• обобщенные диагностические признаки.

В режиме работы с архивом реализуются следующие возможности: просмотр реализаций, хранящихся в архиве, с отображением в окне, возвращение реализаций на дополнительную обработку в режиме разметки.

Разработанное сервисное программное обеспечение позволяет создавать и поддерживать базу данных для хранения регистрируемых сигналов и результатов их обработки. Пользовательский интерфейс работы с базой данных позволяет выполнять операции поиска требуемых реализаций сигналов по идентифицирующей информации, отображение их в графическом окне на мониторе и печатающем устройстве, отображение результатов обработки.

МЕТОДЫ АВТОМАТИЗАЦИИ ОТЛАДКИ В ТЕХНОЛОГИИ ГРАФОСИМВОЛИЧЕСКОГО ПРОГРАММИРОВАНИЯ

К.А. Кудрин, А.Н. Коварцев, С.А. Прохоров

В последнее время в значительной степени возросла сложность разрабатываемых программных продуктов. Одновременно с этим к ним предъявляются высокие требования относительно надежности и качества их функционирования.

Современные средства программирования должны основываться на методах безошибочного проектирования и разработки программ, и одним из основных способов достижения необходимого качества программного продукта по прежнему является отладка.

В настоящее время активно развиваются и совершенствуются методы автоматизации отладки и тестирования программ, которые позволяют в значительной мере облегчить данные процессы и существенно сократить количество потребляемых ими ресурсов.

На решение проблемы разработки высоконадежных программных продуктов ориентировано средство графосимволического программирования GRAF. Поставленная цель в системе GRAF достигается за счет частичной автоматизации процесса порождения кодов программы, более удобной графической формы представления программных продуктов, а также за счет развитой системы отладки порождаемых кодов.

Технология графосимволического программирования (ГС-технология) основана на представлении проектируемой программы в виде графа, где вершины - исполнимые модули, а дуги - предикаты, управляющие ходом выполнения алгоритма. Программирование в данной технологии представляет собой процесс соединения вершин дугами, используя ранее реализованные и отлаженные фрагменты программы в качестве модулей. Составным элементом программы в графосимволической технологии является либо так называемый базовый модуль, либо граф-программа, ранее написанная с использованием данной технологии.

В своей основе технология графосимволического программирования содержит принципы безошибочного проектирования программ. Это обусловлено следующими факторами.

1. Программа в системе GRAF строится из отлаженных и оттестированных модулей, расположенных в системных и пользовательских библиотеках. На этом этапе проектирования программы контроль за обнаружением и ликвидацией ошибок осуществляет подсистема отладки системы GRAF.

2. Другим фактором безошибочного проектирования программ является автоматизация процесса связывания данных предметной области и формальных параметров базовых модулей. Этот процесс при "ручном" программировании традиционно является источником большого числа труднообнаружимых ошибок. В системе GRAF автоматизированная связь по данным полностью исключает ошибки на данном этапе.

3. Для обнаружения более сложных ошибок, заложенных в самом алгоритме, а также для тестирования проектируемой программы применяется метод разбиения графа программы на маршруты и тестирование каждой цепочки модулей в отдельности.

Пусть область данных граф-программы (ОД) представлена в виде множества $\{ X_1, X_2,..., X_n \}$, где X_i - элемент произвольного типа, над которым производятся преобразования (чтение, запись, модификация);

$f_1(), f_2(),..., f_n()$ - функции, реализованные модулями ГС-технологии;

$f_j(X)$: $X \to X$, где $X \subset$ ОД - отображение подмножества X на себя.

Модуль предикат можно описать логической функцией:

$P_k (X_k)$: $X_k \dashrightarrow L$, где $L = \{ 0, 1 \}$.

Агрегированный объект представляет собой граф: где вершинами являются объекты f_1, f_2, f_3, а дугами - предикаты P_1, P_2, P_3.

Учитывая принцип детерминизма алгоритмов работу объекта можно описать явным функциональным выражением, в котором множество данных явно перевычисляется через остальные данные:

$X_{j2} = f_j(X_{j1})$, где X_{j1} - входные параметры для шага j, X_{j2} - вектор выходных параметров для шага j.

Наиболее оптимальным вариантом для тестирования объектов является перебор всех возможных вариантов значений входных параметров. Однако, этот способ не всегда приемлем и в большинстве случаев невозможен, причем достаточно найти хотя бы один допустимый вариант данных, при которых возникает ошибка, чтобы вынести решение о неправильной работе объекта. Поэтому для отладки объектов применяется метод случайного выбора значений. Этот метод при тестировании объектов позволяет существенно ускорить процесс, так как он осуществляет неполный перебор при наличии ошибок и почти полный перебор только при их отсутствии. Однако, полностью гарантировать надежную работу объекта можно только с определенной доверительной вероятностью.

В системе GRAF для отладки и тестирования объектов используется т.н. отладчик. Отладчик выполняет следующие основные функции:

- анализ области определения входных данных над которыми объект производит преобразование;
- генерация значений для входных данных объекта.

Во время выполнения модуля реализующего объект могут сложиться следующие ситуации

а) произошла т.н. "предвиденная ошибка". Не всегда область определения функции, реализованной программным модулем, можно представить граничными значениями параметров. В результате чего, множество значений, которые могут быть переданы в данный модуль, может не совпадать с областью определения функции, а это может привести к ошибкам внутри модуля. Сообщения об этих ошибках перехватываются системой и либо игнорируются, либо о них сообщается пользователю;

б) в процессе выполнения модуля произошла непредвиденная ошибка. Например, в алгоритме модуля было заложено неверное действие, которое не мо-

жет быть выполнено при любых значениях входных параметров. Это ошибка, которую необходимо исправлять и о ней сообщается пользователю;

в) по окончании выполнения алгоритма модуля выходные параметры не лежат в области значений. Это может быть следствием причин, рассмотренных в первом и втором случаях, поэтому выбор того или иного варианта осуществляется пользователем.

Тестирование по данному методу позволяет сделать статистическую оценку характеристик программного модуля, его производительность и эксплуатационную надежность

На следующем этапе проектирования программы алгоритмическая последовательность реализуется в виде направленного графа, вершины и дуги которого представляют собой оттестированные и отлаженные программные модули. Этот процесс графосимволического программирования происходит в удобной и наглядной графической форме, которая позволяет пользователю визуально отсеивать возможные алгоритмические ошибки. Однако всех проблем это не решает, и поэтому для обнаружения ошибок заложенных в самом алгоритме и для окончательной отладки программы применяется метод разбиения графового образа программы на ациклические цепочки.

Рассмотрим этот метод на приведенном выше примере. Элементы области данных граф-программы X_i на самом деле являются функциями времени $X_i = X_i(t)$, т.е. меняют свои значения в процессе работы программы. ЭВМ имеет дискретную природу, поэтому все моменты изменений значений данных можно пронумеровать. Для этого введем верхние индексы для данных: $X^{(j)}$, т.е. $X_k^{(i+1)}$ и $X_k^{(i)}$ значения одного параметра полученные на разных временных срезах.

При отладке программы требуется отладить все пути на графе G_1. Для рассматриваемого примера таких маршрутов два:

а) $f_1 \xrightarrow{P2} f_2 \xrightarrow{P3} f_3$;

б) $f_1 \xrightarrow{P1} f_2$.

Опишем формально реализацию пути а) на графе G_1.

$$
\begin{cases}
X_{12}^{(1)} = f_1(X_{11}^{(0)}) \\
P_2\,(X_{1P}^{(1)}) = 1 \\
P_1\,(X_{1P}^{(1)}) = 0 \\
X_{22}^{(2)} = f_2\,(X_{21}^{(1)}) \quad (1)
\end{cases}
$$

$$\begin{cases} P_3(X_{3P}^{(2)}) = 1 \\ X_{32}^{(3)} = f_3(X_{31}^{(2)}) \end{cases}$$

Система нелинейных уравнений (1) описывает условие работоспособности алгоритма на пути а). Любое ее решение соответствует одной из практически решаемых задач.

Для графа G_1 множество всех данных можно разделить на три группы данных: исходные, выходные, модифицируемые.

Для каждого маршрута (цепочки), учитывая классификацию данных можно построить множество исходных (модифицируемых) данных для всей цепочки. Данное множество представляет собой множество варьируемых параметров программы, значения которых должны быть обязательно заданы и от которых зависят результаты вычислений.

Пусть X_{var} - множество варьируемых параметров цепочки. Из этого множества, используя дополнительную информацию можно выделить подмножество $X_{исход}$ - исходные параметры. Каждое данное $x_k \in X_{исход}$ в GRAF имеет описание области значений (домен): $x_k \sim f_k(x)$, $dom\ x_k = \{ x_k \mid f_k(x) = 0 \}$

Используя эти описания можно построить множество значений множества $X_{исход}$. Тогда можно построить алгоритм, который бы случайным образом сканировал бы множество $dom\ x_k$ с целью определения его подмножества на котором цепочка имеет смысл.

Таким образом, подсистема отладки необходима на двух этапах: разработка и отладка исходных текстов базовых модулей и отладка модулей автоматически порожденных в процессе разработки программ. При использовании уже имеющихся отлаженных базовых модулей из системных или пользовательских библиотек отпадает необходимость в первом, наиболее сложном, этапе отладки (отладка низкого уровня). И дальнейший процесс сводится к контролю за состоянием модулей на разных этапах работы системы. Причем система GRAF позволяет автоматизировать основную часть этой работы, тем самым облегчая работу пользователя.