МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Томский государственный университет
Горно-Алтайский государственный университет
Институт оптики атмосферы им. В.Е. Зуева СО РАН

# НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ИССЛЕДОВАНИИ СЛОЖНЫХ СТРУКТУР

## МАТЕРИАЛЫ ДЕСЯТОЙ РОССИЙСКОЙ КОНФЕРЕНЦИИ С МЕЖДУНАРОДНЫМ УЧАСТИЕМ

Томск
Издательский Дом Томского государственного университета
2014

1. Если мы находимся в вершине, помеченной символом ∨, в существенное поддерево включаем одну из смежных с ней вершин, помеченных символом ∧ и ребро, соединяющее эти вершины.

2. Если вершина, в которой мы находимся, помечена символом ∧, включаем в существенное поддерево все смежные с ней вершины, помеченные символом ∨ или $D_p^{q_i}(X^j)$, и ребра, исходящие из нее и соединяющие смежные вершины; если все вершины строящегося поддерева помечены символами $D_p^{q_i}(X^j)$, то существенное поддерево построено, иначе продолжаем построение.

Объединение всех подмножеств кодовых слов, представляемых существенными поддеревьями, есть всё множество кодовых слов.

**Теорема 2.** Любое произвольное число $l$ можно представить суммой весов некоторых существенных поддеревьев дерева формулы $A$, представляющей ДНФ $\boldsymbol{D_n^m}$, где $\boldsymbol{n = 2m}$.

**Литература**


1. *Матросова А.Ю., Никитин К.В.* Синтез самотестируемого детектора (m,n)-кодов на программируемых логических блоках. // Вестник Томского государственного университета. Приложение. 2003. № 6. С. 124–136.
3. *Butorina N., Ostanin S.* Implementation by the Special Formula of an Arbitrary Subset of Code Words of (m, n)-code for Designing of a Self-Testing Checker // Proceeding of IEEE East-West Design&Test Symposium. IEEE, 2011. P. 255–258.
4. *Butorina N., Burkatovskaya J.* The properties of the essential subtrees of a graph representation of the (m,n)-codewords // Proceedings of the 7th International Forum on Strategic Technology (IFOST 2012). USA: IEEE Computer Society, 2012. P. 472–476.


# ИССЛЕДОВАНИЕ ВОЗМОЖНОСТИ ОБНАРУЖЕНИЯ НЕИСПРАВНОСТЕЙ ПЕРЕХОДОВ ПОСЛЕДОВАТЕЛЬНОСТНОЙ СХЕМЫ В ПРОЦЕССЕ ФУНКЦИОНИРОВАНИЯ


*О.И. Голубева*

Национальный исследовательский Томский государственный университет, Томск, Россия
golubeva-o@yandex.ru


Тестирование электронных систем в автономном режиме не всегда отвечает требованию надежности, тогда как полностью самопроверяемые системы, способные обнаруживать все неисправности в режиме нормального функционирования, как правило, требуют большой аппаратной избыточности. Поэтому получил распространение комбинированный подход к тестированию, при котором система снабжается возможностью проверять в процессе функционирования некоторый класс неисправностей [1–3] с незначительными аппаратными издержками.

В данной работе рассматривается задача обнаружения неисправностей в последовательностной схеме в режиме ее нормального функционирования для неисправностей ведущих к изменению переходов схемы.

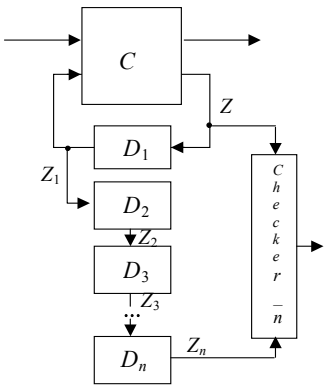



Рис. 1. Архитектура системы

Ранее [4] нами была предложена архитектура системы (рис. 1) для обнаружения неисправностей рассмотренного типа и метод сокращения размеров детектора (на рис. 1 – *Checker_n*) такой системы. Детектор реализует функцию $\delta^n(Z, Z_n)$, где $n \geq 1$, $Z$ и $Z_n$ – векторы, представляющие значения следующего состояния и состояния схемы $n$-1 тактов ранее. Предложенный подход к сокращению аппаратурной реализации детектора основывается на замене некоторых значений функции, реализуемой детектором на неопределенные значения. Замена возможна за счет того, что детектор обнаруживает неисправности не всегда при первом их проявлении.

Проведенные на контрольных примерах эксперименты показали, что в большинстве случаев предлагаемый подход позволяет добиться существенного упрощения задания на синтез детектора.

В данной работе подход расширен на последовательностные схемы, синтезированные на основе частичных автоматов. Было выявлено, что использование детекторов, реализующих функцию $\delta^n(Z, Z_n)$ при $n > 1$, в данном случае позволяет не только сократить размеры детектора, но и, в некоторых случаях, расширить число обнаруживаемых неисправностей.

**Литература**


1. *Eschermann B.* On combining off-line BIST and on-line control flow checking. Digest of Papers of Twenty-Second International Fault-Tolerant Computing Symposium 8–10 July 1992. pp. 298–305.
2. *Robinson S.H., Shen J.P.* Direct methods for synthesis of self-monitoring state machines. Digest of Papers of Twenty-Second International Fault-Tolerant Computing Symposium, 8–10 July 1992. pp. 306–315.
3. *Rochet R., Leveugle R., Saucier G.* Efficiency comparison of signature monitoring schemes for FSMs. Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95, 29 Aug. 1 Sep. 1995. pp. 705–710.
4. *Golubeva O.* On-Line Control-Flow Checking for FSMs // Вестник ТГУ. Приложение. 2007. № 23. С. 264–269.


# МУТАЦИОННОЕ ТЕСТИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ГЕНЕРАТОРА МУТАНТОВ μJAVA


*А.Д. Ермаков*

Национальный исследовательский Томский государственный университет, Томск, Россия
antonermak@inbox.ru


Актуальность тестирования программного обеспечения (ПО) в настоящий момент не вызывает сомнений. Очевидно, что качество тестирования в большой степени определяется качеством тестов, однако вопрос о том, какой тест называть «качественным» далеко не однозначный. Существует несколько критериев качества тестов при тестировании программ. Наиболее известным является критерий покрытия путей в программе, однако известно, что такие тесты не обнаруживают достаточно много функциональных ошибок. Кроме критериев покрытия качество тестов можно оценить на основе мутационного тестирования. Для этого необходимо создать множество модифицированных копий тестируемой программы (мутантов) и произвести тестирование на них. Поскольку известно, что наиболее сложно обнаруживаются одиночные ошибки, во многих программных комплексах для генерации мутантов такие мутанты и генерируются. Одним из наиболее эффективных и свободно распространяемых инструментов для мутационного тестирования Java программ является программный инструмент μJava, который имеет достаточно обширные функциональные возможности и, согласно документации, способен генерировать 34 различных типа мутаций программного кода. Среди них выделяют традиционные ошибки (замена математических операторов, логических операндов и т.д.) и ошибки объектно-ориентированного программирования (ошибки наследования, полиморфизма, и т.д.). Однако при использовании μJava для мутационного тестирования возникает ряд сложных ситуаций, которые не описаны в документации, и для анализа реальных возможностей использования этого инструмента автор провел исследования по мутационному тестированию программных реализаций алгоритмов сортировки.

Ниже представлены результаты мутационного тестирования для программы сортировки массивов методом «пузырька». Проверяющий тест был построен на основе анализа соответствующего алгоритма сортировки, и реализованная автором программа на этом тесте отработала согласно алгоритму. В результате работы инструмента μJava было сгенерировано 134 мутанта, каждый из которых был получен внесением только одной ошибки в программу-спецификацию. Результаты тестирования показали, что 4 мутанта не бы-