3 КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ COMPUTER SCIENCE

УДК 004.05

ВЕРИФИКАЦИЯ ПАРАЛЛЕЛЬНЫХ АВТОМАТНЫХ ПРОГРАММ

М.А. Лукин[a]

[a] Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, Санкт-Петербург, Россия, lukinma@gmail.com

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.

VERIFICATION OF PARALLEL AUTOMATA-BASED PROGRAMS

M. Lukin[b]

[b] Saint Petersburg National Research University of Information Technologies, Mechanics and Optics, Saint Petersburg, Russia, lukinma@gmail.com

The paper deals with an interactive method of automatic verification for parallel automata-based programs. The hierarchical state machines can be implemented in different threads and can interact with each other. Verification is done by means of Spin tool and includes automatic Promela model construction, conversion of LTL-formula to Spin format and counterexamples in terms of automata. Interactive verification gives the possibility to decrease verification time and increase the maximum size of verifiable programs. Considered method supports verification of the parallel system for hierarchical automata that interact with each other through messages and shared variables. The feature of automaton model is that each state machine is considered as a new data type and can have an arbitrary bounded number of instances. Each state machine in the system can run a different state machine in a new thread or have nested state machine. This method was implemented in the developed Stater tool. Stater shows correct operation for all test cases.

Keywords: state machines, parallel automata-based programs, verification, model checking, linear temporal logic, 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 выходят 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, обладающего свойством наглядности.

В данной работе предлагается метод интерактивной верификации параллельных автоматных программ. На основе предложенного подхода 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. На переходе могут также 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, во избежание бесконечной рекурсии. 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 существовать несколько автоматов с одинаковым графом переходов, более того, часть этих автоматов могут быть вложенными, а часть не обладать этим свойством.

Описание процесса верификации

Чтобы провести верификацию программы методом проверки моделей, требуется составить модель программы и формализовать требуемые свойства (спецификацию) на языке темпоральной логики [1]. В данной работе используется верификатор Spin, и 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 больший уровень автоматизации и бóльшая 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. Чтобы уменьшить модель (отсечь лишние подробности), будем строить ее интерактивно. Для этого вводится возможность выбирать, какие уровни абстракции автоматной системы входят в модель, а какие нет. Кроме того, модель структурируется понятным для человека образом, 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 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.

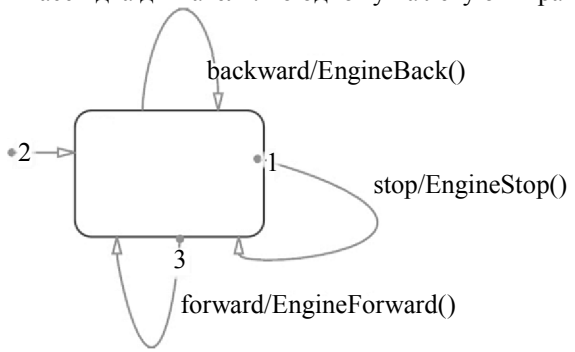

Рис. 1. Граф переходов автоматного типа AEngine

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.`

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.

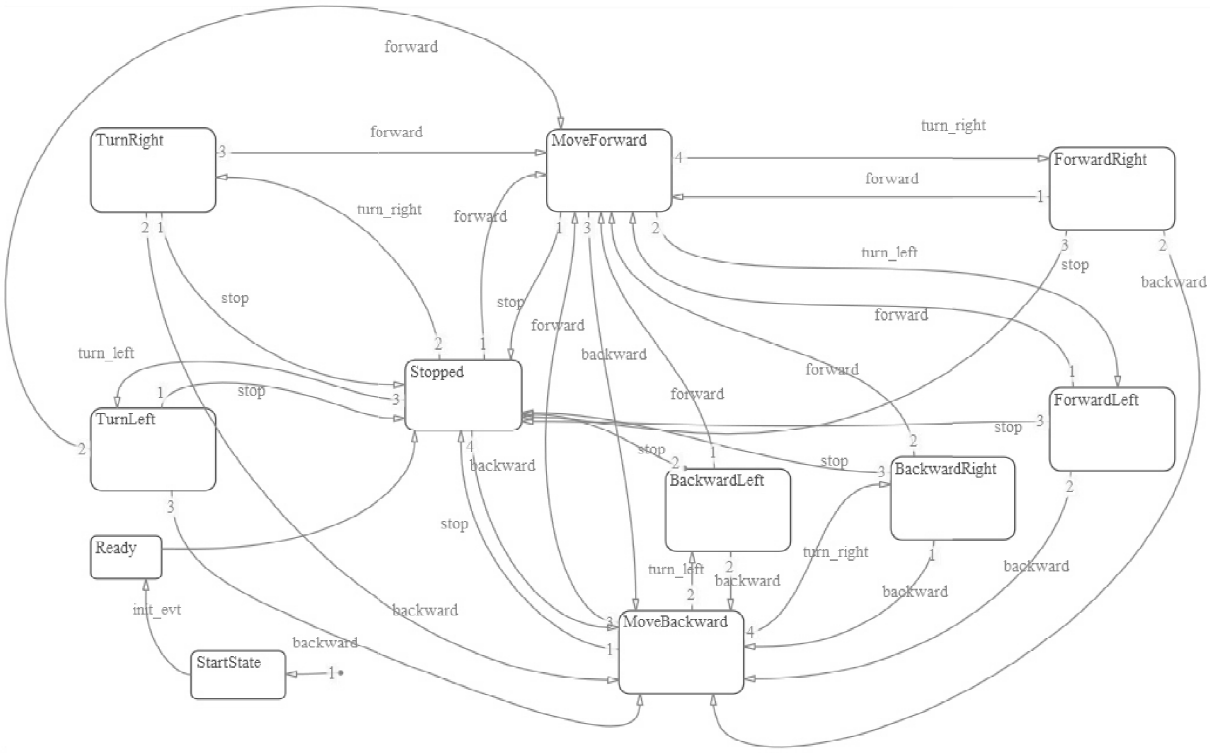

Рис. 2. Граф переходов автоматного типа AManager

Заключение

Таким образом, в работе продемонстрирован метод интерактивной верификации параллельных автоматных 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 и тем самым 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.

References

1. Clarke E.M., Grumberg O., Peled D. *Model Checking.* MIT Press, 1999, 314 p. (Russ. ed.: Klark E.M., Gramberg O., Peled D. *Verifikatsiya modelei programm: Model Checking.* Moscow, Moscow Center for Continuous Mathematical Education Publ., 2002, 416 p.)
2. Vel'der S.E., Lukin M.A., Shalyto A.A., Yaminov B.R. *Verifikatsiya avtomatnykh program* [Verification of automatic programs]. St. Petersburg, Nauka Publ., 2011, 244 p.
3. Karpov Yu.G. *Model Checking: verifikatsiya parallel'nykh i raspredelennykh programmnykh system* [Model Checking: verification of parallel and distributed software systems]. St. Petersburg, BKhV-Peterburg Publ., 2010, 560 p.
4. *Ofitsial'nyi sait instrumental'nogo sredstva Spin* [Official site tool Spin]. Available at: http://spinroot.com (accessed 03.09.2013).

5. Gnesi S., Mazzanti F. *On the fly model checking of communicating UML state machines*. 2004. Available at: http://fmt.isti.cnr.it/WEBPAPER/onthefly-SERA04.pdf (accessed 25.11.2013).
6. Gnesi S., Mazzanti F. A model checking verification environment for UML statecharts. *Proc. of XLIII Congresso Annuale AICA*. 2005. Available at: http://fmt.isti.cnr.it/~gnesi/matdid/aica.pdf (accessed 20.07.2013).
7. Kanzhelev S.Y., Shalyto A.A. Avtomaticheskaya generatsiya avtomatnogo koda [Automatic Generation of Automaton Code]. *Informatsionno-upravlyayushchie sistemy*, 2006, no. 6 (25), pp. 35–42.
8. Vinogradov R.A., Kuz'min E.V., Sokolov V.A. Verifikatsiya avtomatnykh programm sredstvami CPN/Tools [Automatic verification of programs by means of CPN/Tools]. *Modelirovanie i analiz informatsionnykh system – Modeling and analysis of information systems,* 2006, vol. 13, no. 2, pp. 4–15.
9. Vasil'eva K.A., Kuz'min E.V. Verifikatsiya avtomatnykh programm s ispol'zovaniem LTL [Automatic verification of programs using LTL]. *Modelirovanie i analiz informatsionnykh system – Modeling and analysis of information systems,* 2007, vol.14, no. 1, pp. 31–43.
10. Lukin M.A. *Verifikatsiya avtomatnyi programm. Bakalavrskaya rabota* [Verification of automata programs. Bachelor's work]. University ITMO, 2007. Available at: http://is.ifmo.ru/papers/_lukin_bachelor.pdf (accessed 25.11.2013).
11. Yaminov B.R. *Avtomatizatsiya verifikatsii avtomatnykh UniMod-modelei na osnove instrumental'nogo sredstva Bogor. Bakalavrskaya rabota* [Automation of verification of automata UniMod models based tool Bogor. Bachelor's work]. University ITMO, 2007. Available at: http://is.ifmo.ru/papers/jaminov_bachelor.pdf (accessed 25.11.2013).
12. Velder S.E., Shalyto A.A. O verifikatsii prostykh avtomatnykh sistem na osnove metoda Model Checking [Verification of Simple Automata-Based Programs using the Model Checking Method]. *Informatsionno-upravlyayushchie sistemy,* 2007, no. 3 (28), pp. 27–38.
13. Ma G. *Model checking support for CoreASM: model checking distributed abstract state machines using Spin*. 2007. Available at: http://summit.sfu.ca/item/8056 (accessed 25.11.2013).
14. David A., Möller M.O., Yi W. Formal Verification of UML Statecharts with Real-time Extensions. *Lecture Notes in Computer Science,* 2002, vol. 2306, pp. 218–232.
15. Egorov K.V., Shalyto A.A. Metodika verifikatsii avtomatnykh programm [The Method of Automata Programs Verification]. *Informatsionno-upravlyayushchie sistemy,* 2008, no. 5 (36), pp. 15–21.
16. Kurbatsky E.A. Verifikatsiya programm, postroennykh na osnove avtomatnogo podkhoda s ispol'zovaniem programmnogo sredstva SMV [Verification of automata-based programs with SMV tool]. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2008, no. 8 (53), pp. 137–144.
17. Lukin M.A., Shalyto A.A. Verifikatsiya avtomatnykh programm s ispol'zovaniem verifikatora SPIN [SPIN verification of automata-based programs with spin verifier]. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2008, no. 8 (53), pp. 145–162.
18. Gurov V.S., Jaminov B.R. Verifikatsiya avtomatnykh programm pri pomoshchi verifikatora UNIMOD.VERIFIER [Verification of automata-based programs with UNIMOD.VERIFIER tool]. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2008, no. 8 (53), pp. 162–176.
19. Egorov K.V., Shalyto A.A. Razrabotka verifikatora avtomatnykh programm [Development of a verifier for automata-based programs]. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2008, no. 8 (53), pp. 177–188.
20. Gurov V.S., Mazin M.A., Shalyto A.A. Avtomaticheskoe zavershenie vvoda uslovii v diagrammakh sostoyanii [Auto-Completion of Guard Condition Expressions in State Charts]. *Informatsionno-upravlyayushchie sistemy*, 2008, no. 1 (32), pp. 24–33.
21. Prashanth C.M., Shet K.C. Efficient Algorithms for Verification of UML Statechart Models. *Journal of Software*, 2009, vol. 4, no. 3, pp. 175–182.
22. Lukin M.A. *Verifikatsiya vizual'nykh avtomatnykh programm s ispol'zovaniem instrumental'nogo sredstva SPIN. Magisterskaya dissertatsiya* [Verification of visual automata programs using a tool to SPIN. Master's thesis]. University ITMO, 2009. Available at: http://is.ifmo.ru/papers/_lukin_master.pdf (accessed 25.11.2013).
23. Timofeev K.I., Astafurov A.A., Shalyto A.A. Nasledovanie avtomatnykh klassov s ispol'zovaniem dinamicheskikh yazykov programmirovaniya (na primere yazyka RUBY) [Automata Classes Inheritance with Dynamical Languages (with examples in RUBY)]. *Informatsionno-upravlyayushchie sistemy,* 2009, no. 4 (41), pp. 21–25.
24. Remizov A.O., Shalyto A.A. Verifikatsiya avtomatnykh program [Verification of automatic programs]. *Sbornik dokladov nauchno-tekhnicheskoi konferentsii "Sostoyanie, problemy i perspektivy sozdaniya korabel'nykh informatsionno-upravlyayushchikh kompleksov OAO "Kontsern Morinformsistema-Agat"* [Proc. of the Scientific-Technical Conference "Condition, problems and prospects of creation of ship-based information-control complexes of JSC "Concern Morinformsystem-Agat"]. Moscow, 2010, pp. 90–98. Available at: http://is.ifmo.ru/works/_2010_05_25_verific.pdf (accessed 25.11.2013).
25. Klebanov A.A., Stepanov O.G., Shalyto A.A. Primenenie shablonov trebovanii k formal'noi spetsifikatsii i verifikatsii avtomatnykh programm [Application templates requirements for formal specification and verification of automatic programs]. *Trudy seminara "Semantika, spetsifikatsiya i verifikatsiya programm: teoriya i prilozheniya"* [Proc. of the Seminar "Semantics, specification and verification of programs: theory and applications"]. 2010, pp. 124–130. Available at: http://is.ifmo.ru/works/_2010-10-01_klebanov.pdf (accessed 25.11.2013).
26. Velder S.E., Shalyto A.A. Verifikatsiya avtomatnykh modelei metodom redutsirovannogo grafa perekhodov [Model checking automata-based programs using reduced transition graph construction]. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2009, no. 6 (64), pp. 66–77.
27. Yankin Y.Y., Shalyto A.A. Avtomatnoe programmirovanie PLIS v zadachakh upravleniya elektroprivodom [A method of finite-state machine realization in electric motor drives control]. *Informatsionno-upravlyayushchie sistemy,* 2011, no. 1 (50), pp. 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, vol. 14, no. 6, pp. 653–671.
29. Malakhovski Ya.M., Korneev G.A. Primenenie zavisimykh sistem tipov so strukturnoi induktsiei dlya verifikatsii reaktivnykh program [Verification of reactive programs by dependent type systems with structural induction]. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2012, no. 6 (82), pp. 63–67.
30. Katerinenko R.S., Bessmertny I.A. Verifikatsiya dannykh v sistemakh otslezhivaniya zadach s pomoshch'yu produktsionnykh pravil [Data verification in issue supervising systems by production rules]. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2013, no. 1 (83), pp. 86–90.
31. Shalyto A.A. *Switch-tekhnologiya. Algoritmizatsiya i programmirovanie zadach logicheskogo upravleniya* [Switch-technology. Algorithmization and programming the tasks of logical control]. St. Petersburg, Nauka Publ., 1998, 617 p.
32. Cardei I., Jha R., Cardei M., Pavan A. Hierarchical architecture for real-time adaptive resource management. *Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms.* Secaucus, NJ, USA, Springer-Verlag, 2000, pp. 415-434.
33. Polikarpova N.I., Shalyto A.A. *Avtomatnoe programmirovanie* [Automata-based programming]. St. Petersburg, Piter Publ., 2010, 176 p.
34. Dijkstra E.W. Guarded commands, non-determinacy and formal derivation of programs. *Communications of the ACM*, 1975, vol. 18, no. 8, pp. 453–457. Available at: http://www.cs.virginia.edu/~weimer/615/reading/DijkstraGC.pdf (accessed 25.11.2013).

Лукин Михаил Андреевич – программист, Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, Санкт-Петербург, Россия, lukinma@gmail.com

Michael Lukin – programmer, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics, Saint Petersburg, Russia, lukinma@gmail.com

УДК 004.65

АНАЛИЗ ДАННЫХ НА ОСНОВЕ ПЛАТФОРМЫ SQL-MAPREDUCE

А.А. Дергачев[a]

[a] Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, Санкт-Петербург, Россия, dam600@mail.ru

Рассмотрены проблемы, связанные с применением реляционных СУБД в области анализа больших объемов данных, в том числе данных, предоставляемых для аналитики посредством веб-сервисов в Интернет.
Возможность их решения может быть представлена веб-ориентированной распределенной системой анализа данных, исполнительным ядром которой является процессор сервисных запросов. Функции такой системы аналогичны функциям реляционных СУБД, только применительно к веб-сервисам. Процессор сервисных запросов необходим для формирования и исполнения плана вызова веб-сервисов анализа данных. Эффективность такой веб-ориентированной системы зависит от эффективности плана вызова веб-сервисов и программной реализации веб-сервисов, основным элементом которых являются средства хранения анализируемых данных – реляционные СУБД. Развитию возможностей реляционных СУБД для анализа больших объемов данных и уделено основное внимание в данной работе, а именно – оценке перспективности реализации веб-сервисов анализа данных на основе платформы SQL/MapReduce. Для достижения поставленной цели в качестве прикладной была выбрана аналитическая задача, характерная для различных социальных сетей и веб-порталов, связанная с анализом данных об их посещаемости различными пользователями. В рамках практической части исследования был реализован алгоритм формирования плана вызова веб-сервисов для решения прикладной аналитической задачи и выполнен эксперимент, подтверждающий эффективность технологии SQL/MapReduce и перспективность применения ее при реализации веб-сервисов анализа данных.
Ключевые слова: анализ данных, веб-сервисы, SQL, MapReduce, СУБД.

DATA ANALYSIS BY SQL-MAPREDUCE PLATFORM

A. Dergachev[b]

[b] Saint Petersburg National Research University of Information Technologies, Mechanics and Optics, Saint Petersburg, Russia, dam600@mail.ru

The paper deals with the problems related to the usage of relational database management system (RDBMS), mainly in the analysis of large data content, including data analysis based on web services in the Internet. A solution of these problems can be represented as a web-oriented distributed system of the data analysis with the processor of service requests as an executive kernel. The functions of such system are similar to the functions of relational DBMS, only with the usage of web services. The processor of service requests is responsible for planning of data analysis web services calls and their execution. The efficiency of such web-oriented system depends on the efficiency of web services calls plan and their program implementation where the basic element is the facilities of analyzed data storage – relational DBMS. The main attention is given to extension of functionality of relational DBMS for the analysis of large data content, in particular, the perspective estimation of web services data analysis implementation on the basis of SQL/MapReduce platform. With a view of obtaining this result, analytical task was chosen as an application-oriented part, typical for data analysis in various social networks and web portals, based on data analysis of users' attendance. In the practical part of this research the algorithm for planning of web services