ПРОБЛЕМЫ ИНФОРМАТИКИ № 4(12) 2011 г.
Журнал выходит ежеквартально, издается с 2008 г.

Учредитель журнала – Институт вычислительной математики и математической геофизики СО РАН

Журнал издается при содействии и финансовой поддержке следующих организаций:

Институты РАН:

Институт проблем информатики РАН

Университеты:

Новосибирский государственный технический университет
Сибирский государственный университет телекоммуникаций и информатики
Томский политехнический университет
Барнаульский государственный педагогический университет
Санкт-Петербургский государственный университет телекоммуникаций

Предприятия:
ООО "Инотех"' – Издательство (техническая редакция журнала)

Редакция: зам. гл. редактора по эл. версии журнала Г. В. Попков, зав. редакцией О. А. Марусина, отв. секретарь О. Г. Заварзина, корректор Л. Н. Ковалева, логист Л. В. Трофимова

Адрес редакции: 630090, г. Новосибирск, просп. Академика Лаврентьева, д. 6, ИВМиМГ СО РАН;
тел. (383) 330-96-43; e-mail: problem-info@sscc.ru, http://www.problem-info.ru.
Журнал зарегистрирован в Федеральной службе по надзору в сфере массовых коммуникаций, связи и охраны культурного наследия. Свидетельство ПИ N ФС77-32088 от 27 мая 2008 г. Подписной индекс в каталоге "Издания органов научно-технической информации–2011" ОАО «Агентство "Роспечать"» – 69980.

Отпечатано в типографии НГТУ. Адрес: 630092, г. Новосибирск, просп. К. Маркса, д. 20;
тел./факс (383) 346-08-57. Формат 60×84 1/8. Усл. печ. л. 11. Уч.-изд. л. 8,9. Печать офсетная. Тираж 500 экз. Заказ N 128. Подписано в печать 07.12.11 г.

Развитие экономики, рост объемов и количества объектов хозяйствования, естественное стремление к интеграции хозяйствующих субъектов путем объединения их в более крупные целостности (компании, корпорации, ведомства) приводя к росту больших и сложных систем, в том числе распределенных на значительных территориях. Проблемы связи, контроля и управления в таких системах становятся все более сложными и актуальными.

К числу указанных систем относятся также территориальнораспределенные системы (ТРС), в которые входят удаленные и труднодоступные объекты, характеризующиеся низким уровнем урбанизации и слаборазвитой инфраструктурой.

Для эффективного функционирования ТРС должны обладать современными средствами связи, мониторинга и управления, обеспечивающими сбор, накопление и обработку информации, поступающей с подвижных и стационарных объектов, выработку решении команд управления на различные уровни управления, осуществлять автоматизированное документирование информационных объектов, обеспечивать мониторинг состояния объектов и окружающей среды, отображение местоположения и маршрутов движения подвижных объектов на электронных картах центров контроля и управления в реальном масштабе времени.

Данной проблематике была посвящена Международная научно-практическая конференция "Интеллектуальные информационно-телекоммуникационные системы для подвижных и труднодоступных объектов", которая состоялась в Томском политехническом университете. Конференция была приурочена к 20-летию группы компаний "ИНКОМ", которая является разработчиком и поставщиком ТРС для ряда федеральных структур и ведомств, а также для ряда регионов России.

Настоящий тематический выпуск содержит результаты исследований и разработок, выполненных сотрудниками группы компаний "ИНКОМ", а также сотрудниками, аспирантами и магистрами Института кибернетики ТПУ, предваряющие и (или) сопровождающие решения по построению телекоммуникационных систем связи, мониторинга и управления территориально распределенными системами.

Заместитель директора Института кибернетики
Томского политехнического университета
д-р техн. наук проф. В. З. Ямпольский

СОДЕРЖАНИЕ

Системная информатика

Средства и системы обработки и анализа данных

Вычислительные и сетевые ресурсы

Информатика безопасных систем

Моделирование в системах информатики

Информационные ресурсы и системы

Информационные проблемы математического моделирования

АНАЛИЗ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ СТРУКТУРНО-ГРАФИЧЕСКОГО ПРЕДСТАВЛЕНИЯ

А. Ю. Демин, В. И. Рейзлин

Институт кибернетики Национального исследовательского
Томского политехнического университета, 634034, Томск, Россия

УДК 681.322:621.391

Предложены алгоритмы получения структурно-графических представлений программного обеспечения (ПО) из текста программ. Для оценки сложности по критерию связанности предлагается использовать специальный информационный граф, называемый графом потока данных (ГПД). Представление ПО в виде ГПД позволяет решать задачу распараллеливания программы по памяти. Для этого разработан и реализован алгоритм перестроения ГПД в ярусно-параллельную форму.

Ключевые слова: программное обеспечение, структурно-графическое представление, граф потока данных, распараллеливание программ.

The algorithms of making structure-graphical images of software based on the content of programs were introduced. To evaluate the complexity index according to the criterion of connectivity the special information graph is proposed to be used (called a graph of data flow GDF). Introducing software in GDP allows to solve the task of program paralleling on memory. So, the algorithm of GDF reconstructing into a tier parallel structure was developed and implemented.

Key words: software, structural and graphical representation of data flow graph, program parallelization.

В настоящее время при разработке любого программного обеспечения (ПО) используются методы анализа структуры ПО и его структурного проектирования. Высокий уровень развития вычислительной техники позволяет выполнять проектирование ПО на основе структурно-графического представления, в результате чего улучшаются такие свойства программ, как понимаемость, эффективность, сопровождаемость и т. д. Кроме того, на основе структурно-графического представления можно проводить анализ таких свойств, как распараллеливание и оптимальность ПО, а также подготовку ПО к тестированию и т. д. В то же время описанные в литературе графовые модели, отражающие структуру программы, являются ограниченными [1–4]. Например, схемы алгоритма и управляющих связей описывают только управляющие связи и не дают представления о вложенности структур и схеме изменения данных в программе.

В настоящей работе даны теоретико-множественные определения структурно-графических представлений ПО и предложен ряд алгоритмов получения таких представлений из текста программ, написанных на языке программирования высокого уровня паскаль.

Приведем основные определения структурно-графических представлений ПО.

Определение 1. Деревом типов данных называется связанный ациклический граф $D=(T, U)$, где T – множество вершин, соответствующих типам данных; U – множество дуг, отображающих связи построения типов.

Определение 2. Множество деревьев типов данных образует несвязанный ациклический граф, называемый лесом типов данных.

Заметим, что листьями дерева типов данных являются вершины, соответствующие скалярным типам данных, а внутренним вершинам ставятся в соответствие сложные типы данных.

Ниже приводится алгоритм анализа текстового представления типов данных и построения соответствующего дерева типов данных.

Алгоритм 1 (алгоритм анализа типа данных).

Шаг 1. Установить $i=0$, $T=\varnothing$, $U=\varnothing$.

Шаг 2. Читать входную строку символов.

Шаг 3. Если встретилось ключевое слово, описывающее сложные типы данных, то создать объект t_i соответствующего класса. Иначе перейти к шагу 5.

Шаг 3. Если $i=0$, то создать временную переменную $PROM=t_i$, $T=T\cup t_i$, $i=i+1$. Перейти к шагу 2. Иначе перейти к шагу 4.

Шаг 4. $T=T\cup t_i$, $U=U\cup(PROM, t_i)$, $PROM=t_i$, $i=i+1$. Перейти к шагу 2.

Шаг 5. Если встретилось ключевое слово, соответствующее простым типам, то создать объект t_i из класса скалярных типов. $T = T\cup t_i$. $U = U\cup(PROM, t_i)$. $i=i+1$. Перейти к шагу 2. Иначе перейти к шагу 6.

Шаг 6. Если считано слово, описывающее уже встреченный тип данных, то в T найти уже существующую вершину для этого типа и достроить дерево аналогичной ветвью. Перейти к шагу 2. Иначе перейти к шагу 7.

Шаг 7. Если построение ветви дерева типа данных для сложного типа закончено, то установить *PROM* на предшествующую рассмотрению вершину и попытаться построить остальные ветви с помощью шагов 2–6.

Шаг 8. Повторять шаг 7 до тех пор, пока не будет полностью построено дерево типа данных.

Шаг 9. Конец.

Множество деревьев D, полученных с помощью предложенного алгоритма, составят лес типов данных. Если в полученном лесу корни деревьев соединить дугами с дополнительно введенной вершиной, то получим одно большое дерево, описывающее все типы данных.

Определение 3. Всеобщим деревом данных называется связанный ациклический граф $E = (D', W, e_0)$, где e_0 – корень дерева; D' – множество деревьев типов данных; W – множество ребер, связывающих корни деревьев типов данных с вершиной e_0.

Определение 4. Деревом структурных операторов называется ациклический связанный граф $S=(O, U, o_0)$, где O – множество вершин, соответствующих операторам; o_0 – корень дерева; U – множество дуг, определяющих структуру вложенности операторов друг в друга.

Приведем алгоритм построения дерева структурных операторов. Входными данными для этого алгоритма является строка, содержащая текстовое представление алгоритма программы, а именно раздела реализации. Результатом работы предложенного алгоритма является дерево структурных операторов.

Алгоритм 2 (алгоритм анализа структуры операторов).

Шаг 1. Создать объект для вершины o_0. Ввести программную переменную *PROM*. Установить $PROM = o_0$, $i = 0$.

Шаг 2. Читать входную строку до тех пор, пока не встретится лексема, соответствующая какому-либо оператору или данным.

Шаг 3. Если прочитанная лексема соответствует ключевому слову, создать объект для вершины o_i, соответствующий найденному оператору, определить его тип. Установить $O = O \cup o_i$, $U = U \cup (PROM, o_i)$, $PROM = o_i$, $i=i+1$. Запомнить для o_i начало в тексте программы. Перейти к шагу 2.

Шаг 4. Если полученная лексема соответствует логическому условию в сложносоставном операторе, то создать вершину o_i, считая это условие простым оператором. Установить $O=O\cup o_i$, $U=U\cup(PROM, o_i)$, $PROM=o_i$, $i=i+1$. Определить начало для o_i. Перейти к шагу 2. Иначе перейти к шагу 5.

Шаг 5. Если полученная лексема соответствует данным, определить свойства данных: используются или изменяются либо используются и изменяются в данном выражении. Добавить в список данных объекта *PROM* ссылку на объект, соответствующий данным. Перейти к шагу 2, иначе − к шагу 6.

Шаг 6. Если полученная лексема описывает окончание структурного оператора, но не конец программы, запомнить, где закончился оператор *PROM*. Установить *PROM* на предшествующую вершину o_{i-1}. Перейти к шагу 2, иначе − к шагу 7.

Шаг 7. Если лексема соответствует концу тела, запомнить, где закончился оператор *PROM*. Конец.

Определение 5. Дерево процедур и функций – связанный ациклический граф $P=(Q, U, p_0)$, где Q – множество вершин, соответствующих процедурам и функциям; p_0 – корень, соответствующий головной программе; U – множество дуг, задающее отношение вложенности подпрограмм, т. е. для подпрограмм i и j ($i\neq j$) существует дуга $(i, j)\in U$, если подпрограмма j описана внутри подпрограммы i.

Приведем алгоритм построения дерева процедур и функций, однозначно соответствующего иерархической структуре подпрограмм в паскаль-программе.

Алгоритм 3 (алгоритм анализа структуры подпрограмм).

Шаг 1. Создать объект, соответствующий вершине p_0. Ввести промежуточную переменную *PROM*. Установить $PROM = p_0$, $i=0$.

Шаг 2. Читать входную строку до тех пор, пока не встретится какая-либо лексема.

Шаг 3. Если встреченная лексема является описанием подпрограммы (procedure или function), то создать объект соответствующего вида q_i, q_i включить в Q, дугу $(PROM, q_i)$ − в U: $Q=Q\cup q_i$, $U=U\cup(PROM, q_i)$. Установить $PROM=q_i$, $i=i+1$. Читать и заполнять имя подпрограммы, список формальных параметров. Перейти к шагу 2.

Шаг 4. Если встреченная лексема соответствует описанию данных, то перейти к анализу данных, используя шаг 1. Определить указатель в объекте *PROM* на всеобщее дерево данных, полученное в результате работы этого алгоритма. Определить список внутренних переменных и поставить им в соответствие вершины в полученном всеобщем дереве данных. Перейти к шагу 2.

Шаг 5. Если полученная лексема описывает начало тела подпрограммы, то применить алгоритм 2 для анализа структуры операторов. Определить указатель для объекта *PROM* на полученное дерево разбора. Установить *PROM* на вершину q_j, такую что $\exists(q_j,PROM)\in U$. Перейти к шагу 2.

Шаг 6. Если полученная лексема описывает тело основной программы, то аналогично шагу 5 использовать алгоритм 2. Установить указатель в объекте *PROM* на полученное дерево разбора.

Шаг 7. Конец.

Таким образом, общая структура ПО, созданного на алгоритмическом языке, после анализа его текстового описания представляется в виде дерева процедур и функций, каждой вершине которого ставятся в соответствие дерево данных и дерево структурных операторов. Фактически построено дерево синтаксического разбора программы.

В современных версиях алгоритмических языков используется расширение предложенных стандартов. Так, во всех реализациях языка программирования паскаль применяется механизм паскаль-модулей. Следовательно, необходимо определить такую структурно-графическую форму, как схема паскаль-модулей.

Определение 6. Схема паскаль-модулей есть связанный граф $M=(B, U, m_0)$, где B – множество вершин, поставленных в соответствие паскаль-модулям; m_0 – вершина, соответствующая головной программе; U – множество дуг, соответствующих межмодульным связям. При этом $(i, j)\in U$, если в модуль i в предложении USES включено имя модуля j.

Определение 7. Деревом классов называется связанный ациклический граф $C=(K, R, P, U, k_0)$, где K – множество вершин, поставленных в соответствие классам; k_0 – корень дерева, соответствующий классу-предку, который является базовым для всех других классов; R – множество вершин, соответствующих полям классов; P – множество вершин, соответствующих методам классов; U – множество дуг, задающих отношения наследственности и принадлежности полей и методов определенным классам.

Поскольку все иерархические структуры в программе можно представить в виде однотипных деревьев, для них применим единый механизм графического отображения.

В рамках реализации предлагаемой методики построение любого дерева происходит с помощью единого механизма работы с динамически строящимся деревом.

Определение 8. Динамически строящимся деревом для дерева $Y=(X, U)$ называется дерево $Y'=(X', U')$, где $X'\subset X;\ U'\subset U$, при этом дерево Y называется деревом-прототипом.

Определение 9. Динамически строящееся дерево $Y'=(X', U')$ является полностью свернутым, если $U'=\varnothing$, $X'=\{x_0'\}$, x_0' соответствует корню дерева-прототипа.

Определение 10. Динамически строящееся дерево $Y'=(X', U')$ является полностью развернутым, если $U'=U$, $X'=X$ для дерева-прототипа $Y=(X, U)$.

Первоначально динамически строящееся дерево отображается в полностью свернутом состоянии, а затем достраивается вершинами, соответствующими вершинам дерева-прототипа.

Динамически строящееся дерево может быть построено не только для дерева типов данных, дерева процедур и функций, дерева структурных операторов, дерева классов, но и для схемы паскаль-модулей.

Описанные деревья взаимосвязаны. Например, каждой вершине дерева процедур и функций соответствуют дерево структурных операторов и всеобщее дерево данных. Для отображения таких деревьев разработана методика работы с взаимосвязанными деревьями.

Механизм динамического просмотра деревьев с помощью динамически строящегося дерева позволяет выбрать необходимый уровень детализации при рассмотрении ПО.

Положение 1. *Множество вершин* $L'=\{l_i'\}$, *являющихся листьями динамически строящегося дерева* $Y'=(X', U')$ *для дерева структурных операторов* $(x_j \in X' \wedge L' \subset X'$, *но* $\neg\exists(l_i', x_j') \in U')$, *соответствует множеству фрагментов текста программы, полностью покрывающих исполняемую часть программы.*

Из положения 1 следует, что с помощью динамически строящегося дерева для дерева структурных операторов исполняющую часть программы можно декомпозировать на фрагменты. При этом программа разбивается на фрагменты, соответствующие листьям динамически строящегося дерева.

Для выбранного уровня декомпозиции программы можно получить дополнительные структурно-графические представления: блок-схему и граф потока данных.

Определение 11. Блок-схема – ориентированный граф $H=(N, U, n_0)$ с вершинами специального вида, соответствующий следующим условиям:

1) граф G имеет единственную входную вершину n_0, и в эту вершину не входит ни одна дуга;

2) граф G не содержит параллельных дуг и петель;

3) конечное множество дуг $U=\{u_f\}$, $f=1,...,F$ отражает отношение предшествования по выполнению между операторами программы;

4) конечное множество вершин $N=\{n_\mu\}$, $\mu=0,\ldots,M$ представляет собой совокупность операторов $B = \{b_\mu\}$, $\mu=0,\ldots,M$ программы, между которыми существует взаимно однозначное соответствие (каждая вершина представляется только одним оператором программы, и наоборот, каждый оператор программы в графе представляется только одной вершиной);

5) конечное множество вершин $N=\{n_\mu\}$, $\mu=0,\ldots,M$ представляет собой совокупность сложносоставных операторов $B'=\{b'_\mu\}$, $\mu=0,\ldots,M$ программы, причем между ними существует взаимно однозначное соответствие. Поскольку любой вершине n_μ соответствует сложносоставной оператор, она представляет собой группу простых операторов;

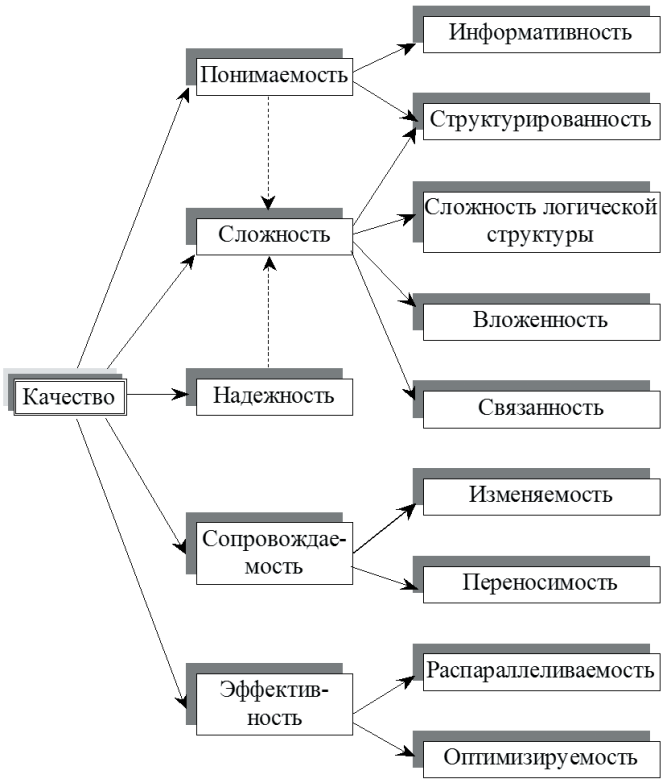
6) для изучения информационных связей в ПО строится граф потока данных.

Определение 12. Графом потока данных называется ориентированный граф $R=(N, D, U)$, где N – множество вершин, соответствующих сложносоставным операторам, представленным в виде листьев динамически строящегося дерева; D – множество вершин, соответствующих данным, используемым в программе, причем единице данных могут быть поставлены в соответствие несколько вершин; U – множество дуг, таких что $\exists(d_i, n_j) \in U$, если $d_i \in D$ используется в $n_j \in N$, либо если $d_i \in D$ изменяется в $n_j \in N$.

Определение 13. Множество вершин N из R, соответствующих сложносоставным операторам, будем называть переходами, а множество вершин D из R, соответствующих данным, − позициями.

Таким образом, структура программы представляется в виде взаимосвязанных деревьев, блок-схем и графов потока данных. Такое формальное представление позволяет проводить оценку сложности программ, уровня распараллеливаемости и т. д.

Оценку качества ПО предлагается проводить на основе системы критериев, приведенной на рисунке.



Система критериев оценки качества ПО

Часть свойств (информативность, структурированность, оптимизируемость), по которым возможно оценить ПО с помощью экспертных оценок, определены в [1]. Дадим определения свойствам ПО, которые можно оценить автоматически с помощью введенных коэффициентов.

Определение 14. Сложность логической структуры – объективная особенность программы, заключающаяся в уровне простоты представления ее логической структуры.

Определение 15. Вложенность – свойство программы, заключающееся в возможности вложения элементов друг в друга.

Определение 16. Связанность – способность программы устанавливать информационные связи между отдельными компонентами.

Определение 17. Распараллеливаемость – свойство программы, определяющее возможность ее параллельного выполнения на многопроцессорной вычислительной системе (МВС) или распределенной системе реального времени (СРВ).

Для оценки качества по критерию вложенности используется представление ПО в форме деревьев.

Определение 18. Коэффициент вложенности ν_T для дерева $T=(N, U, n_0)$ – максимальная длина пути от корня дерева n_0 к его листьям $\{l_i\}\in N$: $\nu_T = \max_i \rho[n_0, l_i]$, если $\forall j \neg\exists(l_i, n_j)\in U$.

Введем операцию присоединения деревьев друг к другу и покажем монотонность и транзитивность коэффициента вложенности для введенной операции.

Определение 19. Присоединением дерева $T_2=(N_2, U_2, n_{20})$ к дереву $T_1=(N_1, U_1, n_{10})$ по вершине n_{1k} ($n_{1k}\in N_1$) называется дерево $T_3=(N_3, U_3, n_{30})$, $T_3=T_1\oplus_k T_2$, полученное следующим образом:

1) вершина n_{1k} дерева T_1 совмещается с корнем дерева n_{20} ($n_{1k}\equiv n_{20}$);

2) множество дуг дерева T_3 является объединением множеств дуг деревьев T_1 и T_2 ($U_3=U_1\cup U_2$);

3) $N_3=N_1\cup N_2$.

Положение 2. *Если дерево T_3 получено в результате присоединения дерева T_2 к дереву T_1 по любому k, то коэффициент вложенности ν_{T3} для дерева T_3 больше или равен коэффициенту вложенности ν_{T1} для дерева T_1 и больше или равен коэффициенту вложенности ν_{T2} для дерева T_2. Если $\forall k\ T_3=T_1\oplus_k T_2$, то $\nu_{T3}\geq\nu_{T2}\wedge\nu_{T3}\geq\nu_{T1}$.*

Положение 3. *Если коэффициент ν_{T2} для дерева T_2 больше коэффициента вложенности ν_{T1} для дерева T_1 и коэффициент вложенности ν_{T3} для дерева T_3 больше коэффициента вложенности для дерева T_2, т. е. $\nu_{T1}<\nu_{T2}\wedge\nu_{T2}<\nu_{T3}$, то $\nu_{T1}<\nu_{T3}$.*

Также для оценки вложенности можно ввести средний коэффициент вложенности.

Определение 20. Средним коэффициентом вложенности μ_T для дерева $T=(N, U, n_0)$ называется величина, пропорциональная сумме длин всех путей из корня n_0 дерева T к его листьям $\{l_i\}=N$ и обратно пропорциональная количеству этих путей:

$$\mu_T=\frac{\sum_{i=1}^{N}\rho[n_0,l_i]}{N}\quad \forall j\neg\exists(l_i,n_j)\in U.$$

Для среднего коэффициента вложенности не выполняется условие монотонности.

Для оценки сложности логической структуры предлагается использовать представление ПО в форме блок-схем, построенных из динамически строящихся деревьев в полностью развернутом состоянии.

Следует отметить, что способы оценки сложности программ по блок-схемам (графам управляющих связей) широко распространены и рассматривались во многих работах. Предлагаемые ниже методы оценки основываются на этих способах.

Определение 21. Коэффициентом логической сложности программы, представленной в виде блок-схемы $H=(N, U, n_0)$ для полностью развернутого динамически строящегося дерева, называется величина

$$\xi=\sum_i P_i[n_0,n_m]\ \text{при}\ d^-(n_0)=0\wedge d^+(n_m)=0.$$

Определение 22. Цикломатическим коэффициентом сложности программы, представленной в виде блок-схемы $H=(N, U, n_0)$ для полностью развернутого динамически строящегося дерева структурных операторов, называется величина $\lambda=X-Y+L$ ($X=|U|$ – количество ребер в блок-схеме; $Y=|N|+1$ – количество вершин (вместе с n_0); L – число связанных компонент). Для блок-схемы, представляющей собой безошибочную программу, $L=1$.

Определение 23. Суммой двух блок-схем $H_1=(N_1, U_1, n_{01})$ и $H_2=(N_2, U_2, n_{02})$ называется блок-схема $H_3=(N_3, U_3, n_{03})=H_1\oplus H_2$, полученная в результате выполнения следующего алгоритма.

Алгоритм 4 (алгоритм сложения блок-схем).

Шаг 1. Исходные данные: $H_1=(N_1, U_1, n_{01})$, $H_2=(N_2, U_2, n_{02})$, где N_1, N_2 – множества вершин; n_{01}, n_{02} – начальные вершины; U_1, U_2 – множества дуг. Пусть n_{k1}, n_{k2} – конечные вершины.

Шаг 2. Добавить в H_3 все вершины из H_1, кроме конечной вершины:

$$N_3 = \bigcup_i n_{i1} \text{ при } d^+(n_{i1}) \neq 0 \quad \forall i.$$

Шаг 3. Добавить в H_3 все вершины из H_2, кроме начальной вершины:

$$N_3 = N_3 \cup \left[\bigcup_i n_{i1} \right] \text{ при } d^-(n_{i1}) \neq 0 \ \forall i.$$

Шаг 4. Добавить в H_3 все дуги из H_1, кроме дуг, один конец которых находится в конечной вершине:

$$U_3 = \bigcup_j (n_{j1}, n_{i1}) \text{ при } \Gamma^+(n_{i1}) \neq n_{k1} \ \forall n_{i1}.$$

Шаг 5. Добавить в H_3 дуги, соответствующие дугам в H_1, которые заходят в n_{k1}:

$$U_3 = U_3 \cup \left[\bigcup_j (n_{i1}, n_{j2}) \right] \text{ при } \Gamma^+(n_{i1}) = n_{k1} \ \forall n_{i1}, \ n_{j2} \in \Gamma^+(n_{i1}) \ \forall j.$$

Шаг 6. Добавить в H_3 дуги из H_2, не берущие начало в n_{02}:

$$U_3 = U_3 \cup \left[\bigcup_j (n_{i2}, n_{j2}) \right] \quad \Gamma^-(n_{i2}) \neq n_{02} \ \forall n_{i2}.$$

Шаг 7. Конец.

Введенная операция суммирования двух блок-схем соответствует блок-схеме, полученной из соединенных текстов программ, соответствующих слагаемым.

Положение 4. *Цикломатический коэффициент сложности* λ_3 *блок-схемы* H_3, *полученной в результате суммирования двух блок-схем* H_1 *и* H_2, *равен сумме цикломатических коэффициентов сложности* λ_1 и λ_2 *блок-схем* H_1 *и* H_2.

Введенная оценка сложности логической структуры ПО через цикломатический коэффициент сложности является аддитивной. Из свойства аддитивности следует монотонность.

Для оценки связанности используется структурно-графическое представление ПО в виде графа потока данных (ГПД).

Определение 24. Коэффициентом связанности программы, представленной в виде ГПД $R = (N, P, U)$, называется величина β, равная числу дуг в этом графе: $\beta = |U|$.

Положение 5. *Коэффициент связанности* β_3 *для ГПД, построенного из суммы двух программ, больше или равен сумме связанности каждой программы* β_1 *и* β_2.

Таким образом, согласно критерию связанности введенный коэффициент связанности применим для оценки качества ПО.

В настоящее время ни одно применяемое или разрабатываемое вычислительное средство не рассматривается без учета возможности использования новых резервов распараллеливания работ. Успешное применение МВС, в том числе распределенных СРВ, невозможно без создания соответствующего ПО (ArcGIS: Complete Integrated System. [2010–2010]. http:// www.esri.com/ software/arcgis/index.html), поэтому актуальной задачей является определение возможности распараллеливания ПО.

Для оценки распараллеливаемости программы предлагается использовать структурно-графическое представление ПО в виде ГПД, причем ГПД представляется специальным образом. Для этого множество структурных операторов O разбивается на подмножества Ω_i.

Определение 25. Подмножества Ω_i множества O называются ярусами, причем множество операторов Ω_1 первого яруса тождественно множеству информационно независимых операторов. Множество операторов Ω_2 второго яруса тождественно множеству операторов, информационно зависящих хотя бы от одного оператора первого яруса. Множество операторов Ω i-го яруса тождественно множеству операторов, информационно зависимых хотя бы от одного оператора (i-1)-го яруса. Также для любого яруса выполняются условия $\Omega_i \neq \varnothing$ $\Omega_i \cap \Omega_j = \varnothing$ $(i \neq j)$ $\forall i$.

Определение 26. ГПД, соответствующий упорядочиванию функциональных операторов по ярусам, отвечающим подмножествам $\Omega_1, \Omega_2, \ldots, \Omega_i, \ldots$ множества O, называется ГПД в ярусно-параллельной форме (ЯПФ).

Для получения ГПД в ЯПФ предложен следующий алгоритм.

Алгоритм 5 (алгоритм перестроения ГПД в ЯПФ).

Шаг 1. Упорядочить множество переходов N, так чтобы каждый переход занимал отдельный ярус от 0 до N и порядок переходов не противоречил порядку выполнения соответствующих операторов. Установить i=1.

Шаг 2. Пронумеровать все переходы в соответствии с занимаемыми ярусами.

Шаг 3. Если для перехода n_i выполняется условие $\Gamma^+(n_i) \cap \Gamma^-(n_j) \neq \varnothing$ для любого n_j, принадлежащего предшествующему ярусу, то перейти к шагу 9.

Шаг 4. Если для перехода n_i выполняется условие $\Gamma^+(n_i) \cap \Gamma^+(n_j) \neq \varnothing$ для любого n_j, принадлежащего предшествующему ярусу, то перейти к шагу 9.

Шаг 5. Если для перехода n_i выполняется условие $\Gamma^-(n_i) \cap \Gamma^+(n_j) \neq \varnothing$ для любого n_j, принадлежащего предыдущему ярусу, то перейти к шагу 9.

Шаг 6. Исключить переход n_i с яруса $\Omega_k = \Omega_k \backslash n_i$, которому он принадлежал. Включить переход n_i на предыдущий ярус $\Omega_{k-1} = \Omega_{k-1} \cup n_i$.

Шаг 7. Если ярус, на котором находился переход n_i, пуст ($\Omega_k = \varnothing$), то $\forall j$ $j = i+1,\ldots|N|$ выполнить операцию переноса перехода на предыдущий уровень $\Omega_{k+(j-i)} = \Omega_{k+(j-i)} \backslash n_j$; $\Omega_{k+(j-i)+1} = \Omega_{k+(j-i)+1} \cup n_s$.

Шаг 8. Перейти к шагу 3.

Шаг 9. Увеличить i ($i = i+1$). Если $i > |N|$, то перейти к шагу 10, иначе − к шагу 3.

Шаг 10. Конец.

Список литературы

1. САРКИСЯН А. А. Повышение качества программ на основе автоматизированных методов. М.: Радио и связь, 1991. 160 с.

2. КОВАЛЕНКО Д. А. Методы и средства автоматического синтеза параллельных программ на базе теории структурных функциональных моделей // Изв. Том. политехн. ун-та. 2008. Т. 312, № 5. С. 39–44.

3. ТАРНАВСКИЙ Г. А., Корнеев В. Д., Вайнер Д. А. и др. Вычислительная система "Поток 3": опыт параллелизации программного комплекса. 1. Идеология распараллеливания // Вычисл. методы и программирование. 2003. Т. 4. С. 33–44.

4. Хожайнова С. А. Многоуровневое представление программ и его использование в автоматическом распараллеливании // Мат. моделирование. 1997. Т. 9. № 2. С. 31–33.

5. Штейнберг Б. Открытая распараллеливающая система // Открытые системы. [Электрон. ресурс]. 2007. № 9. http://www.osp.ru/os/2007/09/4567122/.

Демин Антон Юрьевич – канд. техн. наук, доц. Института кибернетики Томского политехнического университета; тел.: (382-2) 42-63-34; e-mail: ad@tpu.ru;

Рейзлин Валерий Израилевич – канд. физ.-мат. наук, доц. Института кибернетики Томского политехнического университета; тел.: (382-2) 42-63-34; e-mail: vir@tpu.ru.

Дата поступления – 02.11.11