УДК 004.4'242

МИНИМАЛЬНАЯ МОДИФИКАЦИЯ АВТОМАТНЫХ ПРОГРАММ ПРИ ИЗМЕНЕНИИ СЦЕНАРИЕВ ИХ РАБОТЫ

А.В. Шестаков

Предлагается метод модификации автоматных программ при изменении сценариев их работы. Метод, в отличие от разработанных ранее, не использует эволюционные алгоритмы. Его важной особенностью является требование минимальности изменения программы. Метод может быть применен для построения автоматных программ на основе набора сценариев. Его суть заключается в сведении исходной задачи к задаче поиска кратчайшего пути в ациклическом взвешенном графе, построенном по исходному автомату и добавляемому сценарию работы.
Ключевые слова: автоматное программирование, сценарий работы, автомат Мура.

Введение

Одним из подходов к программированию объектов со сложным поведением является использование парадигмы автоматного программирования [1] или «программирования с явным выделением состояний» [2]. При использовании этого подхода поведение программы задается в виде управляющих конечных автоматов. Состояния каждого автомата соответствуют состояниям управляемого объекта, а переходы между состояниями осуществляются при выполнении заданных условий.

Автоматное программирование хорошо подходит для создания событийно-ориентированных приложений [3, 4]. По каждому событию автомат в зависимости от состояния генерирует определенное выходное воздействие, а последовательность событий порождает последовательность выходных воздействий. Такая цепочка пар «событие – воздействие» называется сценарием работы автоматной программы.

Изменение сценариев работы требует модификации автоматной программы. Метод, предложенный в настоящей работе, позволяет изменить программу в соответствии с новыми сценариями. Использование этого метода также гарантирует минимальность изменений. Существуют методы, позволяющие построить автомат на основе определенного множества сценариев, например, с помощью генетических алгоритмов [5]. Предлагаемый метод, в отличие от разработанных ранее, не использует эволюционные алгоритмы.

Таким образом, постановка задачи работы состоит в следующем. Требуется модифицировать автоматную программу таким образом, чтобы полученная программа удовлетворяла измененному набору сценариев работы. Дополнительным требованием является минимальность отличия модифицированной программы от исходной. Способ определения количества изменений приведен в разделе «Нахождение кратчайшего пути».

Предметная область

Логика автоматной программы задается с помощью детерминированного конечного автомата. В работе в качестве автоматной модели используется автомат Мура второго рода [2]. Используемую модель автомата можно представить в виде графа переходов, в котором при входе в каждое состояние генерируется выходное воздействие, переходы осуществляются по событиям и определено стартовое состояние, а также одно или несколько конечных.

Сценарием работы в настоящей работе называется список пар $<e, z>$, где e – событие, z – выходное воздействие. Сценарий работы определяет поведение программы при появлении определенной последовательности событий.

Под рабочим множеством в работе понимается 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:

- добавление сценария;
- удаление сценария;
- изменение существующего сценария.

При удалении сценарий убирается из рабочего множества, а автомат можно не изменять. При добавлении сценария может возникнуть ситуация конфликта. Конфликтом называется ситуация, когда требуется добавить переход из заданного состояния по событию e в состояние с выходным воздействием на входе z_1, но уже существует переход в состояние с воздействием на входе z_2.

Добавить сценарий, конфликтующий с хотя бы одним сценарием из рабочего множества, невозможно. Однако, если добавляемый сценарий конфликтует с автоматом, но не конфликтует со сценариями рабочего множества, то его добавление возможно с помощью предварительного изменения автомата при помощи рефакторинга [6]. Добавление сценария при условии конфликта с автоматом подробно рассмотрено в работе [7]. В случае отсутствия конфликтов сценарий следует добавить с минимальным изменением автоматной программы при помощи изложенного далее алгоритма.

Добавление сценария

Суть алгоритма заключается в сведении исходной задачи модификации автомата к хорошо изученной задаче поиска кратчайшего пути в графе. Перед началом работы алгоритма требуется убедиться, что сценарий не конфликтует с исходным автоматом. В случае обнаружения конфликта его следует устранить с помощью метода, описанного в работе [7]. Алгоритм выявления конфликта представлен ниже.

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

Результатом проверки является либо сообщение о том, что сценарий в автомат добавить нельзя, либо хвост исходного сценария $subSc=\{e_i, z_i\}_{i=m..n}$ и состояние *S*, из которого требуется этот остаток добавить в автомат. Для удобства далее будем считать, что задача состоит в добавлении в автомат сценария *subSc*, пары которого перенумерованы, начиная с единицы. Следовательно, $subSc=\{e_i, z_i\}_{i=1..n-m}$, а стартовым состоянием считается состояние *S*.

Построение графа по автомату и сценарию

Для сведения исходной задачи к задаче нахождения кратчайшего пути строится взвешенный граф. Кратчайший путь в этом графе будет соответствовать такому минимальному изменению автомата, что полученный автомат будет содержать требуемый сценарий.

Для построения графа рассматривается список воздействий $\{z_i\}_{i=1..n}$ добавляемого сценария. Каждому воздействию z_i сопоставляется список состояний автомата, при входе в которые выполняется это воздействие: $z_i \rightarrow \{s_1^i, s_2^i, \ldots\}$. Далее список состояний, соответствующий *i*-му воздействию сценария, называется *i*-м слоем. Для того чтобы добавить сценарий, необходимо построить непротиворечивый путь через эти слои, причем разрешается использовать только одно состояние из слоя.

Для реализации каждого воздействия может использоваться не только состояние из автомата, но и новое состояние, добавленное в автомат,. Для того чтобы избежать ситуации, когда для реализации воздействия невозможно выбрать состояние из списка, в каждый список добавляется *k* новых состояний, где *k* – число раз, когда пара z_i,e_i встречалась в сценарии. Таким образом, если $<z_i,e_i>=<z_j, e_j>$, то:

- $z_i \rightarrow \{s_1^i, s_2^i, \ldots, ns_1^i\}$;
- $z_j \rightarrow \{s_1^j, s_2^j, \ldots, ns_1^j, ns_2^j\}$.

Здесь верхний индекс обозначает номер слоя, а нижний – номер состояния. При этом ns_1^j и ns_1^i указывают на одно и то же состояние, хотя находятся в разных слоях.

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 вершина, соответствующая стартовому состоянию, полученному на этапе выявления конфликта.

Для добавления ребер в граф рассматривается каждая вершина *v*. Пусть этой вершине соответствует состояние *s* исходного автомата. Ребра добавляются по следующим правилам:

- если *s* – стартовое состояние, то надо добавить ребра из *v* во все вершины первого слоя. Вес каждого ребра устанавливается равным единице;
- если вершина *v* лежит в *i*-м слое и у состояния *s* существует исходящий переход $s \rightarrow e_{i+1} \rightarrow s_{zi+1}$, где s_{zi+1} – некоторое состояние с воздействием z_{i+1} на входе, то необходимо добавить ребро $v \rightarrow w$, где *w* – вершина из $(i+1)$-го слоя, соответствующая состоянию s_{zi+1}. Вес этого ребра устанавливается равным нулю;
- если вершина *v* лежит в *i*-м слое и у состояния *s* существует исходящий переход $s \rightarrow e_{i+1} \rightarrow s_{zx}$, где s_{zx} – некоторое состояние с воздействием $z \neq z_{i+1}$ на входе, то ребро не добавляется;
- если вершина *v* лежит в *i*-м слое 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 равным единице;
- 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* – вес добавления нового состояния. Вес добавления нового ребра считается равным единице.

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). Пример построенного графа приведен на рисунке.

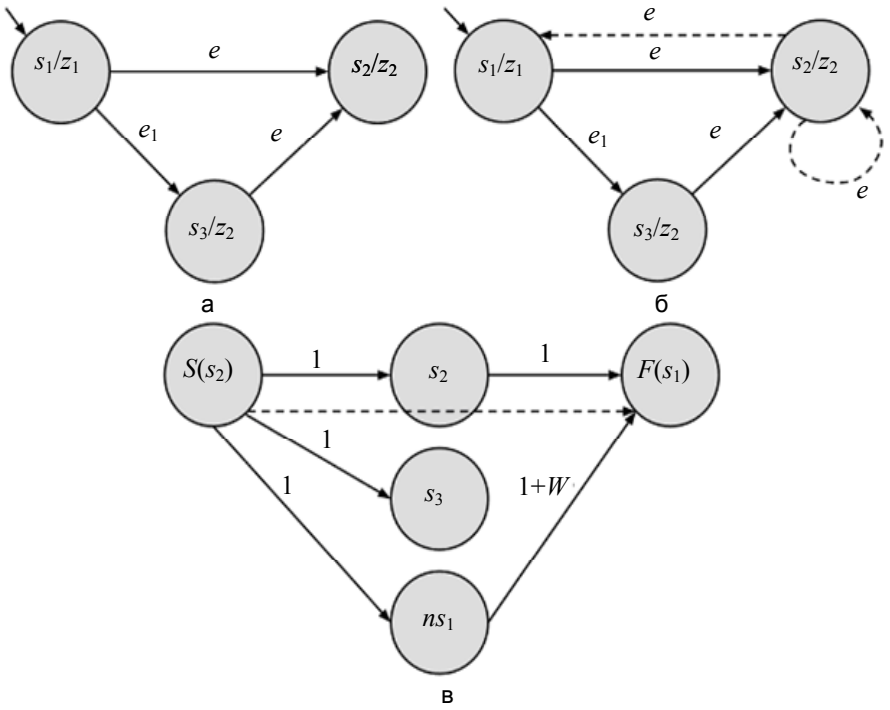

Рисунок. Добавление в автомат сценария $e\ z_2\ e\ z_2\ e\ z_1$: исходный автомат (а); автомат после добавления требуемого сценария (б); граф, построенный по исходному автомату и добавляемому сценарию (в)

Добавляемый сценарий $e\ z_2\ e\ z_2\ e\ z_1$ уже частично присутствует в автомате, поэтому добавляется его остаток $e\ z_2\ e\ z_1$. Стартовым состоянием S считается состояние s_2. Найденный в графе кратчайший путь $s_2\ s_2\ s_1$ (на рисунке отмечен пунктиром) имеет вес два. Соответствующие ему переходы добавляются в исходный автомат.

Нахождение кратчайшего пути

Задача добавления нового сценария свелась к задаче поиска кратчайшего пути в построенном графе. На рисунке такой путь помечен пунктиром. В настоящей работе метрика изменения автомата задается как $\mu=E+W\cdot S$, где E – число добавленных переходов; S – число добавленных состояний; W – вес добавления состояния. Если кратчайший путь найден, то остается добавить в автомат новые состояния, соответствующие вершинам, через которые проходит путь, и переходы, соответствующие ребрам пути ненулевого веса.

Проблема, из-за которой нельзя напрямую применять описанный в работе [8] метод поиска кратчайшего пути для ациклических взвешенных графов, состоит в том, что вес и существование ребра зависят от уже пройденного пути:

- если был совершен переход по ребру $u\rightarrow e\rightarrow v$, то следующий переход по ребру $u\rightarrow e\rightarrow v$ имеет вес ноль;
- если был совершен переход по ребру $u\rightarrow e\rightarrow v$, то дальнейший переход по ребру $u\rightarrow e\rightarrow w$ невозможен;
- пусть u соответствует новому состоянию, которого еще нет в автомате, тогда ребро $u\rightarrow e\rightarrow v$ имеет вес $1+W$, однако следующий переход $u\rightarrow e_1\rightarrow w$ будет иметь вес, равный единице, потому что состояние u уже создано.

Первые две ситуации возможны только в том случае, когда в добавляемом сценарии $sc=\{e_i, z_i\}_{i=1..n}$ существуют повторяющиеся пары $z_i\ e_{i+1}$. Ребра, прохождение по которым изменяет граф, будем называть спорными. Ситуация третьего типа возможна при повторении в сценарии отдельных воздействий и не зависит от событий сценария. В качестве примера рассмотрим граф на рисунке. В добавляемом сценарии $e\ z_2\ e\ z_2\ e\ z_1$ есть повторение пары $z_2\ e$. Путь из вершины S в вершину F не может проходить через вершины $s_2\ s_3\ s_1$, так как из-за наличия ребра $s_3\rightarrow e\rightarrow s_2$ ребро $s_3\rightarrow e\rightarrow s_1$ становится недопустимым.

Поиск кратчайшего пути в описанных условиях реализуется с помощью функции послойной релаксации ребер, которая рекурсивно вызывает себя при прохождении через спорное ребро. Следовательно, функция фиксирует спорное ребро и ищет кратчайший путь в графе с учетом пройденного пути.

Пусть в каждой вершине хранится информация о предке этой вершины в кратчайшем пути и ее расстоянии до стартовой вершины (до начала работы все расстояния считаются бесконечными). Между двумя слоями графа могут быть как спорные, так и обычные переходы, поэтому возникает проблема: 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, что не всегда приводит к корректному результату. Предлагается для каждого слоя сначала запускать рекурсивно функции для каждого спорного ребра, а затем продолжать релаксацию обычных ребер. Также каждая функция должна хранить, какие из вершин следующего слоя доступны из текущего, иначе возможно некорректное построение кратчайшего пути. Кроме того, корректная релаксация ребра $u \to v$ затруднена тем, что текущей функции неизвестно, кто задал вершине v ее текущее расстояние до стартовой вершины, так как это могла сделать и другая функция. Таким образом, у каждой функции должен быть свой идентификатор, который позволял бы определить, можно ли ослабить ребро.

Если в некоторый 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 есть 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* выглядит следующим образом:

```
IF(lastModified != func OR u.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)
```

Приведем псевдокод функции нахождения кратчайшего пути:

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

Выход: кратчайший путь P.

1. Пусть P – кратчайший путь из *start*, $P=\varnothing$.
2. Пусть Q – множество вершин текущего слоя, $Q=\{start\}$.
3. Пока $Q \neq \varnothing$:
    1. $Q'=\varnothing$ – вершины следующего слоя, доступные из текущего. Для всех спорных ребер $u \to v$, таких, что u принадлежит Q:
        1. Если $u \to v$ принадлежит E, то ослабляется ребро $u \to v$, вес считается равным нулю. $Q'=Q'$ в объединении с $\{v\}$.
        2. Если $u \to v$ не принадлежит E, то ослабляется ребро $u \to v$. Рекурсивно вызывается функция из вершины v с учетом сделанного спорного перехода. Обновляется путь P, если путь, возвращенный вызванной функцией, короче.
        3. Пропустить ребро, если переход по нему невозможен из-за совершенных ранее спорных переходов.
    2. Для всех обычных ребер $u \to v$, таких, что u принадлежит Q , ослабить ребро $u \to v$. $Q'=Q'$ в объединении с $\{v\}$.
    3. $Q=Q'$.
4. Если конечная вершина была достигнута по обычным ребрам и расстояние до нее меньше, чем длина P, то обновить P.

По найденному кратчайшему пути добавляются необходимые ребра и состояния в исходный автомат. В автомате появляется путь, соответствующий добавляемому сценарию.

Заключение

В работе исследована проблема изменения автоматной программы при модификации множества ее сценариев работы. Поставлена задача достижения минимального изменения автоматной программы при добавлении сценария в рабочее множество. Для решения предложен метод, который также может быть использован для построения автомата по набору сценариев работы.

Литература

1. Шалыто А.А. Парадигма автоматного программирования // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 8 (53). – С. 3–24.
2. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб: Питер, 2010. – 176 с.
3. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. – СПб: Наука, 1998. – 628 с.
4. Шалыто А.А. Алгоритмизация и программирование для систем логического управления и «реактивных» систем // Автоматика и телемеханика. – 2001. – № 1. – С. 3–29.
5. Царев Ф.Н. Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. – 2010. – № 5. – С. 31–36.
6. Федотов П.В., Степанов О.Г. Внесение изменений в автоматные программы // Научно-технический вестник СПбГУ ИТМО. – 2011. – № 1 (71). – С. 77–83.
7. Шестаков А.В. Разработка методов модификации автоматных программ при изменении их сценариев работы. Бакалаврская работа. СПбГУ ИТМО, 2011 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/2011-bachelor-shestakov/, свободный. Яз. англ. (дата обращения 07.03.2012).
8. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. – М.: Вильямс, 2007. – 677 с.

Шестаков Алексей Викторович – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, студент, shestakov@rain.ifmo.ru