ЭЛЕКТРИФИКАЦИЯ И АВТОМАТИЗАЦИЯ ПРОЦЕССОВ ГОРНОГО ПРОИЗВОДСТВА

УДК 004.05 DOI: 10.21440/0536-1028-2021-7-99-108

Формальное доказательство соответствия программной реализации используемых в системах контроля и управления моделей заданным требованиям

Лапин Э. С.[1], Абдрахманов М. И.[1]*
[1] Уральский государственный горный университет, г. Екатеринбург, Россия
e-mail: marat-ab@mail.ru

Реферат

Цель работы. *Исследование возможности применения метода формального доказательства соответствия программной реализации модели заданным требованиям для систем, модели которых могут быть представлены в виде конечных автоматов.*

Актуальность. *Разработка программного обеспечения для систем контроля и управления на одном из первых этапов этого процесса предполагает создание модели системы. Эта модель строится на основе технического задания, спецификации и различной априорной информации. Большая часть таких моделей для технических систем, которые эксплуатируются на современных добычных предприятиях (конвейерные системы, системы проветривания и т. п.), может быть описана с помощью модели конечного автомата. Такая модель может быть использована для решения широкого круга задач. Следующим шагом является программная реализация модели целиком либо какой-то ее части. В связи с этим возникает задача определения соответствия программной реализации модели ее исходному описанию.*

Результаты. *Один из способов решения данной задачи – это формальное доказательство наличия у программной модели свойств, которые определены в спецификации (описании) исходной модели. В статье на примере шахтной конвейерной системы показано применение метода, состоящего в программной реализации модели соответствующего конечного автомата, формировании предположений о наличии у модели свойств в виде теорем с их последующим доказательством с использованием специальных программных средств.*

Выводы. *Использование формальных методов для спецификации, разработки и верификации программных реализаций моделей систем в совокупности с другими методами позволит повысить качество и надежность разрабатываемых решений.*

Ключевые слова: *конечный автомат; формальное доказательство; тестирование; соответствие спецификации; корректность реализации модели; формализация требований; Coq; автоматизация доказательства теорем; конвейерная система шахты.*

Введение. На современных добычных предприятиях большая часть технических систем и технологических процессов может быть представлена в виде конечного автомата (далее КА) [1–3]. К ним относятся конвейерные системы, подъемные и вентиляционные установки, системы проветривания и дегазации, водоотлив. Особенность этих систем состоит в том, что в них можно выделить конечный набор состояний и определить правила перехода между ними. Далее в статье будет реализована идея формального доказательства соответствия программной реализации модели заданным требованиям для такого типа моделей на примере шахтной конвейерной системы.

Одним из этапов разработки программного обеспечения систем контроля и управления является создание модели системы [4, 5]. Эта модель может быть

представлена явно, в виде набора математических уравнений или программной реализации, либо неявно, в виде некоторого словесного описания.

Математическая модель технической системы может быть использована:

– для тестирования режимов работы системы;

– генерации программного обеспечения системы управления;

– проверки корректности работы системы в процессе ее функционирования;

– расследования аварийных ситуаций, возникших при эксплуатации системы.

Основой для построения модели системы является техническое задание и разного рода априорная информация о функционировании систем, подобных данной (физические законы, теоретические разработки, эмпирические формулы и т. п.).

После разработки модели системы возникает вопрос о ее корректности, т. е. насколько она адекватна рассматриваемому физическому процессу и(или) соответствует техническому заданию с описанием логики ее работы. Далее в статье будет рассматриваться второй аспект проверки корректности модели.

Определение корректности программной реализации модели. Так как для решения задач, обозначенных ранее, можно пользоваться только программной реализацией модели системы, а не ее формальным математическим представлением, то для определения корректности реализации предполагается использовать следующие подходы:

– тестирование (юнит-тесты, интеграционные тесты и т. д.) [6, 7];

– формальное доказательство того, что модель обладает свойствами, которые определяются спецификацией [8, 9].

Тестирование предполагает разработку наборов тестов, общий дизайн которых выглядит следующим образом:

$$A(x_i) == y_i,$$

где A – программная модель системы или ее компонента; x_i – входное значение, для которого известно, какой должна быть реакция модели системы; y_i – известная реакция модели системы на входное воздействие x_i; == – операция сравнения на равенство результата работы модели и ожидаемого результата.

Например, для модели вида $f(x) = x^2$ программная реализация на языке *Python* может выглядеть так:

```
f = lambda x: x**2
```

Для такой функции можно написать следующий тест:

```
def test_with_const():
    assert f(2) == 4
```

Проблема с доказательством через тестирование: в большинстве случаев оно не позволяет полностью проконтролировать соответствие программной реализации тому, что указано в спецификации. Несмотря на то что некоторые модели (например, конечные автоматы) в теории позволяют осуществить полный перебор всех возможных состояний и переходов, на практике это вызывает затруднения ввиду сложности разработки набора тестов, который бы обеспечил полное покрытие всех возможных вариантов. Следует заметить, что современные программные средства для организации тестирования позволяют автоматизировать многие аспекты этого процесса: перебор вариантов, генерацию случайных наборов данных и т. п. [10–12].

Второй подход проверки корректности реализации модели – это формальное доказательство ее свойств, определенных в спецификации [13–18]. Основное ограничение данного решения состоит в том, что оно в большей степени применимо для моделей, разработанных на функциональных языках программирования. Для других языков это теоретически возможно, но тяжело осуществимо на практике. Выполнение программы, написанной на функциональном языке, по сути представляет собой вычисление математического выражения. Это позволяет формально доказывать свойства такой программы. Разработка алгоритма функционирования модели и описание свойств (требований) могут быть выполнены в соответствии с техническим заданием на систему (модель).

Приведенное далее решение основано на общем методе доказательства, который использует некоторый базовый набор исходных утверждений (аксиом) и формальные правила вывода [19–21].

Формализация требований к системе в соответствии со спецификацией. В качестве примера рассмотрим модель работы шахтного конвейера, которая подробно описана в работе [22]. Далее представлено краткое описание модели.

Конвейер может находиться в состояниях: остановлен (*stop*), готов к запуску (*ready*), запущен (*start*) и заблокирован (*block*):

$$ConvState = \{stop, ready, start, block\}$$

На работу моделируемого конвейера оказывают влияние следующие сигналы: сигналы с датчиков контроля схода ленты (*ksl*) и экстренное ограждение (*eo*), сигнал аварии (*avs*), готов к запуску (*ps*), кнопки: стоп (*bsp*), пуск (*bst*) и авария (*bav*). Входной сигнал модели — это кортеж, состоящий из всех перечисленных сигналов. Каждый сигнал подается на дискретный вход контроллера и может находиться в четырех состояниях: замкнут, разомкнут, короткое замыкание, обрыв.

Для упрощения модели выделим два состояния (*zm* – замкнут, *rz* – разомкнут):

$$InSignal = \{zm, rz\}.$$

Наборы всех возможных состояний перечисленных ранее сигналов составляют множество входных сигналов.

Функцию, которая определяет состояние системы по перечисленному набору сигналов, будем обозначать *calcState*:

$$y = calcState(x, z_1, ..., z_i, ..., z_7),$$

где x – текущее состояние системы, $x \in ConvState$; z_i – входные сигналы системы, $z_i \in InSignal$; y – состояние конвейера, $y \in ConvState$.

На основе технического задания (спецификации) и правил функционирования конвейерной системы определим ее свойства, на базе которых будет разработана программная реализация модели. Представим свойства в словесной и математической формулировках.

Свойство 1. Если подан хотя бы один аварийный сигнал, то система переходит в состояние БЛОКИРОВКА:

$$(avs = zm \vee bav = zm), \forall ksl, eo, bsp, bs, bst, state:$$
$$calcState(state, avs, bav, ksl, eo, bsp, ps, bst) = block,$$
$$avs, bav, ksl, eo, bsp, ps, bst \in InSignal, state \in ConvState$$

Свойство 2. Если аварийный сигнал не подан и сработал хотя бы один датчик, нажата кнопка «Стоп» или нет сигнала «Готов к запуску», то система переходит в состояние ОСТАНОВКА:

$$(avs = rz \wedge bav = rz) \wedge (ksl = zm \vee eo = zm \vee bsp = zm \vee ps = rz), \forall state:$$
$$calcState(state, avs, bav, ksl, eo, bsp, ps, bst) = stop,$$
$$avs, bav, ksl, eo, bsp, ps, bst \in InSignal, state \in ConvState$$

Свойство 3. Если аварийный сигнал не подан, нет сработавших датчиков, есть сигнал «Готов к запуску» и система находится в состоянии ОСТАНОВЛЕНА, то она переходит в состояние ГОТОВ К ЗАПУСКУ:

$$(avs = rz \wedge bav = rz \wedge ksl = rz \wedge eo = rz \wedge bsp = rz \wedge ps = zm \wedge bst = rz \wedge state = stop),$$
$$calcState(state, avs, bav, ksl, eo, bsp, ps, bst) = ready,$$
$$avs, bav, ksl, eo, bsp, ps, bst \in InSignal, state \in ConvState$$

Свойство 4. Если система находится в состоянии ГОТОВ К ЗАПУСКУ и нажата кнопка «Пуск», то система переходит в состояние ЗАПУЩЕНА:

$$(avs = rz \wedge bav = rz \wedge ksl = rz \wedge eo = rz \wedge bsp = rz \wedge ps = zm \wedge bst = zm \wedge state = ready),$$
$$calcState(state, avs, bav, ksl, eo, bsp, ps, bst) = start,$$
$$avs, bav, ksl, eo, bsp, ps, bst \in InSignal, state \in ConvState$$

Формальное доказательство того, что для программной реализации модели выполняются указанные требования. Для формального доказательства свойств системы воспользуемся *Coq* – интерактивным программным средством доказательства теорем, которое использует собственный язык функционального программирования с зависимыми типами [23–26]. *Coq* позволяет записывать математические теоремы и их доказательства, модифицировать и проверять их на правильность.

Создадим набор необходимых типов данных:

– состояние конвейера:

```
Inductive ConvState: Type :=
  | stop
  | start
  | ready
  | block.
```

– дискретный сигнал:

```
Inductive InSignal: Type :=
  | rz
  | zm.
```

– вектор состояний входных сигналов системы:

```
Inductive InVectorSignal: Type :=
  | sg (avs bav ksl eo bsp ps bst: InSignal).
```

Построим вспомогательные функции, которые понадобятся для реализации общего алгоритма работы конвейера.

Функция проверки того, что указанный набор сигналов переводит систему в состояние:

– БЛОКИРОВКА:

```
Definition isAlarm (inSigs: InVectorSignal): bool :=
  match inSigs with
    | (sg rz rz _ _ _ _ _) => false
    | _ => true
  end.
```

– ОСТАНОВКА:

```
Definition isStop (inSigs: InVectorSignal): bool :=
  match inSigs with
  | (sg _ _ rz rz rz zm _) => false
  | _ => true
  end.
```

– ГОТОВ К ЗАПУСКУ:

```
Definition isReady (inSigs: InVectorSignal): bool :=
  match inSigs with
    | (sg rz rz rz rz rz zm rz) => true
    | _ => false
  end.
```

– ЗАПУЩЕНА:

```
Definition isStart (inSigs: InVectorSignal): bool :=
  match inSigs with
    | (sg rz rz rz rz rz zm zm) => true
    | _ => false
  end.
```

Используя указанные функции, представим программную реализацию модели системы, которая учитывает текущее состояние системы и вектор входных сигналов:

```
Definition calcState (state: ConvState) (inSigs: InVectorSignal): ConvState :=
  match isAlarm inSigs with
  | true => block
  | false => match isStop inSigs with
        | true => stop
        | false => match isStart inSigs with
              | true => start
              | false => match isReady inSigs, state with
                    | true, start => start
                    | true, _ => ready
                    | false, _ => stop
```

```
                    end
                end
            end
        end.
```

Сформулируем свойства системы в виде теорем на языке *Coq* и докажем их.

Доказательство свойства 1.

```
Theorem alarmState: forall (avs bav ksl eo bsp ps bst: InSignal),
    forall (state: ConvState),
    avs = zm \/ bav = zm
    -> calcState state (sg avs bav ksl eo bsp ps bst) = block.
Proof.
    intros avs bav ksl eo bsp ps bst state.
    intros H.
    destruct H as [Havs | Hbav].
    - rewrite -> Havs. reflexivity.
    - rewrite -> Hbav. destruct avs.
        + reflexivity.
        + reflexivity.
Qed.
```

Доказательство свойства 2.

```
Theorem stopState: forall (avs bav ksl eo bsp ps bst: InSignal),
    forall (state: ConvState),
    (avs = rz /\ bav = rz) /\
    (ksl = zm \/ eo = zm \/ bsp = zm \/ ps = rz)
    -> calcState state (sg avs bav ksl eo bsp ps bst) = stop.
Proof.
    intros avs bav ksl eo bsp ps bst state. intros [[Havs Hbav] H].
    rewrite -> Havs. rewrite -> Hbav.
    destruct H. rewrite -> H. reflexivity.
    destruct H. rewrite -> H. destruct ksl.
        reflexivity. reflexivity.
    destruct H. rewrite -> H. destruct ksl. destruct eo.
        reflexivity. reflexivity. reflexivity.
    rewrite -> H. destruct ksl. destruct eo. destruct bsp.
        reflexivity. reflexivity. reflexivity. reflexivity.
Qed.
```

Доказательство свойства 3.

```
Theorem readyState: forall (avs bav ksl eo bsp ps bst: InSignal),
    forall (state: ConvState),
    avs = rz /\ bav = rz /\ ksl = rz /\ eo = rz /\ bsp = rz /\
    ps = zm /\ bst = rz /\ state = stop
    -> calcState state (sg avs bav ksl eo bsp ps bst) = ready.
Proof.
    intros avs bav ksl eo bsp ps bst state H0.
    destruct H0 as [Havs H1]. rewrite -> Havs.
```

```
    destruct H1 as [Hbav H2]. rewrite -> Hbav.
    destruct H2 as [Hksl H3]. rewrite -> Hksl.
    destruct H3 as [Heo H4]. rewrite -> Heo.
    destruct H4 as [Hbsp H5]. rewrite -> Hbsp.
    destruct H5 as [Hps H6]. rewrite -> Hps.
    destruct H6 as [Hbst Hstop]. rewrite -> Hbst. rewrite -> Hstop.
    reflexivity.
Qed.
```

Доказательство свойства 4.

```
Theorem startState: forall (avs bav ksl eo bsp ps bst: InSignal),
    forall (state: ConvState),
    avs = rz /\ bav = rz /\ ksl = rz /\ eo = rz /\ bsp = rz /\
    ps = zm /\ bst = zm /\ state = ready
    -> calcState state (sg avs bav ksl eo bsp ps bst) = start.
Proof.
    intros avs bav ksl eo bsp ps bst state H0.
    destruct H0 as [Havs H1]. rewrite -> Havs.
    destruct H1 as [Hbav H2]. rewrite -> Hbav.
    destruct H2 as [Hksl H3]. rewrite -> Hksl.
    destruct H3 as [Heo H4]. rewrite -> Heo.
    destruct H4 as [Hbsp H5]. rewrite -> Hbsp.
    destruct H5 as [Hps H6]. rewrite -> Hps.
    destruct H6 as [Hbst Hready]. rewrite -> Hbst.
    rewrite -> Hready. reflexivity.
Qed.
```

Вывод. Таким образом, формально доказано, что построенная программная модель работы конвейера соответствует заданным требованиям. Модель системы – это функция *calcState*, ее можно преобразовать в программный код на языках *Haskell, Ocaml* (в настоящее время существуют разработки трансляции кода на языке *Coq* в код на языке *C*, например https://github.com/akr/codegen). Это позволяет создавать программные модули с формально доказанными свойствами, которые в дальнейшем могут быть использованы как компоненты более крупных систем, решающих задачи, перечисленные в начале статьи. Отметим, что предлагаемая в статье методика может использоваться для оценки адекватности модели и моделируемого объекта, при этом результат определяется лишь полнотой и правильностью формулирования самих требований.

В начале статьи были перечислены системы, эксплуатируемые на добычных предприятиях, для моделирования которых хорошо подходят КА. Помимо них есть и другие: геомониторинг [27, 28]; системы, в основе которых лежат аналоговые модели (например, регулирование) и т. п. Несмотря на то что основные аспекты работы таких систем не могут быть описаны с помощью КА, ряд задач, такие как автоматическая блокировка, защита, определение состояния системы, решается через модель КА.

Представленный в статье метод не может использоваться как единственное и универсальное средство определения корректности программного обеспечения. Он должен применяться совместно с другими методами. Также необходимо помнить, что наличие доказательств свойств системы не делает ее автоматически корректной, так как их состав может быть неполным, а доказываемые положения (леммы и теоремы) могут содержать ошибки в своих формулировках.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Блюмин С. Л., Корнеев А. М. Дискретное моделирование систем автоматизации и управления: монография. Липецк: ЛЭГИ, 2005. 124 с.

2. Ezhilarasu U. P., Mahapatra R. P., Karthick S. Finite automata problems and solutions. LAP LAMBERT Academic Publishing, 2019. 588 p.

3. Baranov Samary. Finite state machines and algorithmic state machines: fast and simple design of complex finite state machines. ISBN Canada, 2018. 185 p.

4. Эванс Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем.: пер. с англ. СПб: Диалектика, 2019. 448 с.

5. Ларман К. Применение UML 2.0 и шаблонов проектирования: практическое руководство: пер. с англ. М.: И. Д. Вильямс, 2013. 736 с.

6. Myers Glenford J. The art of software testing. John Wiley & Sons, Inc., Hoboken, New Jersey, 2012. 240 p.

7. Старолетов С. М. Основы тестирования и верификации программного обеспечения. СПб: Лань, 2020. 344 с.

8. Камкин А. С. Введение в формальные методы верификации программ. М.: МАКС Пресс, 2018. 272 с.

9. Monin Jean-Francois. Understanding formal methods. London–New York: Springer, 2003. 275 p.

10. Никешин А. В., Пакулин Н. В., Шнитман В. З. Мутационное тестирование сетевых протоколов с использованием формальных моделей // Научный сервис в сети Интернет: труды XVII Всероссийской научной конференции / ИПМ им. М. В. Келдыша. М., 2015. С. 259–266.

11. Татарников А. Д. Обзор методов и средств генерации тестовых программ для микропроцессоров // Труды ИСП РАН. 2017. Т. 29. Вып. 1. С. 167–194.

12. Камкин А. С., Сергеева Т. И., Смолов С. А., Татарников А. Д., Чупилко М. М. Расширяемая среда генерации тестовых программ для микропроцессоров // Программирование. 2014. № 1. С. 3–14.

13. Гратинский В. А., Новиков Е. М., Захаров И. С. Экспертная оценка результатов верификации инструментов верификации моделей программ // Труды ИСП РАН. 2020. Т. 32. Вып. 5. С. 7–20.

14. Самонов А. В., Самонова Г. Н. Методика и средства разработки и верификации формальных fUML моделей требований и архитектуры сложных программно-технических систем // Труды ИСП РАН. 2018. Т. 30. Вып. 5. С. 123–146.

15. Boulanger J.-L. Industrial use of formal methods: formal verification. Wiley-ISTE, 2012. 298 p.

16. Cofer D. D. et al. Compositional verification of architectural models // NASA Formal Methods: 4th International Symposium, NFM 2012, Proceedings. Norfolk, P. 126–140.

17. Gnesi S., Margaria T. Formal methods for industrial critical systems: a survey of applications. Wiley-IEEE Press, 2013.

18. Jackson D. Software abstractions: logic, language and analysis. MIT press, 2012.

19. Колмогоров А. Н., Драгалин А. Г. Математическая логика. Введение в математическую логику. М.: ЛАНАНД, 2017. 240 с.

20. Клини С. Введение в метаматематику. М.: Либроком, 2009. 523 с.

21. Карри Х. Б. Основания математической логики. М.: Мир, 1969. 567 с.

22. Лапин Э. С., Абдрахманов М. И. Функциональный подход к моделированию динамики систем детерминированных конечных автоматов // Известия вузов. Горный журнал. 2021. № 2. С. 113–121.

23. Benjamin C. Pierce. Logical foundations. URL: https://softwarefoundations.cis.upenn.edu/lf-current/toc.html (дата обращения: 02.04.2021).

24. Chlipala Adam. Certified programming with dependent types. MIT Press Ltd, 2014. 424 p.

25. Ilya Sergey. Programs and proofs. URL: https://ilyasergey.net/pnp/ (дата обращения: 02.04.2021).

26. Bertot Yves, Castéran Pierre. Interactive theorem proving and program development: Coq art: The calculus of inductive constructions. Springer, 2004. 500 p.

27. Pisetski V. B., Kornilkov S. V., Sashurin A. D., Lapin E. S., Lapin S. E., Balakin V. Y. Concept, system solutions and the results of geotechnical monitoring and forecasting of hazardous geodynamic phenomena in the processes of underground mining // European Association of Geoscientists & Engineers: conference proceedings, 13th Conference and Exhibition Engineering Geophysics 2017, April 2017. Vol. 2017. P. 1–4.

28. Pisetski V. B., Huang R., Patrushev Y. V., Zudilin A. E., Schneider I. V., Shirobokov M. P., Balakin V. Y. The test results of the seismic monitoring systems state of stability of the rock mass in the process of construction of road tunnels // China European Association of Geoscientists & Engineers: conference proceedings, 13th Conference and Exhibition Engineering Geophysics 2017, April 2017. Vol. 2017. P. 1–7.

Поступила в редакцию 4 октября 2021 года

Сведения об авторах:

Лапин Эдуард Самуилович – доктор технических наук, профессор, профессор кафедры автоматики и компьютерных технологий Уральского государственного горного университета. E-mail: lapin.eduard2014@yandex.ru; https://orcid.org/0000-0001-9710-8111

Абдрахманов Марат Ильдусович – кандидат технических наук, преподаватель кафедры автоматики и компьютерных технологий Уральского государственного горного университета. E-mail: marat-ab@mail.ru; https://orcid.org/0000-0002-0391-6204

DOI: 10.21440/0536-1028-2021-7-99-108

Formally proving whether the software implementation of models applied in instrumentation and control systems conform to the specified requirements

Eduard S. Lapin[1], Marat I. Abdrakhmanov[1]
[1] Ural State Mining University, Ekaterinburg, Russia.

Abstract

Research objective *is to study the possibility to formally validate whether the model's software implementation meets all the specified requirements of the systems, the model of which can be represented in the form of finite-state automata.*

Research relevance. *At one of the first stages, the development of software for instrumentation and control systems provides for the creation of the system model. The model is based on the terms of reference, specification, and various a priori information. Most of the models for engineering systems in the modern mining industry (conveyor systems, ventilation systems, etc.) can be described in terms of the finite state automaton model. Such a model can be applied to solve diverse tasks. The next step is to implement the model in whole or in part. In this context, the task arises to determine the model's software implementation conformity to its initial description.*

Results. *One way to solve the task is to formally prove that the software model possesses the properties which are provided in the specification (description) of the initial model. By the example of the mine conveyor system, the paper illustrates the application of the method which consists in the software implementation of the corresponding finite-state automaton model, forecasting whether the model possesses the properties through theorems and their subsequent proof by applying special software.*

Conclusions. *Formal methods of specification, development, and verification of system models' software implementation together with other methods make it possible to improve the quality and reliability of solutions under development.*

Keywords: *finite-state automaton; formal proof; testing; conformity to specification; model implementation correctness; requirements formalization; Coq; automated theorem proving; mine conveyor system.*

REFERENCES

1. Bliumin S. L., Korneev A. M. *Discrete modeling of automation and control systems: monograph*. Lipetsk: Lipetsk Institute for Ecology and Humanities Publishing; 2005. (In Russ.)

2. Ezhilarasu U. P., Mahapatra R. P., Karthick S. *Finite automata problems and solutions*. LAP LAMBERT Academic Publishing, 2019. 588 p.

3. Baranov Samary. *Finite state machines and algorithmic state machines: fast and simple design of complex finite state machines*. ISBN Canada, 2018. 185 p.

4. Evans E. *Domain driven design. Tackling complexity in the heart of software. Translation from English*. St. Petersburg: Dialektika Publishing; 2019. (In Russ.)

5. Larman K. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development. Translation from English*. Moscow: I. D. Viliams Publishing; 2013. (In Russ.)

6. Myers Glenford J. *The art of software testing*. John Wiley & Sons, Inc., Hoboken, New Jersey; 2012. 240 p.

7. Staroletov S. M. *The fundamentals of software verification and testing*. St. Petersburg: Lan; 2020. (In Russ.)

8. Kamkin A. S. *Introduction to formal methods of software verification*. Moscow: MAKS Press Publishing; 2018. (In Russ.)

9. Monin Jean-Francois. *Understanding formal methods*. London–New York: Springer; 2003. 275 p.

10. Nikeshin A. V., Pakulin N. V., Shnitman V. Z. Mutation testing of network protocols with formal models. In: *Scientific service on the Internet: Proceedings of 17th All-Russian Conference*. Moscow: KIAM Publishing; 2015. p. 259–266. (In Russ.)

11. Tatarnikov A. D. A survey of methods and tools for test program generation for microprocessors. *Trudy ISP RAN = Proceedings of ISP RAS*. 2017; 29(1): 167–194. (In Russ.)

12. Kamkin A. S., Sergeeva T. I., Smolov S. A., Tatarnikov A. D., Chupilko M. M. Extendible environment for test programs generation for microprocessors. *Programmirovanie = Programming and Computer Software*. 2014; 1: 3–14. (In Russ.)

13. Gratinskii V. A., Novikov E. M., Zakharov I. S. Expert assessment of verification tool results. *Trudy ISP RAN = Proceedings of ISP RAS*. 2020; 32(5): 7–20. (In Russ.)

14. Samonov A. V., Samonova G. N. Methodology and tools for development and verification of formal fUML models of requirements and architecture for complex software and hardware systems. *Trudy ISP RAN = Proceedings of ISP RAS*. 2018; 30(5): 123–146. (In Russ.)

15. Boulanger J.-L. *Industrial use of formal methods: formal verification*. Wiley-ISTE, 2012. 298 p.

16. Cofer D. D. et al. Compositional verification of architectural models. In: *NASA Formal Methods: 4th International Symposium, NFM 2012, Proceedings*. Norfolk, P. 126–140.

17. Gnesi S., Margaria T. *Formal methods for industrial critical systems: a survey of applications*. Wiley-IEEE Press, 2013.

18. Jackson D. *Software abstractions: logic, language and analysis*. MIT press, 2012.
19. Kolmogorov A. N., Dragalin A. G. *Mathematical logic. Introduction to mathematical logic*. Moscow: LANAND Publishing; 2017. (In Russ.)
20. Kleene S. C. *Introduction to metamathematics. Translation from English*. Moscow: Librokom Publishing; 2009. (In Russ.)
21. Curry H. B. *Foundation of mathematical logic*. Moscow: Mir Publishing; 1969. (In Russ.)
22. Lapin E. S., Abdrakhmanov M. I. Functional approach to deterministic finite-state automata systems dynamic modeling. *Izvestiya vysshikh uchebnykh zavedenii. Gornyi zhurnal = News of the Higher Institutions. Mining Journal*. 2021; 2; 113–121. (In Russ.)
23. Benjamin C. Pierce. *Logical Foundations*. Available from: https://softwarefoundations.cis.upenn.edu/lf-current/toc.html [Accessed: 02 April 2021].
24. Chlipala Adam. *Certified programming with dependent types*: MIT Press Ltd, 2014. 424 p.
25. Ilya Sergey. *Programs and proofs*. Available from: https://ilyasergey.net/pnp [Accessed: 02 April 2021].
26. Bertot Yves, Castéran Pierre. *Interactive theorem proving and program development: Coq art: The calculus of inductive constructions*. Springer, 2004. 500 p.
27. Pisetski V. B., Kornilkov S. V., Sashurin A. D., Lapin E. S., Lapin S. E., Balakin V. Y. Concept, system solutions and the results of geotechnical monitoring and forecasting of hazardous geodynamic phenomena in the processes of underground mining. In: *European Association of Geoscientists & Engineers: conference proceedings, 13th Conference and Exhibition Engineering Geophysics 2017, April 2017*. Vol. 2017. p. 1–4.
28. Pisetski V. B., Huang R., Patrushev Y. V., Zudilin A. E., Schneider I. V., Shirobokov M. P., Balakin V. Y. The test results of the seismic monitoring systems state of stability of the rock mass in the process of construction of road tunnels. In: *China European Association of Geoscientists & Engineers: conference proceedings, 13th Conference and Exhibition Engineering Geophysics 2017, April 2017*. Vol. 2017. p. 1–7.

Received 4 October 2021

Information about authors:

Eduard S. Lapin – DSc (Engineering), Professor, professor of the Department of Automation and Computer Technologies, Ural State Mining University. E-mail: lapin.eduard2014@yandex.ru; https://orcid.org/0000-0001-9710-8111
Marat I. Abdrakhmanov – PhD (Engineering), lecturer, Department of Automation and Computer Technologies, Ural State Mining University. E-mail: marat-ab@mail.ru; https://orcid.org/0000-0002-0391-6204

Для цитирования: Лапин Э. С., Абдрахманов М. И. Формальное доказательство соответствия программной реализации используемых в системах контроля и управления моделей заданным требованиям // Известия вузов. Горный журнал. 2021. № 7. С. 99–108. DOI: 10.21440/0536-1028-2021-7-99-108
For citation: Lapin E. S., Abdrakhmanov M. I. Formally proving whether the software implementation of models applied in instrumentation and control systems conform to the specified requirements. *Izvestiya vysshikh uchebnykh zavedenii. Gornyi zhurnal = News of the Higher Institutions. Mining Journal*. 2021; 7: 99–108 (In Russ.). DOI: 10.21440/0536-1028-2021-7-99-108