Опубликовано в Научно-техническом вестнике СПбГУ ИТМО. 2008. Выпуск 53. Автоматное программирование, с. 211-221.

УДК 004.4'242

ВЕРИФИКАЦИИ ВЗАИМОДЕЙСТВИЯ ЧАСТЕЙ РЕАКТИВНОЙ СИСТЕМЫ, РЕАЛИЗОВАННЫХ С ПОМОЩЬЮ АВТОМАТНОГО ПОДХОДА

С.Ю. Канжелев

(Санкт-Петербургский государственный университет информационных технологий, механики и оптики)

В работе рассматриваются вопросы верификации частей реактивной системы, каждая из которых реализована с помощью автоматного подхода. Рассматриваются проблемы, которые могут возникать в таких системах, а также спопобы их обнаружения для различных видов их семантической интерпретации.
Ключевые слова: верификация программ, реактивная система, автоматное программирование

Введение

Метод верификации систем, основанный на моделях (*Model checking*) [1], находит в последнее время все большее число сторонников. Этот метод предполагает построение модели программы и требований к этой модели на языке темпоральной логики. Актуальность этого метода для реактивных систем, реализованных с помощью автоматного подхода, обусловлена тем, что построения модели в этом случае не требуется – набор взаимодействующих автоматов уже является моделью.

Требования к модели, описанные на языке темпоральной логики, как правило, звучат так: «для любой «справедливой» истории выполнены какие-то утверждения». Например, «для любой справедливой истории» действие «положить трубку» всегда происходит после действия «поднять трубку». При этом «справедливой» историей называется такая история, для которой любое ожидаемое событие когда-нибудь произойдет. Однако на практике, наряду с невыполнением требований к модели, большой проблемой являются «непришедшие» события и «несправедливые» истории.

Если событие не приходит, программа может зависнуть в одном состоянии, ожидая это событие. Такая ситуация может возникнуть как из-за плохого проектирования, так и из-за внешних факторов. Но в любом случае необходим анализ диаграмм автоматов, реализующих логику программы, для выявления в ней ситуаций, в которых событие может не прийти.

В настоящей работе проводится классификация возможных причин непредвиденных остановок автоматов. Предлагается способ статической верификации этих ситуаций для случая взаимодействия частей программы, реализованных с помощью автоматов. При этом рассматриваются различные способы семантической интерпретации взаимодействия автоматов.

Причины ожидания событий

Существуют три возможных причин того, что событие, которого ожидает автомат, не приходит. Перечислим их.

1. *Поздняя подписка*: событие пришло раньше времени, когда его еще не ждали.
2. *Задержка*: событие придет, но позже.
3. Событие не придет никогда. Оно может не приходить по двум причинам.
   1. *Голодание*: по внешней причине.
   2. *Блокировка*: в ожидании действий пользователя.

Рассмотрим эти причины ожидания событий на примере автоматов, реализующих логику работы простейших телефона и автоматической телефонной станции (АТС). Диаграмма переходов и схема связей автомата «Телефон» представлены на рис. 1, а для автомата «АТС» будут рассмотрены позже. Эти диаграммы упрощены и отличаются от реальных меньшим числом действий и переходов.

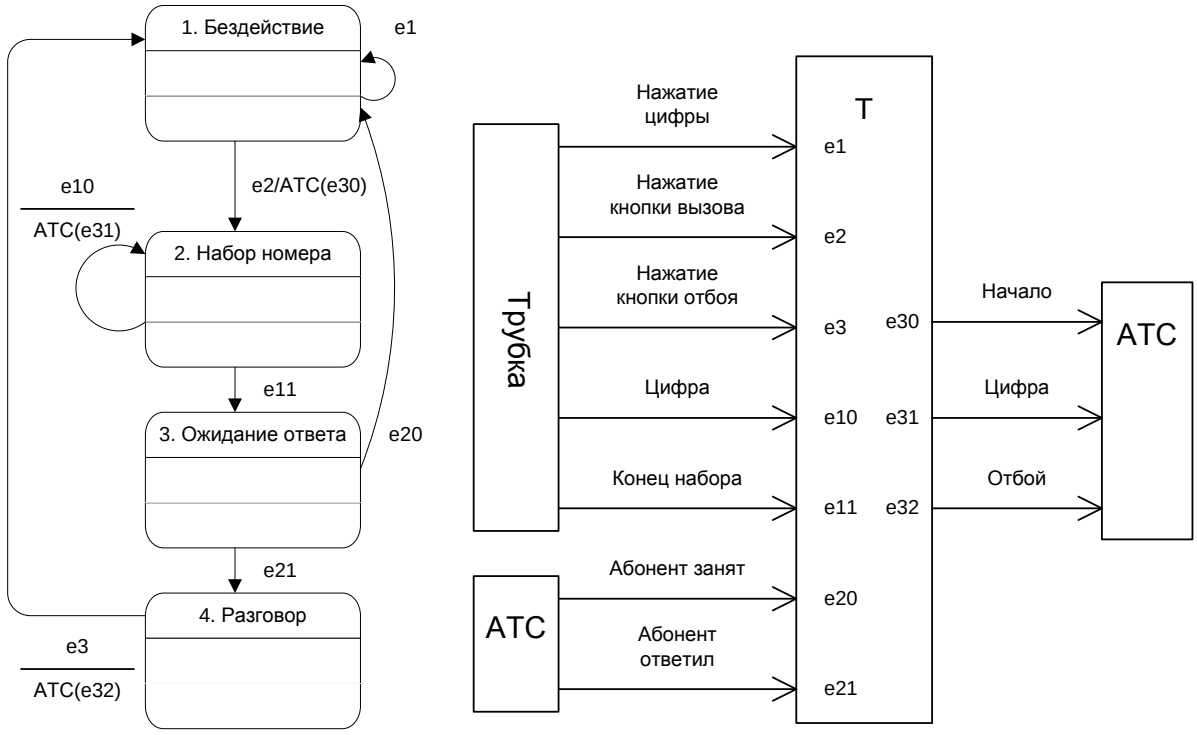

Рис. 1. Диаграмма переходов и схема связей автомата «Телефон»

Представленный автомат обладает ограниченной функциональностью, но в большинстве случаев работает корректно по следующему сценарию:

1. Абонент набирает номер (переходы по событию `e1`).

2. Абонент нажимает кнопку вызова (переход по событию `e2`). Телефон сообщает АТС, что трубка поднята (событие `АТС(e30)`).

3. Телефон начинает обрабатывать цифры, набранные абонентом в пункте 1 (переходы по событию `e10`). О каждой набранной цифре номера телефон сообщает АТС (событие `АТС(e31)`).

4. По окончании набора (событие `e11`) автомат переходит в состояние «Ожидание ответа».

5. В случае если вызываемый абонент занят, автомат возвращается в состояние бездействия (переход по событию `e20`).

6. Если вызываемый абонент ответил (переход по событию `e21`), автомат переходит в состояние «Разговор» и по его завершению (переход по событию `e3`) возвращается в состояние бездействия.

Автомат «Телефон» не во всех случаях работает корректно и может зависать в ожидании событий. Опишем случаи некорректной работы в соответствии с классификацией, данной ранее.

Поздняя подписка. Допустим, что абонент набрал больше цифр, чем необходимо. Если число лишних набранных цифр велико, то, в то время как телефон, находясь в состоянии 2, впустую набирает лишние цифры, вызываемый абонент может успеть отве-

тить и положить трубку. Тогда по событию «Конец набора» телефон, оказавшись в состоянии 3, будет ожидать событие «Абонент ответил», которое уже произошло ранее.

Задержка. Простейший пример задержки – это продолжительное ожидание ответа вызываемого абонента в состоянии 3 «Ожидание ответа».

Голодание. Примером голодания может стать обрыв линий во время набора номера. В этом случае мы не дождемся ответа вызываемого абонента по не зависящей от нас причине.

Блокировка. Блокировка в данном примере возникает, когда набранных цифр недостаточно для вызова абонента. Вместо положенных 7 цифр абонент набрал 6. Набрав короткий номер, автомат окажется в состоянии ожидания ответа, из которого он может выйти, только получив ответ вызываемого абонента или сигнал «занято». В то же время вызываемый абонент не ответит до тех пор, пока не получит от автомата последнюю цифру своего номера.

Необходимо идентифицировать и исправлять перечисленные ошибки проектирования.

Верификация взаимодействия автоматных моделей

Многих перечисленных выше проблем можно избежать на этапе проектирования программы при условии, что взаимодействующие части этих программ реализованы с помощью автоматного подхода.

Одной из эффективных методик анализа взаимодействия автоматов является их перемножение. В книге [2] подробно рассмотрен пример такого анализа, в котором после перемножения трех автоматов – банка, магазина и клиента – автор указал на возможность купить в магазине товар, не заплатив за него деньги. На рис. 2, взятом из книги [2], изображены автоматы «Магазин», «Клиент», «Банк» и их произведение. Состояние `(2, c)` является некорректным, потому что в этом состоянии деньги клиентом не потрачены, а товар получен. Однако это состояние достижимо.

Применим аналогичный подход для выявления некорректных ситуаций зависания автоматов с помощью перемножения автоматов, реализующих логику работы телефона и АТС. На рис. 3 приведены диаграмма переходов и схема связей автомата «АТС». Рассмотрим, как будут взаимодействовать рассматриваемые автоматы.

Построим *автомат-произведение* для этих двух автоматов. Состояниями этого автомата являются пары состояний, первое из которых есть состояние телефона, а второе – состояние АТС. Отметим, что полученный автомат не должен иметь переходов по событиям `e20`, `e21`, `e30`, `e31` и `e32`, так как эти события используются лишь для обозначения взаимодействия автоматов, которое теперь будет закодировано в переходах автомата-произведения.

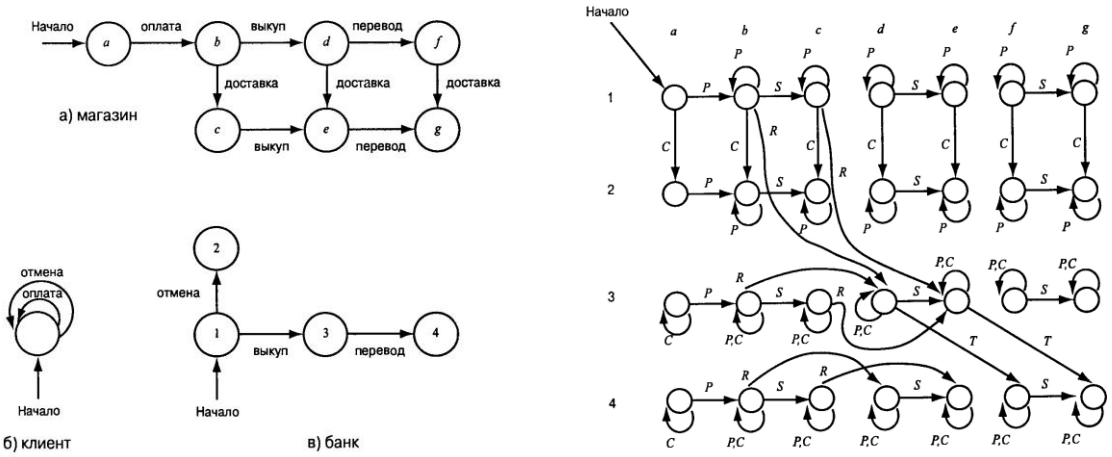

Рис. 2. Автоматы «Магазин», «Клиент» и «Банк» и их произведение [2]

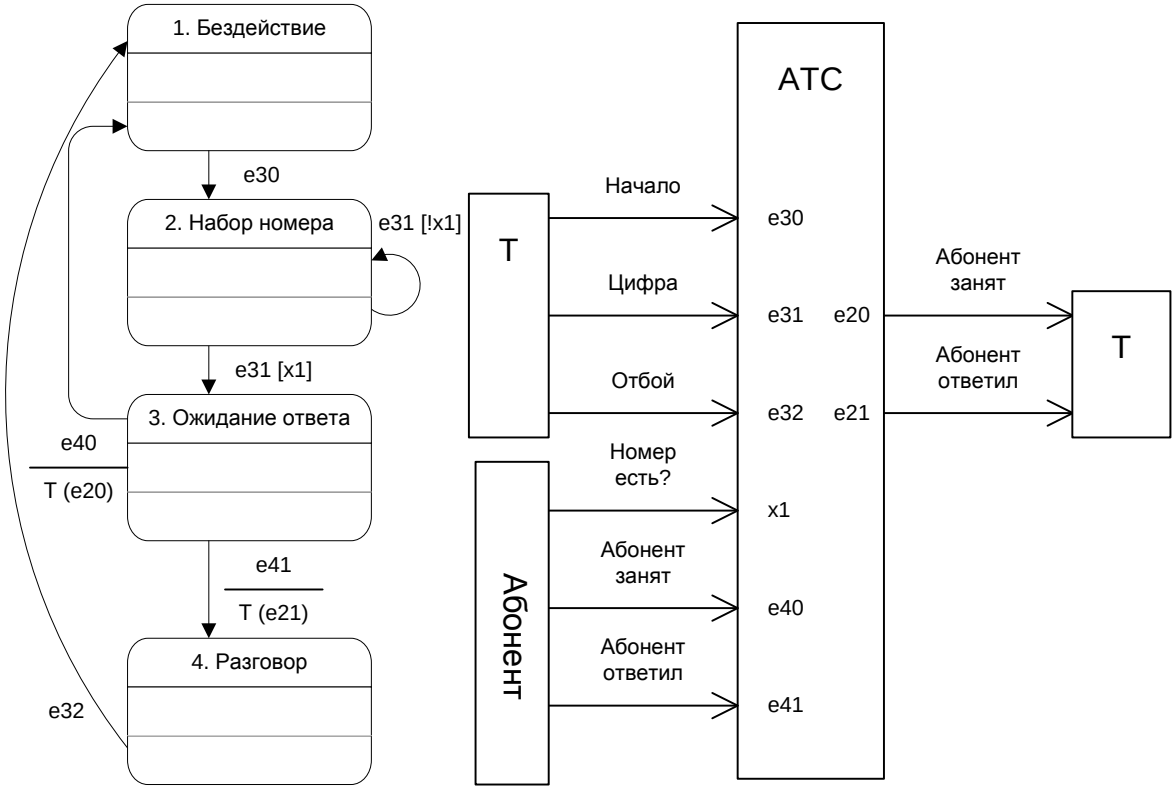

Рис. 3. Диаграмма переходов и схема связей автомата «АТС»

Чтобы правильно построить переходы в автомате-произведении, нужно проследить «параллельную» работу автоматов телефона и АТС. Каждый из двух компонентов автомата-произведения совершает, в зависимости от входных действий, различные переходы. Важно отметить, что если, получив на вход некоторое действие, ни один из этих двух автоматов не может совершить переход по внешнему событию, то автомат-произведение «умирает», поскольку также не может перейти ни в какое состояние.

Правило переходов из одного состояния в другое выглядит следующим образом. Пусть автомат-произведение находится в состоянии `(i, j)`. Это состояние соответствует ситуации, когда телефон находится в состоянии `i`, а АТС – в состоянии `j`. Пусть `e1` означает одно из входных действий. Допустим, автомат телефона или АТС имеет переход из текущего состояния по событию `e1`, и он ведет в состояние `i1` (которое может совпадать с `i`, если автомат, получив на вход `e1`, остается в том же состоянии). Из

списка действий, выполняемых на ребре, выбираются действия посылки событий (пусть это будет `e2`) автомату АТС. Затем, если у автомата телефона есть дуга с меткой `e2`, ведущая в некоторое состояние `j1`, рассматриваются действия на переходе с меткой `e2`. Одно из этих действий может быть действием посылки события обратно телефону. Таким образом, получается последовательность `e1/ATC(e2)/T(e3)/…en`, которая в конечном итоге приводит к состояниям `in` и `jn`. Переход из состояния `(i, j)` в состояние `(in, jn)` по событию `e1` добавляется в автомат-произведение.

В процессе построения автомата-произведения могли получиться ошибки двух типов – состояния, из которых нет выхода и события, посылаемые автомату впустую. На рис. 4 приведено произведение автоматов. Вопросительными знаками отмечены получившиеся терминальные состояния и переходы, действия на которых никто не обрабатывает.

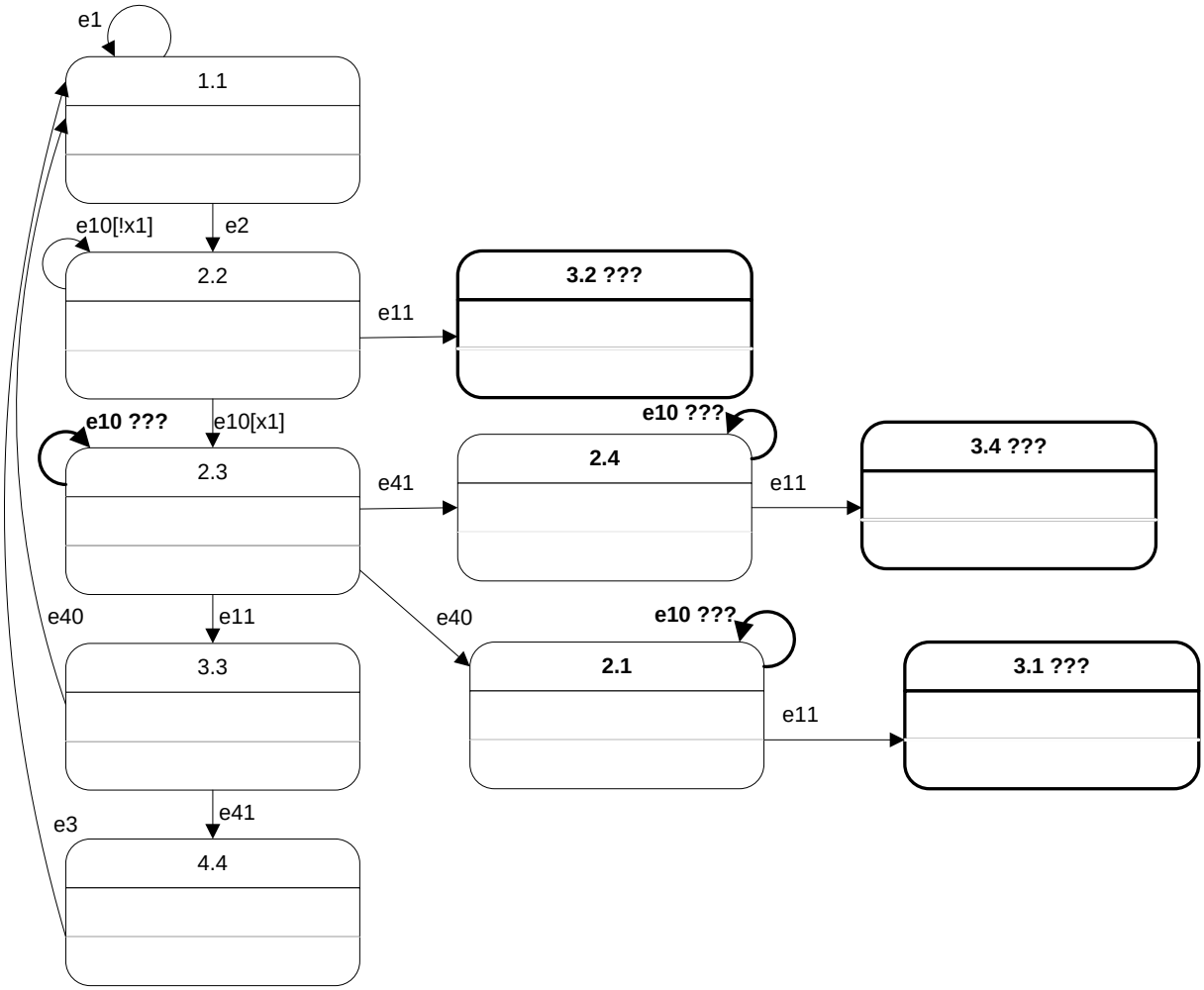

Рис. 4. Произведение автоматов «АТС» и «Телефон»

Полученное произведение автоматов можнет быть использовано как новый автомат, описывающий логику работы сразу двух устройств. У полученного автомата могут быть ошибки двух типов: ребра, посылающие события, которые не могли быть обработаны, и висячие вершины, не являющиеся терминальными состояниями. Посылка событий, которые не могут быть обработаны, говорит о том, что есть вероятность, что произошла поздняя подписка на событие либо событие уже неинтересно. Висячие вершины, не являющиеся терминальными, могут появляться по двум причинам – если

произошла поздняя подписка (на рисунке – состояние `(3, 1)`) или блокировка (состояние `(3, 4)`).

Чтобы исправить эти ошибки, можно изменять исходные автоматы «Телефон» и «АТС» и с каждым изменением получать новый автомат-произведение, который необходимо проверять на наличие ошибок. Однако этот способ достаточно трудоемок. Проще и быстрее исправлять ошибки прямо на получившемся автомате-произведении, не рассматривая исходные автоматы. Этот подход описан в книге [3]. Он хорошо подходит для реализации больших систем сложных автоматов. Сначала можно спроектировать работу каждого автомата по отдельности, а затем, вычислив автомат-произведение, работать уже с одним автоматом, описывающим логику всей системы в целом. В случае телефона и АТС произведение автоматов можно рассматривать так, как если бы АТС непосредственно реагировала на нажатие кнопок (в этом случае можно назвать телефон «тонким клиентом») или телефон умел общаться с базой абонентов и без помощи дополнительных систем соединялся бы с конкретным номером («толстый клиент»).

.

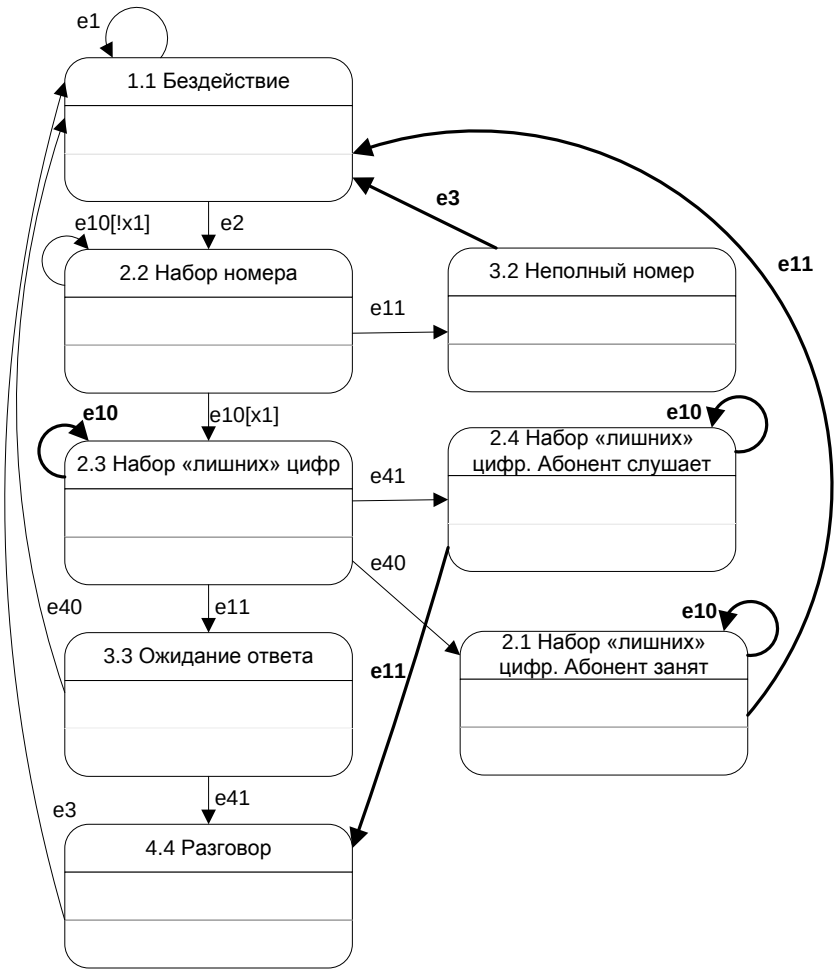

Рис. 5. Исправленное произведение автоматов «АТС» и «Телефон»

Рассмотрим теперь, как можно исправить ошибки в автомате-произведении. Чтобы исключить висячие состояния, не являющиеся терминальными, необходимо все ребра, идущие к этим состояниям, перевести в другие, более подходящие по смыслу состояния. В описанном примере для того, чтобы исключить висячее состояние `(3, 4)`,

необходимо ребро с событием `e11` («Трубка поднята») из состояния `(2, 4)` пустить в состояние `(4, 4)`, минуя висячую вершину. А из вершины `(3, 2)` необходимо добавить ребро по событию `e3` («Нажатие кнопки отбоя») в начальное состояние `(1, 1)`.

Ребра, обработчик которых содержит посылку событий и которые не могут быть обработаны, в представленном случае необходимо просто оставить как есть.

В итоге получается исправленный автомат, описывающий логику всей системы в целом. Он представлен на рис. 5. Жирными линиями на рисунке отмечены ребра, перенаправленные в другие состояния. Состояниям были даны имена в соответствии с логикой работы системы

Важным вопросом при рассмотрении автомата-произведения как нового автомата является сложность автомата. Критерием его сложности может быть, например, количество вершин. Теоретически число вершин равно произведению числда вершин всех автоматов. На практике рассматриваются только достижимые состояния, что часто приводит к сокращению автомата.

В рассмотренном случае перемножения автоматов телефона и АТС получилось 13 вершин вместо 16. Из них действительно имели смысл и рассматриваются в исправленном автомате только 8.

Однако стоит отметить, что при добавлении всего одного ребра, как показано на рис. 6 автомат-произведение увеличивается на 2 вершины, и их количество уже становится близким к максимуму.

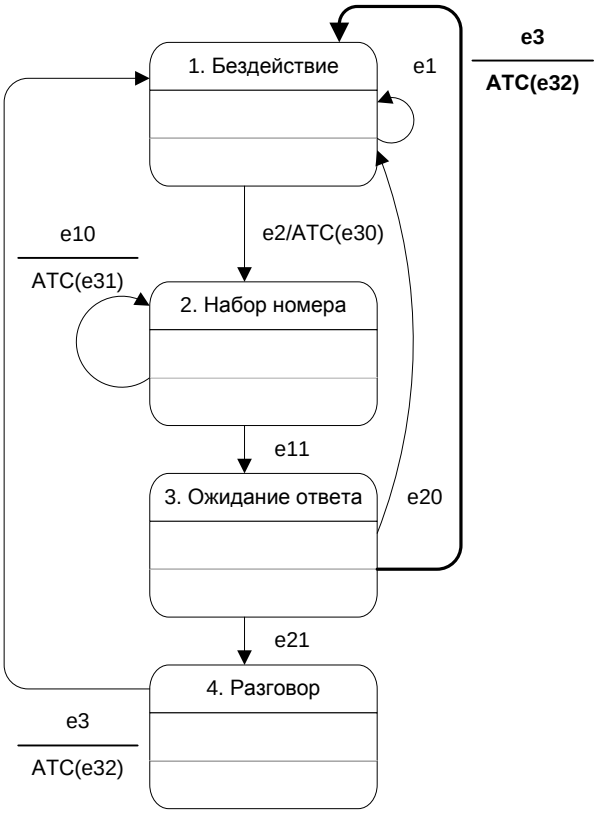

Рис. 6. Измененный граф переходов автомата «Телефон»

Автомат-произведение измененного автомата «Телефон» и автомата «АТС» показан на рис. 7. Отметим, что добавление одного ребра в автомат «Телефон», хоть и уменьшает число висячих нетерминальных состояний, но порождает неопределенность,

связанную с тем, набирает ли АТС новый номер в состоянии `(2, 2)` или продолжает набор после перехода из состояния `(1, 2)`.

Проблема многопоточности при верификации автоматов

Построенный в предыдущем разделе автомат-произведение находит ошибки, допущенные на этапе проектирования автоматов. Для подтверждения этого факта рассматриваемые автоматы были реализованы с помощью технологии *Windows Workflow Foundation (WWF)*. Этим автоматам подавались на вход следующие последовательности событий с различными задержками между событиями (звездочка * обозначает произвольное количество событий):

1. `e1, e2, (e10)*, e11`
4. `x1=false`
5. `x1=true, e40`
6. `x1=true, e41, e3`

При возникновении исключительных ситуаций, когда событие не ожидается, и при попадании в известные терминальные состояния программа продолжала тестирование сначала.

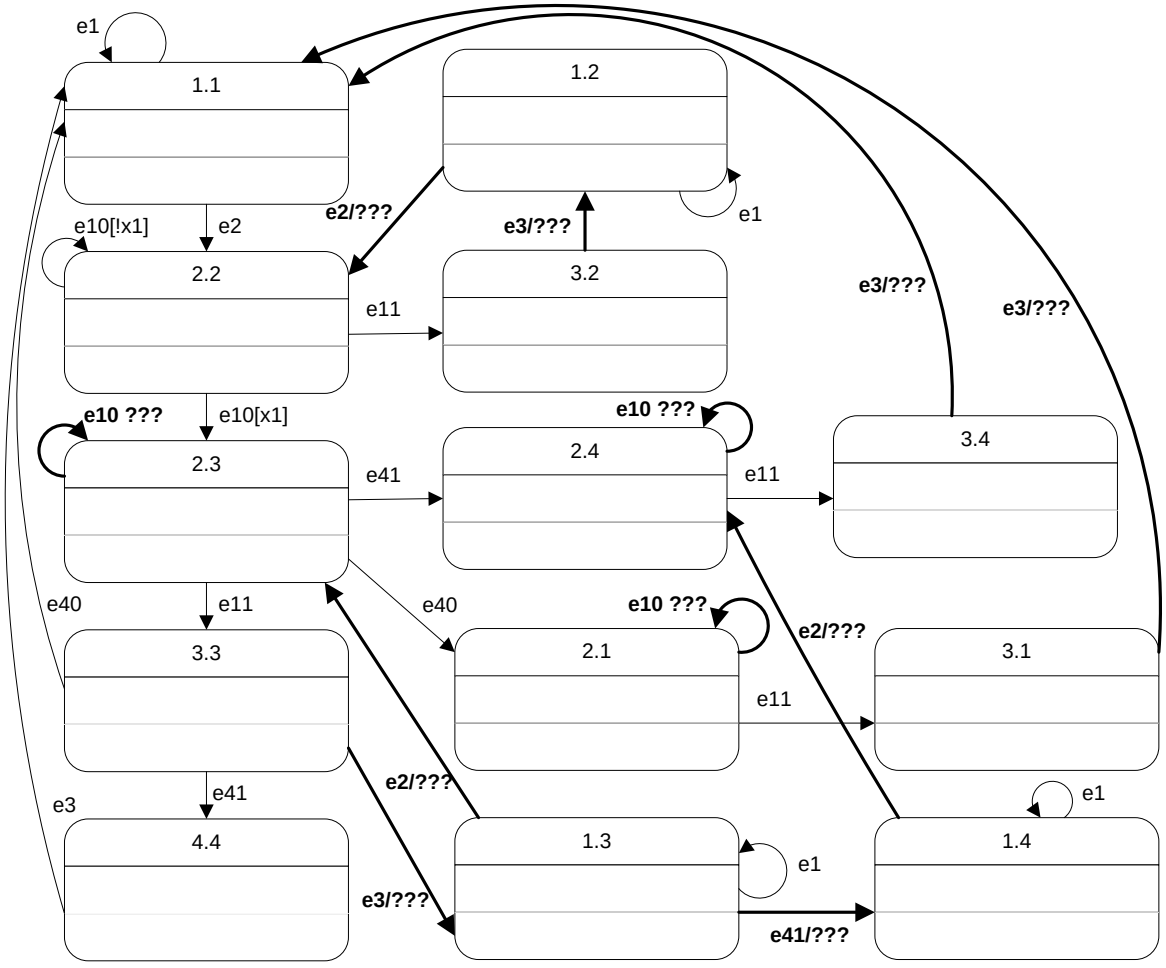

Рис. 7. Произведение измененного автомата «Телефон» и автомата «АТС»

Полученный тест подтвердил, что мы нашли все возможные исключительные состояния. Однако при небольшом изменении автомата, описывающего состояние телефона (рис. 6) и набора подаваемых на вход последовательностей, были найдены новые,

не выявленные автоматом-произведением неожидаемые события – например, событие `e30` в состоянии `(1, 2)`. Рассмотрим более подробно причины возникновения таких событий.

При построении автомата-произведения был использован алгоритм, аналогичный описанному в книге [2]. Однако этот алгоритм основывается на предположении, что за один шаг будут обработаны все ребра всех автоматов, переход по которым вызывает событие, инициировавшее этот шаг.

Однако это предположение неверно для реактивных систем, использующих другую семантику взаимодействия частей автоматных программ. Действительно, в нашем случае автомат, реализующий АТС, может реагировать на события очень медленно, например, в силу большой загрузки. В этом случае абонент успеет набрать номер, нажать кнопку вызова и, не дождавшись никакого ответа, нажать кнопку отбоя. В этом случае у АТС «накопится» список событий `e30`, `(e31)*`, `e32`, который будет постепенно обрабатываться, и может так случиться, что, когда АТС закончит обработку этого списка событий в состоянии `2`, телефон заново поднимет трубку и пришлет событие `e30`.

Отметим, что в случае корректной реализации телефона и АТС, когда нет непредвиденных событий, такое поведение даже более предпочтительно, чем реализация системы в виде автомата-произведения, представленного на рис. 5. В этом случае при большой загрузке АТС у телефона нет необходимости дожидаться, когда АТС обработает события, посланные ей, тогда как автомат-произведение предполагает именно такое поведение.

Таким образом, при верификации реактивных систем, семантика которых сходна с семантикой WWF, построение автомата-произведения становится затруднительным, и необходимо применять другой способ их верификации.

Семантика взаимодействия реактивных систем, реализованных с помощью автоматов

Выше рассмотрены две семантики взаимодействия реактивных систем, реализованных с помощью конечных автоматов. В первом случае все автоматы обрабатываются в одном потоке, реагирующем на внешние события. Во втором случае каждый автомат представляет собой отдельный поток, и внешние события для него не отличаются от внутренних, необходимых только для обозначения взаимодействия автоматов. Рассмотрим подробнее каждую из этих двух семантик взаимодействия.

В реализации семантики взаимодействия автоматов, когда все автоматы исполняются в одном потоке, исполнение автоматов происходит по следующему алгоритму. При поступлении внешнего события оно ставится в очередь. Начинается его обработка. Сначала происходит поиск автомата, который ждет этого события. Начинается обработка действий на соответствующем ребре. Если требуется обработать посылку внутреннего события, автомат переводится в состояние, в которое ведет ребро, а затем вызывается обработчик внутреннего события.

Отметим ограничения, которые накладываются при такой семантике взаимодействия автоматов для того, чтобы гарантировать однозначность обработки событий.

1. Каждое событие может быть обработано только одним из автоматов.
2. Обработчик любого ребра может содержать только один вызов автомата.

Эта реализация взаимодействия автоматов позволяет использовать автомат-произведение для верификации взаимодействия автоматов.

Другая реализация взаимодействия автоматов заключается в том, что все события – внешние и внутренние обрабатываются в одной очереди. При этом автоматы реализуются как независимые потоки. В этом случае, как было показано выше, нельзя

применять способ верификации взаимодействия автоматов, основанный на построении автомата-произведения. Также в этом случае некорректным будет проверка текущего состояния автомата (`y1 == 1`). Вместо таких проверок необходимо использовать события-нотификации о переходе в конкретное состояние.

Вопрос верификации взаимодействия реактивных систем с подобной семантикой взаимодействия рассмотрен в работе [5]. В этой работе предлагается использовать протоколы взаимодействия автоматов – для каждого автомата вычисляются возможные последовательности приходящих и исходящих событий. Затем происходит сравнение протоколов для обнаружения «нестыковок», когда протокол одного автомата не может быть обработан другим автоматом. Подобный подход был применен при тестировании автоматов, описанном выше, когда автоматам на вход подавались всевозможные последовательности событий в произвольном порядке.

Проблема атомарности обработки события в реактивных системах

Как отмечалось в работе [4], диаграмма переходов автомата проще всего реализуются в коде программы с помощью оператора `switch`. Отсюда и происходит одно из названий автоматного подхода – `SWITCH`-технология. Однако многие реализации автоматов не рассчитаны на работу в многопоточной среде. Это также, наряду с плохим проектированием, может привести к проблеме бесконечного ожидания события.

Вернемся к примеру на рис. 1. Допустим, автомат реализован в соответствии с классической нотацией [4]. В этом случае, если мы переходим по ребру из состояния 3 («Ожидание ответа») в состояние 4 («Разговор») и одновременно с этим к нам приходит событие «Кладем трубку», то мы можем начать обрабатывать его, подразумевая, что мы находимся еще в состоянии 3. Такого поведения легко избежать, используя блокировки, гарантирующие атомарность операции перехода из одного состояния в другое. Другой подход описан в работе [5] – для блокировки и корректной работы многопоточной системы использовалась очередь событий для каждого автомата. Также отсутствие такого поведения гарантируется при использовании инструментального средства *UniMod* [6] в режиме обработки событий Queue.

Стоит отметить, что в случае реализации автоматов с помощью библиотеки *Windows Workflow Foundation* [7] следует помнить об отсутствии гарантии такой атомарности. *Windows Workflow Foundation* поддерживают два способа подписки на события:

1. прямая подписка на события через стандартные средства языков *.NET*;
2. подписка через сервис `ExternalDataExchangeService`.

Только при использовании сервиса `ExternalDataExchangeService` гарантируется атомарность перехода по событию и корректная последовательность обрабатываемых событий.

Заключение

В работе приведена классификация возможных причин долгого ожидания событий автоматом. Приведен способ обнаружения такого поведения с помощью автомата-произведения, а также с помощью протоколов событий. Также указаны ограничения, накладываемые на реализацию автомата, при выполнении которых такая верификация возможна.

Литература

1. Вельдер С.Э., Шалыто А.А. Введение в верификацию автоматных программ на основе метода Model Checking. – Режим доступа: http://is.ifmo.ru/verification/modelchecking/
2. Хопкрофт Д., Мотвани Р., Ульман Д. Введение в теорию автоматов, языков и вычислений. – СПб.: Вильямс, 2002, 528 с.
3. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. – СПб.: Наука, 1998. – 628 с.
4. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем // Программирование. – 2001. – № 5. – Режим доступа: http://is.ifmo.ru/works/switch/1/.
5. Канжелев С.Ю., Шалыто А.А. Моделирование кнопочного телефона с использованием SWITCH-технологии. Вариант 2. – Режим доступа:http://is.ifmo.ru/projects/phone/.
6. Гуров В. С., Мазин М. А., Шалыто А. А. UniMod - программный пакет для разработки объектно-ориентированных приложений на основе автоматного подхода // Труды XI Всероссийской научно-методической конференции "Телематика-2004". – 2004. – Т.1 – Режим доступа:http://tm.ifmo.ru.
7. Эспозито Д. Cutting Edge: Windows Workflow Foundation // MSDN Magazine. – 2006. – № 5.