УДК 519.681

К-ЗНАЧНАЯ ЛОГИКА В РАСШИРЕННОЙ АЛГЕБРЕ АЛГОРИТМОВ

В.Г. Акуловский

Академии таможенной службы Украины,
49000, Днепропетровск, ул. Дзержинского 2/4.
Тел.: (056) 745 5596; факс: (0562) 47 1791.
E-mail: academy@amsu.dnp.ukrpack.net

Рассмотрена расширенная алгебра алгоритмов, построенная с использованием трехзначной логики. Показано, что существуют классы задач, для которых возможности приведенного алгебраического аппарата недостаточно эффективны, и данные возможности расширены путем включения в сигнатуру алгебры операций основанных на использовании k-значной логики. Развитие формального аппарата позволило обеспечить качество разработки упомянутых классов задач и наметило пути повышения надежности разрабатываемого программного обеспечения.

The expanded algebra of algorithms constructed with use of three-valued logic was considered. It was shown, that there are classes of problems for which opportunities of the resulted algebraic device are insufficiently effective, and these opportunities are expanded by inclusion in the signature of algebra of operations based on use of k-valued logic. Development of the formal device has allowed to provide quality of development of the mentioned classes of problems and has planned ways of increase of reliability of the developed software.

Введение

В современной компьютерной технике в подавляющем большинстве случаев используется двузначная логика. Это исторически сложившееся положение предопределено её сравнительной простотой и сделало её применение предпочтительным (в сравнении с другими логическими системами) с технической и экономической точек зрения. Однако существенный прогресс в развитии элементной базы компьютеров при сохранении основных архитектурных решений вызывает всё более жесткую критику устоявшейся архитектуры. В частности, в отношении использования двузначной логики в современных ЭВМ критические оценки высказаны в целом ряде публикаций например, [1, 2]. Разработчик трехзначной ЭВМ Н.П. Брусенцов, критикуя современные ЭВМ, утверждает [3]: “… 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. При этом трехзначная логика не только вполне корректна и адекватна действительности, но является даже более удобной и привычной для людей формой мышления”. Данное утверждение показано примером ветвления по знаку величины X (рис.1).

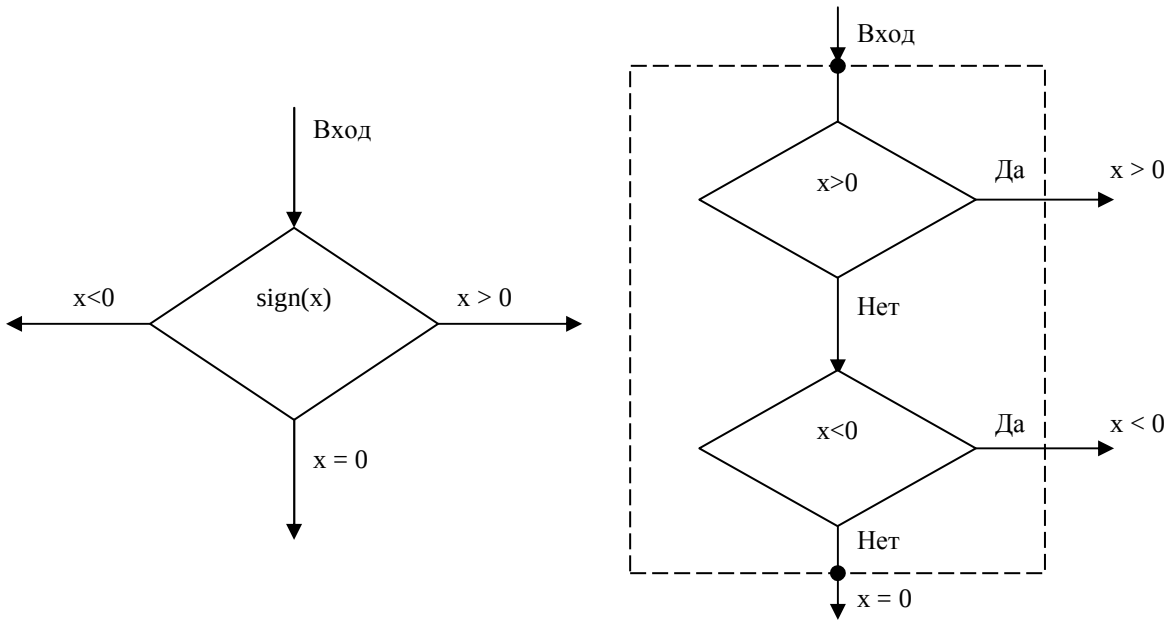

Рис. 1. Троичная схема (а), двоичная схема (б)

На рис. 1. показано, что трехзначный алгоритм компактнее и проще для понимания, чем двузначный. Тут уместно процитировать Э. Дейкстру [4]: “Первейшим достоинством алгоритма является потенциальная компактность рассуждений, на которых может основываться наше проникновение в его сущность”.

Достаточно длительный срок, в течение которого сохраняются сложившиеся архитектурные решения, естественно повлиял и на решения, принимаемые при разработке языков программирования. И повлиял негативно. Здесь мы снова можем сослаться на авторитетное мнение Э. Дейкстры [4]: “Как огорчительное следствие мы нередко обнаруживаем, что аномалии существующих вычислительных машин старательно воспроизводятся в языках программирования, причем это происходит в ущерб интеллектуальной управляемости программ, выражаемых на таком языке”.

Таким образом, использование прогрессивных решений затруднено, как архитектурой современных компьютеров, так и современными языками программирования. Однако, разработка алгоритмов это такой этап программирования, на котором влияние архитектуры компьютеров незначительно, а языков программирования ограничено. Данному этапу и посвящена предлагаемая работа.

Основы алгебры алгоритмов, как средства алгоритмизации, заложены В.М. Глушковым и развиваются его учениками и последователями [5–7]. На базе алгебры алгоритмов разработан метод структурного проектирования программ (МСПП) [6], в рамках которого алгоритмы записываются в виде регулярных схем и представляют собой суперпозицию операторов.

Автором в [8] предложена расширенная алгебра алгоритмов (САА-Р), в сигнатуре которой (наряду с другими дополнительными возможностями) использованы трехзначные логические условия рассмотренные в [9]. Далее рассмотрим данный формальный аппарат в объеме, необходимом для изложения результатов предлагаемой работы.

Основы расширенной алгебры алгоритмов

Расширенная система алгоритмических алгебр построена на основе известной модели функционирования ЭВМ, которая представляет собой схему взаимодействия двух автоматов управляющего – *U* и операционного – *O*.

Выходные сигналы *A* управляющего автомата *U* отождествляются с операторами, 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.

В результате модификации упомянутой модели ЭВМ множество *M* разбито на два подмножества $M = MS \cup MD$, где *MS* – статическая составляющая, которая постоянно определена и которая может интерпретироваться как оперативная память; *MD* – динамическая составляющая, отражающая динамические изменения состояния множества *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.

Рассматриваемый формальный аппарат, используемый для описания алгоритмов, является двухосновной алгебраической системой, основами которой являются множество операций и множество логических условий.

Пусть < U, W, Ω > – 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 статическую составляющую *MS* множество *M* состояний операционного автомата O, переводя его в новое состояние:

$$A(m) = m', \quad \forall m, m' \in MS, \quad \forall A \in U',$$

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

$$A'(m) = m', \quad \forall m \in MS, \ \forall m' \in MD, \ \forall A' \in U''$$

и только после выполнения этих операторов эта составляющая *MD* определена.

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

Множество логических условий разбито 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*.

Элементы множества *P* и *F* принимают истинные значения трехзначной логики $E_3 = \{0, 1, \mu\}$, где 0 – ложь, 1 – истина, μ – промежуточное значение. На множестве E_3 введено отношение порядка < так, что $0<\mu<1$.

К логическим операциям системы Ω_1 относятся следующие обобщенные операции:

$\neg\alpha$ – инверсия, обобщение операции отрицания;

$\vec{\alpha}$ – циклическое отрицание;

дизъюнкция и конъюнкция $\alpha \vee \beta = \max(\alpha, \beta)$, $\alpha \wedge \beta = \min(\alpha, \beta)$, со своими таблицами истинности [8, 10].

Основной отличительной особенностью алгебры алгоритмов Глушкова является наличие операции прогнозирования вычислительного процесса [11], названной операцией левого умножения оператора на логическое условие. В работе [8], такая операция, относящаяся к множеству Ω_1, введена в следующем виде.

Левое умножение $B'p$ оператора $B' \in U''$ на предикат $\in P$ 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:

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

Смысл данного оператора состоит в прогнозировании вычислительного процесса.

Кроме того, в [8] введены операции, специфичные для САА-Р. Для фиксации текущих значений состояния *M* операционного автомата введена операция левого умножения предиката на фиксирующее условие p*f* (где $\in P$, а $f \in F$) следующим образом:

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

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

Для управления состоянием фиксирующих условий к множеству *U* присоединен элементарный оператор b^x, который изменяет состояние условия $f \in F$ в результате применения операции левого умножения этого оператора на фиксирующее условие

$$b^x f(m) = m', \quad \forall m, m' \in MS, \text{ где } m' = m \cup \{f\}, \ f = x, \ x \in E_3.$$

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

Для операторов из U введены следующие операции.

Композиция *A*B* (последовательное выполнение). Композиция оператора *A* и *B* $\forall A, B \in U$ означает последовательное выполнение сначала оператора *A* затем оператора *B*. Иначе:

$$(A * B)(m) = B(A(m)) = B(m') = m'', \quad \forall m, m', m'' \in MS.$$

Операция α_3 – дизъюнкция $[\alpha(m)](A(m) \vee B(m) \vee C(m))$, ориентированная на использование трехзначных логических условий и реализуемая следующим образом:

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

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

Как частные случаи α_3 – дизъюнкции построены следующие распространенные операции.

α - дизъюнкция (соответствует программной конструкции ЕСЛИ...ТО.....ИНАЧЕ):

$$[\alpha_3(m)](A(m) \vee B(m) \vee E(m)) = [\alpha(m)](A(m) \vee B(m)) = \begin{cases} A(m), \text{если } \alpha(\mathrm{m}) = 1; \\ B(m), \text{если } \alpha(\mathrm{m}) = 0. \end{cases}$$

α – фильтрация (посл едовательная фильтрация), соответствующая программной конструкции ЕСЛИ...ТО... и является частным случаем α – дизъюнкции:

$$[\alpha(m)](A(m) \vee E(m)) = [\alpha(m)](A(m))$$

и обеспечивает выполнение оператора *A* только при истинном значении α.

α – итерация $\alpha\{A\}$ (соответствует программной конструкции WHILE). Операция α-итерации оператора А 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.

В качестве логического условия α в операторе α_3 – дизъюнкция и α – итерация могут выступать *p*-предикат, *f* – фиксирующее условие и операции левого умножения $B'p$ и *pf*.

Приведенная система алгоритмических алгебр (САА-Р), позволяет разрабатывать алгоритмы с использованием преимуществ трехзначной логики. Эти преимущества заключаются в первую очередь в повышении при прочих равных условиях компактности записи алгоритма и, таким образом, в повышении уровня его читабельности и понятности. Появление трехзначных компьютеров позволит, по-видимому, получить преимущества и в эффективности выполнения таких алгоритмов.

k-значная логика в САА-Р

В работе [5, 10] рассмотрены примеры алгоритмов и показаны преимущества трехзначной логики (по сравнению с двухзначной) при их решении. Однако существуют классы задач, решение которых хотя и осуществимо средствами САА-Р, но качество полученных решений оказывается неудовлетворительным. Очевидно, это относится к задачам, в которых вычислительный процесс должен разветвляться более, чем по трем направлениям. Например, это задачи сравнения массивов, текстов, файлов и т.д. и обработка результатов сравнения в соответствии с различными степенями их совпадения.

Определим цель данной работы, предпослав ей следующее соображение методологического характера, которое представляется автору достаточно актуальным. Как уже было отмечено, в рамках МСПП (и не только в этих рамках) алгоритм представляет собой иерархию уровней (слоев). При этом естественным и очевидным требованием к средствам описания алгоритма является обеспечение возможности поставить в соответствие каждому уровню его детализации соответствующую степень его абстрактности от деталей разработки, т.е., в процессе уточнения алгоритма задача должна разбиваться на подзадачи не только в результате произвольной декомпозиции, а и с учетом (понижением) степени абстракции для каждого уровня.

Таким образом, цель данной работы – развитие САА-Р, как с целью повышения компактности разрабатываемых алгоритмов, так и с целью удовлетворения сформулированному требованию.

Если в качестве примера взять задачу сравнения и обработки двух массивов в соответствии со степенью их совпадения и воспользоваться конструкцией, использующей двухзначную логику, то алгоритм решения данной задачи в общем виде может быть записан следующим образом:

$$[B'(m)\,p(m)](A_1(m) \vee A_2(m)), \text{ где } B'(m) \in U''.$$

Данный алгоритм выполняет такие действия. Оператор B' подсчитывает количество совпадающих элементов массива и используется в операции левого умножения оператора на предикат. Предикат *p*(*m*) сравнивает полученный результат с длиной сравниваемых массивов и вырабатывает логическое условие позволяющее оценить и обработать некоторые альтернативные варианты совпадения массивов. Например, полное совпадение и полное несовпадение текстов или полное несовпадение и частичное совпадение текстов и т.д.

При использовании трехзначной логики алгоритм запишем в виде:

$$[B'(m)\,p(m)](A_1(m) \vee A_2(m) \vee A_3(m))$$

и, таким образом, получим возможность различать три случая. Например, полное совпадение, полное несовпадение и частичное совпадение текстов или другие варианты.

В случае необходимости анализировать и обрабатывать количество степеней совпадения текста большее чем три, возможности, предоставляемые приведенными конструкциями, окажутся недостаточными. Так как, в этом случае некоторые операторы придется детализовать, т.е. осуществить их декомпозицию и перейти на

следующие уровни иерархии алгоритма. В результате алгоритм определенной степени абстракции будет ”размазан” по нескольким уровням его декомпозиции. Таким образом, компактность записи будет утрачена и, соответственно, объем возрастет, а степень понятности алгоритма понизится. При этом, требование по описанию задачи целиком на одном уровне декомпозиции, естественно не будет выполнено.

Для удовлетворения сформулированных требований необходимо привлечь дополнительные средства ранее в алгебре алгоритмов не рассматриваемые. При этом, нет никаких оснований ограничивать возможности формального аппарата только трехзначной логикой, так как повышение значности логики позволяет рассчитывать на получение дополнительных возможностей необходимых для решения приведенной задачи и вообще задач подобного рода.

Для того, чтобы иметь возможность оперировать многозначными логическими условиями, расширим имеющиеся возможности САА-Р. Будем полагать, что элементы множества логических условий *W* принимают истинные значения *k*-значной логики E_k.

К логическим операциям системы Ω_1 отнесем следующие операции [12]:

$\overline{\alpha} = \alpha + 1 \,(\mathrm{mod}\, k)$ – циклическое отрицание;

$\alpha \wedge \beta = \min(\alpha, \beta)$ – обобщение конъюнкции;

$\alpha \vee \beta = \max(\alpha, \beta)$ – обобщение дизъюнкции.

Заметим сразу, что выбранные в данном случае логические операции, в существенной степени дань традиции. Необходимый и достаточный набор операций для рассматриваемого случая приложения *k*-значной логики – вопрос дополнительного исследования.

Далее введем в множество операторов *U* специальные операторы $\Phi_k \in U$ такие, что результатом их работы является логическое условие $\alpha_k \in W$, где *k*–значность используемой логики. Определим эти операторы для двух случаев, когда $\Phi_k \in U'$ и $\Phi'_k \in U''$, следующим образом:

$$\Phi_k(m) = \alpha_k(m') = m', \quad \text{где } \Phi_k(m) \in U',$$

$$\Phi'_k(m) = \alpha_k(m) = m, \quad \text{где } \Phi'_k(m) \in U''.$$

В первом случае оператор меняет состояние операционного автомата, и логическое условие характеризует его состояние после произошедшего изменения. Во втором случае состояние операционного автомата остается неизменным.

Построенные операторы продуцируют *k*-значные логические условия и таким образом позволяют расширить возможности по разветвлению вычислительного процесса путем включения в сигнатуру операции α_k–дизъюнкция. Этот наиболее общий случай разветвления вычислительного процесса, который запишем в виде:

$$[\Phi'_k(m)](A_1(m) \vee A_2(m) \vee \ldots \vee A_k(m)) = \begin{cases} A_1(m), \text{ если } \alpha_k(m) = k-1; \\ A_2(m), \text{ если } \alpha_k(m) = k-2; \\ \ldots \\ A_k(m), \text{ если } \alpha_k(m) = 0. \end{cases}$$

То есть, выполняется один из *k* возможных операторов, который выбирается в соответствии со значением логического условия $\alpha_к$, принимающего истинные значения *k*-значной логики $E_к$. Заметим, что реализовать такую конструкцию можно, например, на языке Си, но, к сожалению, только для двухзначного случая.

Из определения операции $\alpha_к$–дизъюнкция очевидно, что поставленная задача сравнения массивов и их обработки на основании сравнения, может быть реализована с помощью одной операции, т.е. максимально компактно, на одном уровне декомпозиции алгоритма и при любом необходимом числе вариантов обработки, определяемом степенью их совпадения.

При этом построенная конструкция органично вписывается в сигнатуру алгебры.

Во-первых, двузначный и трехзначный случаи являются частными случаями рассмотренного.

Во-вторых, операции левого умножения оператора на предикат можно рассматривать, как частный случай оператора Φ_k.

Далее, применением *k*-значной логики не ограничено только операцией α_k–дизъюнкция. В частности, такая фундаментальная операция как α-итерация будет записываться подобно исходному определению в

виде: $\Phi_k(m)\{A(m)\}$, а отличие в выполнении будет заключаться в том, что циклирование будет происходить при условии $\alpha_k(m) = k-1$.

Итак, все возможности по сохранению истории вычислительного процесса сохранятся при соответствующей интерпретации операций введенных в [8].

Для фиксации текущих значений состояния *M* операционного автомата введем операцию левого умножения оператора Φ_k на фиксирующее условие f_k (где $\Phi_k(m) \in U$ а $f_k \in F$) следующим образом:

$$\Phi_k(m) f_k = \alpha_k(m) = m', \quad \forall m \in M, \forall m' \in MS, \text{ где } \quad m' = m \cup \{f_k\}, \text{ а } f_k = \alpha_k(m).$$

То есть, мгновенное состояние операционного автомата О фиксируется в виде k-значного логического условия, как элемент множества MS, позволяя сохранять историю вычислительного процесса. Причем сохраняется в виде более информативном, чем в случае двузначных и трехзначных условий.

Элементарный оператор b^x, который используется для управления состоянием фиксирующих условий, полностью сохраняет свои возможности для k-значного случая и записывается следующим образом:

$$b^x f_k(m) = m', \quad \forall m, m' \in MS, \text{ где } m' = m \cup \{f_k\}, \quad f_k = x, \quad x \in E_k.$$

Фиксирующие условия, которые, как уже отмечалось, являются средством сохранения истории вычислительного процесса и дополнительным средством управления ходом вычислений могут использоваться в этом качестве в операции $\alpha_к$ –дизъюнкция, которая в этом случае записывается в виде:

$$[f_k](A_1(m) \vee A_2(m) \vee \ldots \vee A_k(m)).$$

Осуществленное расширение формального аппарата, позволило решить поставленную задачу по обеспечению компактной записи некоторого класса алгоритмов с разбиением на уровни декомпозиции в соответствии со степенью их (уровней) абстракции.

При этом полученные результаты открывают весьма интересные, по мнению автора, перспективы, касающиеся надежности создаваемого программного обеспечения (ПО).

Актуальность проблемы повышения надежности ПО не вызывает сомнений, одним из важнейших аспектов которой является проблема контроля некорректных (недостоверных, неопределенных) данных. Этой проблеме традиционно не уделялось достаточного внимания здесь автор присоединяется к критикам архитектуры ЭВМ, современных языков и трансляторов. В архитектуре ЭВМ никаких средств повышения надежности (в указанном смысле) вообще не предусмотрено, а языки программирования предлагают весьма скромные средства, типа контроля выхода значений переменных за указанный диапазон. Исключением среди языков программирования является, по-видимому, только язык баз данных SQL, в котором трехзначная логика используется для выявления неопределенных данных.

Для того, чтобы формализовать понятие некорректных данных присоединим к информационному множеству *M* специальное неопределенное состояние ν, и будем полагать, что для любой оператора $A(m) \in U$, для которого выполняется $\nu \in m$, выполняется и $A(m) = A(\nu)$, т.е. он вырождается в неопределенный (ошибочный) оператор *N*:

$$A(\nu) = N(\nu) = \nu.$$

Неопределенный оператор переводит операционный автомат *O* в состояние, после перехода в которое, его дальнейшее функционирование будем считать неопределенным. С практической точки зрения – это аварийное завершение программы.

Чтобы избежать появления неопределенных операторов необходимо контролировать данные на корректность. Возложим функции контроля данных на оператор Φ_n, где $n = k + x$, а *k*-значность логики, необходимая для решения собственно задачи. В этом случае операция $\alpha_п$ –дизъюнкция позволит разветвить вычислительный процесс таким образом, что *k* операторов будут решать поставленную задачу, а *x* операторов будут выделены для обработки некорректных данных. Заметим, трехзначную логику можно рассматривать как частный случай, если для решения собственно задачи достаточно двузначной логики.

В данной работе рассмотрим три возможных случая некорректных данных:

- данные не определенны (не инициализированы);
- данные не актуальны (устарели);
- данные недопустимы (не предусмотрены для обработки данным оператором).

Для первых двух случаев существует проблема идентификации некорректных данных. В базах данных эта проблема решена за счет использования специального типа данных (кода) NULL, что в нашем случае не

реализуемо в связи с отсутствием “свободных” кодов. По этому самым эффективным путем является реализация аппаратной поддержки контроля данных. Автор, который не является и не считает себя специалистом в области электроники, тем не менее, взял на себя смелость высказать по этому поводу некоторые рекомендации. Для реализации контроля необходимо каждому элементарному данному (байту) сопоставить служебный бит – признак достоверности данных. В исходном состоянии он сброшен, а при занесении (записи, инициализации) данных взводится. В систему команд необходимо включить, наряду с командами чтения данных, команды их извлечения, которая сбрасывает служебный бит. Если теперь команды, читающие данные, будут контролировать служебный бит, то естественно появится возможность выявлять некорректные данные и избегать, таким образом, аварийных ситуаций, т.е. повысить надежность программного обеспечения.

Пока аппаратура желательный контроль не поддерживает, эту задачу можно решать в рамках рассматриваемого формального аппарата. Для этого необходимо связать каждое элементарное, на данном уровне декомпозиции, данное с фиксирующим логическим условием, функции которого аналогичны вышерассмотренному служебному биту. Управляя состоянием этих фиксирующих условий с помощью соответствующих команд b^x при записи и извлечении данных, можно контролировать их достоверность. Такой подход конечно весьма ресурсоемок и чреват ошибками, но в случае, когда степень ответственности, разрабатываемой программной системы, высока, возможно, им не стоит пренебрегать. Следует добавить, что снизить вероятность возникновения ошибок, можно путем построения соответствующего транслятора.

Последний случай является наиболее зависимым от семантики решаемой задачи и поэтому, контроль этого случая практически невозможно формализовать, автоматизировать и ответственность за реализацию такого контроля целиком возлагается на разработчика. При этом, однако, может быть реализован контроль данных любой степени сложности. В частности, можно контролировать не только выход некоторых значений за заданный диапазон, а и осуществлять контроль с учетом различных сочетаний и взаимоотношений данных.

Заметим, что во всех случаях оператор обработки выявленных ошибок располагается на том же уровне декомпозиции, что и операторы, решающие основную задачу.

Заключение

Результатом данной работы является расширение возможностей формального аппарата за счет использования *k*-значной логики. Полученные результаты, не претендуют на завершенность, они только намечают перспективное, по мнению автора, дальнейшее развитие разрабатываемой САА-Р. Это касается как повышения его изобразительных возможностей, так и вопросов, связанных с надежностью алгоритмов и программ.

Учитывая это, в качестве первоочередных задач, которые предстоит решить в процессе продвижения в указанном направлении, назовем следующие:

- определить необходимый и достаточный набор используемых логических операций;
- рассмотреть и реализовать возможности формального преобразования алгоритмов, построенных с использованием *k*-значной логики;
- проанализировать и расширить класс алгоритмов, для реализации которых *k*-значная логика является предпочтительной;
- разработать средства, повышающие надежность алгоритмов и программ при минимальном потреблении ресурсов компьютера;
- апробировать результаты на конкретных практических задачах.

1. *Кнут Д.* Искусство программирования для ЭВМ // Получисленные алгоритмы. – М.: Мир., 1977. – Т.2. – 822 с.
2. *Брусенцов Н.П., Владимирова Ю.С.* Компьютеризация булевой алгебры // Докл. Академии Наук. – 2004. – Т. 395. – № 1.
3. *Брусенцов Н.П.* Кибернетика – ожидания и результаты // Политехнические чтения. Вып. 2. – М.: Знание, 2002. – С. 104–105.
4. *Дейкстра Э.* Дисциплина программирования. – М.: Мир, 1978. – 265 с.
5. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Алгебра. Языки. Программирование. – Киев: Наук. думка, 1978. – 319 с.
6. *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.
7. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академперіодика, 2007. – 634 с.
8. *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.
9. *Акуловский В.Г.* Расширенная алгебра алгоритмов // Проблеми програмування. – 2007. – № 3. – С. 3–15.
10. *Акуловський В.Г., Костенко В.В.* Тризначна логіка в алгебрі алгоритмів. Вісник Академії митної служби України. – 2007. – № 1. – С. 83–88.
11. *Поспелов Д.А.* Логические методы анализа и синтеза схем. Изд. 3-е, перераб. и доп. – М.: Энергия, 1974. – 368 с.
12. *Цейтлин Г.Е.* Введение в алгоритмику. – Киев: “Сфера”. – 1998. – 310 с.