УДК 519.683

В.Г. Акуловский

НЕКОТОРЫЕ ПОДХОДЫ К КОНТРОЛЮ И ПРЕОБРАЗОВАНИЮ АЛГОРИТМОВ НА ОСНОВЕ АНАЛИЗА СПЕЦИФИЦИРУЕМЫХ ДАННЫХ

Предложены подходы к контролю корректности алгоритмов на основе анализа специфицируемых данных и подходы к оптимизации последовательных алгоритмов в результате преобразования потоков данных.

Роль данных в процессе разработки алгоритмов и программ чрезвычайно высока, что подтверждается, например, следующей цитатой [1]: “Хотя операции и данные, эти активные и пассивные программные компоненты, играют совершенно различные роли в каждой конкретной программе, совершенно невозможно рассматривать одни в отрыве от других”. Необходимость совместной разработки потоков управления и данных давно осознана программистским сообществом. Это осознание нашло свое выражение в большинстве методологий программирования, в которых используются специальные средства для описания данных. В большинстве случаев, однако, это описание, выполненное в графической форме, представляет собой отдельный документ в пакете документов, сопровождающих разработку. Это, очевидно, может служить источником ошибок, связанных с неадекватностью различных документов, что особенно вероятно при большом их объеме.

В данной работе будем рассматривать совместно (в рамках единого формального аппарата) потоки управления и данных и на основании этого подхода решать некоторые аспекты задач построения, контроля корректности и преобразования алгоритмов. Для этого воспользуемся расширенной алгеброй алгоритмов (САА-Р) [2], которая является одним из направлений развития известной алгебры алгоритмов Глушкова [3], и результатами, полученными в [4], где выполнена формализация взаимосвязи операторов и данных и предусмотрена возможность спецификации данных для каждого оператора.

Используемый формальный аппарат представляет собой систему алгоритмических алгебр – $\langle U, L, \Omega \rangle$, где U – множество операторов, L – множество логических условий, Ω – сигнатура операций, состоящая из логических операций – Ω_1, принимающих значения на множестве L и операций – Ω_2, принимающих значения на множестве операторов U. Операции множества Ω_2 это * – композиция (последовательное выполнение операторов), α_3 – дизъюнкция (разветвление вычислительного процесса по трем направлениям), α – итерация (цикл, соответствует программной конструкции WHILE) и их производные.

Известно, что основным подходом к разработке алгоритмов является его декомпозиция, осуществляемая с понижением уровня абстракции на каждом очередном этапе разработки. Этот подход реализован в виде метода структурного проектирования программ (МСПП) [5], формальной основой которого является алгебра алгоритмов.

Поскольку САА-Р, являющаяся средством, определяющим потоки управления, расширена за счет введения в рассмотрение данных, то в рамках МСПП будем анализировать возможности построения алгоритма с учетом общих свойств данных, обрабатываемых этим алгоритмом. Под контролем корректности алгоритма будем понимать контроль ограничений, налагаемых на специфицируемые данные.

В связи с тем, что в данной работе мы коснемся только некоторых аспектов и

этапов разработки алгоритмов, будем при его построении использовать только одну из возможных операций САА-Р – композицию, подразумевая, что любой оператор, образующий алгоритм, может представлять собой и/или содержать любые операций САА-Р.

1. Проектирование структуры и контроль корректности алгоритма

Прежде, чем рассматривать структуру алгоритмов, определим основные используемые далее понятия.

Самым общим случаем оператора алгебры будем считать алгоритм решения задачи (в дальнейшем алгоритм), который определим таким образом.

Определение 1. Произвольным алгоритмом назовем оператор $A \in U$, который запишем в виде

$$(\Delta')A(\Delta''), \qquad (1)$$

где Δ' – спецификация данных, образующих область определения, Δ'' – спецификация данных, образующих область значений оператора, $\Delta = \Delta' \cup \Delta''$ – множество всех данных, доступных алгоритму, которое является его информационным множеством. Область определения будем называть входными, а область значений выходными данными. Будем говорить, что в результате выполнения алгоритма, решается некоторая задача, а вычислитель переходит из состояния Δ' в состояние Δ''.

Поскольку вычислитель не в состоянии выполнить переход из состояния Δ' в состояние Δ'' за один “шаг”, то его (этот переход) необходимо реализовать в виде некоторого количества элементарных, выполняемых вычислителем “шагов”, т.е. записать алгоритм в виде программы на некотором языке программирования.

Для получения программы, адекватной исходному алгоритму, необходимо осуществлять его поэтапную (поуровневую) декомпозицию, т.е. необходимо представить его в виде уровней со множеством связанных и согласованных компонентов (модулей) на каждом из них. При чём степень абстрактности (детализации) на каждом следующем уровне декомпозиции необходимо понижать (повышать).

В качестве вышеупомянутых компонент будем использовать операторы САА-Р, которые в данном случае определим следующим образом.

Определение 2. Операторы, с помощью которых строится алгоритм, это операторы вида:

$$(D)A(D') \in U, \qquad (2)$$

у которых $D \in \Delta$ – область определения (входные данные), а $D' \in \Delta$ – область его значений (выходные данные). Эти операторы обрабатывают некоторое подмножество входных данных $D \subset \Delta$, изменяют их и порождают новые данные, формируя таким образом подмножество выходных данных $D' \subset \Delta$. Для множеств D и D' выполнимо любое из следующих соотношений: $D \subseteq D', D' \subseteq D, \quad D \cap D' = \varnothing$.

С целью повышения строгости изложения приведем следующую аксиому.

Аксиома 1. Любой алгоритм и любой входящий в него оператор (за исключением элементарных) может быть представлен в виде композиции (последовательности) двух (или более) операторов $(D)A(D') = (\hat{D})A'(\hat{D}') * (\tilde{D})A''(\tilde{D}')$, для которых выполняются такие соотношения: $D = \hat{D} \cup \tilde{D}$, $D' = \hat{D}' \cup \tilde{D}'$.

Очевидно, существование подобной аксиомы подразумевалось при создании различных методов и методологий программирования, основанных на декомпозиции алгоритмов. Однако, насколько известно автору, в явном виде она не формулировалась.

Запись оператора в форме, приведенной в аксиоме 1, будем называть регулярной схемой (РС). Оператор, стоящий справа от знака равенства – исходным, а операторы, расположенные слева – производными.

В соответствии с аксиомой 1, и расширяя возможности МСПП, будем согласованно декомпозировать операторы и обрабатываемые ими данные. В результате каждого шага декомпозиции получим новый уровень алгоритма, представляющий собой множество регулярных схем (РС), каждая из которых описывает один

из операторов предыдущего уровня. Таким образом, алгоритм будет представлять собой суперпозицию операторов.

Алгоритм (1) будем называть нулевым уровнем декомпозиции и будем полагать, что на этом уровне специфицированы глобальные входные и выходные данные.

При рассмотрении процесса декомпозиции алгоритма будем анализировать свойства специфицируемых данных и формулировать ограничения на их использование.

В общем виде (для любого уровня декомпозиции) любой оператор (2) представляет собой РС вида:

$$(D_m^i)A_m^i(D_m'^i) = (D_1^i)A_1^{i+1}(D_1'^i) * ... *$$
$$*(D_j^i)A_j^{i+1}(D_j'^i) * ... * (D_n^i)A_n^{i+1}(D_n'^i), \quad (3)$$

где верхний индекс соответствует номеру уровня, а нижний идентифицирует оператор внутри РС и обрабатываемые им данные. Ограничение на обрабатываемые данные в соответствии с аксиомой 1 для этого уровня запишем в виде

$$D_m^i = D_1^i \cup ... \cup D_j^i \cup ... \cup D_n^i, \quad (4)$$

$$D_m'^i = D_1'^i \cup ... \cup D_j'^i \cup ... \cup D_n'^i. \quad (5)$$

Анализируя данные на выходе некоторого оператора, можно заметить, что некоторые из них – промежуточные, т.е. не влияют на результат решения задачи, специфицируемой в рамках данной РС, а потребуются на следующем шаге декомпозиции. И естественным является желание отложить их спецификацию (уменьшить объем специфицируемых данных) до того этапа разработки, на котором эти данные будут востребованы. Исполнению этого препятствуют жесткие ограничения аксиомы 1. Для ослабления упомянутого ограничения разобьем выходные данные на два подмножества $D_i'^i = D_i''^i \cup {}^{\text{пром}}D_i^i$ и $D_i''^i$ будем называть выходными, а ${}^{\text{пром}}D_i^i$ – промежуточными данными. Для этого случая РС (3) перепишем в виде

$$(D_m^i)A_m^i(D_m'^i) = (D_1^i)A_1^{i+1}(D_1''^i, {}^{\text{пром}}D_1^i) * ...$$
$$...*(D_i^i)A_i^{i+1}(D_i''^i, {}^{\text{пром}}D_i^i) * ... *$$
$$*(D_n^i)A_n^{i+1}(D_n''^i, {}^{\text{пром}}D_n^i),$$

а ограничение (5) примет вид

$$D_m'^i = D_1''^i \cup ... \cup D_j''^i \cup ... \cup D_n''^i. \quad (6)$$

В результате количество специфицируемых для исходного оператора данных сократится без потерь в полноте спецификации.

Далее определим понятия глобальных и локальных данных, которые обозначим соответственно ${}^{\text{гл}}D_j^i$ и ${}^{\text{л}}D_j^i$. Будем полагать, что ${}^{\text{гл}}D_j^i$ доступны всем операторам РС и в выражении (3) специфицированы именно глобальные данные. Локальные данные ${}^{\text{л}}D_j^i$ локализованы в рамках каждого оператора, входящего в РС. На введенные типы данных наложено ограничение

$${}^{\text{гл}}D_j^i \cap {}^{\text{л}}D_j^i = \varnothing, \quad (7)$$

для любых i и j.

Когда на некотором i+1-ом уровне декомпозиции специфицируются локальные данные, то выражение (3) необходимо записать в виде

$$(D_m^i)A_m^i(D_m'^i) = ({}^{\text{гл}}D_1^i, {}^{\text{л}}D_1^{i+1})A_1^{i+1}({}^{\text{гл}}D_1'^i) * ... *$$
$$*({}^{\text{гл}}D_j^i, {}^{\text{л}}D_j^{i+1})A_j^{i+1}({}^{\text{гл}}D_j'^i) * ... \quad (8)$$
$$...*({}^{\text{гл}}D_n^i, {}^{\text{л}}D_n^{i+1})A_n^{i+1}({}^{\text{гл}}D_n'^i)$$

и для данного случая имеет место следующее ограничение:

$${}^{\text{л}}D_1^{i+1} \cap ... \cap {}^{\text{л}}D_j^{i+1} \cap ... \cap {}^{\text{л}}D_n^{i+1} = \varnothing. \quad (9)$$

На следующем i+2-ом уровне оператор (8) будет записан в виде

$$({}^{гл}D_j^i, {}^{л}D_j^{i+1})A_j^{i+1}({}^{гл}D'^i_j) =$$
$$= ({}^{гл}D_1^{i+1}, {}^{л}D_1^{i+2})A_1^{i+2}({}^{гл}D'^{i+1}_1)*...$$
$$...*({}^{гл}D_j^{i+1}, {}^{л}D_j^{i+2})A_j^{i+2}({}^{гл}D'^{i+1}_j)*...$$
$$...*({}^{гл}D_n^{i+1}, {}^{л}D_n^{i+2})A_n^{i+2}({}^{гл}D'^{i+1}_n),$$

где ${}^{гл}D_j^{i+1} = {}^{гл}D_j^i \cup {}^{л}D_j^{i+1}$, т. е. локальные на i+1-ом уровне данные рассматриваются как глобальные на i+2 уровне и для ${}^{гл}D_j^{i+1}$ выполняется соотношение (4).

Поскольку дальнейшее рассмотрение будет вестись в рамках одной РС, а их сочетания рассматриваться не будут, упростим систему обозначений, отказавшись от указания глобальности и локальности данных и лишней индексации, в тех случаях, когда в ней нет необходимости.

Для продолжения декомпозиции алгоритма будем детализовать спецификации входных и выходных данных, а РС запишем в виде

$$(D)A(D') = (D_1)A_1(D'_1)*...*$$
$$*(D_j)A_j(D'_j)*...*(D_k)A_k(D'_k)$$

Представим входные данные в виде $D_i = {}^{исх}D_i \cup {}^{пр}D_i$, где ${}^{исх}D_i$ – исходные, а ${}^{пр}D_i$ – производные данные,

на которые накладывается ограничение вида:

$$ {}^{исх}D_i \cap {}^{пр}D_i = \varnothing \qquad (10)$$

и которые в исходном состоянии могут быть как определены (инициализированы), так и находиться в неопределенном (неинициализированном) состоянии. В пос–леднем случае такие неопределенные данные должны быть определены в процессе выполнения алгоритма.

${}^{пр}D_i$ - множество производных данных, которые изменяются в процессе выполнения алгоритма.

Уточняя структуру исходных данных ${}^{исх}D_i$, представим его в виде двух подмножеств ${}^{исх}D_i = {}^{const}D_i \cup {}^{иниц}D_i$, где

${}^{const}D_i$ – множество неизменяемых данных, которые инициируются до начала выполнения алгоритма и в дальнейшем остаются неизменными, и все они используются (должны использоваться) в качестве исходных (входных) данных, т.е.

$${}^{const}D_1 \cup ... \cup {}^{const}D_i \cup ... \cup {}^{const}D_n \subseteq$$
$$\subseteq D_1 \cup ... \cup D_i \cup ... \cup D_k;$$

${}^{иниц}D_i$ – однократно инициализируются в процессе выполнения алгоритма и в дальнейшем остаются неизменными, и все они используются (должны использоваться) в качестве исходных (входных) данных, т.е.

$${}^{иниц}D_1 \cup ... \cup {}^{иниц}D_i \cup ... \cup {}^{иниц}D_n \subseteq$$
$$\subseteq D_1 \cup ... \cup D_i \cup ... \cup D_k.$$

Множество выходных данных представим в виде объединения следующих множеств: $D'_i = {}^{пр}D_i \cup {}^{иниц}D_i$.

Далее введем вспомогательные множества:

${}^{вв}D_i$ – вводимые данные (поступающие с устройств ввода);

${}^{выв}D_i$ – выводимые (на внешние устройства) данные.

Эти множества обладают такими свойствами: ${}^{вв}D_i \subset {}^{иниц}D_i \cup {}^{пр}D_i$, ${}^{выв}D_i \subseteq D_i \cup D'_i$.

Учитывая появление этих типов данных на множестве U выделим два подмножества операторов, элементами которых являются $(УВВ)R_i({}^{вв}D_i) \in U$ – операторы ввода, где УВВ – спецификация устройства ввода и $({}^{выв}D_i)W_i(УВыв) \in U$ – операторы вывода, где УВыв – спецификация устройства вывода.

Теперь на основании приведенных свойств введем ограничения на использование построенных множеств.

Для множества ${}^{const}D_i$ имеют место такие ограничения:

$$ {}^{вв}D_i \cap {}^{const}D_i = \varnothing, \qquad (11) $$

т.е. вводимые данные не изменяют констант;

$$ {}^{const}D_1 \cup ... \cup {}^{const}D_i \cup ... \cup {}^{const}D_n \cap \cap D'_1 \cup ... \cup D'_i \cup ... \cup D'_k = \varnothing, \qquad (12) $$

т. е. константы не могут выступать в качестве выходных данных.

Для ${}^{иниц}D_i$ имеют место следующие ограничения:

$\exists(D_i)A(D'_i)$ и только один, для которого выполняется соотношение:

$$ {}^{иниц}D_i \subseteq D'_i; \qquad (13) $$

$\exists(УВВ)R_i({}^{вв}D_i)$ и только один, для которого выполняется соотношение:

$$ {}^{вв}D_i \subseteq {}^{иниц}D_i, \qquad (14) $$

т.е., в случаях (13, 14) данные инициализируются однократно.

Для ${}^{пр}D_i$ имеют место следующие ограничения:

$\exists(D_i)A(D'_i)$ по крайней мере один, для которого выполняется соотношение

$$ {}^{пр}D_i \subseteq D'_i \qquad (15) $$

и он предшествует любому другому оператору для которого выполняется соотношение ${}^{пр}D_i \subseteq D_i$, т.е. использованию данных должна предшествовать их инициализация.

Приведенные ограничения находятся в очевидных соотношениях, некоторые из них приведем в качестве примера.

Если ${}^{иниц}D_i \subseteq {}^{вв}D_i$, то ограничение (13) становится не актуальным, а если ${}^{иниц}D_i \subseteq D'_i$, то не актуально ограничение (14).

Если ${}^{вв}D_i \subset {}^{иниц}D_i$, то ${}^{вв}D_i \cap {}^{пр}D_i \neq \varnothing$.

Если ${}^{вв}D_i \cap {}^{иниц}D_i \neq \varnothing$, то ограничение (13) выполняется для множества ${}^{иниц}\hat{D}_i = {}^{иниц}D_i / {}^{вв}D_i$.

Если ${}^{пр}D_i \subseteq {}^{вв}D_i$, то ограничение (15) становится не актуальным.

Если ${}^{вв}D_i \subseteq {}^{пр}D_i$, то ${}^{вв}D_i \cap {}^{иниц}D_i = \varnothing$.

Если ${}^{вв}D_i \cap {}^{пр}D_i \neq \varnothing$, то ограничение (15) выполняется для множества ${}^{пр}\hat{D}_i = {}^{пр}D_i / {}^{вв}D_i$ и т.д.

В первом разделе работы рассмотрена возможность на некоторых этапах разработки согласовывать процесс декомпозиции операторов, образующих алгоритм, с декомпозицией специфицированных для этих операторов данных.

В процессе рассмотрения предложены некоторые варианты детализации данных и выявлены ограничения на их использование. В случае спецификации предложенных типов данных на требуемом уровне декомпозиции открывается возможность проверки соответствия алгоритма сформулированным ограничениям и, таким образом, контроля его корректности. При этом на начальных этапах контролируются ограничения (4–7, 9), а на последующих, после детализации данных, ограничения (10–15) с учетом связывающих их соотношений. Заметим, что контроль может быть осуществлен как в ручном, так и в автоматизированном режиме.

2. Преобразование алгоритмов

Структура алгоритма (последовательность выполнения операторов) в существенной мере определяется последовательностью обработки данных и должна учитывать тип разрабатываемой программной системы. В данном случае, имея в виду последовательные алгоритмы, рассмотрим потоки управления и определим потоки данных в алгоритмах.

Будем полагать, что потоки управления, как вышеотмечено, организуются операциями САА-Р (в рассматриваемом случае операцией композиция), и пару операторов

$(D_i)A_i(D'_i)*(D_{i+1})A_{i+1}(D'_{i+1})$, будем называть непосредственно, а $...*(D_j)A_j(D'_j)*...*(D_k)A_k(D'_k)$, где $k > j$ опосредованно связанными по управлению.

Понятие поток данных, определим следующим образом.

Определение 3. Поток данных – это такое множество данных D, которое доступно (специфицировано на входе) некоторому оператору A_i и оно же после обработки этим оператором (специфицировано на его выходе) становится доступно некоторому оператору A_j $(i < j)$ и т. д., т. е. множество данных D трансформируется в процессе обработки последовательностью операторов, в рамках РС. Будем говорить, что данные с выхода предшествующего оператора подаются на вход последующего и/или последующих операторов.

Заметим, что поток данных в последовательном алгоритме носит виртуальный характер, однако такая абстракция оказывается весьма удобной при совместном рассмотрении потоков управления и обрабатываемых этим потоком данных.

Требование к потокам данных в последовательных алгоритмах, удовлетворение которого обеспечит более высокое качество, а именно более высокий уровень понятности и отлаживаемости алгоритма (программы), сформулируем следующим образом.

Количество потоков данных на любом уровне декомпозиции и в любой РС должно быть минимальным. Очевидно, что оптимальным, но крайне редко реализуемым, является алгоритм с единственным потоком данных.

В этом разделе будем решать задачу преобразования РС, с целью удовлетворения сформулированного требования и докажем возможность такого преобразования при некоторых ограничениях. Для решения этой задачи, воспользовавшись результатами, полученными в [4], и развивая их, введем понятие независимых по данным и связанных данными операторов.

Будем различать “прямые” и “обратные” связи по данным и прямую связь определим так.

Определение 4. В регулярной схеме вида

$$(D)A(D') = (D_1)A_1(D'_1)*...*(D_j)A_j(D'_j)*...*(D_k)A_k(D'_k)$$

операторы $(D_i)A_i(D'_i)$ и $(D_j)A_j(D'_j)$, где $i < j$ **прямо связаны по данным** (в дальнейшем прямо связаны), если они непосредственно или опосредованно связаны по управлению и для них выполняется следующее соотношение: $D'_i \cap D_j \neq \varnothing$, т. е. область значений оператора $(D_i)A_i(D'_i)$ пересекается с областью определения оператора $(D_j)A_j(D'_j)$. При наличии такой связи оператор $(D_i)A_i(D'_i)$ всегда должен предшествовать оператору $(D_j)A_j(D'_j)$.

Под длиной прямых связей (расстоянием между прямо связанными операторами) будем понимать количество операторов, разделяющих связанные операторы и, таким образом, длина связи, которую обозначим d, вычисляется по формуле $d = j - i - 1$.

Особого рассмотрения заслуживает случай, когда длина связи $d = 0$, т. е. случай, когда операторы непосредственно связаны по управлению. Для этого случая введем следующее определение.

Определение 5. Любые два оператора $(D_i)A_i(D'_i)$ и $(D_j)A_j(D'_j)$ назовем **непосредственно связанными по данным** (в дальнейшем непосредственно связанными), если $D'_i \cap D_j \neq \varnothing$, а расстояние между ними равно нулю, т. е. $j = i + 1$. Непосредственная связь определяет жестко заданную последовательность обработки данных и эта последовательность выполнения операторов не может быть изменена.

Теперь определим обратную связь.

Определение 6. Между любыми операторами $(D_i)A_i(D'_i)$ и $(D_j)A_j(D'_j)$ присутствуют **обратные связи**, трех видов, определяемые следующими соотношениями:

1) $D_i \cap D_j \neq \varnothing$, 2) $D_i \cap D'_j \neq \varnothing$,

3) $D'_i \cap D'_j \neq \varnothing$, которые могут присутствовать в любом сочетании.

На этом основании введем ещё несколько производных понятий, которыми воспользуемся в дальнейшем.

Определение 7. Многосвязным назовем оператор, который прямо связан (см. определение 4) с несколькими другими операторами, при наличии или в отсутствии непосредственной связи (см. определение 5).

Определение 8. Независимыми по данным (в дальнейшем **независимыми**) назовем любые два непосредственно связанных по управлению оператора $(D)A(D')*(\hat{D})B(\hat{D}')$, если для них выполняются следующие ограничения: $D \cap \hat{D} = \varnothing$, $D \cap \hat{D}' = \varnothing$, $D' \cap \hat{D} = \varnothing$, $D' \cap \hat{D}' = \varnothing$. Очевидно, что для них выполняется тождественное соотношение: $(D)A(D')*(\hat{D})B(\hat{D}') = (\hat{D})B(\hat{D}')*(D)A(D')$.

Определение 9. Полностью независимым назовем оператор, если он независим от любого оператора входящего в РС, и таким образом он может располагаться в любом месте РС.

Прежде чем рассматривать возможные преобразования алгоритмов, введем ещё одну аксиому, по аналогии аксиомы 1.

Аксиома 2. Любая пара операторов непосредственно связанная по управлению (операцией композиции) может рассматриваться, как новый оператор $(\hat{D})A'(\hat{D}')*(\tilde{D})A''(\tilde{D}') = (D)A(D')$, для которых выполняются соотношения $D = \hat{D} \cup \tilde{D}$ и $D' = \hat{D}' \cup \tilde{D}'$.

Приведенные в работе аксиомы – базовые и универсальные средства преобразования алгоритмов, так как позволяют создавать новые операторы за счет разбиения и объединения существующих. В частности, возможно сокращение количества прямых связей в многосвязных операторах (см. определение 7) за счет разбиения такого оператора на несколько других операторов с меньшим числом прямых связей. Однако возможность изменения порядка следования операторов в них не предусмотрена.

Для того, что бы с целью преобразования алгоритмов иметь возможность менять порядок следования операторов в РС, рассмотрим непосредственно связанные операторы и ослабим жесткое ограничение, заданное определением 5. Для этого рассмотрим возможные случаи преобразования операторов

$$(D)A(D')*(\hat{D})B(\hat{D}'), \qquad (16)$$

когда для них выполняются следующие соотношения.

1. Рассмотрим общий случай непосредственной связи, когда выполняется $D' \cap \hat{D} = \bar{D}$, т. е. область значений оператора $(D)A(D')$ пересекается с областью определения оператора $(\hat{D})B(\hat{D}')$. Для даного случая выполним такую последовательность действий.

Оператор $(D)A(D')$ представим (в соответствии с аксиомой 1) в виде двух операторов $(D)A(D') = (D_1)A_1(D'_1)*(D_2)A_2(D'_2)$, где $D'_1 \cap D_2 = \varnothing$, $D'_2 \cap \hat{D} \neq \varnothing$. В свою очередь оператор $(\hat{D})B(\hat{D}')$ представим в виде $(\hat{D})B(\hat{D}') = (\hat{D}_1)B_1(\hat{D}'_1)*(\hat{D}_2)B_2(\hat{D}'_2)$, таким образом, что $\hat{D}'_1 \cap \hat{D}_2 = \varnothing$, $\hat{D}_1 \cap D'_2 \neq \varnothing$. В результате преобразования получены два непосредственно не связанных оператора $(D_1)A_1(D'_1)$, $(\hat{D}_2)B_2(\hat{D}'_2)$ и два непосредственно связанных оператора $(D_2)A_2(D'_2)$, $(\hat{D}_1)B_1(\hat{D}'_1)$. Объединив последние, в соответствии с аксиомой 2, получим три непосредственно не связанных оператора, т. е.

$$(D)A(D')*(\hat{D})B(\hat{D}') = (D_1)A_1(D'_1)* \\ *(\tilde{D})AB(\tilde{D}')*(\hat{D}_2)B_2(\hat{D}'_2) \qquad (17)$$

для которых выполняется $D'_1 \cap \tilde{D} = \varnothing$ и $\tilde{D}' \cap \hat{D}_2 = \varnothing$.

Далее рассмотрим частные случаи непосредственной связи операторов, выполняя аналогичные преобразования.

2. Когда $\hat{D} \subset D'$, последовательность операторов (16) преобразуем с виду

$$(D)A(D')*(\hat{D})B(\hat{D}') = (D'')AB(D''')*(D_2)B_2(D'_2), \quad (18)$$

где $D''' \cap D_2 = \varnothing$.

3. В случае, когда $D' \subset \hat{D}$, последовательность операторов (16) преобразуем к виду:

$$(D)A(D')*(\hat{D})B(\hat{D}') = (D_1)A_1(D'_1)*(\tilde{D})AB(\tilde{D}'), \quad (19)$$

где $D'_1 \cap \tilde{D} = \varnothing$.

4. Когда $\hat{D} = D'$, (20) преобразование с точки зрения непосредственной связи невозможно. В результате нами рассмотренной возможности выделения непосредственно не связанных операторов, проанализируем возможные обратные связи между операторами $(D)A(D')*(\hat{D})B(\hat{D}')$ непосредственно связанными по управлению и непосредственно не связанными по данным. Для этого рассмотрим все возможные варианты всех видов обратных связей (см. определение 6):

1.	2.	3.
$D = \hat{D}$	$D = \hat{D}'$	$D' = \hat{D}'$
$D \subset \hat{D}$	$D \subset \hat{D}'$	$D' \subset \hat{D}'$
$\hat{D} \subset D$	$\tilde{D}' \subset \hat{D}$	$\hat{D}' \subset D'$
$D \cap \hat{D} = \varnothing$	$D \cap \hat{D}' = \varnothing$	$D' \cap \hat{D}' = \varnothing$

Из приведенных соотношений легко увидеть, что во всех случаях и во всех возможных сочетаниях этих случаев (за исключением одного) операторы полностью или частично совместно (в два этапа) обрабатывают данные. Иначе, два оператора решают единую задачу (связанные задачи) обработки данных.

Исходя из вышеизложенного, следует вывод.

Последовательность двух операторов, имеющая обратные (обратную) и не имеющая непосредственную связь некорректна (по крайней мере, неэффективна), так как в этом случае необоснованно “разрывается” поток данных (см. определение 3). В единственном случае отсутствие непосредственной связи оправдано, когда выполняются три условия $D \cap \hat{D} = \varnothing$, $D \cap \hat{D}' = \varnothing$, $D' \cap \hat{D}' = \varnothing$, в результате выполнения которых в соответствии с определением 8 получаем независимые операторы.

Учитывая этот вывод очевидно, что рассмотренные случаи преобразования операторов (17–19), с целью устранения непосредственной связи, имеет практический смысл только в том случае, когда в результате такого преобразования получены независимые операторы. То есть на преобразования (17–19), накладывается достаточно жесткое ограничение, при удовлетворении которого, однако, можно выбирать произвольную последовательность выполнения этих операторов. Заметим, что в случае (20) пара операторов очевидно должна быть преобразована в один.

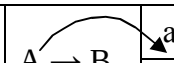
Уточняя выше сформулированное требование к последовательному алгоритму, можно сказать, с учетом приведенных определений, что в РС должно быть минимальное количество прямых связей. Докажем возможность таких преобразований РС, в результате которых число прямых связей будет уменьшено.

Теорема. В любой регулярной схеме алгоритма, состоящей из трех или более операторов, связанных непосредственными и прямыми связями, количество прямых связей может быть сокращено за счет их преобразования в непосредственные связи.

Доказательство. Рассмотрим тройку операторов $(D)A(D')*(\hat{D})B(\hat{D}')*(\hat{D})C(\hat{D}')$, со всеми возможными в такой тройке прямыми и непосредственными связями. Для наглядности и краткости изложения представим эти связи и возможные результаты преобразования этой последовательности опе-

раторов в таблице, с указанием использованных случаев преобразования. При этом будем использовать упрощенную систему обозначений, где прямые стрелки указывают непосредственные связи как по управлению, так и по данным, дугами прямые связи, а “*” – непосредственные связи по управлению. Опустим так же спецификации данных.

Таблица

1		а	$AB \rightarrow C$	
		б	$A_1 * AB \rightarrow C$	(16)
		в	$B_2 * AB \rightarrow C$	(15)
		г	$A_1 * B_2 * AB \rightarrow C$	(14)
2		а	$A \rightarrow BC$	
		б	$A \rightarrow BC * B_1$	(16)
		В	$A \rightarrow BC * C_2$	(15)
		Г	$A \rightarrow BC * B_1 * C_2$	(14)
3		А	$A \rightarrow BC$	
		Б	$AB \rightarrow C$	
		В	$A_1 * AB \rightarrow C$	(16)
		Г	$A \rightarrow BC * B_1$	(16)
		д	$B_2 * AB \rightarrow C$	(15)
		е	$A \rightarrow BC * C_2$	(15)
		ж	$A_1 * B_2 * AB \rightarrow C$	(14)
		з	$A \rightarrow BC * B_1 * C_2$	(14)

Из приведенной таблицы видно, что для каждого возможного случая связи между операторами в результате использования аксиомы 2 и соотношений (17–19) (в случае, когда такие преобразования возможны, и вышеприведенное ограничение удовлетворяется) получены результаты преобразований, в которых прямые связи преобразованы в непосредственные и, таким образом число прямых связей сокращено. Отметим, что в случае получения в результате преобразования полностью независимого оператора (см. определение 9), приведенные соотношения так же выполняются.

Возьмём случай, когда количество операторов в РС больше трёх и/или новые операторы получены в результате преобразований, в частности, в результате разбиения многосвязных операторов. В этом случае при условии выполнения требования по наличию непосредственных и прямых связей, все возможные сочетания, приведены в таблице, всегда можно выбрать тройку операторов, к которым применимы какие-либо из предлагаемых преобразований. В результате реализации выбранных способов преобразования количество прямых связей будет сокращено (заменено непосредственными связями). Теорема, таким образом, доказана.

Описанный в данной работе подход, который намечает пути и возможности преобразования алгоритма, не предлагает конкретных решений. В частности разбиение, объединение и построение новых операторов, а так же выбор способов преобразования их последовательностей, существенно зависит от решаемой алгоритмом задачи и требует творческого подхода.

Заключение

В предлагаемой работе предложен подход к построению и контролю алгоритмов на основании анализа, и детализации специфицируемых данных. В частности, сформулированы свойства и ограничения на используемые данные, проверка которых и является средством контроля. Полнота предлагаемой формы контроля зависит от полноты набора ограничений на обработку данных и дальнейшая детализация данных, очевидно, позволит расширить представленный набор.

Отметим, что в данной работе не рассмотрена ситуация, когда глобальные или локальные данные одного из уровней декомпозиции могут быть востребованы на другом уровне. И хотя такая ситуация является нежелательной, что отмечалось многими авторами (например [1]) и её следует избегать, тем не менее такая методологическая проблема существует. Рассмотрение этой самостоятельной задачи автор намерен осуществить в дальнейшем.

Во втором разделе определено понятие потоков данных в последовательных алгоритмах. Определены различные виды связей в операторах, образующих РС, и предложен подход к преобразованию алгоритмов на основе введенных аксиом и преобразования непосредственно связанных операторов. Доказана возможность сокращения количества прямых связей в последовательных алгоритмах.

Дальнейшее развитие данного подхода может состоять в анализе других типов связей и построении методов преобразования алгоритмов на основе такого анализа.

Перспективным представляется использование потоков данных при построении и преобразовании параллельных алгоритмов. Однако, с этими алгоритмами дело обстоит существенно сложнее, так как требуемая степень их параллелизма определяется не только решаемой задачей, а возможностями, ограничениями аппаратной и операционной среды.

На основании результатов, полученных в данной работе можно сформулировать только некоторые соображения. В частности, в том случае, когда в процессе разработки получен полностью независимый оператор (см. определение 9), то он естественно может выполняться параллельно (псевдопараллельно) с любым другим оператором РС, которой он принадлежит, и в этом случае не требуются дополнительные усилия для распараллеливания алгоритма. Независимый оператор может выполняться параллельно с соседним, но при этом потребуется их синхронизовать по управлению. Для связанных операторов параллельное выполнение сопряжено с синхронизацией по совместно обрабатываемым данным. Несмотря на то, что задача разработки и преобразования параллельных алгоритмов в данной работе не решалась, тем не менее, с точки зрения автора, предложенный подход с использованием потоков данных является перспективным для решения названной задачи. И это является ещё одним направлением развития полученных результатов.

1. *Турский В.* Методология программирования. – М.: Мир, 1981. 264 с.
2. *Акуловский В.Г.* Расширенная алгебра алгоритмов//Проблеми програмування.- 2007. – № 3. С. 3–15.
3. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академперіодика, 2007. – 634 с.
4. *Акуловский В.Г.* Формализация взаимосвязей операторов и данных в рамках расширенной алгебры алгоритмов // Кибернетика и системный анализ. – 2008. – № 6.
5. *Цейтлин Г.Е., Бакулин А.В.* Многоуровневые структурированные проекты программ и их обоснование // Кибернетика и системный анализ. – 1991. – № 5. – С. 98–107.

Получено 03.10.2008

Об авторе:

Акуловский Валерий Григорьевич,
кандидат технических наук, доцент кафедры информационных систем и технологий
Тел. 8 050 941 0566;
e-mail: valeryakulovskiy@rambler.ru

Место работы автора:

Академия таможенной службы Украины.
49000, Днепропетровск,
ул. Дзержинского 2\4.
Тел/Факс: (056) 745 5596, (0562) 471791.
e-mail: academy@amsu.dp.u