Статья опубликована в Научно-техническом вестнике ИТМО. 2014. № 1 (89), с. 145 – 162.
УДК 004.05

ВЕРИФИКАЦИЯ ПАРАЛЛЕЛЬНЫХ АВТОМАТНЫХ ПРОГРАММ

М.А. Лукин

Рассмотрен интерактивный метод верификации параллельных автоматных программ, в которых иерархические автоматы могут реализовываться в разных потоках и взаимодействовать друг с другом. Верификация проводится при помощи инструментального средства Spin, включает в себя автоматическое построение модели на языке Promela, приведение LTL-формулы в формат, определяемый инструментальным средством Spin и построение контрпримера в терминах автоматов. Интерактивная верификация позволяет сократить время верификации и увеличить максимально возможный размер верифицируемых программ. Рассмотренный метод позволяет верифицировать параллельную систему иерархических автоматов, которые взаимодействуют между собой через сообщения и общие переменные. Особенность автоматной модели состоит в том, что каждый автомат объявляется как новый тип данных и может иметь произвольное (но заранее заданное) число экземпляров. Каждый конечный автомат в системе может запускать другой автомат в новом потоке или иметь вложенный автомат. Была проведена апробация инструментального средства Stater, разработанного на основе данного метода. На всех примерах Stater отработал правильно.

Ключевые слова: автоматы, параллельные автоматные программы, верификация, проверка моделей, линейная темпоральная логика, Spin.

Введение

Формальные методы все шире используются для обеспечения качества программного обеспечения. Эти методы не конкурируют с традиционным тестированием, а дополняют его. В данной работе рассматривается верификация методом проверки моделей (*model checking*) [1–3] для автоматных программ при помощи верификатора *Spin* [4]. Метод проверки моделей характеризуется высокой степенью автоматизации [1]. По данной теме проводятся исследования в России и за рубежом [5–30]. Большинство из этих работ верифицируют автоматные модели, которые сложно охватить взглядом. Таким образом, теряется одно из главных достоинств автоматных программ – наглядность. Например, блок-схемы и SDL-диаграммы представляют собой, по сути, одномерную структуру и почти всегда вылезают за пределы одного экрана, т.е. их нельзя охватить взглядом. Автоматные модели в работах [10–12, 15–19, 25, 26] избавлены от этого недостатка, но они предназначены для однопоточных программ. Метод [28] предназначен для многопоточных программ, но в нем нет возможности задавать свою спецификацию при помощи темпоральной или другой логики. Проводится проверка только заранее заданных свойств. В настоящей работе, которая является продолжением исследований [10, 17, 22], ставится задача построения метода верификации параллельных автоматных программ, обладающих свойством наглядности.

В данной работе предлагается метод интерактивной верификации параллельных автоматных программ. На основе предложенного подхода было разработано инструментальное средство *Stater*, которое позволяет создавать параллельную систему конечных иерархических автоматов, импортировать конечные автоматы из инструментального средства *Stateflow*, верифицировать созданную систему конечных автоматов при помощи верификатора *Spin* и генерировать программный код по созданной системе конечных автоматов.

Описание автоматной модели

Предлагаемый подход предназначен для построения распределенных систем взаимодействующих иерархических конечных автоматов [31–33]. При этом каждый такой автомат работает в отдельном потоке. Под иерархическим автоматом в настоящей работе понимается система вложенных автоматов.

В данной работе каждый граф переходов задает не конкретный автомат, а тип автоматов **(**по аналогии с типом данных или классом в объектно-ориентированном программировании (ООП)). Назовем его *автоматным типом*. У каждого автоматного типа может быть несколько экземпляров (по аналогии с объектом в ООП). Назовем эти объекты *автоматными объектами*. Каждый автоматный объект имеет уникальное имя. В дальнейшем, если не указано иное, автоматные объекты будут называться просто автоматами.

Переходы автоматов осуществляются по событиям. Также на переходе могут быть охранные условия [34]**.** Если встретилось событие, по которому нет перехода, то автомат может либо завершить работу и перейти в недопускающее состояние, либо игнорировать это событие. Все события общие для всей системы автоматов.

Вводится специальное событие «*», которое означает переход по любому событию, кроме тех, которые указаны на других переходах из этого состояния (аналог *default* в блоках *switch* для *C*-подобных языков или *else* в условных конструкциях).

Автомат может иметь конечное число переменных целочисленных типов (включая массивы). Для переменных вводятся следующие модификаторы:

– *volatile* – переменная может применяться в любом месте программы;
– *external* – переменная может использоваться другим автоматом;
– *param* – переменная является параметром автомата.

По умолчанию считается, что переменная не используется нигде, кроме как на диаграмме переходов автомата.

Выходные воздействия автомата бывают двух типов:

1. на переходах и в состояниях может быть выполнен любой код, однако верификатор и генератор кода перенесут его без изменений, поэтому код должен быть допустимым в целевом языке;
2. на переходах и в состояниях запускаются функции, определяемые пользователем на целевом языке программирования (после того, как сгенерирован код).

Автомат может иметь вложенные автоматы любого типа, кроме собственного, иначе будет бесконечная рекурсия. Циклическая рекурсия также запрещена.

Автомат может запускать поток с новым автоматом любого типа. Задается тип автомата <StateMachine> и имя <concreteStateMachine>. Нельзя запускать несколько автоматов с одним именем. Нельзя запускать автоматы своего типа.

Автомат может взаимодействовать с другим автоматом, выступая источником событий для него. События формируются асинхронно. Автомат может использовать переменные другого автомата, отмеченные специальным модификатором.

Таким образом, в системе могут быть несколько автоматов с одинаковым графом переходов, более того, часть этих автоматов могут быть вложенными, а часть не обладать этим свойством.

Описание процесса верификации

Для того чтобы провести верификацию программы методом проверки моделей, требуется составить модель программы и формализовать требуемые свойства (спецификацию) на языке *темпоральной логики* [1]. Так как в данной работе используется верификатор *Spin*, то языком темпоральной логики является *LTL* [1]. Так как модель строится для автоматной программы, то это может быть выполнено автоматически. Построение модели описано в разделе «Генерация кода на языке *Promela*».

Обозначим автоматный тип через `AType`, автоматный объект через `aObject`. Пусть состояния `AType` называются `s0`, `s1` и т. д., в автомат поступают события `e0`, `e1` и т. д., а переменные называются `x0`, `x1` и т. д., внешние воздействия второго типа `z0`, `z1` и т. д. Пусть автоматный тип `AType` имеет вложенный автомат `nested`. Пусть `AType` запускает автомат `fork`.

Процесс верификации состоит из следующих этапов.

1. Построение модели – генерация кода на языке *Promela* [4]. Для автоматных программ, как отмечено выше, это выполняется автоматически.
2. Преобразование *LTL*-формул (переход от нотации автоматной программы в нотацию *Spin*).
3. Запуск верификатора *Spin*.
4. Преобразование контрпримера в термины исходной системы автоматов. Это преобразование автоматных программ также выполняется автоматически аналогично работе [22].

Этапы процесса верификации описаны ниже. Эти этапы похожи на этапы ручной верификации при помощи *Spin*. Основным отличием является бо́льший уровень автоматизации и бо́льшая приближенность модели к реализации, чем при верификации неавтоматных программ. Ниже описана реализация интерактивности, а затем все четыре этапа верификации.

Интерактивность

Одна из главных проблем при верификации методом проверки моделей – это размер модели *Крипке*. Для того чтобы уменьшить модель (отсечь лишние подробности), будем строить ее интерактивно. Для обеспечения интерактивности вводится возможность выбирать, какие уровни абстракции автоматной системы входят в модель, а какие нет. Кроме того, модель структурируется понятным для человека образом для того, чтобы пользователь мог самостоятельно модифицировать построенную модель. Ниже описаны уровни абстракции по разным аспектам верификации: по переменным, параллелизму и источникам событий.

Переменные. Для переменных введем следующие уровни абстракции.

1. Переменные в модели не учитываются.

2. Переменные в модель включены, но модель абстрагируется от их значения. Недетерминированно выбирается, какое охранное условие будет верно.
3. Модель вычисляет значения переменных. При этом переменные могут быть следующих видов.
   - Локальные. Эти переменные могут быть изменены только самим конечным автоматом. Все изменения таких переменных находятся только в выходных воздействиях автомата.
   - Параметры. Извне изменяются только один раз при запуске автомата. В остальном они подобны локальным переменным.
   - Публичные. Такие переменные могут быть изменены в любом месте программы, в которую входит построенная автоматная система. В модели перед каждым переходом автомата таким переменным недетерминированно присваивается произвольное значение.
   - Совместно используемые. К таким переменным данного автомата имеют доступ другие автоматы, параллельно работающие с данным автоматом.

Параметры и публичные переменные могут быть также одновременно и совместно используемыми.

Параллелизм. Вводятся два уровня: параллелизм поддерживается либо нет. Если параллелизм не поддерживается, то в модель не вводятся взаимодействия параллельных автоматов, остаются только взаимодействия по вложенности.

Источники событий. В качестве источников событий для автоматов в системе могут выступать внешняя среда и другие автоматы. Внешняя среда как источник событий для каждого автомата может работать в одном из трех режимов:

- внешняя среда не взаимодействует с автоматом (события от внешней среды не приходят);
- внешняя среда отправляет только те события, которые автомат может в данный момент обработать;
- внешняя среда отправляет любые события.

Другие автоматы как источники событий можно отключить, если отключить параллелизм.

Генерация кода на языке Promela

Все состояния каждого автоматного типа перенумеровываются и для них создаются константы. Для каждого автоматного типа состояния нумеруются отдельно. Имя константы состоит из имени автоматного типа и имени состояния, разделенных знаком подчеркивания. Это сделано для того, чтобы состояния разных автоматов с одинаковыми именами не конфликтовали друг с другом.

Пример:

```
#define AType_s0 0
#define AType_s1 1
```

Все события перенумеровываются и для них создаются константы. Для событий применяется сквозная нумерация.

Пример:

```
#define e0 1
#define e1 1
```

Все внешние воздействия второго типа (вызываемые функции) перенумеровываются и для них создаются константы аналогично состояниям.

Все вызовы вложенных и запуски параллельных автоматов перенумеровываются аналогично состояниям.

Каждый тип автоматов записывается в *inline*-функцию, которая моделирует один шаг автомата. Переходы записываются при помощи *охранных команд Дейкстры* [34]. Для каждого типа автоматов создается структура.

Элементы структуры:

- `byte state` – номер текущего состояния;
- `byte curEvent` – номер последнего пришедшего события;
- `byte ID` – номер автомата;
- `byte functionCall` – номер последней запущенной функции, если такая существует;
- `byte nestedMachine` – номер текущего вложенного автомата, если такой существует;
- все переменные автомата.

Для каждого экземпляра автомата создается экземпляр структуры и *канал*, по которому происходит передача событий. Для каждого экземпляра автомата, кроме вложенных, создается *процесс*, который извлекает из канала событие и запускает *встраиваемую* (*inline*) функцию автомата с этим событием.

Для каждого экземпляра автомата, кроме вложенных, создается процесс, который недетерминированно выбирает событие и отправляет его в канал автомата.

Для *публичных переменных* на каждом шаге автомата вызывается специальная функция, которая их недетерминированно изменяет. Для переменных-*параметров* такая функция вызывается один раз – при запуске автомата.

Если по данному событию нет перехода, и в текущем состоянии есть вложенный автомат, то он запускается (запускается встраиваемая функция автомата).

Если в текущем состоянии автомат запускает другой автомат, то запускается заранее созданный процесс запускаемого автомата. Если автомат отправляет событие другому автомату, то он записывает его номер в канал этого автомата.

Апробация метода

Для поддержки описанного метода было разработано инструментальное средство *Stater*, которое позволяет построить параллельную систему автоматов, импортировать автоматы из *Stateflow*, провести верификацию системы автоматов и сгенерировать программный код, эквивалентный этой системе автоматов.

Апробация метода проводилась на нескольких программах. Несколько модулей *Stater* были разработаны при помощи самого инструмента *Stater*, а именно:

- модуль генерации программного кода;
- модуль преобразования *LTL*-формул;
- модуль импорта диаграмм из *Stateflow*;
- модуль загрузки диаграмм из файла.

Также был разработан прототип программы управления гусеничным шасси и несколько иных программ. Продемонстрируем предложенный подход на примере прототипа программы управления гусеничным шасси для робота. В шасси два двигателя: по одному на левую и правую гусеницы.

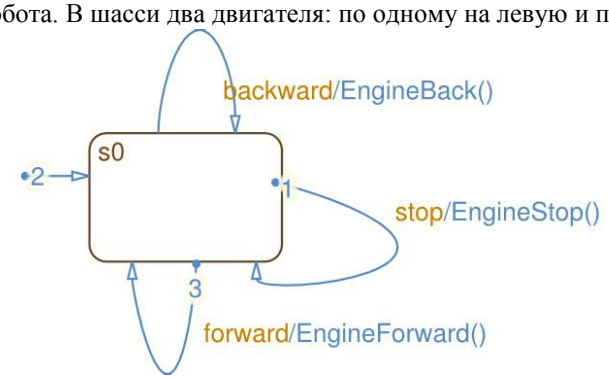

Рис. 1. Граф переходов автоматного типа AEngine

Прототип программы состоит из двух автоматных типов: AEngine и AManager. Два автомата left и right типа AEngine (рис. 1) управляют соответственно левым и правым двигателями.

Автомат типа `AManager` (рис. 2) отправляет команды на управление двигателями в зависимости от команд для шасси. При входе в состояния он отправляет события автоматам AEngine (слева от стрелки написано имя автомата, справа – событие):

- `Stopped: left ← stop, right ← stop.`
- `MoveForward: left ← forward, right ← forward.`
- `MoveBackward: left ← backward, right ← backward.`
- `TurnRight: left ← backward, right ← forward.`
- `TurnLeft: left ← forward, right ← backward.`
- `ForwardRight: left ← stop, right ← forward.`
- `ForwardLeft: left ← forward, right ← stop.`
- `BackwardRight: left ← backward, right ← stop.`
- `BackwardLeft: left ← stop, right ← backward.`

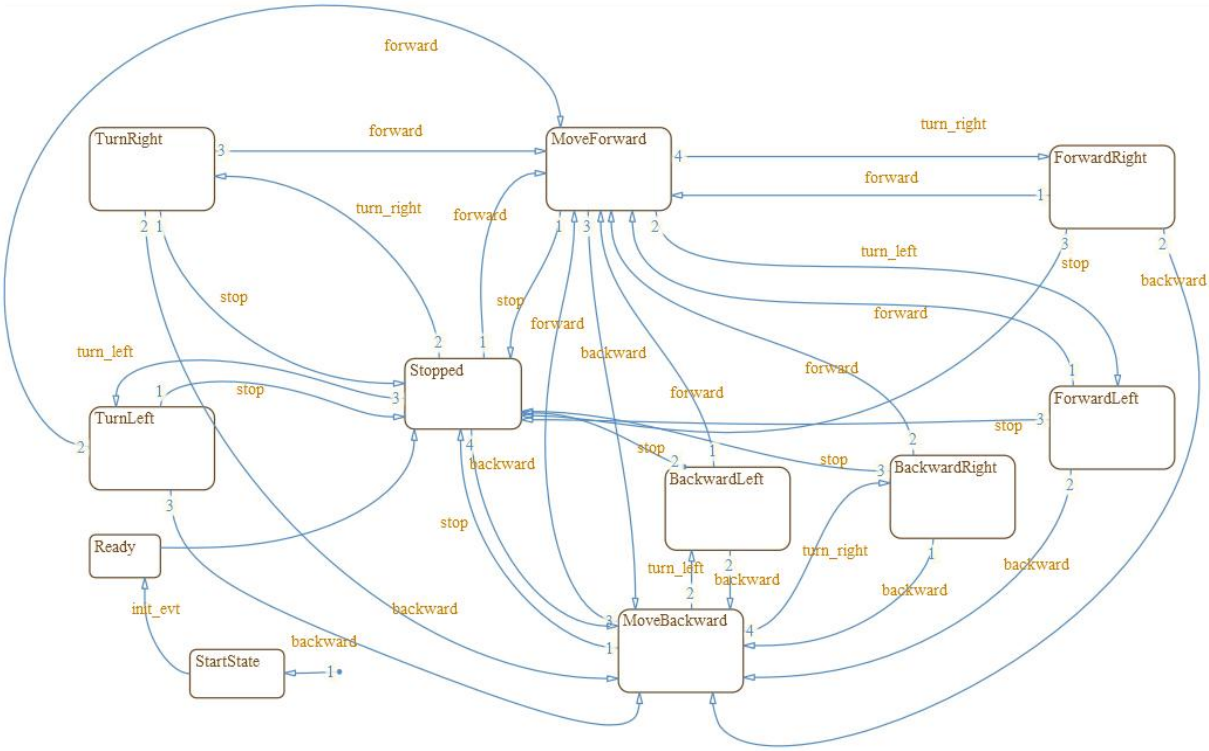

Рис. 2. Граф переходов автоматного типа AManager

Проверим свойство: «В любой момент если поступила команда «стоп», то будет подана команда остановки левого двигателя». Рассматриваемое свойство формализуется следующим образом: в любой момент времени в автомат manager пришло событие `stop`, следовательно, в будущем автомат `left` вызовет функцию `EngineStop`:

G ({manager.stop} => (**F** {left.EngineStop}))

Данное свойство не должно выполняться в следующих состояниях: `StartState`, `Ready` и `Stopped`. В первых двух шасси еще не готово к работе, а в состоянии Stopped двигатель и так остановлен. В итоге получаем следующую формулу:

[] (({manager.stop} && !{manager.StartState} && !{manager.Ready} && !{manager.Stopped}) -> (<> {left.EngineStop})) (1)

Выполняем верификацию с формулой (1) и получаем ответ, который означает, что верифицируемое свойство выполняется в построенной системе:

```
0. [] ( ({manager.stop} && !{manager.StartState} && !{manager.Ready}
&& !{manager.Stopped}) -> (<> {left.EngineStop} ))
Verification successful!
```

Во время апробации *Stater* отработал правильно на всех задачах.

Заключение

Таким образом, в работе продемонстрирован метод интерактивной верификации параллельных автоматных программ. Интерактивность помогает уменьшить модель и, тем самым, увеличить максимально возможный для верификации размер программ. Апробация метода показала его работоспособность. Основные результаты работы состоят в разработке метода верификации параллельных автоматных систем, которые отличаются от остальных важнейшим свойством – наглядностью. Применение интерактивности позволило сократить модель Крипке для верифицируемых программ, тем самым позволило увеличить размер программ, которые можно верифицировать. Перспективы работы: более полная верификация программ, разработанных в *Stateflow* а также статический анализ кода традиционных программ, основанный на автоматическом построении автоматных моделей по программам и верификации построенных моделей.

Литература

1. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. – М.: МЦНМО, 2002. – 416 с.

2. Вельдер С.Э., Лукин М.А., Шалыто А.А., Яминов Б.Р. Верификация автоматных программ. – СПб: Наука, 2011. – 244 с.
3. Карпов Ю.Г. Model Checking: верификация параллельных и распределенных программных систем. – СПб: БХВ-Петербург, 2010. – 560 с.
4. Официальный сайт инструментального средства Spin [Электронный ресурс]. – Режим доступа: http://spinroot.com, свободный. Яз. англ. (дата обращения 03.09.2013).
5. Gnesi S., Mazzanti F. On the fly model checking of communicating UML state machines. 2004 – [Электронный ресурс]. – Режим доступа: http://fmt.isti.cnr.it/WEBPAPER/onthefly-SERA04.pdf, свободный. Яз. англ. (дата обращения 25.11.2013).
6. Gnesi S., Mazzanti F. A model checking verification environment for UML statecharts // Proc. of XLIII Congresso Annuale AICA. 2005 [Электронный ресурс]. – Режим доступа: http://fmt.isti.cnr.it/~gnesi/matdid/aica.pdf, свободный. Яз. англ. (дата обращения 20.07.2013).
7. Канжелев С.Ю., Шалыто А.А. Автоматическая генерация автоматного кода // Информационно-управляющие системы. – 2006. – № 6. – С. 35–42.
8. Виноградов Р.А., Кузьмин Е.В., Соколов В.А. Верификация автоматных программ средствами CPN/Tools // Моделирование и анализ информационных систем. – 2006. – № 2. – С. 4–15.
9. Васильева К.А., Кузьмин Е.В. Верификация автоматных программ с использованием LTL // Моделирование и анализ информационных систем. – 2007. – № 1. – С. 3–14.
10. Лукин М.А. Верификация автоматный программ. Бакалаврская работа. СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/_lukin_bachelor.pdf, свободный. Яз. рус. (дата обращения 25.11.2013).
11. Яминов Б.Р. Автоматизация верификации автоматных UniMod-моделей на основе инструментального средства Bogor. Бакалаврская работа. СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/jaminov_bachelor.pdf, свободный. Яз. рус. (дата обращения 25.11.2013).
12. Вельдер С.Э., Шалыто А.А. О верификации простых автоматных систем на основе метода Model Checking // Информационно-управляющие системы. – 2007. – № 3. – С. 27–38.
13. Ma G. Model checking support for CoreASM: model checking distributed abstract state machines using Spin. 2007 [Электронный ресурс]. – Режим доступа: http://summit.sfu.ca/item/8056, свободный. Яз. англ. (дата обращения 25.11.2013).
14. David A., Moller O., Yi W. Formal Verification of UML Statecharts with Real-time Extensions // Formal Methods. – 2006. – V. 3. – P. 15–22.
15. Егоров К.В., Шалыто А.А. Методика верификации автоматных программ // Информационно-управляющие системы. – 2008. – № 5. – С. 15–21.
16. Курбацкий Е.А. Верификация программ, построенных на основе автоматного подхода с использованием программного средства SMV // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 8 (53). – С. 137–144.
17. Лукин М.А., Шалыто А.А. Верификация автоматных программ с использованием верификатора SPIN // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 8 (53). – С. 145–162.
18. Гуров В.С., Яминов Б.Р. Верификация автоматных программ при помощи верификатора UNIMOD.VERIFIER // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 8 (53). – С. 162–176.
19. Егоров К.В., Шалыто А.А. Разработка верификатора автоматных программ // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 8 (53). – С. 177–188.
20. Гуров В.С., Мазин М.А., Шалыто А.А. Автоматическое завершение ввода условий в диаграммах состояний // Информационно-управляющие системы. – 2008. – № 1. – С. 24–33.
21. Prashanth C.M., Shet K.C. Efficient Algorithms for Verification of UML Statechart Models // Journal of Software. – 2009. – V. 3. P. 175 – 182.
22. Лукин М.А. Верификация визуальных автоматных программ с использованием инструментального средства SPIN. Магистерская диссертация. СПбГУ ИТМО, 2009 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/_lukin_master.pdf, свободный. Яз. рус. (дата обращения 25.11.2013).
23. Тимофеев К.И., Астафуров А.А., Шалыто А.А. Наследование автоматных классов с использованием динамических языков программирования (на примере языка RUBY) // Информационно-управляющие системы. – 2009. – № 4. – С. 21–25.
24. Ремизов А.О., Шалыто А.А. Верификация автоматных программ // Сб. докл. науч.-техн. конф. «Состояние, проблемы и перспективы создания корабельных информационно-управляющих комплексов ОАО «Концерн Моринформсистема Агат». – М., 2010. – С. 90–98 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/_2010_05_25_verific.pdf, свободный. Яз. рус. (дата обращения 25.11.2013).
25. Клебанов А.А., Степанов О.Г., Шалыто А.А. Применение шаблонов требований к формальной спецификации и верификации автоматных программ // Семантика, спецификация и верификация программ: теория и приложения: Тр. семинара. – 2010. – С. 124–130 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/_2010-10-01_klebanov.pdf, свободный. Яз. рус. (дата обращения 25.11.2013).

26. Вельдер С.Э., Шалыто А.А. Верификация автоматных моделей методом редуцированного графа переходов // Научно-технический вестник СПбГУ ИТМО. – 2009. – № 6 (64). – С. 66–77.
27. Янкин Ю.Ю., Шалыто А.А. Автоматное программирование ПЛИС в задачах управления электроприводом // Информационно-управляющие системы. – 2011. – № 1. – С. 50–56.
28. Chen C., Sun J., Liu Y., Dong J., Zheng M. Formal modeling and validation of Stateflow diagrams // International Journal on Software Tools for Technology Transfer. – 2012. – V. 6. – P. 653–671.
29. Малаховски Я.М., Корнеев Г.А. Применение зависимых систем типов со структурной индукцией для верификации реактивных программ // Научно-технический вестник информационных технологий, механики и оптики. – 2012. – № 6 (82). – С. 63–67.
30. Катеринченко Р.С., Бессмертный И.А. Верификация данных в системах отслеживания задач с помощью продукционных правил // Научно-технический вестник информационных технологий, механики и оптики. – 2013. – № 1 (83). – С. 86–90.
31. Шалыто А.А. Switch-технология. Алгоритмизация и программирование задач логического управления. – СПб: Наука, 1998. – 617 с.
32. Cardei I., Jha R., Cardei M. Hierarchical architecture for real-time adaptive resource management. Secaucus. – NJ. USA: Springer-Verlag, 2000. – 20 p.
33. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб: Питер, 2010. – 176 с.
34. Dijkstra E.W. Guarded commands, non-determinacy and formal derivation of programs // CACM. – 1975. – V. 8. – P. 453–457 [Электронный ресурс]. – Режим доступа: http://www.cs.virginia.edu/~weimer/615/reading/DijkstraGC.pdf, свободный. Яз. англ. (дата обращения 25.11.2013).