Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
(СПбГУ ИТМО)

УДК 004.4’242
№ госрегистрации
Инв. №

УТВЕРЖДАЮ
Руководитель работы
докт. техн. наук., профессор
А. А. Шалыто
31 мая 2007 г.

«___» _____ 2007 г.

ОТЧЕТ О ПАТЕНТНЫХ ИССЛЕДОВАНИЯХ
№ 2007.08.31-01 от 31 августа 2007 г.

по теме

РАЗРАБОТКА ТЕХНОЛОГИИ ВЕРИФИКАЦИИ УПРАВЛЯЮЩИХ ПРОГРАММ СО СЛОЖНЫМ ПОВЕДЕНИЕМ, ПОСТРОЕННЫХ НА ОСНОВЕ АВТОМАТНОГО ПОДХОДА

Этап 1. Выбор направления исследований и базовых методов

Шифр 2007-4-1.4-18-02-041.

Декан факультета «Информационных технологий и программирования»
докт. техн. наук, профессор _____ (подпись) В. Г. Парфенов

Начальник отдела интеллектуальной собственности и научно-технической информации _____ (подпись) Л. Н. Казар

Санкт-Петербург
2007

СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель темы — А. А. Шалыто
Заведующий кафедрой
«Технологии программирования»,
докт. техн. наук, профессор

Исполнитель отчета — А. С. Красс
Студент

СОДЕРЖАНИЕ

ОБЩИЕ ДАННЫЕ ОБ ОБЪЕКТЕ ИССЛЕДОВАНИЙ

Патентный поиск проводился с целью определения патентоспособности планируемых результатов научно-исследовательской работы **по лоту «Разработка технологий верификации программного обеспечения» – тема «Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода» (шифр «2007-4-1.4-18-02-041»)**, выполняемой в рамках Федеральной целевой научно-технической программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2012 годы» **по государственному контракту № 02.514.11.4048**, заключенному между Федеральным агентством по науке и инновациям и Государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на основании решения Конкурсной комиссии Роснауки (протокол от 28.04.2007 г. № 15), а также для получения сведений об охранных и иных документах, которые могут препятствовать применению результатов данной НИР в Российской Федерации, и условиях использования таких документов.

Патентный поиск проводился в соответствии с ГОСТ Р. 15.011-96 «Система разработки и постановки продукции на производство. Патентные исследования» [14]. При этом проводился поиск, как патентных источников, так и непатентных источников, а также программных продуктов. Это должно помочь в разработке технологии, в которой будут учтены преимущества и недостатки уже существующих подходов к верификации и соответствующих программных продуктов.

Существуют три метода анализа корректности программ:

- тестирование;
- доказательство корректности программ;
- верификация на модели (*model checking*).

Первый метод используется после или в процессе написания программы. Он не гарантирует, что все ошибки будут найдены.

Второй метод состоит в формальном доказательстве корректности программ, и является крайне трудоёмким. Он связан с семантикой используемого языка программирования, но в случае применения автоматного подхода к программированию после полного построения программы этот метод может быть использован непосредственно для проверки корректности функций, соответствующих входным переменным и выходным воздействиям. Эти функции обычно практически не содержат логики и реализуют небольшую задачу, корректность которой может быть доказана. Таким образом, структура автоматных программ благоприятствует частичному применению метода доказательства корректности программ. Однако этот метод оказывается практически бесполезным при проверке логики программ. Для проверки логики автоматных программ подходит третий метод – верификация на модели [1].

При использовании этого метода в общем случае по программе строится формальная модель с конечным числом состояний, а проверяемые свойства задаются, например, с помощью формул темпоральной логики. Проверка выполнимости темпоральных формул, описывающих свойства модели, происходит автоматическим образом. При этом для традиционно построенных программ обычно возникают проблемы с созданием модели, адекватной исходной программе. Однако при верификации автоматных программ этих проблем можно избежать, так как их поведение задается графами переходов, по которым можно автоматически построить модель Крипке, которую можно верифицировать известными методами [2]. В настоящей работе наибольшее внимание будет уделено методу верификации на модели, как наиболее перспективному.

В настоящее время существует большое число верификаторов, которые используют различные технологии [3–8], входные и выходные форматы и имеют свои достоинства и недостатки. Анализ

верификаторов позволит учесть полученную информацию в дальнейшем при выполнении работ по теме.

Патентный поиск (приложения А, Б, В) проводился с 01.06.2007 г. по 31.08.2007 г.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ, ТЕРМИНОВ

НИР — научно-исследовательская работа.
ВОИС – Всемирная организация интеллектуальной собственности.
EPO – Европейское патентное бюро (European Patent Office).
ИКА – иерархический конечный автомат.
BMC – ограниченная проверка моделей (Bounded Model Checking).

1. ИССЛЕДОВАНИЯ ПАТЕНТОВ

1.1. Выбор области для исследований

Продуктом, разрабатываемым в ходе данной научно-исследовательской работы, является технология верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода, предпосылки к разработке которой приведены во введении.

Темы, по которым проводился поиск, определялись исходя из состояния исследуемой области. Такими темами стали:

- проверка моделей;
- темпоральная логика;
- предотвращение ошибок с помощью тестирования или отладки программного продукта;
- компиляция или интерпретация языков программирования высокого уровня.

Соответственно этому определились и разделы международной патентной классификации, по которым был проведен поиск патентов.

Поиск патентной информации проводился в патентных базах данных Федеральной службы по интеллектуальной собственности, патентам и товарным знакам Российской Федерации (Роспатент, www.fips.ru), Бюро по патентам и товарным знакам США (USPTO, www.uspto.gov), Европейского патентного бюро (EPO, ep.espacenet.com) и Всемирной организации интеллектуальной собственности (ВОИС, www.wipo.int).

1.2. Особенности патентной системы США

В данном разделе приводится пример патента, выданного патентным бюро США корпорации *IBM*. Факт выдачи данного патента отражают общую ситуацию, сложившуюся в настоящее время в американской патентной системе и получающую все большее распространение в других странах мира, по крайней мере, в отношении патентования изобретений в области программного обеспечения и связанных с ней областях.

1.2.1. Патент на конечные автоматы

Патент США № 5,317,757. «Система и метод для работы конечного автомата с использованием векторов действий». Владелец: *International Business Machines Corporation* (корпорация *IBM*) [8].

Краткое описание патента. *Общий набор стандартных модулей, которые производят определенные действия в конечном автомате и являются строго модульными по своей структуре. Набор этих модулей состоит из модулей для задач, характерных для типа ресурса и модулей, которые не зависят от типа ресурса. Для каждого типа ресурса создается конечный автомат для управления шагами активации и деактивации ресурса. Каждый автомат уникально определяет новое состояние и произведение действия для каждого типа ресурса. Для привязки стандартных модулей действий к каждому конечному автомату для каждого типа ресурсов создаются вектора действия. Вектор действия устанавливает отношение определенного действия, выделенного конечным автоматом, к диспетчеризации одного или более стандартных модулей действий. Вектор действий может включать множество элементов. Каждый из этих элементов идентифицирует модуль действий, которому передается управление, и указатель вызова функции. Указатель на функцию идентифицирует определенную функцию, которую должен выполнить указанный модуль действий. Стандартные модули действий вызываются в порядке следования элементов в векторе действий.*

Патентное требование № 1. *Метод обработки данных для управления конечным автоматом в системе обработки данных, которая имеет память для хранения программных инструкций, входные устройства для получения входных данных, процессорный модуль для выполнения программных инструкций в ответ на указанные входные данные, а также выходные устройства для создания выходных данных в ответ на исполнение указанным процессорным модулем указанных программных инструкций. Включает следующие этапы:*

– хранение матрицы переходов первого конечного автомата в указанной памяти, которая имеет множество указателей на следующее состояние и указателей на вектора, к которым осуществляется доступ по значению текущего состояния в указанной памяти и значению входа;

– хранение таблицы векторов, состоящей из векторов действий, в указанной памяти, указанные указатели на вектора идентифицируют вектора действий в указанной таблице векторов, каждый из указанных векторов действий включает последовательность элементов векторов, имеющих указатели на модули и указатели на функции;

– хранение множества модулей действий в указанной памяти, каждый модуль действий включает в себя программные инструкции, организованные в множество сегментов функций, каждый из указанных указателей на модули идентифицирует один из модулей действий указанного множества и указанные указатели на функции идентифицируют одно из сегментов функций указанного множества;

– получение значения входа из указанного входного устройства и доступ к указанной матрице переходов первого конечного автомата с указанным значением входа и указанным значением текущего состояния для получения первого указателя на следующее состояние и первого указателя на вектор;

– доступ к первому вектору действий из указанной таблицы векторов и использованием указанного первого указателя на вектор, для получения первого указателя на модуль и первого указателя на функцию из первого элемента вектора в первой последовательности;

– исполнение программных инструкций в первом сегменте функций, на который указывает указанный первый указатель на функцию, в первом модуле действий, на который указывает указанный первый указатель на модуль, для произведения первого выходного действия указанным выходным устройством.

Замечания. Предлагается метод использования конечных автоматов для выполнения произвольных действий. Заявленные патентные требования настолько широки, что могут быть отнесены к использованию конечных автоматов практически для любых целей в программировании, сфере телекоммуникаций и других областях. Таким образом, использование конечных автоматов при написании программ, для реализации протокольных стеков, а также для иных целей может нарушать данный патент.

1.2.2. Последствия неправильной патентной политики

Недостаток приведенного выше патента, а также других патентов, выданных патентным бюро США, состоит в том, что патентуются идеи или вполне очевидные, или уже давно и широко применяемые, или же совершенно абстрактные

Многие исследователи и специалисты в соответствующих областях (программисты, системные инженеры и прочие) выражают свою озабоченность в связи со сложившейся ситуацией в сфере патентного права как в мире в целом, так и в США в частности [9, 10]. Предлагаемые пути выхода из сложившегося кризиса включают полный запрет на выдачу патентов в сфере программного обеспечения или радикальное сокращение сроков их действия. Следует напомнить, что выдача патентов на программное обеспечение, научные теории и методы интеллектуальной и хозяйственной деятельности запрещена патентными законами России [11] и стран Европы.

1.3. Патент США на верификацию моделей, постоенных на базе иерархических конечных автоматов

Патент США № 6,324,496 «Верификация проверкой моделей иерархических конечных автоматов». Владелец: *Lucent Technologies Inc* [10]. Авторы – R. Alur, M. Yannakakis.

Краткое описание патента. *Верификация проверкой моделей иерархических конечных автоматов (автоматов, имеющих как минимум одно состояние (супер-состояние), которое само является конченым автоматом) производится без предварительного развертывания иерархического конечного автомата (ИКА). В данном случае проверка моделей может включать проверку достижимости, наличия циклов, анализ требований с линейным и ветвящимся временем. Для анализа достижимости, в дополнение к наблюдению за тем, посещены ли состояния, алгоритм учитывает, посещены ли точки выхода для каждого супер-состояния. Для анализа требований линейного времени по формуле строится автомат Бюхи. Вычисляется произведение этого автомата и данного ИКА. Получается новый ИКА. Он проверяется на наличие циклов. Для анализа требований ветвящегося времени по исходной темпоральной формуле строится список подформул, расположенных в порядке увеличения размера.*

Замечания. В патенте под иерархическим автоматом понимается иерархическая структура *Крипке.* В нем описываются методы проверки достижимости, наличия циклов, анализа требований с линейным и ветвящимся временем, не требующие предварительной развёртки автомата.

1.4. Результаты исследования патентов

В результате предварительного исследования существующих патентов было решено исследовать раздел международной патентной классификации (МПК) «предотвращение ошибок с помощью тестирования или отладки программного продукта» *(G06F 11/36, "Preventing errors by testing or debugging of software").* Также были исследованы патенты разделов: *G06 F9/44*, *G06F 9/45*, *G06F 9/445*, *G06F 17/50*. Поиск дал отрицательный результат: в данных разделах МПК не было найдено патентов, которые мешали бы на данном этапе проводить разработку технологии.

Однако стоит заметить, что бюро по патентам и товарным знакам США в 2001 году выдало патент «Проверка модели иерархических автоматов» *("Model checking of hierarchical state machines", US6324496, current U.S. Class: 703/17, 703/22, 714/39, 716/4; current International Class: G06F 11/36 (20060101), G06F 017/50 ())***,** который может создать сложности при применении разрабатываемой технологии верификации на территории США. Этот патент описан в предыдущем разделе.

Сказанное, однако, не означает, что в других разделах МПК, которые не относятся напрямую к рассматриваемой области, не существует патентов, которые могут быть потенциально нарушенными в процессе эксплуатации разрабатываемой технологии. Пример патента *IBM* на конечные автоматы был приведен выше в разд. 1.2.1.

2. ИССЛЕДОВАНИЯ НЕПАТЕНТНЫХ ИСТОЧНИКОВ

2.1. Предпосылки к непатентным исследованиям

Согласно Патентному закону Российской Федерации [11], уровень техники, в сравнении с которым выявляется новизна изобретения, определяется не только зарегистрированными в России патентами, но также имеющейся во всем мире общедоступной информацией.

Применительно к рассматриваемой технологии такой информацией могут являться сведения о свободно (а также не свободно, что в данном случае это не имеет значения) распространяемых в сети Интернет программных продуктах, реализующих технологии, аналогичные разрабатываемой. Таким образом, в силу требования технического задания о получении свидетельств об официальной регистрации программ для ЭВМ, возникает необходимость исследовать аналоги создаваемой технологии и сравнить ее с этими аналогами.

Таким образом, данное исследование будет решать следующие задачи:

- анализ имеющихся в мире инструментов верификации программ;
- определение критериев, по которым можно сравнивать эти инструменты;
- отбор инструментов, подходящих для верификации управляющих программ со сложным поведением, которые построены на основе автоматного подхода, и сравнение этих инструментов.

2.2. Источники информации

В качестве основных источников информации по тематике «инструменты верификации» были выбраны энциклопедия свободного доступа *WikiPedia* [12] и база инструментов верификации *YAHODA* [13]. Указанная статья важна тем, что в ней приводится список инструментов (раздел «*Tools*»), которые могут быть использованы для верификации программ. Эти инструменты сильно отличаются видом входных данных, принципами работы, а также пользовательскими интерфейсами. Кроме основных были использованы и другие источники информации – различные информационные ресурсы, найденные при помощи поисковых систем. Информация, полученная из альтернативных источников, по большей части пересекалась с той, на которую имеются ссылки в энциклопедии *WikiPedia*. Это свидетельствует о том, что основные источники информации обладают достаточной полнотой.

2.3. Общий обзор источников

Полный список инструментов, которые найдены при поиске, в основном совпадает с приводимыми в статье [12] из энциклопедии *WikiPedia* или в базе инструментов верификации *YAHODA* [13] (на 31 августа 2007 года). Приведём этот список, состоящий из 94 инструментов:

- *EmbeddedValidator, The Simulink/Stateflow/Targetlink Verification Environment*
- *Prover Plug-In commercial proof engines*
- *0-In Formal Verification*
- *DiVinE (Distributed Verification Environment)*
- *PRISM (PRrobabilistIc Symbolic Model checker)*
- *RuleBase*
- *SATABS (predicate abstraction for C/C++ programs)*
- *SAL*
- *SLAM project*
- *SMV (Symbolic Model Checker)*
- *SPIN*
- *StEAM (State Exploring Assemblylevel Model Checker) Verification of concurent C++*

- *MRMC (Markov Reward Model Checker)*
- *Alloy language (лицензия GPL)*
- *Magellan*
- *APMC (Approximate Probabilistic Model Checker, лицензия GPL v2+)*
- *AcPeg (AcPeg Access Control Systems verification tool through Model Checking)*
- *Prover iLock*
- *BLAST (Berkeley Lazy Abstraction Software Verification Tool, MIT-style license)*
- *LoTREC*
- *Bogor*
- *BOOP Toolkit*
- *Cadena*
- *Cadence SMV*
- *CADP (Construction and Analysis of Distributed Processes)*
- *CBMC (A Bounded Model Checker for C/C++ programs)*
- *CHIC*
- *COSPAN*
- *Prevent*
- *GEAR (a game based model checking tool capable of CTL, modal μ-calculus and specification patterns)*
- *Helena (High Level Colored Petri Nets Analyzer, лицензия GPL)*
- *HOL theorem prover*
- *Java Pathfinder*
- *LASH (Liège Automata-based Symbolic Handler)*
- *KRONOS (TCTL-based timed model checker)*
- *LTSA (Labelled Transition System Analyser)*
- *[mc]square (a model checker for microcontroller (ATMEL ATmega and Infineon XC167) assembly code)*
- *MOPED*
- *MOPS (Modelchecking Programs for Security properties)*
- *μCRL (основан на ACP, лицензия GPL)*
- *mCRL2 Toolset (основан на ACP, лицензия GPL)*
- *NuSMV (A New Symbolic Model Checker)*
- *ORIS (uses a CTL-like temporal logic with real-time bounds, action and state based)*
- *ProB*
- *programs*
- *TLC*
- *UPPAAL (Uppaal Model Checker)*
- *VIS (Verification Interacting with Synthesis)*
- *dSPIN*
- *MAGIC*
- *ABC (Another Bisimulation Checker)*
- *ACL2 (A Computational Logic for Applicative Common Lisp)*
- *Atelier B*
- *Bandera*
- *CADiZ*
- *CWB – NC (The Concurrency Workbench of New Century)*
- *DBRover*
- *DREAM (Distributed Real-time Embedded Analysis Method)*
- *Edinburgh CWB (Edinburgh Concurrency Workbench)*
- *Expander2*
- *Fc2Tools (Fc2Tools and Autograph)*
- *FDR2*
- *HSolver*
- *HyTech*
- *IF*
- *INA (Integrated Net Analyzer)*
- *IOA Toolkit*
- *Isabelle*
- *LP (LP: The Larch Prover)*
- *MetaPRL*
- *Mocha*
- *Mucke*
- *MWB (The Mobility Workbench)*
- *PEP (Programming Environment based on Petri nets)*
- *PROD*
- *PVS (Prototype Verification System)*
- *Reactis Tester*
- *SGM (State-Graph Manipulators)*
- *SMCWWI (Simple Model Checker With Web Interface)*
- *STeP (Stanford Temporal Prover)*
- *Temporal Rover*
- *The Kit (The Model-Checking Kit)*
- *TIMES (A Tool for Modeling and*

- *ProofPower*
- *PROSPER*
- *Rabbit*
- *RAVEN (Real-Time Analysis and Verification Environment)*
- *TPS (TPS Theorem Proving System)*
- *TRON (Uppaal TRON)*
- *Truth*
- *TwoTowers*
- *VeriSoft*
- *VSE-II*
- *Ymer*
- *J-LO (The Java Logical Observer)*

Эти инструменты или разработки были достаточно подробно рассмотрены, и был произведён анализ их функциональности. По результатам анализа составлен краткий обзор разработок, включающий цитаты со страниц сайтов источников, переведенные на русский язык, и небольшое заключение, отражающее общее впечатление о каждой разработке. В таблице приведен общий результат предварительного анализа – список всех инструментов, близких к разрабатываемому по контракту, с указанием того, что представляет собой каждый из них. Жирным шрифтом выделены те верификаторы, которым уделено наибольшее внимание (после таблицы помещён их список).

Таблица. Краткое описание разработок

Название	**Назначение, краткое описание, заключение**
EmbeddedValidator, The Simulink/Stateflow/Targetlink Verification Environment	Верифицирует диаграммы последовательности состояний. *Ценность для исследования неизвестна.*
Prover Plug-In commercial proof engines	Продукт схож с *ProofPower*, но субъективно более удобен в использовании и обладает расширенными возможностями. *Может представлять ценность только на этапе написание собственного программного продукта.*
0-In Formal Verification	Программный продукт, предназначенный для верификации дизайна. Имеет графический интерфейс. *Ценность для исследования неизвестна.*
DiVinE (Distributed Verification Environment)	Программный продукт с открытым кодом, который позволяет производить проверку модели для языков *PROMELA* (была написана специальная программа-транслятор) и собственного входного языка *DiVinE* с помощью распределённых алгоритмов для требований корректности, заданных с помощью *LTL*. Проект начался в 2004 году. Последнее обновление сайта 16.05.2007, а последняя версия – 20.09.2006. *Изучение этого продукта может оказаться полезным, только если будет принято решение писать собственный программный продукт, использующий распределённые алгоритмы.*
PRISM (PRrobabilistIc Symbolic Model checker)	Производит вероятностную проверку модели кода программы, написанного на языках *PRISM*, основанном на состояниях, *PEPA (Performance Evaluation Process Algebra), Plain MC (.tra, .lab, .rew, .rewi text files)*. Поддерживается многопоточность. Использует

	вероятностные языки темпоральной логики *PCTL (Probabilistic Computation Tree Logic)* и *CSL (Continuous Stochastic Logic)*. Последняя версия – 15.11.2006. Существует серия статей, датированных 2007 годом. *Скорее всего, не несёт ценности для исследования, но сам по себе крайне любопытен.*
MRMC (Markov Reward Model Checker)	Обладает схожими функциональностью и методами работы с продуктом *PRISM*. Работает как с марковскими моделями непрерывного времени *(continuous-time)*, так и дискретного. Использует логики: *CSL (Continuous Stochastic Logic), CSRL (Continuous Stochastic Reward Logic), PCTL (Probabilistic Computation Tree Logic), PRCTL (Probabilistic Reward Computation Tree Logic)*. Последняя версия – 21.08.2007. *Скорее всего, не подходит по тематике исследования, но сам по себе крайне любопытен.*
Alloy language (GPL)	Проверяет модель, записанную на языке *Alloy*. Это язык моделирования, основанный на *FOL (first-order logic)*. На него оказала большое влияние *Z*-нотация. *Ценность неизвестна, но может быть изучен.*
Magellan	Программный продукт для поиска ошибок в дизайне схем. *Не несёт ценности для исследования.*
APMC (Approximate Probabilistic Model Checker, лицензия GPL v2+)	Позволяет «почти» распределённо производить проверку модели для полностью вероятностных систем. Этот продукт использует рандомизированный алгоритм с целью получения приближения вероятности того, что темпоральная формула верна. *APMC* использует распределённую модель вычислений. Это позволяет производить генерацию путей и верификацию формулы на рабочих станциях кластера. В последней версии поддерживается как дискретное, так и непрерывное время. Поддерживает логики *PCTL (Probabilistic Computation Tree Logic)* и *PLTL (Probabilistic LTL)*. *Вероятно, не несёт ценности для исследования.*
AcPeg (AcPeg Access Control Systems verification tool through Model Checking)	Система создана для верификации систем контроля доступа. *Скорее всего, не несёт ценности для исследования.*
Prover iLock	Эта система, генерирующая код и верифицирующая исходную модель. *Не представляет ценности.*
BLAST (Berkeley Lazy Abstraction Software Verification Tool, MIT-style license)	Позволяет производить верификацию программ, написанных на языке *C* (однако нельзя использовать рекурсивные функции и указатели на функции). По программе строится граф потока управления. Может проверять выполнимость только постоянных свойств *(safety properties)*. Строит контрпримеры. Существует плагин с

	графическим интерфейсом под *Eclipse™*. Требования корректности формулируются с помощью *Monitor automata*. *Скорее всего, не представляет ценности для исследования.*
LoTREC	Производит верификацию программы, записанной на высокоуровневом мета языке, причём сам процесс верификации записан на нём же. Продукт преобразует соответствующую программу в модель *Крипке* и для неё проверяет справедливость заданной темпоральной формулы. *Как уверяют разработчики, наиболее интересный момент этого проекта – метаязык. Остальное, скорее всего, не представляет интереса.*
Bogor	Производит проверку модели для собственного языка *BIR*. Поддержка большинства основных способов работы с многопоточностью. Язык *BIR* удобен для использования объектно-ориентированной парадигмы программирования. Имеет графическую оболочку (на основе *Eclipse™*). *Мощный инструмент, имеет много достоинств, его подробное изучение может оказаться крайне полезным.*
BOOP Toolkit	Проверяет достижимость заданных точек программы, написанной на языке *C* с помощью проверки извлечённой модели. *Может представлять интерес, но, скорее всего, подробное изучение нецелесообразно.*
Cadena	Программа ориентирована на моделирование и верификацию *CMM-систем (CORBA Component Model), EJB (Enterprise Java Beans)*. Графический интерфейс реализован как плагин для *Eclipse™*. *Возможно, стоит изучить.*
Cadence SMV	*Cadence SMV* может производить доказательства поверх символьного верификатора моделей (проверяет требования корректности, заданные на языках *LTL* и *CTL*. Входные языки: *Cadence SMV input language*, *SMV input language*, *Verilog*). Доказатель поддерживает несколько различных техник для сокращения больших или бесконечных задач верификации состояний к конечным или тем, которые могут быть проверены верификатором моделей. Имеет графический интерфейс. Последняя версия датирована 10.10.2001. *Интересный подход, но стоит обратить внимание на дату последней версии.*
CADP (Construction and Analysis of Distributed Processes)	*CADP* – набор инструментов для разработки протоколов. Он предлагает широкий спектр функциональных возможностей: от интерактивной симуляции до самых последних техник формальной верификации. Обеспечивает эффективную компиляцию, симуляцию, формальную верификацию и тестирование описаний, записанных на языке *LOTOS (Language of Temporal Ordering Specifications, ISO standard*

	8807). Набор инструментов, включает в себя *EVALUATOR* – программу для проведения верификации на лету, и *XTL (eXecutable Temporal Language)* – язык, реализующий различные темпоральные логики. Например, *HML*, *CTL*, *ACTL* и *LTAC*. Последняя версия продукта датирована 12.12.2006. *Определённо стоит изучения.*
CBMC (A Bounded Model Checker for C/C++ programs)	Этот продукт производит проверку модели для языков *ANSI-C* и *C++*. Он позволяет верифицировать выход за границы массивов, безопасность указателей, исключения и пользовательские утверждения *(assertions)*. Поддерживает наиболее важные элементы *ANSI-C*: многомерные массивы, арифметику указателей, дробную арифметику с фиксированной точкой и т.д. Есть плагин под *Eclipse*™. *Возможно, стоит подробнее изучить алгоритмы, используемые после получения модели.*
CHIC	Верифицирует с помощью проверки модели, что программное и аппаратное обеспечения соответствуют друг другу. *Не отвечает целям исследования.*
COSPAN	Крайне старая разработка. *Не представляет ценности.*
Prevent	Есть версии для языков *C/C++* и *Java*. Этот коммерческий проект берёт своё начало от продукта *«Stanford Checker» (Meta-level Compilation)*. Успешно использовался для поиска ошибок, но имеет мало общего с понятием верификация. *Не представляет интереса для исследования.*
GEAR (a game based model checking tool capable of CTL, modal μ-calculus and specification patterns)	Графическая среда, которая позволяет строить модели и проводить их верификацию, используя логику *CTL* и *μ*-вычисления. Присутствует большое количество визуального материала, предназначенного для обучения. Сотрудничает с проектом *«Specification patterns»*, который представляет собой организацию репозитория готовых требований корректности. *Крайне любопытный интерфейс. Возможно, стоит изучить его достоинства и недостатки. Мощная система задания требований корректности. Целесообразно отнестись крайне внимательно к проекту «Specification patterns».*
Helena (High Level Colored Petri Nets Analyzer, лицензия GPL)	Производит верификацию многопоточных программ, использует представление в виде цветных сетей Петри. Проверяет только свойства системы, которые должны быть выполнены во всех достижимых вершинах, например, отсутствие дедлоков. Нет своего сайта в Интернете. *Изучение по этой причине затруднено.*
HOL theorem prover	Программа аналогична *ProofPower*. HOL – Higher-Order

	Logic *Скорее всего, ценности не представляет.*
Java Pathfinder	Верифицирует исполняемый *Java* байт-код. Может находить пути выполнения, которые ведут к необработанным исключениям, дедлокам и т.п. Проверяет программы до 10 000 строк кода. Применяется в *NASA Ames Research Center*. Последняя версия датирована 21.11.2006. *Требует подробного изучения.*
LASH (Liège Automata-based Symbolic Handler)	Верифицирует конечные автоматы, извлечённые из исходных кодов программ, написанных на языке *C* с использованием специальных библиотек или на языке *PROMELA*. *Целесообразно ознакомиться.*
KRONOS (TCTL-based timed model checker)	Программный продукт разработан для верификации систем реального времени, компоненты которых должны быть промоделированы в виде временных автоматов *(timed automata)*. Требования корректности выражаются с помощью подмножества формул темпоральной логики реального времени *TCTL*. Последняя версия программы датирована 05.09.2002. *Изучение может оказаться полезным, если целью будет разработка продукта с аналогичными возможностями.*
LTSA (Labelled Transition System Analyser)	Продукт предназначен для верификации распределённых и параллельных систем *(concurrent systems)*. Они задаются с помощью *LTS (Labelled Transition System)* и *FSP (Finite State Processses)*. Требования корректности задаются в виде конечных автоматов (в более ранних версия поддерживались формулы *LTL*). Существуют интересные плагины, например, для верификации веб-сервисов, и графический интерфейс *(Eclipse™)*. *Может представлять интерес с точки зрения учёта достоинств и недостатков пользовательского интерфейса, а также изучения специализированных плагинов.*
[mc]square (a model checker for microcontroller (ATMEL ATmega and Infineon XC167) assembly code)	Предназначена для верификации моделей, извлечённых из ассемблерного кода для микроконтроллеров. Использует для проверки требований корректности, заданных *CTL*-формулами. *Скорее всего, не представляет ценности для исследования.*
MOPED	Верификатор моделей для *PDS (pushdown systems)*. Также может работать с булевыми программами и взаимодействовать с программным продуктом *SLAM*. *Ценность для исследования неизвестна.*
MOPS (Modelchecking Programs for Security properties)	Верификатор моделей, извлечённых из кода программ, написанных на языке *C*. Требования корректности задаются в специальном виде и соответствуют утверждениям, так называемого «защитного» программирования *(defensive*

	programming). Содержит готовую базу таких утверждений, планируется написание графического пользовательского интерфейса. *Стоит внимательно изучить соответствующую базу готовых утверждений.*
μCRL (основан на ACP, лицензия GPL)	Предназначен для верификации распределённых системами и протоколов. После получения пространства состояний использует проект *CADP*. *Принципы работы плохо понятны.*
mCRL2 Toolset (основан на ACP, лицензия GPL)	Набор инструментов для моделирования, валидации и верификации распределённых и параллельных систем, а также протоколов, написанных на языке *mCRL2*. Большое число инструментов с графическим интерфейсом. Существует версии под *Microsoft Windows, Linux, FreeBSD, Apple Mac OS X*. *Исходя из числа инструментов, можно сделать вывод, что проект целесообразно изучить.*
NuSMV (A New Symbolic Model Checker)	Это обновлённая версия программы *SMV* – символьного верификатора на моделях *(symbolic model checker)*. Он производит верификацию, комбинируя *BDD (binary decision diagrams)* и проверку моделей, основанную на *SAT (SAT-based model checking)*. Поддерживаемые способы задания требований корректности: *CTL, LTL.* Последняя версия – 22.05.2007. *Возможно, стоит ознакомиться.*.
ORIS (uses a CTL-like temporal logic with real-time bounds, action and state based)	Система для построения, моделирования, анализа и валидации временных сетей Петри. Производит также верификацию на основе проверки модели. Использует интересную визуальную интерпретацию темпоральных формул. *Следует изучить визуальную интерпретацию темпоральных формул.*
ProB	Система визуализирующая и проводящая проверку модели для *B*-метода. Суть метода заключается в генерировании кода из спецификации, написанной на *ANS (Abstract Machine Notation)*. Напоминает применение *Z*-нотации. Утверждается, что система используется в индустрии. *Стоит ознакомиться с теорией и, возможно, с реализацией.*
ProofPower	Предоставляет возможность доказать соответствие модели спецификации с помощью доказательства на языке *HOL* или *Z*-нотации. Также может верифицировать программы, написанные на языке *Ad*a, используя *Z*-нотацию. Последняя версия: 16.06.2007. *Может представлять ценность только на этапе написание собственного программного продукта, где будет возможно применение аналогичных технологий для непосредственного*

	доказательства корректности отдельных функций.
PROSPER	Информация недоступна.
Rabbit	Верификатор моделей систем реального времени. Использует *BDD (binary decision diagrams)*. Существуют реализации под *Microsoft Windows (9x, NT, 2000), Sun Solaris 2.6 (SunOS 5.6), Linux.* *Ценность неизвестна, скорее всего, не высока.*
RAVEN (Real-Time Analysis and Verification Environment)	Инструмент для анализа и верификации систем реального времени. Использует *CCTL (clocked computation tree logic)*. Можно задавать максимальную длину, просматриваемого пути. *Ценность, скорее всего, низка.*
RuleBase	Разработан *IBM Haifa Research Laboratory*. Последнее обновление в 2007 году. Использует подход к верификации схем, основанный на утверждениях (*assertions*). Разработчики полагают, что продукт может применяться в реальных проектах. *Не соответствует тематике исследования.*
SATABS (predicate abstraction for C/C++ programs)	Клон *CBMC*. *См. CBMC.*
SAL	Производит символьную верификацию. Использует *BDD*. Проверка модели основана на *SAT (boolean satisfiability problem)*. *Ценность неизвестна, возможно, целесообразнее исследовать аналогичные, но более известные продукты.*
SLAM project	Производит верификацию кода, написанного на языке *C*. Требования корректности задаются на специализированном языке *SLIC*. Продукт ориентирован на верификацию драйверов. Разработка *Microsoft Research*. *Возможно, целесообразно потратить время на изучение.*
SMV (Symbolic Model Checker)	Производит символьную верификацию систем, заданных в виде иерархических автоматов, где требования корректности задаются темпоральной логикой *CTL*. Утверждается, что системы можно задавать, как детализировано, так и абстрактно. Использует преобразование автоматов в модель Крипке. *По-видимому, целесообразно изучить.*
SPIN	Производит верификацию модели для языка *PROMELA*. Этот язык поддерживает только дискретные типы данных, а также функции работы с многопоточностью. Использует логику *LTL*. Утверждается, что он приспособлен для решения крупномасштабных задач. Производит верификацию на лету. *Мощный инструмент, имеет много достоинств. Его подробное изучение может оказаться полезным.*
StEAM (State Exploring Assemblylevel Model Checker)	Производит верификацию программ, написанных на языке *C++*, но уже преобразованных в машинное представление

	(используется виртуальная машина для языка *C++*). Может искать дедлоки, ошибки сегментации, переполнение переменных и бесконечные циклы. *Для данного проекта полезной информации практически не несёт.*
TLC	Верифицирует модели, описанные на языке *TLA+*. Язык предназначен для написания спецификаций реактивных, распределённых и параллельных систем. *Microsoft Research, 2003*. *Ценность неизвестна.*
UPPAAL (Uppaal Model Checker)	Среда для моделирования и верификации систем реального времени. Использует сети временных автоматов, расширенные типами данных (ограниченные целые, массивы и т.п.). Языки моделирования – *Timed Automata*. Поддерживаемые способы задания требований корректности: *TCTL* (подмножество). Последняя версия – март 2007 года. *Достаточно красивая среда, стоит посмотреть.*
VIS (Verification Interacting with Synthesis)	Производит верификацию модели, используя иерархические конечные автоматы, язык *Verilog*. Проверяемые утверждения задаются с помощью логики *CTL*. Разработка 2003 года. *Изучение может оказаться полезным.*
dSPIN	Расширение инструмента *SPIN*. Позволяет более эффективно верифицировать распределённые и параллельные системы. *dSPIN* расширяет возможности инструмента *SPIN*. Обеспечивает некоторое количество новых возможностей, реализованных сверх обычных алгоритмов исследования пространства состояний и сокращения состояний инструмента *SPIN*: указатели, динамическое выделение/освобождение памяти, рекурсивные функции, указатели на функцию, сборщик мусора, симметричное сокращение. Достаточно мощный входной язык (расширение языка *PROMELA*). *Может быть полезен для данного исследования, но, скорее всего, на поздних стадиях.*
MAGIC	Верифицирует модели, полученные из исходных кодов программ, написанных на языке *C*. Поддерживает распределённые и параллельные системы. Верификатор основан на подходе, использующем контрпримеры. Сначала из исходного кода извлекается модель, например, абстрагированием предикатов *(predicate abstraction)*. Далее модель проверяется на соответствие спецификации, которая задаётся конечным автоматом. Для верификации широкомасштабных систем существует возможность декомпозиции на небольшие фрагменты, которые можно затем верифицировать отдельно.

	Последняя публикация датирована 2004 годом. *Следует ограничиться рассмотрением аналогичных разработок.*
ABC (Another Bisimulation Checker)	*ABC* – это инструмент для определения эквивалентности между термами π-вычислений. Написан на языке программирования *OCaml*. Инструмент реализует проверку эквивалентности верификатора *MWB (Mobility Workbench)*. Строит контрпримеры. *Вероятно, не представляет ценности для исследования.*
ACL2 (A Computational Logic for Applicative Common Lisp)	*ACL2* – это и язык программирования, в котором можно моделировать компьютерные системы, и инструмент, позволяющий доказывать свойства этих моделей. *ACL2* поддерживает рекурсивно-определённые функции и автоматическое доказательство свойств таких функций. Сайт содержит большое количество сопутствующей литературы. Последняя версия инструмента датирована 03.12.2006. *Возможно стоит ограничиться аналогами.*
Atelier B	Инструмент производит проверку синтаксиса и типов. Содержит многопроходный и интерактивный доказатели, трансляторы языков *C*, *C++, Ada*, а также аниматор спецификации. Использует *B*-метод. Применялся компанией *Siemens* для разработки программного обеспечения *MENTOR*. Доступен только по коммерческой лицензии. Последняя версия – 01.01.2002. *Скорее всего, не представляет непосредственной ценности для исследования.*
Bandera	Инструмент ориентирован на извлечение моделей из исходных кодов, написанных на языке *Java*. Поддерживает возможность перевода естественных языков в логические утверждения. Последние данные о проекте заканчиваются в 2005 году. *Требуется изучить способ перевода естественных языков в логические утверждения. Ценность самого инструмента для исследования неизвестна. Скорее всего, низка.*
CADiZ	Инструмент, написанный для манипуляций со спецификациями, заданными на *ISO*-стандарте *Z*-нотации. Проверяет типы, производит доказательства. Доказатель имеет готовый набор утверждений, тактический язык и некоторые процедуры принятия решений, в частности, которые пытаются использовать инструмент *SMV* для задач, которые могут быть решены верификацией модели. Последняя версия датирована 17.06.2002. *Не соответствует цели исследования.*
CWB – NC (The Concurrency Workbench of New Century)	Система производит анализ достижимости, верификацию моделей для μ-вычислений и проверку эквивалентности.

	Входные языки: *CCS (Calculus of Communicating Systems), CSP (Communicating Sequential Processes), LOTOS (Language of Temporal Ordering Specifications), TCCS (Timed CCS)*. Поддерживаемые способы задания требований корректности: *AFMC (Alternation Free Modal mu-Calculus), CTL (Computation Tree Logic), GCTL* (Generalized CTL*)*. Последняя версия датирована 01.06.2000. *Скорее всего, не представляет ценности для исследования.*
DBRover	Монитор времени исполнения (*runtime monitor*) для темпоральных правил, написанных на языках *LTL* и *MTL*. Это автоматическая, удалённая и графическая версия инструмента *TemporalRover*. Содержит редактор темпоральных формул, генератор и компилятор кода, симулятор. Языки моделирования: *Ada, C, C++, Java,VHDL, Verilog*. Поддерживаемые языки задания требований корректности: LTL (Linear Time Logic), LTL with Temporal Da, MTL (Metric Temporal Logic). Доступен только по коммерческой лицензии. Последняя версия датирована 31.08.2002. *Целесообразно ознакомиться с редактором темпоральных формул.*
DREAM (Distributed Real-time Embedded Analysis Method)	Инструмент ориентирован на работу с встроенными распределёнными системами реального времени *(distributed real-time embedded, DRE)*. Разработан собственный метод формального анализа таких систем. Цель: автоматизировать процесс верификации, разработки, конфигурирования и интеграции *DRE*. Языки моделирования: *C++, Timed Automata*. Требования корректности формулируются с помощью *Monitor automata*. Последняя версия продукта датирована 06.07.2006. *Может обладать ценностью для исследования только при условии, что будет разрабатываться аналогичный продукт.*
Edinburgh CWB (Edinburgh Concurrency Workbench)	Входные языки: *CCS (Calculus of Communicating Systems), SCCS (Synchronous calculus of communicating systems), TCCS (Timed CCS)*. Способ задания требований корректности – *μ-исчисление.* *Ценность для исследования, скорее всего, низка.*
Expander2	*Expander2* – это гибкий многоцелевой инструмент для интерактивной отладки кода, доказательства теорем, нахождения ограничений, анализа графа потока управления и родственных процедур. Поддерживаемые логики: *AFMC (Alternation Free Modal mu-Calculus), CTL (Computation Tree Logic), Swinging Types (many-sorted predicate logic with Horn/co-Horn axioms)*. Последняя версия – 25.11.2002. Последняя публикация датирована 2007 годом.

	Скорее всего, бесполезен для исследования.
Fc2Tools (Fc2Tools and Autograph)	Последняя версия – 1996. *С большой вероятностью, бесполезен для исследования.*
FDR2	*FDR*2 – инструмент для проверки процессов, описанных на языке *CSP (Communicating Sequential Processes)*. Доступен как по коммерческой лицензии, так и в академических целях. Последняя версия – 23.07.2007. *Скорее всего, не стоит изучения.*
HSolver	*HSolver* – это программа для верификации гибридных систем, основанных на абстрактной детализации (*abstraction refinement)*. Язык моделирования – гибридные автоматы. *Ценность неизвестна.*
HyTech	*HyTech* – символьный верификатор для гибридных систем. Язык моделирования – гибридные автоматы. Поддерживаемые способы задания требований корректности: *CTL, Monitor automata*. *Скорее всего, бесполезен для исследования.*
IF	Входные языки: *IF (Intermediate Format), SDL (Specification and Description Language)*. Поддерживаемые способы задания требований корректности – *Monitor automata*. Последняя версия – 01.01.2005. *Скорее всего, бесполезен для исследования*
INA (Integrated Net Analyzer)	Использует сети Петри. Последняя версия – 23.03.2001 *Скорее всего, бесполезен для исследования.*
IOA Toolkit	Система для верификации доказательством утверждений. Языки моделирования: *IOA (I/O Automata), LSL (Larch Shared L).* Последняя версия – 01.01.2002. *Скорее всего, бесполезен для исследования.*
Isabelle	*Isabelle* – среда генерирующая доказательства, разработанная в *Cambridge University (Larry Paulson)* и *TU Munich (Tobias Nipkow)*. Могут быть определены новые формализмы. Поддерживает *HOL (Higher-Order Llogic)* и теорию *ZF*-множеств. Последняя версия – 30.09.2005. *Скорее всего, бесполезен для исследования.*
LP (LP: The Larch Prover)	Система для верификации доказательством утверждений. Языки моделирования: *IOA (I/O Automata), LSL (Larch Shared L)*. Последняя версия – 28.01.1999. *Бесполезна для исследования.*
MetaPRL	Система для верификации доказательством утверждений с широким классом поддерживаемых логик. Основная особенность этой системы состоит в том, что *MetaPRL* производит семантическое связывание с языком

	программирования. Таким образом, система выполняет функции логической среды программирования, в которой программы конструируются, реализуются и производится их верификация. Последняя публикация датирована 2004 годом. *Скорее всего, не представляет интереса для исследования. Только на поздних этапах может быть исследована соответствующая среда.*
Mocha	Главной задачей инструмента *Mocha* является разработка, а не нахождение ошибок структуры, дизайна в процессе формальной верификации систем, основанных на компонентной модели. Имеется графический интерфейс. *Вероятно, может оказаться интересен для исследования.*
Mucke	Символьный верификатор моделей для μ-вычислений. Входной язык похож на языке *C++*. Последняя версия – 26.03.2002. *Скорее всего, бесполезен для исследования.*
MWB (The Mobility Workbench)	*MWB* инструмент для манипулирования и анализа мобильных распределённых и параллельных систем, описанных с помощью π-вычислений. Последняя версия – 19.08.2006. *Ценность неизвестна.*
PEP (Programming Environment based on Petri nets)	Инструмент *PEP* – среда разработки и верификации для различных спецификаций и языков программирования. Все части инструмента доступны из графического пользовательского интерфейса. Верификация производится на автоматически сгенерированных сетях Петри. Пакет *PEP* легко расширяется и уже предлагает большое количество формальных моделей и техник верификации. Входные языки: *B(PN)^2 (Basic Programmin Notation for Petri Nets), PBC (Petri Box Calculus), PFA (Parallel Finite Automata), Petri Nets, SDL (Specification and Description Language)*. Поддерживаемые способы заданий требований корректности: *CTL, LTL*. Последняя версия датирована – 09.09.2004. *Ценность неизвестна, но инструмент, скорее всего, бесполезен для исследования.*
PROD	Инструмент реализует несколько различных методов анализа достижимости. Использует техники генерации сокращённого графа достижимости: метод «упрямых» множеств *(the stubborn set method)*, метод неподвижных множеств *(the sleep set method)*, определения приоритетов и использования симметрий. Инструмент поддерживает верификацию *CTL*-формул для графа достижимости и верификацию на лету для *LTL*-формул, используя метод «упрямых» множеств. Входной язык – *Petri Nets*.

	Инструмент использовался в индустрии. Последняя версия – 26.01.2006. *Соответствующие методы сокращения графа достижимости могут быть изучены. Ценность остальной части неизвестна.*
PVS (Prototype Verification System)	*PVS* – интерактивный доказатель, основанный на языке *HOL* с предикатными подтипами, зависимыми типами, рекурсивными типами данных, рекурсивными и индуктивными определениями, параметризованными теориями. Доказатель поддерживает пропозициональные доказательства, упрощение, верификацию моделей и т.д. Язык моделирования – *CSP (Communicating Sequential Processes)*. Последняя версия – 22.11.2006. *Ценность неизвестна, скорее всего, стоит ограничиться изучением аналогов.*
Reactis Tester	Моделирует и тестирует реактивные системы. Языки моделированием: *Simulink/Stateflow*. Доступен только по коммерческой лицензии. Последняя версия датирована – 27.11.2006. *Вероятно, может оказаться интересен для исследования.*
SGM (State-Graph Manipulators)	Это инструмент для высокоуровневой верификации систем реального времени. В последних версиях были произведены улучшения, позволяющие использовать продукт для верификации встроенных и *SoC (System-on-Chips)* систем. Имеет модульную архитектуру. Использует некоторое количество различных техник сокращения пространства состояний (симметричное сокращение и т.п.). Входные языки: *PFA (Parallel Finite Automata), Timed Automata*. Поддерживает *CTL*. Последняя версия датирована – 30.10.2002. *Соответствующая теория сокращения пространства состояний может быть изучена, но, скорее всего, не несёт никакой новизны относительно аналогичных современных разработок.*
SMCWWI (Simple Model Checker With Web Interface)	Цель этого средства – обучение. Последняя версия – 18.12.2002. *Для данного исследования бесполезен.*
STeP (Stanford Temporal Prover)	*STeP* – многосторонний верификатор. Использует алгоритмические и дедуктивные методы для верификации реактивных и гибридных систем, а также систем реального времени. Для сокращения доказательств предоставляет возможность работы с правилами верификации и верификационными диаграммами. В состав инструмента включён валидатор и упроститель формул, которые часто могут определить эти условия верификации автоматически. Интерактивный доказатель позволяет справиться с теми

	условиями, которые не могут быть доказаны автоматически. Инварианты могут быть сгенерированы автоматическим путём для поддержки дедуктивных и дедуктивно-алгоритмических методов. Входной язык – *SPL (Simple Programm (Simple Programming Language)*. Поддерживаемые способы задания требований корректности – *LTL*. Последняя версия датирована 1999 годом. *Ценности для исследования не представляет.*
Temporal Rover	Производит проверку требований, сформулированных на языке *LTL* с ограничениями реального времени *(with real-time constraints)*. Генерирует исполняемый код для спецификаций, записанных в виде комментариев. Поддерживает темпоральную валидацию, например, «после возникновения события *A*, значение переменной *x* в пяти процентах случаев не изменяется, и её среднее значение больше 100 в течение одного часа или до того момента, когда событие *B* произойдёт дважды». Входные языки: *Ada, C, C++, Java, VHDL, Verilog*. Поддерживаемые способы задания требований корректности: *LTL, LTL with Temporal Da, MTL (Metric Temporal Logic)*. Существует также проект *StateRover*, который, по-видимому, основывается на этом инструменте (оба проекта размещены на одном сайте), но является более новым и мощным. *StateRover* имеет более широкие возможности по сравнению с *Temporal Rover* и оперирует с подмножеством языка *UML*. Может генерировать код на языках *Java, C++, C*. Имеет мощную графическую оболочку, реализованную как плагин для *Eclipse™*. Доступен только по коммерческой лицензии. *Целесообразно изучить.*
The Kit (The Model-Checking Kit)	Основная особенность этого верификатора моделей – возможность верифицировать модель, написанную на одном языке моделирования, но разными методами. Входные языки: *B(PN)^2 (Basic Programmin Notation for Petri Nets), PFA (Parallel Finite Automata), Petri Nets*. Поддерживаемые способы задания требований корректности: *CTL, LTL*. Последняя версия – 21.09.2004. *Скорее всего, не представляет ценности для исследования.*
TIMES (A Tool for Modeling and Implementation of Embedded Systems)	*TIMES* – инстумент моделирования и анализа расписания *(schedulability analysis)* для встроенных систем реального времени. Разработан в *Uppsala University*. Применим для систем, которые могут быть описаны как множество приоритетных или неприоритетных задач, запускаемые периодически или спонтанно по времени или при появлении определённых внешних событий. Предоставляет графический интерфейс для редактирования и симуляции.

	Скорее всего, бесполезен для исследования.
TPS (TPS Theorem Proving System)	Ещё один доказатель. Последняя версия – 01.12.2002. *Бесполезен для исследования.*
TRON (Uppaal TRON)	*Uppaal TRON* – инструмент для проведения тестирования согласованности (используется верификация моделей). Языки моделирования – *Timed Automata*. Последняя версия – 27.04.2007. *Ценность для исследования неизвестна.*
Truth	Среда для верификации. Имеет модульную структуру. Поэтому процесс добавления новых языков весьма прост. Поддерживает языки: *CCS (Calculus of Communicating Systems), Petri Nets*. Задание требований корректности – *AFMC (Alternation Free Modal mu-Calculus)*. Последняя версия – 07.04.2002. *Ценности для исследования не представляет.*
TwoTowers	*TwoTowers* – программное обеспечение для функциональной верификации, анализа безопасности, вычисления быстродействия. Использует верификатор *NuSMV* для символьной верификации, марковские цепи, симуляцию дискретных событий, производит анализ потока информации. Входные языки: *AEmilia, EMPAgr (Extended Markovian Process Algebra with generative-reactive synchronizations)*. Поддерживаемые способы задания требований корректности – язык *LTL*. Последняя версия – 17.01.2006. *Ценность для исследования, скорее всего, низка.*
VeriSoft	Система для тестирования программного обеспечения. Поддерживаемые языки: *C, C++, Java.* Поддерживаемые способы задания корректности: *LTL, Monitor automata*. Последняя версия датирована 2006 годом. Имеет графический интерфейс. *Скорее всего, не представляет ценности для исследования.*
VSE-II	Ещё один доказатель. Язык моделирования – *VSE-SL*. Последняя версия – 06.10.2003. *Бесполезен для исследования.*
Ymer	Инструмент *Ymer* реализует алгоритм верификации, который не зависит от модели. Предназначен для верификации свойств реального времени систем с дискретными событиями с помощью отбора образцов *(acceptance sampling)*. Поддерживаемый входной язык – *PRISM language (PRISM model description language)*. Поддерживаемые способы задания требований корректности: *CSL (Continuous Stochastic Logic), PCTL (Probabilistic Computation Tree Logic).* Последняя версия – 01.02.2005. *Ценности для исследования, скорее всего, не представляет.*

	Стоит ограничиться изучением аналогов.
J-LO (the Java Logical Observer)	*J-LO* – инструмен для проверки темпоральных утверждений во время исполнения *(tool for runtime-checking temporal assertions)* для приложений, написанных на языке *Java 5*. Темпоральные свойства могут быть заданы на языке *LTL*, с использованием *AspectJ*. Свойства записываются в прямо в коде в виде аннотаций языка *Java*. Последняя версия инструмента датирована 2006 годом. *Скорее всего, бесполезен для исследования*

2.4. Обзор выбранных инструментов

На основе предварительного анализа, были выделены наиболее интересные на данном этапе исследования верификаторы. Каждый из них имеет ту или иную важную особенность или использует теоретические положения, которые могут быть полезны для исследования, проводимого по контракту:

- Bogor;
- Cadena;
- CADP (Construction and Analysis of Distributed Processes;
- CBMC (A Bounded Model Checker for C/C++ programs);
- GEAR (a game based model checking tool capable of CTL, modal μ-calculus and specification patterns);
- Java Pathfinder;
- LTSA (Labelled Transition System Analyser);
- MOPS (Modelchecking Programs for Security properties);
- NuSMV (A New Symbolic Model Checker);
- ORIS (uses a CTL-like temporal logic with real-time bounds, action and state based);
- ProB;
- SMV (Symbolic Model Checker);
- SPIN;
- TLC;
- UPPAAL (Uppaal Model Checker);
- VIS (Verification Interacting with Synthesis);
- dSPIN;
- Atelier B;
- Bandera;
- DBRover;
- FDR2;
- Mocha;
- PROD;
- Reactis Tester;
- Temporal Rover;
- J-LO (the Java Logical Observe).

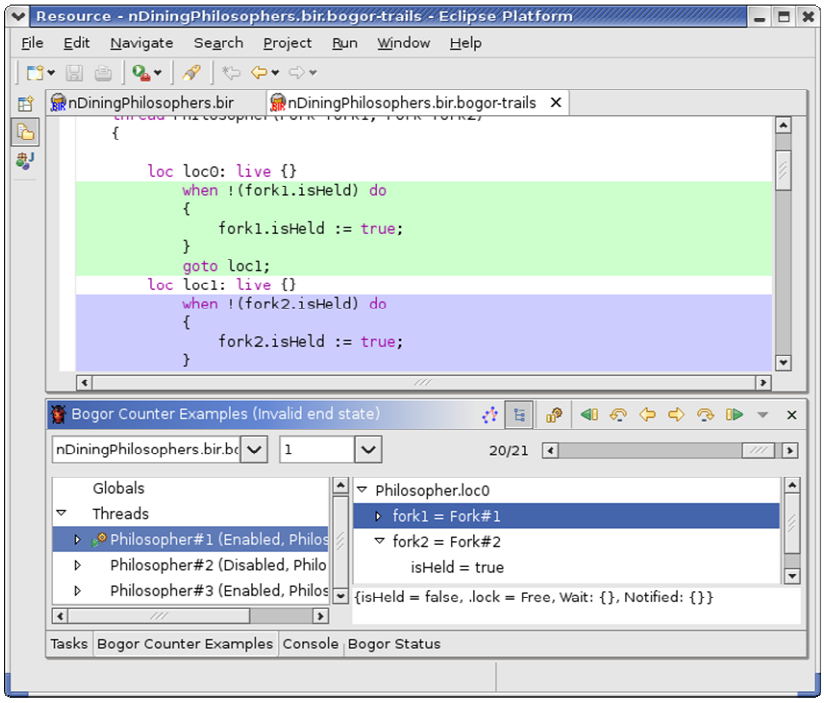
Приведем описание некоторых из этих инструментов. Это верификаторы, в которых интересен принцип работы.

2.4.1. Bogor

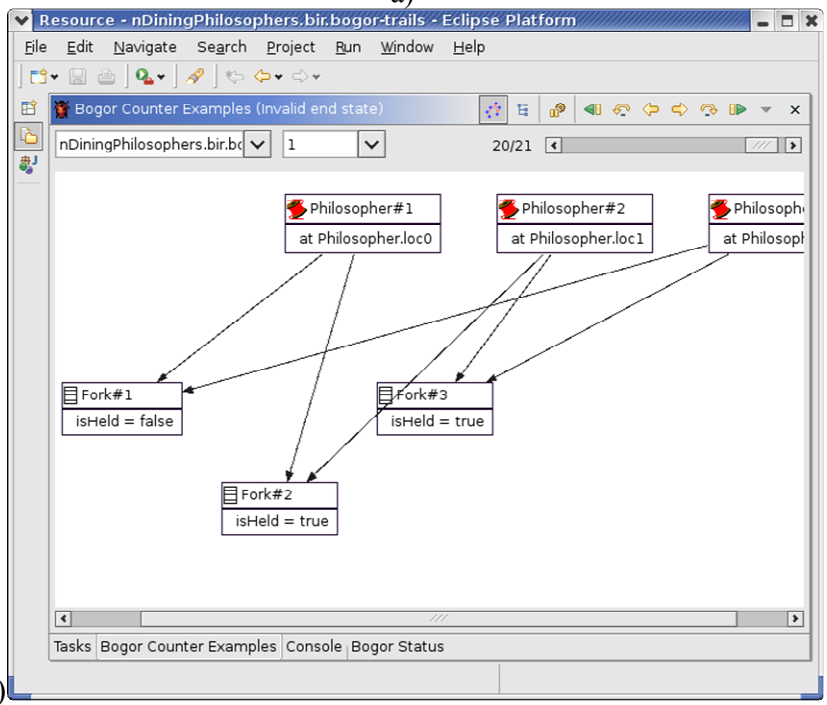
Bogor – расширяемый верификатор моделей программного обеспечения. Включает в себя системы визуализации и графический пользовательский интерфейс, разработанные для поддержки как основной, так и специализированной области *(domain-specific)* проверки моделей. Хотя существует много других программ, способных осуществлять проверку моделей, верификатор Bogor обеспечивает оригинальные возможности, которые делают его подходящим для проверки свойств разного рода программных продуктов, для построения собственного специализированного инструмента и для обучения концепции проверки моделей.

Этим инструментом обеспечивается:

- прямая поддержка возможностей объектно-ориентированных языков, позволяющих эффективно строить распределённые и параллельные системы, такие как динамическое создание потоков и объектов, наследование, виртуальные методы, исключения, сборщик мусора и т.д.;
- язык моделирования инструмента *Bogor* может быть расширен новыми примитивными типами, выражениями и командами, ассоциированными с предметной областью (например, мультиагентные системы, авионика, протоколы безопасности и т.д.) и может выбран соответствующий уровень абстракции (например, разработка модели, исходный код, байт-код и т.д.);
- открытая архитектура инструмента *Bogor* и хорошо организованная система модулей позволяют добавлять новые алгоритмы (например, для исследования пространства состояний, хранения состояний и т.д.) и разновидности оптимизации (например, эвристические алгоритмы поиска стратегий, специфическое для предметной области назначение сроков (domain-specific scheduling) и т.д.). Они могут заменять загруженные по умолчанию алгоритмы проверки моделей инструмента *Bogor*;
- *Bogor* обладает мощным графическим интерфейсом (рис. 1), реализованным как плагин для *Eclipse*™ (универсальная платформа с открытым кодом). Этот пользовательский интерфейс обеспечивает механизм для сборки различных конфигураций инструмента *Bogor*, описания коллекций свойств, а также различные варианты визуализации и навигации.



а)



а)

б)

Рис. 1. Графический интерфейс инструмента *Bogor*, реализованный как плагин для *Eclipse*™

- *Bogor* – это удобный педагогический инструмент для обучения верификации моделей, так как он позволяет наблюдать чистую реализацию базовых алгоритмов проверки моделей, легко улучшать и расширять эти алгоритмы.

Отметим, что инструмент *Bogor* разработан не только для получения мощного средства проверки моделей современных широкомасштабных систем, но и может служить основой для построения других специфических для предметной области верификаторов моделей.

Bogor проверяет системы, описанные на исправленной версии *BIR (Bandera Intermediate Representation)*. Предыдущая версия *BIR* была разработана для промежуточного языка, используемого в верификаторе *Bandera* для трансляции *Java*-программ во входной язык существующего верификатора моделей, такого как *SPIN*. Таким образом, эта более ранняя версия давала прямую поддержку для моделирования таких возможностей языка *Java* как, например, потоки и ограниченную форму выделения памяти в куче. Для того чтобы облегчить конструирование трансляторов для таких верификаторов моделей, как *SPIN*, действия и поток управления языка *BIR* задаются в формате, который достаточно близок к языку *PROMELA*.

Особенностью языка *BIR*, относительно языков большинства других верификаторов моделей (включая *SPIN, NuSMV, SLAM, BLAST*), является то, что он поддерживает неограниченное динамическое создание потоков и объектов, хранящихся в куче, с освобождением памяти сборщиком мусора. Кроме того, язык *BIR* обеспечивает эффективное каноническое представление кучи, которое оказывается незаменимым для эффективной проверки распределённых и параллельных программных систем. Такие системы генерируют на протяжении своей работы большое количество различных экземпляров куч, которые отличаются положениями одних и тех же объектов, но при этом имеют одинаковый их набор (такие экземпляры куч неразличимы операциями работы с памятью языка *Java*). В инструменте *Bogor* используется каноническое представление куч (было разработано для верификатора *dSPIN*), которое обеспечивает соответствие неразличимых куч одному состоянию. Это может позволить сильно сократить число сгенерированных состояний при верификации.

2.4.2. CBMC

CBMC – верификатор, который обеспечивает возможность ограниченной проверки моделей *(Bounded Model Checker)* для языков *ANSI-C* и *C++*. Он позволяет верифицировать выход за границу массива (переполнение буфера), безопасность указателей, исключения и пользовательские утверждения *(user-specified assertions)*. Кроме того, он может проверять языки *ANSI-C* и *C++* на совместимость с другими языками, такими как *Verilog*. Это реализовано с помощью развертывания циклов в программе и пересылкой результата в компонент, решающий *SAT (boolean satisfiability problem)*.

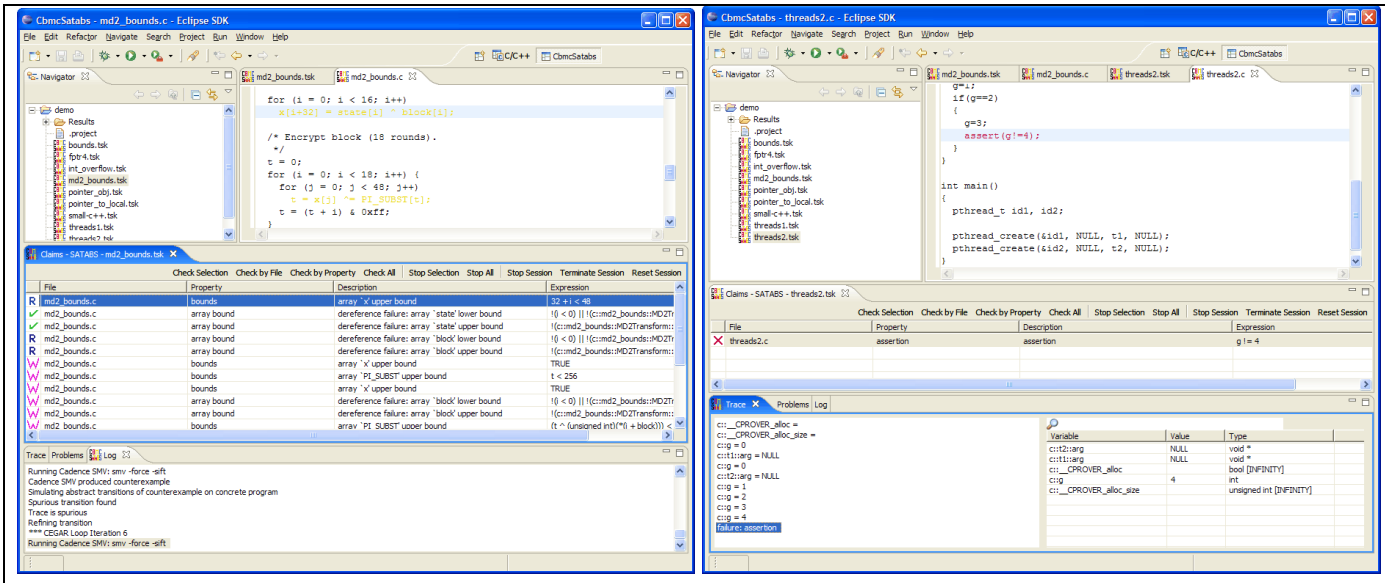


Рис. 2. Графический интерфейс инструмента CBMC, реализованный как плагин для Eclipse™

Хотя верификатор *CBMC* нацелен на встроенное программное обеспечение, он также поддерживает динамическое выделение памяти.

2.4.3. NuSMV

NuSMV – символьный верификатор моделей, разработанный как общий проект *Formal Methods Group* в подразделении *Automated Reasoning System* в *ITC- IRST*, *the Model Checking group* в *Carnegie Mellon University*, *the Mechanized Reasoning Group* в *University of Genova* и *Mechanized Reasoning Group* в *University of Trento*.

NuSMV – ещё одна реализация и расширение верификатора *SMV*, первого верификатора моделей, основанного на *BDD (Binary Decision Diagrams)*. *NuSMV* был разработан в качестве системы с открытой архитектурой для верификации моделей, которая может быть использована для верификации индустриальных разработок, как ядро составных верификационных инструментов и как основа для тестирования других технологий формальной верификации.

NuSMV со второй версии использует компонент, реализующий метод верификации моделей, основанный на *BDD*, и компонент, реализующий метод верификации моделей, основанный на *SAT*, также включающий верификатор моделей, основанный на *RBC (RBC-based Bounded Model Checker)*, подключённый к библиотеке *SIMSAT*.

2.4.4. SPIN

SPIN – это популярный инструмент с открытым кодом, используемый во многих организациях мира. Он может быть использован для формальной верификации распределённых систем. Инструмент был разработан в *Bell Lab* в 1980 году. Программный продукт был доступен свободно, начиная с 1991 года, и продолжал разрабатываться. Инструмент был награждён *ACM* престижной премией *System Software Award* за 2001 год.

Описание основных возможностей

Перечислим некоторые возможности, которые выделяют *SPIN* среди других верификаторов.

- Целью инструмента *SPIN* является эффективная верификация программного, а не аппаратного обеспечения. Он поддерживает высокоуровневый язык, с помощью которого можно специфицировать описание системы, названный *PROMELA (a PROcess MEta LAnguage)*. *SPIN* используется для отслеживания ошибок логического дизайна в операционных системах, протоколах обмена данными, системах переключателей, параллельных и распределённых алгоритмах, протоколах железнодорожной сигнализации и т.д. Инструмент проверяет логическую связность спецификации. Он сообщает о дедлоках, неспецифицированных приёмах, флагах незавершённости, состоянии гонки *(race conditions)* и негарантированных допущениях об относительной скорости процессов.
- *SPIN* (начиная с четвёртой версии) обеспечивает прямую поддержку для использования языка *C* как части спецификации модели. Это делает возможным проведение прямой верификации уровня реализации спецификации программного обеспечения, используя *SPIN* как драйвер и логический инструмент для верификации высокоуровневых темпоральных свойств.
- *SPIN* работает на лету. Поэтому нет необходимости в предварительном построении графа глобальных состояний или структуры *Крипке*.
- *SPIN* может быть использован как полная система *LTL*-проверки модели, поддерживающая все требования, выразимые в темпоральной логике линейного времени. Он может быть также эффективно использован как инструмент, верифицирующий на лету большое число постоянных и «живущих» свойств *(safety and liveness properties)*. Многие из этих свойств могут быть выражены и верифицированы без использования *LTL*.
- Свойства корректности могут быть описаны как инварианты системы или процесса (используются утверждения *(assertions*)), *LTL*-требования, автоматы Бюхи или более широко, как главные омега-регулярные свойства *(general omega-regular properties)* в синтаксисе *«never claims»*.
- Инструмент поддерживает динамические рост и уменьшение количества процессов, используя технику эластичных векторов состояний *(rubber state vector technique)*.
- Инструмент поддерживает организованный *(rendezvous)* и буферизированный обмен сообщениями, а также общение через разделяемую память. Поддерживаются также смешанные системы, использующие синхронный и асинхронный способ коммуникации. Идентификаторы канала обмена сообщениями могут быть посланы от одного процесса другому в сообщениях при любом способе организации обмена сообщениями.
- Инструмент поддерживает случайную, интерактивную и управляемую симуляцию для полной и частичной техники доказательств, основанной на поиске в глубину или в ширину. Инструмент был специально спроектирован с целью обеспечения хорошей масштабируемости и возможности обеспечения эффективности даже для задач достаточно большого объёма.
- Для оптимизации верификации *SPIN* использует эффективную технику сокращения частичного порядка *(partial order reduction)* и (опционально) *BDD*-схожие техники хранения *(BDD-like storage techniques)*.
- Для верификации дизайна формальная модель строится, используя язык *PROMELA*. *PROMELA* – это недетерминированный язык, не жёстко основанный на нотации командного языка Дейкстры и *CSP*-языке Хоара для ввода/вывода.

SPIN может быть использован в трёх базовых режимах:

- как симулятор, позволяющий переводить быстрое прототипирование со случайными, управляемыми или интерактивными симуляциями;
- как полный верификатор, предназначенный для точного доказательства валидности пользовательской спецификации требований корректности (используя теорию сокращения частичного порядка для оптимизации поиска);
- как система приближённого доказательства валидности даже очень больших моделей систем с максимальным покрытием пространства состояний.

Программный продукт *SPIN* полностью написан на языке *ANSI-C* и может быть портирован на все версии *Unix, Linux, cygwin, Plan9, Inferno, Solaris, Mac,* и *Windows*.

ЗАКЛЮЧЕНИЕ

После проведения подробного анализа существующих патентов и верификаторов, был сделан вывод, что на данный момент нет никаких серьёзных препятствий для разработки технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода.

Можно утверждать, что на данном этапе исследования верификаторы *Bogor* и *SPIN* являются наиболее приемлемыми для использования в исследований, проводимых, по контракту, так как это одни из наиболее популярных и мощных на данный момент инструментов верификации моделей.

Отметим также, что прямых аналогов разрабатываемой технологии верификации найдено не было.

ИСТОЧНИКИ

1. *Clark E., Emerson E., Sistla A.* Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications //ACM Transactions on Programming Languages and Systems. April 1986. Vol. 8. No 2, pp. 244–263.
2. Clarke *E.,* Grumberg *O.*, Long *D*. Model Checking /In Proceedings of the International Summer School on Deductive Program Design. Marktoberdorf, Germany, 1994.
3. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО, 2002. 416 с.
4. *Holzmann G*. Design and Validation of Computer Protocols. NJ: Bell Lab, 1991. 558 p.
5. *Holzmann G.* The Model Checker SPIN // Software Engineering Journal. 1997. Vol. 23. No. 5, pp. 279–295.
6. *Robby A., Dwyer M., Hatcliff J.* Bogor: an extensible and highly-modular software model checking framework / In Proceedings European Software Engineering Conference (ESEC'03), 2003.
7. *Cimatti A., Clarke E., Giunchiglia E et al*. NuSMV 2: An open source tool for symbolic model checking / In Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02). Lecture Notes in Computer Science 2404, pp.359–364. 2002.
8. *Brayton R. et al.* VIS: A system for verification and synthesis / In Proceedings of the Eighth International Conference on Computer Aided Verification (CAV'96), pp. 428–432. Springer Verlag. Rutgers University. 1996.
9. *US Patent Database Number Search.* http://patft.uspto.gov/netahtml/PTO/srchnum.htm, 5,317,757.
10. *US Patent Database Number Search.* http://patft.uspto.gov/netahtml/PTO/srchnum.htm, 6,324,496.
11. *Патентный закон РФ.* http://www.legal-support.ru/information/laws/intellect/patent-law.html
12. *Model checking.* http://en.wikipedia.org/wiki/Model_checking
13. *База инструментов верификации YAHODA.* http://anna.fi.muni.cz/yahoda/
14. ГОСТ Р. 15.011-96 «Система разработки и постановки продукции на производство. Патентные исследования»

ПРИЛОЖЕНИЕ А

УТВЕРЖДАЮ
Руководитель работ
докт. техн. наук., профессор
А. А. Шалыто
31 мая 2007 г.

ЗАДАНИЕ № 2007.05.31-01
на проведение патентных исследований

Наименование работы (темы): Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода.
Шифр работы (темы): 2007-4-1.4-18-02-041.
Этап работы: определение уровня развития техники и патентоспособности.
Сроки выполнения: 01.06.2007 г. – 31.08.2007 г.
Задачи патентных исследований: определение уровня развития техники и патентоспособности разрабатываемой технологии, получение сведений об охранных и иных документах, которые будут препятствовать применению этих результатов на территории Российской Федерации.

КАЛЕНДАРНЫЙ ПЛАН

Виды патентных исследований	Подразделения-исполнители (соисполнители)	Ответственные исполнители (Ф.И.О.)	Сроки выполнения патентных исследований. Начало. Окончание	Отчетные документы
Определение уровня развития техники и патентоспособности		Красс А.Л.	01.06.2007 – 31.08.2007	Отчет о патентных исследованиях

Декан факультета «Информационные технологии и программирование»
СПбГУ ИТМО
докт. техн. наук, профессор — В. Г. Парфенов

Начальник отдела
интеллектуальной собственности и
научно-технической информации — Л. Н. Казар

ПРИЛОЖЕНИЕ Б

Регламент поиска № 2007.05.31-01

31 мая 2007 г.

Наименование работы (темы): Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода
Шифр работы (темы): 2007-4-1.4-18-02-041.
Этап работы: определение уровня развития техники и патентоспособности.
Номер и дата утверждения задания: 2007.05.31-01, 31.05.2007 г.
Цель поиска информации: создание перечня документов, которые потенциально могут нарушить патентоспособность разрабатываемой технологии и препятствовать их применению на территории Российской Федерации.
Начало поиска: 01 июня 2007 г.
Окончание поиска: 31 августа 2007 г.

Предмет поиска (объект исследования, его составные части, товар)	Страна поиска	Источники информации, по которым будет проводиться поиск								Ретро-спектив-ность	Наименование информационной базы (фонда)
		патентные		НТИ*		конъюнктурные		другие			
		Наименование	Классификационные рубрики: МПК (МКИ)*, МКПО*, НКИ* и другие	Наименование	Рубрики УДК* и другие	Наименование	Код товара: ГС*, СМТК*, БТН*	Наименование	Классификационные индексы		
1	2	3	4	5	6	7	8	9	10	11	12
Предотвращение ошибок с помощью тестирования или отладки программного продукта Верификация программного кода, на пример, верификация байт-кода	Россия, США, Европа	База данных Федеральной службы по интеллектуальной собственности, патентам и товарным знакам Российской Федерации. Бюро по патентам и товарным знакам США. Европейского патентного бюро. Всемирной организации интеллектуальной собственности.	G06F 11/36 G06F 9/445V	-	-	-	-	-	-	20 лет	Роспатент, www.fips. ru USPTO, www.uspto. gov EPO, ep.espacenet. com ВОИС, www.wipo. int

Руководитель темы заведующий кафедрой
«Технологий программирования»,
докт. техн. наук, профессор _____ А. А. Шалыто
подпись

Начальник отдела
интеллектуальной собственности и _____ Л. Н. Казар
научно-технической информации подпись

*МПК (МКИ) — международная патентная классификация (международная классификация изобретений);
НКИ — национальная классификация изобретений;
МПКО — международная классификация промышленных образцов;
НТИ — научно-техническая информация;
ГС— гармонизированная система (гармонизированная товарная номенклатура);
СМТК — стандартная международная торговая классификация ООН;
БТН — Брюссельская таможенная номенклатура;
УДК— универсальная десятичная классификация.

ПРИЛОЖЕНИЕ В

Отчёт о поиске № 2007.08.31-01

УТВЕРЖДАЮ
Руководитель работы
докт. техн. наук., профессор
А. А. Шалыто
31 мая 2007 г.

Поиск проведен в соответствии с заданием на проведение патентных исследований № 2007.05.31-01 от 31 мая 2007 г. и регламентом поиска 2007.05.31-01 от 31 мая 2007 г.

Этап работы: обеспечение патентоспособности разрабатываемой технологии верификации на первом этапе исследований по контракту.

Начало поиска – 01 июня 2007 г. Окончание поиска – 31 августа 2007 г.

Сведения о выполнении регламента поиска: поиск выполнен полностью. Документов, которые могут препятствовать применению разрабатываемой технологии на территории Российской Федерации, не найдено.

На дальнейших этапах работ по указанному контракту необходимо проведение дополнительных патентных исследований для выявления новых патентов и непатентных работ в рассматриваемой области.

Материалы, отобранные для последующего анализа: отсутствуют.

Декан факультета «Информационные технологии и программирование»
СПбГУ ИТМО
докт. техн. наук, профессор — В. Г. Парфенов

Начальник отдела
интеллектуальной собственности и
научно-технической информации — Л. Н. Казар