УДК 681.3

Г.Е. ЦЕЙТЛИН, Е.А. ЯЦЕНКО

ЭЛЕМЕНТЫ АЛГЕБРАИЧЕСКОЙ АЛГОРИТМИКИ И ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ СИНТЕЗ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

1. Введение

К числу важных и перспективных областей современной компьютерной науки относится алгебраическая алгоритмика, интенсивно развиваемая в Украине и за рубежом [1]. Основным объектом исследований в этой области служат алгоритмы, обработка математических построений из различных разделов классической алгебры (полугруппы, группы, кольца, поля и т.д.). Однако особый интерес представляет одно из актуальных направлений алгебраической алгоритмики, посвященное исследованиям различных алгебр алгоритмов. Весомый вклад в развитие этого направления внесен фундаментальными работами украинской алгебро-кибернетической школы [2, 3]. Ретроспектива, современное состояние и перспективы исследований в данном направлении отражены в [4]. Настоящая статья посвящена результатам, полученным в плане дальнейшей интеграции аппарата алгебры алгоритмики [5] и современных объектно-ориентированных средств [6–8].

Изложение материала подчинено следующей структуре. В разделе 2 рассмотрены средства построения семейств алгоритмических алгебр, ассоциированных с современными методами разработки программ (структурным, неструктурным, объектно-ориентированным). Особое внимание уделено взаимосвязанным формализмам (аналитическим, естественно-лингвистическим и граф-схемным). В разделе 3 приведен обзор результатов по абстрактно-автоматным моделям управления, концепции дискретного преобразователя (ДП) и распределенных систем. В терминах ДП приводится трактовка взаимодействия агентов со средой, изложенная в [9–11]. Базирующийся на аппарате алгебры алгоритмики подход к построению баз алгоритмических знаний предложен в разделе 4. Суть подхода демонстрируется на задачах символьной мультиобработки (параллельные алгоритмы сортировки, поиска, языкового процессирования).

2. Понятие клона и алгебраические спецификации программ

Данный раздел посвящен проблематике, связанной с концепцией клона – одного из важнейших понятий общей алгебры и некоторым его актуальным приложениям в современном программировании.

Под клоном [12] будем понимать универсальную алгебру: $\mathrm{K} = <O;\ СУПЕР>$, где *O* – основа – множество операций определенного типа, а *СУПЕР* – сигнатура, содержащая лишь суперпозицию операций [5, 12, 13]. Данное понятие может быть использовано для описания, построения и исследования как различных клонов (семейств алгебр), так и отдельных представителей этих семейств. При этом сигнатура операций каждой конкретной алгебры из рассматриваемого семейства служит функционально полной системой или системой образующих (СО) соответствующего клона.

Классическим примером одноосновного клона является двузначная алгебра логики – алгебра Поста (АП): $А\Pi = <L(2);\ СУПЕР>$, где $L(2)$ – множество всех булевых функций (БФ). АП – клон, который, в частности, представляет собой составную компоненту различных алгоритмических клонов, ассоциированных с современными методами программирования [14, 15].

Алгебры алгоритмов, принадлежащие соответствующим клонам и представляющие практический интерес, подлежат дальнейшим исследованиям в связи с решением для них проблемы аксиоматизации, построением теории нормальных форм, их минимизации и др.

В формализованном виде под алгоритмическим клоном будем понимать многоосновную систему: $A\Lambda\mathrm{K} = <\{O, L(2)\};\ СУПЕР>$, где O – операторные или объектные основы, состоящие из множеств неинтерпретированных схем, специфицирующих соответствующие алгоритмические компоненты. Выбор СО АЛК определяет систему алгоритмических конструкций, присущую тому или иному методу конструирования алгоритмов и программ. Так, для метода структурного программирования одну из подобных систем составляют конструкции, входящие в сигнатуру операций алгебры Дейкстры (АД) [5]. Замыкание указанной системы алгоритмических конструкций по суперпозиции порождает алгоритмический клон $КД = <\{АСС, L(2); СУПЕР>\}$, где $АСС$ – алгебра структурных неинтерпретированных схем (операторная основа КД); $L(2)$ – АП (логическая основа КД). СО КД адекватны сигнатурам операций различных алгебр алгоритмов, подобных АД. Сигнатуры операций этих алгебр могут содержать разнообразные циклические структуры, а также другие средства взаимодействия между операторами, объектами и условиями. (В частности, сигнатура систем алгоритмических алгебр (САА) и их расширений – модифицированных САА (САА-М) содержит операции, ориентированные на формализацию последовательных и параллельных вычислений (табл. 1).) Суперпозиции операций, входящих в сигнатуру САА-М, специфицирующие асинхронные операторные взаимодействия, называются параллельными регулярными схемами (ПРС).

К конструкциям САА-М, ориентированным на асинхронные параллельные вычисления, относится асинхронная дизъюнкция $A \dot{\vee} B$ – бинарная операция, состоящая в параллельном выполнении операторов A и B на различных подструктурах операционной структуры (раздел 3). При этом синхронизация параллельных ветвей осуществляется с помощью контрольных точек и зависящих от них синхронизаторов [16]. Контрольные точки представляют собой фиксированные позиции между операторами в схеме. С каждой контрольной точкой T ассоциировано условие u, ложное до тех пор, пока процесс вычислений не достиг точки T, истинное с момента достижения данной точки и не определенное при наличии аварийных остановок на пути, ведущем к точке T данной схемы. Условие u называется условием синхронизации, ассоциированным с точкой T, которая обозначается $T(u)$. Синхронизатор $S(u)$, установленный в некотором месте ПРС, осуществляет задержку вычислений в данном месте схемы вплоть до момента, когда его условие синхронизации u (сопряженное с прохождением соответствующих контрольных точек) станет истинным.

Пример 1. Посредством суперпозиции перечисленных в табл. 1 элементарных конструкций может быть построена ПРС C_1, которая специфицирует структуру алгоритмов, принадлежащих некоторому семейству и состоящих в совместном выполнении двух параллельных ветвей, осуществляющих циклическую обработку данных:

$$C_1 ::= \{[u_1]\ A\} * C\ \dot{\vee}\ \{[u_2]\ B\} * D,$$

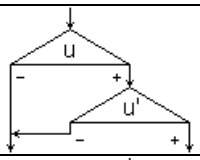
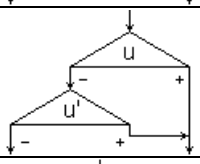
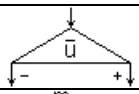
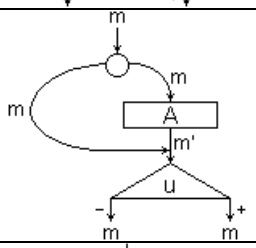
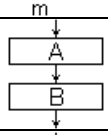
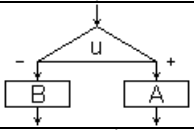
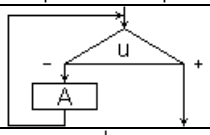
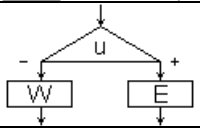
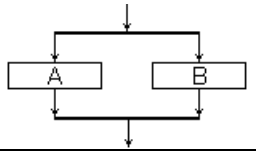
где u_1, u_2 – логические, а A, B, C, D – операторные переменные.

Схема C_1 служит примером неинтерпретированной схемы. Переход от неинтерпретированных схем к алгоритмам связан с интерпретацией операторных и логических переменных, входящих в неинтерпретированную схему. Построение интерпретаций ассоциировано с концепцией разметки обрабатываемых данных [14], которая состоит во введении специальных маркеров, фиксирующих определенные позиции в последовательностях данных, и указателей, перемещающихся по обрабатываемым последовательностям в указанных направлениях. На размеченных последовательностях данных определяются интерпретации операторных и логических переменных, входящих в схемы.

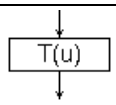
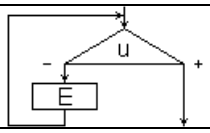
Пример 2. Рассмотрим размеченную последовательность данных, подлежащих обработке: $D: H\ У_1\ a_1\ a_2\ \ldots\ a_n\ У_2\ K$, где H и K – маркеры, отмечающие начало и, соответственно, конец последова-

тельности *Д*; a_i – элемент данных; Y_1, Y_2 – указатели. Определим на *Д* следующие предикаты и операторы: предикат $d(Y_1, Y_2) = 0$, истинный в момент слияния указателей Y_1 и Y_2; предикат $d(Y_1, Y_2) \le k$, истинный в момент, когда расстояние между указателями не превосходит коэффициента синхронизации k (положительное целое число). Коэффициент k используется для предотвращения наползания друг на друга встречных направлений обработки. К операторам относятся: контрольная точка $T(d(Y_1, Y_2) = 0)$; синхронизатор $S(d(Y_1, Y_2) = 0)$, реализующий ожидание момента завершения вычислений по первой ветви.

Таблица 1. Основные операции, входящие в сигнатуру модифицированных систем алгоритмических алгебр (САА-М)

Тип	Название операции	Форма		
		аналитическая	естественно-лингвистическая	Графовая
Логические	Конъюнкция	$u \wedge u'$ $u \,.\, u'$	*u И u'*	
	Дизъюнкция	$u \vee u'$ $u + u'$	*u ИЛИ u'*	
	Отрицание	$\overline{u}$	*НЕ(u)*	
	Прогнозирование (левое умножение условия на оператор)	$A \bullet u$	*ПОСЛЕ А УСЛОВИЕ u*	
Операторные	Композиция	$A * B$	*А ЗАТЕМ В*	
	Альтернатива	$([u]\ A, B)$	*ЕСЛИ u* *ТО А* *ИНАЧЕ В*	
	Цикл	$\{[u]\ A\}$	*ПОКА НЕ u* *ЦИКЛ А*	
	Фильтр	$\underline{u}$	*ФИЛЬТР(u)*	
	Асинхронная дизъюнкция	$A \dot{\vee} B$	*А ПАРАЛЛЕЛЬНО В*	

ISSN 1028-9763. Математичні машини і системи, 2003, № 2

	Контрольная точка	$T(u)$	*Ë u*	
	Синхронизатор	$S(u)$	*ЖДАТЬ(u)*	

Введем следующую частичную интерпретацию переменных схемы C_1: $u_1 \to d(Y_1, Y_2) = 0$; $u_2 \to d(Y_1, Y_2) \le k$; $C \Rightarrow T(d(Y_1, Y_2) = 0)$; $D \Rightarrow S(d(Y_1, Y_2) = 0)$, где $\to$ и $\Rightarrow$ обозначают интерпретацию логических, соответственно, операторных, переменных в схемах. В результате указанной интерпретации схемы C_1 получим следующую частично-интерпретированную схему:

$$C_2 ::= \{[d(Y_1, Y_2) = 0]\ A\} * T(d(Y_1, Y_2) = 0) \dot{\vee} \{[d(Y_1, Y_2) \le k]\ B\} * S(d(Y_1, Y_2) = 0).$$

Приведенная ПРС представляет класс алгоритмов, ориентированных на двустороннюю динамическую мультиобработку [16]. При динамической мультиобработке разбиения данных между ветвями могут меняться в зависимости от динамики обработки, в отличие от статической, при которой указанные разбиения остаются неизменными на протяжении всего процесса вычислений.

Условие синхронизации $d(Y_1, Y_2) = 0$ является примером замкнутого условия синхронизации [16]. Такие условия, став истинными в некоторый момент вычислений, продолжают быть истинными и далее, в отличие от локально замкнутых условий, обеспечивающих взаимодействие параллельных ветвей в циклах.

Отметим, что приведенные ПРС можно обобщить на случай асинхронной конвейерной мультиобработки [16]. Такая мультиобработка состоит в одновременном применении упорядоченной последовательности в общем случае разнотипных операторов $A_1, A_2, ..., A_n$ к потоку перерабатываемых данных $M = (m_1, m_2, ..., m_{\_})$, так что результат обработки *i*-го оператора поступает на вход $(i+1)$--му ($i = 1, 2, ..., n$-1). Передача данных от оператора к оператору может осуществляться с помощью очереди. При двусторонней конвейерной обработке левостороннюю и правостороннюю обработку могут выполнять несколько параллельных ветвей.

Асинхронные алгоритмы в современных языках программирования могут быть реализованы с помощью многопоточных программ (раздел 3).

В цикле работ, представленном в [15], [5], отражены следующие основные результаты для итеративных алгоритмических клонов, соответствующих известным семействам алгебр алгоритмов:

– исследованные клоны – алгебры континуального типа;

– каждый из клонов включает бесконечно порожденные подалгебры (с бесконечным базисом и без базиса);

– построена поверхность КД и его обобщений, имеющих *q* выходов из цикла ($q = 1, 2, ...$);

– описана совокупность максимальных подалгебр клона КЯ, канонический представитель которого – алгебра Янова;

– установлено, что КЯ – теоретико-множественный предел исследованных обобщений КД;

– описана совокупность максимальных подалгебр для клона КГ, представителем которого служит алгебра Глушкова, и ассоциированной с ним логической компоненты с операцией прогнозирования.

Полученные результаты могут быть распространены на алгоритмические клоны граф-схем и их обобщения, рассмотренные в [5]. Граф-схемные представления неинтерпретированной и частично-интерпретированной схем из примеров 1 и 2 приведены на рис. 1 и 2.

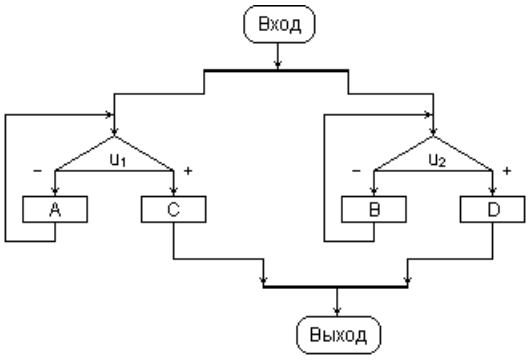


Рис. 1. Граф-схемное представление неинтерпретированной схемы из примера 1

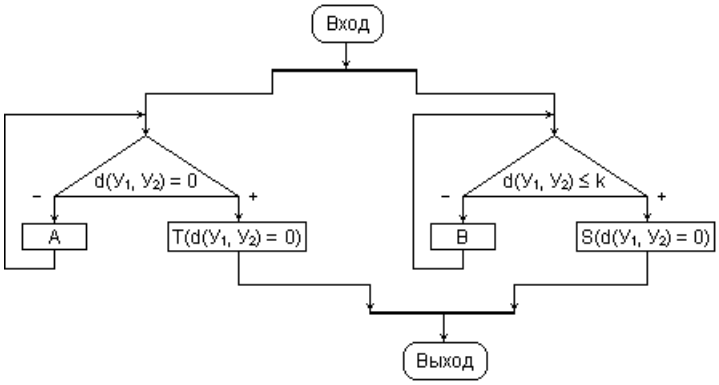


Рис. 2. Граф-схемное представление частично-интерпретированной схемы из примера 2

На основе проведенного исследования установлены критерии функциональной полноты и выразимости схем для рассмотренных клонов. Получение таких критериев имеет важное практическое значение для реализации инструментальных (программных и аппаратных) средств проектирования и синтеза алгоритмов и программ, а также их классов, ассоциированных с актуальными предметными областями [5, 17, 18]. Отметим взаимодополнительность полученных результатов по отношению к исследованиям по проблеме полноты для известных алгоритмических алгебр, в частности, для примитивных программных алгебр [19].

3. Концепция абстрактно-автоматной модели управления

В данном разделе рассматривается ряд результатов, относящихся к абстрактно-автоматным моделям управления последовательными и параллельными процессами. Приведены приложения развиваемого алгебро-автоматного подхода к проектированию параллельных программ легковесными средствами, характерными для объектно-ориентированных языков программирования, подобных Java.

Особенность функционирования достаточно сложной кибернетической системы, состоящей из управляющего – *U* и операционного – *O*, автоматов, заключается в наличии, наряду с прямым, обратного канала связи между упомянутыми автоматными структурами (рис. 3). По прямому каналу (от *U* к *O*) выполняются операторы *A* – действия для целенаправленного изменения состояния автомата *O* (объекта, подлежащего управлению); в то же время, управляющий автомат *U* по каналу обратной связи (справа налево) получает информацию о текущем состоянии автомата *O*. Эта информация поступает в виде значений логических усло-

вий *u*, с требуемой степенью полноты отражающих состояние объекта *O*. При этом автомат *U* – конечен, он, как принято считать, имеет конечное число состояний; тогда как автомат *O* – бесконечен, в силу, обычно, высокой степени сложности объекта *O*.

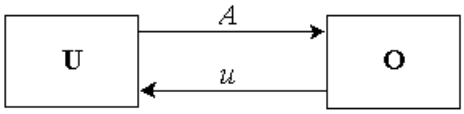


Рис. 3. Абстрактно-автоматная модель управления

В соответствии с приведенным описанием итеративного взаимодействия с *O*, управляющий автомат *U* либо переходит в заключительное состояние, завершая процесс управления, либо продолжает взаимодействие с объектом *O*. В литературе управляющий автомат часто называется дискретным преобразователем (ДП) [20]. Следует отметить, что с рассмотренной схемой взаимодействия автоматов *U* и *O* связан аппарат систем алгоритмических алгебр (САА) Глушкова [13] и их модификаций (табл. 1). В рамках указанной абстрактно-автоматной модели и теории САА получены следующие фундаментальные результаты.

Теорема 1. Произвольно выбранная кибернетическая система представима в форме рассмотренной абстрактно-автоматной модели.

Следствие. Замена в ДП функций переходов и выходов недетерминированными отображениями приводит к моделированию недетерминированных процессов. Отметим, что рассмотренная абстрактно-автоматная модель удачно сочетается с другими, в том числе, динамическими автоматными моделями и обеспечивает возможность многоаспектного интегрированного описания исследуемых процессов.

Алгоритмы функционирования ДП в приведенной концепции абстрактно-автоматной модели описываются посредством регулярных схем (РС) – операторных формул в САА.

Теорема 2 [3]**.** Любой алгоритм (в частности, программа или микропрограмма) представим функционально эквивалентным РС в САА.

Действительно, посредством РС описывается поведение ДП, который в результате анализа полученных по каналу обратной связи наборов значений логических условий (характеризующих текущее состояние операционного автомата) вырабатывает операторы, поступающие по прямому каналу на вход операционного автомата и целевым образом изменяющие его состояние.

Следствие 1. Поведение ДП произвольно выбранной абстрактно-автоматной модели динамической системы описывается посредством РС в САА.

Следствие 2. В рамках САА осуществимо прогнозирование процесса управления, представленного ДП, а также обоснование проверки правильности его функционирования.

Справедливость данного следствия вытекает из наличия в сигнатуре САА операции прогнозирования, а также из двухосновности данной системы.

Алгебраический метод, как известно, базируется на описании объектов посредством формул, составленных из операций – конструкций, удовлетворяющих фундаментальным законам, представленным, например, в виде равенств (тождеств, соотношений и т.д.). На основании упомянутых законов осуществляются преобразования формул с целью их оптимизации по тем или иным критериям, приведения к стандартным представлениям (полиномы, нормальные формы и т.д.), решения проблемы эквивалентности процессов и т. п.

Теорема 3. Посредством применения совокупности оптимизирующих преобразований, развитой в САА, осуществима оптимизация по выбранным критериям РС – аналитического описания алгоритмов функционирования ДП (с прогнозированием процесса вычислений).

Теорема 4. Наряду с аналитическими (формульными) описаниями алгоритмов функционирования ДП посредством РС, эти алгоритмы представлены и визуально в виде структурных граф-схем в соответствующих системах, изоморфных САА.

Аналитические описания алгоритмов – РС в САА, положены в основу САА-схем – многоуровневых алгоритмических проектов, оформленных в естественно-лингвистическом виде (например, на украинском или русском языках), доступном для широкого круга пользователей (не только математикам или программистам).

Теорема 5. По САА-схемам могут быть синтезированы (собраны) вручную или автоматизированным способом (с применением специально разработанного инструментария) программные модели-прототипы соответствующих АРМов и автоматизированных систем, реализованные в современных объектно-ориентированных средах.

Полученные результаты изложены в [4, 13]. Следует подчеркнуть, что с ДП неструктурного типа могут быть связаны алгоритмические алгебры, относящиеся к клону КЯ и его обобщениям (раздел 2).

Отметим, что взаимодействие ДП с объектом *O* может носить и распределенный характер. При этом подлежащая обработке информация размещена по сети ДП, обеспечивающих систематический контроль и реакцию на изменения в состоянии среды. Тем самым представляют интерес мультипроцессоры, а также ассоциированные с ними алгебро-алгоритмические системы, подобные модифицированным САА, для описания параллельных процессов, функционирующих в реальное время.

Взаимодействие ДП с объектом *O* может быть описано в терминах теории агентов и сред [9–11]. Основным понятием данной теории является понятие действия, которое изменяет состояние окружающей среды и выполняется агентом [9]. Для представления агентов и сред используется понятие транзиционной системы (ТС) [11]. ТС над множеством действий A определяется как множество S состояний вместе с отношением переходов $T \subset S \times A \times S$. Элемент $(s, a, s') \in T$ означает, что система S переходит из состояния s в состояние s', выполняя действие *a*. Последовательности переходов ТС могут быть конечными или бесконечными. В ТС также выделяют множество состояний успешного завершения функционирования системы и множество неопределенных или расходящихся состояний. Последние могут возникнуть, например, в случае, когда отношение переходов определяется с помощью алгоритма, содержащего бесконечные циклы. Транзиционные системы обычно бывают недетерминированными. Примерами ТС могут служить компоненты компьютерных систем, программы, объекты реального мира, рассматриваемые во взаимодействии один с другим и со средой, в которой они существуют. Агентом называется ТС, поведение которой отождествляется с ее состоянием [11]. Среда является специальным типом агента, в который могут быть погружены другие агенты. Погружение агента в некоторую среду приводит к ее преобразованию в новую среду. Примером среды может служить сервер в компьютерной сети или программная система, которая обрабатывает запросы, рассматриваемые как действия программ (агентов) [9].

Таким образом, понятие агента является обобщением ДП, а понятие среды – объекта *O*. ДП может быть определен как агент, реализующий операции из основы клона, соответствующего выбранному методу программирования. При этом, наряду с ДП, зависящими от конечного числа состояний, могут быть использованы и ДП над внутренней памятью (магазины, стеки, очереди и их различные сочетания [13, 21, 22]). Совокупность взаимодействующих ДП (или терминалов [16]) вместе с операционной структурой, каждая из подструктур которой ассоциирована с соответствующим ДП, представляет собой среду (в случае распределенной обработки).

ДП, осуществляющие асинхронную обработку данных, в таких языках программирования, как Java [6], $C++$ могут быть реализованы с помощью потоков (называемых также подпроцессами или легковесными процессами) (threads). Поток в многопоточных операционных системах является наименьшей единицей выполнения. Для каждого процесса ОС создает один главный поток, который является потоком выполняющихся

по очереди команд центрального процессора. При необходимости главный поток может создавать другие потоки, пользуясь для этого программным интерфейсом ОС. Все потоки, созданные процессом, выполняются в адресном пространстве этого процесса и имеют доступ к его ресурсам. Если процесс создал несколько потоков, то все они выполняются параллельно, причем время центрального процессора (или нескольких центральных процессоров в мультипроцессорных системах) распределяется между этими потоками. Каждому потоку дается определенный интервал времени, в течение которого он находится в активном состоянии.

Язык Java предоставляет возможность реализации легковесных процессов с помощью класса Thread [6]. Данный класс инкапсулирует все средства, необходимые для создания потоков, управления их состоянием и синхронизации. Для организации многопоточных программ существуют две возможности: первая предполагает создание подкласса класса Thread, вторая – создание класса, реализующего интерфейс Runnable. В этих классах переопределяется метод run, содержащий код, предназначенный для выполнения в рамках отдельного потока. Примером многопоточных программ являются сервлеты – программы на языке Java, выполняющиеся в рамках серверов, способные обрабатывать сложные клиентские запросы и динамически генерировать ответы на них. Каждое обращение клиента к серверу приводит к созданию нового потока, в котором обрабатывается запрос [6].

Потоки, работающие параллельно в многопоточной системе, могут обращаться одновременно к одним и тем же объектам в памяти, что может привести к неправильной работе программ. Разрешением этой проблемы является синхронизация потоков. Так, в Java используется синхронизация на основе мониторов Хоара, при которой в каждый конкретный момент времени доступ к совместно используемым ресурсам предоставляется только одному из потоков.

Средства синхронизации САА-М (раздел 2) могут быть реализованы, например, с помощью обмена сообщениями между потоками. В частности, Java предоставляет механизм общения между потоками, основанный на методах wait, notify и notifyAll [6]. Метод wait позволяет перевести поток в состояние ожидания, в котором он будет находиться до тех пор, пока другой поток не вызовет для него метод notify или notifyAll.

Синтез параллельных программ на языке Java по САА-схемам алгоритмов позволяет осуществлять конструктор синтаксически правильных программ, описанный в [23]. Данный инструментарий был апробирован на задачах символьной обработки.

Приведем примеры алгоритмов асинхронной обработки, которые могут быть реализованы с помощью перечисленных средств Java.

Пример 3. Рассмотрим алгоритм двусторонней асинхронной сортировки массива *ЧЕЛНОК><*, полученный путем распараллеливания маятниковой челночной сортировки [5].

Начальная разметка обрабатываемого массива имеет следующий вид: $M\colon H\, У_2\, У_1\, a_1\, a_2 \ldots a_n\, У_3\, У_4\, K$. В основу алгоритма положены схемы *ЧЕЛН>* и *ЧЕЛН<* левосторонней и правосторонней челночных сортировок массива. *ЧЕЛН>* осуществляет поиск слева направо неупорядоченного элемента $l > r$ по указателю $У_1$ и установку его на место левее $У_1$ согласно челночной стратегии обработки. При этом *ЧЕЛН<* осуществляет поиск справа налево неупорядоченного элемента $l > r$ по указателю $У_3$ и устанавливает его на место справа от $У_3$. При сближении указателей на расстояние, не превышающее коэффициента синхронизации $k = 1$, второй процесс переходит в состояние ожидания, продолжающееся до тех пор, пока левосторонний процесс не завершит обработку. Функционирование левостороннего процесса завершается в момент слияния указателей $У_1$ и $У_3$, фиксирующих слева от $У_1$ и справа от $У_3$ отсортированные каждым процессом подмассивы. ПРС алгоритма будет следующей:

ЧЕЛНОК>< [ ] *ЧЕЛН>}* T(d(У$_1$, У$_3$) = 0) ∨ {[d(У$_1$, У$_3$) ≤ k] ЧЕЛН<}* S(d(У$_1$, У$_3$) = 0) ;

*ЧЕЛН> ::= {[(l > r| У$_1$) ∨ (d(У$_1$, У$_3$) = 0)] P(У$_1$) * P(У$_2$)} * {[l < r| У$_2$] ТРАНСП(l, r| У$_2$) * L(У$_2$)} * УСТ(У$_2$, У$_1$) ;*

*ЧЕЛН< ::= {[(l > r| У$_3$) ∨ (d(У$_1$, У$_3$) = 0)] L(У$_3$) * L(У$_4$)} * {[l < r| У$_4$] ТРАНСП(l, r| У$_4$) * P(У$_4$)} * УСТ(У$_4$, У$_3$) ,*

где $l > r|У_i$ – условие, истинное, если указанное отношение выполняется для пары элементов, расположенных соответственно слева и справа от указателя $У_i$; P, L – операторы сдвига указателей на один элемент вправо и влево соответственно; $ТРАНСП(l, r|\ У_i)$ – оператор перестановки элементов слева и справа от указателя $У_i$; $УСТ(У_i, У_j)$ – установка указателя $У_i$ в позицию, в которой находится указатель $У_j$ ($i, j = 1, 2, ..., 4$).

Отметим, что посредством замены структур данных и переориентации алгоритмов сортировки могут быть получены соответствующие алгоритмы поиска [5, 24].

Пример 4. Приведем алгоритм асинхронного поиска применительно к задаче поиска документов в Web [25]. Поисковая система, в которой реализуется рассматриваемый алгоритм, состоит из двух частей: программы-индексатора, осуществляющей подготовку информации для последующего поиска, и программы, непосредственно осуществляющей поиск и выдачу результатов обработки запросов пользователей.

Индексатор просматривает Web-документы и сохраняет о них информацию в двух файлах. Первый файл содержит пронумерованный список всех просмотренных документов с указанием их места расположения (адреса), названия, размера, даты создания, описания (фразы из документа). Во втором (индексном) файле хранятся записи о словах, содержащихся в документах, с указанием номера документа и количества вхождений слова в данный документ.

Программа поиска ищет в индексном файле каждое слово из запроса пользователя, составляет и обрабатывает список документов, удовлетворяющих запросу, и выводит их на экран. При этом может осуществляться параллельная обработка нескольких запросов.

Опишем алгоритм параллельного поиска документов по ключевым словам, заданным в запросах. Поиск производится в индексном файле $Ф_u$: $ПЛ a_1\ a_2\ ...\ a_n\ У_2\ К$, где ПЛ и К – маркеры, отмечающие соответственно начало и конец файла; a_i – запись, содержащая информацию об *i*-м слове. Поиск записей в файле $Ф_u$ осуществляется по массиву запросов M,: $н\ \pi_1\ \pi_2\ ...\ \pi_s\ \oplus$, где каждый запрос $\pi_i: Д \to E_2$ представляет собой предикат, определенный на множестве *Д* элементов файла $Ф_u$ и принимающий значения из множества $E_2 = \{0, 1\}$ (0 – ложь, 1 – истина); Н и $\oplus$ – маркеры. Если $\pi_i(a) = 1$ для некоторого элемента $a \in Д$, то будем говорить, что элемент a удовлетворяет запросу π_i.

Схема алгоритма состоит из s параллельных ветвей $П_i$, каждая из которых реализует последовательный поиск записей, удовлетворяющих *i*-му запросу:

$$nn ::= \overset{s}{\underset{i=1}{\dot{\vee}}}\ n_i,$$

где

$$n_i ::= УУТ(У_i, ПЛ)*\{[\delta \vee d(У_i, К([\lambda]\ A, E)* P(У_i)\}*$$

$$*\ ([\varphi]\ E, NOT)* T(\alpha_i)*([i = s]\ S(\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_{s-1})* B, E),$$

где δ – условие, истинное, если в процессе поиска в файле Φ_u уже были найдены записи для всех слов из запроса; λ – условие, истинное, если элемент файла Φ_u, обозреваемый указателем Y_i (расположенный непосредственно справа от него), удовлетворяет *i*-му запросу; φ – условие, истинное, если в процессе сканирования по файлу уже была найдена по крайней мере одна запись, удовлетворяющая запросу; *A* – оператор занесения в список записей о документах, содержащих слово, обозреваемое указателем Y_i; *NOT* – оператор реакции на отсутствие в файле Φ_u удовлетворяющих запросу записей; $T(\alpha_i)$ – контрольная точка; α_i – условие синхронизации; $S(\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_{s-1})$ – синхронизатор, выполняющийся в *s*-й ветви и реализующий ожидание момента завершения вычислений в ветвях $i = 1, 2, ..., s\text{-}1$; B – оператор вывода списка документов, удовлетворяющих запросам.

Суть РС Π_i, обрабатывающей *i*-й запрос, состоит в сканировании указателя Y_i в направлении слева направо по файлу Φ_u вплоть до нахождения записей обо всех словах из запроса или достижения Y_i маркера K. При обнаружении элемента, удовлетворяющего запросу, выполняется оператор A со сдвигом указателя на запись вправо с продолжением процесса сканирования в поиске остальных вхождений слов, удовлетворяющих запросу. Если таковые в файле не найдены, то выполняется оператор *NOT*, сигнализирующий об их отсутствии. По завершении функционирования в s-й ветви осуществляется синхронизация – ожидание завершения вычислений в остальных ветвях. После чего выводится список документов, удовлетворяющих запросам.

4. Объектно-ориентированное проектирование предметных областей и баз алгоритмических знаний

В данном разделе предлагается подход к проектированию предметных областей и баз алгоритмических знаний на основе понятия алгебры алгоритмики и использования ПИК-технологии.

В основу предлагаемых алгебраических средств проектирования алгоритмических знаний из различных предметных областей положена концепция алгебры алгоритмики. Эта алгебра представляет собой двухуровневую систему:

– в основу верхнего уровня положена теория клонов. В зависимости от поставленной задачи, выбранного метода разработки классов алгоритмов, технологической среды программирования осуществляется построение и исследование требуемой алгебры алгоритмов (АА) из семейства, специфицированного соответствующим клоном. Сигнатура построенной АА удовлетворяет теореме о функциональной полноте для данного клона, являясь, тем самым, его СО. Построенная АА может быть исследована с привлечением соответствующего алгебраического аппарата (раздел 2);

– на нижнем уровне (интерпретационном) осуществляется построение вполне конкретной прикладной алгебры алгоритмов (ПАА), ориентированной на представление алгоритмических знаний о выбранной предметной области. Схема образования ПАА базируется на интерпретации элементарных операторных и предикатных переменных для АА, построенной на верхнем уровне. В свою очередь, разработка совокупностей интерпретаций сопряжена с построением многоосновных (многосортных) алгебраических систем, ассоциированных с АТД, механизмами классификации и наследования, характерными для объектно-ориентированного подхода, положенного в основу современных и перспективных информационных технологий [5, 7].

Следует отметить универсальность разработанного формализма – возможность его применения для проектирования самых различных алгоритмических моделей предметных областей.

Полученные в [5] результаты отражают сущность процесса представления алгоритмических знаний, апробированного при построении классификационного графа, который характеризует взаимосвязи между различными стратегиями (неинтерпретированными и частично интерпретированными схемами алгоритмов). Вершинам графа соответствуют стратегии, характеризующие классы алгоритмов (интерпретированных схем), а дугам – переходы от одних стратегий к другим посредством соответствующих метаправил конструирования. В [5] аппарат алгебры алгоритмов апробирован для решения задач символьной обработки (сортировка, поиск в многоуровневых файлах, языковое процессирование).

Для представления алгоритмических знаний может быть также применена ПИК-технология повторного использования компонент [26]. Повторно используемыми компонентами (ПИК) называются элементы знаний о прошлом опыте разработки систем программирования, которые могут быть использованы другими разработчиками и адаптированы для создания новых систем. Компоненты конструируются как абстракции, которые имеют две части – видимую и скрытую. Видимая часть является спецификацией тех знаний, которые необходимы для использования ПИК. Скрытая часть содержит детали реализации компонент, невидимые на уровне их спецификации. Аппарат алгебры алгоритмики и, в частности, описание алгоритмических знаний посредством схем представляют собой формализм, ориентированный на ПИК-технологию построения алгоритмических знаний.

Компоненты для представления специализированных знаний из различных предметных областей предлагается называть капсулами знаний (КЗ). База знаний, содержащая КЗ, ориентированные на решение задач асинхронной символьной обработки [24, 27], может содержать компоненты следующих типов:

1) компоненты, соответствующие стратегиям обработки. При этом на нижнем уровне описываются операторные и предикатные средства, входящие в сигнатуру САА-М. Такое описание включает их представление в естественно-лингвистической, алгебраической и визуализированной форме (табл. 1), а также отображение в целевой язык программирования. Далее приводятся неинтерпретированные схемы – суперпозиции упомянутых сигнатурных операций, описывающие классы алгоритмов асинхронной обработки. Примером такого компонента служит приведенная в примере 1 неинтерпретированная схема – суперпозиция асинхронной дизъюнкции, композиции и цикла, представляющая собой стратегию двусторонней асинхронной обработки. Компоненты, представляющие частично интерпретированные схемы, содержат суперпозиции неинтерпретированных схем, элементарных операторов и предикатов, общих для решения задач символьной обработки (сдвиги указателей, их установка в требуемые места размеченных данных, проверка достижения указателями маркеров и др.). Так, схема из примера 2 является суперпозицией неинтерпретированной схемы из примера 1, а также элементарных предикатов проверки расстояния между указателями и элементарных операторов – контрольной точки и синхронизатора. Данный компонент представляет собой стратегию динамической двусторонней мультиобработки. База знаний может быть пополнена стратегиями, которые классифицируются в зависимости от количества параллельных ветвей, обрабатывающих данные; статического или динамического распределения данных между функционирующими параллельными ветвями [16]; стратегий, используемых при последовательной обработке в каждой из ветвей: челночной, пузырьковой, Шелла и др. [5];

2) компоненты, соответствующие алгоритмам параллельной сортировки и поиска. Данные элементы базы знаний представляют собой описание интерпретированных схем, полученных из стратегий и содержащих базовые предикаты и операторы, специфичные для решения указанных задач (перестановка неупорядоченной пары элементов, обработка записи в файле и т. п.). Помимо предметной области, к которой они относятся, алгоритмы могут быть классифицированы в зависимости от стратегий асинхронной символьной обработки, которые они используют. Такими КЗ являются схемы алгоритмов сортировки и поиска, приведенные в примерах 3 и 4. В частности, алгоритм двусторонней асинхронной сортировки получен на основе стратегии из примера 2, а также стратегии последовательной челночной обработки, которая используется в каждой из параллельных ветвей.

Описание базовых элементов на целевом объектно-ориентированном языке может храниться в программных компонентах – классах, инкапсулирующих данные и средства доступа к ним, – методы, реализующие операторы и предикаты схем. Например, в случае задач сортировки в таком классе описывается обрабатываемый массив, переменные для представления указателей и маркеров. Методы реализуют обработку данных: сдвиги указателей, перестановку элементов, проверку упорядоченности элементов и т.д. В языке Java подобные классы могут быть реализованы при использовании JavaBeans [6]. JavaBean представляет собой совокупность, состоящую из одного или более классов Java, которая служит в качестве компонента повторного использования.

Пример 5. В табл. 2 приведены примеры описания в базе знаний элементарных предикатов и операторов для задач сортировки. Естественно-лингвистическая и аналитическая форма описания базисного элемента содержит в себе имена формальных параметров, обрамленные круглыми скобками. Фактические параметры, задаваемые в САА-схемах, заменяют при синтезе программы соответствующие формальные параметры в тексте реализации базисного понятия на языке программирования (Java). Признаком формального параметра в реализации служит символ "%", за которым следует порядковый номер параметра в тексте базисного оператора или предиката. Например, в базисном операторе "Сдвинуть У(k) на (n) вправо" присутствуют два параметра: номер указателя и величина сдвига, значения которых будут подставлены вместо соответствующих формальных параметров в реализации этого оператора, который имеет вид: s.mover(%1, %2) , где S – экземпляр класса, описывающего данные для задач сортировки и средства доступа к ним; mover – метод данного класса, осуществляющий сдвиг указателя вправо на указанное количество позиций. Реализации остальных базисных элементов, приведенных в таблице, также представляют собой вызовы методов.

Таблица 2. Примеры описания в базе знаний элементарных операторов и предикатов для задач сортировки

Тип	Естественно-лингвистическая форма	Аналитическая форма	Реализация на языке Java
Операторы	"Сдвинуть У(k) на (n) вправо"	Р(У(k), (n))	s.mover(%1, %2)
	"Переставить l, r по У(k)"	ТРАНСП(l, r| У(k))	s.transp(%1)
	"Установить У(i) на У(j)"	УСТ(У(i), У(j))	s.place(%1, s.getpos(%2))
	…	…	…
Предикаты	'l > r по У(k)'	l > r| У(k)	s.larger(%1)
	'Расстояние между У(i) и У(j) равно (k)'	d(У(i), У(j)) = (k)	s.mover(%1, %2, %3)
	…	…	…

5. Заключение

В работе получены новые результаты в рамках интеграции аппарата алгебры алгоритмики и современных объектно-ориентированных средств. Подчеркнем основные преимущества предлагаемого подхода, определяющие перспективы дальнейших исследований:

– универсальность и возможность создания баз знаний, связанных с решением задач, относящихся к актуальным и важным предметным областям различной ориентации и назначения;

– сочетание и взаимосвязь аналитических, естественно-лингвистических и визуализированных форм представления алгоритмических проектов направлена на удобство восприятия, обеспечение их правильности, достижение необходимых критериев качества (требуемая память, быстродействие и т.д.), модифицируемость синтезируемых программ в процессе их эксплуатации и т. п.;

– проекты алгоритмов (последовательных и параллельных) инвариантны к выбранному императивному языку программирования, могут рассматриваться в качестве документации на создаваемый программный продукт и допускают сборку программ на различных, в том числе и объектно-ориентированных, языках программирования;

– развиваемые инструментальные средства адаптируемы к различным операционным платформам для обеспечения автоматизации программирования в этих платформах [28–30];

– интеграция алгебраической алгоритмики с распространенными средствами проектирования и поддерживающими их инструментами синтеза программ (UML и его инструментарий) нацелена на дальнейшее привлечение аппарата алгебры и логики к практике современного программирования и послужит существенным стимулом развития компьютерной науки.

СПИСОК ЛИТЕРАТУРЫ

1. Ноден П., Китте К. Алгебраическая алгоритмика (с упражнениями и решениями). – М.: Мир, 1999. – 720 с.
2. Калужнин Л. А. Об алгоритмизации математических задач // Проблемы кибернетики. – 1959. – Вып. 2. – С. 51 – 69.
3. Глушков В. М. Теория автоматов и формальные преобразования микропрограмм // Кибернетика. – 1965. – № 5. – С. 1 – 10.
4. Цейтлин Г. Е. Системы алгоритмических алгебр и автоматизация программирования // Проблемы программирования. – 2002. – № 1 – 2. – С. 15 – 25.
5. Цейтлин Г. Е. Введение в алгоритмику. – Киев: Сфера, 1998. – 310 с.
6. Холл М., Браун Л. Программирование для Web. Библиотека профессионала: Пер. с англ. – М.: Вильямс, 2002. – 1264 с.
7. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++: Пер. с англ. – 2-е изд.– М.: "Издательство Бином", СПб.: Невский диалект, 2000. – 560 с.
8. Боггс У., Боггс М. UML и Rational Rose. – М.: ЛОРИ, 2000. – 582 с.
9. Letichevsky A. A., Gilbert D. R. A general theory of action languages // Cybernetics and Systems Analysis. – 1998. – N1. – P. 16–36.
10. Letichevsky A. A., Gilbert D. R. Agents and environments // In 1st International scientific and practical conference on programming, Proc. 2 – 4 September, 1998. – Kiev: Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine, 1998. – P. 32 – 44.
11. Капитонова Ю. В., Летичевский А. А., Волков В. А. Дедуктивные средства системы алгебраического программирования // Кибернетика и системный анализ. – 2000. – № 1. – С. 17 – 34.
12. Кон П. Универсальная алгебра. – М.: Мир, 1968. – 341 с.
13. Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л. Алгебра. Языки. Программирование. – 3-е изд. – Киев: Наукова думка, 1989. – 340 с.
14. Цейтлін Г. О. Алгебра логіки та конструювання програм. – Київ: Наукова думка, 1994. – 90 с.
15. Цейтлин Г. Е. Проблема функциональной полноты в итеративных мета-алгебрах // Кибернетика и системный анализ. – 1998. – № 2. – С. 28 – 45.
16. Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л. Методы символьной мультиобработки. – Киев: Наукова думка, 1980. – 252с.
17. Цейтлин Г. Е., Амонс А. А., Головин О. В., Зубцов А. Ю. Интегрированный инструментарий проектирования и синтеза классов алгоритмов и программ // Кибернетика и системный анализ. – 2000. – № 3. – С. 165 – 170.
18. Цейтлин Г. Е., Теленик С. Ф., Амонс А. А. Алгебро-логическая формализация в объектно-ориентированных технологиях // Проблемы программирования. – 2002. – № 1– 2. – С. 136 – 146.
19. Буй Д. Б., Редько В. Н. Примитивные программные алгебры вычислимых функций // Кибернетика. – 1987. – № 3. – С. 68 – 74.
20. Глушков В. М., Летичевский А. А. Теория дискретных преобразователей // Избранные вопросы алгебры и логики. – Новосибирск: Наука, 1973. – С. 5 – 39.
21. Ющенко Е. Л., Цейтлин Г. Е., Галушка А. В. Алгебро-грамматические спецификации и синтез структурированных схем программ // Кибернетика. – 1989. – № 6. – С. 5 – 16.
22. Анисимов А. В. Рекурсивные преобразователи информации. – Киев: Высшая школа, 1987. – 231 с.
23. Яяценко Е. А. Конструирование параллельных объектно-ориентированных программ // Проблемы программирования. – 2002. – № 1– 2. – С. 188 – 197.
24. Цейтлин Г. Е. Поиск и сортировка: классификация, трансформация, синтез. I, II // Автоматика и телемеханика. – 1992. – № 4. – С. 147 – 154; № 5. – С. 156 – 165.
25. Тихонов В. Поисковые системы в сети Интернет. – http://www.citforum.ru/ internet/search/searchsystems.shtml.
26. Бабенко Л. П., Лаврищева К. М. Основи програмної інженерії: Навчальний посібник. – К.: Т-во "Знання", КОО, 2001. – 269 с.
27. Цейтлин Г. Е. Проектирование алгоритмов параллельной сортировки // Программирование. – 1989. – № 6. – С. 4 – 19.
28. Погорілий С. Д., Калита Д. М., Захаров О. І. Інструментальний комплекс синтезу програм в середовищі Delphi за методом багаторівневого структурного проектування // Вісник Київського університету. Серія фізико-математичні науки. – 2000. – Вип. 1. – С. 268 – 275.
29. Погорілий С. Д., Калита Д. М. Оптимізація алгоритмів маршрутизації з використанням систем алгоритмічних алгебр // УСіМ. – 2000. – № 4. – С. 20 – 30.
30. Погорілий С. Д., Захаров О. І. Створення інструментальних засобів синтезу програм у візуальних середовищах // Проблемы программирования. – 2002. – № 1. – С. 499 – 505.