5. *Нечаев Ю. И.* Нелинейная динамика и парадигмы вычислений при анализе экстремальных ситуаций // Мат. Междунар. науч. конф. „Леонард Эйлер и современная наука". СПб, 2007. С. 385—390.

6. *Хаяси Т.* Нелинейные колебания в физических системах. М.: Мир, 1968.

7. *Zadeh L.* Fuzzy logic, neural networks and soft computing // Commutation on the ASM-1994. Vol. 37, N 3. P. 77—84.

8. Интеллектуальные системы в морских исследованиях / Под ред. *Ю. И. Нечаева.* СПб: Изд-во СПбГМТУ, 2002. 320 с.

Сведения об авторе

Юрий Иванович Нечаев — д-р техн. наук, профессор; Санкт-Петербургский морской технический университет, кафедра вычислительной техники; E-mail: petr_oleg@mail.ru

Рекомендована институтом

Поступила в редакцию
10.03.09 г.

УДК 681.3

А. А. Шалыто, Е. А. Мандриков, Ю. К. Чеботарева

АВТОМАТНОЕ ПРОГРАММИРОВАНИЕ И ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Представлены основные положения автоматного программирования, обоснована предпочтительность его использования при разработке программного обеспечения. Описываются возможности применения параллельных вычислительных технологий для генерации автоматов.

Ключевые слова: *автоматное программирование, система управления, генетический алгоритм, фитнес-функция.*

Большинство программистов-практиков считают, что в программировании как таковом нет особых проблем. Видимое отсутствие проблем приводит к тому, что на практике при создании программного обеспечения (ПО) в большинстве случаев используются частные (от *лат.* ad hoc — экспромт, или спонтанное решение) подходы, основанные на персональном опыте программиста. Если трудности при создании программ и возникают, то их смиренно считают „неизбежным злом профессии" [1]. Тот факт, что при таком подходе достаточно много проектов заканчиваются неудачей, не изменяет точки зрения большинства.

Принципиально другое мнение у теоретиков программирования, которые еще в 1968 г. „открыто признали кризис программного обеспечения" [1]. В настоящее время возможным вариантом выхода из кризиса ряд теоретиков считают переход от „искусства программирования" [2] к программной инженерии (Software Engineering) [3, 4]. Однако современные специалисты по программной инженерии почти не используют подходы, разработанные в других инженерных областях.

Альтернативой инженерии является использование междисциплинарных исследований и подходов (Interdisciplinary Software Engineering Network — ISEN). В результате исследований в этом направлении сформировалось мнение, что при разработке ПО, видимо, может быть полезен опыт создания систем автоматического управления, что, в итоге, породило термин „программная кибернетика" (Software Cybernetics) [5].

Ниже излагаются основы автоматного программирования — междисциплинарного направления, относящегося как к программной инженерии, так и к программной кибернетике, которое базируется на положениях теории автоматов и теории автоматического управления. При этом особо подчеркнем, что под автоматным программированием авторы подразумевают

не программирование с применением автоматов, а соответствующую технологию программирования, направленную на создание систем со сложным поведением [6]. В этом смысле уместно провести параллель между автоматами и автоматным программированием, с одной стороны, и UML (Unified Modeling Language) и RUP (Rational Unified Process) — с другой. Так, автоматы и UML — это нотации, в то время как автоматное программирование и RUP — это процессы, использующие указанные нотации.

Парадигма автоматного программирования. Упрощенная трактовка автоматного программирования состоит в том, что это стиль программирования, при его использовании поведение программ предлагается описывать автоматами, которые в дальнейшем преобразуются в код [7]. В частности, в работе [8] такой подход был назван „программирование от состояний". Однако „программирование с автоматами" нельзя рассматривать как парадигму программирования, так как при этом остается не ясно, как с использованием автоматов проектировать и реализовывать программы в целом.

Вместе с тем автоматное программирование можно рассматривать как *новую парадигму* программирования. Очень многие системы являются автоматизированными объектами управления, представляющими собой совокупность системы управления (СУ) и объекта управления (ОУ), охваченных обратными связями (рис. 1, *а*).

Задача построения автоматизированных объектов управления рассматривается в любом курсе теории автоматического управления применительно к объектам различных типов. Удивительно, что это почти не коснулось практики программирования, несмотря на то что в теории алгоритмов в качестве одной из основных моделей используется машина Тьюринга (рис. 1, *б*).

Машина Тьюринга по сути является автоматизированным объектом управления, в котором система управления — конечный автомат (КА), а объект управления — лента (ее ячейки памяти) (рис. 1, *в*).

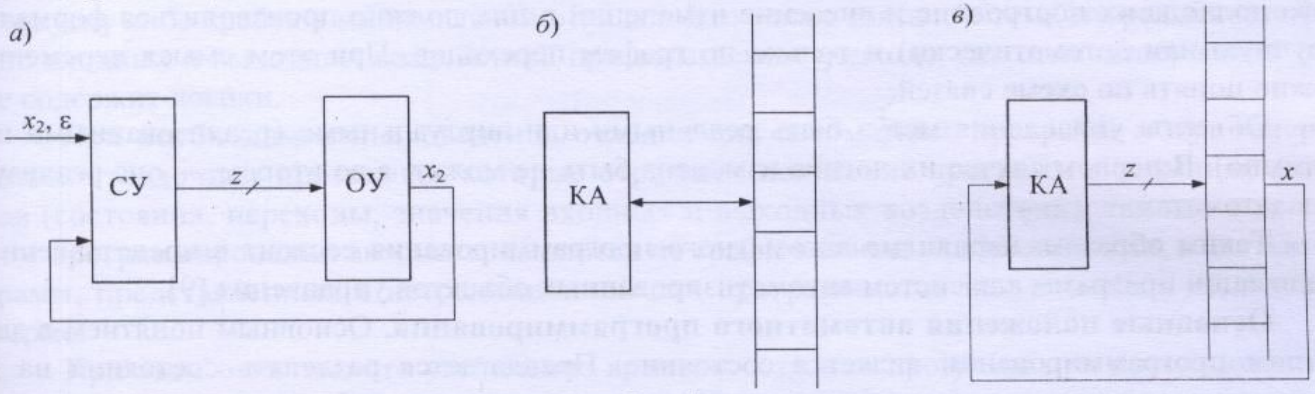

Рис. 1

Сложность программирования на машине Тьюринга определяется тем, что в ней используются очень простые объекты управления (ячейки памяти), которые могут выполнять только простейшие действия (операции) по сдвигу головки, записи и стиранию отдельных символов. В этой ситуации „вычисления" приходится выполнять конечному автомату, который для этой цели не приспособлен, так как его предназначение — управление. Другая особенность машины Тьюринга, которая может резко усложнять программы, — это использование только одного автомата, чего достаточно для проведения теоретических исследований, но бывает недостаточно при практическом применении.

Переход от тьюрингова программирования к практическому (автоматному) осуществляется за счет усложнения объектов управления, которые могут выполнять сколь угодно сложные действия (операции), и применения в качестве системы управления системы взаимодействующих автоматов. Из изложенного следует, что универсальность предлагаемого подхода определяется тем, что он основан на расширении машины Тьюринга, которая позволяет реализовать произвольные алгоритмы.

При этом теоретические положения о том, что автоматы позволяют распознавать регулярные языки, а магазинные автоматы — языки с контекстно-свободными грамматиками, отходят на второй план, так как в рассматриваемом подходе используются не автоматы, а автоматизированные объекты управления, в которых число объектов управления и их сложность не фиксированы, как и число автоматов.

Исходя из этого в автоматном программировании предлагается создавать программы по тем же принципам, как производится автоматизация технологических (и не только) процессов. При этом на основе анализа предметной области выделяются источники входных воздействий и автоматизированные объекты управления, каждый из которых содержит систему управления (систему взаимодействующих конечных автоматов) и объекты управления. Эти объекты реализуют выходные воздействия и формируют значения дополнительной разновидности входных воздействий, которые по обратным связям передаются системе управления.

Все составляющие каждого автоматизированного объекта управления отображаются на схеме связей, которая может совмещаться со схемой взаимодействия автоматов. На схеме связей для каждого входного и выходного воздействия указывается его полное название и краткое символьное обозначение, которое в дальнейшем и используется в качестве пометки в графах переходов автоматов и идентификатора соответствующей переменной в программе. Использование кратких символьных обозначений позволяет даже весьма сложные алгоритмы отражать так компактно, что часто граф переходов удается размещать на одном экране монитора, позволяя разработчику „охватить" весь граф одним взглядом, что облегчает понимание таких графов.

Использование символьных, а не лингвистических идентификаторов, являющихся обычно сокращенными английскими словами, смысл которых обычно забывается через некоторое время, не ухудшает восприятие автоматных программ, так как в рамках рассматриваемого подхода их построение и внесение изменений в них должно производиться формально (вручную или автоматически) и только по графам переходов. При этом смысл переменных можно понять по схеме связей.

Объекты управления могут быть реальными или виртуальными (реализованными программно). В первом случае их логика изменена быть не может, а во втором — она реализуется в автоматах.

Таким образом, парадигма автоматного программирования состоит в представлении и реализации программ как систем автоматизированных объектов управления [9].

Основные положения автоматного программирования. Основным понятием в автоматном программировании является состояние. Предлагается разделять состояния на два класса: управляющие и вычислительные. При этом с помощью небольшого числа управляющих состояний, как и в машине Тьюринга, можно управлять сколь угодно большим числом вычислительных состояний. Во введенной классификации управляющие состояния могут быть названы качественными, а вычислительные — количественными. В рамках автоматного программирования основное внимание уделяется управляющим состояниям, которые, если это не оговаривается особо, и рассматриваются в дальнейшем. При этом справедливо соотношение: „Состояния + входные воздействия = конечный автомат без выхода". Справедливо также: „Автомат без выхода + выходные воздействия = автомат".

Автоматы могут быть абстрактными (входные и выходные воздействия формируются последовательно) и структурными (входные и выходные воздействия формируются „параллельно"). В автоматном программировании, в отличие, например, от программирования компиляторов, обычно применяются структурные автоматы.

Понятие времени в автоматах в явном виде не используется. В случае применения элементы задержки рассматриваются как объекты управления. При этом задержки „запускаются" и

„сбрасываются“ из автоматов, а информация об истечении времени нахождения в состоянии поступает в них в виде входных воздействий.

Автоматы могут задаваться в различном виде, однако при проектировании и использовании они должны обладать когнитивными свойствами, что достигается при задании поведения автоматов в виде графов переходов (диаграмм состояний). В случае, если автоматы генерируются автоматически, они могут задаваться иначе — например, в табличной форме. При этом отметим, что даже при сравнительно небольшом числе состояний и переходов такое задание затрудняет понимание работы автоматов человеком, так как отражение переходов в них ненаглядно.

Понятность графов переходов достигается во многом за счет того, что состояния декомпозируют множество всех входных воздействий соответствующего автомата на группы, каждая из которых определяет переходы из рассматриваемого состояния.

Достоинства автоматного программирования. В рамках автоматного программирования предполагается, что собственно написание (генерация) программы начинается только после ее проектирования. При этом в инженерной практике (в отличие от традиционного программирования) проект или его этап обязательно завершается выпуском проектной документации. Поэтому при автоматном программировании, основной областью использования которого являются встроенные системы (чисто инженерная область), должна выпускаться проектная документация, а не только документация пользователя, как это обычно принято в программных проектах. При этом для автоматных программ необходимым компонентом, входящим в состав проектной документации, должны быть графы переходов.

Проектирование автоматов, описывающих логику программ, которая при традиционном программировании неупорядочена и поэтому сложна, а также формальный и изоморфный переход от автоматов к реализующим их программам, приводит к тому, что программы готовы к запуску либо требуют минимальной отладки. Это также связано с тем, что реализация функций входных и выходных воздействий при излагаемом подходе, как отмечалось выше, почти не содержит логики.

При необходимости проведения отладки для автоматных программ могут генерироваться отладочные протоколы, которые отражают поведение программ в терминах автоматов (состояния, переходы, значения входных и выходных воздействий), так как в автоматном программировании автоматы являются не графическими отображениями, а частью программ, представленных в нетрадиционной для программистов визуальной, а не текстовой форме.

При этом отметим, что увеличение времени создания программ при использовании автоматного подхода компенсируется сокращением времени их отладки. Это приводит к тому, что для большинства программ уровень трудоемкости разработки на основе автоматного и традиционного подходов практически одинаков. Однако в первом случае при разработке дополнительно получаются диаграммы, понятные человеку, по которым программа была построена формально, а во втором — только программа, понимание логики которой для представителей заказчика или даже для ее автора через некоторое время часто представляет большую проблему.

Создание программ со сложным поведением без использования диаграмм приводит к трудностям на всех этапах жизненного цикла. Особенно сложно в этом случае реализовывать программы, так как все особенности их поведения приходится „держать в голове“ в течение всего времени их написания, вместо того чтобы отобразить их на диаграмме и на время забыть. Еще одним преимуществом автоматных программ является простота внесения изменений в них специалистами в предметной области, не являющимися профессиональными программистами.

Следующее преимущество автоматного подхода — эффективность верификации автоматных программ на основе метода Model Checking (верификация на модели) [9, 10]. Это объясняется тем, что модель для верификации программ этого класса может строиться автоматически по графу переходов и иметь относительно небольшой размер, так как в графах переходов используются только управляющие состояния.

Для автоматных программ как класса отсутствует семантический разрыв между требованиями к программе и к модели, так как он устраняется в ходе разработки графов переходов на этапе проектирования. Это позволяет считать автоматные программы приспособленными к верификации. Таким образом, процесс верификации программ схож с контролем схем со сложной логикой — эти схемы не удается проверить, если они не спроектированы специальным образом для обеспечения „контролепригодности".

Автоматическая генерация автоматных программ. Основная трудоемкость построения автоматных программ связана с проектированием автоматов. Однако существуют задачи, про которые известно, что они могут быть решены с использованием автоматов, но эвристически построить автоматы в этих задачах крайне трудно или даже невозможно. Поэтому в данном случае допустимо использовать различные подходы эволюционного (динамического) программирования, в частности, генетические алгоритмы. Генетические алгоритмы — это методы оптимизации, базирующиеся на эволюции популяции особей, в процессе которой ведется поиск максимального значения целевой функции. Основной идеей генетических алгоритмов является использование принципа естественного отбора, заключающегося в том, что наиболее приспособленные особи дают потомство, формирующее следующее поколение. В среднем, следующее поколение является более приспособленным к „окружающей среде", чем предыдущее.

Генерация автоматов посредством генетических алгоритмов позволяет до 80 % кода автоматных программ строить почти автоматически, так как в программах этого класса объем кода, порождаемого автоматами, может достигать указанной величины.

В качестве практической реализации такого подхода рассмотрим человекоподобного робота *Nao* французской компании Aldebaran Robotics. Этот робот является официальной моделью лиги Standard Platform международных соревнований по футболу среди роботов [11]. В рамках международного проекта RoboCup в целях привлечения внимания ученых и разработчиков к проблемам искусственного интеллекта, робототехники и смежных областей ученым в форме соревнования предлагается решать стандартную задачу обучения робота игре в футбол. Участникам соревнования предлагается с помощью симулятора Webots написать программы для управления роботом. Готовые программы можно загружать на сайт, где периодически устраиваются соревнования.

Но даже с использованием парадигмы автоматного программирования создание конкурентоспособной управляющей программы для робота затруднительно без выделения набора базовых действий: шаг вперед, шаг назад, поворот налево, направо, удар по мячу, встать из положения лежа на спине и из положения лежа на животе. В то же время задать эти действия весьма непросто. Каждое действие состоит из некоторого числа фаз (в примерах описаний движений, поставляемых вместе с симулятором, от 10 до 100 и более), каждая из которых определяет значения углов сервомоторов робота. Подобрать требуемые значения вручную практически невозможно. Методы, основанные на знаниях о физике и биомеханике движений, сложны для реализации [12]. Поэтому предлагается использовать генетические алгоритмы для генерации базовых действий, равно как и для генерации автоматных программ, в которых полученные базовые действия будут интерпретированы как управляющие воздействия. В качестве особей генетического алгоритма будем рассматривать конечные автоматы. Представление конечных автоматов в виде хромосом и генетические операторы для работы с данными хромосомами подробно рассматриваются в работах [13—15].

Применение параллельных вычислений для генерации конечных автоматов. Применение генетических алгоритмов для автоматической генерации конечных автоматов является крайне ресурсоемкой процедурой. При этом основной проблемой в описанном подходе к решению задачи автоматического построения системы управления является реализация функции оценки особей (функции приспособленности, или фитнес-функции). Для большинства задач не составляет труда произвести имитацию автоматизированного объекта в виртуальной среде (в данном случае — в среде Webots) и по результатам его работы произвести оценку особи. Однако такая операция требует больших затрат вычислительных ресурсов и, как следствие — времени. Поэтому предлагается выполнять вычисление фитнес-функции параллельно в рамках модели распределенной памяти (рис. 2).

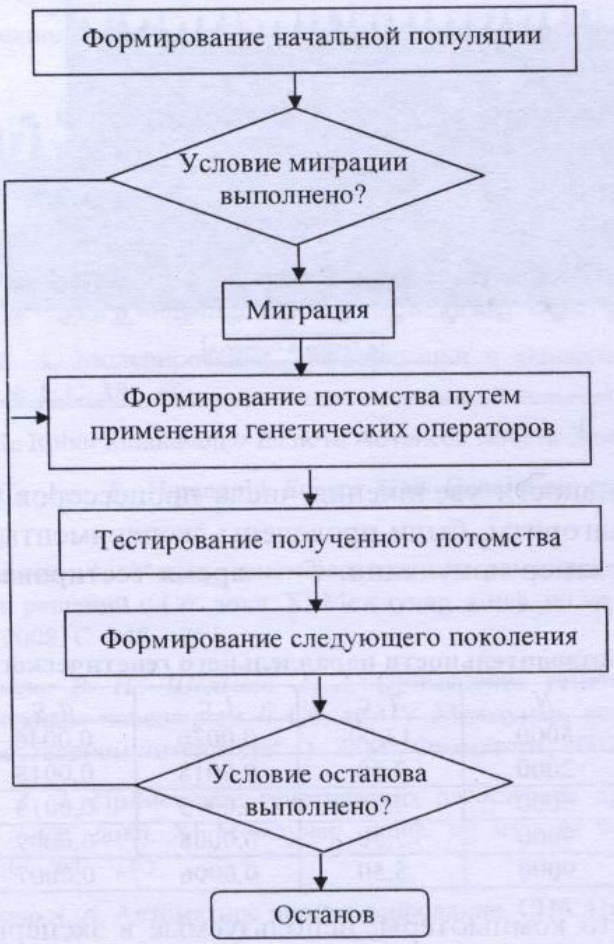

Рис. 2

Рассмотрим эффективность реализации параллельных вычислений фитнес-функции на примере задачи построения автомата, управляющего муравьем (задача „Умный муравей“ [13]). Эта задача состоит в определении оптимальной стратегии муравья, передвигающегося по игровому полю. Игровое поле представляет собой двумерный тор, в определенных клетках которого расположена „еда“. Муравей видит перед собой лишь одну клетку. В начале игры муравей находится в клетке с координатами (0, 0). За один ход может быть сделано одно из действий: шаг вперед, поворот направо, поворот налево. Если в клетке, в которую попадает муравей, есть „еда“, то она „съедается“. Цель игры — „съесть“ наибольшее количество еды за ограниченное число ходов. „Еда“ в процессе игры не возобновляется. Муравей „жив“ на протяжении игры — „еда“ не является необходимым ресурсом для его существования. Поведение муравья будем описывать конечным автоматом Мили.

На рис. 3 приведен график зависимости времени работы генетического алгоритма от номера итерации. Первые 128 итераций проводятся на одном компьютере (Intel Celeron M,

1,73 ГГц). После этого к вычислениям подключается второй компьютер (Intel Core 2, 2,50 ГГц), наблюдаемый на графике пик — момент его подключения.

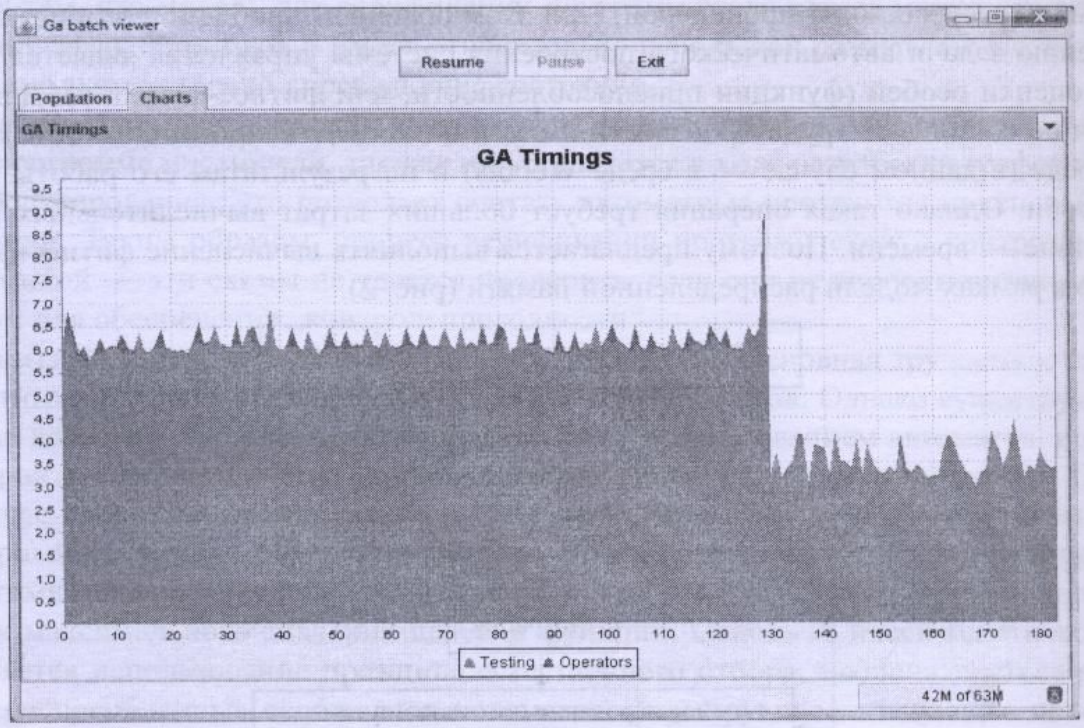

Рис. 3

Для оценки эффективности увеличения числа процессоров (p), реализующих рассматриваемый генетический алгоритм, были проведены эксперименты, результаты которых приведены в таблице (N — размер популяции, T — время тестирования популяции, t — время тестирования одной особи).

Анализ производительности параллельного генетического алгоритма

p	N	T, с	t, с	a, с	S_p
1	5000	13,00	0,0026	0,0046	1,00
3	2000	3,50	0,0018	0,0018	2,50
5	5000	7,25	0,0015	0,0013	3,57
10	9000	7,50	0,0008	0,0009	5,26
18	9000	5,50	0,0006	0,0007	6,67

Следует отметить, что компьютеры, используемые в экспериментах, имели различные характеристики и, как следствие — различную производительность. В таблице также приведены теоретические оценки времени тестирования особи (a) и ускорения (S_p) алгоритма за счет его параллельного исполнения.

В результате анализа полученных данных в соответствии с законом Амдала было установлено, что доля последовательных вычислений составляет всего 10 % от всех, производимых в рассматриваемом генетическом алгоритме. Отметим также, что существенное увеличение числа процессоров в системе увеличивает производительность незначительно. Более того, с определенного числа добавление новых процессоров в систему может увеличивать время тестирования популяции за счет затрат на передачу данных между ними.

Таким образом, совместное использование автоматного программирования [16], генетических алгоритмов и распределенных вычислений, по мнению авторов настоящей работы, является перспективным направлением разработок в области автоматизации процесса построения систем управления, в том числе и человекоподобными роботами.

СПИСОК ЛИТЕРАТУРЫ

1. *Дейкстра Э.* Смиренный программист // Лекции лауреатов премии Тьюринга за первые двадцать лет. 1966—1985. М.: Мир, 1993.

2. *Кнут Д.* Искусство программирования. Т. 1. Основные алгоритмы. М.: Вильямс, 2000.

3. Software Engineering. Germany: NATO Science Committee, 1968. [Electronic resource]: <http://www.europrog.ru/book/nato1968e.pdf>.

4. Software Engineering Techniques. Italy: NATO Science Committee, 1969. [Электронный ресурс]: <http://www.europrog.ru/book/nato1969e.pdf>.

5. *Cai K.-Yu., Chen T. Y., Tse T. H.* Towards Research on Software Cybernetics // Proc. of 7th IEEE Int. on High-assurance Systems Engineering (HASE 2002). Los Alamitos: IEEE Computer Society Press, 2002.

6. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб: Наука, 1998.

7. *Непейвода Н. Н.* Стили и методы программирования. М.: Интернет-Университет Информационных Технологий, 2005.

8. *Непейвода Н. Н., Скопин И. Н.* Основания программирования. М.—Ижевск: Институт компьютерных исследований, 2003.

9. *Шалыто А. А.* Автоматное программирование // Тез. докл. Междунар. науч. конф. памяти проф. А. М. Богомолова „Компьютерные науки и информационные технологии". Саратов: СГУ, 2007.

10. *Кузьмин Е. В., Соколов В. А.* Моделирование, спецификация и верификация „автоматных" программ // Программирование. 2008. № 1. С. 38—60.

11. *Michel O.* Professional Mobile Robot Simulation // Int. J. of Advanced Robotic Systems. 2004. Vol. 1, N 1. P. 39—42.

12. *Zhou C., Hu L., Acosta C., Yue P.* Humanoid Soccer Gait Generation and Optimization Using Probability Distribution Models, 2006.

13. *Данилов В. Р., Шалыто А. А.* Метод генетического программирования для генерации автоматов, представленных деревьями решений // Сб. докл. XI Междунар. конф. по мягким вычислениям и измерениям. СПб: СПбГЭТУ „ЛЭТИ", 2008. С. 248—251.

14. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для реализации систем со сложным поведением // Сб. тр. IV Междунар. конф. „Интегрированные модели и мягкие вычисления в искусственном интеллекте". Т. 2. М.: Физматлит, 2007. С. 598—604.

15. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для создания системы управления танком в игре Robocode // Сб. докл. XI Междунар. конф. по мягким вычислениям и измерениям. СПб: СПбГЭТУ „ЛЭТИ", 2008. С. 261—265.

16. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб: Питер, 2009.

Сведения об авторах

Анатолий Абрамович Шалыто — д-р техн. наук, профессор; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра технологии программирования; зав. кафедрой; E-mail: shalyto@mail.ifmo.ru

Евгений Андреевич Мандриков — студент; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра компьютерных технологий; E-mail: mandrikov@rain.ifmo.ru

Юлия Константиновна Чеботарева — студентка; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра компьютерных технологий; E-mail: chebj@rain.ifmo.ru

Рекомендована институтом

Поступила в редакцию
10.03.09 г.