УДК 621.396
doi:10.31799/1684-8853-2026-5-21-36
EDN: IBKLIO

Научные статьи
Articles

Валидация автоматных спецификаций на основе перемножения автоматов

В. А. Назаров[a], *бакалавр, orcid.org/0009-0000-0352-7885*
А. А. Шалыто[a], *доктор техн. наук, профессор, orcid.org/0000-0002-2723-2077, anatoly.shalyto@gmail.com*
[a]*Университет ИТМО, Кронверкский пр., 49, Санкт-Петербург, 197101, РФ*

Введение: *при разработке ответственных систем все чаще применяются автоматные спецификации. При этом особую важность и актуальность представляет их валидация. Тестирование и верификация не решают эту проблему, особенно при использовании вложенных конечных автоматов.* ***Цель:*** *разработать метод построения автомата-произведения для рассматриваемого класса спецификаций на основе перемножения графов переходов, в том числе для вложенных автоматов, для упрощения эвристической процедуры его последующей валидации.* ***Методы:*** *в работе применяется построение достижимой части пространства состояний, обход графа в ширину и имитационная проверка выходных последовательностей событий.* ***Результаты:*** *предлагаемый метод формирует автомат-произведение с множеством достижимых пар состояний и отображением переходов, что позволяет при визуальном анализе обнаружить и вручную исключить тупиковые (дедлоки) и другие некорректные состояния и переходы. Последовательное перемножение нескольких автоматов обеспечивает пошаговую проверку корректности всей системы. В рассмотренном примере расхождений выходных последовательностей событий между откорректированной системой и автоматом-произведением на использованных входных последовательностях событий не обнаружено.* ***Практическая значимость:*** *представленный метод позволяет расширить арсенал средств для обеспечения валидации автоматных спецификаций, содержащих, в том числе, вложенные автоматы, однако не гарантирует упрощения автомата-произведения по сравнению с исходной системой.* ***Обсуждение:*** *метод особенно полезен при анализе автоматически построенных или модифицированных систем автоматов, поскольку помогает обнаруживать скрытые логические противоречия, которые трудно выявить тестированием, верификацией или имитационным моделированием.*

Ключевые слова *— автоматы, вложенные автоматы, перемножение автоматов, валидация.*

Для цитирования: Назаров В. А., Шалыто А. А. Валидация автоматных спецификаций на основе перемножения автоматов. *Информационно-управляющие системы*, 2026, № 5, с. 21–36. doi:10.31799/1684-8853-2026-5-21-36, EDN: IBKLIO
For citation: Nazarov V. A., Shalyto A. A. Validation of automata specifications based on automata composition. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2026, no. 5, pp. 21–36 (In Russian). doi:10.31799/1684-8853-2026-5-21-36, EDN: IBKLIO

Введение

Программное обеспечение (ПО) для управления ответственными объектами обычно представляет собой системы со сложным поведением, зависящим от предыстории [1]. Цена ошибки в таких системах может быть очень высока, поэтому к ним применяют строгие методы формальной верификации [2, 3]. Одним из подходов к созданию ПО для таких объектов является автоматное программирование.

В рамках парадигмы автоматного программирования [1] ключевыми компонентами программы считаются конечные автоматы, совокупность которых представляет собой спецификацию и ее поведение. Среди достоинств конечных автоматов можно отметить наглядность их представления в виде графов переходов, удобство и естественность их верификации [4], которая может дополнять тестирование, применяемое обычно при отладке программ. По мнению Л. Лэмпорта, код не предназначен для размышлений о поведении системы, графы переходов для этой цели подходят значительно лучше. При использовании инструментальных средств, таких как Stateflow [5], графы переходов являются также и сверхвысоким языком программирования. Современные работы продолжают развивать формальную верификацию таких моделей, в том числе Stateflow [6] и автоматных спецификаций на основе теории автоматов [7]. Поэтому автоматные спецификации все шире используются на практике, однако и они требуют валидации.

Как в свое время заметил Э. Дейкстра, отсутствие ошибок при тестировании не гарантирует их отсутствия. Верификация также не дает полной гарантии: известны случаи, когда автоматная спецификация успешно проходила значительное число тестов и проверку многих темпоральных свойств, однако ошибка обнаруживалась лишь при добавлении дополнительного свойства. Таким образом, ни число тестов, ни число проверяемых свойств заранее не известны, а тестирование и верификацию следует рассматривать как составляющие валидации [8] наряду с имитационным моделированием и эвристическими проверками. Валидации автоматных и statechart-спецификаций посвящены и современные работы, например [9].

Особое значение валидация приобретает при использовании для управления ответственными объектами систем взаимосвязанных автоматов, в том числе вложенных, каждый из которых задается графом переходов. Корректность таких систем исследуют методами проверки на моделях и автоматного анализа [10, 11], в том числе для систем взаимодействующих конечных автоматов [12, 13]. В настоящей работе предлагается новый подход к валидации таких графов переходов, состоящий в их замене одним эквивалентным графом. Метод позволяет обнаруживать дефекты в исходной системе, в том числе тупиковые достижимые состояния. Подход, который при наличии более двух автоматов применяется многократно, состоит из двух стадий: построения эквивалентного графа для двух автоматов и последующего эвристического анализа полученного результата.

Известно, что задача построения одного автомата по системе автоматов может быть решена их перемножением. В работе «Автоматное программирование: определение, модель, реализация» (https://habr.com/ru/articles/682422/) рассматривается перемножение схем структурных автоматов, а не графов переходов автоматов, что предлагается в данной работе. Перемножение графов переходов автоматов рассмотрено в классической литературе по теории автоматов, где эта задача сформулирована как построение «системы в целом как автомата» (https://www.klex.ru/1q56). Однако излагаемый там метод не позволяет перемножать вложенные графы переходов, что чрезвычайно важно при их применении на практике. В настоящей работе эта проблема решается за счет разработки метода перемножения таких графов переходов. Отметим, что близкие задачи активно исследуются как в отечественных, так и в зарубежных работах. Конформному тестированию конечных автоматов и синтезу проверяющих тестов с гарантированной полнотой посвящен ряд работ [14–17]. Методы построения, активного обучения и тестирования автоматных моделей, в том числе недетерминированных и взаимодействующих автоматов, развиваются в [18–23]. Современные подходы к верификации, проверке на моделях и поведенческой эквивалентности автоматных спецификаций изложены в [24–28].

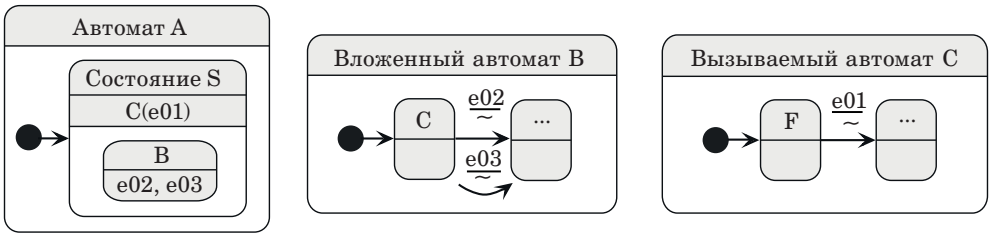
Обозначения и определения. Постановка задачи

В данной работе конечные автоматы рассматриваются как графы переходов. Автомат Ai задается кортежем $(Si, s_0i, Ei, \delta i, Ni)$, где Si — конечное множество состояний, $s_0i \in Si$ — начальное состояние, Ei — множество входных воздействий, δi: $Si \times Ei \rightharpoonup Si$ — частичная функция переходов, Ni: $Si \to 2^{Id}$ — функция вложенности, сопоставляющая каждому состоянию множество идентификаторов вложенных автоматов. Переход содержит событие, условие v, выходные воздействия. Вершины также могут иметь выходные воздействия, генерируемые при входе в них. На графах переходов дуга записывается в виде e [& v] / z, где слева от «/» указываются событие и условие, справа — выходные воздействия (символ «~» означает их отсутствие, что используется на рисунках данной работы). Если из одного состояния по одному событию существует несколько переходов, срабатывает тот, условие v которого истинно. Если подходящего перехода нет, состояние автомата не изменяется.

Глобальное состояние системы автоматов — это кортеж текущих состояний всех автоматов. В отличие от классического перемножения графов, где одно входное событие адресуется всем автоматам одновременно, в рассматриваемой системе внешнее входное воздействие адресуется конкретному автомату системы. Этот автомат может изменить свое состояние и породить внешние выходные воздействия или межавтоматные вызовы. В нотации переходов используются следующие обозначения: e — входное событие; v — условие перехода (булева формула над x и yi), где x — входная переменная, yi — номер текущей вершины автомата i; z — внешнее выходное воздействие; $Ai(e)$ — вызов автомата i с событием e как выходное воздействие. Запись $yi = j$ означает проверку того, что автомат i находится в вершине с номером j; запись $yi! = j$ означает обратное. Знак «=» для компактности заменяет знак «==» языка С. Знак «!» в булевых формулах заменяет отрицание.

Под межавтоматным взаимодействием понимаются два механизма: вызов другого автомата $Ai(e)$ и условие на переходе, зависящее от состояния другого автомата, например $yi = j$. Вызов выполняется синхронно: вызываемый автомат обрабатывает переданное событие в рамках того же вызова. В рассматриваемой реализации предполагается, что один переход или вход в состояние порождает не более одного межавтоматного вызова, адресованного одному из двух перемножаемых автоматов.

Вложенный автомат B — это автомат, указанный в одном или нескольких состояниях автомата A. Если A получает входное воздействие, находясь в таком состоянии, воздействие сначала передается вложенному автомату B, а затем обрабатывается самим автоматом A. В данной работе рассматриваются случаи с не более чем одним автоматом, вложенным в состояние другого автомата. Дополнительным ограничением является запрет обратного вызова: автомат не может в рамках одного шага вызвать автомат, который уже вызвал его. Это ограничение предотвращает циклическое распространение межавтоматных воздействий.



■ ***Рис. 1.*** Пример системы с вложенным и вызываемым автоматом
■ ***Fig. 1.*** An example of a system with a nested and a called automaton

Вложенный в вершину автомат обозначается дробью: в числителе — его имя, в знаменателе — события, на которые он реагирует. Рассмотрим используемую в дальнейшем нотацию на примере. На графе переходов (рис. 1) в состоянии S автомата A присутствуют два вида межавтоматного взаимодействия. Первое — межавтоматное воздействие $C(e01)$: при входе A в состояние S вызывается автомат C с событием $e01$. Второе — вложенный автомат B, который реагирует только на два воздействия: $e02$, $e03$. Автомат может одновременно быть вложенным для одного автомата и вызываемым другим автоматом.

Задача состоит в построении автомата-произведения, состояния которого соответствуют достижимым глобальным состояниям перемножаемых автоматов. Такой автомат используется для последующего анализа спецификации: поиска запрещенных комбинаций состояний, некорректных переходов и тупиковых состояний, — а также последующей модификации исходной системы автоматов на основе данного преобразования.

Под поведенческой эквивалентностью будем понимать совпадение последовательностей внешних выходных воздействий исходной системы и автомата-произведения для любой в инженерном смысле этого слова конечной последовательности внешних входных воздействий. В программной реализации это свойство проверяется имитационным моделированием на конечном наборе последовательностей событий.

Описание подхода, основанного на методе перемножения автоматов

Процесс последовательного перемножения и валидации системы конечных автоматов ($A1$, $A2$, $A3$, …) включает следующие шаги.

Шаг 1. Выбор пары автоматов. Из исходной системы выбираются два конечных автомата Ai и Aj, для которых требуется построить автомат-произведение.

Шаг 2. Инициализация обхода. Формируется очередь Q, содержащая пары вершин. Изначально она содержит пару стартовых вершин автоматов Ai и Aj. Дополнительно вводится множество посещенных вершин V и отображение переходов T, в котором для каждой пары вершин и входного воздействия фиксируется переход в соответствующую пару вершин.

Шаг 3. Итеративное построение множества вершин. Пока очередь Q не пуста, выполняются следующие действия, представленные в виде псевдокода:

```
while Q is not empty:
  (a, b) = Q.pop()
  for each input event e in events(a, b):
    for each (c, d, out) in imitate(a, b, e):
      T[(a, b)][e].add(c, d, out)
      if (c, d) not in V:
        V.add(c, d)
        Q.push(c, d)
```

Здесь a, b — текущие вершины автоматов; $\text{events}(a, b) = \{e \in Eenv \mid \delta i(a, e) \neq \varnothing\} \cup \{e \in Eenv \mid \delta j(b, e) \neq \varnothing\}$ — множество входных воздействий, на которые реагирует хотя бы один из автоматов в текущих вершинах; $\text{imitate}(a, b, e)$ — функция имитации, возвращающая множество результатов (c, d, out): по одному для каждого перехода, применимого при событии e. Если из одной вершины по одному событию существует несколько переходов с разными условиями x, каждый порождает отдельный результат и отдельную дугу в итоговом графе.

Шаг 4. Формирование автомата-произведения. По завершении обхода множество посещенных пар вершин образует множество состояний автомата-произведения $Ai * Aj$, а сформированное отображение T определяет его функцию переходов.

Шаг 5. Валидация результата. Полученный автомат эвристически анализируется относительно явно заданных ограничений корректности: отсутствия запрещенных комбинаций состояний, некорректных переходов и тупиковых достижимых состояний. В случае обнаружения несоответствий исходные автоматы Ai, Aj модифицируются, после чего процедура перемножения повторяется.

Шаг 6. Последовательное перемножение системы автоматов. Если система содержит более двух автоматов, построенный автомат-произведение *Ai* * *Aj* считается новым автоматом *Ai*, а следующий автомат системы принимается за новый *Aj*. После этого метод вновь начинает работу с шага 1.

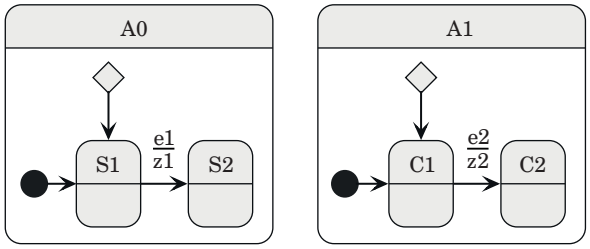
Далее подробно рассматриваются шаги 2, 3 и 4 предложенного метода, тогда как этапы валидации будут продемонстрированы на примере в следующем разделе.

Шаг 2. Инициализация обхода. До начала обхода оба автомата *A*0, *A*1 (рис. 2) находятся в начальных состояниях *S*1, *C*1 соответственно.

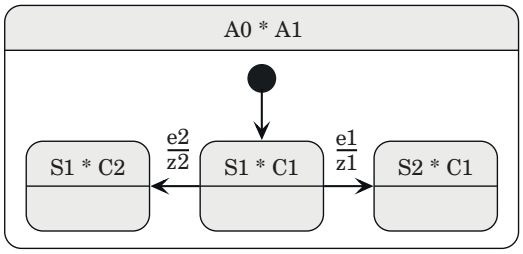
Далее на представленных графах в данной работе черной точкой отмечены стартовые вершины, а ромбом — вершины, в которых автоматы находятся в текущий момент.

Шаг 3. Итеративное построение множества вершин. Метод основан на вызове конкретного конечного автомата системы с заданным входным событием (воздействием). При его обработке автомат выполняет не более одного перехода и может вызвать другой автомат, передав ему межавтоматное воздействие. Такой процесс распространения воздействий и изменения состояний автоматов назовем имитацией (imitate(*a*, *b*, *e*)) *входного события* (рис. 3, *а* и *б*). Результат каждой имитации отражается в итоговом графе переходов: определяется соответствующая комбинация вершин, которая либо уже присутствует в графе, либо добавляется в него, а после этого проводится дуга от предыдущей пары вершин с пометкой *входного* и возникших *выходных воздействий* (рис. 4).

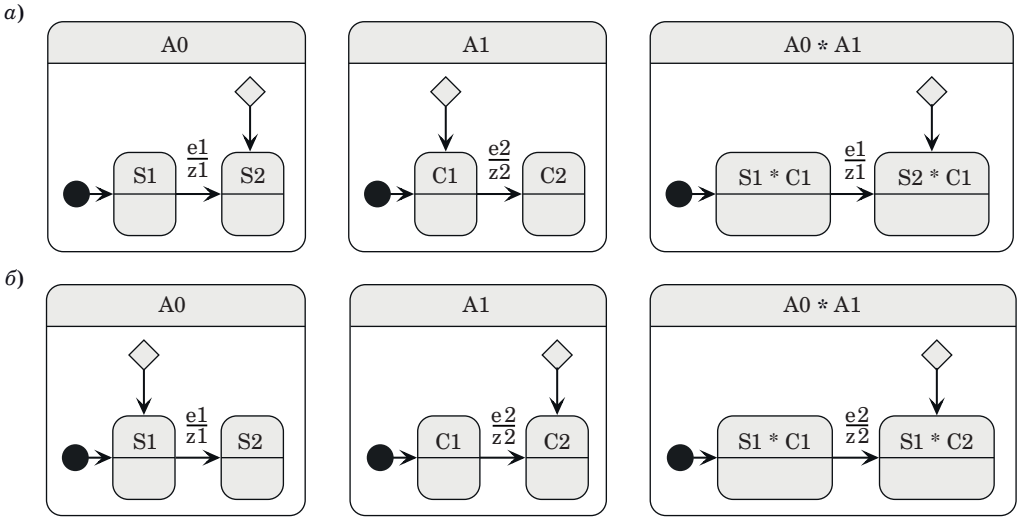
В процессе имитации выходное воздействие считается межавтоматным только в том случае, если оно адресовано второму автомату из данной пары. В случае, когда в процессе имитации автомат вызывает второй из текущей пары (рис. 5), переход затрагивает состояния обоих автоматов одновременно, и результирующая пара рассматривается как единая комбинация, в отличие от случая без межавтоматного взаимодействия (рис. 6, *а* и *б*, рис. 7). Воздействия, адресованные автоматам вне текущей пары, на данном шаге не учитываются. Они будут обработаны при последующем перемножении автомата-произведения с остальными автоматами системы.



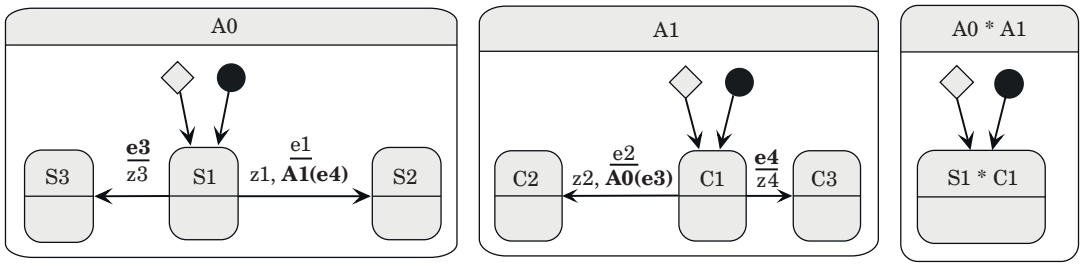
■ ***Рис. 2.*** Система графов в начальных состояниях
■ ***Fig. 2.*** The system of graphs in their initial states



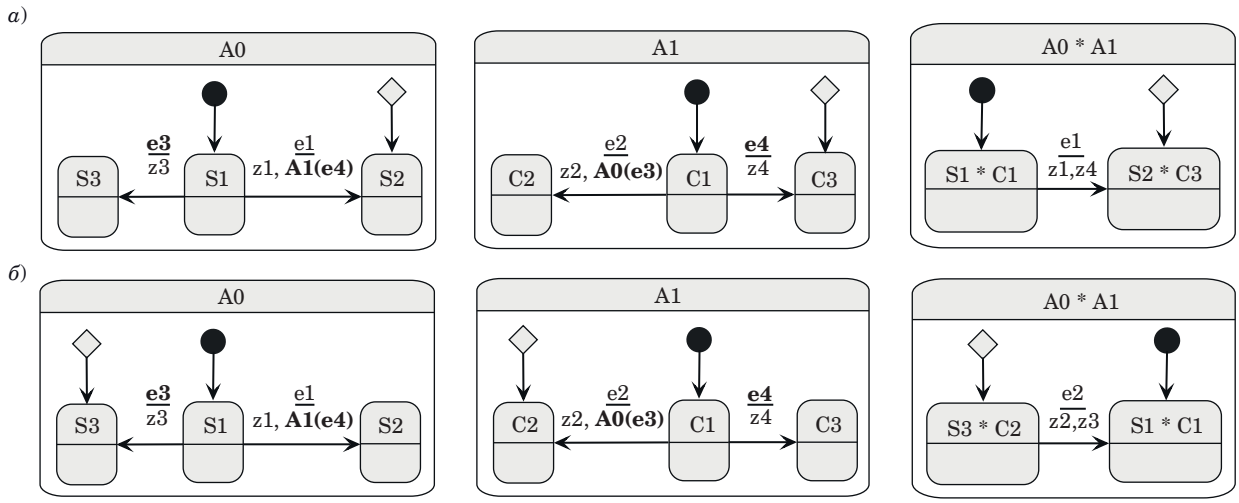
■ ***Рис. 4.*** Граф переходов после двух вызовов
■ ***Fig. 4.*** The transition graph after two calls



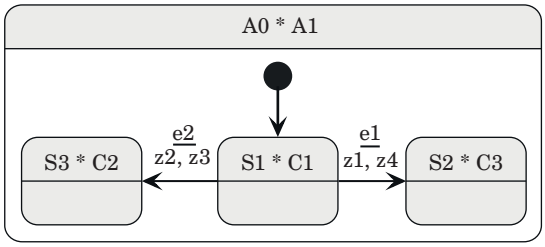
■ ***Рис. 3.*** Результат вызова автомата *A*0 с событием *e*1 (*а*) и *A*1 с событием *e*2 (*б*)
■ ***Fig. 3.*** The result of calling automaton *A*0 with event *e*1 (*а*) and *A*1 with event *e*2 (*б*)



■ ***Рис. 5.*** Система конечных автоматов, содержащая межавтоматные воздействия
■ ***Fig. 5.*** A system of finite automata containing inter-automata actions

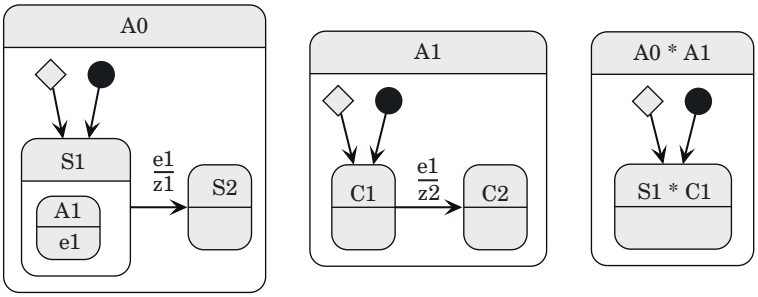


■ ***Рис. 6.*** Результат вызова с межавтоматным воздействием автомата $A0$ с событием $e1$ (*а*) и $A1$ с событием $e2$ (*б*)
■ ***Fig. 6.*** The result of calling with inter-automata actions between automaton $A0$ with event $e1$ (*а*) and $A1$ with event $e2$ (*б*)

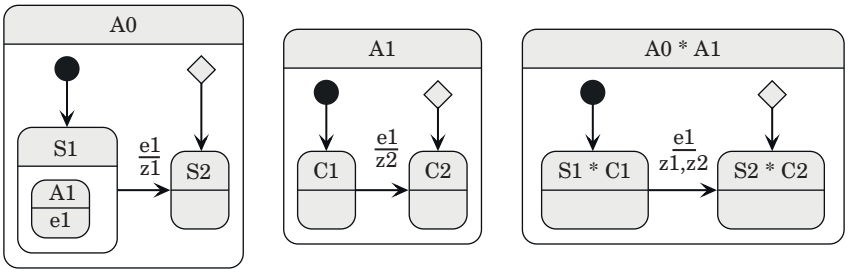


■ ***Рис. 7.*** Граф переходов автомата-произведения
■ ***Fig. 7.*** The transition graph of the product automaton

Входное воздействие всегда поступает из внешней среды — межавтоматные вызовы $Ai(e)$ выполняются при имитации и в итоговом графе как входные события не фигурируют. Вложенные автоматы также учитываются в процессе имитации (рис. 8, 9): воздействие сначала передается вложенному автомату $A1$, затем обрабатывается автоматом $A0$. Также, если перемножение в данном шаге выполняется с другим автоматом, вложенный автомат в имитации не участвует — ссылка на него в состояниях сохраняется и будет учтена при последующем перемножении с ним.



■ ***Рис. 8.*** Система конечных автоматов, содержащая вложенный автомат $A1$
■ ***Fig. 8.*** A system of finite automata containing a nested automaton $A1$

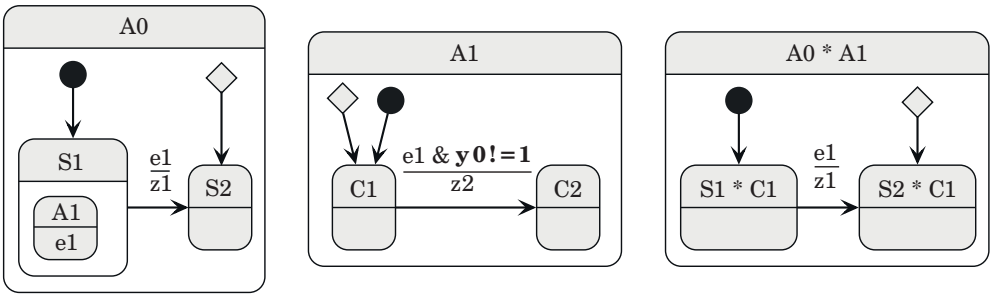


■ ***Рис. 9.*** Результат вызова автомата $A0$ (с вложенным автоматом $A1$) с событием $e1$
■ ***Fig. 9.*** The result of calling automaton $A0$ (with nested automaton $A1$) with event $e1$

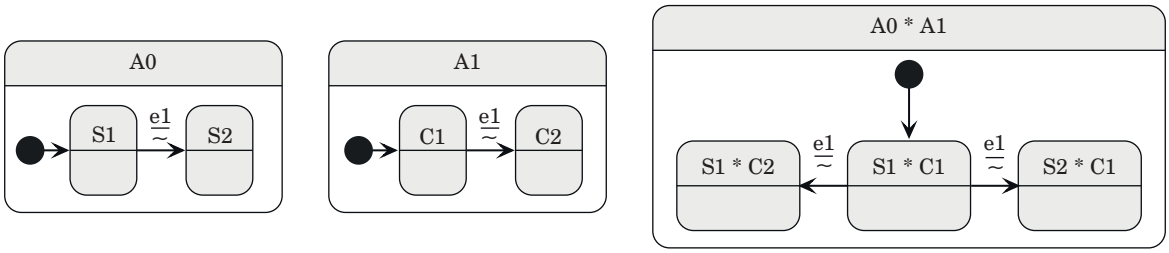
На первый взгляд, обработка вложенного автомата напоминает стандартное перемножение, где каждый автомат независимо реагирует на воздействие. Разница в порядке: управление передается сначала вложенному автомату, затем возвращается обратно «внешнему» автомату. Условия на переходах вида x вычисляются в момент выбора перехода и не зависят от состояния другого автомата. Условие $yi = j$, напротив, обращается к текущему состоянию автомата, j — это межавтоматное взаимодействие, которое влияет на выбор перехода (рис. 10).

Отметим, что если оба автомата независимо реагируют на одно и то же внешнее воздействие, в автомате-произведении из одного состояния возникают два перехода с одинаковым входным воздействием (рис. 11). Автомат-произведение становится недетерминированным: из одного состояния по одному входному воздействию невозможно однозначно выбрать переход. Условия детерминизма для систем автоматов исследованы в [12].

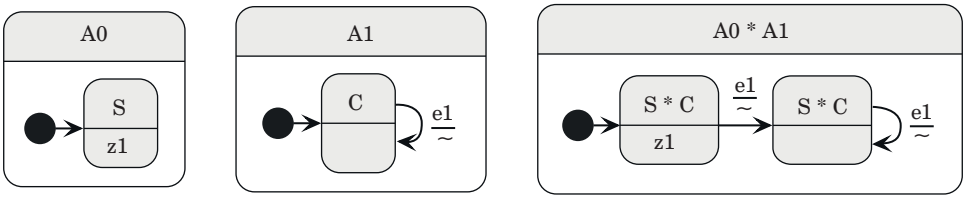
Шаг 4. Формирование автомата-произведения. Метод оперирует состояниями и условиями переходов исходных автоматов. Он позволяет итеративно рассматривать все возможные входные воздействия для пар вершин, с помощью процесса имитации определять достижимые вершины и добавлять их в множество вершин итогового графа. Для каждой вновь полученной пары вершин процесс имитации повторяется. Таким образом, результирующий автомат формируется как граф достижимых состояний системы автоматов, а не как декартово произведение автоматов.



■ ***Рис. 10.*** Результат вызова автомата $A0$ (с условием $y0!=1$) с событием $e1$
■ ***Fig. 10.*** The result of calling automaton $A0$ (with condition $y0!=1$) with event $e1$



■ ***Рис. 11.*** Система конечных автоматов, не допускающая корректного перемножения
■ ***Fig. 11.*** A system of finite automata that does not admit a correct composition



■ ***Рис. 12.*** Пример перемножения автоматов с выходными воздействиями в вершинах
■ ***Fig. 12.*** An example of automata composition with output actions in the vertices

В результирующем графе возможно появление вершин, соответствующих одной и той же паре базовых состояний, но различающихся выходными воздействиями, связанными с входом в вершину (рис. 12). В этом случае происходит дублирование вершин в итоговом графе. Поэтому размер автомата-произведения определяется не только числом пар базовых состояний, но и числом различимых наборов таких выходных воздействий. В общем случае метод не гарантирует уменьшения числа состояний.

Межавтоматные взаимодействия связывают переходы двух автоматов в один и тем самым могут сокращать число состояний и переходов в автомате-произведении.

Пример. Рассмотрим систему конечных автоматов из работы «Имитатор игрового автомата класса “Однорукий бандит”» [29].

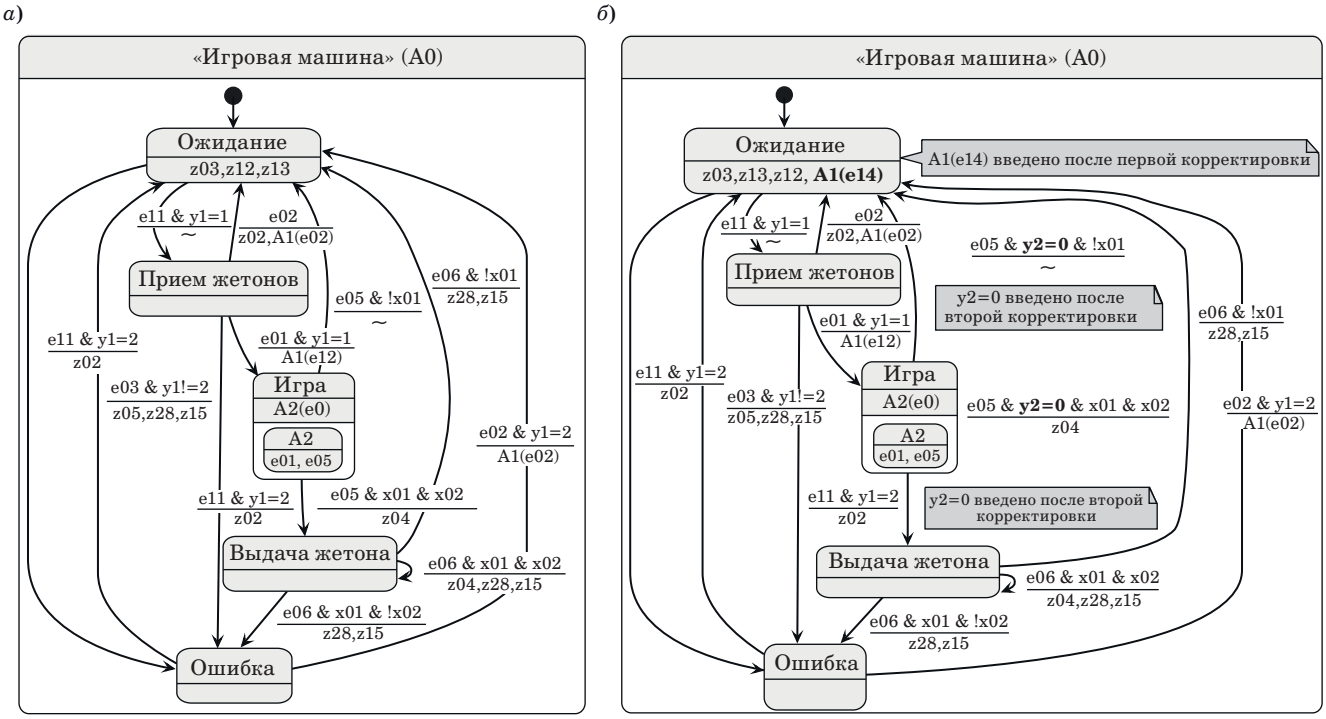
Система состоит из трех автоматов с суммарным числом состояний 12. Автомат 0 «Игровая машина» (*A*0) — пять состояний: «Ожидание», «Прием жетонов», «Игра», «Ошибка», «Выдача жетона». Схема связей, определяющая интерфейс автомата *A*0, и его граф переходов приведены на рис. 13 и 14, *а*. Графы переходов из [29] были уточнены и приведены к единообразной нотации.

Особенность графа переходов автомата *A*0 на рис. 14 состоит в том, что в вершине «Игра» автомат *A*2 упоминается два раза. Вверху вершины он изображен как автомат, вызываемый с событием *e*0; это означает, что *A*0 вызовет *A*2 при входе в состояние «Игра». Снизу автомат *A*2 изображен как вложенный согласно введенной нотации, в числителе которой приведено его обозначение, в знаменателе — события *e*01 и *e*05, на которые он реагирует. На рис. 14, *б* представлен итоговый граф переходов автомата *A*0, включающий корректировки, обоснование которых приводится в последующих разделах.

Автомат 1 «Монетоприемник» (*A*1) — три состояния: «Монетоприемник пуст», «Жетон принят», «Жетон не распознан». Схема связей, определяющая интерфейс автомата *A*1, и его граф переходов приведены на рис. 15 и 16, *а*. Автомат *A*1, находясь в состояниях «Жетон принят» и «Жетон не распознан», вызывает автомат *A*0 с событием *e*11. Исходное выходное воздействие *z*13 «Поместить жетоны в банк» в автомате *A*1 из [29] заменено на *z*16, так как оно конфликтовало с *z*13 «Сбросить ставку» в автомате *A*0. Итоговый граф переходов автомата *A*1, включающий корректировки, которые рассматриваются далее, представлен на рис. 16, *б*.

		A0			
Среда	Нажата кнопка «Игра»	e01	z02	Закрыть монетоприемник	Среда
	Нажата кнопка «Возврат»	e02	z03	Открыть монетоприемник	
	Нажата кнопка «Тех. сброс»	e03	z04	Выдать жетон из банка	
	Сработал таймер третьего барабана	e05	z05	Сбросить сумму выигрыша	
	Выдан жетон из банка	e06	z12	Обновить индикатор «Ставка»	
	Сумма выдачи выигрыша больше нуля	x01	z13	Сбросить ставку	
	В банке есть жетоны	x02	z15	Обновить индикатор «Банк»	
			z28	Обновить индикатор «Выигрыш»	
A1	Опущен жетон	e11			
	Жетон принят	y1=1	e02	Нажата кнопка «Возврат»	A1
	Жетон не распознан	y1=2	e12	Необходимо перевести жетоны в банк	
			e14	**Открыть монетоприемник**	
A2	**Барабаны прокручены**	**y2=0**			
			e0	Начало игры	A2
			e01	Нажата кнопка «Игра»	
			e05	Сработал таймер третьего барабана	

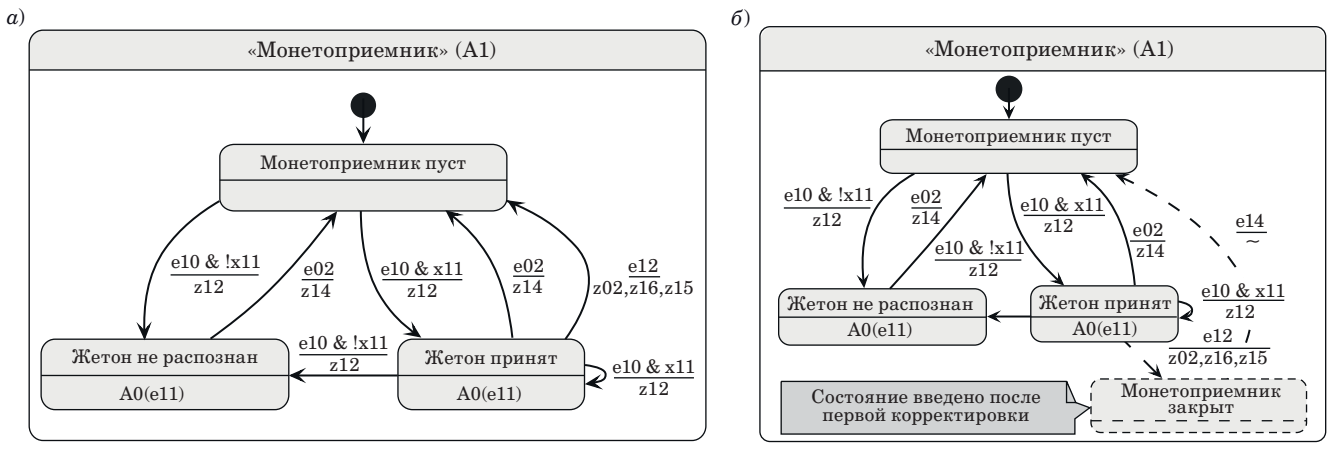
■ ***Рис. 13.*** Интерфейс автомата «Игровая машина» (*A*0)
■ ***Fig. 13.*** The interface of the “Gaming machine” automaton (*A*0)



■ ***Рис. 14.*** Граф переходов автомата «Игровая машина» (*A*0) до (*а*) и после двух корректировок (*б*)
■ ***Fig. 14.*** The transition graph of the “Gaming machine” automaton (*A*0) before (*a*) and after two corrections (*б*)

		A1			
Среда	Опущен жетон	e10	z02	Закрыть монетоприемник	Среда
	Жетон подлинный	x11	z12	Обновить индикатор «Ставка»	
			z14	Вернуть деньги	
A0	Нажата кнопка «Возврат»	e02	z15	Обновить индикатор «Банк»	
	Необходимо перевести жетоны в банк	e12	z16	Поместить жетоны в банк	
	Открыть монетоприемник	**e14**			
			e11	Опущен жетон	A0

■ ***Рис. 15.*** Интерфейс автомата «Монетоприемник» (*A*1)
■ ***Fig. 15.*** The interface of the “Coin acceptor” automaton (*A*1)



■ ***Рис. 16.*** Граф переходов автомата «Монетоприемник» (*A*1) до (*а*) и после одной корректировки (*б*)
■ ***Fig. 16.*** The transition graph of the “Coin acceptor” automaton (*A*1) before (*a*) and after one correction (*б*)

Автомат 2 «Игровые барабаны» (*A*2) — четыре состояния: «Барабаны остановлены», «Барабаны запущены», «Первый барабан остановлен», «Второй барабан остановлен». Схема связей, определяющая интерфейс автомата *A*2, и его граф переходов приведены на рис. 17, *а* и *б*.

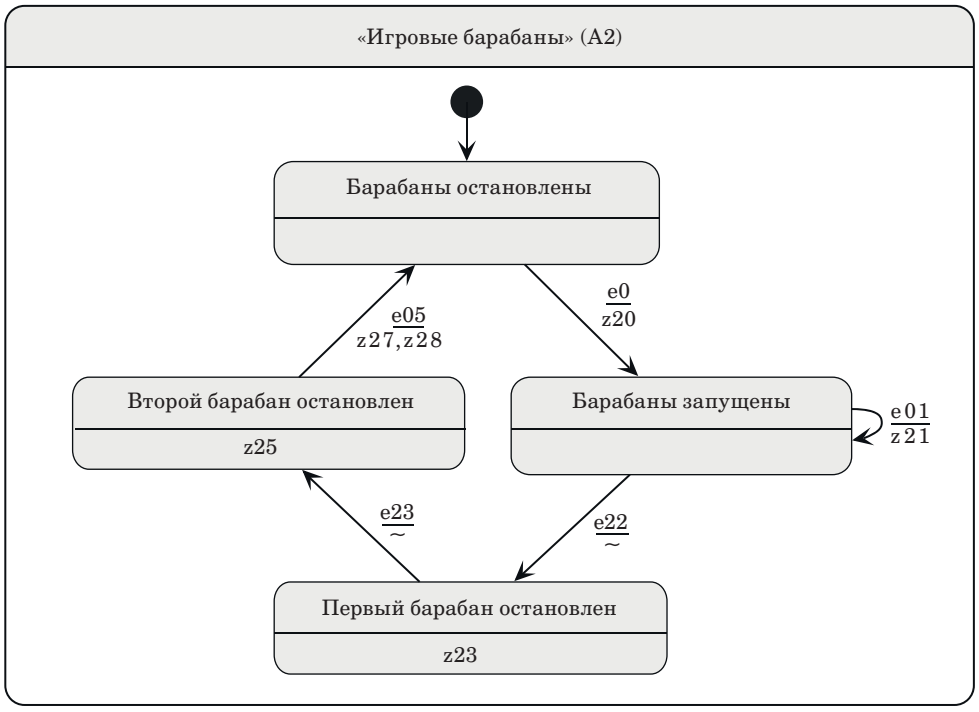
Краткое описание функционирования системы: автомат *A*0 разделяет работу на несколько фаз. Первая соответствует состояниям «Ожидание» и «Прием жетонов» автомата *A*0 и описывает процесс приема жетонов. В этой фазе задействованы автоматы *A*0 и *A*1, тогда как *A*2 не получает входных событий. Далее пользователь нажимает кнопку «Игра», и начинается следующая фаза, соответствующая состояниям «Игра» и «Выдача жетона» автомата *A*0, которая описывает процесс прокрутки барабанов, подсчета и выдачи выигрыша. В этой фазе задействованы автоматы *A*0 и *A*2, тогда как *A*1 не получает входных событий до завершения игры. Данное разделение на фазы определяет порядок перемножения: сначала *A*0 * *A*1, что соответствует первой фазе, а затем результат перемножается с *A*2 в соответствии со второй фазой.

Для анализа результата в спецификацию работы были добавлены два ограничения корректности. Они не сформулированы явно в исходной работе, однако логически вытекают из постановки задачи: их нарушение способно нарушить корректность работы системы в целом. Во-первых, прием жетонов разрешен только в состояниях «Ожидание» и «Прием жетонов» автомата *A*0; во время игры монетоприемник должен быть закрыт. Во-вторых, автомат *A*0 не должен выходить из состояния «Игра» по событию *e*05 до тех пор, пока автомат *A*2 не перешел в состояние «Барабаны остановлены».

а)

		A2			
Среда	Сработал таймер первого барабана	e22	z20	Запустить барабаны	Среда
	Сработал таймер второго барабана	e23	z21	Запустить таймер первого барабана	
			z23	Запустить таймер второго барабана	
A0	Начало игры	e0	z25	Запустить таймер третьего барабана	
	Нажата кнопка «Игра»	e01	z27	Установить сумму выигрыша	
	Сработал таймер третьего барабана	e05	z28	Обновить индикатор «Выигрыш»	

б)



■ ***Рис. 17.*** Интерфейс (*а*) и граф переходов (*б*) автомата «Игровые барабаны» (*A*2)

■ ***Fig. 17.*** The interface (*а*) and the transition graph (*б*) of the “Game reels” automaton (*A*2)

Предложенный метод был применен к рассматриваемой системе. Перемножение автоматов $A0$ и $A1$ привело к автомату, содержащему 13 состояний (рис. 18, *а*). Анализ полученного автомата показал, что он содержит состояния и переходы, нарушающие первое ограничение корректности (они выделены пунктиром). Состояние «Ожидание * Жетон не распознан» является тупиковым достижимым состоянием: оно достижимо в автомате-произведении, но из него не следует допустимое продолжение, соответствующее логике системы.

Из спецификации работы системы следует, что прием жетонов разрешен только в состояниях «Ожидание» и «Прием жетонов» автомата $A0$. В автомате-произведении есть состояния, нарушающие это ограничение. В состояниях «Игра», «Выдача жетона» и «Ошибка» автомата $A0$ обнаружена возможность вносить жетоны. Программа запрещает такое поведение пользователю за счет физической блокировки монетоприемника на программном уровне, хотя сама система конечных автоматов допускает неправильные состояния, которые приведены на рис. 18, *а*. При этом отметим, что в автоматной спецификации из работы «Однорукий бандит» [29], которая валидируется в данной работе, это не оговаривалось. Полагаться на внешнюю программу как на единственный механизм защиты от некорректного поведения небезопасно: спецификация автоматов должна явно исключать недопустимые состояния, а не перекладывать ответственность на внешние факторы.

Для устранения этой проблемы в автомат $A1$ было добавлено состояние «Монетоприемник закрыт» (см. рис. 16, *б*), а в $A0$ — событие $e14$ для явного открытия монетоприемника при возврате автомата $A0$ в состояние «Ожидание» (см. рис. 14, *б*). Тем самым запрет приема жетонов во время игры был перенесен из внешней среды в саму автоматную спецификацию, что исключает зависимость корректности от конкретной реализации физической блокировки монетоприемника.

Откорректированные конечные автоматы $A0$ и $A1$ были вновь перемножены. Внесенные изменения позволили сократить размер перемноженного автомата $A0 * A1$ с 13 до шести состояний (рис. 18, *б*).

Далее было произведено перемножение откорректированного автомата $A0 * A1$ (см. рис. 18, *б*) с автоматом $A2$ (см. рис. 17, *б*), что привело к автомату-произведению с 35 состояниями (рис. 19). В состояниях-группах, отмеченных на рисунке пунктиром, скрыты соответствующие состояния и переходы.

При анализе графа нет необходимости рассматривать его целиком, достаточно анализировать состояния в отдельности: «Выдача жетона * * Монетоприемник закрыт * Барабаны запущены», «Ожидание * Монетоприемник пуст * Барабаны запущены», «Прием жетонов * Жетон принят * * Барабаны запущены», «Ошибка * Монетоприемник закрыт * Барабаны запущены» и другие. Все эти состояния нарушают второе ограничение корректности: при запущенных барабанах невозможно начать прием новых жетонов, а автомат $A0$ не должен переходить в состояние «Ожидание» до завершения прокрутки всех барабанов.

На графе (см. рис. 19) также были найдены некорректные переходы: из состояний «Игра * * Монетоприемник закрыт * Барабаны запущены» и «Игра * Монетоприемник закрыт * Первый барабан остановлен» с событием $e05$ (сработал таймер третьего барабана). Они порождали все остальные некорректные состояния и переходы.

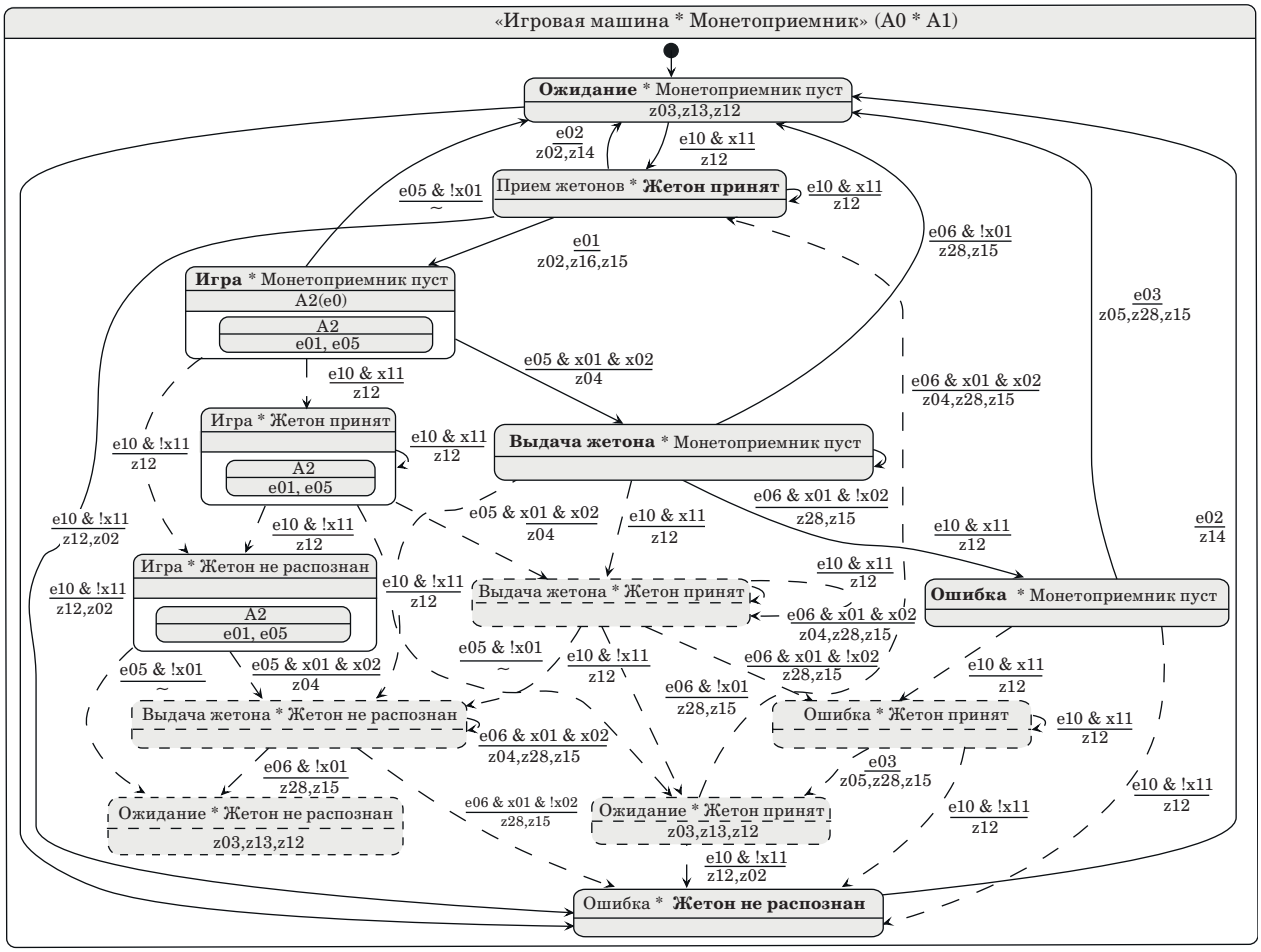
В процессе игры среда вызывает автомат $A2$ с событиями $e22$, $e23$ и автомат $A0$ с событием $e05$. Эти события описывают поочередную остановку барабанов по таймеру, при этом при наступлении события $e05$ автомат $A0$ выходит из состояния «Игра». Предполагается, что среда соблюдает строгий порядок этих событий. Однако в системе автоматов это не учитывается, и допускается переход в некорректное состояние в случае, если автомат $A0$ будет вызван с событием $e05$ без вызовов автомата $A2$ с событиями $e22$ и $e23$. Для того чтобы $A0$ не выходил из состояния «Игра» до остановки барабанов, к переходам из этого состояния по событию $e05$ добавлено условие $y2 = 0$ (см. рис. 14, *б*), которое явно проверяет остановку всех барабанов.

Отметим, что после внесения изменений в автомат $A0$ изменяется результат произведения $A0 * A1$. Поэтому схема, представленная на рис. 15, становится неактуальной и требует повторного перемножения откорректированного автомата $A0$ с автоматом $A1$. При этом изменения в автомате-произведении $A0 * A1$ являются незначительными и сводятся к добавлению условия $y2 = 0$ на переходы из состояния «Игра * Монетоприемник закрыт». При этом условия этих переходов станут следующими: $e05$ & $(y2 = 0)$ & $!x01$ и $e05$ & $(y2 = 0)$ & $x01$ & $x02$ / $z04$ (где $x01$, $x02$ — входные переменные автомата $A0$, см. рис. 14, *б*).

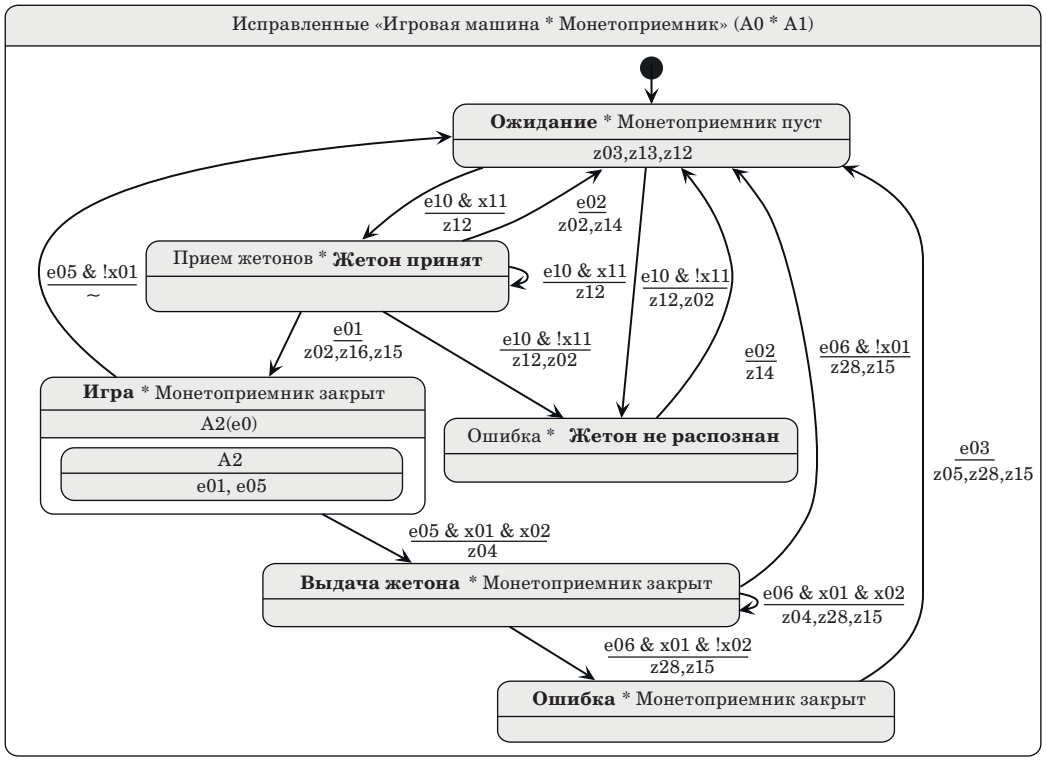
В результате интерфейс автомата-произведения $A0 * A1 * A2$ не содержит события $e0$ (рис. 20, *а*), а его граф переходов имеет всего лишь восемь состояний (рис. 20, *б*). Отсутствие события $e0$ связано с тем, что в исходной системе данное событие участвует только в межавтоматных взаимодействиях и не имеет связи с внешней средой.

После этих изменений в автомате-произведении не осталось выявленных ранее недопустимых состояний и переходов относительно двух сформулированных ограничений. При этом часть проверок во внешней программе стала избыточной, поскольку соответствующие ограничения были перенесены в автоматную спецификацию.

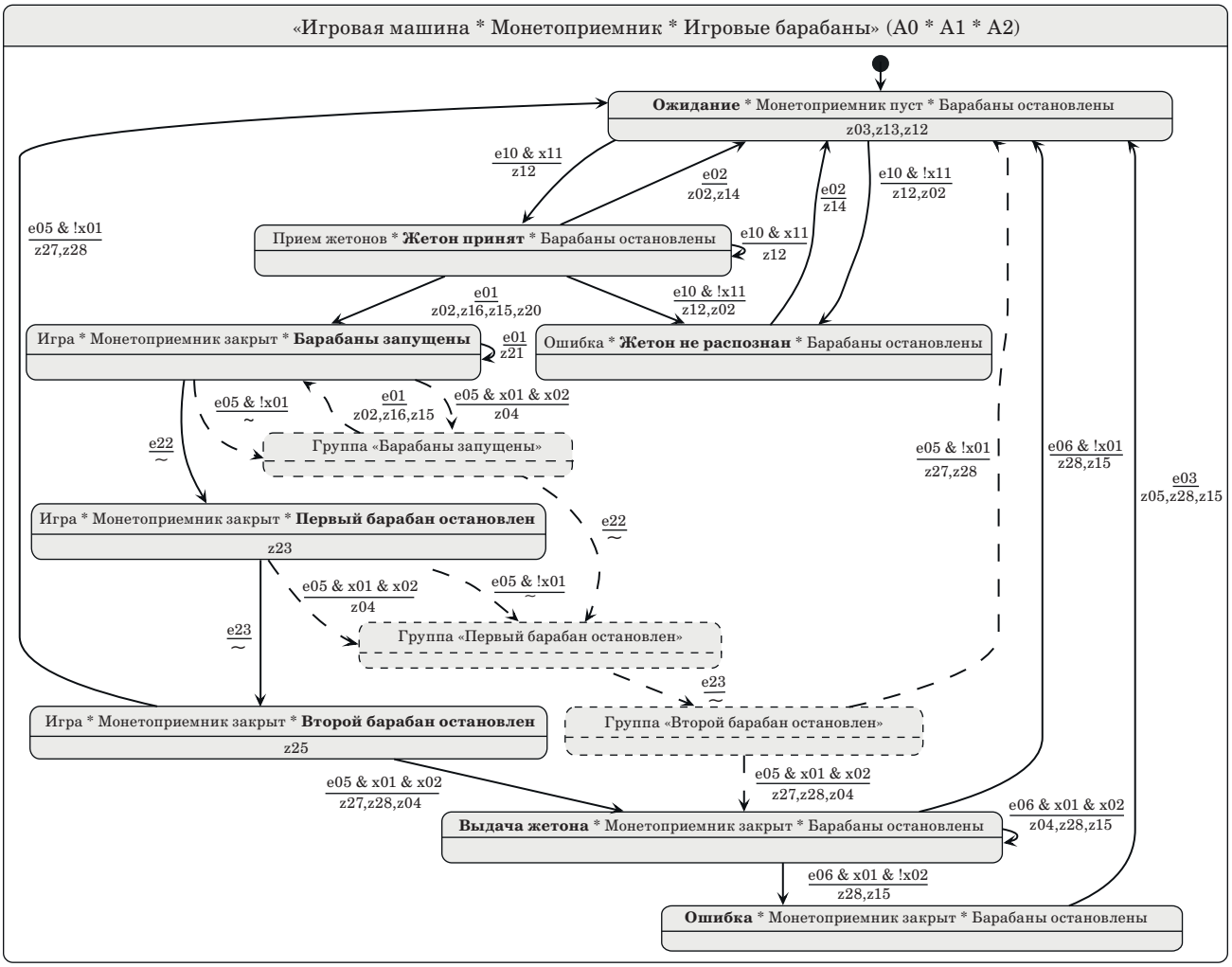
а)



б)



■ ***Рис. 18.*** Граф переходов перемноженного автомата *A*0 * *A*1 до (*а*) и после корректировок (*б*)
■ ***Fig. 18.*** The transition graph of the product automaton *A*0 * *A*1 before (*а*) and after corrections (*б*)



■ ***Рис. 19.*** Граф переходов автомата-произведения *A*0 * *A*1 * *A*2
■ ***Fig. 19.*** The transition graph of the product automaton *A*0 * *A*1 * *A*2

Заключение

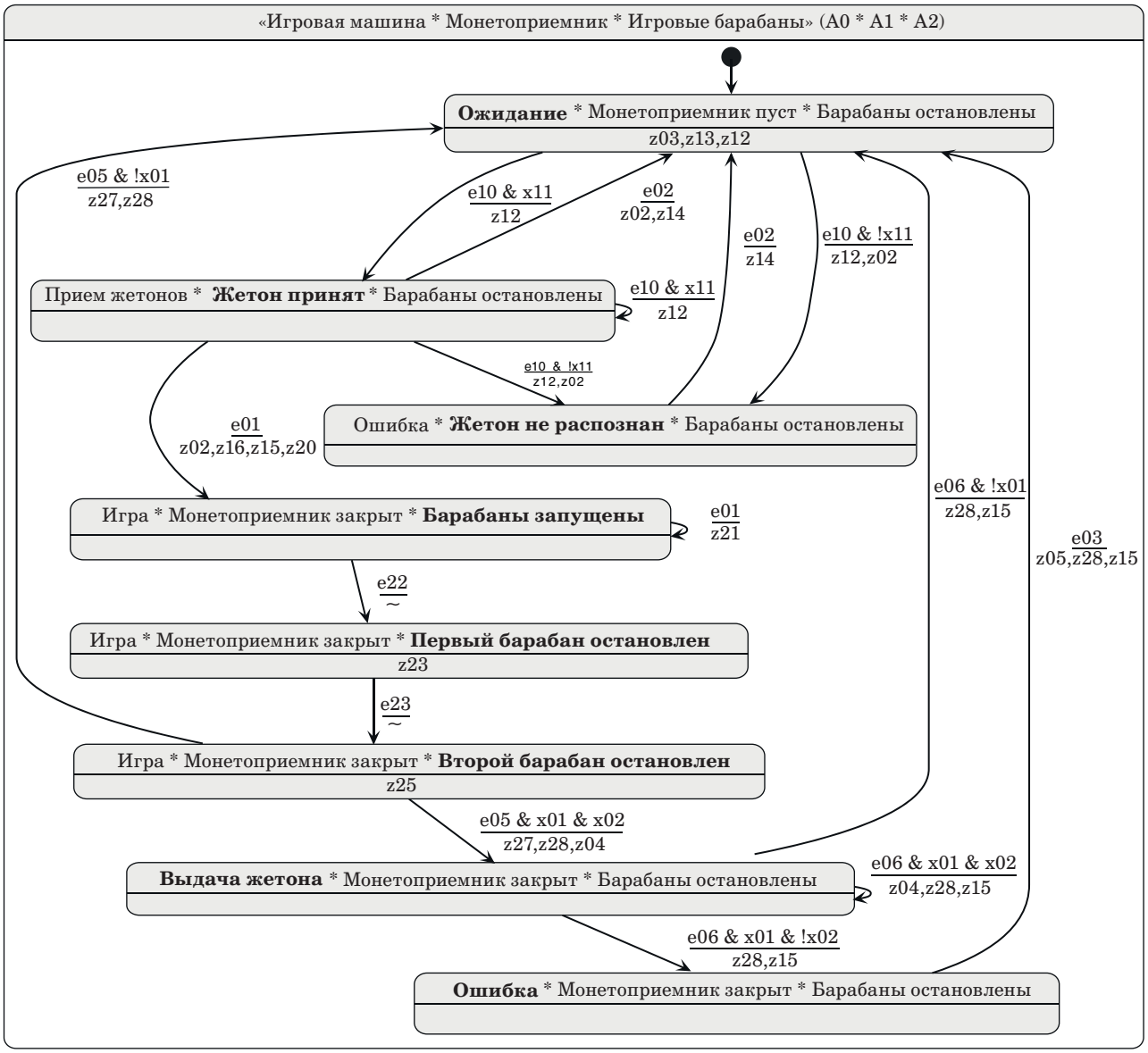
Полученные результаты показывают преимущества предложенного метода по сравнению с современными подходами к валидации автоматных спецификаций. В отличие от методов формальной верификации и проверки на моделях [6–10], требующих построения полной формальной модели и явной формулировки проверяемых свойств, предложенный подход выявляет дефекты (в том числе тупиковые достижимые состояния) на основе эвристического анализа автомата-произведения, построенного перемножением исходных графов переходов, включая вложенные. В отличие от методов активного обучения и тестирования автоматных моделей [17–28], требующих построения тестовых наборов или обучающих последовательностей и зависящих от их полноты, предложенный метод применяется непосредственно к уже заданной системе автоматов и не требует построения таких последовательностей.

Метод перемножения реализован программно (https://github.com/PhoenixNazarov/fsm-multiply). Там же реализован модуль имитационного моделирования (имитационной проверки): откорректированная система автоматов и автомат-произведение исполняются на одинаковых последовательностях внешних входных воздействий, после чего сравниваются последовательности внешних выходных воздействий (экспериментальный стенд с исходной, откорректированной и перемноженной реализациями автомата «Однорукий бандит» доступен по адресу https://github.com/PhoenixNazarov/fsm-slot-machine). Проверка включала полный перебор до глубины 20 и 1000 псевдослучайных последовательностей длиной 100. Расхождений обнаружено не было. Этот результат подтверждает согласованность программной реализации с построенным автоматом-произведением на указанных входных последовательностях событий.

а)

		A0 * A1 * A2			
Среда	Нажата кнопка «Игра»	e01	z02	Закрыть монетоприемник	Среда
	Нажата кнопка «Возврат»	e02	z03	Открыть монетоприемник	
	Нажата кнопка «Тех. сброс»	e03	z04	Выдать жетон из банка	
	Сработал таймер третьего барабана	e05	z05	Сбросить сумму выигрыша	
	Выдан жетон из банка	e06	z12	Обновить индикатор «Ставка»	
	Опущен жетон	e10	z13	Сбросить ставку	
	Сработал таймер первого барабана	e22	z14	Вернуть деньги	
	Сработал таймер второго барабана	e23	z15	Обновить индикатор «Банк»	
	Сумма выдачи выигрыша больше нуля	x01	z16	Поместить жетоны в банк	
	В банке есть жетоны	x02	z20	Запустить барабаны	
	Жетон подлинный	x11	z21	Запустить таймер первого барабана	
			z23	Запустить таймер второго барабана	
			z25	Запустить таймер третьего барабана	
			z27	Установить сумму выигрыша	
			z28	Обновить индикатор «Выигрыш»	

б)



■ ***Рис. 20.*** Интерфейс (*а*) и граф переходов (*б*) автомата-произведения откорректированных *A*0 * *A*1 * *A*2
■ ***Рис. 20.*** The interface (*а*) and the transition graph (*б*) of the product automaton of the corrected *A*0 * *A*1 * *A*2

Из изложенного следует, что предложенный подход является полуавтоматическим. После каждого автоматического перемножения пары автоматов специалист анализирует полученный автомат-произведение относительно заданных ограничений корректности. При обнаружении нарушений исходные автоматы корректируются. После этого перемножение автоматов повторяется. Процедура может выполняться многократно.

Приведенный пример показывает, что на первый взгляд корректно работающая система автоматов может содержать неточности, которые трудно обнаружить только имитационным моделированием. Так, в построенной системе из трех автоматов (формальной спецификации поведения программы) из [29] такие неточности на основе моделирования обнаружены не были. Поэтому предложенный метод следует рассматривать как дополнительный инструмент валидации автоматных спецификаций, особенно в случаях, когда система строится или модифицируется автоматически. В заключение отметим, что в работе [30] показано, как итеративные, а в других работах — рекурсивные алгоритмы могут быть преобразованы в графы переходов автоматов. Поэтому авторы предполагают обобщить предложенный метод валидации на указанные классы алгоритмов.

Выявленная в работе закономерность состоит в том, что небольшое изменение межавтоматного взаимодействия может существенно повлиять на размер автомата-произведения. В рассмотренном примере после устранения ошибок автомат $A0 * A1$ сократился с 13 до шести состояний, и потом $A0 * A1 * A2$ — с 35 до восьми состояний. Это позволяет высказать гипотезу: значительное увеличение размера автомата-произведения по отношению к сумме размеров исходных автоматов может служить эвристическим признаком наличия ошибок в спецификации, а после их устранения автомат-произведение нередко становится более компактным. Предлагаемый метод не гарантирует упрощения автомата-произведения, но является еще одним инженерным средством для обеспечения возможности повышения качества автоматных спецификаций.

Дальнейшее развитие метода связано с его расширением на случай нескольких автоматов, вложенных (в том числе и друг в друга) в состояние другого автомата.

Литература

1. **Поликарпова Н. И., Шалыто А. А.** *Автоматное программирование*. СПб., Питер, 2008, 167 с. http://is.ifmo.ru/books/_book.pdf (дата обращения: 22.03.2026).
2. **Adhikary S., Gurung A., Thakkar J., Bruto da Costa A. A., Dey S., Hazra A., Dasgupta P.** SMT-based verification of safety-critical embedded control software. *IEEE Embedded Systems Letters*, 2021, vol. 13, no. 3, pp. 138–141. doi:10.1109/LES.2020.3035560
3. **André É., Liu S., Liu Y., Choppy C., Sun J., Dong J. S.** Formalizing UML state machines for automated verification — A survey. *ACM Computing Surveys*, 2023, vol. 55, no. 13s, art. 277. doi:10.1145/3579821
4. **Вельдер С. Э., Лукин М. А., Шалыто А. А., Яминов Б. Р.** *Верификация автоматных программ*. СПб., Наука, 2011. 244 с.
5. **Янкин Ю. Ю., Шалыто А. А.** Автоматное программирование ПЛИС в задачах управления электроприводом. *Информационно-управляющие системы*, 2011, № 1, с. 50–56. EDN: NEKQXX
6. **Filipovikj P., Ung G., Gurov D., Nyberg M.** Bounded invariant checking for Stateflow. *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, 2022, vol. 371, pp. 38–52. doi:10.4204/EPTCS.371.3
7. **Xiao M., Xu Y., Zhang Y., Yang K., Yan S., Cen L.** Formal verification of solidity smart contracts via automata theory. *Symmetry*, 2025, vol. 17, no. 8, art. 1275. doi:10.3390/sym17081275
8. **Шалыто А. А.** Валидация автоматных спецификаций. *Научно-технический вестник информационных технологий, механики и оптики*, 2023, т. 23, № 2, с. 436–438. https://ntv.ifmo.ru/file/article/21921.pdf (дата обращения: 20.05.2026). doi:10.17586/2226-1494-2023-23-2-436-438, EDN: YHATJW
9. **Wu R., Du Y., Li M.** A model transformation method based on Simulink/Stateflow for validation of UML statechart diagrams. *Mathematics*, 2025, vol. 13, no. 5, art. 724. doi:10.3390/math13050724
10. **Barth M., Jakobs M.-C.** Test-case generation with automata-based software model checking. *Model Checking Software (SPIN 2024)*, *LNCS*, vol. 14624, Springer, 2024, pp. 248–267. doi:10.1007/978-3-031-66149-5_14
11. **Sankur O.** Timed automata verification and synthesis via finite automata learning. *Journal of Automated Reasoning*, 2025, vol. 69, art. 15. doi:10.1007/s10817-025-09730-z
12. **Бурдонов И. Б., Косачев А. С.** Система автоматов: условия детерминизма и тестирование. *Труды Института системного программирования РАН*, 2016, т. 28, № 1, с. 151–184. doi:10.15514/ISPRAS-2016-28(1)-9, EDN: WAPNTP
13. **Бурдонов И. Б., Косачев А. С.** Тестирование системы автоматов с буферизацией сообщений. *Труды Института системного программирования РАН*, 2016, т. 28, № 1, с. 103–130. doi:10.15514/ISPRAS-2016-28(1)-7, EDN: WAPNQX
14. **Бурдонов И. Б., Косачев А. С.** Развитие теории конформности: семантики, формальные модели, алгоритмы. *Труды Института системного программирования РАН*, 2014, т. 26, № 1, с. 27–72. doi:10.15514/ISPRAS-2014-26(1)-2, EDN: RUMWWB

15. **Ермаков А. Д., Евтушенко Н. В.** К синтезу адаптивных проверяющих последовательностей для недетерминированных автоматов. *Труды Института системного программирования РАН*, 2016, т. 28, № 3, с. 123–144. doi:10.15514/ISPRAS-2016- 28(3)-8, EDN: WKFGUV
16. **Петрухнова Г. В., Усачев М. Г., Попова Е. В.** Тестовый контроль конечных автоматов. *Вестник Воронежского государственного технического университета*, 2023, т. 19, № 6, с. 47–52. doi:10.36622/VSTU.2023.19.6.007, EDN: ENHULJ
17. **Turker U. C., Hierons R. M., El-Fakih K., Mousavi M. R., Tyukin I. Y.** Accelerating finite state machine-based testing using reinforcement learning. *IEEE Transactions on Software Engineering*, 2024, vol. 50, no. 3, pp. 574–597. doi:10.1109/TSE.2024.3358416
18. **Kruger L., Junges S., Rot J.** Small test suites for active automata learning. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2024), LNCS,* vol. 14571, Springer, 2024, pp. 109–129. doi:10.1007/978-3-031-57249-4_6
19. **Borsotti A., Breveglieri L., Crespi Reghizzi S., Morzenti A.** Multientry DFA with reduced initial states to speed-up parallel recognition. *Implementation and Application of Automata (CIAA 2025), LNCS,* vol. 15981. Springer, 2025, pp. 55–72. doi:10.1007/978-3-032-02602-6_5
20. **Vaandrager F., Melse I.** New fault domains for conformance testing of finite state machines. *36th International Conference on Concurrency Theory (CONCUR 2025), LIPIcs*, vol. 348, Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2025, art. 34, pp. 34:1–34:22. doi:10.4230/LIPIcs.CONCUR.2025.34
21. **Ferreira D., Lima L.** Verifying integrated designs of UML state machines and activities using CSP. *Formal Methods: Foundations and Applications (SBMF 2024), LNCS,* Springer, 2024, pp. 68–85. doi:10.1007/978-3-031-78116-2_5
22. **Rahaman H., Chattopadhyay S., Sengupta I.** Conformance testing for finite state machines guided by deep neural network. *Journal of Circuits, Systems and Computers*, 2022, vol. 31, iss. 09, art. 2250156, pp. 1–19. doi:10.1142/S0218126622501560
23. **Muškardin E., Tappler M., Aichernig B. K., Pill I.** Active model learning of stochastic reactive systems. *Software and Systems Modeling*, 2024, vol. 23, no. 2, pp. 503–524. doi:10.1007/s10270-024-01158-0
24. **Fortz S., Ghassemi F., Henry L., Howar F., Neele T., Rot J., Suilen M.** A research agenda for active automata learning. *International Journal on Software Tools for Technology Transfer*, 2026. doi:10.1007/s10009-026-00839-z
25. **Marksteiner S., Schögler D., Sirjani M., Sjödin M.** Learning single and compound-protocol automata and checking behavioral equivalences. *International Journal on Software Tools for Technology Transfer*, 2025, vol. 27, pp. 35–52. doi:10.1007/s10009-025-00797-y
26. **Henry L., Mousavi M. R., Neele T., Sammartino M.** Compositional active learning of synchronizing systems through automated alphabet refinement. *CONCUR 2025, LIPIcs*, vol. 348, Schloss Dagstuhl, 2025, pp. 20:1–20:22. doi:10.4230/LIPIcs.CONCUR.2025.20
27. **Manini A., Rossi M., San Pietro P.** Random testing of model checkers for timed automata with automated oracle generation. *TASE 2025, LNCS,* vol. 15841, Springer, 2025, pp. 343–360. doi:10.1007/978-3-031-98208-8_20
28. **Grobelna I.** Formal verification of control modules in cyber-physical systems. *Sensors*, 2020, vol. 20, no. 18, art. 5154. doi:10.3390/s20185154
29. **Хокканен А. В., Шалыто А. А.** Имитатор игрового автомата класса «Однорукий бандит». СПб., ИТМО, 2003. 44 с. https://is.ifmo.ru/download/SlotMachine_Report.pdf (дата обращения: 20.05.2026).
30. **Шалыто А. А., Туккель Н. И.** Преобразование итеративных алгоритмов в автоматные. *Программирование*, 2002, № 5, с. 12–26. http://is.ifmo.ru/works/iter/ (дата обращения: 20.05.2026).

UDC 621.396
doi:10.31799/1684-8853-2026-5-21-36
EDN: IBKLIO

Validation of automata specifications based on automata composition

V. A. Nazarov[a], Bachelor, orcid.org/0009-0000-0352-7885
A. A. Shalyto[a], Dr. Sc., Tech, Professor, orcid.org/0000-0002-2723-2077, anatoly.shalyto@gmail.com
[a]ITMO University, 49, Kronverksky Pr., 197101, Saint-Petersburg, Russian Federation

Introduction: As automata-based specifications are increasingly used in the development of safety-critical systems, their validation is of particular importance and relevance. Testing and verification do not solve this problem, especially when nested finite automata are used. **Purpose:** To develop a method for constructing a product automaton for the class of specifications under consideration, based on the product of transition graphs and applicable to nested automata, in order to simplify the heuristic procedure for its subsequent validation. **Methods:** The study employs the construction of the reachable part of the state space, breadth-first graph traversal, and simulation-based checking of output event sequences. **Results:** The method forms a product automaton with a set of reachable state pairs and a transition mapping, which allows detecting with visual analysis and manually eliminating deadlocks and other incorrect states and transitions. Sequential composition of several automata provides step-by-step verification of the correctness of the entire system. In the example considered, no discrepancies have been found in the output event sequences between the corrected system and the product automaton on the used input event sequences. **Practical relevance:** The proposed method extends the range of tools available for validating automata-based specifications, including those containing nested automata, although it does not guarantee that the product automaton will be simpler than the original system. **Discussion:** The method is particularly useful for analyzing automatically constructed or modified systems of automata, since it helps detect hidden logical contradictions that are difficult to reveal by testing or simulation alone.

Keywords – automata, nested automata, automata composition, validation.

For citation: Nazarov V. A., Shalyto A. A. Validation of automata specifications based on automata composition. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2026, no. 5, pp. 21–36 (In Russian). doi:10.31799/1684-8853-2026-5-21-36, EDN: IBKLIO

Reference

1. Polikarpova N. I., Shalyto A. A. *Avtomatnoe programmirovanie* [Automata-based programming]. Saint Petersburg, Piter Publ., 2008, 167 p. Available at: http://is.ifmo.ru/books/_book.pdf (accessed 22 March 2026) (In Russian).
2. Adhikary S., Gurung A., Thakkar J., Bruto da Costa A. A., Dey S., Hazra A., Dasgupta P. SMT-based verification of safety-critical embedded control software. *IEEE Embedded Systems Letters*, 2021, vol. 13, no. 3, pp. 138–141. doi:10.1109/LES.2020.3035560
3. André É., Liu S., Liu Y., Choppy C., Sun J., Dong J. S. Formalizing UML state machines for automated verification — A survey. *ACM Computing Surveys*, 2023, vol. 55, no. 13s, art. 277. doi:10.1145/3579821
4. Velder S. E., Lukin M. A., Shalyto A. A., Iaminov B. R. *Verifikatsiia avtomatnykh programm* [Verification of automata-based programs]. Saint Petersburg, Nauka Publ., 2011, 244 p. (In Russian).
5. Yankin Y. Y., Shalyto A. A. A method of finite-state machine realization in electric motor drives control. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2011, no. 1, pp. 50–56 (In Russian). EDN: NEKQXX
6. Filipovikj P., Ung G., Gurov D., Nyberg M. Bounded invariant checking for Stateflow. *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, 2022, vol. 371, pp. 38–52. doi:10.4204/EPTCS.371.3
7. Xiao M., Xu Y., Zhang Y., Yang K., Yan S., Cen L. Formal verification of solidity smart contracts via automata theory. *Symmetry*, 2025, vol. 17, no. 8, art. 1275. doi:10.3390/sym17081275
8. Shalyto A. A. Validation of state machine specifications. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2023, vol. 23, no. 2, pp. 436–438 (In Russian). doi:10.17586/2226-1494-2023-23-2-436-438, EDN: YHATJW
9. Wu R., Du Y., Li M. A model transformation method based on Simulink/Stateflow for validation of UML statechart diagrams. *Mathematics*, 2025, vol. 13, no. 5, art. 724. doi:10.3390/math13050724
10. Barth M., Jakobs M.-C. Test-case generation with automata-based software model checking. *Model Checking Software (SPIN 2024), LNCS*. Springer, vol. 14624, 2024, pp. 248–267. doi:10.1007/978-3-031-66149-5_14
11. Sankur O. Timed automata verification and synthesis via finite automata learning. *Journal of Automated Reasoning*, 2025, vol. 69, art. 15. doi:10.1007/s10817-025-09730-z
12. Burdonov I. B., Kossatchev A. S. Automata system: determinism conditions and testing. *Proceedings of the Institute for System Programming of the RAS,* 2016, vol. 28, no. 1, pp. 151–184 (In Russian). doi:10.15514/ISPRAS-2016-28(1)-9, EDN: WAPNTP
13. Burdonov I. B., Kossatchev A. S. Testing of automata system. *Proceedings of the Institute for System Programming of the RAS,* 2016, vol. 28, no. 1, pp. 103–130 (In Russian). doi:10.15514/ISPRAS-2016-28(1)-7, EDN: WAPNQX
14. Burdonov I. B., Kossatchev A. S. Conformance theory development: semantics, formal models, algorithms. *Proceedings of the Institute for System Programming of the RAS,* 2014, vol. 26, no. 1, pp. 27–72 (In Russian). doi:10.15514/ISPRAS-2014-26(1)-2, EDN: RUMWWB
15. Ermakov A. D., Yevtushenko N. V. Deriving adaptive checking sequence for nondeterministic Finite State Machines. *Proceedings of the Institute for System Programming of the RAS,* 2016, vol. 28, no. 3, pp. 123–144 (In Russian). doi:10.15514/ISPRAS-2016-28(3)-8, EDN: WKFGUV
16. Petrukhnova G. V., Usachev M. G., Popova E. V. Test control of finite state machines. *The Bulletin of Voronezh State Technical University*, 2023, vol. 19, no. 6, pp. 47–52 (In Russian). doi:10.36622/VSTU.2023.19.6.007, EDN: ENHULJ
17. Turker U. C., Hierons R. M., El-Fakih K., Mousavi M. R., Tyukin I. Y. Accelerating finite state machine-based testing using reinforcement learning. *IEEE Transactions on Software Engineering*, 2024, vol. 50, no. 3, pp. 574–597. doi:10.1109/TSE.2024.3358416
18. Kruger L., Junges S., Rot J. Small test suites for active automata learning. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2024), LNCS,* vol. 14571, Springer, 2024, pp. 109–129. doi:10.1007/978-3-031-57249-4_6
19. Borsotti A., Breveglieri L., Crespi Reghizzi S., Morzenti A. Multi-entry DFA with reduced initial states to speedup parallel recognition. *Implementation and Application of Automata (CIAA 2025), LNCS*, vol. 15981. Springer, 2025, pp. 55–72. doi:10.1007/978-3-032-02602-6_5
20. Vaandrager F., Melse I. New fault domains for conformance testing of finite state machines. *36th International Conference on Concurrency Theory (CONCUR 2025), LIPIcs*, vol. 348, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, art. 34, pp. 34:1–34:22. doi:10.4230/LIPIcs.CONCUR.2025.34
21. Ferreira D., Lima L. Verifying integrated designs of UML state machines and activities using CSP. *Formal Methods: Foundations and Applications (SBMF 2024), LNCS*, Springer, 2024, pp. 68–85. doi:10.1007/978-3-031-78116-2_5
22. Rahaman H., Chattopadhyay S., Sengupta I. Conformance testing for finite state machines guided by deep neural network. *Journal of Circuits, Systems and Computers*, 2022, vol. 31, iss. 09, art. 2250156, pp. 1–19. doi:10.1142/S0218126622501560
23. Muškardin E., Tappler M., Aichernig B. K., Pill I. Active model learning of stochastic reactive systems. *Software and Systems Modeling*, 2024, vol. 23, no. 2, pp. 503–524. doi:10.1007/s10270-024-01158-0
24. Fortz S., Ghassemi F., Henry L., Howar F., Neele T., Rot J., Suilen M. A research agenda for active automata learning. *International Journal on Software Tools for Technology Transfer*, 2026. doi:10.1007/s10009-026-00839-z
25. Marksteiner S., Schögler D., Sirjani M., Sjödin M. Learning single and compound-protocol automata and checking behavioral equivalences. *International Journal on Software Tools for Technology Transfer*, 2025, vol. 27, pp. 35–52. doi:10.1007/s10009-025-00797-y
26. Henry L., Mousavi M. R., Neele T., Sammartino M. Compositional active learning of synchronizing systems through automated alphabet refinement. *CONCUR 2025, LIPIcs*, vol. 348, Schloss Dagstuhl, 2025, pp. 20:1–20:22. doi:10.4230/LIPIcs.CONCUR.2025.20
27. Manini A., Rossi M., San Pietro P. Random testing of model checkers for timed automata with automated oracle generation. *TASE 2025, LNCS,* vol. 15841, Springer, 2025, pp. 343–360. doi:10.1007/978-3-031-98208-8_20
28. Grobelna I. Formal verification of control modules in cyber-physical systems. *Sensors*, 2020, vol. 20, no. 18, art. 5154. doi:10.3390/s20185154
29. Khokkanen A. V., Shalyto A. A. *Imitator igrovogo avtomata klassa "Odnorukii bandit"* [Simulator of the slot machine of the "One-armed bandit" class]. Saint Petersburg, ITMO Publ., 2003, 44 p. Available at: https://is.ifmo.ru/download/SlotMachine_Report.pdf (accessed 20 May 2026) (In Russian).
30. Shalyto A. A., Tukkel N. I. Translating iterative algorithms into automaton ones. *Programming and Computer Software,* 2002, vol. 28, no. 5, pp. 250–260. doi:10.1023/A:1020208127964