УДК 519.712.1:510.51:004.42.001

В.И. ШИНКАРЕНКО, В.М. ИЛЬМАН, В.В. СКАЛОЗУБ

СТРУКТУРНЫЕ МОДЕЛИ АЛГОРИТМОВ В ЗАДАЧАХ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ. II. СТРУКТУРНО-АЛГОРИТМИЧЕСКИЙ ПОДХОД К МОДЕЛИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Ключевые слова: *алгоритмические структуры, модель алгоритма, структурно-алгоритмический подход, отношения на алгоритмах, межмодельные преобразования.*

В первой части статьи предложены алгоритмические структуры и на их основе структурная и путевая модели алгоритмов [1]. Во второй части описано применение этих моделей в задачах прикладного программирования. Предварительно рассмотрены вспомогательные вопросы отношений между алгоритмами и взаимного преобразования структурной и путевой моделей алгоритмов.

ОТНОШЕНИЯ НА МНОЖЕСТВЕ АЛГОРИТМОВ

В практическом программировании сходство и различие алгоритмов разного назначения и представления определяются на интуитивном уровне. Задачи определения соответствия блок-схемы и программы на языке высокого или низкого уровня, существенности различия алгоритмов и другие не имеют необходимых формальных средств для их представления, анализа и решения.

Предлагается использовать алгоритмические структуры и модели алгоритмов, построенные в первой части статьи, для формализации отношений между алгоритмами.

Определение 1. Два пути алгоритма $A \in \Omega(C_A)\, P_k(A) = \sum_{i=1}^{N} A_{ik}^0$ и $P_m(A) = \sum_{i=1}^{N} A_{im}^0$ будут равны $P_k(A) \stackrel{\Delta}{=} P_m(A)$, если $(\forall i \in [1 \ldots N], (A_{ik}^0 = A_{im}^0; X(A_{ik}^0) = X(A_{im}^0); Y(A_{ik}^0) = Y(A_{im}^0)))$. Здесь под равенством образующих алгоритмов следует понимать, что это один и тот же алгоритм из V.

Определение 2. Два алгоритма A и B назовем тотально эквивалентными $A \stackrel{t}{=} B$ в алгоритмической структуре C_A, если $\forall P(A) \exists P(B) \,|\, P(A) = P(B)$ и $\forall P(B)$ $\exists P(A) \,|\, P(B) = P(A)$, т.е. $\overline{P}(A) = \overline{P}(B)$.

Отличия тотально эквивалентных алгоритмов может быть только в части именования данных.

Определение 3. Два алгоритма $A\big|_{x \in X}^{y \in Y}$ и $B\big|_{x \in X^*}^{y \in Y^*}$ назовем функционально эквивалентными $A \stackrel{f}{=} B$, если $X^* = X$, $Y^* = Y$ и при одних и тех же входных данных $x \in X$ оба алгоритма получают одни и те же выходные данные $y \in Y$.

Функционально эквивалентные алгоритмы представляют различные методы решения некоторой задачи. Например, функционально эквивалентны все алгоритмы сортировки.

Определение 4. Если при любых $x \in X^* \subset X$ результаты выполнения алгоритмов $A\big|_{x \in X}^{y \in Y}$ и $B\big|_{x \in X^*}^{y \in Y^*}$ достаточно близкие в некоторой метрике, то алгоритмы частично функционально эквивалентны $B \stackrel{f \subset}{=} A$ на множестве X^*.

[1]Начало см. в № 3, 2009.

Частичная функциональная эквивалентность алгоритмов связана с особенностями вычислительных устройств. Так, в цифровых ЭВМ вещественные числа представляются с некоторой погрешностью, вследствие чего результаты алгоритмов приближенные.

Функциональная эквивалентность не предполагает построение алгоритмов в одинаковой алгоритмической структуре.

Тотально эквивалентные алгоритмы являются и функционально эквивалентными. Обратное, в общем случае, неверно.

Определение 5. Назовем путь $P_m(A)=\sum_{i=1}^{N_m} A_{im}^0$ алгоритма A подобным пути $P_k(A)=\sum_{i=1}^{N_k} A_{ik}^0$ (с точностью до перестановки составляющих или удаления излишних) $P_m(A) \overset{\Delta}{\approx} P_k(A)$, если: $X(P_m(A))=X(P_k(A))$; $Y(P_m(A))=Y(P_k(A))$, пути как алгоритмы функционально эквивалентны, путь $P_m(A)$ может быть получен из пути $P_k(A)$ с помощью удаления одного из составляющих алгоритмов $A_{j+1,k}^0$ — $P_m(A)=\prod_{i=1}^{j} A_{ik}^0 \cdot \prod_{i=j+2}^{N_k} A_{ik}^0$, перестановки одного из составляющих $P_m(A)=\prod_{i=1}^{j} A_{ik}^0 \cdot A_{il}^0 \cdot \prod_{i=j+1}^{l-1} A_{ik}^0 \cdot \prod_{i=l+1}^{N_k} A_{ik}^0$, повторения удаления или перестановки.

Подобие путей транзитивно.

Определение 6. Алгоритм B подобен алгоритму A в алгоритмической структуре C_A ($B \approx A$), если $\forall P(A)\ \exists P(B) \big| P(A) \approx P(B)$ и $\forall P(B)\ \exists P(A) \big| P(B) \approx P(A)$.

Подобие алгоритмов отражает их непринципиальные различия, не затрагивающие идею и метод, реализованные в алгоритме, однако влияющие на их эксплуатационные характеристики. Подобные алгоритмы получаются в результате оптимизации.

Определение 7. Два алгоритма A и B назовем структурно подобными $A \overset{\text{Str}}{\approx} B$, если они функционально эквивалентны и структура одного из них может быть получена путем преобразования структуры другого в рамках одной алгоритмической структуры.

Структурное подобие связано с изменением структуры алгоритма для ее упрощения и улучшения удобочитаемости.

Определение 8. Алгоритм B назовем экспликацией алгоритма A, $B \overset{\rightarrow}{=} A$, если $\text{Str}(B \setminus C_A)$ может быть получена из $\text{Str}(A \setminus C_A)$ путем замены в ней некоторых алгоритмов из V структурой из других алгоритмов того же V ($A_i^0 \rightarrow \text{Str}(A_i^0 \setminus C_A)$) при сохранении их функциональной эквивалентности.

Экспликация позволяет работать с алгоритмом в разной степени его детализации, повышая технологичность работы с ним. Отношение может использоваться как средство формализации в экспликативном программировании [2].

Определение 9. Алгоритмы A и B назовем структурно эквивалентными $A \overset{\text{Str}}{\equiv} B$, если они построены в одной или разных алгоритмических структурах ($A \in \Omega(_1)$ и $B \in \Omega(_2)$) и их структуры $\text{Str}(A \in \Omega(C_1) \setminus (C_1 = \langle M_1, V_1, \Sigma, \Lambda \rangle))$ и $\text{Str}(B \in \Omega(C_2) \setminus (C_2 = \langle M_2, V_2, \Sigma, \Lambda \rangle))$ построены на подмножествах соответствующих множеств образующих алгоритмов $V^*(C_1) \subset V(C_1)$ и $V^*(C_2) \subset V(C_2)$, и между $V^*(C_1)$ и $V^*(C_2)$ может быть установлено такое взаимно однозначное соответствие, что при одновременной замене образующих алгоритмов из $V^*(C_2)$ в структуре $\text{Str}(B)$ на соответствующие алгоритмы из $V^*(C_1)$ может быть получена структура $\text{Str}(A)$.

В частности, структурно эквивалентными являются алгоритмы обработки данных разного типа. Они отличаются способом реализации операций доступа и обработки данных.

Определение 10. Алгоритмы $A \in \Omega(C_1)$ и $B \in \Omega(C_2)$ назовем межструктурно эквивалентными, если они функционально эквивалентны и $\exists C_3 \,|\, C_1 \prec C_3$ и $C_2 \prec C_3$ и $\exists D_1, D_2, D_3, D_4 \in \Omega(C_3) \,|\, A \overset{\text{Str}}{\equiv} D_1, B \overset{\text{Str}}{\equiv} D_2, D_1 \approx D_3, D_2 \approx D_4$ и $D_3 \overset{\rightarrow}{=} D_4$.

Отличия межструктурной и просто структурной эквивалентности заключаются в том, что в первом случае они обязательно связаны с разними АС и функционально эквивалентны.

В отношении межструктурной эквивалентности могут находиться алгоритмы, разработанные для разных исполнительных устройств, например для машины Тьюринга и на языке программирования ПАСКАЛЬ для ПЭВМ.

Определение 11. Реструктуризацией $\text{Str}(A \setminus C_1) \to \text{Str}(B \setminus C_2)$ алгоритма A в B назовем изменение его структуры с сохранением одного из отношений структурного подобия, экспликации, структурной или межструктурной эквивалентности.

Полезными в технологическом плане частными случаями реструктуризации являются введенные в алгебрах алгоритмов [3, 4] свертка, развертка, переинтерпретация и трансформация. При нисходящем проектировании алгоритмов (развертке) и обратном переходе к более абстрактному представлению (свертке) в процессе реструктуризации алгоритма сохраняется отношение экспликации, при переинтерпретации (замене базисных элементов) — структурной эквивалентности, а при трансформации (переобразовании структуры с учетом свойств операций) — структурного подобия.

МЕЖМОДЕЛЬНЫЕ ПРЕОБРАЗОВАНИЯ

Имея две модели алгоритма, структурную и путевую, естественно возникает проблема их взаимного преобразования.

Путевая модель предпочтительней для изучения межалгоритмических связей, но проблематична с точки зрения ее практической реализации.

Решение задачи построения множества путей алгоритма при наличии его структуры (назовем ее прямым межмодельным преобразованием) достаточно простое.

Задавая все допустимые комбинации значений алгоритмов выбора в структуре алгоритма, получаем все пути. Правда, при этом часто будут повторяться одинаковые пути. Количество повторений будет значительно сокращено, если рассматривать только алгоритмы выбора перед алгоритмом управления выполнением в структуре алгоритма.

Определение 12. Определителем пути алгоритма назовем последовательность значений алгоритмов выбора, стоящих перед алгоритмом управления выполнением в структуре алгоритма.

Определитель пути однозначно определяет последовательность ветвлений и соответственно последовательность выполнения образующих алгоритмов при конкретном выполнении алгоритма, т.е. конкретный путь алгоритма.

В примере 1 [1] в структуре алгоритма $\text{Str}(A)$ перед алгоритмом условного выполнения стоят алгоритмы $\widetilde{A}_8\big|_{t_1,t_4}^{t_5}$ и $\widetilde{A}_6\big|_{t_3,0}^{t_6}$ со значениями t_5 и t_6 соответственно. Определитель пути (0) соответствует пути $P_1(A)$, (1,1,0) — $P_2(A)$, (1,0,0) — $P_3(A)$ и т.д.

Обратная задача — восстановление структуры алгоритма по известному множеству путей (обратное межмодельное преобразование) достаточно сложная. Как показывает опыт восстановления грамматик [5] и графов [6], для этого необходимо наличие дополнительной информации.

Для алгоритмов такой информацией являются определители путей.

В частном случае, когда множество путей построено по структуре одного алгоритма, обратное межмодельное преобразование выполняется просто и заключается в объединении частей путей между соответствующими алгоритмами выбора, значения которых входят в определители пути.

В общем случае, когда исходное множество путей — результат операций объединения и пересечения алгоритмов, задача обратного межмодельного преобразования значительно усложняется.

ПРЕОБРАЗОВАНИЯ АЛГОРИТМОВ В ПРОЦЕССЕ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Предлагаемые модели алгоритмов позволяют изучать алгоритмическую составляющую ПО на различных стадиях его разработки. В процессе алгоритмизации, кодирования, отладки, тестирования, верификации и др. заложенные в ПО алгоритмы подвергаются различным преобразованиям. Изменения и уточнения алгоритмов связаны с необходимостью совершенствования функциональности и эксплуатационных характеристик.

Решение задач оптимизации и управления преобразованиями должны основываться на изучении соответствующих моделей. Структурно-алгоритмический подход (САП) к решению этих задач заключается в применении специальных методов и средств и основан на аппарате структурных моделей алгоритмов. При этом структурное моделирование есть неотъемлемое средство САП.

Далее рассматриваются возможности структурного моделирования процессов разработки ПО в рамках САП.

Алгоритмизация — процесс разработки алгоритма. Результатом процесса является алгоритм в некотором представлении.

Конструирование алгоритмов выполняется в некоторой БАС, далее $C^{\text{Ал}}$. Множество базовых алгоритмов в таком представлении состоит из тех алгоритмов, процесс выполнения которых общеизвестен или известен разработчику. Допускается наличие абстрактных алгоритмов. Например, в блок-схеме алгоритма может быть указан блок «сортировка массива», что имеет много известных реализаций.

В процессе алгоритмизации множество базовых алгоритмов $W^{\text{Ал}}$ включает подмножества $W^{\text{Ал}} = W^p \cup W^d \cup W^a$, где:

- W^p — множество примитивных алгоритмов, например, реализующих основные арифметические операции, сравнения, управления последовательностью выполнения и др.;
- W^d — множество производных алгоритмов, построенных на примитивных алгоритмах: $\forall A_i^{0d} \in W^d \;\; \exists \text{Str}(A_i^{0d} \setminus \langle M, W^p, \Sigma, \Lambda \rangle)$;
- W^a — множество абстрактных алгоритмов, при этом $\forall A_i^{0a} \in W^a \;\; \exists \text{Str}(A_i^{0a} \setminus \langle M, W^p \cup W^d, \Sigma, \Lambda \rangle)$, однако эта структура может быть не единственной и на момент разработки алгоритма, возможно, не определена.

Модель алгоритмизации в рамках САП — описание процесса разработки структуры алгоритма из $\Omega(C)$ на основе образующих, включающих примитивные, производные и абстрактные базовые алгоритмы.

Кодирование — процесс представления алгоритма и требуемых структур данных на языке программирования (ЯП).

Процессы алгоритмизации и кодирования могут совмещаться.

Множество базовых алгоритмов языка программирования $W^{\text{ЯП}}$ состоит из алгоритмов, реализующих управление последовательностью вычислений, математические операции и функции, операции обмена (ввода/вывода), доступ к данным, системные операции и др.

Часть базовых алгоритмов $W^{\text{ЯП}}$ является частью языка программирования, другая реализована в виде библиотек транслятора, ОС и прикладных библиотек.

Согласно САП кодирование программ — это реструктуризация алгоритмов, разработанных на этапе алгоритмизации. Реструктуризация заключается в переходе от одной АС к другой и построении межструктурно эквивалентного алгоритма. При этом $\text{Str}(A \setminus C^{\text{Ал}}) \to \text{Str}(A \setminus C^{\text{ЯП}})$ и $\forall A_i^{0\text{Ал}} \in W^{\text{Ал}}\ \exists A_i^{0\text{ЯП}} \in W^{\text{ЯП}}$: $A_i^{0\text{ЯП}} \overset{f\subset}{=} A_i^{0\text{Ал}}$, а также $\text{Str}(A \setminus C^{\text{ЯП}}) \overset{f\subset}{=} \text{Str}(A \setminus C^{\text{Ал}})$.

Такой подход позволяет формализовать описание процесса кодирования и предоставить инструментарий для сопоставления алгоритмов в различном представлении: блок-схем или схем Насси–Шнейдермана и алгоритмов на ЯП.

Трансляция программ (с точки зрения САП) — последующая реструктуризация алгоритмов подобная предыдущей. Множество образующих алгоритмов исходной АС — алгоритмы, предоставляемые ЯП, и доступные алгоритмы прикладных библиотек и ОС; образующие алгоритмы АС для представления результата трансляции — множество алгоритмов, реализуемых аппаратно процессором ЭВМ и доступных к выполнению.

Оптимизация программ (САП) — приведение к структурно-подобному алгоритму, имеющему лучшие эксплуатационные характеристики.

Структурно-алгоритмический подход позволяет формализовать и автоматизировать сопоставление структурно-подобных алгоритмов, обеспечивает поддержку разработки инструментария для обеспечения структурного подобия.

Конкретизируем некоторые задачи САП. Считая, что путь алгоритма $P(A)$ имеет некоторые эксплуатационные характеристики, такие как время выполнения на некотором исполнительном устройстве — $t(P(A))$, требования к объему памяти — $V_{\text{ОП}}(P(A))$, функциональную эффективность — $\rho(P(A))$, можно сформулировать ряд задач оптимизации и адаптации алгоритмов.

Например, традиционно задача оптимизации программ ставится следующим образом. Известно время выполнения базовых алгоритмов. Требуется найти такое преобразование структурного подобия, чтобы общее время выполнения алгоритмов, реализованных в виде программ, было минимальным. САП позволяет сформировать задачу иначе. Учитывая, что время выполнения пути алгоритма нелинейно зависит от исполнительного устройства [7, 8], требуется найти рациональные методы реструктуризации алгоритмов, чтобы время выполнения путей алгоритмов в среднем было близким к оптимальному. Более того, учитывая, что в процессе алгоритмизации, кодирования, трансляции и оптимизации алгоритм подвергается последовательности из нескольких реструктуризаций, задачу оптимизации можно обобщить. Следует искать рациональные методы всех реструктуризаций и рациональный набор образующих алгоритмов промежуточных и окончательных структур.

Решить такие задачи можно лишь при наличии необходимых средств, а именно формализмов преобразования алгоритмов, какими и являются формальные алгоритмические структуры.

АЛГОРИТМИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Рассмотрим некоторые приложения структурно-алгоритмического подхода к моделированию ПО различной направленности.

1. Сопрограммы — две или больше программ, которые выполняются поочередно частями, порядок выполнения которых определяется ими самими.

Пусть одна программа представляет собой некоторый алгоритм $A\big|_{X_1}^{Y_1}$, а вторая — $B\big|_{X_2}^{Y_2}$. Тогда сопрограмма будет представлять собой алгоритм $D\big|_{X_1\times X_2}^{Y_1\times Y_2}$, при этом $D = A \cup B$.

Сопрограммы с точки зрения САП — объединение двух или больше алгоритмов с известной структурой в заданной конкретной АС.

Задача 1 (здесь и далее речь идет об актуальных частных задачах прикладного программирования, которые можно рассматривать как объект для применения САП). Определить такой алгоритм D: Str($D = A \cup B$), чтобы эксплуатационные характеристики его были оптимальными.

2. Псевдопараллельные процессы (независимые) — две или больше программ, которые выполняются поочередно частями, порядок выполнения которых определяется внешней программой — диспетчером параллельных процессов.

Пусть несколько процессов представлены алгоритмами A_i, тогда общий алгоритм будет $D = \bigcup_i A_i \cup S$, где S — алгоритм диспетчера параллельных процессов.

Хотя известна структура всех алгоритмов A_i и S, структура алгоритма D неоднозначна и общий алгоритм определяется только множеством путей $\overline{P}(D)$.

Результат операции объединения алгоритмов неоднозначный. В случае сопрограмм неоднозначность устраняется наличием структуры результирующего алгоритма. При псевдопараллельных процессах диспетчер снижает эту неоднозначность (накладывая дополнительные ограничения на результат), не устраняя ее.

Задача 2. Определить реализацию операции объединения алгоритмов (посредством алгоритмизации диспетчера) таким образом, чтобы общий алгоритм имел оптимальные эксплуатационные характеристики.

Связные параллельные процессы отличаются от независимых наличием общих данных и синхронизацией.

В этом случае, как и для независимых параллельных процессов, общий алгоритм можно представить как $D = \bigcup_i A_i \cup S$, но на операцию объединения накладываются ограничения, связанные с синхронизацией.

Необходимым условием для построения параллельных процессов в рамках одного алгоритма является коммутативность операции композиции. Если соблюдаются условия коммутативности для композиции алгоритмов A_1 и A_2, то они могут выполняться в любом порядке, в том числе и «одновременно» (на нескольких или одном вычислительном устройстве). В структуре алгоритма этот факт отражается как $A_1 \cup A_2$. Приведем пример алгоритмов с возможностью параллельного выполнения: $A = A_1 \cdot (A_2 \cup A_3) \cdot A_4$. Здесь видно начало и конец возможного распараллеливания.

Для обеспечения синхронизации алгоритмов, выполняемых параллельно, множество базовых алгоритмов (МБА) должно содержать алгоритм (алгоритмы) синхронизации. Такие алгоритмы накладывают дополнительные ограничения на результат операции объединения алгоритмов: соответствующие алгоритмы синхронизации должны следовать непосредственно друг за другом в каждом пути результирующего алгоритма.

3. Аппаратно реализуемое ПО. Как известно, ЭВМ состоит из процессора, устройств хранения информации, устройств ввода/вывода и устройств обмена.

С точки зрения САП, каждое из устройств реализует некоторый набор базовых алгоритмов. Процессор реализует алгоритмы арифметических преобразований данных, логических операций, управления, обмена данными и т.д. Оперативная память — алгоритмы чтения и записи.

Отметим, что в структуре некоторых базовых алгоритмов устройства используются другие базовые алгоритмы как этого устройства, так и других. Так, базовый алгоритм сложения целых в формате «регистр-память» использует базовый алгоритм ОП чтения данных.

ЭВМ (САП) — устройство, реализующее множество базовых алгоритмов обработки и передачи данных в рамках определенной АС.

Алгоритмический интерфейс ЭВМ — это МБА, соответствующих командам процессора.

4. Алгоритм ОС можно представить как

$$A = A_1 \cdot \left(\left(\bigcup_{i=2}^{n} A_i \right) \cup \left(\bigcup_{i=1}^{m} B_i \right) \right),$$

где A_1 — алгоритм загрузки ОС; A_i — алгоритмы, обеспечивающие решение основных задач ОС, таких как управление ресурсами, устройствами, печатью и др.; B_i — алгоритмы прикладного ПО. Здесь A_i и B_i — алгоритмы, реализованные в виде отдельных процессов.

Множество образующих алгоритмов алгоритмической структуры, позволяющей моделировать ОС, включает МБА ЭВМ, алгоритмы драйверов устройств, API-функций и ряд других алгоритмов в библиотечных и исполнимых модулях.

Как отмечалось ранее, результат операции объединения алгоритмов неоднозначный. Один из алгоритмов ОС — диспетчер параллельных процессов — устраняет неоднозначность на фиксированном наборе объединяемых алгоритмов и фиксированной архитектуре ЭВМ.

Прикладная программа (однопроцессная) — алгоритм, структура которого построена на основе МБА ЭВМ и МБА ОС: $A = \mathrm{Str}(A \setminus \langle M, W^{\text{ЭВМ}}, \Sigma_1, \Lambda_1 \rangle \cup \cup \langle M, W^{\text{ОС}}, \Sigma_2, \Lambda_2 \rangle)$.

5. Объектно-ориентированное программирование (ООП). Класс (САП) — множество новых образующих алгоритмов, основанное на некотором МБА. Процесс проектирования ПО в рамках ООП можно представить в виде постепенного расширения W: $W = W^{\text{ОСН}} \bigcup \left(\bigcup_{i=1}^{n} A_i^{\text{КЛ}} \right)$, $W^{\text{ОСН}} = W^{\text{ЭВМ}} \cup W^{\text{ЯП}}$, $A_i^{\text{КЛ}}$ — множество алгоритмов i-го класса или $W = \{W^{\text{ОСН}}, A_1^{\text{КЛ}}, A_2^{\text{КЛ}} \ldots A_n^{\text{КЛ}}\}$. При формировании $A_i^{\text{КЛ}}$ учитывают их свойства: связность, сцепление и объем [9].

Задача 3 (рефакторинга [10]). Построить методы распределения алгоритмов по классам $A_i^{\text{КЛ}}$, чтобы свойства связности и сцепления классов были как можно лучше.

Задача 4 (оптимизации). Каким образом распределить алгоритмы по классам $A_i^{\text{КЛ}}$, реструктуризировать их или разбить на подалгоритмы, чтобы эксплуатационные характеристики общего алгоритма конкретного программного обеспечения были оптимальными по заданным критериям.

ЗАКЛЮЧЕНИЕ

Разработка алгоритмов ПО не заканчивается алгоритмизацией. Алгоритм, как предмет исследований, должен рассматриваться на всех этапах разработки и выполнения ПО с учетом того, что в процессе разработки алгоритмы претерпевают ряд видоизменений. Серьезными проблемами являются улучшение структуры алгоритма в ООП [10], адаптация алгоритмов к аппаратным средствам в процессе трансляции и выполнения.

Разработанные алгоритмические структуры позволяют выполнять моделирование ПО с учетом вышесказанного для изучения и совершенствования их свойств и эксплуатационных характеристик.

В рамках САП возможно изучение межалгоритмических и внутриалгоритмических связей, затрагивая «внешнее» алгоритмическое окружение. Реализуются возможности решения задач моделирования ПО различного назначения, разработанного на основе разных парадигм программирования. Не исключается моделирование алгоритмов функционирования исполнительных устройств и программно-аппаратной реализации алгоритмов.

Предложенный структурно-алгоритмический подход по моделированию алгоритмов позволяет в полном объеме исследовать программное обеспечение на различных стадиях разработки. В частности, показано, как структурно-алгоритмиче-

ские конструкции согласуются с основополагающими понятиями, такими как подпрограмма, сопрограмма, параллельно выполняемые программы, системное и прикладное ПО.

Унифицированное представление ПО, как некоего алгоритма, позволило формализовано сформулировать ряд задач, стимулируя развитие новых методов их решения.

Предоставляемые в рамках АС возможности формализации отношений между алгоритмами, реализованными различными средствами представления и выполняемыми различными исполнительными устройствами, в том числе и абстрактными, позволяют формулировать и решать задачи оптимизации и управления качеством алгоритмов в процессе их разработки и преобразований.

Структурные и путевые модели алгоритмов позволяют установить формальные отношения между их разноплановыми представлениями: в виде блок-схемы, программы, машины Тьюринга и др.

На основе предложенных моделей возможно рассмотрение всего многообразия программ, методов и способов их разработки с учетом структурных особенностей алгоритмов.

СПИСОК ЛИТЕРАТУРЫ

1. Шинкаренко В.И., Ильман В.М., Скалозуб В.В. Структурные модели алгоритмов в задачах прикладного программирования. I. Формальные алгоритмические структуры // Кибернетика и системный анализ. — 2009. — № 3 — С. 3–14.
2. Редько В.Н. Экспликативное программирование: ретроспективы и перспективы // Проблемы программирования. — 1998. — № 2 — С. 22–41.
3. Цейтлин Г.Е. Введение в алгоритмику.— Киев: Сфера, 1998. — 310 с.
4. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгеброалгоритмические модели и методы параллельного программирования. — Киев: Академпериодика, 2007. — 634 с.
5. Ільман В.М., Шинкаренко В.І. Структурний підхід до проблеми відтворення граматик // Проблемы программирования. — 2007. — № 1 — С. 5–16.
6. Ільман В.М., Скалозуб В.В., Шинкаренко В.І. Утворюючі системи графів // Вісник Дніпропетровського нац. ун-ту залізничного транспорту імені академіка В. Лазаряна. — 2007. — Вип. 18. — С. 85–94.
7. Шинкаренко В.И. Зависимость временной эффективности алгоритмов и программ обработки больших объемов данных от их кэширования // Математические машины и системы. — 2007. — № 2. — С. 43–55.
8. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.
9. Зиглер К. Методы проектирования программных систем. — М.: Мир, 1985. — 328 с.
10. Фаулер М. Рефакторинг: улучшение существующего кода. — СПб.: Симво-плюс, 2003. — 432 с.

Поступила 28.05.2008