УДК 004.054

А.С. Марковский[1]
В.В. Петров[2]
И.О. Бурова[3]

АЛГОРИТМ ПОСТРОЕНИЯ ОПИСАНИЯ ЛОГИЧЕСКОЙ СХЕМЫ ПРОГРАММЫ ПО ЕЕ ТЕКСТУ

В статье дано неформальное описание алгоритма решения задачи описания логической схемы программы, а также основных этапов алгоритма. Алгоритм построения описания логической схемы программы по ее тексту рассматривается на примере языка СИ++.

Ключевые слова: *алгоритмизация схемы программы, описание данных, структуры данных.*

A.S. Markovsky
V.V. Petrov
I.O. Burova

ALGORITHM FOR CONSTRUCTING A DESCRIPTION OF PROGRAM'S LOGICAL SCHEME ACCORDING TO ITS TEXT

The article gives an informal description of the algorithm for solving the problem of the logical scheme of a program, as well as the main steps of the algorithm. The algorithm for constructing a description of the logical scheme of a program by its text is examined using the example of a language C++.

Keywords: *algorithmization of program scheme, description of data, data structures.*

На сегодняшний день активное и повсеместное внедрение информационных технологий, важнейшим элементом которых является программное обеспечение (ПО), обусловило высокий уровень требований к надежности, производительности, защищенности и другим характеристикам ПО. В то же время, как показывает практика в нашей стране, так и за рубежом, чем сложнее и объемнее программное обеспечение, тем больше в нем дефектов.

В рамках решения данной проблемы на передний план выходит задача алгоритмизации анализа логической схемы программы. Алгоритм построения описания логической схемы программы по ее тексту состоит из двух основных этапов. На первом этапе происходит разбиение программы на отдельные фрагменты, строится описание логической схемы каждого фрагмента в виде формулы языка Схем Янова (ЯС). В этом описании при помощи схемных переменных и знаков операций указывается, с какими другими фрагментами и как связан данный фрагмент. Все фрагменты, связанные с данным фрагментом, представлены в описании последнего схемными переменными. На втором этапе происходит сборка описания логической схемы программы из описаний логических схем фрагментов путем подстановки в их описания (вместо схемных переменных) описаний логических схем соответствующих фрагментов.

Строго понятие фрагмента определяется действием алгоритма построения описания логической схемы, а именно той его частью, которая выполняет разбиение текста программы на фрагменты и строит их описание на языке ЯС [1; 2; 5].

Для содержательного описания фрагмента введем понятие линейного блока [2; 4]. *Линейным блоком* называется последовательность операторов программы, в которой:

1) каждый из операторов, за исключением, быть может, первого, имеет ровно одного непосредственного предшественника;

2) все операторы в последовательности имеют ровно одного преемника;

3) все операторы, за исключением, быть может, первого, не имеют меток.

[1] Старший научный сотрудник, кандидат технических наук, начальник лаборатории, Военно-космическая академия им. А.Ф. Можайского.

[2] Старший научный сотрудник, начальник лаборатории, Военно-космическая академия им. А.Ф. Можайского.

[3] Младший научный сотрудник, Военно-космическая академия им. А.Ф. Можайского.

Логическую схему линейного блока назовем линейной схемой. Очевидно, что линейная схема имеет один вход и один выход. Ранг формулы, описывающей линейную схему, равен (1,1).

Оператором управления назовем оператор, который, в терминах примитивных схем, имеет не менее одного выхода.

Фрагмент есть либо линейный блок, либо оператор управления. Каждое описание фрагмента имеет имя вида C^i, где C – схемная переменная, а i – номер фрагмента. Номер фрагмента равен либо номеру метки первого оператора фрагмента (если таковая есть), либо $i = max+j$, где max – максимальный номер метки среди всех авторизованных меток программы, а j – порядковый номер фрагмента среди всех непомеченных фрагментов (т.е. тех фрагментов, первый оператор которых не имеет метки) при просмотре программы сверху вниз.

Соответствие между операторами языка СИ++ и элементами языка ЯС мы зададим, указав, как каждый оператор изображается средствами языка ЯС. Это изображение будет использовано при построении описания логической схемы программы к ее тексту.

Операторы, организующие линейные вычисления:

– оператор присваивания;

– пустой оператор;

– составной оператор;

– оператор – выражение вызова функции, которая не возвращает никакого значения.

Например, <имя функции> (<список аргументов>). Данные операторы будут изображаться формулой s. Все операторы спецификации в описании логической схемы никак отражаться не будут.

Управляющие конструкции языка СИ++ образует следующая группа операторов: IF, SWITCH, FOR, WHILE, DO ... WHILE, GOTO, BREAK, CONTINUE, RETURN и др.

GOTO n – оператор безусловной передачи управления. Этот оператор указывает на то, что фрагмент, оканчивающийся оператором – предшественником GOTO n, связан с фрагментом с именем $\tilde{N}^n_{1,1}$. При описании схемы на языке ЯС эта связь между соответствующими фрагментами устанавливается либо при помощи операции композиции (если утверждение с меткой n следует в тексте программы за оператором GOTO n), либо при помощи операции замыкания (если оператор с меткой указан в тексте программы перед оператором GOTO n).

Оператор IF:

а) IF(a) n_1, n_2 – полный IF. Этому оператору сопоставляется схема с одним входом и тремя выходами, которая на ЯС описывается так:

$(p \cdot (C^{n_1} + C^{n_2})$;

б) IF(a) n – сокращенный IF.

Этому оператору сопоставляется формула вида $(p \cdot C^n)$, где C^i – формула, соответствующая оператору s.

Оператору FOR (k, l, m) соответствует следующая формула ЯС: $\frac{1}{2}(p \cdot (C^i + e^1_1))$, где C^i – имя первого фрагмента тела цикла.

RETURNa.

Этому оператору соответствует формула e^1_1. Заметим, что этим он определяет точки выхода программы (подпрограммы).

Операторы ввода–вывода.

Всем формам этой группы операторов cout <<, cin >> соответствует формула s.

Операторы присваивания изображаются на ЯС формулой s.

Описание алгоритма.

Алгоритм фрагментации текста программы

Прежде чем перейти к описанию самого алгоритма, уточним понятие фрагмента применительно к операторам языка СИ++. Фрагментом программы может быть:

а) линейный блок,

б) одна из управляющих конструкций, кроме безусловного GOTO, CONTINUE, RETURN;

в) операторы ввода – вывода.

Будем говорить, что оператор ссылается сверху (ссылка сверху) на другой оператор с меткой n, если первый предшествует второму в тексте программы. В противном случае будем говорить, что ссылка происходит снизу.

Исходной информацией для алгоритма фрагментации текста программы служат: текст программы и таблица REF. Эта таблица состоит из четырех столбцов и p строк, где p – количество авторизованных меток в программе. Изначально в первом столбце таблицы REF расположены авторизованные метки программы в порядке их появления в тексте программы. Эта информация нам потребуется для определения вида ссылки (сверху, снизу), а также на этапе сборки описания логической схемы программы из описаний фрагментов. В последнем случае эта информация нам будет нужна для сохранения на множестве фрагментов программы отношения следования, определяемого следованием операторов в тексте программы. Во втором столбце таблицы REF мы будем называть число ссылок сверху на метки программы; в третьем столбце – число ссылок снизу на эти же метки. Четвертый столбец будет использоваться для обработки циклов. Перед началом работы алгоритма на второй, третий и четвертый столбцы засылаются нули.

Результатом работы алгоритма фрагментации являются: список описаний фрагментов, таблица входов программы ENTR и таблица выходов программы EXIT. В эти таблицы заносятся имена фрагментов, содержащих утверждения, которые определяют входы и выходы программы соответственно.

В алгоритме фрагментации использованы:

Q – переменная типа целый;

CL – счетчик числа непомеченных фрагментов;

CYCLE – стек обработки циклов;

Next – процедура, которая выдает очередной оператор программы.

Значение переменной Q и оператор программы (который обрабатывается в данный момент) определяют действие алгоритма.

Алгоритм последовательно просматривает операторы программы. При появлении метки или оператора управления текущий фрагмент закрывается, и открывается новый фрагмент. При этом в закрываемый фрагмент ставится имя открываемого фрагмента. Операция, стоящая перед этим именем, указывает, как закрываемый фрагмент связан с открываемым.

Несколько отличным образом обрабатывается оператор цикла. Когда алгоритм встречает оператор цикла FOR (*n, l, m*), то в таблице REF в элемент REF (*n*, 4), (т.е. в четвертом элементе строки, соответствующей метке *n*, добавляется единица. Число, указанное в этом элементе таблицы, определяет количество операторов цикла, для которых оператор, помеченный меткой *n*, является последним оператором в теле соответствующих циклов. Далее в стек CYCLE заносится имя фрагмента, соответствующего этому оператору цикла. После этого обрабатывается первый оператор тела цикла, и открывается новый фрагмент со своим именем. Имя этого фрагмента также заносится в стек CYCLE. Таким образом, в этом стеке располагаются (в случае вложенных циклов) пары имен фрагментов: первое соответствует фрагменту оператора цикла, второе – первому фрагменту тела цикла. Фрагмент, соответствующий оператору цикла, порождается в момент встречи оператора с меткой *n*. При этом используется информация из стека CYCLE.

Алгоритм сборки фрагментов

Эта часть алгоритма построения описания логической схемы программы по тексту программы собирает описание логической схемы из описаний фрагментов. Сборка происходит путем последовательной подстановки в описание головного фрагмента (указанного в таблице ENTR), который объявляется основной формулой других фрагментов вместо соответствующих схемных переменных. Если данный фрагмент связан одновременно с несколькими фрагментами, то сначала в основной формуле все схемные переменные, соответствующие данному фрагменту (т.е. являющиеся его именем), собираются в одном месте согласно специальным правилам приведения подобных. Только после этого выполняется подстановка вместо данной схемной переменной описания соответствующего фрагмента.

Алгоритм

1. *Начальная установка.* Фрагмент, указанный в таблице ENTR, объявляется основной формулой. Перейти к Просмотру основной формулы;

2. *Просмотр основной формулы:* Последовательно просматриваем основную формулу, пока не встретим либо схемную переменную, либо признак конца;

2.1. Если встретили C^n, то:

2.1.1. Если REF [*n*, 2] ≠ 0, то перейти к Подстановке фрагмента C^n;

2.1.2. Перейти к Просмотру основной формулы.

2.2. Если встретили признак конца, то:

2.2.1. Если в основную формулу были сделаны подстановки, то перейти к Приведению подобных;

2.2.2. Находим в списке фрагментов первый, имя C^n которого есть в основной формуле;

2.2.3. Перейти к Подстановке фрагмента C^n;

2.2.4. Если в основной формуле нет схемных переменных, то алгоритм окончен.

3. *Подстановка фрагмента:*

3.1. Среди описаний фрагментов находим фрагмент с именем C^n;

3.2. Если REF [*n*, 3] ≠ 0 ∨ REF [*n*, 2] ≠ 0, то метим найденный фрагмент меткой C^n;

3.3. Переписываем найденный фрагмент на место C^n в основную формулу;

3.3.1. Если при перезаписи встретилось имя фрагмента C^m, то:

3.3.1.1. Если фрагмент C^m расположен в списке фрагментов перед фрагментом C^n, то REF [*m*, 3] : = REF [*m*, 3] – 1, и перейти к 3.3;

3.3.1.2. REF [*m*, 2]: = REF [*m*, 2] – 1.

3.3.2. Если дошли до конца фрагмента, то уничтожить фрагмент.

3.4. Перейти к Просмотру основной формулы (шаг 2).

Пояснения к алгоритму сборки фрагментов.

Начальная установка: *основная формула* – это формула, в которую подставляются описания

фрагментов вместо соответствующих схемных переменных. Таким образом, именно основная формула в процессе сборки фрагментов будет преобразована в формулу, описывающую логическую схему программы.

Просмотр основной формулы: условие REF [n, 2] = 0 (шаг 2.1) означает, что все C^n, соответствующие ссылкам сверху на фрагмент с именем C^n, уже вошли в основную формулу. Поэтому, собрав в основной формуле все C^n, соответствующие одному и тому же фрагменту, в одно место, т.е. приведя подобные члены, мы можем заменить эту единственную схемную переменную C^n на соответствующий фрагмент. (Подробно о приведении подобных членов сказано в описании процедуры «Приведение подобных».)

Шаг 2.2.1: если при просмотре основной формулы оказалось, что:

а) в основной формуле есть схемные переменные;

б) у всех из них соответствующие REF [n, 2] = = 0, т.е. ни одной подстановки выполнить нельзя (будем называть такой случай «тупиковой ситуацией»), то среди тех фрагментов, чьи имена есть в основной формуле, ищется и подставляется тот, который прежде других встречается в программе.

Шаг 2.2.4: отсутствие схемных переменных в основной формуле является признаком окончания сборки.

Подстановка фрагмента. Истинность условия REF [n, 3] $\neq 0 \vee$ REF [n, 2] $\neq 0$ (шаг 3.2) говорит о том, что:

а) либо мы подставляем фрагмент, на который еще встретятся ссылки снизу (REF[n, 3] $\neq 0$);

б) либо мы подставляем фрагмент для того, чтобы выйти из тупиковой ситуации (REF [n, 2] $\neq$ $\neq 0$).

Как в первом, так и во втором случае при дальнейшей работе алгоритма нам встретятся схемные переменные, соответствующие уже подставленному фрагменту. Поэтому его надо пометить, чтобы впоследствии уметь найти его начало и согласовать ссылки на него. Для этого введем в язык ЯС понятие метки. Синтаксически метка совпадает с понятием схемной переменной, с той лишь разницей, что после метки (перед формулой, соответствующей описанию фрагмента) ставится символ «:». Заметим, что метки используются исключительно на этапе сборки и в результирующей формуле, описывающей логическую схему программы, не участвуют.

Шаг 3.3.1: если фрагмент C^m расположен в списке фрагментов перед C^n (который в данный момент подставляется), то, следовательно, ссылки из C^n на C^m могут быть только снизу (REF [m 3] – 1), в противном случае – сверху (REF [m, 2] – 1).

Заключение

Несмотря на представленный выше алгоритм построения описания логической схемы программы по ее тексту, считать решенной задачу алгоритмизации анализа логической схемы программы еще рано. На следующих этапах, используя результаты, представленные в данной статье, при помощи процедуры «Приведения подобных» преобразуем основную формулу к виду, в котором ни одна схемная переменная C^n не встречается дважды [3–5]. В дальнейшем опишем минимальное вхождение схемной переменной и представим алгоритм ее выделения, что позволит решить озвученную выше задачу.

Литература

1. Агафонов В.Н. Спецификация программ: понятийные средства и их организация. – Новосибирск : Наука (Сибирское отделение), 1987. – С. 30–73.

2. Андерсон Р. Доказательство правильности программ / пер. с англ. Б.Н. Зобниной; под ред. Д.Б. Подшивалова. – М. : Мир, 1982. – С. 154–159.

3. Донаху Дж. Взаимодополняющие определения семантики языка программирования // Семантика языков программирования. – М. : Мир, 1980. – С. 222–394.

4. Ершов А.П. Современное состояние теории схем программ // Проблемы кибернетики : сборник статей. – Вып. 27. – М. : Наука, 1973. – С. 87–110.

5. Ершов А.П. Об операторных схемах Янова // Проблемы кибернетики : сборник статей. – Вып. 20. – М. : Наука, 1967. – 181–200.

6. Новиков А.Н. Математическая модель обоснования вариантов реконфигурации распределенной автоматизированной контрольно-измерительной системы / А.Н. Новиков, А.А. Нечай, А.В. Малахов // Вестник Российского нового университета. Серия «Сложные системы: модели, анализ и управление». – 2016. – Выпуски 1–2. – С. 56–59.

7. Нечай А.А. Методика повышения надежности функционирования систем, организованных на перепрограммируемых элементах / А.А. Нечай, П.Е. Котиков // Вестник Российского нового университета. Серия «Сложные системы: модели, анализ и управление». – 2016. – Выпуски 1–2. – С. 87–89.