Преподавание математики и компьютерных наук в классическом университете. Доклады II-й научно-метод. конференции преподавателей матем. ф-та и ф-та ИВТ Ярославского гос. ун-та им. П.Г. Демидова, Ярославль: ЯрГУ, 2007. С. 91-101.

О дисциплине специализации «Верификация программ» [1]

Е. В. Кузьмин, В. А. Соколов

Обсуждаются распределение нагрузки на аудиторную и внеаудиторную работу студентов по дисциплине специализации «Верификация программ» и затрагиваемые в лекциях и заданиях научные направления по тематике предмета.

Библиография: 7 названий.

В Ярославском государственном университете им. П.Г. Демидова на факультете информатики и вычислительной техники для студентов 5-го курса специальности «Прикладная математика и информатика» преподаётся дисциплина специализации «Верификация программ». В рамках этого учебного курса отводится значительное число внеаудиторных часов на самостоятельную работу студентов. В этой статье обсуждаются распределение нагрузки на аудиторную и внеаудиторную работу студентов и затрагиваемые в лекциях и заданиях научные направления по тематике предмета.

Верификация представляет собой анализ корректности программных систем относительно спецификации, в которой задаются исследуемые программные свойства. Современные методы анализа корректности включают в себя тестирование, метод доказательства теорем (theorem proving) [1] и метод проверки модели (model checking) [2, 3].

Тестирование применяется после окончательного написания программы. Но, как известно (Э. Дейкстра), если при тестировании ошибки найдены не были, это ещё не означает, что их нет вовсе. Отсюда следует необходимость рассмотрения формальных методов доказательства корректности программ. Поскольку тестирование – это способ проверки правильности программ, с которым студенты так или иначе (обычно в бессистемной форме) сталкивались при выполнении различных лабораторных работ, связанных с программированием, это направление предлагается студентам на самостоятельное изучение. Аудитории ставится задача подготовить ряд докладов и рефератов, отражающих современное положение дел в науке по тестированию. Основная цель – обратить внимание студентов на то, что тестирование представляет собой серьёзную проблематику, по которой в настоящее время проводится активная

[1] Работа поддержана РФФИ, грант №07-01-00702-а.

научно-исследовательская работа, а также определить круг проблем и ограничений, существующих в тестировании.

Метод доказательства теорем является очень трудоёмким методом с сильной привязкой к семантике языка программирования. Поэтому в рамках курса (на лекциях) вводится «простой» язык программирования [1], включающий привычные конструкции условного перехода и цикла, но имеющий ограничения на типы данных: допускаются переменные только целочисленного типа (или булевого типа). После определения семантики операторов введённого языка программирования, разъяснения основных принципов доказательства программ, написанных на этом языке, и рассмотрения нескольких примеров каждому студенту предлагается (по вариантам) доказать корректность небольшой программы. При этом допускаются «мягкая» и «жёсткая» формы задания. В первом случае формальная спецификация даётся вместе с текстом программы, во втором случае студенту необходимо самостоятельно произвести спецификацию формальным образом по сопровождению на естественном (русском) языке, т.е. построить ряд предикатов, отражающих начальные условия и ожидаемый результат выполнения программы. Пример задания приведён ниже.

Докажите формально, что следующий алгоритм записывает в a приближённое значение квадратного корня числа n:

$a, b := 0, n{+}1;$
do $a{+}1 \neq b \ \rightarrow \ d := (a{+}b){\div}2;$
if $d{*}d \leq n \ \rightarrow \ a := d$
[] $d{*}d > n \ \rightarrow \ b := d$
fi
od

Спецификация: $\{Q\colon\ 0 \leq n\}$ – *предусловие*, $\{R\colon\ 0 \leq a^2 \leq n < (a{+}1)^2\}$ – *постусловие*, $\{P\colon\ a < b \leq n{+}1 \wedge a^2 \leq n < b^2\}$ – *инвариант цикла*, $\{t\colon\ b - a + 1\}$ – *ограничивающая функция*.

Важно отметить, что доказательство корректности такого рода программы проходит в терминах предикатов (что позволяет на практике применять полученные ранее знания по теории исчисления предикатов), занимает от трёх до пяти страниц и требует от студента предельной точности рассуждений. Более того, особенностью метода доказательства теорем является то, что в ходе рассуждений можно установить корректность программы относительно спецификации, однако если этого не удаётся сделать, то это, вообще говоря, не означает наличия ошибок в программе, а требует более тонкого разбора. Перефразируя известное изречение Э. Дейкстры (о тестировании) можно сказать, что метод доказательства теорем способен доказать отсутствие о шибок в программе (относительно спецификации), но не всегда доказы-

вает их присутствие. С позиции технологии программирования метод доказательства теорем следует рассматривать как дополнение тестирования, позволяющее получать более совершенную стратегию отладки программ.

Метод проверки модели (model checking) – это метод автоматической верификации программных систем. При этом подходе для программы строится модель с конечным числом состояний, а свойства модели (спецификация) выражаются на языке темпоральной логики. Конечная модель и формулы темпоральной логики подаются на вход программе-верификатору, и далее проверка истинности формул для модели осуществляется автоматически (в основном с применением переборных алгоритмов). Поэтому в рамках обсуждаемой дисциплины по методу проверки модели важно рассмотреть следующие разделы: построение конечной модели программы, спецификация свойств на языках темпоральных логик и программы-верификаторы с описанием методов и алгоритмов, применяемых в этих программных средствах.

Необходимо заметить, что построение *конечной адекватной* (исходной программе) модели является весьма серьёзной задачей, поскольку модель может не учитывать ряд программных свойств или порождать несуществующие. Несмотря на то, что проверяются свойства модели, объектом исследования всё же по-прежнему остаётся программа.

Для спецификации свойств программ используются наиболее распространённые и широко применяемые темпоральные логики: логика линейного времени LTL (linear-time temporal logic) и логика ветвящегося времени CTL (branching-time temporal logic или computation tree logic). Логики имеют довольно простые синтаксис и семантику и не встречают трудностей в понимании у аудитории. В то же время прикладной аспект, касающихся применения этих логик, студентам не кажется столь прозрачным и полезным без достаточно простых, но семантически нагруженных примеров. Важно отметить, что в имеющейся литературе (статьи, учебники и монографии) таких примеров либо нет, либо они имеют вырожденный характер (применение логик в этом случае становится бессмысленным). В публикациях основной упор сделан на алгоритмы и методики. В работах, касающихся практического применения метода проверки модели, объектами рассмотрения являются или (довольно нетривиальные) параллельные алгоритмы, или протоколы передачи данных, перегруженные деталями реализации и требующие специальной подготовки и дополнительных знаний.

По мнению авторов, выходом из сложившейся ситуации может служить демонстрация метода проверки модели на программах, построенных с применением автоматного подхода к программированию [5–7]. Технология автоматного программирования является достаточно эффективной при построении программного обеспечения для «реак-

тивных» систем и систем логического управления. Эта технология, не исключая других методов построения программного обеспечения «без ошибок», существенно более конструктивна, так как позволяет начинать «борьбу с ошибками» еще на стадии алгоритмизации. Автоматный подход к программированию с точки зрения моделирования и анализа программных систем имеет ряд преимуществ по сравнению с традиционным подходом. При автоматном программировании исключается проблема адекватности модели программе, поскольку набор взаимодействующих автоматов, описывающий логику программы, уже является адекватной конечной моделью, по которой формально и изоморфно строится программный модуль (что является бесспорным плюсом автоматной технологии). К тому же свойства программной системы в виде автоматов формулируются и специфицируются естественным и понятным образом (в том числе и на языках темпоральных логик). Проверка свойств осуществляется в терминах, которые естественно вытекают из автоматной модели программы.

При автоматном подходе к проектированию и построению программ выделяются две части: системно независимая и системно зависимая. Первая часть реализует логику программы и задаётся системой взаимодействующих автоматов Мура–Мили. Проектирование каждого автомата состоит в создании по словесному описанию (декларации о намерениях) схемы связей, описывающей его интерфейс, и графа переходов, определяющего его поведение. По этим двум документам формально и изоморфно может быть построен модуль программы (и затем реализована системно зависимая часть), соответствующий автомату.

Таким образом, в качестве задания студентам может быть предложено произвести построение поведенческой модели системы автоматов и спецификации для «автоматной» программы, построенной, например, по иерархической модели [4]. Под поведенческой моделью понимается структура Крипке (или конечная система переходов), которая непосредственно описывает поведение (последовательность действий и состояний при исполнении) программы. При этом описание структуры Крипке должно преследовать следующую цель. Спецификация в виде формул темпоральной логики с учётом структуры Крипке должна охватывать максимально возможное число свойств, которые были (и могут быть) выражены (для программы) на естественном языке.

Далее следует пример модели «автоматной» программы, предлагаемой для верификации методом проверки модели.

Рассмотрим системно независимую часть автоматной программы, которая отвечает за логику управления основными операциями банкомата (пользовательский интерфейс, работа с картой, выдача денег).

Процесс работы банкомата заключается в следующем. После того, как карта вставлена в банкомат, на дисплее появляется надпись «Вве-

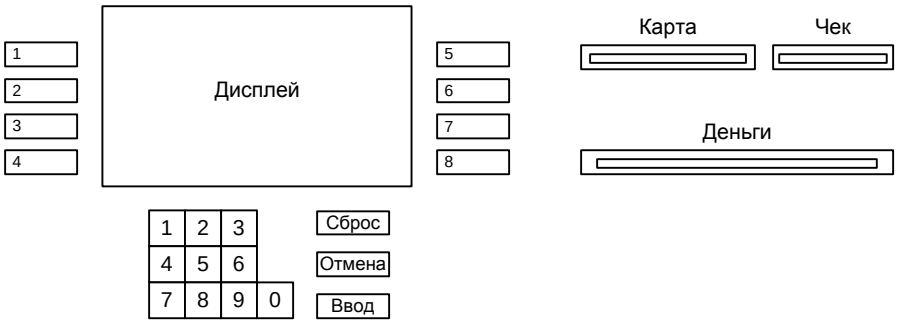

Рис. 1. Пользовательский интерфейс банкомата

дите Ваш персональный PIN-код». С помощью оцифрованных кнопок вводится PIN-код. Если PIN-код верен, предлагается выбрать одно из двух действий: «Снятие наличных» (обезличенная кнопка 6) и «Остаток на счете» (обезличенная кнопка 7). Если нажата кнопка «Остаток на счёте», банкомат выдает чек и возвращает карту. Для получения денег нажимается кнопка «Снятие наличных», а на дисплее появляется надпись «Печатать чек?». После выбора нужного варианта появляется надпись «Выберите сумму», а обезличенным кнопкам присваиваются значения: 3000, 2000, 1500, 1000, 500, 200, 100 и «другая». Далее происходит проверка введенной денежной суммы на предмет превышения лимита. Если лимит превышен, предлагается ввести другую сумму. В случае корректности указанной суммы появляется карта и чек, если он был затребован. После взятия карты банкомат выдает деньги. Кнопка «Сброс» прерывает текущую операцию и возвращает карту, а кнопка «Отмена» позволяет ввести заново неверно набранную информацию. Если деньги или карта не были взяты в течение 20 секунд, они захватываются банкоматом.

Банкомат разбивается на чётко выраженные подобъекты управления (пользовательский интерфейс, механизмы захвата карты и выдачи денег), которые имеют свои собственные подсистемы управления. Подсистемы управления представлены в виде конечных автоматов Мура–Мили, выстроенных в иерархию. В результате логическая часть системы управления банкоматом имеет вид иерархической модели системы взаимодействующих автоматов [4]. Автомат находящийся выше по иерархии, управляет своими вызываемыми автоматами путем генерации событий и передачи им управления с этими событиями. Кроме того, автомат следит за состояниями вызываемых автоматов, так как от них могут зависеть его собственные переходы по состояниям. На рис. 2 представлена схема взаимодействия автоматов. События обозначаются символом $\mathbf{e}$, запросы к объекту управления (входные переменные) — $\mathbf{x}$,

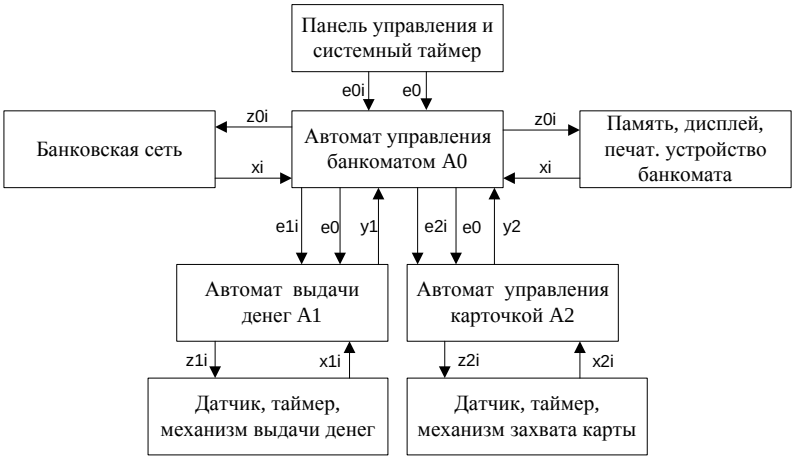

Рис. 2. Схема взаимодействия автоматов управления

текущее состояние некоторого автомата **Ai** хранится в переменной **yi**, а выходные воздействия обозначаются **z**.

Основной автомат $A0$ отвечает за логику взаимодействия с пользователем через панель управления. Автомат $A0$ реагирует на нажатие кнопок на панели инструментов, производит процедуру чтения/записи служебной информации в память банкомата, запрашивает необходимую информацию в банковской сети, и, кроме того, постоянно (по системному таймеру, генерирующему событие $e0$) проверяет возможность (условия) перехода в следующее состояние. Автомат $A0$ имеет вызываемые автоматы $A1$ и $A2$, взаимодействие с которыми осуществляется передачей управления посредством отправки им определённых событий и отслеживанием их текущих состояний. Схема связей и граф переходов автомата управления банкоматом $A0$ представлена на рис. 4.

Автомат $A1$ взаимодействует с автоматом управления $A0$, передавая последнему своё текущее состояние и получая от него события «извлечь деньги» и «проверить переходы». Автомат $A1$ обращается с запросами параметров к датчику денег и таймеру паузы и осуществляет выходные воздействия на механизм выдачи денег и таймер.

Автомат $A2$ отвечает за управление механизмом захвата/возврата карты, обращаясь с запросами к датчику карты и таймеру паузы и воздействуя на таймер и механизм управления захвата/выдачи карты.

Схемы связей и графы переходов автоматов управления механизмами выдачи денег $A1$ и захвата/возврата карты $A2$ представлены на рис. 3.

Для всей системы взаимодействующих автоматов правило перехода в новое состояние в общем случае можно описать следующим образом. После получения события автомат в зависимости от своего текущего

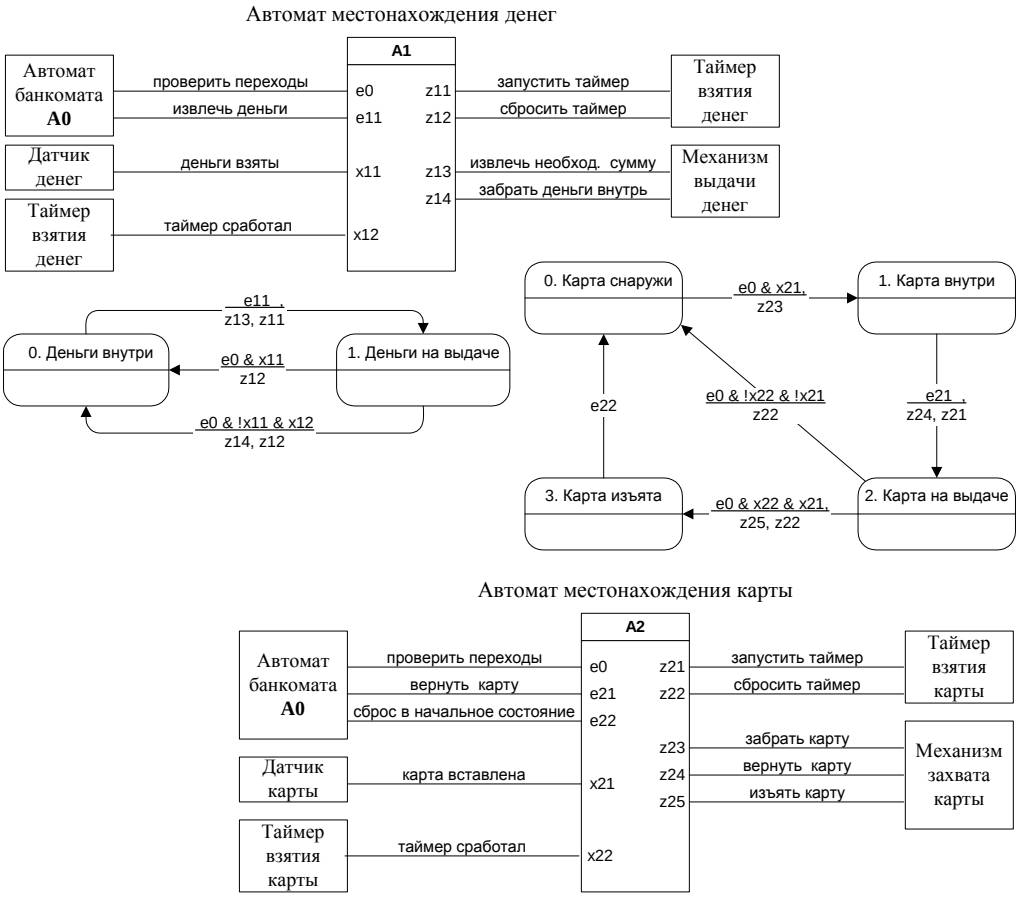

Рис. 3. Схемы связей и графы переходов автоматов $A1$ и $A2$

состояния реагирует (или никак не реагирует) на событие, опрашивает параметры объекта управления (входные переменные), учитывает состояния вложенных автоматов, затем производит последовательность выходных воздействий, включая и те выходные воздействия, которые необходимо совершить при попадании в новое состояние, и только после этого переводится в новое состояние (которое в случае петли может быть тем же самым).

Выходное воздействие первого типа, направленное на объект управления, считается сразу же осуществлённым после его применения. Выходное воздействие второго типа, представляющее собой передачу управления с событием вызываемому автомату, считается выполненным только лишь после реакции вложенного автомата на это событие, которая заключается в том, что либо автомат переходит в новое состояние (срабатывает один из переходов вызываемого автомата), либо событие игнорируется вызываемым автоматом (ни один из переходов сработать не может). До тех пор пока выходное воздействие второго

типа не выполнится, работа (процесс перехода в новое состояние) главного автомата приостанавливается.

Важно отметить, что правила переходов, а точнее условия срабатывания переходов, должны удовлетворять условию детерминированности (или ортогональности), т. е. при поступлении некоторого события может быть готов к срабатыванию не более чем один переход. Если ни один переход в текущем состоянии при поступлении некоторого события сработать не может (условия переходов не выполняются), то событие игнорируется.

Рассмотрим ряд примеров **темпоральных свойств** автоматной модели системы управления банкоматом, заданных на естественном (русском) языке.

«Сброс в начальное состояние». Если нажата кнопка «Сброс» до перехода банкомата в состояние снятия денег, то в любом случае до того, как банкомат начнёт новый цикл работы (т. е. попросит вставить карту), все автоматы управления вернутся в свои начальные состояния.

«Корректное завершение». Если банкомат перешёл в состояние снятия денег, то он обязательно вернётся в начальное состояние, и до того, как он запросит вставить карту, автомат выдачи денег и автомат захвата карты вернутся в свои начальные состояния.

«Корректный сброс». После перехода банкомата в состояние «Снятие денег» вплоть до начального состояния «Ожидание карты» нельзя произвести операцию сброса (т. е. завершить транзакцию нажатием кнопки «Сброс»).

«Обнуление поля суммы». После того, как банкомат вышел из состояния «Выбор суммы» без применения кнопки «Сброс», поле суммы обязательно очищается прежде, чем банкомат опять вернётся в это же состояние.

«Возврат карты». Если карта была вставлена, то рано или поздно она будет возвращена (т. е. автомат карты $A2$ обязательно перейдёт в состояние «Карта на выдаче»), если в процессе работы использовались кнопки «Ввод» и «Сброс».

Важно заметить, что построение структуры Крипке, описывающей поведение автоматной модели (системы взаимодействующих автоматов), представляется довольно сложной задачей. В то же время, при предоставлении вместе с автоматной моделью описания структуры Крипке подавляющее большинство студентов успешно (и не без интереса) справляются с задачей формальной спецификации указанных выше темпоральных свойств на языках логик LTL и CTL. При этом ознакомление с технологией автоматного программирования требует небольшого количества времени, отведённого на самоподготовку. Всё это в большой мере способствует пониманию мотивации применения

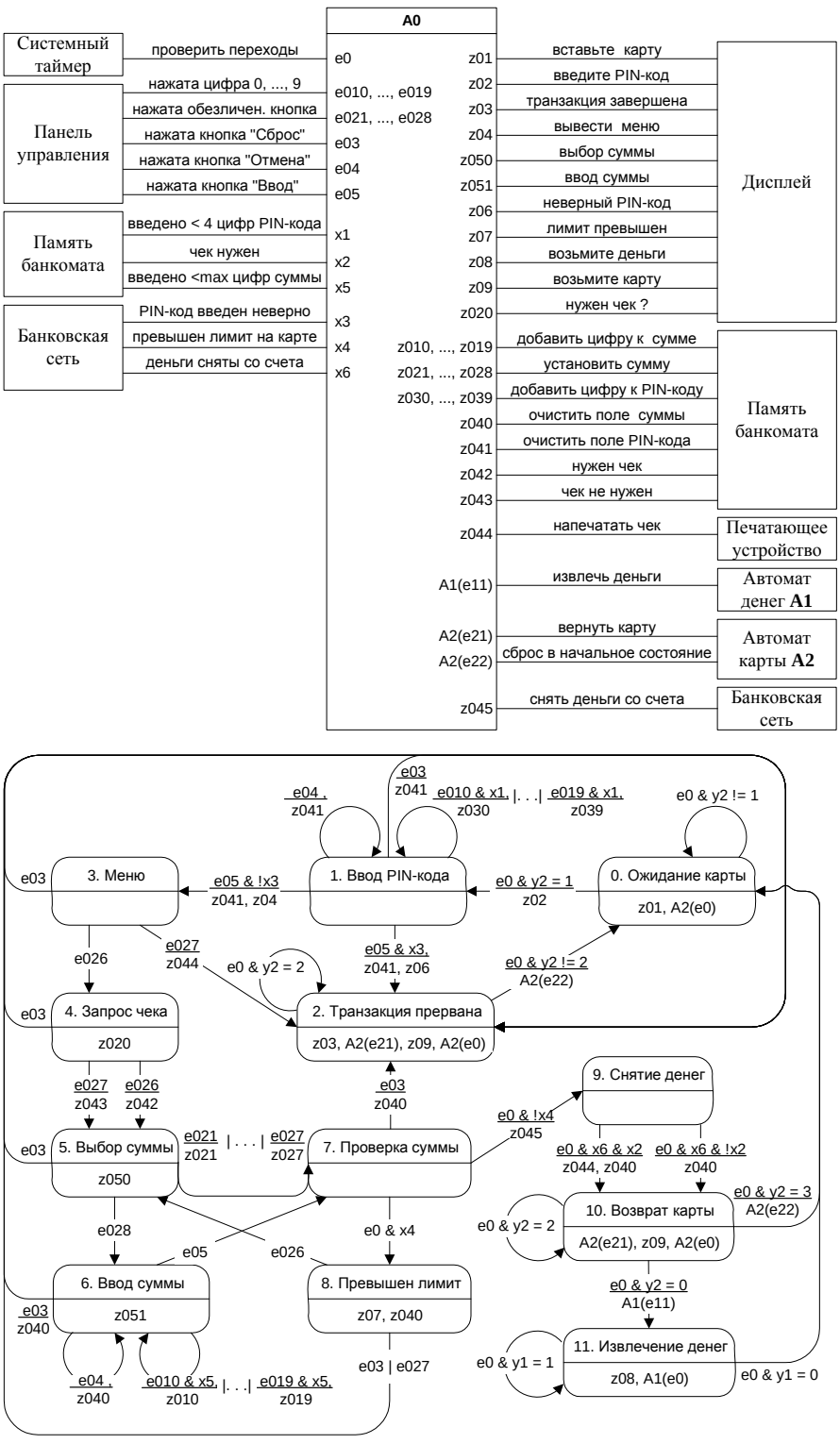

Рис. 4. Схема связей и граф переходов автомата управления банкоматом $A0$

темпоральных логик для спецификации и верификации программных систем.

После того, как поведенческая модель, т. е. структура Крипке, построена для системы взаимодействующих автоматов, и произведена спецификация свойств автоматной модели на языке темпоральной логики, следующими разделами, которые могли бы быть затронуты в рамках курса «Верификация программ», являются способы представления множества состояний структуры Крипке, алгоритмы проверки истинности формулы темпоральной логики для структуры Крипке, а также программы-верификаторы, разрабатываемые и поддерживаемые ведущими научно-исследовательскими лабораториями мира. Студентам предлагается самостоятельно изучить первые две тематики, отсылая их, например, к [2], а также применить для задачи рассмотренного только что вида, один из верификаторов по выбору. При этом важным является понимание принципов, на которых работает выбранная программа-верификатор, а также осознание ограничений, накладываемых в верификаторе на размер множества состояний поведенческой модели и вид формул используемой темпоральной логики. Кроме того, необходимо детально изучить интерфейсный формализм (взаимодействующие процессы, сети Петри, системы переходов специального вида и т. д.), с помощью которого в верификаторе задаётся входная проверяемая модель (структура Крипке), т. к. само представление исследуемой модели в рамках этого формализма может оказаться далеко не тривиальной задачей. Всё это требует от исследователя определённых творческих усилий при выполнении анализа корректности программной системы.

Заключение. Таким образом, в статье были рассмотрены тематики и задания, предлагаемые в рамках курса «Верификация программ». Выделены направления для самостоятельного изучения с учётом большого объёма внеаудиторных часов, отведённых на самоподготовку. Предложены и обоснованы задачи для самостоятельной и лабораторной работы, направленные на лучшее понимание материалов лекций и предмета дисциплины.

Список литературы

[1] *Грис Д.* Наука программирования /Д. Грис; пер. с англ. – М.: Мир, 1984. – 416 с.

[2] *Кларк Э.М., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. – М.: МЦНМО, 2002. – 416 с.

[3] *Кузьмин Е.В., Соколов В.А.* Структурированные системы переходов. – М.: ФИЗМАТЛИТ, 2006. – 178 с.

[4] *Кузьмин Е.В.* Иерархическая модель автоматных программ // Моделирование и анализ информационных систем. Т. 13, 1. ЯрГУ, 2006. С. 27–34.

[5] *Шалыто А.А.* Switch-технология. Алгоритмизация и программирование задач логического управления. – СПб.: Наука, 1998. – 628 с. – http://is.ifmo.ru/books/switch/1/

[6] *Шалыто А.А.* Автоматное проектирование программ. Алгоритмизация и программирование задач логического управления // Известия академии наук. Теория и системы управления. 2000. № 6. С. 63–81. (http://is.ifmo.ru, «Статьи»)

[7] *Шалыто А.А., Туккель Н.И.* SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. 2001. № 5. С. 45–62. – http://is.ifmo.ru/works/switch/1/