldots(\xi_i, V_i)\ldots$$

именуем *историей выполнения программы на паре состояний* $(\overline{\xi}_0, \xi_0)$.

Из описания функционирования программы с процедурами следует

Утверждение 1. *История выполнения программы с процедурами на паре состояний* $(\overline{\xi}_0, \xi_0)$ *не зависит от выбора* $\overline{\xi}_0$.

Опираясь на утверждение 1, примем следующее соглашение: исходное состояние регистра $\overline{\xi}_0$ всегда брать таким, что при любом r' из R'

$$\overline{\xi}_0(r') = \xi_0(r), \quad r' \in R',$$

где ξ_0 — исходное состояние памяти, а r — прообраз переменной r' при отображении R в R'.

Данное соглашение позволяет в дальнейшем

- говорить о выполнении программы с процедурами на состоянии памяти ξ_0 (вместо пары $\overline{\xi}_0, \xi_0$),
- считать результатом выполнения заключительное состояние памяти (вместо заключительных состояний регистра и памяти).

Основываясь на этом, мы будем ассоциировать с программой отображение множества Ξ в себя, называя его *функцией, реализуемой программой.*

Две программы назовем *эквивалентными,* если реализуемые ими функции совпадают.

Множество всех программ с процедурами, построенных над базисом $\overline{\Sigma}$, обозначим K.

§ 3. Обобщенные стандартные схемы программ с процедурами

Здесь мы определим новую модель вычислений, объектами которой являются обобщенные стандартные схемы программ с процедурами (о.стандартные схемы), введем отношение эквивалентности в множестве таких схем и докажем теорему 1 о фундаментальном свойстве этого отношения эквивалентности.

О.стандартная схема строится по описанной в § 2 формализованной программе с процедурами. Она строится на том же графе, что и программа, и отличается от нее только разметкой преобразователей и распознавателей. Последняя осуществляется следующим образом:

1. всякий оператор присваивания вида (1), не совпадающий с (1′), заменяется синтаксической конструкцией

$$r_0 := f^{(k)}(r_1, \ldots, r_k), \quad k \geqslant 0, \tag{8}$$

называемой *оператором над памятью* R; здесь $f^{(k)}$ — функциональный символ арности k;

2. всякое выражение вида (2) заменяется синтаксической конструкцией

$$\pi^{(l)}(r_1, \ldots, r_l), \quad l \geqslant 0, \tag{9}$$

называемой *предикатом над памятью* R; здесь $\pi^{(l)}$ — предикатный символ арности l.

При такой замене соблюдается требование: разным операциям вида op соответствуют разные функциональные символы, разным отношениям вида rel — разные предикатные символы.

Структура о.стандартной схемы описана.

На рис. 3 представлена о.стандартная схема, построенная для программы с рис. 2.

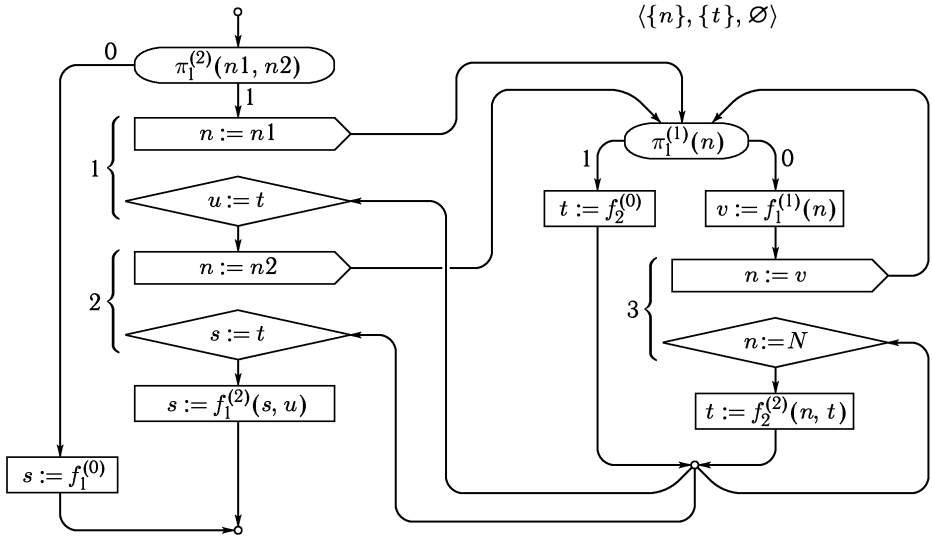


Рис. 3

Совокупность предложенных конструкций вида (8), (9), а также операторы пересылок типа (1′), (1″), принадлежащие базису $\overline{\Sigma}$, образуют базис о.стандартной схемы, который обозначим S. Описанная выше биекция $\overline{\Sigma}$ в S, в которой пересылки типа (1′), (1″) неподвижны, индуцирует отображение множества K программ в множество K_1 о.стандартных схем.

Обозначим F и Π совокупности функциональных и соответственно предикатных символов, используемых в базисе S.

Функционирование о.стандартной схемы из K_1 определяется посредством интерпретации множеств R, F, Π. По определению, *интерпретация* I — это комплекс, состоящий из множества D_I элементов произвольной природы, всюду определенных операций

$$(If^{(k)})\colon {D_I}^k \to D_I, \quad f^{(k)} \in F,$$

и всюду определенных отношений

$$(I\pi^{(l)})\colon {D_I}^l \to \{0, 1\}, \quad \pi^{(l)} \in \Pi.$$

Задание интерпретации I превращает всякий оператор вида (8) в абстрактный оператор присваивания, называемый I-оператором, всякий предикат вида (9) — в абстрактное булево выражение, именуемое I-выражением, произвольную о.стандартную схему, обозначаемую sch, — в абстрактную программу, обозначаемую $\text{sch}\,|_I$.

Предположим, что sch из K_1 порождена программой prog из K. Тогда абстрактная программа $\text{sch}\,|_I$ отличается от prog только тем, что в ней вместо привычных операций и отношений, определенных на D и используемых в prog, берутся операции и отношения той же арности, но определенные «абстрактно» на D_I.

Заметим, что I может быть выбрана так, что $\text{sch}\,|_I$ совпадет с prog.

Интерпретация I порождает множество состояний памяти R

$$\Xi_I = \{\xi \mid \xi\colon R \to D_I\}$$

и множество состояний регистра R'

$$\overline{\Xi}_I = \{\overline{\xi} \mid \overline{\xi}\colon R' \to D_I\}.$$

Подобно операторам и булевым выражениям, принадлежащим базису $\overline{\Sigma}$, I-операторы и I-выражения осуществляют отображения Ξ_I в себя и Ξ_I в $\{0, 1\}$ соответственно. Выполнение пересылок в prog и в $\mathrm{sch}|_I$ трактуются одинаково.

Процесс выполнения программы $\mathrm{sch}|_I$ определяется на заданной паре начальных состояний $(\overline{\xi}_0, \xi_0)$, где $\overline{\xi}_0 \in \overline{\Xi}_I$, $\xi_0 \in \Xi_I$, и при пустом магазине. Этот процесс состоит в путешествии по графу, сопровождающемся преобразованиями текущих состояний памяти R, регистра R' и магазина. Сами преобразования осуществляются по тем же правилам, что и при выполнении prog на паре начальных состояний регистра и памяти.

Остается справедливым и утверждение 1, где роль программы prog из K играет абстрактная программа $\mathrm{sch}|_I$, построенная по sch из K_1. Поэтому так же, как и в случае prog, полагаем, что $\overline{\xi}_0$ определяется соглашением, принятым в конце § 2, и будем говорить о выполнении программы $\mathrm{sch}|_I$ на состоянии ξ_0, $\xi_0 \in \Xi_I$, и ассоциировать с $\mathrm{sch}|_I$ отображение Ξ_I в себя, в общем случае частичное; называем его функцией, реализуемой программой $\mathrm{sch}|_I$.

Пусть $\mathrm{sch}\,1, \mathrm{sch}\,2$ — о.стандартные схемы, построенные над базисом S. По определению, $\mathrm{sch}\,1, \mathrm{sch}\,2$ *эквивалентны* тогда и только тогда, когда при любой интерпретации I множеств R, F, Π программы $\mathrm{sch}\,1|_I, \mathrm{sch}\,2|_I$ реализуют равные функции.

Так как среди интерпретаций I имеется такая, которая возвращает нас к программам над базисом $\overline{\Sigma}$, то справедливо

Утверждение 2. *Из эквивалентности о.стандартных схем над базисом S следует эквивалентность программ над $\overline{\Sigma}$, породивших эти схемы.*

Теоремой 1 устанавливается фундаментальное свойство эквивалентности схем над S. Введем для этого понятие свободной интерпретации множеств R, F, Π.

На множествах R и F по индукции определим множество T термов.

Базис индукции: всякий элемент из R — терм; всякий нуль-местный символ из F — терм.

Шаг индукции: пусть $f^{(k)}$ — k-местный символ из F и $t_1, \ldots, t_k$ — термы, тогда $f^{(k)}(t_1, \ldots, t_k)$ — терм.

Других термов нет.

Интерпретация I^* называется *свободной*, если $D_{I^*} = T$, а отображения $(I^* f^{(k)})$, $f^{(k)} \in F$, строятся по следующему правилу:

$$(I^* f^{(k)})(t_1, \ldots, t_k) = f^{(k)}(t_1, \ldots, t_k),$$

где $t_1, \ldots, t_k \in T$.

Таким образом, одна свободная интерпретация отличается от другой лишь отображениями, приписываемыми предикатным символам $\pi^{(l)}$, $\pi^{(l)} \in \Pi$.

Теорема 1 (о свободных интерпретациях). *Схемы* $\mathrm{sch}\,1, \mathrm{sch}\,2$ *из K_1 эквивалентны в том и только том случае, если при любой свободной интерпретации I^* множеств R, F, Π программы* $\mathrm{sch}\,1|_{I^*}$ *и* $\mathrm{sch}\,2|_{I^*}$ *реализуют равные функции.*

Доказательству теоремы предпошлем лемму 1.

Для формулировки леммы нам потребуется ввести понятие *значения терма t* на паре I, ξ_0, где I — интерпретация множеств R, F, Π, а $\xi_0 \in \Xi_I$; само значение записывается как $t|_{I, \xi_0}$, а вычисление его определяется по индукции следующей процедурой.

Базис индукции: если $t = r$, где $r \in R$, то $t|_{I, \xi_0} = \xi_0(r)$; если $t = f^{(0)}$, где $f^{(0)}$ — символ нулевой арности из F, то $t|_{I, \xi_0} = (If^{(0)})$.

Шаг индукции: пусть $t = f^{(k)}(t_1, \ldots, t_k)$, и вычислены $t_1|_{I,\xi_0}, \ldots, t_k|_{I,\xi_0}$, тогда $t|_{I,\xi_0} = (If^{(k)})(t_1|_{I,\xi_0}, \ldots, t_k|_{I,\xi_0})$.

Далее заметим, что программа $\mathrm{sch}|_I$ строится на том же графе, что и порождающая ее о.стандартная схема sch. На этом основании, какими бы ни были интерпретации I, I' и состояния ξ_0, ξ_0', где $\xi_0 \in \Xi_I$, $\xi_0' \in \Xi_{I'}$, можно сравнивать на совпадение маршрут путешествия по программе $\mathrm{sch}|_I$ на ξ_0 и маршрут путешествия по программе $\mathrm{sch}|_{I'}$ на ξ_0'.

Теперь сформулируем лемму.

Лемма 1 (о сохранении маршрутов). *По всяким интерпретации I множеств R, F, Π и состоянию ξ_0 из Ξ_I можно построить свободную интерпретацию I^* и состояние ξ_0^* из Ξ_{I^*}, таким образом, что для любой схемы* sch *из K_1 маршрут путешествия по программе* $\mathrm{sch}|_I$ *на ξ_0 совпадет с маршрутом путешествия по программе* $\mathrm{sch}|_{I^*}$ *на ξ_0^*, и если он представляет собой последовательность вершин $V_0, V_1, \ldots, V_i, \ldots$, то история*

$$(\xi_0, V_0)(\xi_1, V_1)\ldots(\xi_i, V_i)\ldots$$

выполнения программы $\mathrm{sch}|_I$ *на ξ_0 и история*

$$(\xi_0^*, V_0)(\xi_1^*, V_1)\ldots(\xi_i^*, V_i)\ldots$$

выполнения программы $\mathrm{sch}|_{I^*}$ *на ξ_0^* удовлетворяют условию*

$$\xi_i(r) = \xi_i^*(r)|_{I,\xi_0}, \qquad r \in R, \quad i = 0, 1, \ldots. \tag{10}$$

Доказательство. Пусть заданы I и ξ_0; построим I^* и ξ_0^*. Для построения I^* достаточно определить только $(I^*\pi^{(l)})$ для любого $\pi^{(l)}$ из Π.

Положим, что для любых $t_1, \ldots, t_l$ из T

$$(I^*\pi^{(l)})(t_1, \ldots, t_l) = (I\pi^{(l)})(t_1|_{I,\xi_0}, \ldots, t_l|_{I,\xi_0}). \tag{11}$$

Начальное состояние ξ_0^* памяти R выберем следующим образом:

$$\xi_0^*(r) = r, \quad r \in R. \tag{12}$$

Покажем, что для построенных I^* и ξ_0^* утверждение леммы истинно.

Доказательство будем проводить индукцией по длине маршрута.

Базис индукции. После прохождения вершины-входа маршруты совпадают и начальные состояния ξ_0, ξ_0^* в силу выбора ξ_0^* и по правилу вычисления значений термов на I, ξ_0 удовлетворяют условию (10).

Шаг индукции. Пусть V — вершина, отличная от входа,

$$(\xi_0, V_0)(\xi_1, V_1)\ldots(\xi_i, V) \quad \text{и} \quad (\xi_0^*, V_0)(\xi_1^*, V_1)\ldots(\xi_i^*, V) \ —$$

истории выполнения $\mathrm{sch}|_I$ на ξ_0 и $\mathrm{sch}|_{I^*}$ на ξ_0^* соответственно на подходе к V, и эти истории удовлетворяют условию (10). Рассмотрим следующие случаи.

1. V имеет один из двух типов: преобразователь или вызов. Так как из преобразователя и из вызова исходит по одной дуге, то при переходе через V путешествия по обеим программам продолжаются по одному и тому же маршруту. Из функционирования преобразователя, определяемого равенством (3), и из функционирования вызова, описанного в § 2, следует, что при переходе через V условие (10) выполняется.
2. V является распознавателем, которому сопоставлен предикат (9). В этом случае равенства (11), определяющие свободную интерпретацию I^*, и индуктивное предположение о выполнении условия (10)

на подходе к V гарантируют, что путешествие по обеим программам при переходе через V продолжится по одной и той же дуге. Поскольку при переходе через распознаватель текущее состояние памяти не меняется, то условие (10) при переходе через V выполняется.

3. Пусть V — финальная вершина подграфа. Так как при подходе к ней, согласно индуктивному предположению, была пройдена одна и та же последовательность вершин в обоих случаях, то из магазина снимутся одинаковые номера вызовов, и путешествия продолжатся по одной и той же дуге. При этом в регистры будут занесены состояния, удовлетворяющие условию (10). Состояния памяти при этом не меняются.
4. Пусть V — возврат. Очевидно, что маршруты при переходе через V сохранятся. Поскольку в возврат мы пришли из финальной вершины, то, согласно описанному выше ее прохождению, состояние регистра по выходе из нее удовлетворяет условию (10). Но тогда из функционирования возврата, описанного в § 2, следует, что при переходе через V условие (10) продолжает выполняться.

На этом шаг индукции исчерпан.

Лемма 1 доказана.

Доказательство теоремы 1. Необходимость ее условия очевидна. Докажем его достаточность.

Пусть для любой свободной интерпретации I^* программы $\mathrm{sch}\,1|_{I^*}$ и $\mathrm{sch}\,2|_{I^*}$ реализуют равные функции. Возьмем произвольную интерпретацию I и произвольное начальное состояние ξ_0 из Ξ_I. По интерпретации I и состоянию ξ_0 построим свободную интерпретацию I^* согласно равенствам (11) и выберем начальное состояние ξ_0^* согласно (12). Тогда путешествия по программе $\mathrm{sch}\,1|_I$ на ξ_0 и по программе $\mathrm{sch}\,1|_{I^*}$ на ξ_0^* проходят по одинаковому маршруту. То же можно сказать о путешествиях по программе $\mathrm{sch}\,2|_I$ на ξ_0 и по программе $\mathrm{sch}\,2|_{I^*}$ на ξ_0^*. Поскольку программы $\mathrm{sch}\,1|_{I^*}$ и $\mathrm{sch}\,2|_{I^*}$ реализуют равные функции, то путешествия по ним, начинающиеся на одном и том же начальном состоянии ξ_0^*, или оба бесконечны, или оба завершаются на одном и том же заключительном состоянии ξ^*. Отсюда, по лемме 1, следует, что путешествия по программам $\mathrm{sch}\,1|_I$ и $\mathrm{sch}\,2|_I$ или оба бесконечны, или оба завершаются на одинаковых состояниях памяти. Достаточность условия теоремы доказана.

Теорема 1 доказана.

Она позволяет рассматривать в дальнейшем только свободные интерпретации множеств R, F, П.

§ 4. Алгебраические модели программ с процедурами

Здесь мы определим алгебраические модели программ с процедурами. Принадлежащие им схемы, в отличие от о.стандартных, будем называть алгебраическими.

Далее выполним следующие действия.

1. По базису S, над которым построен класс K_1 о.стандартных схем, определим базис Bs, над которым строятся алгебраические схемы.
2. Отобразим инъективно класс K_1 в это множество алгебраических схем; образ K_1 обозначим K_2.
3. Введем такое отношение эквивалентности алгебраических схем, которое удовлетворяет требованию: схемы из K_1 эквивалентны тогда и только тогда, когда их образы в K_2 эквивалентны.

Приступим к выполнению этого плана.

Алгебраические схемы определяются над символьным базисом Bs, представляющим собой объединение четырех непересекающихся конечных алфавитов: Y, P, C, Rt. Элементы P называются логическими переменными; элементы остальных алфавитов — символами. Алгебраические схемы задаются конечным ориентированным графом такой же структуры, как и для стандартных схем. Разметка вершин такого графа осуществляется следующим образом. Преобразователям сопоставляются символы из Y, распознавателям — переменные из P, вызовам — символы из C, возвратам — символы из Rt. Разметка дуг такая же, как для стандартных схем.

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

Функционирование алгебраической схемы осуществляется на функциях разметки. *Функция разметки* — это отображение $\mu\colon H \to X$, где

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

Множество всех функций разметки, построенных над Bs, обозначим $\mathfrak{L}$.

Выполнение G на функции μ, $\mu \in \mathfrak{L}$, состоит в путешествии по схеме G, идущим в направлении дуг схемы и сопровождающимся 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 пустом магазине и пустой цепочке. Пусть при путешествии по схеме мы подошли к вершине V с текущей цепочкой h и некоторым состоянием магазина. Тогда различаем следующие случаи.

1. V — преобразователь с сопоставленным ему символом y из Y. При переходе через V к цепочке h справа приписывается символ y, состояние магазина не меняется. Путешествие продолжается по единственной дуге, исходящей из V.
2. V — распознаватель с сопоставленной ему переменной p из P. При переходе через V состояние магазина и текущая цепочка h не меняются, путешествие продолжается по дуге, исходящей из V и помеченной значением $\mu h(p)$.
3. V — вызов, помеченный символом c из C и имеющий номер n. При переходе через V в магазин заносится номер n, к текущей цепочке h справа приписывается символ c, путешествие продолжается по единственной дуге, исходящей из V и ведущей в инициальную вершину подграфа, инцидентного вызову V.
4. V — финальная вершина. При переходе через V с макушки магазина, который заведомо не пуст, снимается номер, и путешествие продолжается по дуге, ведущей к возврату с этим номером.
5. V — возврат с символом rt из Rt. При переходе через V к текущей цепочке h справа приписывается символ rt, состояние магазина не меняется, путешествие продолжается по единственной дуге, идущей из V.
6. V — выход. Путешествие заканчивается. В этом случае говорим, что схема G остановилась на μ, и построенную цепочку h называем *результатом выполнения* G на μ.

Путь, пройденный в схеме при ее выполнении на μ, именуем *прокладываемым в схеме функцией* μ. Если путешествие в схеме G бесконечно, то результат ее выполнения на μ считается неопределенным. Таким образом, схемой G реализуется частичная функция, отображающая $\mathfrak{L}$ в H.

Эквивалентность схем над базисом Bs определяется двумя параметрами: эквивалентностью ν в множестве H и подмножеством L множества $\mathfrak{L}$

функций разметки. Она вводится так. Схемы G_1, G_2 являются (ν, L)-*эквивалентными* в том и только в том случае, если, какой бы ни была функция из L, всякий раз, как на ней останавливается одна из схем G_1, G_2, останавливается и другая, и результаты их выполнения на этой функции разметки — это ν-эквивалентные цепочки.

Множество всех схем над Bs, в котором введена (ν, L)-эквивалентность схем, называется (ν, L)-*моделью программ над* Bs. Определенные таким образом модели программ именуются *алгебраическими.*

Перейдем к построению рассматриваемого далее класса K_2 алгебраических схем. Сначала опишем базис Bs, над которым строятся схемы из K_2. Для этого создадим множества $T1$, $T2$, $T3$, $T4$, состоящие соответственно

- из всех операторов над памятью, принадлежащих базису S,
- из всех предикатов над памятью, принадлежащих S,
- из всевозможных кортежей вида (6), построенных над S,
- из всевозможных кортежей вида (7), построенных над S.

Вводя символьные обозначения элементов этих множеств и полагая при этом, что разным элементам даются различные обозначения, мы получим из множеств $T1$, $T2$, $T3$, $T4$ те алфавиты Y, P, C, Rt, соответственно, которые в совокупности образуют базис Bs.

Обозначим φ взаимно однозначное соответствие между элементами множеств $T1$, $T2$, $T3$, $T4$ и символами базиса Bs. Оно индуцирует отображение множества K_1 всех о.стандартных схем в множество всех алгебраических схем над базисом Bs, определяемое следующим образом: в каждой схеме из K_1 конструкция, приписанная ее вершине, какой бы та ни была, заменяется символом, поставленным в соответствие этой конструкции отображением φ; кортежи, приписанные неглавным подграфам схемы, опускаются; полученная алгебраическая схема объявляется соответствующей данной о.стандартной схеме. Так построенное отображение K_1 в множество схем над Bs обозначим $\overline{\varphi}$. Образ K_1 при отображении $\overline{\varphi}$, по определению, дает искомый класс K_2. Отметим, что $\overline{\varphi}$ биективно отображает K_1 на K_2.

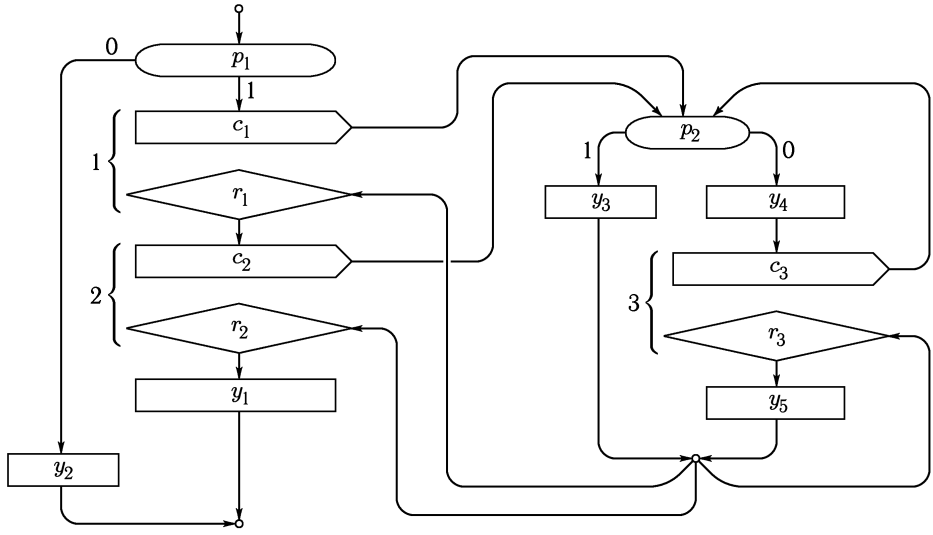


Рис. 4

Обратимся к примеру. На рис. 4 изображена алгебраическая схема, построенная для схемы с рис. 3 по описанным выше правилам. Полагаем, что

$$y_1, y_2, y_3, y_4, y_5 \in Y, \qquad p_1, p_2 \in P, \qquad c_1, c_2, c_3 \in C, \qquad r_1, r_2, r_3 \in Rt.$$

Имеет место теорема.

Теорема 2. *Эквивалентность ν_0 в H и множество L_0 функций разметки для базиса Bs можно выбрать такими, что схемы из K_2 будут (ν_0, L_0)-эквивалентными в том и только в том случае, если их прообразы из K_1 эквивалентны.*

Доказательство теоремы состоит в построении параметров ν_0, L_0 и в проверке того, что они — требуемые.

Эквивалентность ν_0 в H строится на основе свободных интерпретаций множеств R, F, П, определяемых базисом S.

Пусть I — свободная интерпретация. Независимо от ее выбора она индуцирует однозначно определяемые отображения, приписываемые элементам множеств $T1$, $T3$, $T4$. Если A — рассматриваемый элемент, то реализуемое им отображение обозначим (IA). Оно строится в соответствии со структурой элемента A так, как это описано в § 2. Если $A \in T1 \cup T3$, то (IA) отображает множество Ξ_I в себя. Если $A \in T4$, то (IA) отображает декартово произведение $\overline{\Xi}_I \times \Xi_I$ в множество Ξ_I; здесь $\overline{\Xi}_I$ — множество состояний регистра R'.

Со всякой цепочкой h из H будем ассоциировать отображение множества Ξ_I в себя. Обозначим его $[Ih]$ и определим.

Пусть $h = b_1 b_2 \ldots b_k$. Если $k = 0$, то $[Ih]$ — тождественное отображение. Рассмотрим случай, когда $k > 0$. Цепочке h сопоставим последовательность

$$A_1, A_2, \ldots, A_k, \tag{13}$$

где A_i — прообраз символа b_i в отображении φ, $i = 1, 2, \ldots, k$. По произвольному состоянию ξ_0 из Ξ_I, руководствуясь последовательностью (13), построим последовательность

$$\xi_1, \xi_2, \ldots, \xi_k \tag{14}$$

состояний из Ξ_I и будем говорить, что $[Ih]$ отображает ξ_0 в ξ_k.

Последовательность (14) строится по индукции.

Базис индукции. Поскольку h — правильная цепочка, $A_1 \in T1 \cup T3$. Полагаем, что ξ_1 — это результат применения (IA) к ξ_0.

Шаг индукции. Пусть построены $\xi_1, \ldots, \xi_i$, $i \geqslant 1$. Если $A_{i+1} \in T1 \cup T3$, то ξ_{i+1} получаем применением (IA_{i+1}) к ξ_i.

Предположим, что $A_{i+1} \in T4$. Так как h — это правильная цепочка, то в ней символу b_{i+1} соответствует единственный символ b_j, где $j < i+1$, такой, что $b_j \in C$ и между b_j и b_{i+1} число вхождений символов из C равно числу вхождений символов из Rt. По состоянию ξ_j построим состояние $\overline{\xi}_j$ из $\overline{\Xi}_I$, используя правило: для любой переменной r' из R'

$$\overline{\xi}_j(r') = \xi_j(r),$$

где r — прообраз переменной r' в отображении R в R'. Построив $\overline{\xi}_j$, полагаем, что ξ_{i+1} — это результат применения (IA_{i+1}) к паре $(\overline{\xi}_j, \xi_i)$.

Итак, построение последовательности (14) описано и вместе с этим определено отображение $[Ih]$.

Эквивалентность ν_0 в H определим требованием: цепочки h_1, h_2 из H являются ν_0-эквивалентными тогда и только тогда, когда

$$[Ih_1] \equiv [Ih_2].$$

Легко проверить, что это требование можно ослабить, заменив его следующим:

$$[Ih_1]\xi_0^* = [Ih_2]\xi_0^*,$$

где ξ_0^* — состояние из Ξ_I, определенное условием (12).

Опишем теперь множество L_0 функций разметки. Всякой свободной интерпретации I множеств R, F, Π сопоставим функцию μ_I из $\mathscr{L}$ и положим

$$L_0 = \{\mu_I \mid I \text{ — свободная интерпретация множеств } R, F, \Pi\}.$$

Пусть h — цепочка из H и $p \in P$; определим значение $\mu_I h(p)$.

Предположим, что p является образом предиката над памятью, имеющего вид (9). Найдем состояние ξ из Ξ_I, определяемое равенством

$$\xi = [Ih]\xi_0^*.$$

Полагаем, что

$$\mu_I h(p) = (I\pi^{(l)})(\xi(r_1), \ldots, \xi(r_l)).$$

Итак, параметры ν_0, L_0 построены. Доказательство того, что они требуемые, проводится на основании леммы 2.

Лемма 2. *Для любой схемы* sch *из* K_1 *и любой свободной интерпретации* I *множеств* R, F, Π *выполнение программы* $\mathrm{sch}|_I$ *на* ξ_0^* *и выполнение алгебраической схемы* $\overline{\varphi}(\mathrm{sch})$ *на функции* μ_I *идут по общему маршруту.*

Лемма 2, подобно лемме 1, доказывается индукцией по длине маршрута. Само доказательство мы опускаем.

Теорему 2 считаем нами доказанной.

Она позволяет сделать

Утверждение 3. *Из эквивалентности схем, принадлежащих классу* K_2, *следует эквивалентность программ, породивших эти схемы.*

Здесь процесс порождения понимается так: по программе над базисом $\overline{\Sigma}$ находится соответствующая ей стандартная схема в K_1, а по последней — ее образ в отображении $\overline{\varphi}$, т. е. алгебраическая схема из K_2.

Заключение

Таким образом, (ν_0, L_0)-модель, где ν_0, L_0 определяются теоремой 2, является аппроксимирующей для исходных программ. Отметим (см. [2]), что в множестве алгебраических моделей, построенных над общим базисом, выполняется

Утверждение 4. *Из* (ν_1, L_1)*-эквивалентности схем следует их* (ν_2, L_2)*-эквивалентность, если*

$$\nu_1 \longrightarrow \nu_2, \qquad L_1 \supseteq L_2.$$

Опираясь на это утверждение, можно сказать, что вместе с построением (ν_0, L_0)-модели мы получили множество аппроксимирующих (ν, L)-моделей, где

$$\nu_0 \longrightarrow \nu, \qquad L_0 \supseteq L.$$

СПИСОК ЛИТЕРАТУРЫ

1. Подловченко Р. И. Рекурсивные программы и иерархия их моделей // Программирование. — 1991. — № 6. — С. 44–51.
2. Подловченко Р. И. От схем Янова к теории моделей программ // Математические вопросы кибернетики. Вып. 7. — М.: Наука, 1998. — С. 281–302.
3. Подловченко Р. И. Эквивалентные преобразования схем программ для «запутывания» самих программ // Программирование. — 2002. — № 2. — С. 66–80.
4. Alur R., Yanakakis M. Model checking of hierarchical state machines. — Proc. of the Sixth ACM symposium on the foundations of Software Engineering (FSE 98), 1998. — P. 175–188.

Поступило в редакцию 9 VII 2003