УДК 519.681

В.Г. Акуловский

РАСШИРЕННАЯ АЛГЕБРА АЛГОРИТМОВ

Предлагается формальный аппарат (расширенная система алгоритмических алгебр), ориентированный на детальную разработку алгоритмов с учетом особенностей языка, на котором будет реализован алгоритм, с использованием трехзначной логики и декларативных знаний совместно с процедурными. Показаны возможности формального преобразования алгоритмов и построения производных алгоритмических конструкций. Приведены примеры использования формального аппарата.

Введение

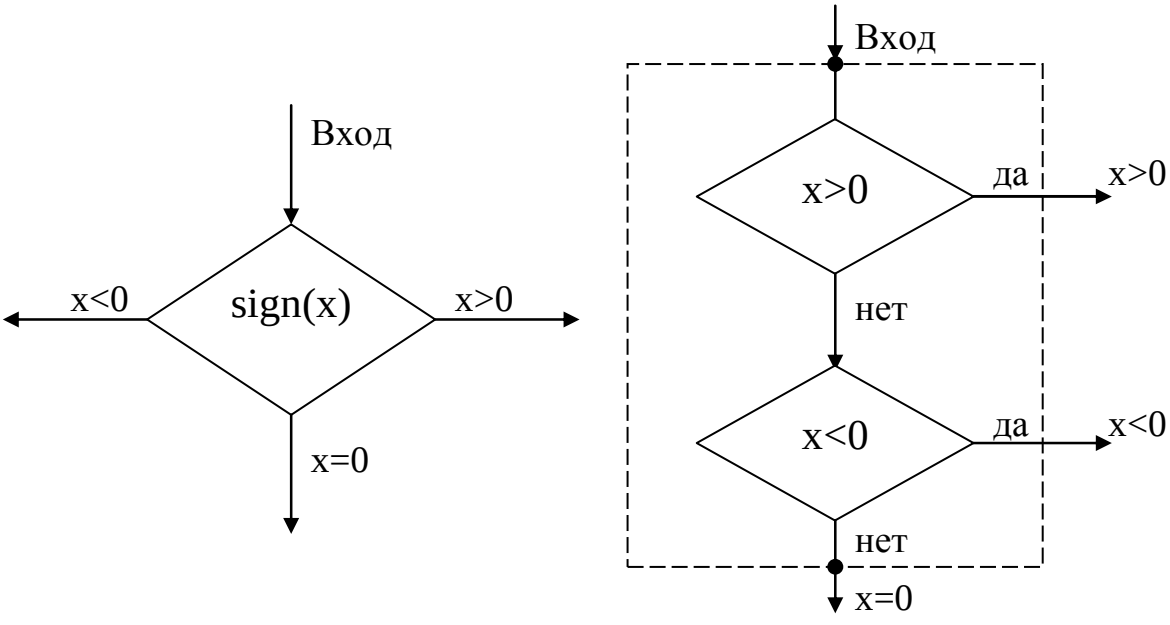
Система алгоритмических алгебр, разработанная В.М. Глушковым и его учениками [1–3], называемая обычно алгебра алгоритмов Глушкова (в дальнейшем алгебра алгоритмов), хорошо зарекомендовала себя при решении весьма широкого класса задач, подтверждая, таким образом, перспективность использования алгебраического аппарата в рамках прикладной теории алгоритмов (алгоритмики, в терминах Г.Е. Цейтлина [4]). Данная теория, как и любая другая, развивается, а предлагаемая работа – очередной шаг в направлении её развития.

Предпосылками к выполнению данной работы явилось следующее.

Первым побудительным мотивом послужили критические оценки архитектуры современных ЭВМ, высказанные в ряде публикаций, в частности Кнута [5] и разработчика трехзначной ЭВМ «СЕТУНЬ» Н.П. Брусенцова [6].

Последний утверждает [7]: “… ущербность современных компьютеров обусловлена тем, что логические выводы, которыми они оперируют, исчерпываются дискретной двоицей – "да", "нет", а иные модальности аксиоматически отсечены "законом исключенного третьего", что не согласуется с тем, как устроен и функционирует природный мир. В тоже время трехзначная логика не только вполне корректна и адекватна действительности, но является даже более удобной и привычной для людей формой мышления”. Это утверждение иллюстрируется примером ветвления по знаку величины X (рисунок).

В рассматриваемом примере троичное ветвление по знаку величины X опи-



а – Троичная схема б – Двоичная схема

Рисунок. Ветвление по знаку

сано заданием единственной трехзначной операции sign(X) и выполняется за один шаг, в то время как такое же ветвление, осуществляемое средствами двузначной логики, связано с необходимостью двух операций и выполняется, вообще говоря, за два шага.

Поскольку преимущество трехзначной логики в приведенном примере, очевидно, то можно с достаточной уверенностью рассчитывать на то, что многие алгоритмические построения в трехзначной логике могут быть реализованы более компактно и эффективно, обеспечивая качество создаваемых алгоритмов и программ.

Учитывая, что нечто выразимое в трехзначной логике выразимо и в двузначной, можно использовать преимущества трехзначной логики в разработках для современных (двухзначных) ЭВМ. Появление же трехзначных компьютеров, когда и если это произойдет, позволит повысить качество реализации алгоритмов, построенных с использованием трехзначной логики.

Во-вторых. Известно, что качество разрабатываемого алгоритма в существенной мере зависит от соответствия средств его описания специфике решаемой задачи.

В-третьих. Известно также, что чем детальнее, с учетом всех нюансов, разработан алгоритм, тем меньше проблем возникает на последующих этапах разработки программного изделия. При этом чем детальнее разрабатывается алгоритм, тем больше открывается возможностей для формальных его преобразований. Однако возможности его детализации определяются в существенной мере соответствием, используемых средств разработки алгоритма, изобразительным возможностям целевого языка, т.е. языка программирования, на котором этот алгоритм будет записан.

В-четвертых. Исторически в программировании сложилась ситуация, при которой в большинстве случаев используются процедурные знания, т. е. знания, растворенные в алгоритмах. А декларативные знания – знания, отделенные от алгоритма, используются в системах, основанных на знаниях (экспертные системы, логическое программирование и т.п.).

Представляется интересным и перспективным использование декларативных знаний совместно с процедурными в рамках единого формального аппарата, так как это позволит расширить область применения этого формального аппарата на новые классы задач и упростить (в некоторых случаях) решение задач традиционных.

Исходя из приведенных предпосылок, можно сказать, что основной задачей, решаемой в данной работе, является разработка расширенной алгебры алгоритмов, которая обеспечивает следующие дополнительные возможности:

использование преимуществ трехзначной логики при разработке алгоритмов (следует отметить, что трехзначная логика использовалась в алгебре алгоритмов, однако в данной работе мы попытаемся использовать её возможности в полной мере);

построение алгоритмических конструкций, ориентированных на конкретные приложения, с приемлемым качеством их реализации;

детализация алгоритмов до уровня, на котором программисту уже практически не приходится принимать алгоритмических решений, в частности, за счет учета особенностей целевого языка программирования;

использование декларативных знаний в задачах, где в этом возникает потребность или где это просто удобней.

При этом, естественно, необходимо сохранить положительные свойства алгебры алгоритмов, в частности, возможности преобразования алгоритмов с целью их оптимизации.

1. Система алгоритмических алгебр

Для разработки алгебраической системы В.М. Глушковым была предложена известная абстрактная модель функционирования ЭВМ, которая представляет собой схему взаимодействия двух автоматов управляющего – U и операционного – O.

Выходные сигналы A управляющего автомата U отождествляются с опе-

раторами, которые изменяют состояние операционного устройства O. Выходные сигналы операционного автомата O, представляют собой значения различных элементарных логических условий α, определенных на операционном устройстве. Множество M состояний операционного автомата O называется информационным множеством.

Уточним данную модель следующим образом. Разобъем множество M на два подмножества M=MS ∪ MD, где

MS – статическая составляющая, которая постоянно определена и которая может интерпретироваться, как оперативная память;

MD – динамическая составляющая, которая отражает динамические изменения состояния множество M, не фиксируемые в памяти. Эта составляющая определена (доступна для анализа) только сразу после выполнения некоторых операторов и не определена во всё остальное время. MD может интерпретироваться, как регистровая память.

Таким образом, возникает необходимость включить в рассмотрение два вида операторов (изменяющих состояние MS и изменяющих состояние MD), поступающих на вход операционного устройства O.

Для описания алгоритмов рассмотрим двухосновную алгебраическую систему, основами которой являются множество операций и множество логических условий.

Пусть < U, W, Ω > - CAA, 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 статическую составляющую MS множество M состояний операционного автомата O, переводя его в новое состояние: $A(m) = m'$, $\forall m, m' \in MS$, $\forall A \in U'$;

а операторы из множества U'' изменяет его динамическую составляющую MD:

$A'(m) = m'$, $\forall m \in MS$, $\forall m' \in MD$, $\forall A' \in U''$ и только после выполнения этих операторов эта составляющая MD определена.

Можно сказать (в программистской терминологии), что оператор $A \in U'$ отличается от оператора $A' \in U''$ тем, что первый выполняет операции присваивания, а второй таких операций не содержит.

Множество операторов U включают пустой оператор E, не изменяющий состояния информационного множества, т. е. $E(m) = m$, $\forall m \in M$.

Для того чтобы ввести понятие неопределенного оператора N присоединим к информационному множеству M специальное неопределенное состояние ν. Будем трактовать неопределенный оператор как ошибочный (недопустимый), выполнение которого N(m) = ν переводит операционный автомат O в состояние, после перехода в которое его дальнейшее функционирование будем считать неопределенным.

Множество логических условий разобъем на два подмножества $W = P \cup F$, где P – множество всюду определенных на множестве M предикатов, результатом вычисления которых является логическая переменная α, характеризующая текущее состояние операционного автомата O. Состояние операционного автомата O при вычислении предиката p $\forall p \in P$ не изменяется, т. е. $p(m) = \alpha(m) = m$, $\forall m \in M$.

F – множество логических условий, такое, что для любого оператора $x \in U$, результат применения которого $m' = x(m)$ к текущему состоянию m $(m, m' \in M)$ выполняется $f(m) = f(m')$ для любого $f \in F$. Другими словами, условия множества F сохраняют свои значения независимо от действия операторов множества U и называются фиксирующими, так как позволяют сохранить (зафиксировать) текущее состояние операционного автомата O.

Отметим, что фиксирующие логические условия не являются чем-то принципиально новым в программировании, так как они, по видимому, всегда исполь-

зовались на практике под именами “флажки”, ”признаки” и т.п. В данном случае это средство только формализовано в соответствующем контексте.

(В дальнейшем для краткости и там где это не затруднит понимания, вместо термина фиксирующее логическое условие будем использовать термин – “фиксирующее условие”, а в некоторых случаях просто – “условие”).

Элементы множества P и F принимают истинные значения трехзначной логики $E_3 = \{0, 1, \mu\}$, где 0 – ложь, 1 – истина, μ – промежуточное значение. (Отметим, что вместо традиционного термина – неопределенность будем использовать термин промежуточное значение, так как это значение в работе не трактуется как неопределенное).

На множестве E_3 введено отношение порядка $<$ так, что $0 < \mu < 1$.

К логическим операциям системы Ω_1 относятся следующие обобщенные операции со своими таблицами истинности [8]:

$\neg\alpha$ – инверсия, обобщение операции отрицания (табл. 1);

$\vec{\alpha}$ – циклическое отрицание (табл. 2);

дизъюнкция и конъюнкция $\alpha \vee \beta = \max(\alpha, \beta)$, $\alpha \wedge \beta = \min(\alpha, \beta)$ (табл. 3).

Обобщенные булевы операции удовлетворяют всем законам булевой алгебры, кроме закона исключенного третьего и закона противоречия.

Очевидно, однако, выполняется закон исключенного четвертого $\alpha \vee \vec{\alpha} \vee \vec{\vec{\alpha}} = 1$ и закон аналогичный закону противоречия $\alpha \wedge \vec{\alpha} \wedge \vec{\vec{\alpha}} = 0$.

Кроме этих операций определим три операции левого умножения

Левое умножение ′ предиката $\in P$ на оператор $B' \in U''$ представляет собой условие α, характеризующее состояние M, которое было бы получено после выполнения оператора $B \in U'$, однако статическое состояние MS операционного автомата O при этом остается неизменным, так как:

$$B'p(m) = p(B'(m)) = p(m \cup m') = \alpha(m \cup m') = m,$$
$$\forall m \in MS, \forall m' \in MD.$$

Смысл оператора состоит в прогнозировании вычислительного процесса.

Таблица 1

α	$\neg\alpha$
0	1
μ	μ
1	0

Таблица 2

α	$\vec{\alpha}$
0	μ
μ	1
1	0

Таблица 3

α	β	$\alpha \vee \beta$	$\alpha \wedge \beta$
0	0	0	0
0	μ	μ	0
0	1	1	0
μ	0	μ	0
μ	μ	μ	μ
μ	1	1	μ
1	0	1	0
1	μ	1	μ
1	1	1	1

Левое умножение фиксирующего условия на предикат p*f*

где $\in P$, а $f \in F$.

Для фиксации текущих значений состояния M операционного автомата введем операцию левого умножения фиксирующего условия на предикат следующим образом:

$$f(\mathrm{m}) = \alpha(m) = m',$$

$\forall m \in M, \forall m' \in MS$, где $m' = m \cup \{f\}$, а $f = \alpha(m)$.

То есть, текущее значение состояния операционного автомата O фиксируется как элемент множества MS, позволяя сохранять историю вычислительного процесса.

Левое умножение фиксирующего условия на оператор $b^x f$

Для управления состоянием фиксирующих условий присоединим к множеству U элементарный оператор b^x, который изменяет состояние условия $f \in F$ в результате применения операции левого умножения этого оператора на фиксирующее условие $b^x f(m) = m'$, $\forall m, m' \in MS$, где $m' = m \cup \{f\}$, $f = x$, $x \in E_3$.

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.

Для операторов из U введем следующие операции

Композиция A∗B (последовательное выполнение). Композиция оператора A и B $\forall A, B \in U$ означает последовательное выполнение сначала оператора A и затем оператора B. Иными словами: $(\mathrm{A} * \mathrm{B})(\mathrm{m}) = \mathrm{B}(\mathrm{A}(\mathrm{m})) = \mathrm{B}(\mathrm{m}') = \mathrm{m}''$, $\forall m, m', m'' \in \mathrm{MS}$.

Эта операция обладает свойством ассоциативности (A*B)*C=A*(B*C) и для нее выполняются тождества

A*E=E*A=A, где E тождественный оператор,

$$\mathrm{A}*\mathrm{N}=\mathrm{N}*\mathrm{A}=\mathrm{N}, \qquad (1)$$

где N неопределенный оператор.

(Символ ∗ между операторами в очевидных случаях будем опускать, т. е. $A * B = AB$).

Операция α_3 – дизъюнкция [α] (A $\vee$ B $\vee$ C). Эта операция ориентирована на использование трехзначных логических условий:

$$D(m) = \begin{cases} A(m), \text{если } \alpha(\mathrm{m}) = 1; \\ B(m), \text{если } \alpha(\mathrm{m}) = 0; \\ C(m), \text{если } \alpha(\mathrm{m}) = \mu, \end{cases}$$

т. е., выполняется один из трех возможных операторов, который выбирается в соответствии со значением логического условия α, принимающего истинные значения трехзначной логики E_3.

α – итерация ${}_{\alpha}\{A\}$ (соответствует программной конструкции WHILE). Операция α-итерации оператора A по условию α, примененная к некоторому состоянию информационного множества $m \in M$, состоит сначала в проверке условия α. Затем, если α истинно, применяется оператор A и вновь проверяется условие α. Этот циклический процесс, состоящий в проверке условия α и выполнении оператора A при истинном α, осуществляется до тех пор, пока условие α не станет ложным, чем в этом случае завершается выполнение α-итерации. Для этой операции выполняются такие тождественные соотношения:

$${}_{\alpha}\{A\} = {}_{\alpha}\{A\}\neg\alpha \qquad (2)$$

$${}_{\alpha}\{E\} = \mathrm{N}, \qquad (3)$$

где E – тождественный, а N – неопределенный оператор.

$${}_{0}\{A\} = \mathrm{E}, \qquad (4)$$

где 0 – тождественно ложное условие, а E – тождественный оператор.

${}_{1}\{A\} = \mathrm{N}$, где 1 – тождественно истинное условие, а N – неопределенный оператор.

В последнем случае следует обратить внимание на тот факт, что существует достаточно широкий класс систем управления (например, системы управления нижнего уровня иерархических АСУ ТП, реализуемые на программируемых контроллерах), для которых оператор вида $_1\{A\}$, называемый бесконечным циклом, не только применим, но и необходим. Поэтому, введем дополнительную операцию цикла $_\alpha\langle A\rangle$, которая полностью эквивалентна операции α – итерация, но для которой выполняется соотношение $_1\langle A\ \rangle \neq N$.

В качестве логического условия α в операторе α_3 – дизъюнкция и α – итерация могут выступать p-предикат, *f* – фиксирующее условие и операции левого умножения ′ и p*f*. Отметим, что операция левого умножения p*f* может использоваться не только в указанных операциях, но и как самостоятельная конструкция, с помощью которой формируется значение фиксирующего условия.

Результатом данного раздела (и основным результатом всей работы) является расширенная система алгоритмических алгебр (САА-Р), возможности которой решать выше поставленные задачи демонстрируются в следующих разделах.

2. Производные операции САА-Р

Очевидно, что качество разрабатываемых алгоритмов, учитывая их многообразие, несомненно и существенно зависит от изобразительных возможностей, предоставляемых средствами их разработки. Предлагаемая САА-Р позволяет строить производные операции, учитывающие специфику разрабатываемых алгоритмов и целевого языка программирования. Продемонстрируем это на примерах наиболее широко используемых конструкций:

α – дизъюнкция (соответствует программной конструкции ЕСЛИ...ТО…. .....ИНАЧЕ) вводится, как частный случай α_3 – дизъюнкции: $[\alpha](A \vee B) = [\alpha](A \vee B \vee E)$, таким образом

$$[\alpha](A \vee B)(m) = \begin{cases} A(m), \textit{ если } \alpha(m) = 1; \\ B(m), \textit{ если } \alpha(m) = 0; \end{cases}$$

$\forall m \in M$.

Эту операцию удобно использовать в тех случаях, где возможности трехзначной логики излишни, например, при сравнении двух величин на равенство.

α-фильтрация (последовательная фильтрация) $[\alpha](A)$, которая соответствует программной конструкции ЕСЛИ... …ТО... и является частным случаем α - дизъюнкции: $[\alpha](A) = [\alpha](A \vee E)$ и обеспечивает выполнения оператора А только при истинном значении α. Для этой операции справедливы такие тождественные соотношения:

$$[\alpha]\,([\alpha](A)) = [\alpha](A), \qquad (5)$$

$$[\alpha]\,([\neg\alpha](A)) = E\,. \qquad (6)$$

continue – $_\alpha\{A^*[\beta](continue)^* B\}$, операция позволяет перейти к следующей итерации цикла до завершения предыдущей, т. е., эта операция адекватна одноименному оператору языков программирования. Воспользовавшись операцией последовательной фильтрации, получаем $_\alpha\{A[\beta](continue)B\} = {}_\alpha\{A[\neg\beta](B)\}$, где фильтрующее условие β позволит пропустить оператор В и перейти к очередной итерации.

break – $_\alpha\{A^*[\beta](break)^* B\}$, операция соответствует одноименному оператору в языках программирования и позволяет прервать в любой момент выполнение цикла, потребность, в чем возникает достаточно часто.

Для реализации осуществляемого оператора воспользуемся вспомогательным фиксирующим условием *f*:

$$_\alpha\{A[\beta](break)B\} = b^1 f^*_{\alpha\wedge f}\{A^*[\beta](b^0 f \vee B)\}.$$

В построенной конструкции условие *f* устанавливается в исходное истинное состояние для входа в цикл. В цикле при

истинном условии β (условии завершения цикла) α – дизъюнкция приводит к установке условия f в состояние ложь, в результате чего оператор B не будет выполнен и цикл завершится (по условию $\alpha \wedge f$), а в противном случае (при ложном β) выполняется оператор B и циклирование продолжается.

Переключатели – удобные во многих случаях производные операции, которые позволяют сократить размер и сделать более понятным алгоритм, содержащий вложенные α – дизъюнкции, который запишем следующим образом:

$$[\alpha_1](A_1 \vee [\alpha_2](A_2 \vee [\alpha_3](A_3 \vee ... \\ ... \vee [\alpha_k](A_k \vee E)...) = \prod \begin{pmatrix} \alpha_1, \alpha_2, ... \alpha_k \\ A_1, A_2, ... A_k \end{pmatrix}.$$

Аналогично строится операция для более общего случая вложенных α_3-дизъюнкций:

$$[\alpha_1](A_1 \vee A_2 \vee [\alpha_2](A_2 \vee A_3 \vee ... \\ ... \vee [\alpha_k](A_k \vee A_k \vee E)...) = \prod \prod \begin{pmatrix} \alpha_1, \alpha_2, ..., \alpha_k \\ A_1, A_2, ..., A_k \\ B_1, B_2, ..., B_k \end{pmatrix}.$$

При построении производных операций необходимо учитывать возможности целевых языков реализовывать новые операции. В противном случае, вновь созданная удобная алгоритмическая конструкция при её реализации на некотором языке программирования может столь существенно понизить эффективность полученного программного продукта, что его использование окажется неприемлемым.

Проиллюстрируем справедливость этого утверждения следующим примером.

В работе [4] приведена производная операция цикла вида $\{B[\alpha]A\}$, где контроль условия циклирования осуществляется в середине цикла.

Реализация такого оператора не вызвала бы никаких затруднений при его реализации на Ассемблере (по крайней мере, для процессора 80x86), но она отсутствует в большинстве современных языков программирования. Реализация данной операции в виде $\{B[\alpha]A\} = A_\alpha \{BA\}$ имеет тот недостаток, что оператор A записывается дважды. Если же оператор A достаточно объемен, а операция используется достаточно часто, то это может привести к такому увеличению алгоритма, а в последствии и программы, что возникшие издержки сделают применение этой конструкции недопустимой.

В связи с этим рассмотрим возможность реализации данного оператора другим (одним из возможных) способом и запишем его в виде

$$\{B[\alpha]A\} = b^1 f *_f \{B * [\alpha](A \vee b^0 f)\}.$$

Данная реализация выполняется следующим образом. Мы ввели служебное фиксированное условие f и, используя элементарные операторы b^1 и b^0, устанавливаем соответственно истинное и ложное значения этого условия. Оператор b^1 устанавливает исходное истинное значение условия f (параметр цикла) для входа в цикл. В котором выполняется оператор B и при истинном условии α операция α-дизъюнкция выполняет оператор A. Цикл выполняется до тех пор, пока условие α не примет значение ложь. В этом случае оператор A не будет выполнен, а условие f будет установлено в ложное состояние оператором b^0, что и приведет к завершению цикла.

В данном случае некоторая громоздкость полученного выражения не должна смущать, так как дополнительно используемые операторы элементарны по определению и не могут оказать существенного влияния на эффективность создаваемого оператора. Реализовать его можно практически на любом языке программирования.

Производные операции, приведенные в этом разделе, демонстрируют возможности, предлагаемого формального аппарата, создавать удобные для различных приложений алгоритмические конструкции (операции), а так же конструкции,

адекватные возможностям языков программирования. При этом показана возможность, не только строить такие конструкции, но и строить их с учетом эффективности реализации на целевом языке программирования. И хотя, приведенные операции, могут использоваться для решения практических задач, главным результатом следует считать иллюстрацию практически неограниченных возможностей конструирования новых операций.

3. Эквивалентные преобразования элементарных алгоритмических конструкций

Повышение качества любой программы, достигается наиболее результативно на этапе разработки алгоритма этой программы. А наиболее эффективным средством преобразования является формальный аппарат, с помощью которого описывается алгоритм. Таким аппаратом является расширенная алгебра алгоритмов с развитой системой соотношений, позволяющей осуществлять глубокие эквивалентные преобразования.

Эквивалентными, в данном случае, считаются такие, и только такие алгоритмические конструкции, которые описывают действия, приводящие к одинаковым результатам.

В данной работе будем рассматривать соотношения, характеризующие общие свойства регулярных схем (РС), и продемонстрируем возможности эквивалентных преобразований, широко используемых элементарных алгоритмических конструкций (некоторые аспекты этой проблемы рассмотрены в [9]).

Рассмотрим возможность разложения операции α – дизъюнкции на последовательные фильтры:

$$[\alpha](A \vee B) = [\alpha](A) * [\neg\alpha](B).$$

Отличие правой части выражения от левой заключается в том, что в правой части логическое условие α проверяется дважды. И после выполнения оператора А, в случае истинности последовательного фильтра по условию α, этот оператор может повлиять на состояние логического условия, нарушив эквивалентность выполнения. Наложив ограничение на оператор А такое, что он не изменяет логического условия α, обеспечим эквивалентность преобразования. Если выполнить такое ограничение не удается оператор можно переписать в виде

$$[\alpha](A \vee B) = [\neg\alpha](B) * [\alpha](A),$$

наложив аналогичные ограничения на оператор В. Исходя из этого, можно сделать следующий вывод. В том случае, когда на условие α влияет один из двух операторов А или В, указанное ограничение можно снять за счет выбора порядка следования этих операторов.

Такое разложение применимо и для более общего случая операции α_3-дизъюнкция, которую запишем в виде

$$[\alpha](A \vee B \vee C) = [\alpha](A) * [\vec{\alpha}](B) * [\vec{\vec{\alpha}}](C),$$

с аналогичными ограничениями на входящие в выражение операторы и на порядок их следования.

В том случае, когда в качестве логического условия фигурирует фиксирующее условие *f*, никаких ограничений на разложение α_3-дизъюнкции и α – дизъюнкции на последовательные фильтры не накладывается, так как фиксирующие условия по определению не зависят от выполнения операторов и, таким образом:

$$[f](A \vee B) = [f](A) * [\neg f](B),$$

$$[f](A \vee B \vee C) = [f](A) * [\vec{f}](B) * [\vec{\vec{f}}](C).$$

Рассмотрим возможности преобразования следующих вложенных α-дизъюнкций и α – итераций:

1. $[\alpha]([\alpha](A \vee B) \vee C) =$

разложив α-дизъюнкции по последовательным фильтрам, получаем

$$= [\alpha]([\alpha](A)) * [\alpha]([\neg\alpha](B)) * [\neg\alpha](C) =,$$

преобразовав выражение в соответствии с соотношениями (5, 6) получаем

$$= [\alpha](A) * E * [\neg\alpha](C) =,$$

исключив, тождественный оператор в соответствии с (1) и свернув фильтры в α-дизъюнкцию получаем такой эквивалентный результат:

$$= [\alpha](A \vee C).$$

2. $[\alpha]([\beta](B \vee A) \vee A) = [\alpha \wedge \beta](B \vee A)$.

Эквивалентность такого преобразования становится очевидной, если обратить внимание на тот факт, что оператор B выполняется только при истинности условий α и β, а оператор A во всех остальных случаях.

3. $[\alpha](A \vee [\beta](A \vee B)) = [\alpha \vee \beta](A \vee B)$.

В данном случае доказательство аналогично предыдущему случаю, так как оператор B выполняется только при ложных значениях условий α и β, а оператор A во всех остальных случаях.

4. ${}_{\alpha}\{{}_{\beta}\{A\}\} = [\alpha]({}_{\beta}\{A\})$.

Доказательством эквивалентности является утверждение о том, что условие α проверяется однократно, т. е. по завершению выполнения оператора A оба логические условия α и β принимают ложное значение. Докажем справедливость этого утверждения от противного. Полагая, что по завершению вложенного цикла условие α истинно и применяя последовательно соотношения (2, 4, 3) получаем

${}_{\alpha}\{{}_{\beta}\{A\}\} =_{\alpha} \{{}_{\beta}\{A\} \neg \beta\} =_{\alpha} \{{}_{0}\{A\}\} =_{\alpha} \{E\} = N,$

т. е. конструкция вырождается в неопределенный (ошибочный) оператор и, таким образом, эквивалентнось преобразования доказана.

5. $[\alpha]({}_{\beta}\{A\} \vee {}_{\beta}\{B\}) = [\alpha]({}_{\beta}\{A\}) *_{\beta} \{B\}$.

В данном случае доказательством служит тождественное соотношение (2), в соответствии с которым выполнение в цикле оператора A (при истинном α) приведет к невыполнению в цикле оператора B, а при ложном α будет пропущено выполнение в цикле оператора A.

Данный раздел иллюстрирует возможности преобразования РС за счет использования соотношений, характеризующих их общие свойства, без привлечения соотношений, отражающих специфику предметной области. Открытой и весьма актуальной остается проблема автоматизации формальной трансформации регулярных схем.

4. Примеры использования расширенной алгебры алгоритмов

Продемонстрируем некоторые возможности построенной алгебраической системы на простейших примерах, которые выбраны таким образом, чтобы иллюстрировать эти возможности, не загромождая процесс восприятия сложностями алгоритмов как таковых.

Пример 1.

Первой рассмотрим задачу обработки массивов, которая иллюстрирует использование операции α_3 – дизъюнкция.

Пусть

$$X = \lceil\!| \, x_1 x_2 \ldots x_n *$$
$$Y = \lceil\!| \, y_1 y_2 \ldots y_n *$$

исходное состояние двух произвольных числовых последовательностей (массивов чисел), где $\lceil\!|$ – сдвоенный указатель, перемещающийся по массиву, причем указатель $|$ может перемещаться отдельно; * – маркер, отмечающий конец массива.

Необходимо сравнить одноименные элементы массивов X и Y и больший элемент скопировать на место меньшего. Совпадающие элементы из массива удаляются. Для простоты будем полагать, что все указатели синхронно перемещаются по обоим массивам.

Введем следующие элементарные операторы:

$\Rightarrow$

C – синхронно перемещает указатели на один шаг по массивам X и Y; КОП(x,y) – копирует элемент, находящийся справа от указателей из массива X в массив Y; КОП(y,x) – копирует элемент, находящийся справа от указателей из массива Y в массив X; СДВИГ(X,Y) – перемещает часть массивов X и Y (включая маркер *), находящуюся справа от указателя на одну позицию влево, удаляя, таким образом, совпадающие элементы;

и предикаты:

СРАВН(x,y) – значения которого для элементов, расположенных слева от указателей, трактуются следующим образом: 1 – элемент массива X больше элемента массива Y, 0 – элемент массива X

меньше элемента массива Y, μ – сравниваемые элементы равны;

d(*) – истинное, когда указатель достигает маркеров конца массивов.

Алгоритм обработки массивов ОБР(X,Y) запишем таким образом:

$$ОБР(X,Y)=_{d(*)}\left\{ \overrightarrow{\overrightarrow{C}}\,[СРАВН(x,y)]\,(КОП(x,y)\vee \right.$$

$$\left.\vee КОП(y,x)\vee СДВИГ(X,Y)\ )\ \right\}.$$

Для детализации построенной регулярной схемы разработаем алгоритм реализации оператора СДВИГ(X,Y). Для чего введем следующие элементарные операторы:

$\vec{C}$ – синхронно перемещает указатели | на один шаг по массивам X и Y;

$\overleftarrow{C}$ – возвращает указатели | к указателям ⌉;

ТРАНСП(x,y) – синхронно перемещает в массивах X и Y элементы, расположенные справа от указателей ⏐ (включая маркер *), на место элементов расположенных слева от указателей |;

и предикат:

d(*|) – принимающий истинное значение, когда указатель | выходит за границу (за маркер) массива.

Запишем регулярную схему в виде

$$СДВИГ(X,Y)=\underset{d(*|)}{\left\{\vec{C}*\ \ ТРАНСП(x,y)\right\}}\overleftarrow{C}.$$

Подставив, полученный оператор в исходное выражение, получаем результирующую регулярную схему:

$$ОБР(X,Y)=\underset{d(*)}{\left\{ \overrightarrow{\overrightarrow{C}}\,[СРАВН(x,y)]\,(КОП(x,y)\vee \right.}$$

$$\left.\vee КОП(y,x)\vee \underset{d(*|)}{\left\{\vec{C}\ ТРАНСП(x,y)\right\}}\overleftarrow{C}\ \right)\ \right\}.$$

Полученный алгоритм представляет собой циклическую процедуру обработки массивов X и Y, где после сравнения одноименных элементов больший из них копируется на место меньшего. В случае равенства элементов необработанные части массивов сдвигаются на одну позицию влево удаляя, таким образом, совпадающие элементы в обоих массивах. Цикл прекращает свою работу, когда указатель ⌉⌉ достигает конца массива.

Этот пример демонстрирует преимущества трехзначной логики при разработке данного алгоритма, так как легко представить, что использование двузначной логики приведет к вложенности конструкций и, таким образом, затруднит понимание алгоритма.

Пример 2.

Возможность разработки средствами САА-Р алгоритмов использующих декларативные знания (некоторые аспекты этой задачи рассмотрены в [10]) продемонстрируем на примере системы контроля за состоянием некоего гипотетического парового котла.

В качестве базовой выберем наиболее простую и распространенную продукционную модель представления знаний [11], в которых знания (используется простейший вариант) представляются с помощью правил вида

ЕСЛИ условие ТО реакция.

При этом в качестве фактов (посылок) будем использовать фиксирующие условия, а в качестве правил последовательные фильтры.

Допустим, что система вводит данные с датчиков давления и температуры, анализирует введенную с них информацию и реализует алгоритм управления в соответствии со следующими правилами:

правило № 1

ЕСЛИ «аварийная ситуация » И «давление превысило норму»

ТО «открыть аварийный клапан»

правило № 2

ЕСЛИ «нормальная ситуация» И «температура нормальная»

ТО «закрыть клапан»

правило № 3

ЕСЛИ «предаварийная ситуация» И «температура растет»

ТО «аварийная ситуация»

правило № 4

ЕСЛИ «температура превысило норму» И «давление растет»

ТО «предаварийная ситуация»

правило № 5

ЕСЛИ «давление нормальное» ИЛИ «давление ниже нормы»

ТО «нормальная ситуация»

Для построения регулярной схемы будем использовать следующие элементарные операторы:

ВВОД(DT)-ввод информации с датчиков давления и температуры;

предикаты, принимающие значения в следующей трактовке и сохраняемые таким фиксирующими условиями:

сост_d = 1 – давление превысило норму, 0 – давление ниже нормы, μ – давление нормальное, фиксирующее условие – sd;

изм_d = 1 – давление растет, 0 – давление не изменяется, μ – давление падает, фиксирующее условие – id;

сост_t = 1 – температура превысила норму, 0 – температура ниже нормы, μ – температура нормальная, фиксирующее условие – st;

изм_t = 1 – температура растет, 0 – температура не изменяется, μ – температура снижается, фиксирующее условие –it;

Регулярную схему алгоритма работы системы запишем в таком виде:

$$КОТЕЛ =_1 \langle ВВОД(DT)*$$
$$*[сост\_d](b^1 sd \vee b^0 sd \vee b^\mu sd)*$$
$$*[изм\_d](b^1 id \vee b^0 id \vee b^\mu id)*$$
$$*[сост\_t](b^1 st \vee b^0 st \vee b^\mu st)*$$
$$*[изм\_t](b^1 it \vee b^0 it \vee b^\mu it)*$$
$$*ПРАВИЛА\rangle$$

Смысл регулярной схемы состоит в следующем. Контроль работы котла осуществляется непрерывно, поэтому в данном случае использован бесконечный цикл (цикл по тождественно истинному условию, о необходимости и возможности использования которого упоминалось выше). В цикле прежде, чем использовать правила, собираются факты о состоянии объекта управления. Для этого после ввода данных (оператор ВВОД(DT)), используются операции α_3 – дизъюнкция, с помощью которых фиксируется состояние объекта управления. Далее обрабатываются правила.

Детализуем оператор ПРАВИЛА, для чего введем следующие операторы:

ОКЛ – открыть клапан;

ЗКЛ – закрыть клапан;

и промежуточные фиксирующие условия:

авс – аварийная ситуация;

предавс – предаварийная ситуация;

норм – нормальная ситуация.

Регулярная схема выглядит таким образом:

$$ПРАВИЛА =_k \{[авс \wedge sd](ОКЛ)*$$
$$*[норм \wedge \vec{st}](ЗКЛ)*$$
$$*[предавс \wedge it](b^1 авс)*$$
$$*[st \vee id](b^1 предавс)*[\vec{sd} \vee \vec{\vec{sd}}](b^1 норм)\}.$$

Работа алгоритма заключается в следующем. После установки исходных состояний фиксирующих логических условий в цикле выполняется последовательность α-фильтров. Каждый из фильтров ведет к выполнению (невыполнению) некоторого действия (оператора) или к установке (не установке) истинного значения промежуточного фиксирующего условия. Условием завершения цикла (условие k) является факт невыполнения ни одного из правил, включенных в цикл.

Воспользовавшись операцией левого умножения предиката на фиксирующее условие, заменив ею операцию α_3 – дизъюнкция, и подставив оператор ПРАВИЛА, получаем окончательный вид РС:

$$КОТЕЛ =_1 \langle ВВОД(DT)*$$
$$*[сост\_d]sd*[изм\_d]id*$$
$$*[сост\_t]st*[изм\_t]it*$$
$$*_k \{[авс \wedge sd](ОКЛ)*[норм \wedge \vec{it}](ЗКЛ)*$$
$$*[предавс \wedge it]авс*$$
$$*[st \vee id]предавс*[\vec{sd} \vee \vec{sd}]норм\} \rangle.$$

Таким образом, построен алгоритм системы слежения, в котором органично сочетается использование декларативных и процедурных знаний. Такое сочетание обладает, по крайней мере, одним очевидным преимуществом. Использование знаний, представленных в форме правил, может существенным образом повысить понятность и читабельность программной документации. Это облегчит взаимодействие специалистов (заказчиков, технологов) из разных предметных областей с разработчиками программного обеспечения. Кроме того, можно рассчитывать, на то, что правила могут использоваться в качестве спецификации алгоритмов (или их основных фрагментов), что позволит решать (по крайней мере, частично) проблему верификации алгоритмов и программ.

Заключение

Основным результатом данной работы является расширенная система алгоритмических алгебр (САА-Р) – <U, W, Ω>, построенная на основе детализации абстрактной модели функционирования ЭВМ, в которой информационное множество М операционного автомата О разбито на два подмножества статическое MS и динамическое MD.

Принципиально важными особенностями построенной САА-Р является следующее.

Введена операция α_3-дизъюнкция $\in \Omega_2$, позволяющая полностью использовать возможности трехзначной логики, обеспечивая компактность записи и эффективность некоторых классов алгоритмов.

Выделено подмножество операторов $U' \in U$, меняющих динамическую составляющую MD информационного множества М и оставляющих статическую составляющую MS неизменной. С помощью операторов из подмножества $U'' \in U$ реализуется операция левого умножения условия на оператор $B'p \in \Omega_1$, $(B' \in U'', p \in P)$, т.е. прогнозирование вычислительного процесса.

Множество логических W условий разбито на множество предикатов P и множество фиксирующих логических условий F, причем последние сохраняют свои значения не зависимо от действия любого из операторов $A \in U$. Введенная операция левого умножение фиксирующего условия на предикат $pf \in \Omega_1$ ($p \in P, f \in F$) сохраняет состояние логического условия, характеризующее текущее состояние вычислительного процесса (сохраняет предысторию), которое может быть использовано на последующих этапах вычислительного процесса. А операция умножения фиксирующего условия на оператор $b\ f \in \Omega$ $(b^x \in U', f \in F)$, позволяет задать требуемое значение фиксирующему условию. Таким образом, создано дополнительное средство управления вычислительным процессом, расширяющее, в частности, возможности построения производных алгоритмических конструкций. Кроме того, фиксирующее условие может трактоваться как элемент декларативных знаний (факт) и в этом качестве позволяет расширить область применения САА-Р на класс задач, в которых удобно использовать декларативные знания совместно с процедурными.

Построенная расширенная алгебра алгоритмов обеспечивает вышеуказанные возможности, сохраняя при этом способность формального преобразования алгоритмов с целью их оптимизации, и открывает достаточно широкие перспективы для дальнейшего развития.

Ближайшей перспективой развития САА-Р является следующее. Апробация формального аппарата на примере решения конкретных практических задач. Разработка “библиотеки” производных операций и эквивалентных преобразований алгоритмических конструкций. Использование более развитых моделей представления знаний и алгоритмов логического вывода.

Учитывая, что проектирование управляющих структур неразрывно связано с проектированием структур данных [12], в качестве следующего основного

этапа развития САА-Р назовем включение в рассмотрение данных.

Поскольку аппарат САА-Р позволяет сочетать использование декларативных и процедурных знаний, то перспективным представляется направление развития, при котором знания, представленные в виде правил, используются не только для решения внешних по отношению к алгоритму задач, а и как средство управления собственно вычислительным процессом.

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. – Киев: 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. – Киев: “Сфера”, 1998. – 310 с.
5. *Кнут Д.* Искусство программирования для ЭВМ. Получисленные алгоритмы. – М.: Мир, 1977. – Т.2. – 822 с.
6. *Брусенцов Н.П., Владимирова Ю.С.* Компьютеризация булевой алгебры // Докл. Академии наук, 2004. – Т. 395.– № 1.
7. *Брусенцов Н.П.* Кибернетика – ожидания и результаты. Политехнические чтения. Вып. 2. – М.: Знание, 2002. – С. 104 – 105.
8. *Поспелов Д.А.* Логические методы анализа и синтеза схем. Изд. 3–е, перераб. и доп. – М.: Энергия, 1974. – 368 с.
9. *Акуловський В.Г., Костенко В.В.* Перетворення елементарних алгоритмічних конструкцій за допомогою засобів алгебри алгоритмів // Вісн. АМС України. – 2006. – № 3. – С. 93 – 99.
10. *Акуловський В.Г., Костенко В.В.* Розробка алгоритмів на основі продукційної моделі подання знань // Вісн. АМС України. – 2006. – № 2. – С. 41– 46.
11. *Гаврилова Т.А., Хорошевский В.Ф.* Базы знаний интеллектуальных систем. – СпБ: Питер. – 2001. – 384 с.
12. *Вирт Н.* Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.

Получено 18.04.2007

Об авторе:

Акуловский Валерий Григорьевич,
кандидат технических наук, доцент кафедры информационных систем и технологий.
e-mail: akulovski@rambler.ru

Место работы автора:

Академия таможенной службы Украины.
49000, Днепропетровск,
ул. Дзержинского 2/4.
Тел/факс. канцелярии
(общий отдел) – (0562) 45 5596;
дом.тел. (0562) 67 5004;
моб. тел. 80509410566.
e-mail: academy@amsu.dnp.ukrpack.net