УДК 681.142.2

А. А. Гедич, А. Г. Зыков, А. В. Лаздин, В. И. Поляков

ПОИСК ПРОЦЕДУР ПО ГРАФУ ПЕРЕХОДОВ ФУНКЦИОНАЛЬНОЙ ПРОГРАММЫ ПРИ ВЕРИФИКАЦИИ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Решена задача поиска процедур в программном продукте при верификации вычислительного процесса, реализованного в виде программы. Предложен алгоритм поиска начальных адресов процедур и их распространения в адресном пространстве программы.

Ключевые слова: *верификация, граф переходов, функциональная программа, процедура.*

Введение. Экспоненциальный рост сложности вычислительных систем и программного обеспечения требует повышенного внимания к верификации их как на этапах проектирования и разработки, так и при эксплуатации. Анализ вычислительных процессов, реализуемых в виде программ, может проводиться как по исполнимому коду, так и по графоаналитическим моделям. В настоящей работе рассматривается задача поиска процедур по графу переходов функциональной программы, восстановленному из исполнимого кода программы.

При решении задачи верификации или оценки структурной сложности программы возникает необходимость ее представления в виде графа переходов (ГП) [1]. В настоящей работе под программами понимаются функциональные программы (ФП), алгоритм формирования графа переходов для которых приведен в работе [2]. Процедура — часть кода ФП, предназначенного для многократного использования. Структура процедуры во многом совпадает со структурой программы: наличие линейных участков, ветвлений, циклов, вызовов других процедур. Следовательно, выделение процедур в отдельные функциональные блоки, которые могут быть представлены единственной вершиной графа переходов ФП, существенно упрощает анализ структуры программы. Тождественность структуры программ и процедур позволяет использовать методы оценки структурной сложности программ применительно к процедурам.

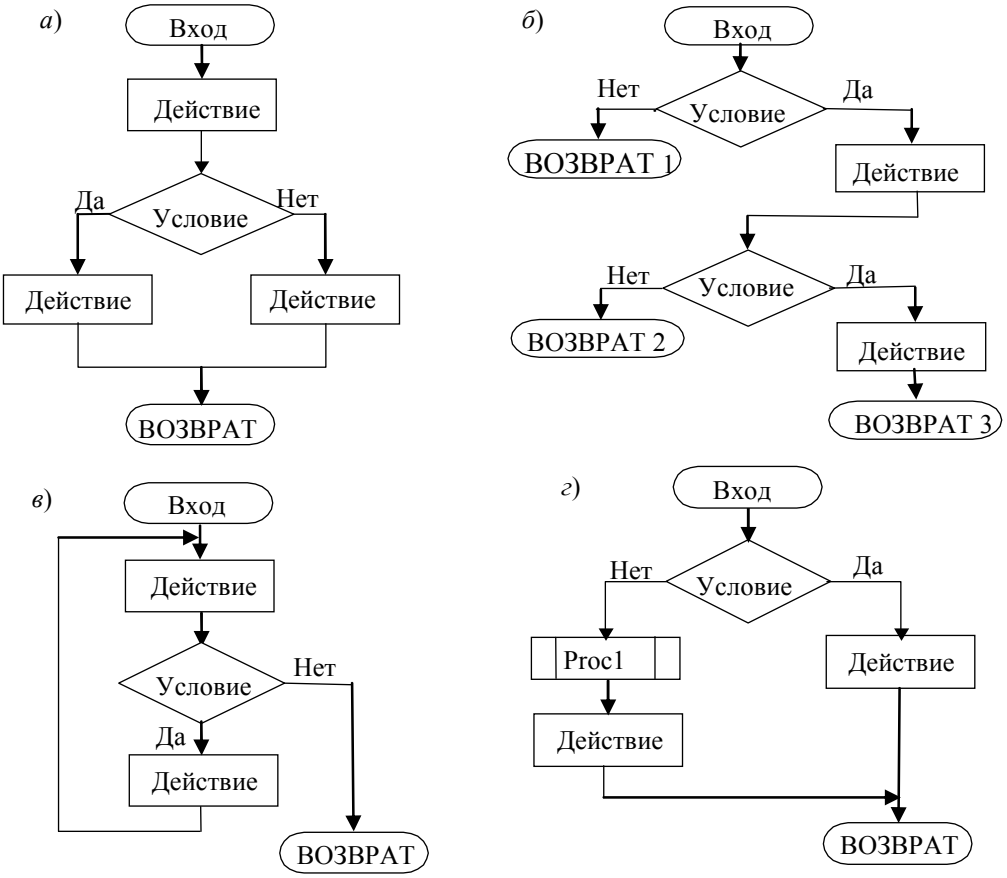
Анализ и постановка задачи. В общем случае структура процедуры заранее не известна, она зависит от назначения последней. Обращения к процедуре и возвраты из нее могут быть осуществлены посредством команд условного или безусловного переходов. При написании программ на языке Ассемблера допустимы переходы из тела процедуры к произвольным командам в теле программы (в том числе и к другим процедурам) и обращения к процедурам путем перехода (условного или безусловного) к любой помеченной команде в теле процедуры или выполнения команд вызова, так как имя процедуры рассматривается как метка, т.е. можно говорить о множестве точек входа в процедуру. Вышеперечисленное существенно затрудняет поиск произвольно оформленных процедур. Произвольное оформление процедур ухудшает структурированность программы и затрудняет выделение части кода в независимые блоки.

В отличие от Ассемблера большинство современных языков программирования (Си, Паскаль, Фортран) существенно ограничивают способы передачи управления к процедурам и выходам из них. Например, в языке Си имена меток локализуются в теле функции и к ним не допускаются обращения из других частей программы. Тем не менее, на практике структура процедур часто оптимизируется компиляторами, что ведет к появлению участков кода, называемых сопрограммами, которые невозможно выделить в независимые блоки.

Необходимо заметить, что тело процедуры может содержать косвенные безусловные переходы, использующие таблицы адресов и в некоторых случаях — вспомогательные таблицы индексов. Так называемая switch-конструкция используется не только во многих языках высокого уровня, но и в программах на Ассемблере, она затрудняет локализацию последней команды процедуры.

В настоящей работе под процедурой будет пониматься группа команд, предназначенная для решения локальной задачи, имеющая единственную точку входа, передача управления которой может осуществляться только посредством выполнения команд вызова; выход из процедур осуществляется выполнением команд возврата. Другие способы входа и выхода из процедур запрещены. Наложенные ограничения позволяют существенно упростить поиск процедур, оформленных по правилам, принятым в языках высокого уровня, или с использованием директивы PROC в языке Ассемблера. Для процедур выполняется правило: ближнему вызову соответствует ближний возврат, а дальнему — дальний. Команда возврата не обязательно должна быть расположена последней в последовательности машинных инструкций процедуры, последней также может быть команда передачи управления инструкции в теле процедуры или команда вызова другой процедуры, если вызываемая процедура является источником исключения или использует функции среды выполнения, завершающие работу программы. В таком случае компилятор оптимизирует эпилог процедуры. Если в коде программы находятся процедуры, не соответствующие наложенным ограничениям, они рассматриваются как часть головной программы. Для процедур, вход в которые был выполнен командой вызова, недопустимы переходы из тела процедуры.

Кроме того, необходимо учесть, что в теле процедуры может выполняться обращение к другим процедурам. Возможные структуры процедур приведены на рисунке (*а* — единственная команда возврата; *б* — несколько команд возврата; *в* — команда возврата не является последней инструкцией процедуры; *г* — вложенный вызов процедуры).



Для определения адресов перехода, используемых конструкцией switch, необходимо найти границы таблицы, которые чаще всего встраиваются в конец процедуры или, в некоторых случаях, в тело процедуры. Для доступа к таблице адресов переходов (ТАП) используется ассемблерная инструкция вида JMP CS:[reg*ptr_size + table_label], где reg — регистр, содержащий индекс элемента, ptr_size — размер указателя (4 на 32-битной архитектуре), table_label — абсолютный адрес таблицы. Ближней границей (БГ) ТАП называется абсолютный адрес, близкий к table_lable, дальней (ДГ) ТАП называется граница, противоположная БГ.

Стоит заметить, что в целях оптимизации компилятором может быть отрицательный индекс, т.е. ТАП будет увеличиваться в сторону меньших адресов. Таким образом, таблица может иметь два направления роста, задаваемых знаком индекса. Часто используется конструкция *N*-based switch, где *N* — число начальных индексов, запрещенных для использования. Это делает невозможным применение table_label в качестве начального адреса ТАП. В редких случаях компилятором может проводиться оптимизация, при которой table_label указывает на середину ТАП. Этот случай в работе не рассматривается.

Отметим, что, даже точно определив начальный адрес ТАП, можно точно определить конечный. Логично предположить, что конечным будет адрес последнего элемента таблицы, являющийся валидным. Однако считанное значение может быть валидным адресом, являясь на самом деле инструкцией. Не совсем верно предположение, что считываемый адрес указывает на тело анализируемой процедуры, чаще всего он указывает на часть процедуры, которая еще не получена, что делает невозможной данную проверку валидности адреса. Если адрес указывает на ту же секцию кода, в которой находится анализируемая процедура, то часто получаются неверные результаты.

В ТАП приводится неупорядоченная последовательность значений, возможно повторяющихся, представляющих адреса. Упорядочив значения адресов, можно получить зависимость, близкую к линейной.

Локализация границ ТАП производится следующим образом. Сначала считывается *N* значений в оба направления от table_label, где *N* — входной параметр поиска. Далее для полученных групп значений рассчитывается коэффициент корреляции Пирсона, отражающий линейность системы. Для расчета коэффициента требуются две координаты, в качестве второй вводится индекс элемента ТАП. Выбирается группа значений с наибольшим коэффициентом, а вторая отбрасывается.

Таким образом, находится направление увеличения ТАП. Далее происходит последовательное удаление значений из группы со стороны ДГ до тех пор, пока удаление этого значения приводит к снижению коэффициента линейности системы. После того как ДГ определена, аналогичным образом производится нахождение БГ ТАП.

Алгоритмы поиска процедур. Поиск процедур осуществляется в два этапа. Сначала в списке узлов ГП производится поиск всех обращений к процедурам, не совпадающие адреса вызовов помещаются в список начальных адресов процедур. Далее для каждого начального адреса выполняется поиск адреса последней команды процедуры с целью ее последующей локализации в рамках ГП ФП.

Алгоритм 1. Формирование списка начальных адресов процедур

1. Установить счетчик найденных процедур (СЧНП) в нуль. Установить указатель вершин на первый элемент в списке вершин графа.

2. Считать тип текущего узла.

3. Если это не команда вызова, перейти к п. 6.

4. Если адрес обращения к процедуре совпадает с одним из элементов списка адресов процедур, перейти к п. 6.

5. Инкрементировать СЧНП. Добавить новый элемент в список начальных адресов процедур (НАП).

6. Для текущего элемента в списке НАП установить тип вызова (дальний или ближний).

7. Если обработанный узел последний, перейти к п. 10.

8. Переместить указатель узла на следующий элемент списка узлов графа.

9. Перейти к п. 2.

10. Конец.

После завершения формирования списка начальных адресов процедур осуществляется поиск адреса последней команды процедуры.

Алгоритм 2. Выделение тела процедур

1. Установить счетчик обработанных процедур в единицу.

2. В списке вершин ГП найти узел i такой, что $АУ_i > НАП_{K+1}$ и $АУ_{i-1} < НАП_{K+1}$ ($АУ_i$ — адрес i-го узла).

3. Если найденный узел не соответствует командам возврата или переходов, то перейти к п. 7.

4. Если типы вызова и возврата не совпадают, то выдать сообщение об ошибке и перейти к п. 9, иначе — запомнить адрес узла в качестве потенциального адреса конца процедуры.

5. Если узел соответствует команде перехода и адрес перехода меньше адреса текущего рассматриваемого узла и меньше $НАП_{K+1}$, то запомнить адрес узла в качестве потенциального адреса конца процедуры, если это переход вперед, то запомнить адрес (если в ходе рассмотрения встречается несколько команд перехода вперед, запоминается самый старший адрес перехода).

6. Если в списке вершин ГП нет узлов, ссылающихся на адрес, который был выбран в качестве потенциального адреса конца процедуры, то выбрать следующий узел ГП и перейти к п. 3; иначе — считать его адресом завершения процедуры.

7. Если адрес конца процедуры меньше найденного адреса перехода вперед, выдать сообщение об ошибке и перейти к п. 9.

8. Увеличить счетчик обработанных процедур на единицу. Если остались необработанные элементы в списке НАП, перейти к п. 2.

9. Конец.

Предложенные алгоритмы реализованы на языке С++.

Заключение. В работе проанализирована проблема поиска процедур в программном продукте при верификации вычислительного процесса, реализованного в виде программы. Предложен алгоритм поиска начальных адресов процедур и их распространения в адресном пространстве программы. В развитие работы [3] представлены результаты исследований по поиску локальных переменных и аргументов процедур. Результаты работы использованы при модернизации САПР верификации вычислительных процессов, разрабатываемой на кафедре информатики и прикладной математики НИУ ИТМО [4, 5].

Работа выполнена при финансовой поддержке РФФИ (грант 12-07-00376-а).

СПИСОК ЛИТЕРАТУРЫ

1. *Липаев В. В.* Обеспечение качества программных средств: Методы и стандарты. М.: СИНТЕГ, 2001. 380 с.

2. *Лаздин А. В., Немолочнов О. Ф.* Метод построения графа функциональной программы для решения задач верификации и тестирования // Научно-технический вестник СПБ ГИТМО (ТУ). 2002. Вып. 6. С. 118—122.

3. *Гедич А. А., Зыков А. Г., Лаздин А. В.* Автоматический поиск локальных переменных и аргументов процедуры в исполняемом коде программы при верификации вычислительных процессов // Научно-технический вестник информационных технологий, механики и оптики. 2013. № 5 (87). С. 117—122.

4. *Зыков А.Г., Безруков А.В., Немолочнов О.Ф., Поляков В.И., Андронов А.В.* Графоаналитические модели вычислительных процессов в САПР // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2011. № 4 (74). С. 116—120.

5. *Зыков А. Г., Немолочнов О. Ф., Поляков В. И., Безруков А. В., Македонский А. А.* Учебно-исследовательская САПР верификации вычислительных процессов // Тр. Конгресса по интеллектуальным системам и информационным технологиям „IS&IT'11“. М.: Физматлит, 2011. Т. 2. С. 109—112.

Сведения об авторах

Андрей Алексеевич Гедич — аспирант; Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, кафедра информатики и прикладной математики;
E-mail: muzhedgehog@list.ru

Анатолий Геннадьевич Зыков — канд. техн. наук; Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, кафедра информатики и прикладной математики; доцент;
E-mail: zykov_a_g@mail.ru

Артур Вячеславович Лаздин — канд. техн. наук; Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, кафедра информатики и прикладной математики; доцент;
E-mail: lazdin@yandex.ru

Владимир Иванович Поляков — канд. техн. наук, доцент; Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, кафедра вычислительной техники;
E-mail: v_i_polyakov@mail.ru

Рекомендована кафедрой вычислительной техники

Поступила в редакцию 23.12.13 г.