УДК 519.681

В.Г. Акуловский

РЕАЛИЗАЦИЯ ФОРМАЛИЗОВАННОГО ПЕРЕХОДА ОТ АЛГОРИТМА К ПРОГРАММЕ СРЕДСТВАМИ РАСШИРЕННОЙ АЛГЕБРЫ АЛГОРИТМОВ

На примере продемонстрирована возможность формализованного перехода от алгоритма к программе за счет развития и использования средств расширенной системы алгоритмических алгебр. Показаны возможности преобразования полученной программы и автоматического перехода на требуемый язык программирования.

Введение

В процессе разработки любого алгоритма в некоторый момент времени достигается такой уровень детализации, после которого осуществляется переход от алгоритма к программе. При этом алгоритм, в каком бы виде он не был записан, необходимо переписать на выбранном языке программирования. Отметим, что выбор момента такого перехода не формализован и определяется главным образом двумя причинами: квалификацией (предпочтениями) разработчика и уровнем целевого языка программирования. На этом этапе разработки возникают в больших количествах трудно выявляемые ошибки, вызванные неадекватностью полученной программы исходному алгоритму. Наличие таких ошибок подтверждает тот факт, что степень детализации алгоритма в существенной мере определяет качество получаемой программы (т.е., чем детальнее проработан алгоритм, тем выше качество программы). Таким образом, очевидно, что для минимизации числа ошибок подобного рода необходимо, во-первых, обеспечить возможность детализации алгоритма до максимально низкого уровня, во-вторых, автоматизировать процесс перехода от алгоритма к программе.

Для решения этой задачи формализуем процесс перехода от алгоритма к программе, для чего воспользуемся расширенной системой алгоритмических алгебр (САА-Р), предложенной в [1], которая восходит к системе алгоритмических алгебр (САА) В. Глушкова [2, 3]. При этом применим простой прием. Будем осуществлять детализацию алгоритма до тех пор, пока каждый оператор алгебры не будет описывать одну операцию, определенную на элементарных данных и доступную в некотором (целевом) языке программирования. Заметим, что элементарными будем считать данные, допускающие обработку с помощью одного оператора (одной операции целевого языка).

Поскольку данные являются важнейшей и неотъемлемой составляющей любой программы, то для “плавного” перехода от алгоритма к программе необходимо включить в рассмотрение данные на этапе разработки алгоритма. С этой целью модифицируем САА-Р таким образом, чтобы формализовать взаимосвязь операторов и данных.

Формализация данных в расширенной алгебре алгоритмов

Для включения в рассмотрение данных, воспользуемся абстрактной моделью ЭВМ [2, 3] в трактовке, предложенной в [4]. В данной трактовке в качестве управляющего автомата УА рассматривается произвольный алгоритм, в качестве автомата операционного ОА – обрабатываемая этим алгоритмом информация (множество данных Δ). Выходные сигналы A управляющего автомата УА отождествляются с операторами, которые изменяют данные, т.е. состояние операционного устройства ОА. Выходные сигналы операционного автомата, представляют собой значения различных элементарных логических условий α, харак-

теризующих значения данных и/или соотношения между ними.

Разобъем множество Δ на два непересекающихся подмножества $\Delta = DS \cup DD$, где

DS – статические данные, которые можно интерпретировать как данные, расположенные в оперативной памяти;

DD – динамические данные, которые можно интерпретировать как данные, расположенные в регистровой памяти, т.е. динамически изменяемые данные, нефиксируемые в памяти.

Известные основные понятия (оператор, операция, логическое условие), введенные в САА [2, 3] и расширенные в САА-Р [1] будем интерпретировать в соответствии с рассматриваемой трактовкой абстрактной модели ЭВМ. Отметим, что в данной работе будут рассмотрены только те основные алгоритмические конструкции, которые найдут применение при решении поставленной задачи.

Пусть $< U, W, \Omega >$ – САА-Р, где
U – множество операторов; W – множество логических условий; Ω – сигнатура операций, состоящая из логических операций – Ω_1, принимающих значения на множестве W и операций – Ω_2, принимающих значения на множестве операторов *U*.

Выделим на множестве операторов два подмножества $U = U' \cup U''$, где операторы из множества U' изменяют статические (из множества *DS*), операторы из множества U'' динамические (из множества *DD*) данные.

Операторы из множества U' определим таким образом:

$$A(D') = D'' \quad \forall A \in U', \ \forall D', D'' \in \mathrm{DS}.$$

Множество данных D' – область определения, а множество данных D'' – область значений операторов из U' и для множеств D' и D'' может выполняться любое из следующих соотношений:

$$D' \subseteq D'', \ D'' \subseteq D', \ D' \cap D'' \neq \varnothing,$$
$$D' \cap D'' = \varnothing, \ D' = D''.$$

Такой оператор будем называть бинарным. Запишем его в виде $(D'')A(D')$.

В частном случае, когда $D' = D''$, оператор запишем в виде $A(D')$ и назовем унарным.

Операторы из множества U'' определим так:

$$A'(D') = D'' \ \forall \mathrm{A}' \in U'', \ \forall D' \in DS, \quad \forall D'' \in DD \qquad (1)$$

и только после выполнения данных операторов множество *DD* определено, т.е. эти данные доступны для анализа. Множество D' – область определения, множество D'' – область значений операторов из U″. Для множеств D' и D'' естественно выполняется соотношение $D' \cap D'' = \varnothing$. Оператор будем записывать в виде $A'(D')$.

Можно сказать, что оператор $(D'')A(D') \in U'$ отличается от оператора $A'(D') \in U''$ тем, что первый изменяет данные множества *DS*, т.е. изменяет данные в оперативной памяти, а второй на данные не влияет.

Множество операторов *U* включают тождественный оператор, не изменяющий множества данных Δ, который определим таким образом:

$$E(D) = D \quad \forall D \in \Delta \qquad (2)$$

и будем записывать в виде *E*.

Определим операции на множестве *W*.

Множество предикатов P, определенных на множестве Δ, введем следующим образом:

$$p(D) = \alpha(D) = D, \ \forall \mathrm{p} \in \mathrm{P}, \quad \forall \mathrm{D} \in \Delta, \ D \neq \varnothing. \qquad (3)$$

Результатом вычисления предиката является логическое условие α(*D*), характеризующее текущее состояние множества данных $D \in \Delta$, состояние операционного автомата (ОА), которое при вычислении предиката p не изменяется.

Элементы множества P принимают истинные значения трехзначной логики $E_3 = \{0, 1, \mu\}$, где 0 – ложь, 1 – истина, μ – промежуточное значение. На множестве E_3 введено отношение порядка < так, что $0 < \mu < 1$ и известны обобщенные логические операции (дизъюнкция, конъюн-

кция, отрицание, циклическое отрицание) со своими таблицами истинности [1, 5–7].

Кроме этих операций определим операцию (введенную В.М. Глушковым [2, 3]), позволяющую осуществлять прогнозирование вычислительного процесса.

В предлагаемой работе реализуем данную возможность так. Операцию левого умножения предиката на оператор в общем случае с учетом определений (1), (3) запишем таким образом:

$$A'(D')p = p(A'(D')) = p(D'') = \alpha(D'')$$

$$\forall D' \in DS,\ \forall D'' \in DD,\ \forall A'(D') \in U''. \quad (4)$$

Результатом выполнения данной операции является условие $\alpha(D'')$, характеризующее изменение состояния множества данных DS, соответствующее тому, которое было бы получено после выполнения оператора $(D'')A(D') \in U'$. Однако, статические данные множества *DS* при этом остаются неизменными, так как при $A'(D') \in U''$ $D'' \in DD$.

Смысл данного оператора состоит в прогнозировании вычислительного процесса, а результат выполнения такого оператора – логическое условие $\alpha(D'')$, которое является прогнозом выполнения оператора $(D'')A(D') \in U'$.

Операции, определенные на множестве операторов *U*, введем таким образом.

Композиция (последовательное выполнение). Композиция операторов $(\hat{D}')A(\hat{D})*(\tilde{D}')B(\tilde{D})$, $\forall (\hat{D}')A(\hat{D}),(\tilde{D}')B(\tilde{D}) \in U'$, $\forall \hat{D},\hat{D}',\tilde{D},\tilde{D}' \in A$ означает, что последовательно выполняются сначала оператор $(\hat{D}')A(\hat{D})$, затем $(\tilde{D}')B(\tilde{D})$.

Рассмотрим два возможных варианта данной операции.

Первый – когда $(\hat{D}')A(\hat{D})$, $(\tilde{D}')B(\tilde{D}) \in U'$. При этом некоторое состояние множества *DS*, полученное после выполнения оператора $(\hat{D}')A(\hat{D})$, поступает (передается) оператору $(\tilde{D}')B(\tilde{D})$.

Второй – когда $A(\hat{D}) \in U''$, а $(\tilde{D}')B(\tilde{D}) \in U'$ тогда операция записывается в виде

$$A(\hat{D})*(\tilde{D}')B(\tilde{D}). \quad (5)$$

В данном случае оператору $(\tilde{D}')B(\tilde{D})$ передается состояние множества $\Delta = DS \cup DD$.

Отметим, что операцию композиции операторов "*" в очевидных случаях будем опускать.

α – итерация $_{\alpha(D)}\{(D'')A(D')\}$ (соответствует программной конструкции WHILE). Операция α-итерации оператора $(D'')A(D')$ состоит сначала в проверке условия $\alpha(D)$, затем, если условие истинно, выполняется оператор $(D'')A(D')$ и вновь проверяется условие $\alpha(D)$. Данный циклический процесс осуществляется до тех пор, пока условие $\alpha(D)$ не станет ложным. Для завершения операции α–итерация необходимо выполнение такого соотношения: $D \cap D'' \neq \varnothing$.

Операция α_3 – дизъюнкция $\langle\alpha(D)\rangle((\hat{D}'')A(\hat{D}') \vee (\tilde{D}'')B(\tilde{D}') \vee (\breve{D}'')C(\breve{D}'))$. Данная операция, об удобстве и эффективности изложена в [5, 6]. Она ориентирована на использование трехзначных логических условий, которую запишем в виде

$$\langle\alpha(D)\rangle((\hat{D}'')A(\hat{D}') \vee (\tilde{D}'')B(\tilde{D}') \vee (\breve{D}'')C(\breve{D}')) =$$

$$= \begin{cases} (\hat{D}'')A(\hat{D}'), \text{если } \alpha(D) = 1; \\ (\tilde{D}'')B(\tilde{D}'), \text{если } \alpha(D) = 0; \\ (\breve{D}'')C(\breve{D}'), \text{если } \alpha(D) = \mu; \end{cases}$$

где $\hat{D}'', \hat{D}', \tilde{D}'', \tilde{D}', \breve{D}'', \breve{D}' \in DS$, $D \in DD$, $D \neq \varnothing$.

Результат выполнения данной конструкции – выполнение одного из трех возможных операторов, выбирающегося в соответствии со значением логического условия $\alpha(D)$, и принимающего истинные значения трехзначной логики E_3.

α–дизъюнкция, которая вводится как частный случай α_3 – дизъюнкции (с учетом (2)), записывается в виде

$$[\alpha(D)]((\hat{D}'')A(\hat{D}') \vee (\tilde{D}'')B(\tilde{D}')) =$$
$$= [\alpha(D)]((\hat{D}'')A(\hat{D}') \vee (\tilde{D}'')B(\tilde{D}') \vee E) =$$
$$= \begin{cases} (\hat{D}'')A(\hat{D}'), \text{если } \alpha(\mathrm{D}) = 1; \\ (\tilde{D}'')B(\tilde{D}'), \text{если } \alpha(\mathrm{D}) = 0; \end{cases}$$

и обеспечивает выбор одного из двух операторов в зависимости от значения логического условия $\alpha(\mathrm{D})$.

α-фильтрация (последовательная фильтрация)**,** – частный случай α–дизъюнкции

$$[\alpha(D)]((\hat{D}'')A(\hat{D}')) = [\alpha(D)]((\hat{D}'')A(\hat{D}') \vee E)$$

и обеспечивает выполнение оператора $(\hat{D}'')A(\hat{D}')$ только при истинном значении $\alpha(D)$.

Отметим, что в качестве логического условия в операциях α_3 – дизъюнкция, α – дизъюнкция, α – фильтрация и α – итерация могут выступать предикат и операции левого умножения предиката на оператор и, что данные операции записываются аналогично в случае унарных операторов.

Предлагаемый подход позволил в рамках САА-Р включить в рассмотрение данные, т.е. специфицировать взаимосвязи между операторами и данными. При этом сохраняются все возможности по построению производных алгоритмических конструкций и преобразованию регулярных схем (РС), которые являются средством описания алгоритма рассматриваемого формального аппарата.

Разработка алгоритмов на требуемом уровне детализации

Наиболее наглядным способом продемонстрировать возможности предлагаемого подхода является решение конкретной задачи, что мы и сделаем, выбрав некоторую элементарную задачу. Данная задача, иллюстративная, по мнению автора, не носит, отвлеченного от практических задач характера. Будем полагать, что в процессе декомпозиции некоторого алгоритма, возникла следующая подзадача.

Имеется два массива:

$$X = x_1, x_2, \ldots, x_k,$$
$$Y = y_1, y_2, \ldots, y_n, .$$

Необходимо вычислить сумму элементов массива X, если $k < n$, сумму элементов массива Y, если $k > n$. В том случае, когда $k = n$ необходимо, если k четное переставить элементы массива X на место одноименных элементов массива Y, а в противном случае – переставить элементы массива Y на место одноименных элементов массива X.

Запишем требуемый алгоритм в виде следующей регулярной схемы:

$$(S, X, Y)A(X, Y) =$$
$$= \langle\alpha\rangle((S)сум(X) \vee (S)сум(Y) \vee$$
$$\vee ([\beta]((Y)перест(Y, X) \vee (X)перест(X, Y))),$$

где α – условие $k < n$;

β–условие k-четное;

(S)сум(X) – оператор, суммирующий элементы массива X;

(S)сум(Y) – оператор, суммирующий элементы массива Y;

(Y)перест(Y,X) – оператор, переставляющий элементы массива X на место одноименных элементов в массив Y;

(X)перест(X,Y) – оператор, переставляющий элементы массива Y на место одноименных элементов в массив X.

В данном алгоритме мы воспользовались операцией α_3 – дизъюнкции, поэтому остановимся на работе данной операции.

Известно, что предикат – это функция, вычисляющая некоторое отношение, определенное на множестве данных. В большинстве языков программирования используются (и поддерживаются аппаратно) следующие отношения: “<”, ”>”, "≤", "≥", "=", ≠. Будем считать, что в операторе α_3 – дизъюнкции используются только отношения “<”, “>”. В результате вычисления которых логическое условие принимает значение 1(истина), в случае если отношение выполняется, 0 (ложь) –

противоположного значения и μ (промежуточное значение) – равенства сравниваемых величин.

Очевидно уровень детализации приведенного алгоритма можно переписывать в виде программы на некотором языке программирования. В данном случае реализацию всех алгоритмических конструкций необходимо переписать в терминах языка программирования. Затем принять решения о программной реализации операции суммирования и перемещения массивов. Мы не будем проходить этот традиционный путь, а осуществим более глубокую детализацию РС.

Для построения алгоритмов требуемого уровня детализации необходимо строить алгоритмические конструкции, адекватные языковым конструкциям. САА-Р предоставляет достаточно богатые возможности построения таких конструкций. В частности, в данном случае операцию суммирования и перестановки элементов массива естественно (для большинства языков программирования) реализовать с помощью конструкции типа оператора for.

Построим в общем виде данную широко применяемую алгоритмическую конструкцию, которой воспользуемся для решения рассматриваемой задачи. Известно, что она легко реализуется как частный случай $\tilde{\alpha}$-итерации, т.е. циклической конструкции типа WHILE:

$$I(D)\,{}_{p(D)}\{(\hat{D}')A(\hat{D})S(D)\}, \qquad (6)$$

где $I(D)$ – оператор инициализации переменных цикла;

$p(D) = \alpha(D)$ – условие завершения цикла;

$(\hat{D}')A(\hat{D})$ – тело цикла;

$S(D)$ – оператор, изменяющий переменные цикла.

Построенную конструкцию запишем в виде

$$(I(D);\ P(D);\ S(D))\left\{(\hat{D}')A(\hat{D})\right\}.$$

Далее будем полагать (как вышеотмеченно), что на данном этапе декомпозиции алгоритма любой оператор алгебры реализуют одну элементарную операцию над данными, а предикат вычисляет элементарное бинарное отношение. Введем и традиционно обозначим такие элементарные операторы, выполняющие соответствующие элементарные операции: ":=" – присваивание; "+" – сложение; "++" – инкремент; "%" – остаток от деления и отношения "<" – меньше; "=" – равно.

Будем полагать, что элементы множества обрабатываемых данных Δ – это объекты, обладающие именем и значением, которые мы будем называть переменными и указывать в РС. Кроме того, множество данных содержит константы. Будем полагать, что элементы массивов нумеруются с нуля.

С помощью введенного оператора типа for, запишем РС для случая суммирования элементов массива X в виде:

$$(S)\text{сум}(X) =$$
$$= (i := 0 * S := 0;\ i < k;++i)\ \{S := S + x_i\}.$$

Отметим, что в данном случае для повышения читабельности мы опустили скобки при записи операторов, т.е., записали бинарный оператор $(i) := (0)$ в виде $i := 0$, унарный оператор $++(i)$ в виде $++i$. В последующем изложении будем придерживаться таких обозначений. Кроме того, отметим, что оператор $S:=S+x_i$ реализуется композицией операторов $+ \in U''$ и $:= \in U'$ в соответствии с (5).

Для случая перестановки элементов массива Y на место элементов массива X РС запишем в виде:

$$(X)\textit{перест}(X,Y) =$$
$$= (i := 0; i < k;++i)\{x_i := y_i\}.$$

Операторы (S)сум(Y) и (Y)перест(Y,X) реализуются аналогично, а РС будет выглядеть таким образом:

$$(X,Y,S,i)A(X,Y,S,\mathrm{i},k,n) =$$
$$= \langle k < n\rangle (i := 0 * S := 0; i < k;++i)\{S := S + x_i\} \vee$$
$$\vee\, (i := 0 * S := 0; i < k;++i)\{S := S + y_i\} \vee$$
$$\vee\, (\underset{[}{k\%2 = 0}\ \underset{]}{}(i := 0; i < k;++i)\{x_i := y_i\} \vee$$
$$\vee\, (i := 0; i < k;++i)\{y_i := x_i\})).$$

Заметим, что при вычислении условия $[k\%2 = 0]$ использована операция левого умножения бинарного оператора

$k\%2 \in U''$ на предикат (см. определение (4)).

Приведенный алгоритм внешне напоминает программу, при этом всем алгоритмическим конструкциям соответствуют языковые конструкции. Таким образом алгоритм, очевидно, может быть не только легко переписан на требуемый язык программирования, но и автоматически преобразован в программу на этом языке. Для этого необходимо заменить алгоритмические конструкции соответствующими конструкциями языка программирования. Учитывая, что программа – это форма записи алгоритма в терминах конкретного языка программирования, введем следующее определение.

В рамках применяемого формального аппарата программой на языке РС будем называть алгоритм, степень декомпозиции и набор используемых алгоритмических конструкций которого таковы, что каждой операции и каждому оператору алгоритма можно поставить в соответствие адекватные оператор и операцию некоторого языка программирования.

Важно заметить, несмотря, что в соответствии с приведенным определением, мы получили программу на языке РС, которая остается под "юрисдикцией" формального аппарата. Таким образом, можно воспользоваться возможностями для преобразования программы. Предварительно заметим, что в процессе преобразований используем как общие свойства РС, так и специфические особенности рассматриваемой программы.

Обратив внимание на наличие в программе вложенной α-дизъюнкции, рассмотрим её отдельно и постараемся исключить из программы.

$$[k\%2=0](i:=0;i<k;++i)\{x_i:=y_i\}\vee$$
$$\vee(i:=0;i<k;++i)\{y_i:=x_i\}.$$

Для этого, воспользуемся тем, что используемые циклы однотипны и просто организованы, перейдем от цикла for к циклу WHILE (α–итерация).

В качестве первого шага, учитывая тождественное соотношение вида

$$(I(D);P(D);S(D))\{(\hat{D}')A(\hat{D})\}=$$
$$=I(D)*(E;P(D);S(D))\{(\hat{D}')A(\hat{D})\}= \quad (7)$$
$$=I(D)*(;P(D);S(D))\{(\hat{D}')A(\hat{D})\}$$

вынесем оператор i:=0 из цикла, в результате чего цикл, например, перестановки элементов массива Y на место элементов массива X может быть записан в виде

$$(i:=0;i\le k;++i)\{x_i:=y_i\}=$$
$$=i:=0*(;i<k;++i)\{x_i:=y_i\}.$$

Преобразуем в соответствии с (6) данный цикл в цикл типа while:

$$i:=0*(;i\le k;++i)\{x_i:=y_i\}=$$
$$=i:=0*_{i<k}\{x_i:=y_i*++i\}.$$

Удобным, в полученном варианте цикла, представляется применение операции левого умножения оператора на предикат (см. определение (4)). Однако, в этом случае следует учитывать, что инкремент переменной i будет осуществляться до входа в цикл, а не после выполнения цикла. Для исключения нежелательного эффекта следует предварительно уменьшить эту переменную цикла на единицу и записать следующий его вариант в виде:

$$i:=-1*_{++i<k}\{x_i:=y_i\}.$$

Цикл, переставляющий элементы массива X на место элементов массива Y, работает и записывается аналогично. В результате проведенных преобразований вложенная α-дизъюнкция может быть записана в виде

$$[k\%2=0](i:=-1*_{++i<k}\{x_i:=y_i\}\vee$$
$$\vee i:=-1*_{++i<k}\{y_i:=x_i\}).$$

Далее, воспользовавшись тождественным соотношением вида

$$[\alpha]((\hat{D}')A(\hat{D}')*(\tilde{D}'')B(\tilde{D}')\vee$$
$$\vee(\hat{D}')A(\hat{D}')*(\breve{D}'')C(\breve{D}')=$$
$$=(\hat{D}')A(\hat{D}')[\alpha]((\tilde{D}'')B(\tilde{D}')\vee(\breve{D}'')C(\breve{D}'))$$

вынесем оператор i:=-1 за оператор α-дизъюнкции. Кроме того, применив тождественным соотношением вида:

$$[\beta[\tilde{D}](_{\alpha(D)}\{(D'')A(D')\}\vee_{\alpha(D)}\{(\hat{D}')B(\hat{D})\}=$$
$$=_{\alpha(D)}\{[\beta(\tilde{D})]((D'')A(D')\vee(\hat{D}')B(\hat{D}))\} \quad (8)$$

можно внести оператор α – дизъюнкций

внутрь цикла. Однако на применение данного соотношения накладываются очевидные ограничения вида: $D'' \cup \tilde{D} = \varnothing$, $\hat{D}' \cup \tilde{D} = \varnothing$, обусловленные тем, что в исходном операторе условие $\beta(\tilde{D})$ проверяется однократно, а в результирующем в процессе всего времени работы цикла. В нашем случае эти ограничения удовлетворяются и, таким образом, искомая конструкция может быть записана в виде

$$(i := -1 *_{++i<k} \{[k\%2 = 0](x_i := y_i \vee y_i := x_i\}$$

Перепишем всю программу с учетом полученного результата:

$$(X,Y,S,i)A(X,Y,S,\mathrm{i},k,n) =$$
$$= \langle k < n \rangle (i := 0 * S := 0; i < k; ++i)\{S := S + x_i\} \vee$$
$$\vee (i := 0 * S := 0; i < k; ++i)\{S := S + y_i\} \vee$$
$$\vee (i := -1 *_{++i<k} \{[k\%2 = 0](x_i := y_i \vee$$
$$\vee y_i := x_i\})).$$

Полученная программа теперь не содержит вложенной α-дизъюнкции, но представляет собой оператор α_3–дизъюнкция, которая в настоящее время не поддерживается языками программирования. Учитывая это, приведем α_3 – дизъюнкцию к последовательности α – дизъюнкций:

$$(X,Y,S,i)A(X,Y,S,\mathrm{i},k,n) =$$
$$= [k < n](i := 0 * S := 0; i < k; ++i)\{S := S + x_i\} \vee$$
$$\vee [k > n](i := 0 * S := 0; i < k; ++i)\{S := S + y_i\} \vee$$
$$\vee (i := -1 *_{++i\le k} \{[k\%2 = 0](x_i := y_i \vee$$
$$\vee y_i := x_i\})).$$

Заметим, что операция α_3–дизъюнкция в данном случае преобразована “вручную” для дальнейшего преобразования программы. Однако, эта операция может быть представлена в виде $\alpha_.$ – дизъюнкций автоматически, т.е. в результате трансляции.

После последнего преобразования мы снова получили вложенность α–дизъюнкций, т.е. проблему, решенную на предыдущем шаге преобразований. С учетом некоторых несущественных отличий выполним те же преобразования для устранения вложенности. В частности, вынесем за операцию цикла операторы $S := 0$ и $i := -1$ (в соответствии с (7)) а цикл, суммирующий элементы массива Y, преобразуем к циклу типа while (в соответствии с (6)) . В результате получаем выражение вида

$$(X,Y,S,i)A(X,Y,S,\mathrm{i},k,n) =$$
$$= [k < n](i := 0 * S := 0; i < k; ++i)\{S := S + x_i\} \vee$$
$$\vee [k > n](i := -1 * S := 0 *_{++i<k} \{S := S + y_i\} \vee$$
$$\vee i := -1 *_{++i<k} \{[k\%2 = 0](x_i := y_i \vee$$
$$\vee y_i := x_i\})).$$

Далее аналогично предыдущему случаю внесем оператор $\alpha_.$ – дизъюнкций внутрь цикла в соответствии с (8) и получим выражение:

$$(X,Y,S,i)A(X,Y,S,\mathrm{i},k,n) =$$
$$= [k < n](i := 0 * S := 0; i < k; ++i)\{S := S + x_i\} \vee$$
$$\vee (i := -1 * S := 0 *_{++i<k} \{[k > n](S := S + y_i \vee$$
$$\vee [k\%2 = 0](x_i := y_i \vee y_i := x_i\})).$$

Выполнив точно такие же преобразования над последним сохранившимся неизменным фрагментом программы, получаем программу в окончательном виде:

$$(X,Y,S,i)A(X,Y,S,\mathrm{i},k,n) =$$
$$i := -1 * S := 0 *$$
$$*_{++i<k} \{[k < n](S := S + x_i \vee$$
$$\vee [k > n](S := S + y_i \vee [k\%2 = 0](x_i := y_i \vee$$
$$\vee y_i := x_i)))\}.$$

Полученный в результате преобразований алгоритм существенно лучше (компактнее) исходного, однако, при этом следует сделать такие замечания.

Во-первых, для программы с более сложной структурой такую степень преобразования реализовать, вероятно, не удалось бы.

Во-вторых, преобразования на более высоком уровне декомпозиции алгоритма более эффективны, однако нет оснований пренебрегать такой возможностью на рассматриваемом уровне.

Для реализации трансляции полученной на языке РС программы на требуемый язык программирования необходимо дополнить её указаниями типов переменных. Это можно сделать как в ручном режиме (непосредственно перед трансляцией), так и в процессе разработки РС, записав её, например, в виде

$$
\begin{aligned}
&(float\_X, float\_Y, float\_S, \text{int}\_i)\\
&A(float\_X, float\_Y, float\_S, \text{int}\_i,\\
&\text{int}\_k, \text{int}\_n) = i := -1 * S := 0 *\\
&{}^{*}_{++i<k}\{[k<n](S := S + x_i \vee\\
&\vee [k>n](S := S + y_i \vee\\
&\vee [k\%2 = 0](x_i := y_i \vee y_i := x_i)))\}.
\end{aligned}
$$

Отметим, что данного этапа можно избежать, если целевой язык программирования допускает умолчания (как, например, FORTRAN) или если подобную систему умолчаний принять при разработке транслятора. Так же отметим, что контроль совместимости типов данных и возможные их преобразования мы оставляем “на совести” компилятора с целевого языка программирования.

Заключение

Полученные в работе результаты демонстрируют возможность построить алгоритм, степень детализации которого соответствует уровню языка программирования, т.е., осуществить формализованный переход от алгоритма к программе. При этом сохраняются все возможности формализованного преобразования программы, предоставляемые формальным аппаратом. Достигнутый уровень детализации позволяет осуществить автоматический переход от алгоритма к программе на требуемом языке программирования.

Практическое использование полученных результатов возможно по трем направлениям.

Во-первых, можно за счет разработки и включения в алгоритм специализированных операторов строить язык, ориентированный на решение некоторого класса задач. Например, при регулярном решении класса задач, аналогичных приведенной в качестве примера, следует включить в язык в качестве элементарных операции суммирования и перемещения массивов.

В этом случае, естественно, возникает непростая проблема трансляции созданного языка. Если указанная проблема будет решена, то это явится средством, позволяющим реализовать призыв известного специалиста в области программирования К. Хоар к тому, чтобы отказаться от языков-монстров, а использовать небольшие специализированные языки. Более того, уровень специализации языка может быть достигнут сколь угодно глубокий, т.е. класс решаемых задач может быть сколь угодно узким. Такая специализация может в существенной мере положительно повлиять на качество программных продуктов.

Во-вторых, можно разработать транслятор, осуществляющий перевод непосредственно из РС (с языка регулярных схем) в машинный код. Однако, это весьма затруднительно в силу нетривиальности задачи разработки компилятора.

В-третьих, можно в процессе разработки ориентироваться на конкретный язык программирования и осуществлять трансляцию путем разработки и последующей замены конструкций РС, адекватными языковыми конструкциями. При этом можно повысить читабельность РС, за счет подбора системы обозначений наиболее соответствующих синтаксису целевого языка. В данной работе мы не ориентировались на какой-либо конкретный язык программирования, хотя влияние языка Си, заметно.

Так как последнее направление наиболее просто в реализации, в качестве ближайшей перспективы в дальнейшей работе назовем реализацию рассмотренных возможностей для конкретных приложений и для конкретного языка программирования.

1. *Акуловский В.Г.* Расширенная алгебра алгоритмов // Проблеми програмування. – 2007. – № 3. – С. 3 – 15.
2. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Алгебра. Языки. Программирование. – Киев: Наук. думка, 1978. – 319 с.
3. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.*. Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 634 с.
4. *Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П., Терзян Т.К.* Многоуровневое структурное проектирование программ: Теоретические основы, инструментарий. – М.: Финансы и статистика, 1989. – 208 с.
5. *Брусенцов Н.П., Владимирова Ю. С.* Компьютеризация булевой алгебры // Доклады Академии наук, 2004. – Т. 395. – № 1.
6. *Брусенцов Н.П.* Кибернетика – ожидания и результаты. Политехнические чтения. – М.: Знание, 2002. – Вып. 2. – С. 104 – 105.
7. *Поспелов Д.А.* Логические методы анализа и синтеза схем. Изд. 3-е, перераб. и доп. – М.: Энергия, 1974. – 368 с.

Получено 01.11.2007

Об авторе:

Акуловский Валерий Григорьевич,
кандидат технических наук, доцент кафедры информационных систем и технологий.

Место работы автора:

Академия таможенной службы Украины,
49000, Днепропетровск, ул. Дзержинского 2\4.
Тел/факс.: (0562) 45 5596;
тел.: (0562) 67 5004;
моб.: тел. 8050–9410566.
e-mail: akulovski@rambler.ru
e-mail: academy@amsu.dnp.ukrpack.net