ВЕРИФИКАЦИЯ В ИССЛЕДОВАТЕЛЬСКИХ, УЧЕБНЫХ И ПРОМЫШЛЕННЫХ СИСТЕМАХ

О.Ф. Немолочнов, А.Г. Зыков, А.В. Лаздин, В.И. Поляков

Рассматривается метод верификации программ относительно друг друга на основе алгебро-топологического подхода с применением теории множеств и теории графов.

Постановка задачи

Верификация в самом общем виде предполагает установление соответствия между различными объектами. Сами объекты могут задаваться математическими моделями либо разного уровня представления, либо одного уровня [1]. В первом случае модель более высокого уровня представления является метамоделью по отношению к модели низшего уровня представления (например, алгоритм и программа). В этом случае приходится разрабатывать метамодель, построение которой представляет собой сложную и трудоемкую задачу. Затраты на построение такой модели могут многократно превосходить затраты на построение программы, что и является основным недостатком данного подхода. Во втором случае модели представляются на одном уровне, например, в виде программ, и отпадает задача построения метамоделей. Однако сами методы верификации усложняются, и возникает задача их разработки и исследования.

Не отвергая в принципе первый подход, остановимся на втором подходе как более конструктивном.

Рассмотрим задачу верификации программ в некоторых областях производственной деятельности. При разработке программного продукта возникает задача исследования его, так как разработка одного и того же программного продукта может быть предложена разным группам проектантов, например, на конкурсной основе. Кроме того, на одно и то же техническое задание на разработку (спецификацию программы) одна и та же проектная группа может представить несколько вариантов решений, например, с использованием различных машинных алгоритмов, с использованием различных языков программирования, операционных сред и вычислительных платформ. В этом случае для повышения объективности и качества принятия решений необходимо верифицировать представленные программы между собой.

После того, как программа разработана и исследована, она передается для эксплуатации. В ходе своего жизненного цикла программа может подвергаться различным модификациям: какие-то функции могут добавляться в нее, какие-то исключаться за ненадобностью. И в этом случае любые изменения необходимо верифицировать с целью сопровождения программы и учета так называемых побочных эффектов. Например, что-то в программе модернизировали, но одновременно и внесли ошибку, которая может проявиться самым неожиданным образом. Верификация позволяет зафиксировать эту ошибку и повысить качество и объективность при рассмотрении различных ситуаций, возникающих при эксплуатации программного продукта.

При обучении студентов (слушателей) различным дисциплинам программирования проектируются те или иные учебные программы. В этом случае также возникает задача верификации разработанных программ с эталонными. Здесь верификация позволяет повысить качество и объективность оценки знаний студентов (обучаемых).

Из изложенного следует, что задача верификации программ друг относительно друга существует и является актуальной.

Метод

Пусть имеются две программы: программа А (*PA*) и программа В (*PB*). Необходимо провести их верификацию между собой. Чтобы конкретизировать задачу и снять вопрос о языках программирования, будем рассматривать программы на уровне исполняемых кодов (ИК) [2, 3]. Обозначим через *XPA*, *XPB* и *ZPA*, *ZPB* входные и выходные параметры (переменные) программ *PA* и *PB* соответственно, а через *XA, XB* и *ZA, ZB* значение соответствующих параметров на наборах теста верификации (*TV*). Предполагается, что между входными и выходными параметрами программ установлены взаимно однозначные соответствия, например, либо порядком перечисления, либо списками соответствий. Тест верификации должен отвечать условиям полноты покрытия всех вершин и дуг графовой модели программы отрезками существенных (активных) путей [4].

По аналогии с [5] введем операции пересечения ($\bigcap$) и вычитания (#) для множеств $ZA = \{za\}$ и $ZB = \{zb\}$, которые являются алгебро-топологическими. Прежде чем формулировать правила выполнения операций, сделаем два допущения.

1. Будем полагать, что для элементов $za \in ZA$ и $zb \in ZB$ при выполнении условия $|za - zb| \le \Delta$ следует назначение $za \bigcap zb \neq$ zero и $za \# zb = zb \# za =$ zero, где Δ определяется экспертным путем на основе инструментальной и методической погрешностями вычислений, а zero обозначает пустое множество для элементов множеств *ZA* и *ZB*. Знак $\varnothing$ зарезервируем для самих множеств. В противном случае, если условие не выполняется, то $za \bigcap zb =$ zero и $za \# zb = za$ и $zb \# za = zb$. Это позволяет работать с размытыми множествами в области действительных величин

2. Если некоторый параметр *xpa* не задан, то его значение считается равным zero, и, если некоторый выходной параметр *zpa* не вычисляется, то его значение $za \in ZA$ также равно zero (аналогично и для множества *ZB*). Это допущение позволяет работать либо с незаданными входными параметрами, либо с невычисляемыми выходными параметрами программ на тех или иных сегментах теста *TV*.

Теперь с учетом правил, введенных для элементов множеств *ZA* и *ZB,* можно сформулировать правила вычисления пересечения и вычитания для самих множеств *ZA* и *ZB*.

Пересечение множеств *ZA* и *ZB* будем формировать по следующему правилу:

$$ZA \bigcap ZB \begin{cases} = \varnothing, & \text{если } \forall\ za,\ zb \in ZA, ZB \mid za \bigcap zb = zero, \\ \neq \varnothing, & \text{если } \exists\ za,\ zb \in ZA, ZB \mid za \bigcap zb \neq zero. \end{cases}$$

Вычитание множеств *ZA* и *ZB* будем формировать по правилу

$$ZA \# ZB = \begin{cases} \varnothing, & если\ \forall\ za,\ zb \in ZA, ZB \mid za \# zb = zero, \\ \bigcup za \mid za \# zb \neq zero, & если\ \exists\ za, zb \in ZA, ZB \mid za \# zb \neq zero. \end{cases}$$

Заметим, что $ZA \bigcap ZB = ZB \bigcap ZA$ и $ZA \# ZB \neq ZB \# ZA$.

При верификации программ *PA* и *PB*, имеющих одну и ту же спецификацию, на основе множеств *ZA* и *ZB* возможны пять ситуаций, которые последовательно рассмотрим.

1. Если для программ *PA* и *PB* на тесте *TV* зафиксировано, что $ZA \bigcap ZB = \varnothing$, то сравниваются разные программы. Это положение иллюстрируется рис. 1, где представлены фрагменты графов программ *PA* и *PB*.

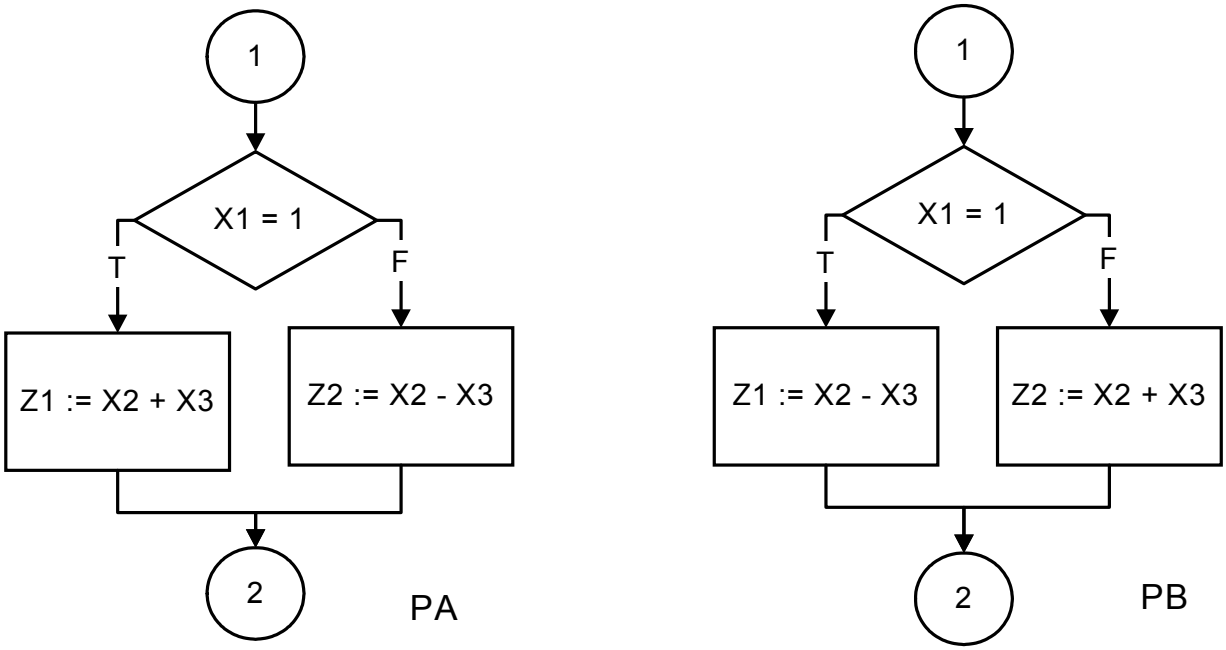

Рис. 1. Фрагменты разных программ

Заметим на этом примере, что сравнение программ необходимо осуществлять не на произвольных последовательностях, а на специально подобранных тестовых воздействиях. Если в данном примере положить $x_3 = 0$, то, какие бы значения ни принимал параметр x_2, результаты вычислений z_1 и z_2 будут совпадать для *PA* и *PB*. И вообще, значений параметров, равных нулю и единице, следует избегать, так как прибавление или вычитание нуля и умножение или деление на единицу никак не влияет на итоговый результат, а умножение на нуль просто уничтожает ранее вычисленные результаты. Возможны и другие специальные случаи, на которые следует обращать внимание при составлении тестовых наборов.

2. Пусть для программ *PA* и *PB* зафиксировано: $ZA \# ZB = \varnothing$ и $ZB \# ZA \neq \varnothing$, то есть программа *PB* вычисляет все то, что и программа *PA* и еще что-то дополнительное. Данное соотношение иллюстрируется на рис.2.

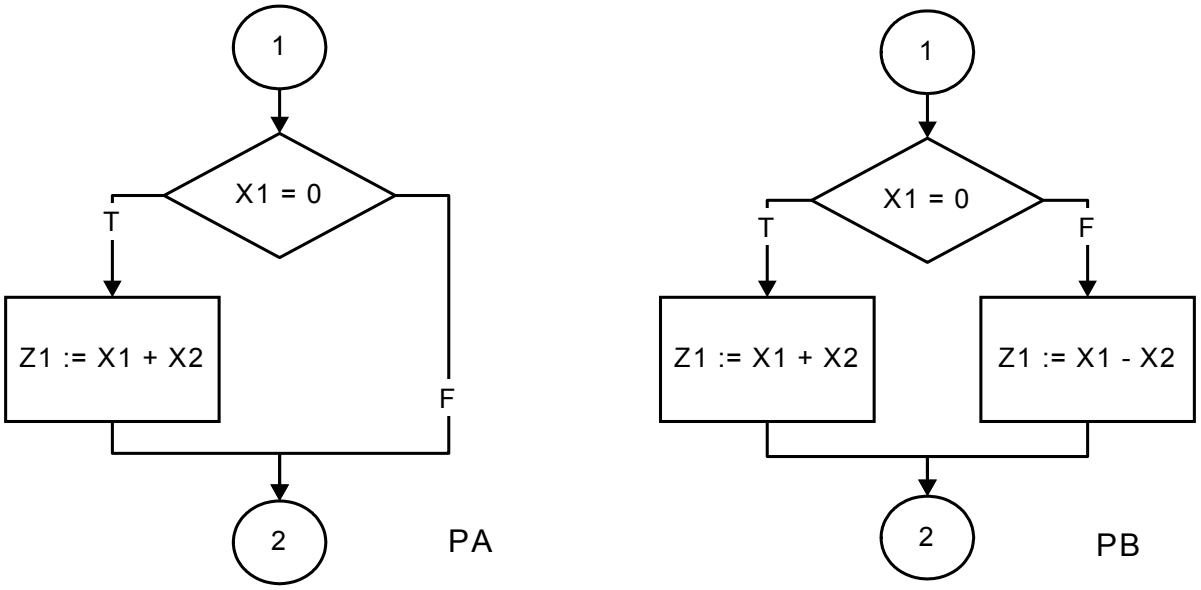

Рис. 2. Фрагмент программ по включению

В данном случае программа *PA* при невыполнении условия x_1 просто не вычисляет параметр z_1.

3. Данный случай является инверсным относительно второго варианта и при сравнении множеств *ZA* и *ZB* будет зафиксировано: $ZA \# ZB \neq \varnothing$ и $ZB \# ZA = \varnothing$.

Действительно, третий вариант получается, если на рис. 2 фрагменты программ *PA* и *PB* поменять местами.

4. Данный вариант характеризуется тем, что у программ *PA* и *PB* есть общая часть, в которой они функционируют одинаково, и есть участки программ, где они функционируют по-разному. Данное положение фиксируется следующим образом: $ZA \cap ZB \neq \varnothing$, $ZA \# ZB \neq \varnothing$ и $ZB \# ZA \neq \varnothing$.

Описываемый случай иллюстрируется рис. 3.

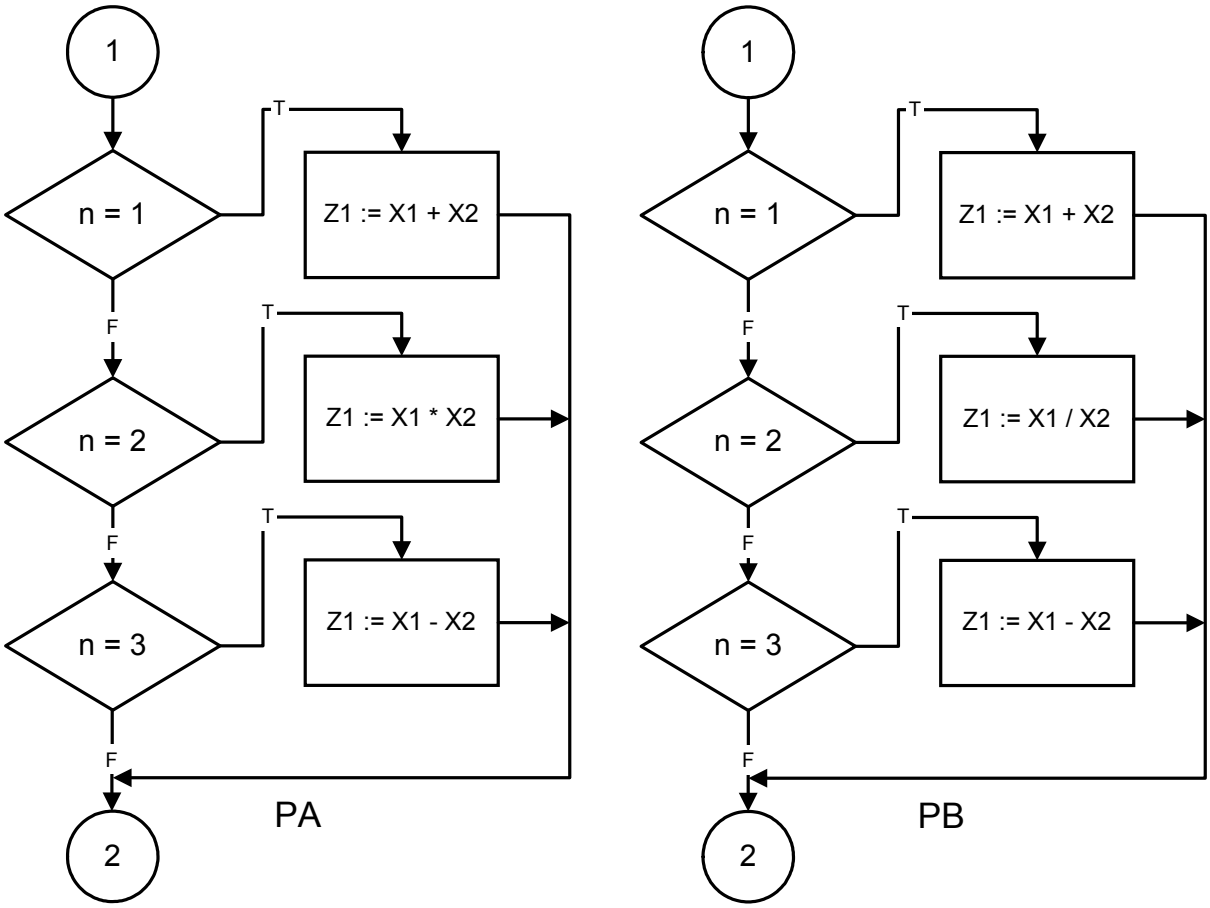

Рис. 3. Фрагменты программ с общими частями

Программы *PA* и *PB* при n=1 и n=3 функционируют одинаково, а при n=2 – различно, что и должно быть предметом исследования, и при необходимости данное расхождение должно быть устранено.

5. Данная ситуация характеризуется полной верификацией, для которой должны быть выполнены следующие условия: $ZA \# ZB = \varnothing$ и $ZB \# ZA = \varnothing$. Указанное соотношение иллюстрируется рис. 4. Из примера наглядно видно, что графы фрагментов программ различны, а вычисляют они параметры *zp* и *zm* одинаково. Это определяет необходимость анализа графов функциональных программ [2, 3], но недостаточно при верификации нескольких программ с одной спецификацией.

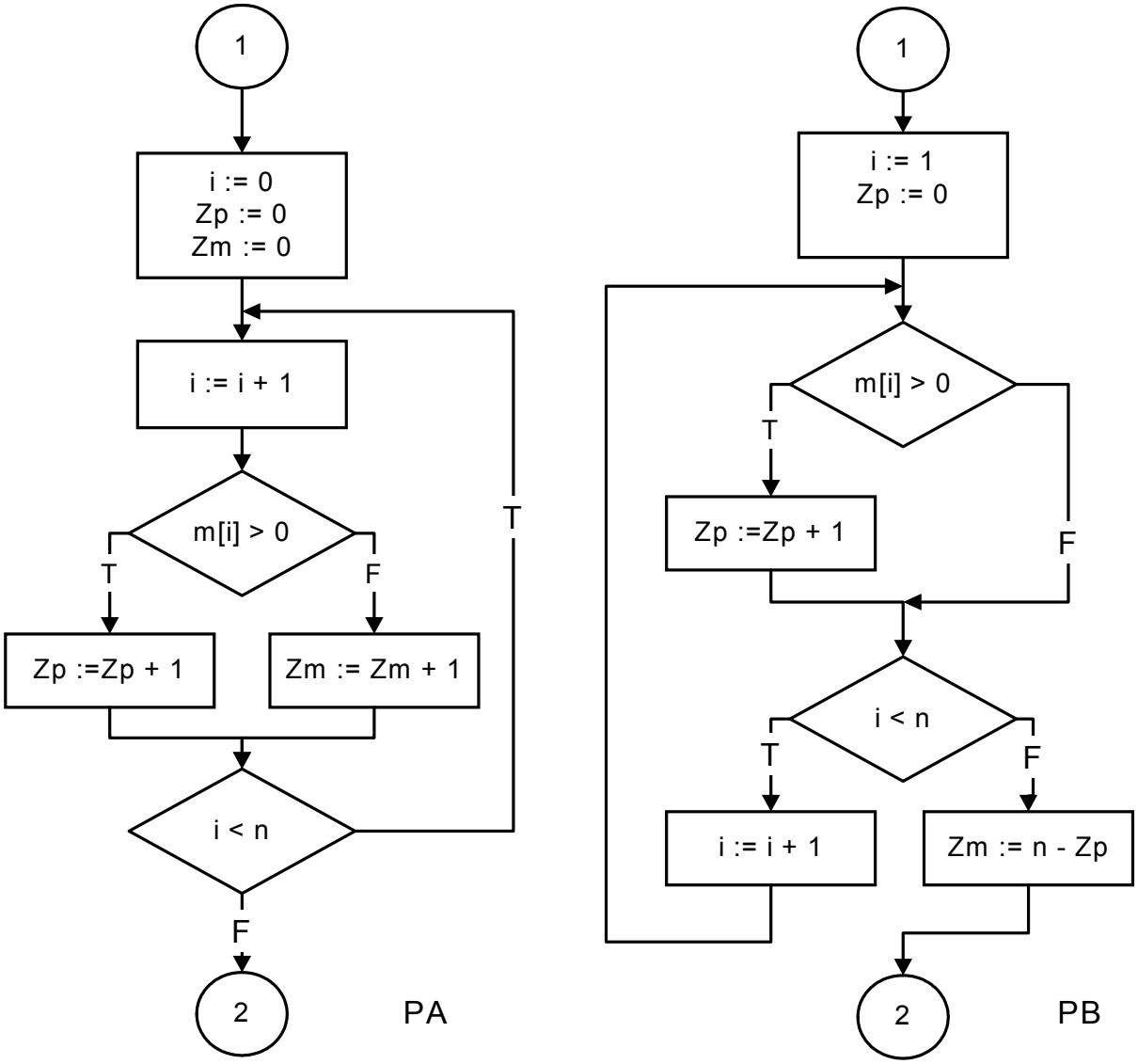

Рис. 4. Фрагменты программ, работающих идентично

Выводы

Предложенный метод позволяет проводить верификацию программного продукта на основе анализа потоков данных и открывает перспективные направления тестирования аппаратных средств вычислительных платформ опосредованно через тесты верификации программ и только той ее части, которая задействована в данных программах. Развитие средств верификации позволит также повысить качество защиты информации в виде программного продукта.

Литература

1. Липаев В.В. Обеспечение качества программных средств. Методы и стандарты / Издательство Синтер. Серия: Информационные технологии, 2001. 380 с.
2. Лаздин А.В., Немолочнов О.Ф. Оценка сложности графа функциональной программы // Научно-технический вестник СПбГИТМО (ТУ). Выпуск 6. Информационные, вычислительные и управляющие системы / Гл. ред. В.Н. Васильев. СПб: СПбГИТМО (ТУ), 2002.

3. Лаздин А.В., Немолочнов О.Ф. Метод построения графа функциональной программы для решения задач верификации и тестирования // Научно-технический вестник СПбГИТМО (ТУ). Выпуск 6. Информационные, вычислительные и управляющие системы / Гл. ред. В.Н. Васильев. СПб: СПб ГИТМО (ТУ), 2002.
4. Немолочнов О.Ф. Методы технической диагностики / Методическое пособие. Л: ЛИТМО, 1977.
5. Немолочнов О.Ф. Контроль и диагностика сочетаний неисправностей в комбинированных системах. // УС и М. 1973. №4.