Санкт-Петербургский государственный университет
информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра «Компьютерные технологии»

А.В. Шестаков

Разработка методов модификации автоматных программ при изменении сценариев их работы

Бакалаврская работа

Научный руководитель –
Ф.Н. Царев

Санкт-Петербург
2011

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Одним из подходов к программированию объектов со сложной логикой является автоматное программирование или «программирование с явным выделением состояний» [1]. При использовании этого подхода логика программы задается в виде управляющего автомата. Состояния автомата соответствуют состояниям реализуемого объекта, переходы между состояниями осуществляются при выполнении заданных условий. При таком подходе облегчаются тестируемость программы, проверка соответствия спецификации, а визуализация автоматов в виде графов переходов позволяет облегчить поддержку программы.

Автоматное программирование хорошо подходит для создания событийно-ориентированных приложений, когда действия реализуемого объекта зависят от внешних событий [2, 3]. По каждому событию объект выполняет определенные действия, последовательность событий порождает последовательность действий. Такая цепочка пар «событие – действие» называется сценарием работы автоматной программы. Сценарий работы описывает поведение автомата при определенной последовательности событий.

В процессе разработки приложения может возникнуть ситуация, когда внешние условия несколько изменились, вызывая тем самым необходимость изменения логики программы. Обычно это требует внесения изменений в код и может повлечь за собой появление ошибок. Использование парадигмы автоматного программирования [4] позволяет избежать ошибок на уровне генерации кода, но остается проблема с возможным появлением ошибок в логике работы. В данной работе предлагается подход, позволяющий контролировать изменение логики программы.

Целью работы является разработка метода, позволяющего автоматически модифицировать автоматную программу при изменении сценариев работы. Это позволяет избежать ошибок на этапе внесения изменений в программу, а следовательно, увеличивает надежность конечного продукта. Одним из основных требований к этому методу является минимальность изменения автоматной программы. Выполнение этого требования гарантирует сохранение общей структуры автомата, что может быть полезно при внесении изменений в программу, написанную вручную, а не сгенерированную на основе сценариев.

ГЛАВА 1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

«Одна из центральных идей автоматного программирования состоит в отделении описания логики поведения от описания его семантики» [1]. Логика программы задается с помощью детерминированного конечного автомата, который формально определяется как пятерка $< X, Y, \sigma, y_0, F >$, где X – конечный алфавит входных символов, Y – конечное множество состояний, $\sigma: X \times Y \to Y$ – функция переходов, y_0 – начальное состояние, $F \subset Y$ – множество допускающих состояний. В данной работе в качестве автоматной модели используется автомат Мура второго рода [1]. Автомат этого типа при получении входного воздействия обновляет свое состояние и на основе этого состояния формирует выходное воздействие. Пример автомата Мура второго рода приведен на рис. 1.

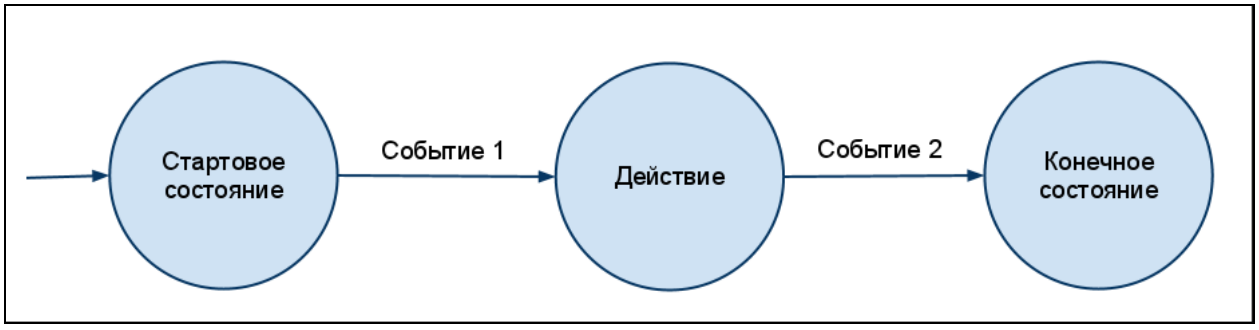

Рис. 1. Пример автомата Мура – в состоянии выполняется действие

Таким образом, используемую модель автомата можно представить в виде графа переходов, в котором переходы осуществляются по событиям и определены стартовое состояние и одно или несколько конечных.

Сценарием работы (далее сценарий) называется список пар $< e, a >$, где e – событие, a – действие автомата. Сценарий работы определяет

поведение программы при появлении определенной последовательности событий.

Таким образом, логику работы программы можно задать с помощью множества сценариев, и наоборот – имея заданную в форме автомата логику программы можно получить множество рабочих сценариев (возможно, бесконечное). При изменении требований к программе изменяется множество сценариев. Задача состоит в том, чтобы по измененным сценариям работы определить необходимые изменения в управляющем автомате.

1.2. Классификация изменений сценариев работы

Далее будем называть рабочим множеством – множество сценариев определенное пользователем, причем автомат должен удовлетворять сценариям из этого множества. Сценариям, не принадлежащим рабочему множеству, автомат может, как удовлетворять, так и не удовлетворять.

Рабочее множество может быть изменено несколькими способами:

1. Добавление сценария.
2. Удаление сценария.
3. Изменение существующего сценария.

При удалении сценария он исключается из множества, автомат можно не изменять. При добавлении сценария может возникнуть ситуация конфликта. Конфликтом (рис. 2) далее называется ситуация, когда требуется добавить переход из заданного состояния по событию e в состояние с действием a_1, но уже существует переход в состояние с действием a_2.

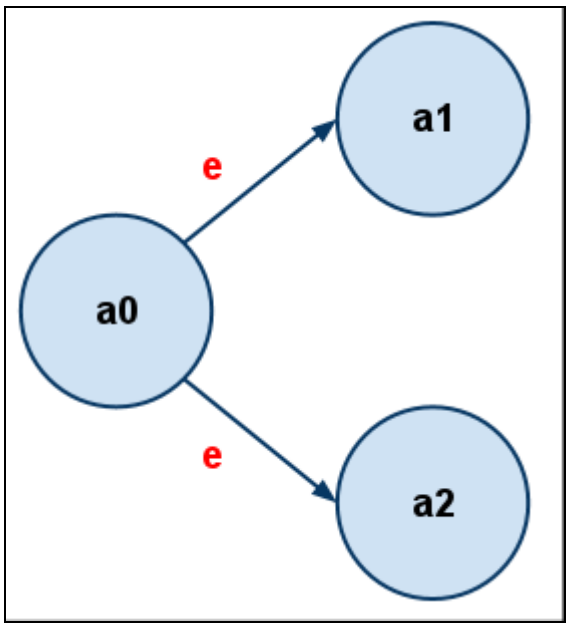

Рис. 2. Ситуация конфликта сценариев

Добавить сценарий, конфликтующий с хотя бы одним сценарием из рабочего множества, невозможно. Однако, если добавляемый сценарий конфликтует с автоматом и не конфликтует со сценариями рабочего множества, то его можно добавить, предварительно изменив автомат с помощью рефакторинга [5]. Если же конфликтов нет, то сценарий можно добавить с помощью предложенного в данной работе алгоритма. Изменение существующего сценария можно свести к операциям удаления старого сценария и добавления нового.

1.3. Существующие методы модификации автоматных программ

Большая часть существующих методов не модифицируют исходный автомат, а строят новый на основе определенного множества сценариев. Для построения автомата успешно используются генетические алгоритмы [6, 7]. Этот подход обеспечивает соответствие генерируемого автомата набору сценариев, однако, так как для работы алгоритма не требуется знание структуры исходного автомата (его может вообще не быть), то трудно гарантировать, что результирующий автомат будет похож на исходный.

Если же в качестве исходных данных задан автомат, то новый автомат, удовлетворяющий изменившимся сценариям, можно построить, опять-таки

используя генетические алгоритмы. В качестве функции приспособленности можно задать не только соответствие автомата сценариям, но и степень изменения относительно исходного автомата. Этот подход решает задачу, поставленную в данной работе, однако из-за неявности используемого алгоритма нельзя заранее определить время его работы. В отличие от генетических, использование явных алгоритмов позволяет оценить время работы программы.

Таким образом, ключевыми отличиями данной работы от других работ в этой области являются:

1. Требование минимальности изменения автомата.
2. Использование явных алгоритмов для изменения автомата.

ГЛАВА 2. МЕТОДЫ МОДИФИКАЦИИ АВТОМАТНЫХ ПРОГРАММ

2.1. Рефакторинг автоматных программ

Одной из подзадач данной работы является добавление сценария, конфликтующего с автоматом. В качестве примера предложен простой автомат, изображенный в виде графа переходов на рис. 3.

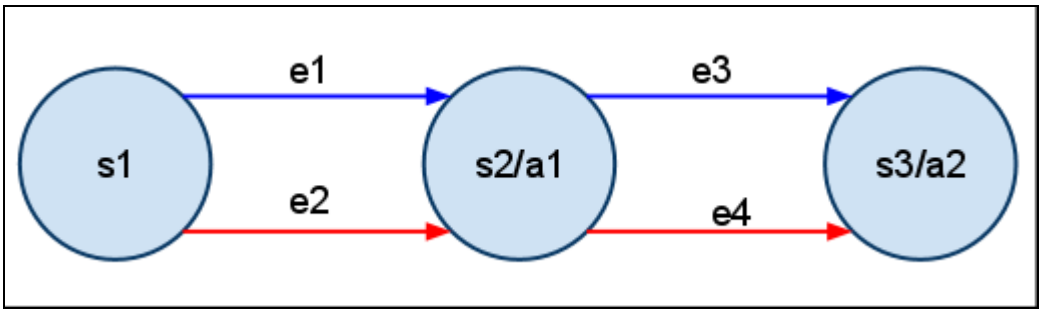

Рис. 3. Ситуация конфликта между добавляемым сценарием и автоматом

Пусть в рабочее множество входят сценарии $e_1 a_1 e_3 a_2$, $e_1 a_1 e_4 a_2$, $e_2 a_1 e_3 a_2$. Дополнительно этот автомат содержит сценарий, не входящий в рабочее множество – $e_2 a_1 e_4 a_2$, который надо добавить. При добавлении возникает конфликт:

1. Необходимо добавить переход $s_2 \rightarrow e_4 \rightarrow s_4$, где s_4 – состояние с действием a_3.
2. Существует переход $s_2 \rightarrow e_4 \rightarrow s_3$.
3. Переход $s_2 \rightarrow e_4 \rightarrow s_3$ нельзя удалить, так как он входит в один из сценариев рабочего множества.

Для решения возникшей проблемы можно применить рефакторинг «разбиение состояния по входящим переходам». Рефакторингом автомата называется «изменение в программе, имеющее целью облегчить понимание ее работы и упростить модификацию, не затрагивая наблюдаемого поведения» [8]. Методика разбиения состояния:

1. Пусть у состояния s есть входящие переходы $t_1 \dots t_n$, где $n > 1$.
2. Требуется разбить состояние s на два состояния s_1 и s_2, которые обладают входящими переходами соответственно $t_1 \dots t_m$ и $t_{m+1} \dots t_n$, где $1 < m < n$. Логика автомата при этом измениться не должна.
3. Состояние s копируется в состояние s_2. При копировании копируются исходящие переходы и действие на входе в состояние.
4. Переходам $t_{m+1} \dots t_n$ назначается новое конечное состояние s_2.
5. Состояние s переименовывается в s_1 (опционально). Рефакторинг закончен.

После разбиения состояния автомат сохранил исходные сценарии работы. Новых сценариев не добавилось. Следовательно, это действие не изменяет логику программы и является рефакторингом. Пример проведения рефакторинга «разбиение состояния» приведен на рис. 4.

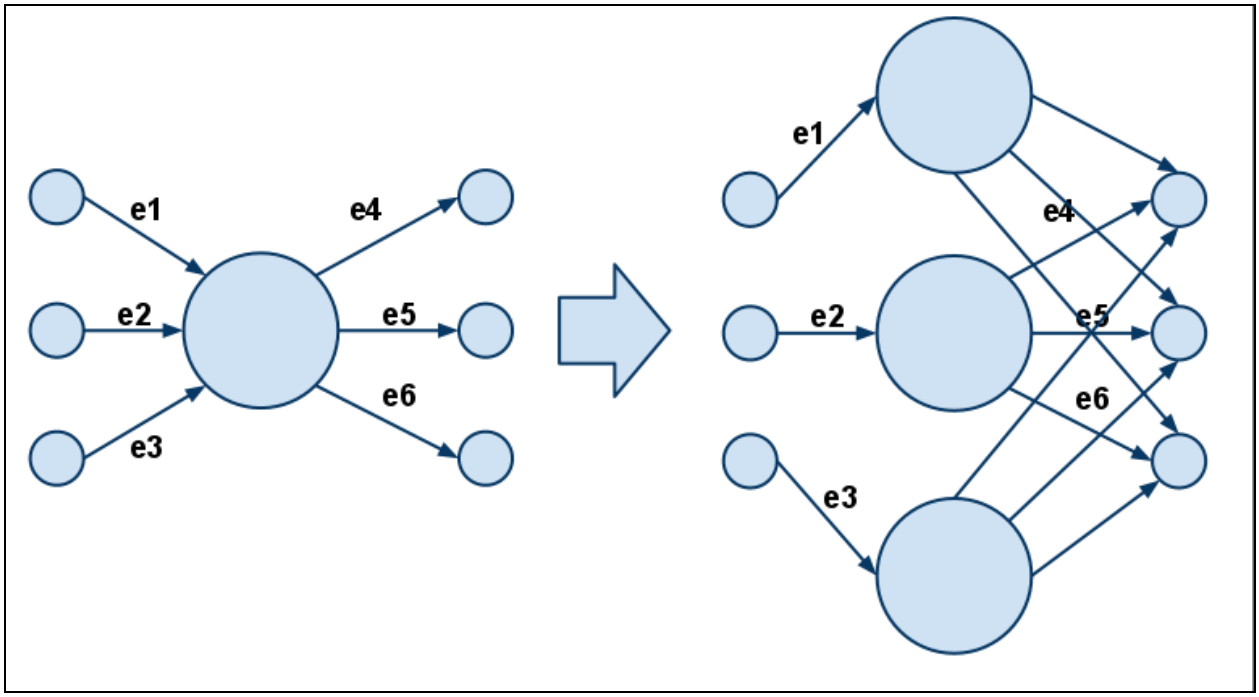

Рис. 4. Пример проведения рефакторинга «разбиение состояния»

Теперь необходимо применить этот рефакторинг к состоянию s_2 из примера. Полученный автомат приведен на рис. 5. В нем состояние s_2 разбито на состояния s_6 и s_7. Теперь можно удалить переход $s_7 \rightarrow e_4 \rightarrow s_3$, добавить состояние s_4/a_3 и добавить новый переход $s_7 \rightarrow e_4 \rightarrow s_4$. Результат приведен на рис. 6.

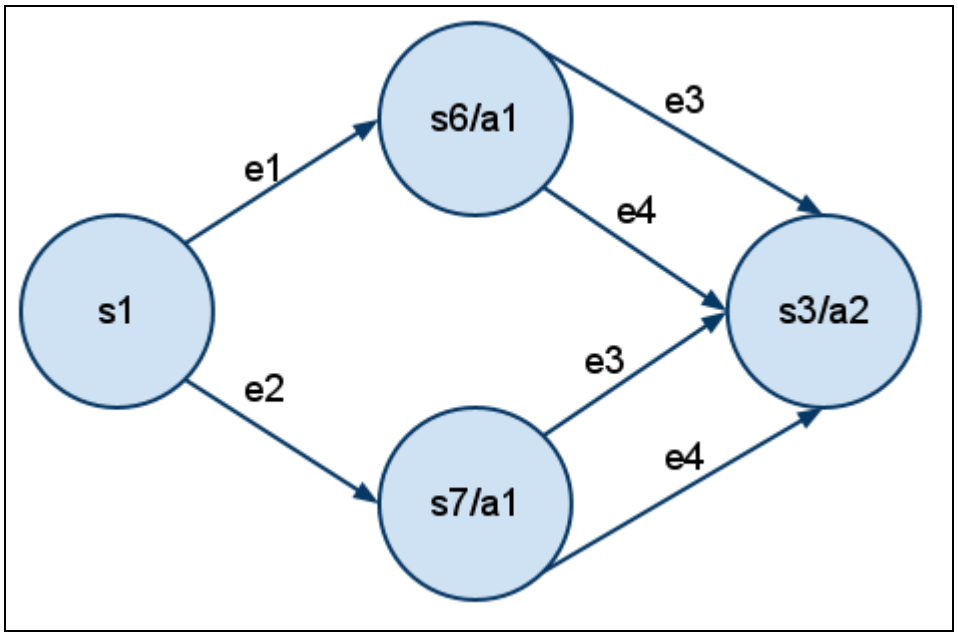

Рис. 5. Автомат после рефакторинга

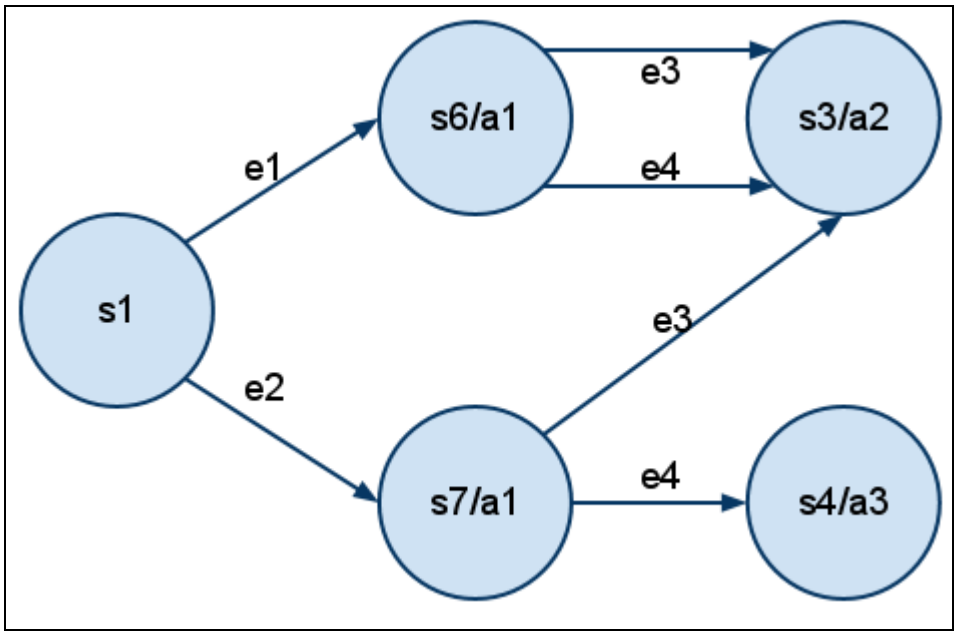

Рис. 6. Результат добавления конфликтного сценария

Общий алгоритм действий в случае добавления сценария при условии конфликта описан далее.

2.2. Предварительное изменение автомата при добавлении конфликтного сценария

Пусть в автомат необходимо добавить сценарий $sc = \{e_i, a_i\}_{i=1..l}$, но при добавлении возникает конфликт. Пусть при этом в рабочем множестве нет сценария $sc1 = \{e_i, a_i\}_{i=1..k}$, такого, что $sc.a_i = sc1.a_i, sc.e_i = sc1.e_i, sc.e_t = sc1.e_t, sc.a_t \neq sc1.a_t$ для некоторого $t > 0, \forall i < t$, и нет сценария, являющегося точным префиксом добавляемого сценария. Утверждается, что в таком случае можно изменить автомат таким образом, чтобы появилась возможность добавить требуемый сценарий. При этом сценарии из рабочего множества не пострадают.

Алгоритм удаления конфликта.

1. Проверить, что при добавлении сценария возникает конфликт с автоматом.
2. Проверить, что в рабочем множестве нет сценариев, совпадающих с добавляемым сценарием по префиксу указанного вида.
3. Найти среди сценариев рабочего множества сценарий, путь которого по автомату совпадает с путем добавляемого сценария по суффиксу максимальной длины. Для примера рассмотрим рис. 7. Пусть в рабочем множестве автомата содержатся сценарии b, c. Сценарий a необходимо добавить. Переход, отмеченный пунктиром, является конфликтным. Тогда сценарием с указанным свойством будет сценарий b.

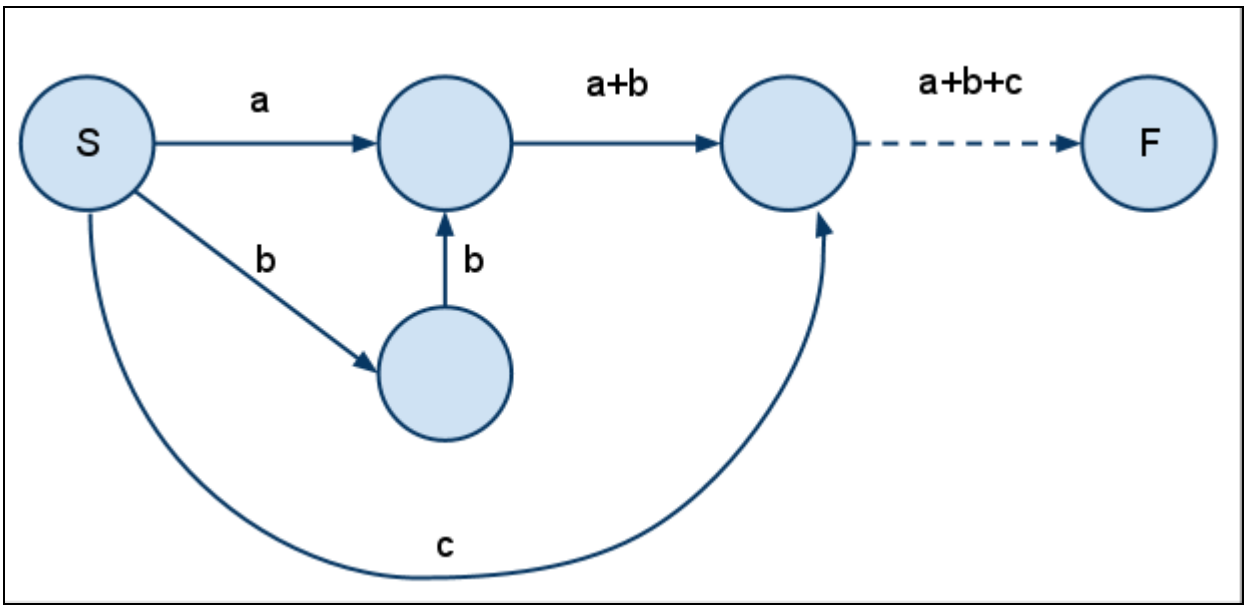

Рис. 7. Пример к третьему пункту алгоритма удаления конфликта

4. Пусть путь найденного сценария совпадает с путем добавляемого сценария по суффиксу, состоящему из состояний $s_1, s_2, \dots, s_n$. К этим состояниям необходимо применить рефакторинг «разбиение состояния» по входящему переходу, для того чтобы разделить конфликтующие пути.
   4.1. Начальные значения переменных: $currentState = s_0$ – состояние, предшествующее состоянию s_1 в пути добавляемого сценария, i – номер пары $< e_i, a_i >$ из добавляемого сценария, соответствующей переходу $s_0 \to e_i \to s_1$.
   4.2. Если из состояния $currentState$ существует переход по событию e_i и переход осуществляется в состояние $tarState$ с действием на входе a_i, то применить рефакторинг «разбиение состояния» к состоянию $tarState$. Разбиение осуществить по ребру $currentState \to e_i \to tarState$. Обновить переменные: $currentState = tarState, i = i + 1$, перейти к началу шага.
   4.3. Если из состояния $currentState$ существует переход по событию e_i и переход осуществляется в состояние $tarState$ с действием на входе $a_j \neq a_i$, то удалить ребро $currentState \to e_i \to tarState$.

Пусть добавляемому сценарию в автомате соответствовал путь по состояниям $\{s_1, s_2, s_3, \dots, s_k\}$, где из состояния s_k требовалось сделать конфликтный переход. В результате работы алгоритма в автомате появился новый путь $\{s_1, s_2, \dots s_i, s'_{i+1}, \dots, s_k{}'\}$, соответствующий добавляемому сценарию (рис. 8). Состояния этого пути являются результатом разбиения $s_j \to s'_j + s''_j$. Так как добавляемый сценарий не совпадает по префиксу указанного вида ни с одним из сценариев рабочего множества, то переход $s'_k \to e_k \to \cdots$ не принадлежит ни одному из сценариев рабочего множества и может быть удален.

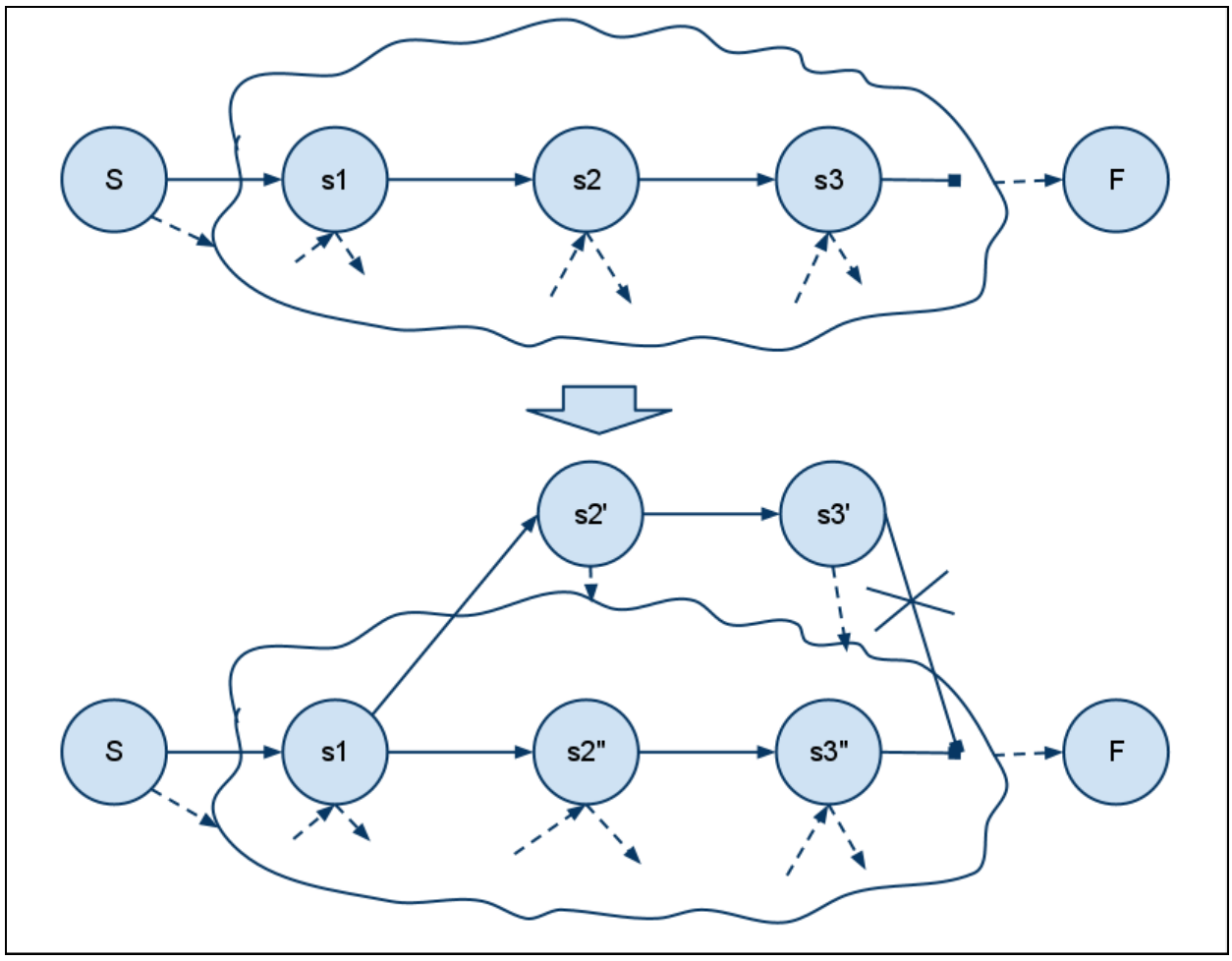

Рис. 8. Изображен автомат до и после работы алгоритма. Пунктир обозначает несколько переходов, сплошная линия – переходы добавляемого сценария

Таким образом, данный алгоритм позволяет избавиться от конфликта, если это возможно, однако он не оптимален с точки зрения минимальности

изменения автомата, так как в результате разбиения добавляются новые состояния. Для уменьшения числа добавленных состояний удаляем те из них, которые не используются сценариями рабочего множества.

Можно заметить, что разбиение любого состояния, кроме s_1 на первом шаге четвертого пункта, не приведет к разделению путей добавляемого сценария и сценария, найденного на третьем шаге. Следовательно, это состояние необходимо разбить. Таким же образом можно показать, что необходимо разбить и состояния $s_2, \dots, s_k$.

Описанный алгоритм устраняет конфликт, добавляя минимально возможное число новых состояний. Его реализация на языке *Java* приведена в приложении.

Оценим время работы алгоритма. Проверка на конфликт выполняется за время $O(l)$, где l – длина добавляемого сценария. Проверка того, что этот конфликт устранимый, выполняется за время $O(m)$, где m – сумма длин всех сценариев рабочего множества. Выполнение третьего пункта можно осуществить также за время $O(m)$. Разбиение состояний выполняется за время $O(l)$. Для удаления неиспользуемых новых состояний необходимо, чтобы были помечены ребра, принадлежащие хотя бы одному из путей, соответствующих сценариям рабочего множества. Это можно сделать, перебрав все пути за время $O(m)$. В результате, оценка времени работы алгоритма $O(l + m)$, где l – длина добавляемого сценария, m – сумма длин сценариев из рабочего множества.

2.3. Добавление сценария

В этой части работы описан алгоритм добавления сценария при условии отсутствия конфликта. Суть алгоритма состоит в сведении исходной задачи модификации автомата к хорошо изученной задаче поиска кратчайшего пути.

Требуется убедиться, что сценарий не конфликтует со сценариями из рабочего списка. Если конфликт есть, то требуется устранить его ранее описанным методом. Алгоритм выявления конфликта:

1. Пусть необходимо добавить сценарий $sc = \{e_i, a_i\}_{i=1..n}$.
2. Текущее состояние $currentState = startState$ (стартовое состояние). Номер рассматриваемого элемента сценария $pairNum = 1$.
3. Если $pairNum$ совпадает с длиной сценария n и текущее состояние выделено как конечное, то автомат уже удовлетворяет сценарию, и добавление не требуется.
4. Если $pairNum$ совпадает с длиной сценария n и текущее состояние не является конечным, то сценарий является префиксом другого сценария.
5. Если существует исходящий переход из currentState по событию $\mathrm{e}_{\mathrm{pairNum}}$ в состояние $state$ с действием на входе $\mathrm{a}_{\mathrm{pairNum}}$, то $currentState = state, pairNum = pairNum + 1$. Переход к шагу 3.
6. Если существует исходящий переход из $currentState$ по событию $\mathrm{e}_{\mathrm{pairNum}}$ в состояние $state$ с действием на входе $a \neq \mathrm{a}_{\mathrm{pairNum}}$, то обнаружен конфликт.
7. Если не существует исходящего перехода из $currentState$ по событию $\mathrm{e}_{\mathrm{pairNum}}$, то конфликта нет, и можно добавлять сценарий.

Результатом проверки является либо сообщение о том, что сценарий в автомат добавить нельзя, либо хвост исходного сценария $subSc = \{e_i, a_i\}_{i=m..n}$ и состояние s, из которого требуется этот остаток добавить в автомат. Для удобства далее считается, что задача состоит в добавлении в автомат сценария $subSc$, пары которого заново

перенумерованы. Следовательно, $subSc = \{e_i, a_i\}_{i=1..l}, l = n - m,$ а стартовым состоянием считается состояние s.

Следующим шагом является построение ациклического ориентированного взвешенного графа по сценарию и автомату. Ориентированным графом называется упорядоченная пара $G =< V, A >$, где V – непустое множество вершин, A – множество упорядоченных пар, называемых ребрами графа. Граф называется взвешенным, если каждому ребру соответствует число w, называемое весом ребра. Граф называется ациклическим, если он не содержит циклов.

Для построения графа, рассмотрим список действий добавляемого сценария $\{a_i\}_{i=1..n}$. Каждому действию a_i сопоставим список состояний автомата, при входе в которые выполняется это действие: $a_i \to \{s_1^i, s_2^i, \ldots\}$. Далее список состояний, соответствующий i-му действию сценария, будем называть i-м слоем.

Пояснение. Для того чтобы добавить сценарий, необходимо построить непротиворечивый путь через эти слои, можно использовать только одно состояние из слоя (рис. 9).

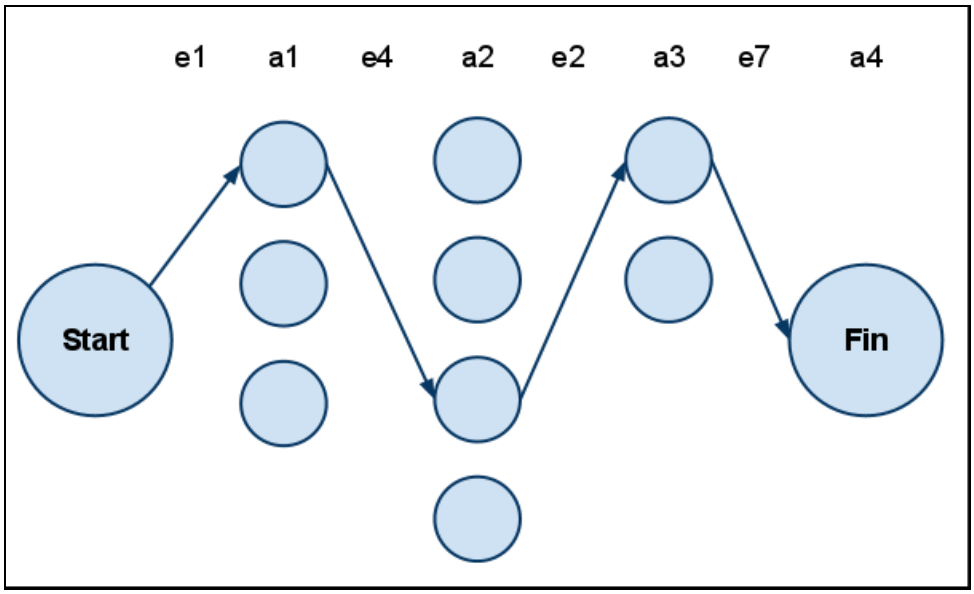

Рис. 9. Пояснение смысла построения графа

Для каждого действия может использоваться не только состояние из автомата, но и новое созданное состояние. Пусть действие a_i встречается в

сценарии несколько раз – $a_i = a_j, i \neq j$. Если предположить, что для реализации действия a_i было использовано новое состояние ns_1, то это же состояние может быть использовано и для реализации действия a_j. Однако возможна ситуация, когда повторное использование состояния невозможна. Пусть был необходим переход $a_i \to e \to a_x$, и для реализации действия a_i было выбрано состояние s. Теперь, если потребуется сделать переход $a_j \to e \to a_y, a_j = a_i, a_x \neq a_y$, то использовать состояние s для реализации a_j нельзя. Для того чтобы избежать ситуации, когда для реализации действия невозможно выбрать состояние из списка, в каждый список добавляется k новых состояний, где k – число раз, когда пара a_i, e_i встречалась в сценарии. Следовательно, если $< a_i, e_i >=< a_j, e_j >$, то:

$$a_i \to \{s_1^i, s_2^i, \dots, ns_1^i\}$$

$$a_j \to \{s_1^j, s_2^j, \dots, ns_1^j, ns_2^j\}$$

Здесь верхний индекс обозначает номер слоя, а нижний – номер состояния. При этом ns_1^j, ns_1^i указывают на одно и то же состояние, хотя находятся в разных слоях. Также следует заметить, что для последнего действия сценария состояния могут быть выбраны только из списка конечных состояний автомата.

Построенные списки состояний используются для построения графа. Каждый список рассматривается как множество вершин, каждой из которых соответствует либо состояние автомата, либо новое состояние, которое возможно придется добавить в автомат. В граф добавляется стартовое состояние s, полученное на этапе выявления конфликта.

Теперь необходимо добавить в граф ребра. Рассматривается каждая вершина графа v. Пусть этой вершине соответствует состояние s исходного автомата. Ребра добавляются по следующим правилам.

Если v – стартовая вершина, то надо добавить ребра из v во все вершины первого слоя. Вес каждого ребра устанавливается в единицу.

Если вершина v лежит в i-м слое, ей соответствует состояние s исходного автомата и у состояния s существует исходящий переход $s \to e_{i+1} \to s_{ai+1}$, где s_{ai+1} – некоторое состояние с действием a_{i+1} на входе, то надо добавить ребро $v \to w$, где w – вершина из $i+1$-го слоя, соответствующая состоянию s_{ai+1}. Вес этого ребра устанавливается в ноль.

Если вершина v лежит в i-м слое, ей соответствует состояние s исходного автомата и у состояния s существует исходящий переход $s \to e_{i+1} \to s_{ax}$, где s_{ax} – некоторое состояние с действием $a \neq a_{i+1}$ на входе, то никакое ребро не добавляется.

Если вершина v лежит в i-м слое, ей соответствует состояние s исходного автомата и у состояния s нет исходящего перехода по событию e_{i+1}, то добавляются ребра из v во все вершины следующего слоя. Вес ребер устанавливается в единицу.

Если вершина v лежит в i-м слое, ей соответствует новое состояние ns, не представленное в исходном автомате, то добавляются ребра из v во все вершины следующего слоя. Вес ребер устанавливается в $1 + W$, где W – некоторое положительное число. W – параметр, позволяющий регулировать насколько предпочтительно добавление в автомат нового ребра перед добавлением нового состояния. Пусть $W = 2$. Тогда, если задача добавления пути может быть решена с помощью добавления двух ребер и одного состояния или добавления трех новых ребер, то алгоритм выберет добавление трех новых ребер. Если же второй вариант заключается в добавлении пяти новых ребер, то алгоритм выберет добавление двух ребер и нового состояния. Так как длина добавляемого пути равняется длине n

добавляемого сценария, то всегда можно выбрать $W = n$. При этом алгоритм будет действовать по следующей логике: если возможно добавить путь, не добавляя новых состояний в автомат, то этот путь будет выбран, иначе – будет добавлен путь с новыми состояниями.

Так как возможно, что не из всех вершин графа были добавлены исходящие ребра, то необходимо удалить все вершины, из которых недостижима хотя бы одна из вершин, соответствующих конечным состояниям исходного автомата. Следует заметить, что исходя из построения, в графе нет вершин, недостижимых из стартовой вершины (в каждом слое существует хотя бы одна вершина, из которой достижим весь следующий слой).

На основе сценария и исходного автомата был построен ациклический взвешенный ориентированный граф. Пример такого графа приведен на рис. 10.

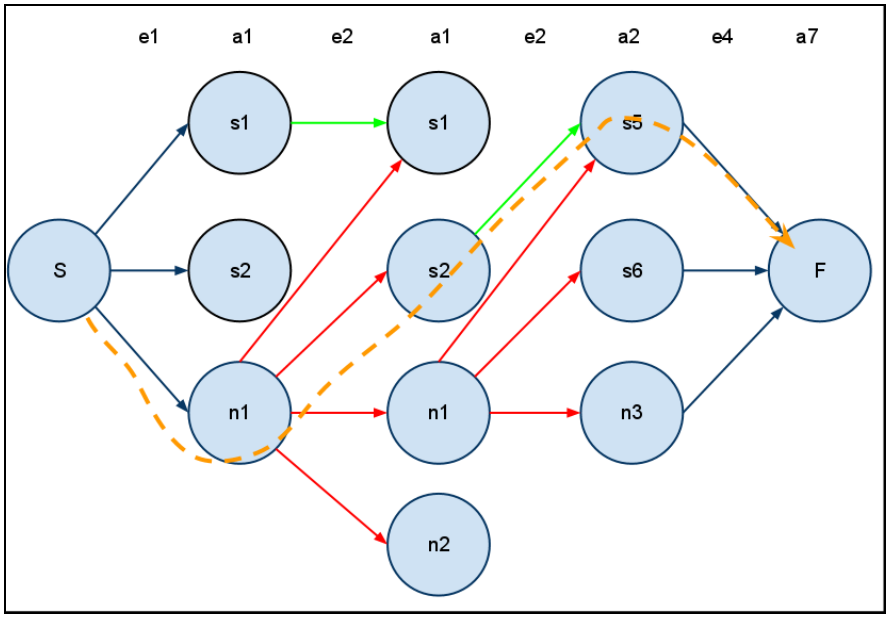

Рис. 10. Пример графа, построенного на основе сценария и исходного автомата

В верхней части рисунка написан добавляемый сценарий $e_1 a_1 e_2 a_1 e_2 a_2 e_4 a_7$. Под каждым действием a_i расположены вершины графа, соответствующие состояниям с действием a_i на входе. Зеленым цветом

отмечены ребра с весом ноль, синим – ребра с весом один, красным – ребра с весом $1 + W$. На рисунке не отмечены ребра, исходящие из вершины соответствующей состоянию n_2, для того, чтобы не загромождать рисунок. В этом графе еще не удалены висячие вершины, что необходимо сделать для уменьшения его размера.

Задача добавления нового сценария свелась к задаче поиска кратчайшего пути в построенном графе (на рис. 10 такой путь помечен оранжевым пунктиром). При этом кратчайший путь (путь с наименьшим весом) соответствует наименьшему относительно заданной метрики изменению автомата.

Метрика для изменения автомата задается как $\mu = E + W * S$, где E – число добавленных переходов, S – количество добавленных состояний, W – параметр, определяющий предпочтительность добавления ребер перед добавлением состояний.

Если кратчайший путь найден, то остается добавить в автомат новые состояния, соответствующие вершинам, через которые проходит путь, и переходы, соответствующие ребрам пути не нулевого веса.

Построенный граф не только взвешенный, ациклический и ориентированный, но еще и топологически отсортированный. Топологическая сортировка (*topological sort*) ориентированного ациклического графа $G =< V, E >$ представляет такое линейное упорядочение всех его вершин, что если граф G содержит ребро (u, v), то u при таком упорядочении располагается до v (если граф не является ациклическим, то такая сортировка невозможна) [9]. Если в построенном графе перечислить все вершины в порядке следования слоев, то полученный список будет отвечать требованиям топологической сортировки, причем не имеет значения, в каком порядке перечислять вершины внутри слоя.

«Ослабляя ребра взвешенного ациклического графа $G =< V, E >$ в порядке, определенном топологической сортировкой его вершин, кратчайшие пути из одной вершины можно найти в течение времени $\theta(V + E)$. В ориентированном ациклическом графе кратчайшие пути всегда вполне определены, поскольку даже если у некоторых ребер вес отрицателен, циклов с отрицательными весами не существует» [9]. Таким образом, найти кратчайший путь можно, ослабляя ребра в порядке следования слоев.

Проблема, из-за которой нельзя напрямую применять описанный выше метод поиска кратчайшего пути, состоит в том, что вес и существование ребра зависит от уже пройденного пути. Пусть, например, алгоритм пытается ослабить ребро $u \to v$. Если это ребро соответствует еще не существующему переходу между состояниями автомата, то его вес отличен от нуля. Однако если по пути до вершины u этот переход уже был совершен, то вес данного ребра ноль. Рассмотрим подробнее случаи изменения графа в зависимости от пройденного пути:

1. Если был совершен переход по ребру $u \to e \to v$, то следующий переход по ребру $u \to e \to v$ имеет вес 0.
2. Если был совершен переход по ребру $u \to e \to v$, то дальнейший переход по ребру $u \to e \to w$ невозможен.
3. Пусть u соответствует новому состоянию, которого еще нет в автомате, тогда ребро $u \to e \to v$ имеет вес $1 + W$, однако следующий переход $u \to e_1 \to w$ будет иметь единичный вес, так как состояние u уже создано.

Первые две ситуации возможны только в том случае, когда в добавляемом сценарии $sc = \{e_i, a_i\}_{i=1..n}$ существуют повторяющиеся пары $a_i e_{i+1}$ (далее будем называть их «повторами»). Пусть некая пара $a_x e_y$ встречается в сценарии дважды. Тогда если в обоих случаях после пары идет одинаковое

действие ($sc = \{\dots a_x e_y a_z \dots a_x e_y a_z \dots\}$), то реализуется первый вариант возможного изменения графа. Если же после пары идут различные действия ($sc = \{\dots a_x e_y a_z \dots a_x e_y a_q \dots\}, a_z \neq a_q$), то реализуется второй вариант возможного изменения графа. Ребра, прохождение по которым изменяет граф, будем называть «спорными». При реализации алгоритма полезным будет следующее замечание: если из вершины выходит спорное ребро, то все ребра, выходящие из данной вершины, являются спорными. Ситуация третьего типа возможна при повторении в сценарии отдельных действий и не зависит от событий сценария.

В качестве примера рассмотрим граф на рис. 11. В добавляемом сценарии имеется повторение пары $a_1 e_2$. Путь от вершины S до вершины F не может проходить через вершины $n_1 n_1 s_6$, так как из-за наличия ребра $n_1 \to e_2 \to n_1$ ребро $n_1 \to e_2 \to s_6$ становится недопустимым.

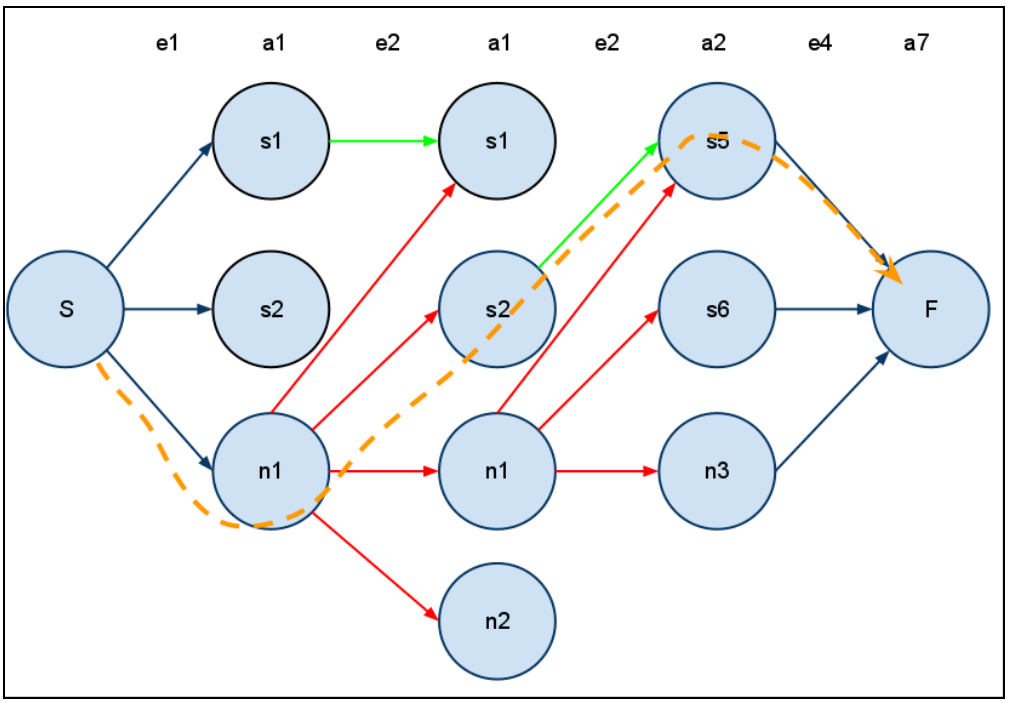

Рис. 11. Пример добавления сценария с повтором пары a1e2

Одним из возможных решений этой проблемы является расширение графа. Пусть каждое спорное ребро $u \to v$ ведет в отдельный подграф,

который является копией части графа достижимой из вершины v, но измененной с учетом данного ребра. Расширенный граф будет экспоненциального размера относительно числа повторов в добавляемом сценарии. Плюсом данного решения является то, что к расширенному графу можно применить метод ослабления ребер в порядке топологической сортировки без введения дополнительных структур данных и каких-либо осложнений. Однако при этом подходе явно происходит многочисленное дублирование информации, так как большинство подграфов будут идентичны с точностью до ребер.

Поиск кратчайшего пути в данном случае будет организован с помощью функции послойного ослабления ребер, которая будет рекурсивно вызывать себя при прохождении через спорное ребро. Смысл данной операции состоит в том, что на время спорное ребро фиксируется и функция ищет кратчайший путь в графе с учетом этого ребра. В качестве результата эта функция должна возвращать кратчайший путь от заданной вершины графа до конечной. Пусть в каждой вершине хранится информация о предке этой вершины в кратчайшем пути и ее расстоянии до стартовой вершины (до начала работы все расстояния установлены в INF). Между двумя слоями графа могут быть как спорные переходы, так и обычные, поэтому возникает проблема, связанная с тем, что функция может ослабить обычное ребро, обновив расстояние в вершине v, затем для следующего спорного ребра вызвать себя рекурсивно и снова обновить расстояние в вершине v, но уже с учетом спорного ребра. Эта операция не является корректной, так как при прохождении через обычное ребро и через спорное остаточные графы различаются и кратчайшие пути различны. Тогда для каждого слоя будем сначала запускать рекурсивно функции для каждого спорного ребра, а затем продолжать релаксацию обычных ребер. Каждая функция должна хранить, какие из вершин следующего слоя доступны из текущего, иначе возможно некорректное построение кратчайшего пути. В качестве примера рассмотрим

граф на рис. 12. Пусть при прохождении по первому пути ребро a недоступно, а при прохождении по второму пути недоступно ребро b. Тогда функция, вызванная для прохождения спорного ребра первого пути, дойдя до предпоследнего слоя, установит расстояние от S до v в некоторое число меньшее INF, а F останется недостигнутой. Функция, вызванная для прохождения спорного ребра второго пути, дойдя до предпоследнего слоя, сможет ослабить ребро a и будет построен путь от S до F, хотя на самом деле корректного пути не существует.

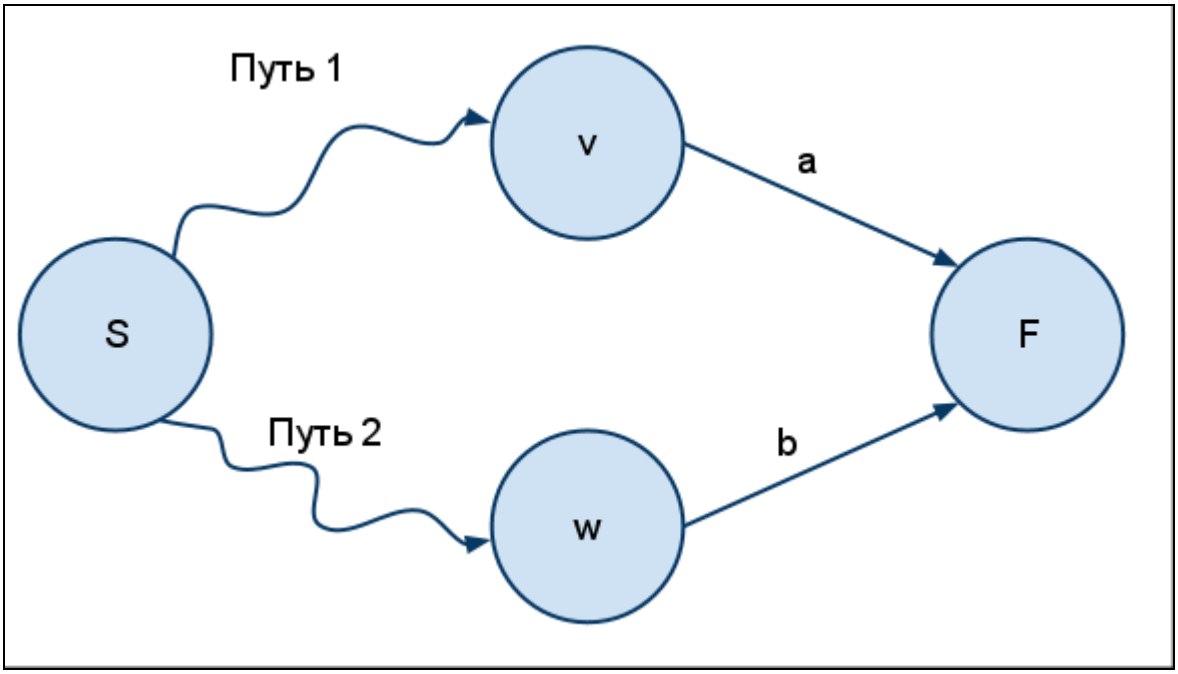

Рис. 12. Пример некорректного построения пути

Корректная релаксация ребра $u \to v$ затруднена тем, что текущей функции неизвестно кто задал вершине v ее текущее расстояние до стартовой вершины, это могла сделать и другая функция. Таким образом, у каждой функции должен быть свой идентификатор, который позволял бы определить, можно ли ослабить нужное ребро.

Если в данный момент времени функция находится в вершине v, то из вершины v уже известен кратчайший путь до стартовой вершины. Эта

информация позволяет определить, было ли уже создано состояние, соответствующее v.

Пусть у каждой вершины графа имеются следующие поля:

1. $parent$ – предок вершины в кратчайшем пути.
2. $distance$ – расстояние до стартовой вершины.
3. $createdStates$ – множество состояний, созданных при прохождении по кратчайшему пути к этой вершине.
4. $lastModified$ – идентификатор функции, которая последней обновляла поля вершины.

Тогда псевдокод ослабления ребра $u \to v$ с весом w функцией с идентификатором $func$ имеет следующий вид:

```
IF(lastModified != funcORu.distance + w < v.distance)
    v.distance = u.distance + w
    v.parent = u
    v.lastModified = func
    v.createdStates = u.createdStates
    IF(u соответствует новому состоянию)
        v.createdStates.add(u)
```

Общий алгоритм работы функции нахождения кратчайшего пути:

Вход: стартовая вершина $start$, множество сделанных спорных переходов E.

Выход: кратчайший путь P.

1. Пусть P – кратчайший путь из $start$, $P = \emptyset$.

2. Пусть Q – множество вершин из текущего слоя, $Q = \{start\}$.
3. Пока $Q \neq \emptyset$:

   3.1. $Q' = \emptyset$ – вершины следующего слоя, доступные из текущего.

   3.2. Для всех спорных ребер $u \to v$, таких что $v \in Q$:

      3.2.1. Если $(u \to v) \in E$, то ослабляем ребро $u \to v$, считая вес равным нулю. $Q' = Q' \cup \{v\}$.

      3.2.2. Если $(u \to v) \notin E$, то ослабляем ребро $u \to v$. Рекурсивно вызываем функцию из вершины v с учетом сделанного спорного перехода. Обновляем путь P если путь, возвращенный вызванной функцией, оказался короче.

      3.2.3. Если пройти по данному ребру невозможно из-за совершенных ранее спорных переходов, то переходим к следующему пункту.

   3.3. Для всех обычных ребер $u \to v$, таких что $v \in Q$:

      3.3.1. Ослабить ребро $u \to v$. $Q' = Q' \cup \{v\}$.

   3.4. $Q = Q'$
4. Если конечная вершина была достигнута по обычным ребрам и расстояние до нее меньше, чем длина P, то обновить P.

Отметим, что перед релаксацией любого ребра $u \to v$ необходимо проверить, является ли состояние s соответствующее вершине u новым и если является, то проверить создано ли оно при прохождении по кратчайшему пути от стартовой вершины до u.

По найденному кратчайшему пути надо добавить необходимые ребра и состояния в исходный автомат. В автомате появится путь, соответствующий добавляемому сценарию. Реализация данного алгоритма на языке *Java* приведено в приложении.

2.4. Оценка асимптотики времени работы алгоритма

Оценю среднее время работы представленного алгоритма. Пусть заданы следующие параметры:

1. l – длина добавляемого сценария.
2. n – число состояний автомата.
3. a – число различных действий.
4. e – число различных событий.
5. r – число ситуаций, когда между слоями построенного графа есть спорные переходы.

Оценка времени работы шагов алгоритма, описанного выше:

1. Определение существования конфликта за время $O(l)$.
2. Конструирование графа.
   2.1. Составление списков соответствия действий сценария состояниям автомата за время $O(n)$.
   2.2. Добавление ребер в граф за время $O(m)$, где m – число вершин графа.
   2.3. Удаление висячих состояний за $O(m)$, где m – число вершин графа.
3. Нахождение кратчайшего пути будет рассмотрено отдельно.
4. Добавление в автомат новых ребер и состояний за время $O(l)$.

С учетом того, что число вершин в графе может превышать число состояний автомата не более чем в два раза, то все действия, кроме нахождения кратчайшего пути, можно выполнить за время $O(l + n)$.

Представленный алгоритм нахождения кратчайшего пути в графе проходит по одному разу по каждому ребру расширенного графа, соответственно время его работы $O(E)$, где E – число ребер расширенного графа. Число E зависит от многих параметров, поэтому для его оценки будут сделаны некоторые упрощения исследуемой модели.

Пусть $r = 0$, тогда расширенный граф и граф, построенный изначально, совпадают. В среднем в слое графа находится $\frac{n}{a} + 1$ вершин. Здесь единица добавляется за счет одного нового состояния, которое присутствует в каждом списке. Оценим среднее число ребер между слоями графа. Пусть в среднем из вершины графа выходит $\frac{e}{2}$ переходов. Тогда вероятность того, что из вершины есть ребра во все вершины следующего слоя, равна вероятности того, что из соответствующей вершины графа переходов нет перехода по заданному событию. Вероятность того, что из состояния есть переход по заданному событию, с учетом сделанного допущения равна $\left(\frac{e}{2}\right)/e = \frac{1}{2}$. Таким образом, вероятность того, что из вершины есть ребра во все вершины следующего слоя, равна ½. Вероятность того, что из вершины есть ровно одно ребро, равна произведению вероятности существования перехода по заданному состоянию и вероятность того, что цель перехода обладает необходимым действием на входе. Таким образом, вероятность одного ребра равна $\frac{1}{2a}$. Тогда математическое ожидание числа переходов из вершины равно $\frac{n}{2a} + \frac{1}{2a}$, а число ребер между слоями $\left(\frac{n}{2a} + \frac{1}{2a}\right)\left(\frac{n}{a}\right) + \frac{n}{a} + 1$, что примерно равно $\frac{n^2}{2a^2} + \frac{n}{a} + 1$. Пусть $z = \frac{n^2}{2a^2} + \frac{n}{a} + 1$ – среднее число ребер между слоями. Тогда верно неравенство $z \geq 1$, причем равенство достигается только на пустом автомате. С учетом того, что всего слоев l, получаю оценку для среднего числа ребер в графе lz. Следовательно, время работы функции нахождения кратчайшего пути при отсутствии спорных моментов составляет $O(lz)$.

Теперь рассмотрю ситуацию, когда имеются спорные переходы между слоями. При наличии спорных переходов, алгоритм будет проходить по графу, начиная со слоя, содержащего спорные переходы, столько раз, сколько спорных переходов между слоями. Пусть слои со спорными

переходами распределены равномерно по длине добавляемого сценария, тогда если $r = 1$, то, исходя из рис. 13, математическое ожидание числа ребер расширенного графа будет равно $(lz/2)z + \frac{lz}{2}$, где z – математическое ожидание числа ребер между слоями.

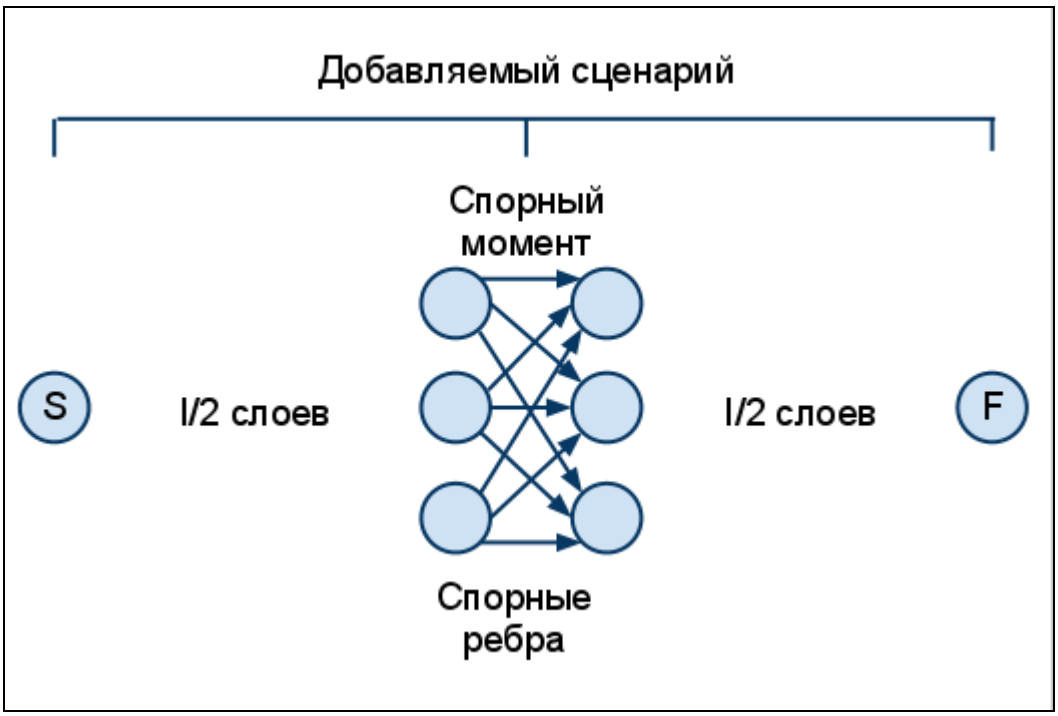

Рис. 13. Оценка числа ребер в расширенном графе при наличии спорного момента

Общий вид зависимости числа ребер в расширенном графе относительно r: $E \approx \left(\frac{l}{r+1}\right)\left(\frac{z^{r+1}-1}{z-1}\right)z$, где $z \geq 1$. Таким образом, время функции нахождения кратчайшего пути при заданных параметрах составляет $O\left(\left(\frac{l}{r+1}\right)\left(\frac{z^{r+1}-1}{z-1}\right)z\right)$. Среднее время добавления нового сценария в автомат равно $O\left(\left(\frac{l}{r+1}\right)\left(\frac{z^{r+1}-1}{z-1}\right)z + n + l\right)$, где $z \geq 1$ и зависит от числа состояний автомата и числа возможных действий.

Из изложенного следует:

1. Если спорных моментов нет, то время работы линейно зависит от длины сценария при заданных размере автомата и числе действий.

2. Если имеются спорные моменты, то время работы экспоненциально растет с числом спорных моментов. Рост времени работы будет медленнее при увеличении a и уменьшении n.

Эти выводы относительно скорости работы алгоритма экспериментально подтверждены в следующей главе.

ГЛАВА 3. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ МЕТОДА

3.1. Экспериментальная оценка времени работы алгоритма

Для экспериментальной оценки времени работы алгоритма были проведены испытания на случайных тестах с замером времени работы. Для этого был реализован генератор случайных тестов с заданным числом спорных моментов и генератор случайных автоматов заданного размера. Алгоритм запускался десять раз при фиксированных значениях параметров n, a, e, l, r, записывались минимальное, среднее и максимальное время работы алгоритма.

- Ситуация, когда n больше a, и есть спорные моменты.

$l = 20, r = 0..10, n = 20, a = 10, e = 10$, графики зависимости времени добавления от числа спорных моментов приведены ниже (рис. 14,15).

Рис. 14. Зависимость среднего времени работы от числа спорных мест

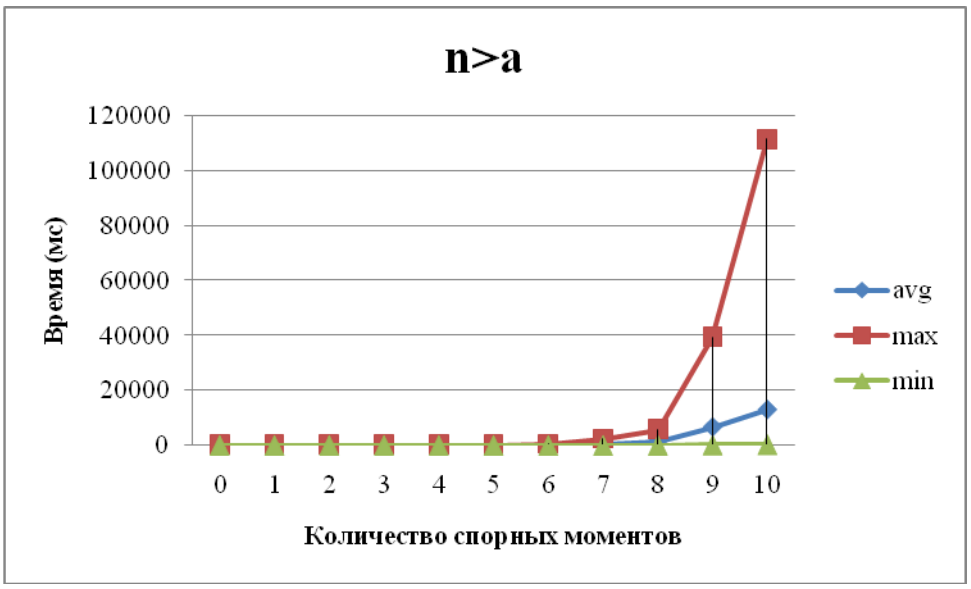

Рис. 15. Зависимость времени работы от числа спорных мест, показан коридор колебания

Как и было предположено ранее, в этом случае наблюдается экспоненциальный рост времени работы относительно числа спорных моментов.

- Ситуация, когда n меньше a, и есть спорные моменты.

$l = 20, r = 0..10, n = 20, a = 30, e = 10$, графики зависимости времени добавления от числа спорных моментов приведены ниже (рис. 16, 17).

Время работы растет, однако ощутимо медленнее, чем в предыдущем случае. Среднее время работы в этой ситуации снизилось с 13 с в худшем случае до 0.2 с, максимальное время работы также упало с 112 с до одной секунды.

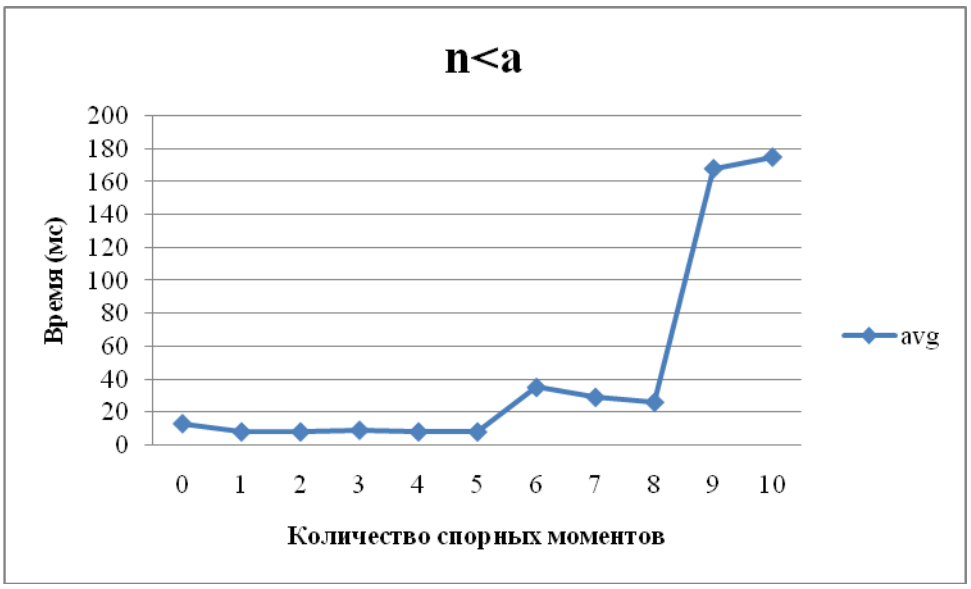

Рис. 16. Зависимость среднего времени работы от числа спорных мест

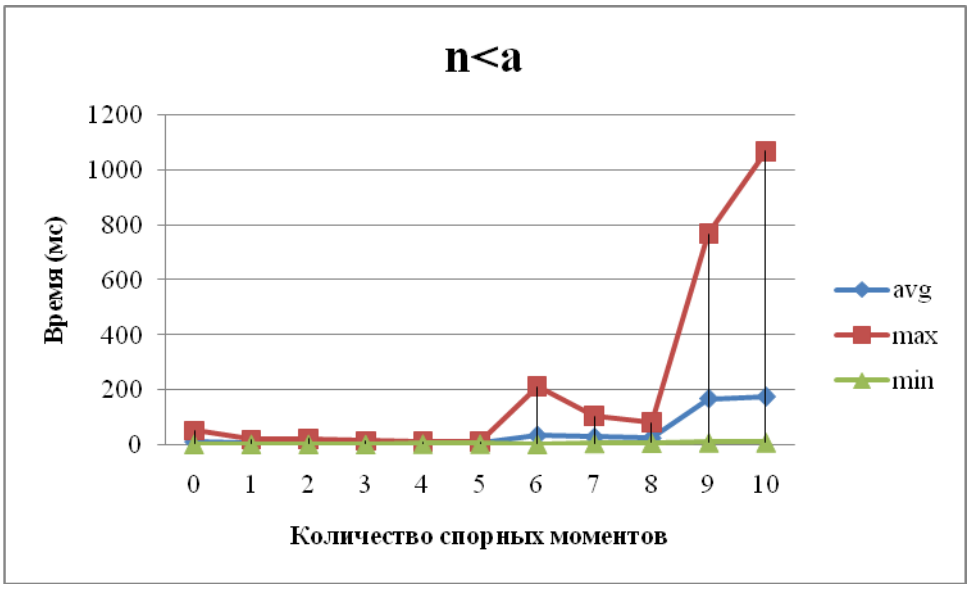

Рис. 17. Зависимость времени работы от числа спорных мест, изображен коридор колебания

- Ситуация, когда нет спорных моментов.

$l = 10..40$. В данном случае зависимость должна быть линейной относительно длины добавляемого сценария. Зависимость изображена на рис. 18.

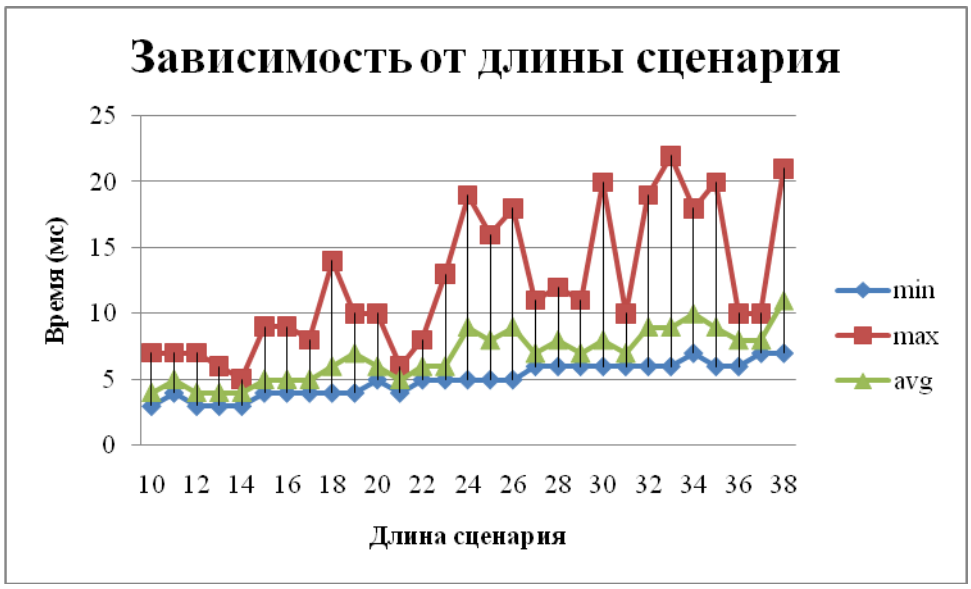

Рис. 18. Зависимость времени работы от длины сценария, изображен коридор колебания

Из диаграммы видно, что среднее время работы постепенно растет с длиной сценария. Максимальное время работы в худшем случае составило 22 миллисекунды.

Таким образом, можно сделать вывод, что данный алгоритм можно применять в следующих ситуациях:

1. Спорных моментов нет или немного.
2. Число состояний автомата меньше или не сильно превосходит число возможных действий.

3.2. Пример работы метода

В качестве примера работы метода сгенерирую автомат для решения задачи нахождения выхода из лабиринта по правилу одной руки, при условии, что лабиринт односвязный. Пусть задан управляемый робот, который может двигаться вперед, поворачивать налево и поворачивать направо. События, по которым могут совершаться переходы в автомате:

1. Финиш – робот достиг выхода из лабиринта.
2. Слева – есть стена слева, стены спереди нет.

3. Спереди – есть стена спереди, стена слева отсутствует.
4. Спереди и слева – тупик, есть стена спереди, и есть стена слева.
5. Пусто – ни спереди, ни слева стен нет.

Для начала добавим в пустой автомат сценарий, соответствующий прямолинейному движению робота вдоль стены. Этот сценарий имеет следующий вид: стена слева – движение вперед, стена слева – движение вперед, финиш – стоп. Полученный автомат представлен на рис. 19.

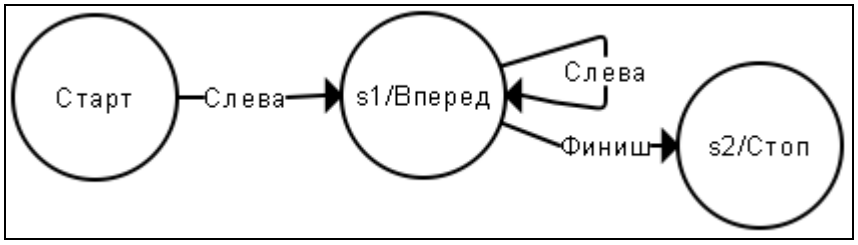

Рис. 19. После добавления одного сценария

Визуализация автомата выполнена с помощью инструментального средства *aiSee* [10], которое позволяет визуализировать графы, описанные с помощью языка *GDL* (*Graph Description Language*).

Теперь добавим сценарии, отвечающие за поворот робота налево в случае отсутствия после предыдущего шага стены слева. Для удобства действия поворота налево и движения вперед объединены в одно действие. Таким образом, добавляемые сценарии имеют вид: двигаться вперед пока есть стена слева, нет стены слева – поворот и движение вперед, финиш – стоп (рис. 20).

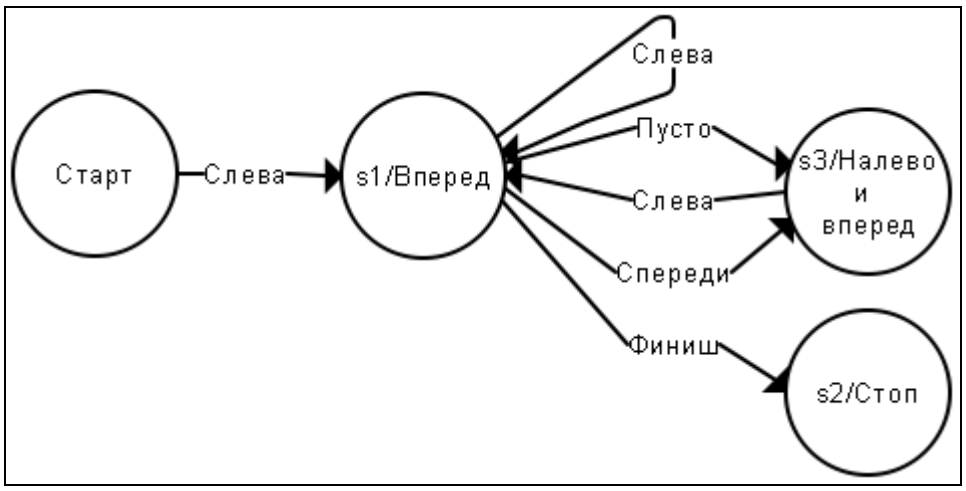

Рис. 20. После добавления сценария, соответствующего повороту налево

Осталось добавить сценарии, соответствующие повороту направо в случае тупика. Результат представлен на рис. 21. Алгоритм оптимально добавил эти сценарии и создал только одно дополнительное состояние, отвечающее за поворот направо. Старые состояния не изменились. Полученный автомат является оптимальным для данного набора сценариев, так как каждому возможному действию в этом автомате соответствует ровно одно состояние, каждый переход принадлежит хотя бы одному сценарию из рабочего множества.

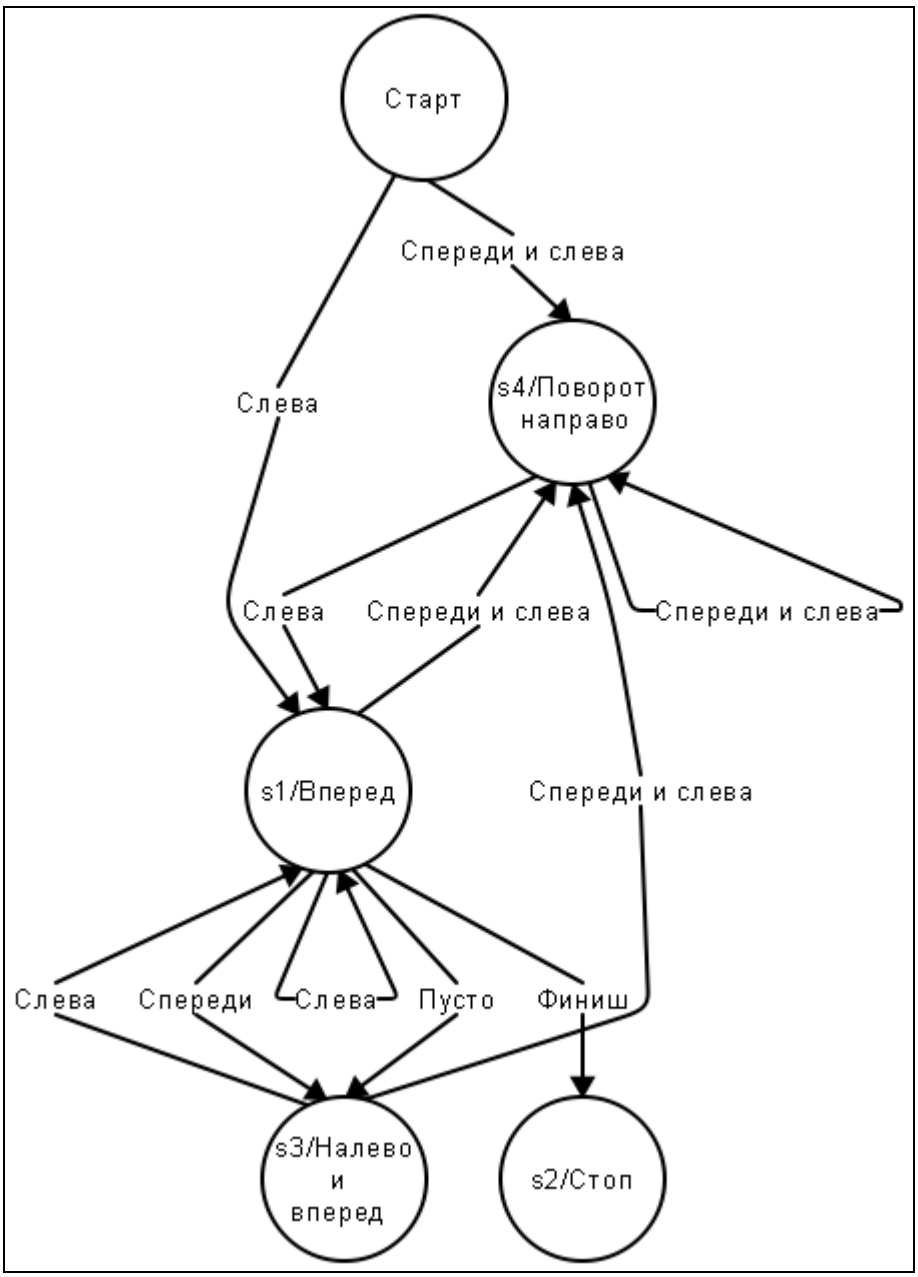

Рис. 21. Автомат, построенный для решения задачи нахождения выхода из лабиринта по правилу одной руки

ЗАКЛЮЧЕНИЕ

В настоящей работе исследована проблема изменения автоматной программы при модификации множества ее сценариев работы. При подробном исследовании проблемы были поставлены конкретные задачи:

1. Задача минимального изменения автоматной программы при добавлении сценария в рабочее множество.
2. Задача изменения автомата для устранения конфликта между автоматом и добавляемым сценарием.

Для решения поставленных задач предложен метод модификации автоматных программ при изменении множества сценариев работы. В ходе создания метода:

1. Разработан и реализован алгоритм добавления в рабочее множество сценария с минимальными, относительно введенной метрики, изменениями автомата.
2. Разработан и реализован алгоритм добавления в рабочее множество сценария при наличии конфликта между автоматом и добавляемым сценарием.
3. Проведено теоретическое и экспериментальное исследование асимптотики времени работы предложенных алгоритмов.

ИСТОЧНИКИ

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2010. – 176 с.
2. *Шалыто А. А., Туккель Н. И.* SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем // Промышленные АСУ и контроллеры. 2000. № 10, с. 44 – 48.
3. *Шалыто А.А.* Алгоритмизация и программирование для систем логического управления и «реактивных» систем // Автоматика и телемеханика. 2001. №1, с. 3 – 35.
4. *Шалыто А. А.* Парадигма автоматного программирования. http://is.ifmo.ru/works/_paradigma_automata.pdf
5. *Фаулер М.* Рефакторинг: улучшение существующего кода. СПб.: Символ-Плюс, 2006. – 432 с.
6. *Царев Ф. Н.* Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования. http://is.ifmo.ru/works/_zarev.pdf
7. *Царев Ф. Н.* Разработка методов совместного применения генетического и автоматного программирования. Магистерская диссертация. http://is.ifmo.ru/papers/_tsarev_masters.pdf
8. *Степанов О. Г.* Методы реализации автоматных объектно-ориентированных программ. http://is.ifmo.ru/disser/stepanov_disser.pdf
9. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ. М.: Вильямс, 2007. – 1296 с.
10. Инструментальное средство для визуализации графов *aiSee*. http://www.aisee.com