УДК 519.681

В.Г. АКУЛОВСКИЙ

ФОРМАЛИЗАЦИЯ ВЗАИМОСВЯЗЕЙ ОПЕРАТОРОВ И ДАННЫХ В РАМКАХ РАСШИРЕННОЙ АЛГЕБРЫ АЛГОРИТМОВ

Ключевые слова: *система алгоритмических алгебр, регулярные схемы, формализация данных, разработка и преобразование алгоритмов.*

ВВЕДЕНИЕ

Достаточно известным и перспективным средством разработки алгоритмов является система алгоритмических алгебр (САА) [1–4]. На базе этого формального аппарата построен метод структурного проектирования программ (МСПП) [5, 6], в рамках которого структура управления алгоритмом отождествляется с регулярной схемой, состоящей из операторов и логических условий различной степени детализации. Данные специфицируются средствами алгебры структур данных [5]. Основным подходом метода является декомпозиция операторов, образующих алгоритм, с последовательным понижением уровня абстракции операторов и данных.

Прежде чем сформулировать цель данной работы, приведем следующую цитату [7]: «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». Мысль, высказанную в цитате, можно рассматривать как указание на общеметодологическую проблему разработки алгоритмов и программ.

В рамках этой проблемы будем решать задачу формализации взаимосвязей операторов и данных. В качестве средства решения воспользуемся расширенной системой алгоритмических алгебр (САА–Р) [8], что является очередным шагом в развитии формального аппарата.

1. ДАННЫЕ В РАСШИРЕННОЙ АЛГЕБРЕ АЛГОРИТМОВ

Для включения в рассмотрение данных воспользуемся абстрактной моделью ЭВМ [1, 2] в трактовке, предложенной в [3]. В качестве управляющего автомата (УА) рассматривается произвольный алгоритм, в качестве операционного автомата (ОА) — обрабатываемая этим алгоритмом информация (множество данных Δ). Выходные сигналы УА отождествляются с операторами, которые изменяют данные, т.е. состояние ОА. Выходные сигналы ОА представляют собой значения различных элементарных логических условий, характеризующих значения данных и/или соотношения между ними.

Разобьем 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, не фиксируемые в памяти. Эти данные определены (доступны для анализа) только сразу после выполнения некоторых операторов и не определены в остальное время.

Для описания алгоритмов рассмотрим двухосновную алгебраическую систему, основами которой являются множество операций и множество логических условий. При этом будем интерпретировать известные основные понятия (оператор, операция, логическое условие), введенные в САА [1–3], модифицированные и расширенные в САА–Р [8], в соответствии с рассматриваемой трактовкой абстрактной модели ЭВМ.

Пусть $\langle U, W, \Omega \rangle$ — 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 следующим образом:

$$A(D') = D'' \tag{1}$$

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

Множество данных D' — область определения, а множество данных D'' — область значений оператора $A(D')$. Эти множества независимы, т.е. для них может выполняться любое из следующих соотношений:

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

Далее введем понятие инверсного оператора:

$$\overline{A}(D'') = D'. \tag{2}$$

Здесь оператор $\overline{A}(D'')$ выполняет действия, обратные по отношению к оператору $A(D')$, реализуя таким образом операцию «откат». Естественное ограничение на использование этого оператора следующее. Для любого оператора $B(\hat{D}\ ) = \hat{D}'$, где $B(\hat{D}) \in U'$, $\hat{D}, \hat{D}' \subset DS$, на интервале алгоритма от $A(D')$ до $\overline{A}(D'')$ выполняется условие $\hat{D}\ ' \cap D'' = \varnothing$.

Определим операторы из множества U'' следующим образом:

$$A'(D') = D'' \tag{3}$$

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

Только после выполнения этих операторов множество DD определено, т.е. данные доступны для анализа. Множество D' — область определения, множество D'' — область значений оператора $A'(D')$, и для этих множеств, очевидно, выполняется соотношение $D' \cap D'' = \varnothing$.

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

Для операторов из множества U'' введем частично инверсный оператор в виде

$$\overline{A}'(D'') = D' \tag{4}$$

$$\forall\, D'' \subset DS, \quad \forall\, D' \subset DD, \quad \forall\, \overline{A}'(D'') \in U'',$$

где оператор $\overline{A}'(D'') = D'$ частично инверсный по отношению к оператору $A(D') = D''$, так как данные множества $D' \subset DD$ после выполнения оператора $\overline{A}'(D'')$ существуют только в динамической памяти, а в статической памяти не фиксируются. На выполнение этого оператора накладываются ограничения, такие же, как и на инверсный оператор.

Множество операторов U включает тождественный оператор, не изменяющий множества данных Δ, который определим как

$$E(D) = D \ \ \forall\, D \subset \Delta. \tag{5}$$

Определение операции на множестве W. Множество логических условий разобьем на два подмножества: $W = P \cup \Phi$. Множество предикатов P, определенных на множестве Δ, введем следующим образом:

$$p(D) = \alpha(D) = D \tag{6}$$

$$\forall\ p \in P, \quad \forall\ D \subset \Delta,\ D \neq \varnothing.$$

Результатом вычисления предиката является логическое условие $\alpha(D)$, характеризующее текущее состояние множества данных D, состояние которого (состояние ОА) при вычислении предиката p не изменяется.

Множество логических условий $\Phi \subset DS$ таково, что выполнение любого оператора $X(D') \in U$ не изменяет $\phi(D')$ для любого $\phi(D') \in \Phi$. Другими словами, условия множества Φ характеризуют состояние данных множества $D' \subset \Delta$, сохраняются в оперативной памяти, не зависят от действия операторов множества U и называются фиксирующими, так как позволяют сохранить (зафиксировать) некоторые характеристики текущего состояния ОА, т.е. значения и соотношения между данными множества D'.

Элементы множеств P и Φ 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 таким образом, что $0 < \mu < 1$ и известны обобщенные логические операции (дизъюнкция, конъюнкция, отрицание, циклическое отрицание) со своими таблицами истинности [8–10].

Кроме этих операций определим операции, позволяющие осуществлять прогнозирование вычислительного процесса. Впервые подобная операция (операция левого умножения условия на оператор) была введена В.М. Глушковым [1, 2], и именно она является отличительной особенностью построенной им алгебры [4].

В предлагаемой работе реализуем возможность прогнозирования следующим образом. Операцию левого умножения предиката на оператор (предложенную в [8]) в общем случае с учетом определений (1)–(4), (6) и ограничений $D, D' \neq \varnothing,\ D'' \cap D \neq \varnothing$ запишем в виде

$$X(D')\, p(D) = p(D \cup X(D')) = p(D \cup D'') = \alpha(\widetilde{D}). \tag{7}$$

Здесь $p(D) \in P,\ X(D') \in U,\ D, D', D'' \subset \Delta,\ \widetilde{D} = D \cup D'',\ D'' = X(D')$. Результатом выполнения этой операции является логическое условие $\alpha(\widetilde{D})$, характеризующее состояние множества данных $\widetilde{D}$.

Рассмотрим два варианта ее реализации.

1. В случае, когда оператор принадлежит множеству U'', операцию левого умножения с учетом (3), (6), (7) определим следующим образом:

$$A'(D')\, p(D) = p(D \cup (A'(D'))) = p(D \cup D'') = \alpha(\widetilde{D}) \tag{8}$$

$$\forall\ D, D' \subset DS, \quad \forall\ D'' \subset DD,\ \widetilde{D} = D \cup D'', \quad D'' = A'(D').$$

Результатом выполнения операции является условие $\alpha(\widetilde{D})$, характеризующее состояние множества данных $\widetilde{D}$, которое было бы получено после выполнения оператора $A(D') \in U'$, однако множество DS в данном случае остается неизменным, так как $D'' \subset DD$. Смысл операции состоит в прогнозировании вычислительного процесса.

В частном случае, когда в определении (8) множество $D'' = D$, эту операцию запишем в виде

$$p(A'(D')) = p(D'') = \alpha(D'') \tag{9}$$

$$\forall\ D' \subset DS, \quad \forall\ D'' \subset DD,\ D'' = A'(D').$$

Результатом выполнения такого оператора является логическое условие $\alpha(D'')$ — прогноз выполнения оператора $A(D') \in U'$, так как результат выполнения оператора $A'(D')$ не изменяет состояния множества DS. В данном случае получили операцию, адекватную операции, введенной В.М. Глушковым.

При использовании частично инверсного оператора эту операцию с учетом определения (4) запишем в виде

$$p(\overline{A}'(D'')) = p(D') = \alpha(D')$$

$$\forall D'' \subset DS, \quad \forall D' \subset DD, \; D' = \overline{A}'(D'').$$

В результате получаем возможность не только прогнозировать развитие вычислительного процесса, но и учитывать его предысторию. Очевидно, что адекватность выполнения такой операции зависит от выполнения ограничений на использование частично инверсных операторов.

2. В случае, когда оператор принадлежит множеству U', операцию левого умножения запишем с учетом определений (1), (6), (7):

$$A(D')\, p(D) = P(D \cup (A(D'))) = p(D \cup D'') = \alpha(\widetilde{D}) \tag{10}$$

$$\forall D, D', D'' \subset DS, \; \widetilde{D} = D \cup D'', \quad D'' = A(D').$$

Результатом выполнения этой операции является логическое условие $\alpha(\widetilde{D}\;)$, значение которого характеризует состояние данных множества $\widetilde{D}$, с учетом изменений, вызванных выполнением оператора $A(D')$.

В частном случае, когда в (10) $D'' = D$, данную операцию запишем в виде

$$p(A(D')) = p(D'') = \alpha(D'') \tag{11}$$

$$\forall D', D'' \subset DS, \; D'' = A(D').$$

Результатом выполнения такого оператора является логическое условие $\alpha(D'')$, характеризующее изменения данных, вызванные выполнением оператора $A(D')$.

В случае использования инверсного оператора эту операцию запишем в соответствии с определением (2) в виде

$$p(\overline{A}(D'')) = p(D') = \alpha(D') \tag{12}$$

$$\forall D'', D' \in DS, \; D' = \overline{A}(D'').$$

Результат ее выполнения — логическое условие $\alpha(D')$, характеризующее состояние данных после отката.

Заметим, что операторы второй группы не являются средством прогнозирования вычислительного процесса, и в этом смысле без них можно обойтись. Однако эти конструкции могут найти практическое применение, что будет показано в разд. 2 при рассмотрении операций α-дизъюнкции и α-итерации.

Кроме приведенных операций в [8] введены следующие операции левого умножения, которые будем также рассматривать в контексте данной работы.

Левое умножение фиксирующего условия на предикат. Операция используется для фиксации текущих характеристик данных некоторого множества D' и записывается следующим образом:

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

$$\forall D' \subset \Delta, \quad \forall \phi(D') \in \Phi, \quad p(D') \in P, \quad \Phi \subset DS.$$

Логическое условие $\alpha(D')$, характеризующее текущее состояние элементов множества D', запоминается в виде фиксирующего логического условия $\phi(D')$ как

элемент множества DS, позволяя сохранять историю вычислительного процесса. Учитывая жесткие ограничения, накладываемые на использование инверсных операторов, можно заключить, что возможности, предоставляемые данной операцией, достаточно важны, а в некоторых случаях просто необходимы.

Левое умножение фиксирующего условия на оператор. Для управления состоянием фиксирующих условий присоединим к множеству U элементарный оператор b^x, который изменяет состояние условия $\phi \in \Phi$ в результате применения операции левого умножения этого оператора на фиксирующее условие:

$$b^x \phi = \phi \tag{13}$$

$$\forall\, \phi \in \Phi,\ \phi = x,\quad x \in E_3,\quad \Phi \subset DS.$$

Таким образом, может быть установлено произвольное состояние фиксирующего логического условия в границах, определяемых значностью логики. Возможность управления состоянием фиксирующих логических условий является дополнительным средством управления ходом всего вычислительного процесса.

Отметим, что хотя последняя операция не имеет непосредственного отношения к теме данной работы, она приведена в связи с использованием в дальнейшем изложении (разд. 5).

Операции, определенные на множестве операторов из U, введем следующим образом.

Композиция $A(\hat{D}) * B(\tilde{D})$ (последовательное выполнение). Композиция операторов $A(\hat{D})$ и $B(\tilde{D})$ $\forall\, A(\hat{D}), B(\tilde{D}) \in U$, $\forall\, \hat{D}, \tilde{D} \in DS$ означает, что оператор $A(\hat{D})$ предшествует оператору $B(\tilde{D})$, а оператор $B(\tilde{D})$ следует за оператором $A(\hat{D})$, т.е. последовательно выполняются сначала оператор $A(\hat{D})$, затем оператор $B(\tilde{D})$. При этом состояние множества DS, полученное после выполнения оператора $A(\hat{D}\ )$, поступает (передается) оператору $B(\tilde{D})$. Множества $\hat{D}, \tilde{D}$ являются областями определения операторов $A(\hat{D})$ и $B(\tilde{D})$ соответственно, а множества $\hat{\tilde{D}} = A(\hat{D})$ и $\check{D} = B(\tilde{D})$ — областями их значений. При этом области определения операторов взаимно независимы, т.е. для них допустимы любые из соотношений:

$$\hat{D} \subseteq \tilde{D},\ \tilde{D} \subseteq \hat{D},\ \hat{D} \cap \tilde{D} = \varnothing,\ \hat{D} \cap \tilde{D} \neq \varnothing,\ \hat{D} = \tilde{D}.$$

В рассматриваемой трактовке эта операция может быть принципиально уточнена и детализирована. Введем понятие независимых и связанных операторов и определим их следующим образом.

Независимыми назовем операторы $A(\hat{D})$ и $B(\tilde{D})$, если для них выполняется соотношение $\hat{\tilde{D}} \cap \tilde{D} = \varnothing$. Множество данных $\hat{\tilde{D}}$, образующих область значений оператора $A(\hat{D})$ $(\hat{\tilde{D}} = A(\hat{D}))$, не пересекается с множеством данных $\tilde{D}$, образующих область определения оператора $B(\tilde{D})$, и результаты выполнения оператора $A(\hat{D})$ не влияют на выполнение оператора $B(\tilde{D})$.

Для таких операторов операция композиции реализуется следующим образом: $A(\hat{D}) * B(\tilde{D}) = D,\ D = \hat{\tilde{D}} \cup \check{D}$.

Если введем два дополнительных ограничения: $\hat{\tilde{D}} \cap \check{D} = \varnothing$, т.е. потребуем, чтобы область значений $\check{D}$ оператора $B(\tilde{D})$ не пересекалась с областью значений $\hat{\tilde{D}}$ оператора $A(\hat{D})$; $\hat{D} \cap \check{D} = \varnothing$, т.е. потребуем, чтобы область значений $\check{D}$ оператора $B(\tilde{D})$ не пересекалась с областью определения $\hat{D}$ оператора $A(\hat{D})$, то, таким образом, введем понятие — полностью независимые операторы.

Для полностью независимых операторов выполняется следующее тождественное соотношение:

$$A(\hat{D}) * B(\tilde{D}) = B(\tilde{D}) * A(\hat{D}). \tag{14}$$

Связанными назовем операторы $A(\hat{D})$ и $B(\tilde{D})$, если для них выполняется соотношение $\hat{D} \cap \tilde{D} \neq \varnothing$, т.е. область значений $\hat{D}$ оператора $A(\hat{D}\ )$ пересекается с областью определения $\tilde{D}$ оператора $B(\tilde{D})$.

Рассмотрим три частных случая этой операции, обозначая результирующую область значений в виде $\hat{\tilde{D}}$.

1. При $\hat{D} = \tilde{D}$, т.е. когда область значений $\hat{D}$ оператора $A(\hat{D})$ совпадает с областью определения $\tilde{D}$ оператора $B(\tilde{D})$, операцию композиции запишем в виде

$$A(\hat{D}) * B(\tilde{D}) = B(A(\hat{D})) = B(\hat{D}) = \hat{\tilde{D}} \,..$$

В этом частном случае реализуется вариант операции композиции, который рассматривается в САА.

2. При $\hat{D} \subset \tilde{D}$, т.е. когда область значений оператора $A(\hat{D})$ полностью включена в область определения оператора $B(\tilde{D})$, операцию композиции запишем в виде

$$A(\hat{D}) * B(\tilde{D}) = B(\hat{D} \cup \tilde{D}) = \hat{\tilde{D}}.$$

3. При $\tilde{D} \subset \hat{D}$, т.е. когда область определения оператора $B(\tilde{D})$ полностью включена в область значений оператора $A(\hat{D})$, операцию композиции запишем в виде

$$A(\hat{D}) * B(\tilde{D}) = \hat{D} \cup B(\hat{D}') = \hat{\tilde{D}},\ \hat{D}' = \tilde{D}.\ ,\ \hat{D}' = \tilde{D}.$$

В общем случае эту операцию запишем в виде

$$A(\hat{D}) * B(\tilde{D}) = \hat{D} \cup B(\tilde{D}') \cup B(\tilde{D}'') = \hat{D} \cup \breve{D}' \cup \breve{D}'' = \hat{\tilde{D}},$$

$$\tilde{D}' = \tilde{D} \cap \hat{D},\ \tilde{D}'' \cap \hat{D} = \varnothing.,$$

При рассмотрении операции композиции в данной интерпретации удалось получить ее детальное и достаточно строгое описание, позволившее учесть нюансы ее выполнения. При этом выделено подмножество данных (область значений), полученных в результате последовательного выполнения двух операторов, т.е. данных, связывающих операторы. Этот результат будет использован в процессе рассмотрения вопроса о преобразовании операции композиции в разд. 4.

Операция α_3-дизъюнкции. Данную операцию [8], ориентированную на использование трехзначных логических условий, запишем в виде

$$[\alpha(\tilde{D})](A(D') \vee B(D'') \vee C(D''')) = \begin{cases} A(D'), & \text{если} \quad \alpha(\tilde{D}\ ) = 1, \\ B(D''), & \text{если} \quad \alpha(\tilde{D}) = 0, \\ C(D'''), & \text{если} \quad \alpha(\tilde{D}) = \mu, \end{cases} \tag{15}$$

где $D', D'', D''' \subset DS,\ \tilde{D} \subset \Delta,\ \tilde{D} \neq \varnothing.$

Результатом реализации этой конструкции является выполнение одного из трех возможных операторов, который выбирается в соответствии со значением логического условия $\alpha(\tilde{D})$, принимающего истинные значения трехзначной логики E_3.

Операция α-итерации:

$$_{\alpha(D')}\{A(D'')\} \tag{16}$$

(соответствует программной конструкции WHILE). Операция α-итерации оператора $A(D'')$ состоит в проверке условия $\alpha(D')$, затем, если $\alpha(D')$ истинно, выполняется оператор $A(D'')$ и вновь проверяется условие $\alpha(D')$. Этот циклический процесс осуществляется до тех пор, пока условие $\alpha(D')$ не станет ложным.

Необходимым условием завершения операции является выполнение следующего соотношения: $D' \cap D''' \neq \varnothing$, где $D''' = A(D'')$.

В качестве логического условия $\alpha(D')$ в операциях α_3-дизъюнкции и α-итерации могут выступать: предикат, фиксирующее условие и операции левого умножения.

Предлагаемый подход позволил в рамках САА–Р специфицировать взаимосвязи между операторами и данными и отчасти собственно данные. При этом не только сохранились все возможности по построению производных алгоритмических конструкций и преобразованию регулярных схем, описывающих произвольный алгоритм, но и открылись существенные дополнительные возможности, что будет рассмотрено в следующих разделах.

2. ПРОИЗВОДНЫЕ АЛГОРИТМИЧЕСКИЕ КОНСТРУКЦИИ

Качество разрабатываемых алгоритмов, учитывая их многообразие, существенно зависит от изобразительных возможностей, предоставляемых средствами их разработки. В рамках рассматриваемого подхода сохраняются все возможности создавать производные алгоритмические конструкции. Продемонстрируем это на примере следующих принципиально важных операций:

α-дизъюнкция, которая является частным случаем α_3-дизъюнкции в соответствии с (5), (15),

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

$$= \begin{cases} A(D'), & \text{если} \quad \alpha(\widetilde{D}) = 1, \\ B(D''), & \text{если} \quad \alpha(\widetilde{D}) = 0, \end{cases}$$

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

α-фильтрация (последовательная фильтрация), которая является частным случаем α-дизъюнкции,

$$[\alpha(\widetilde{D})](A(D')) = [\alpha(\widetilde{D})](A(D') \vee E(D)), \tag{17}$$

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

Для этих операций, как и для операций α_3-дизъюнкции и α-итерации, в качестве логического условия могут выступать: предикат, фиксирующее условие и операции левого умножения.

Далее рассмотрим возможности использования операции левого умножения в случаях (10)–(12), о чем упоминалось в разд. 1.

Определение (10) может быть использовано при построении производной операции α_3-дизъюнкции (см. определение (15)), для которой приведем тождественное соотношение

$$A(\widetilde{D}) * [p(D)](B(D') \vee C(D'') \vee G(D''')) =$$

$$= [A(\widetilde{D})\, p(D)](B(D') \vee C(D'') \vee G(D''')).$$

Здесь условием выбора одной из ветвей вычислений будет $\alpha(\widetilde{D}' \cup D)$, где $\widetilde{D}' = A(\widetilde{D})$. Естественным требованием к такой конструкции является $\widetilde{D}' \cap D \neq 0$.

В частном случае в соответствии с определением (11) эту конструкцию можно записать в виде

$$A(\widetilde{D}) * [p(D)](B(D') \vee C(D'') \vee G(D''')) =$$

$$= [p(A(\widetilde{D}))](B(D') \vee C(D'') \vee G(D''')).$$

Аналогичные соотношения выполняются для операций α-дизъюнкции и α-фильтрации.

Используя соотношение (10), можно модернизировать операцию α-итерации, записав ее в соответствии с определением (16) в виде $A(\widetilde{D})\, p(D')^{\{B(D'')\}}$. В этом случае условие завершения цикла задается соотношениями: $D' \cap D''' \neq \varnothing$ и/или $D' \cap \widetilde{D}' \neq \varnothing$, где $D''' = B(D'')$, а $\widetilde{D}' = A(\widetilde{D}\ )$.

В частном случае в соответствии с (11) этот оператор запишем в виде $p(A(\widetilde{D}))^{\{B(D'')\}}$. Тогда для выхода из цикла необходимо выполнение соотношения $\widetilde{D}' \cap D''' \neq \varnothing$, где $\widetilde{D}' = A(\widetilde{D}\ )$, $D''' = B(D'')$, либо, если указанное условие не выполняется, выход из цикла будет определять только значение логического условия $\alpha(\widetilde{D}\ ') = p(A(\widetilde{D}))$.

В результате получили возможность строить операции α-итерации, в которых выход из цикла может быть организован как в зависимости, так и независимо от выполнения оператора $B(D'')$.

Приведенными примерами возможности построения модифицированных операций α-итерации далеко не исчерпываются. Можно предложить следующую, достаточно экзотическую конструкцию (см. (12)): $p(\overline{B}(D'''))^{\{B(D'')\}}$. Здесь $B(D'')$ и $\overline{B}(D''')$ — взаимно инверсные операторы. Такая конструкция позволяет проследить перспективу использования оператора $B(D'')$ без изменения значений обрабатываемых данных.

Таким образом, создавая различные модификации операции α-итерации, можно реализовывать наиболее удобные в каждом конкретном случае условия завершения операции.

Кроме того, построим широко применяемую алгоритмическую конструкцию, аналогичную оператору for в языках программирования. Известно, что она легко реализуется как частный случай α-итерации, т.е. циклической конструкции типа WHILE:

$$I(D)_{p(D')}\{A(D'')S(D)\}.$$

Здесь $I(D)$ — оператор инициализации переменных цикла; $A(D'')$ — тело цикла; $S(D)$ — оператор, изменяющий переменные цикла; $\alpha(D') = p(D')$ — условие завершения цикла, при этом $D' = I(D)$ в начале и $D' = S(D)$ в процессе выполнения цикла.

Воспользовавшись возможностями построенного алгебраического аппарата, реализуем предлагаемую конструкцию более, по мнению автора, изящно. Если это решение покажется излишне изощренным, можно рассматривать его как еще одну демонстрацию обширных возможностей по созданию новых алгоритмических конструкций.

Оператор инициализации $I(D)$ и оператор изменения переменных цикла $S(D)$ можно рассматривать как операции левого умножения оператора на предикат, однако особенность состоит в том, что оператор инициализации $I(D)$ выполняется только один (первый) раз, а в остальное время работы цикла выполняется оператор $S(D)$, изменяющий переменные цикла. Для разрешения этой проблемы введем вспомогательное фиксирующее условие ϕ и с учетом (11), (13), (16), (17) реализуем требуемую конструкцию следующим образом:

$$b^1\varphi^*{}_{p([\varphi](I(D) * b^0\varphi) \neq S(D))}\{A(D'')\}.$$

В данном случае фиксирующее условие ϕ, исходно установленное в состояние истинности, позволяет с помощью операции α-дизъюнкции при пер-

вом вхождении в цикл выполнить оператор $I(D)$ в качестве оператора в операции левого умножения оператора на предикат, после чего условие ϕ сбрасывается в 0. В остальное время функционирования цикла с помощью той же операции α-дизъюнкции при ложном фиксирующем условии ϕ выполняется операция левого умножения оператора $S(D)$ на предикат.

Построенную конструкцию запишем в виде

$$\langle I(D)\rangle\, p\,(S(D))^{\{A(D'')\}}. \tag{18}$$

Заметим, что оператор $I(D')$ заключен в угловые скобки, чтобы отметить тот факт, что он выполняется однократно и только в первый раз.

В данном разделе показано, что в рамках рассматриваемого подхода сохраняются и в известной мере расширяются все возможности формализованного построения производных алгоритмических конструкций, ориентированных как на класс решаемых задач, так и на особенности языка реализации алгоритма.

3. РЕАЛИЗАЦИЯ МЕХАНИЗМА ПОДПРОГРАММ В РАМКАХ ФОРМАЛЬНОГО АППАРАТА

Ни одна достаточно большая программа не обходится без такой мощной алгоритмической конструкции, как подпрограмма. Решение о создании подпрограммы в процессе проектирования программы принимается в двух случаях [7]:

— выявлены действия, которые в ходе работы программы должны неоднократно выполняться в разных ее местах;

— специфицирована некоторая частная задача, реализация которой откладывается или передается другому исполнителю.

Часто имеют место оба случая, но любой из них является достаточной причиной для создания подпрограммы.

Определим подпрограмму следующим образом. Оператор, снабженный средствами вызова, передачи параметров и механизмом возврата в точку вызова, является подпрограммой.

Полагая, что средства вызова подпрограммы и возврата из нее реализуются на аппаратном уровне, и в связи с этим абстрагируясь от них, остановимся на вопросах, связанных с передачей параметров в подпрограммы. С целью обеспечения универсальности и независимости от языка реализации алгоритма рассмотрим некоторый обобщенный механизм передачи параметров (интерфейс).

Для этого на основании элементарного анализа литературных источников (например, [11]) определим три возможных типа передаваемых параметров: входные, выходные и параметры, которые используются как в качестве входных, так и в качестве выходных. Последний тип назовем проходными параметрами. Будем различать два типа подпрограмм — процедуры и функции.

Вызов процедуры запишем в виде

$$\Pi(D';D'';D''') \tag{19}$$

$$\forall\, D', D'', D''' \subset DS,$$

где D' — множество входных, D'' — множество выходных, D''' — множество проходных фактических параметров. Областью определения процедуры является множество входных и проходных параметров $D' \cup D''$, а областью значений — множество выходных и проходных параметров $D'' \cup D'''$. При этом, как любое из множеств D', D'', D''', так и все они могут быть пустыми, например, если процедура выводит на экран или печать некоторый заранее заданный текст.

Вызов функции запишем в виде

$$\widetilde{D}F(D';D'';D''') \quad \forall\, D', D'', D''' \subset DS, \quad \forall\, \widetilde{D} \subset DD,$$

где D' — множество входных, D'' — множество выходных, D''' — множество проходных параметров, $\tilde{D}$ — множество данных, возвращаемых функцией в качестве результата, для которого выполняется условие $\tilde{D} \neq \varnothing$. Множества D', D'', D''' могут быть пустыми по причине, аналогичной приведенной для процедуры.

Отметим, что определение подпрограммы записывается в виде декомпозиции оператора вызова подпрограммы, полностью аналогичной случаю декомпозиции обычного оператора, за тем исключением, что при этом указываются формальные параметры. Такое определение будет выполнено в разд. 5 при рассмотрении конкретного примера.

Функция позволяет расширить возможности операции левого умножения предиката на оператор (8), которую запишем в виде

$$\tilde{D}F(D'; D''; D''')p(\hat{D}) = p(\tilde{D} \cup \hat{D}) = p(\hat{\tilde{D}}) = \alpha(\hat{\tilde{D}}).$$

Заметим, что возможность использования функции в операции левого умножения оператора на предикат обусловлена тем, что $\tilde{D} \in DD$, и, таким образом, данные множества DS не изменяются.

В частном случае, когда $\tilde{D} = \hat{D}$, это выражение в соответствии с определением (9) запишем в виде

$$p(\tilde{D}F(D'; D''; D''')) = p(\tilde{D}) = \alpha(\tilde{D}).$$

Формализация аппарата подпрограмм, помимо сокращения объема регулярной схемы, описывающей требуемый алгоритм, позволяет реализовать один из подходов к решению важнейших задач методологии программирования — модуляризации алгоритма.

4. ПРЕОБРАЗОВАНИЕ КОМПОЗИЦИИ ОПЕРАТОРОВ

Далее рассмотрим возможности преобразования композиции связанных операторов, сделав следующие предварительные замечания.

Во-первых, для любых двух операторов, очевидно, выполняется тождественное соотношение

$$A(\hat{D}) * B(\tilde{D}) = C(\hat{\tilde{D}}), \tag{20}$$

где $B(\tilde{D}), A(\hat{D}) \in U'$, область определения $\hat{\tilde{D}} = \hat{D} \cup \tilde{D}$, а область значений $\bar{\tilde{D}} = \breve{D} \cup \hat{D}$. Иными словами, операторы $A(\hat{D})$ и $B(\tilde{D})$ могут быть объединены в один оператор $C(\hat{\tilde{D}})$, который, выполняя действия, адекватные последовательному выполнению оператора $A(\hat{D})$, а затем $B(\tilde{D})$, позволит получить такой же результат.

Во-вторых, допустим, что некоторый оператор $A(\hat{D}) \in U'$ можно представить в виде двух независимых операторов

$$A(\hat{D}) = K(\hat{D}') * G(\hat{D}''), \tag{21}$$

область определения которых $\hat{D} = \hat{D}' \cup \hat{D}''$, а область значений $\bar{D} = \bar{D}' \cup \bar{D}''$, где $\bar{D} = A(\hat{D})$, $\bar{D}' = K(\hat{D}')$, $\bar{D}'' = G(\hat{D}'')$.

Отметим, что высказанное допущение не является особенно вольным, так как на практике возможность представить оператор подобным образом встречается достаточно часто. Затруднения для такого представления операторов могут возникнуть только на завершающих стадиях декомпозиции алгоритма.

Возможность преобразования композиции операторов будем рассматривать

в рамках данного допущения. Начнем рассмотрение преобразования композиции операторов $A(\hat{D}) * B(\tilde{D})$ с частных случаев, в том же порядке, который был принят при рассмотрении операции композиции в разд. 1.

1. В первом простейшем случае, когда $\hat{D} = \tilde{D}$, связанные операторы $A(\hat{D})$ и $B(\tilde{D})$ могут быть естественно в соответствии с (20) представлены в виде одного оператора:

$$A(\hat{D}) * B(\tilde{D}) = C(\hat{\tilde{D}}) = \bar{\bar{D}}.$$

2. В случае, когда $\hat{D} \subset \tilde{D}$, для преобразования операции композиции представим в соответствии с (21) оператор $B(\tilde{D})$ в виде двух независимых операторов $B(\tilde{D}) = G(\hat{D}) * K(\tilde{D}')$, обрабатывающих соответственно данные, полученные после выполнения оператора $A(\hat{D})$, и независимую от $A(\hat{D})$ часть данных. Далее в соответствии с (20) объединяем два связанных оператора: $A(\hat{D}) * G(\hat{D}) = C(\hat{\tilde{D}})$. В результате получим

$$A(\hat{D}) * B(\tilde{D}) = C(\hat{\tilde{D}}) * K(\tilde{D}').$$

3. В случае, когда $\tilde{D} \subset \breve{D}$, представим (см. (21)) оператор $A(\hat{D})$ в виде $A(\hat{D}) = K(\hat{D}') * G(\hat{D}'')$, где оператор $K(\hat{D}')$ продуцирует «независимую» часть данных (не поступающих на вход $B(\tilde{D})$), а оператор $G(\hat{D}'') = \hat{D}' = \tilde{D}$ вырабатывает множество данных, передаваемых оператору $B(\tilde{D})$. Очевидным следующим шагом является объединение (см. (20)) связанных операторов $G(\hat{D}'') * B(\tilde{D}) = C(\hat{\tilde{D}})$, в результате чего получим

$$A(\hat{D}) * B(\tilde{D}) = K(\hat{D}') * C(\hat{\tilde{D}}).$$

В общем случае композицию связанных операторов рассмотрим аналогично частным случаям 2 и 3. Оператор $A(\hat{D})$ представим в виде

$$A(\hat{D}) = K(\hat{D}') * G(\hat{D}''),$$

где $G(\hat{D}'') = \hat{D}' = \tilde{D}'$ обрабатывает множество данных, поступающих на вход оператора $B(\tilde{D})$, а $K(\hat{D}')$ — независимую часть данных.

Оператор $B(\tilde{D})$ представим в виде

$$B(\tilde{D}) = H(\tilde{D}') * M(\tilde{D}''),$$

где $H(\tilde{D}')$ обрабатывает данные, переданные оператором $G(\hat{D}'')$, а $M(\tilde{D}'')$ — независимые данные.

Далее, объединив связанные операторы $G(\hat{D}'') * H(\tilde{D}') = C(\hat{\tilde{D}})$, получим

$$A(\hat{D}) * B(\tilde{D}) = K(\hat{D}') * C(\hat{\tilde{D}}) * M(\tilde{D}'').$$

В том случае, когда при разбиении операторов удастся обеспечить их полную независимость, порядок следования этих операторов не имеет значения в соответствии с тождественным соотношением (14).

Завершая рассмотрение возможностей преобразования связанных операторов, сделаем небольшое отступление методологического характера. Модуляризация программ предполагает, что модуль — это небольшой объем кода, предназначенный для решения локальной задачи [7, 11]. Возможность объединения связанных и выделение независимых операторов позволяет решать задачу модуляризации (по крайней мере, в некоторых случаях) на достаточно формаль-

ном уровне. Высказанное соображение в полной мере относится и к подпрограммам, в том смысле, что открывается возможность удалять из подпрограммы операторы, не связанные решением общей задачи, минимизируя, таким образом, объем подпрограммы и обеспечивая ее функциональную завершенность (замкнутость).

5. ПРЕОБРАЗОВАНИЕ РЕГУЛЯРНЫХ СХЕМ АЛГОРИТМОВ

В отношении преобразования алгоритмических конструкций в рамках рассматриваемой интерпретации формальный аппарат ничего не утратил. Многие преобразования могут быть реализованы более формально. Покажем это на примере разложения операции α-дизъюнкции на последовательные фильтры, что было выполнено в [8], где ограничения на использование аналогичного преобразования приводились на содержательном уровне. В данном случае запишем такое преобразование в виде

$$[a(\widetilde{D})](A(D') \vee B(D'')) = [\alpha(\widetilde{D})](A(D')) * [\neg\alpha(\widetilde{D})](B(D'')).$$

В связи с тем, что логическое условие $\alpha(\widetilde{D})$ в правой части выражения проверяется дважды, для эквивалентности преобразования необходимо, чтобы выполнялось соотношение $\widetilde{D} \cap D'' = \varnothing$, где $D'' = A(D')$. Если данное соотношение не выполняется, оператор можно переписать в виде

$$[\alpha(\widetilde{D})](A(D') \vee B(D'')) = [\neg\alpha(\widetilde{D})](B(D'')) * [\alpha(\widetilde{D})](A(D')),$$

потребовав, чтобы выполнялось соотношение $\widetilde{D} \cap D''' = \varnothing$, где $D''' = B(D'')$. В результате можно сделать вывод о том, что в некоторых случаях ограничение на данное преобразование может быть снято за счет выбора порядка следования операторов.

Поскольку наиболее наглядным способом продемонстрировать возможности преобразования регулярных схем в рамках предлагаемого подхода является решение конкретной задачи, рассмотрим следующую элементарную задачу.

Имеется два массива разной длины:

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

Необходимо вычислить сумму элементов массива X, в том случае, если он содержит меньше или ровно 100 элементов. В противном случае, при условии, что массив Y содержит меньше или ровно 200 элементов, выполнить то же вычисление. Если последнее условие не выполняется, вычислить сумму элементов массива Y.

Запишем алгоритм, который назовем СУММ, в виде регулярной схемы

$$\text{SUMM}(X, Y) = [\alpha(k)](\text{sum}(X) \vee [\beta(n)](\text{sum}(X) \vee \text{sum}(Y))),$$

где α — условие $k \le 100$; β — условие $n \le 200$; $\text{sum}(X)$, $\text{sum}(Y)$ — операторы, суммирующие элементы массивов X и Y соответственно.

Учитывая, что операция суммирования повторяется, оформим ее в виде процедуры в соответствии с определением (19) и запишем очередной вариант алгоритма в виде

$$\text{SUMM}(X, Y) = [\alpha(k)](\Pi\,\text{sum}(X, k; A; \varnothing)$$

$$\vee ([\beta(n)](\Pi\,\text{sum}(X, k; A; \varnothing) \vee \Pi\,\text{sum}(Y, n; A; \varnothing))),$$

где X, k, Y, n — фактические входные параметры, A — фактический выходной параметр, множество фактических проходных параметров пусто.

Далее, воспользовавшись построенной операцией итерации типа for (18), определим подпрограмму (о чем упоминалось в разд. 3) следующим образом:

$$\Pi\,\mathrm{sum}\,(Z, m; L; \varnothing) = {}_{\langle I(j)\rangle\, p(S(j), m)}\,\{s(z_j)\}.$$

Здесь $s(z_j)$ — оператор, осуществляющий суммирование элементов массива, Z, m — входные формальные параметры, L — выходной формальный параметр, а множество проходных формальных параметров пусто.

Наконец, воспользовавшись тождественным соотношением [8] вида

$$[\alpha(D)](A(\hat{D})) \vee [\beta(D')](A(\hat{D}) \vee B(\tilde{D})) = [\alpha(D) \vee \beta(D')](A(\hat{D}) \vee B(\tilde{D})),$$

можно, осуществив наиболее эффективное преобразование, свести регулярную схему к виду

$$\mathrm{SUMM}\,(X, Y) = [\alpha(k) \vee \beta(n)](\Pi\,\mathrm{sum}\,(X, k; A; \varnothing) \vee \Pi\,\mathrm{sum}\,(Y, n; A; \varnothing)).$$

ЗАКЛЮЧЕНИЕ

Основным результатом данной работы является очередной шаг в направлении развития формального аппарата, ориентированного на разработку алгоритмов и программ. В частности, удалось на достаточно формальном уровне специфицировать взаимосвязь операторов и данных, что, при сохранении существующих, открыло новые возможности по построению производных алгоритмических конструкций и преобразованию регулярных схем. При этом новые алгоритмические конструкции могут строиться ориентированными как на класс решаемых задач, так и на возможности языка, на котором разрабатываемый алгоритм будет реализовываться. Достаточно важным результатом являются новые средства решения такой актуальной методологической задачи, как модуляризация алгоритма.

Дальнейшее развитие полученных результатов предполагается проводить в направлении расширения возможностей формального аппарата по построению и преобразованию регулярных схем алгоритмов, а также решения задач методологического характера.

СПИСОК ЛИТЕРАТУРЫ

1. Глушков В.М. Теория автоматов и формальные преобразования микропрограмм // Кибернетика. — 1965. — № 5. — С. 1–10.
2. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. — К.: Наук. думка, 1978. — 319 с.
3. 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.
4. Цейтлин Г.Е. Введение в алгоритмику. — К.: Сфера, 1998. — 310 с.
5. Цейтлин Г.Е. Алгоритмическая алгебра структур данных и многоуровневое проектирование программ // Программирование. — 1986. — № 3. — С. 8–16.
6. Цейтлин Г.Е., Бакулин А.В. Многоуровневые структурированные проекты программ и их обоснование // Кибернетика и системный анализ. — 1991. — № 5. — С. 98–107.
7. Турский В. Методология программирования. — М.: Мир, 1981. — 264 с.
8. Акуловский В.Г. Расширенная алгебра алгоритмов // Проблеми програмування. — 2007. — № 3. — С. 3–15.
9. Акуловський В.Г., Костенко В.В. Тризначна логiка в алгебрi алгоритмiв // Вiсн. Акад. митн. служби УкраВни. — 2007. — № 1. — С. 83–88.
10. Поспелов Д.А. Логические методы анализа и синтеза схем. — Изд. 3-е, перераб. и доп. — М.: Энергия, 1974. — 368 с.
11. Мейер Б., Бодуэн К. Методы программирования. — М.: Мир, 1982. — Т. 1. — 356 с.

Поступила 27.07.2007