Моделирование и анализ информационных систем. Ярославль: ЯрГУ. Т.**14**, №**1** (2007) 3–14

УДК 519.68/.69

Верификация автоматных программ с использованием LTL

Васильева К. А., Кузьмин Е. В.
Ярославский государственный университет
150 000, Ярославль, Советская, 14

Аннотация

В статье рассматривается один из способов моделирования, спецификации и верификации программ, построенных на основе автоматного подхода к программированию. Технология автоматного программирования является достаточно эффективной при создании программного обеспечения для «реактивных» систем и систем логического управления. С точки зрения моделирования и анализа программных систем эта технология имеет ряд преимуществ по сравнению с традиционным подходом, так как исключает проблему адекватности построенной программной модели исходной программе. Набор взаимодействующих автоматов, описывающий логику программы, уже является адекватной моделью, по которой формальным образом строится программный модуль. Свойства программной системы в виде автоматов могут быть сформулированы и специфицированы естественным и понятным образом. Проверка свойств осуществляется в терминах, которые естественно вытекают из автоматной модели программы. Практическим результатом работы является применение инструментального средства SPIN и логики LTL для спецификации и верификации иерархических автоматных программ.

1. Введение

При автоматном подходе к проектированию и построению программ выделяются две части: системно независимая и системно зависимая [1, 2, 3, 4]. 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.

В статье рассматривается технология спецификации и верификации автоматных программ, построенных по иерархической модели [5], которая основывается на применении метода проверки модели (метода model checking) [6, 7]. Исследуется возможность спецификации (с точки зрения простоты и адекватности восприятия) структурных и семантических свойств автоматных программ с помощью темпоральной логики LTL. Оценивается удобство и целесообразность применения для анализа корректности автоматных программ широко известного LTL верификатора SPIN. Предлагаемая технология спецификации и верификации автоматных программ, построенных по иерархической модели, демонстрируется на примере системы управления банкоматом [8].

2. Автоматная модель системы управления банкоматом

Рассмотрим пример системно независимой части 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 «Введите Ваш персональный PIN-код». С помощью оцифрованных (см. рис. 1) 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. Если указанная сумма корректна, 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.

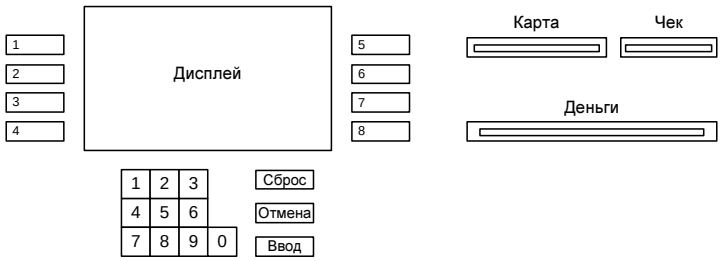

Рис. 1. Пользовательский интерфейс банкомата

Банкомат разбивается на чётко выраженные подобъекты управления (пользовательский интерфейс, механизмы захвата карты и выдачи денег), которые имеют свои собственные подсистемы управления. Каждая подсистема управления представлена в виде конечного смешанного автомата. Автоматы выстраиваются в иерархию по вложенности. В результате логическая часть системы управления банкоматом имеет вид иерархической модели системы взаимодействующих автоматов [5]. Автомат находящийся выше по иерархии, управляет своими вложенными автоматами путем генерации событий и передачи им управления с этими событиями. Кроме того, автомат следит за состояниями вложенных автоматов, так как от них могут зависеть его собственные переходы по состояниям. На рис. 2 представлена схема взаимодействия автоматов.

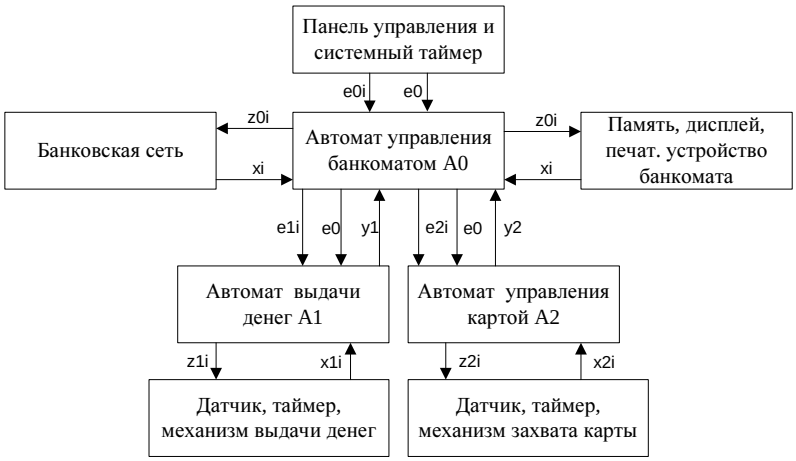

Рис. 2. Схема взаимодействия автоматов управления

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 вложенные 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 представлена на рис. 3.

Автомат $A1$ взаимодействует с автоматом управления $A0$, передавая последнему своё текущее состояние и получая от него события «извлечь деньги» и «проверить переходы». Также 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. 4.

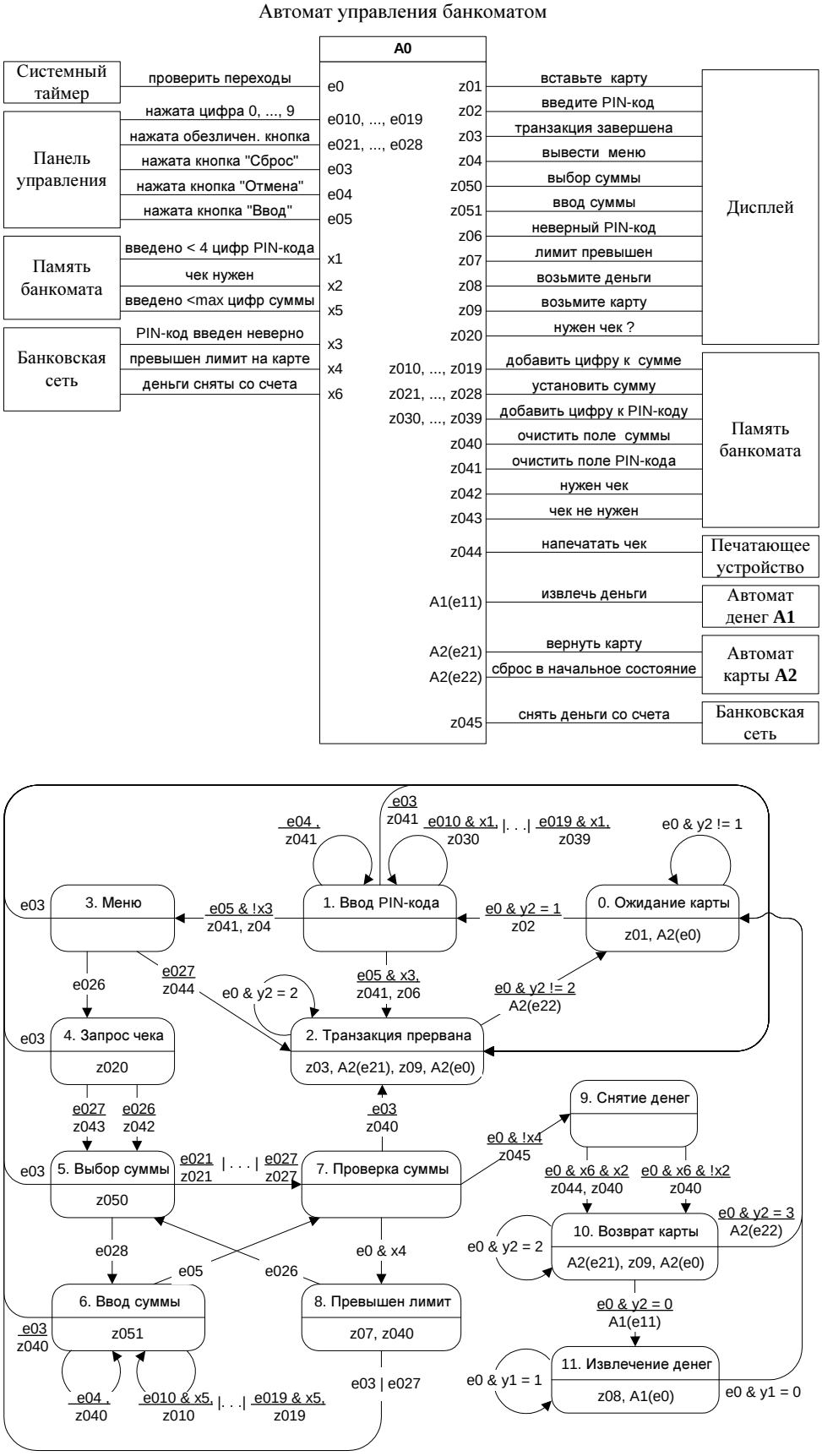

Рис. 3. Схема связей и граф переходов автомата управления банкоматом $A0$

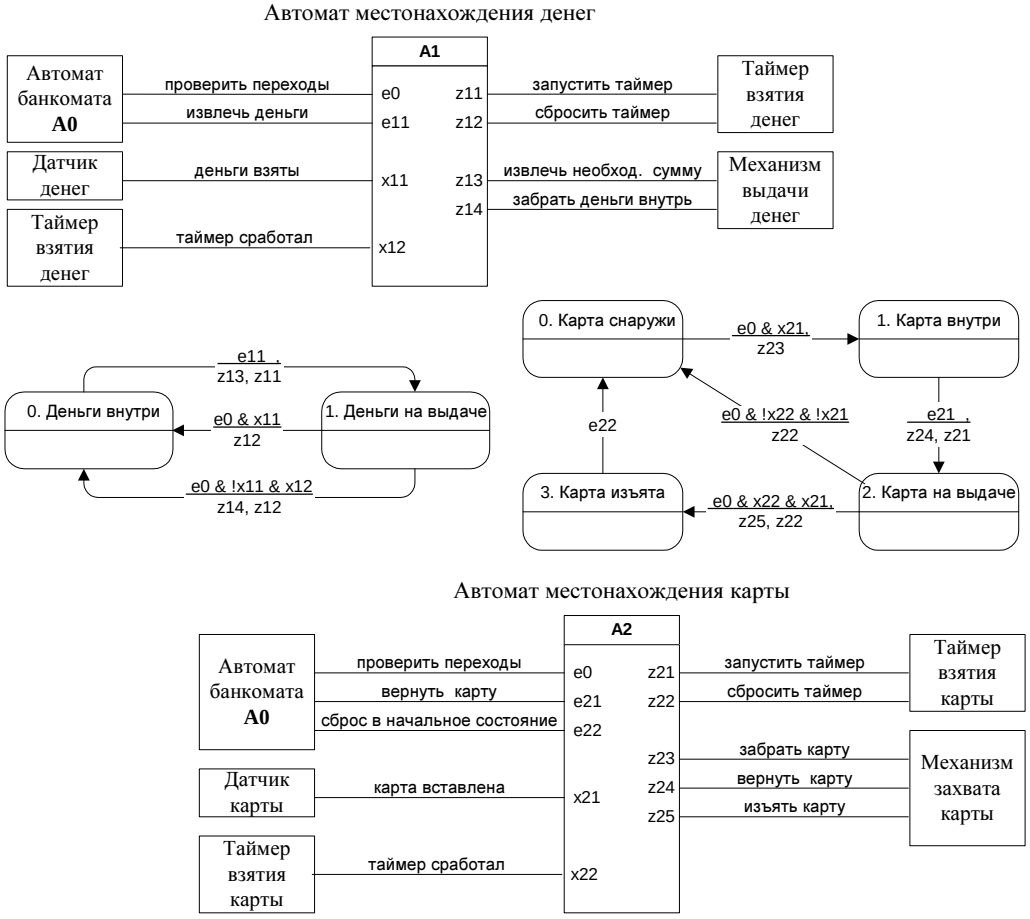

Рис. 4. Схемы связей и графы переходов автоматов $A1$ и $A2$

3. Спецификация и верификация автоматных моделей

При построении автоматной программы в рамках иерархической модели логика программы сосредотачивается в основном автомате, который распределяет управление вложенным автоматам в зависимости от поведения управляемого объекта. Каждый автомат программы взаимодействует только со своими главным и вложенными автоматами, что облегчает понимание программы. Во время проектирования или верификации такой автоматной программы возможно рассмотрение части или некоторого поддерева системы автоматов в зависимости от той функции, которая реализуется выделенной подсистемой автоматов. Любая подсистема взаимодействующих автоматов в иерархической модели представляет собой дерево автоматов, которое можно рассматривать как отдельную систему (как отдельную автоматную программу). Это позволяет менять масштаб всей системы, относя не интересующие проектировщика в данный момент времени автоматы к внешней среде, т.е. к внешнему объекту управления, и удерживать во внимании только анализируемую подсистему. И, наконец, при верификации спецификация и анализ свойств автоматной программы проводятся проще при понятной и не сложной по структуре верифицируемой модели.

Автоматная программная модель является очень удобным объектом для автоматической верификации методом проверки модели (model checking) [6, 7], смысл которого состоит в следующем.

Поведение программной модели описывается конечной системой переходов, которая называется структурой Крипке. Конечность системы переходов означает, что система имеет конечное число состояний. При некотором упрощении конечную систему переходов можно представить как конечный ориентированный граф с явно выделенной начальной вершиной.

Далее в рамках структуры Крипке с использованием языка темпоральной логики специфицируются свойства программной модели, истинность которых для этой модели затем и проверяется. Используя такие простые объекты, как конечная система переходов и формулы темпоральной логики, возможно автоматическим способом выполнение проверки свойств модели, заданных в виде формул.

4. Структура Крипке автоматной модели

Структурой Крипке над множеством элементарных высказываний P называется система переходов $\mathcal{S} = (S, s_0, \rightarrow, L)$, где

- S — конечное множество состояний;
- $s_0 \in S$ — начальное состояние;
- $\rightarrow \subseteq S \times S$ — тотальное отношение переходов (тотальность означает, что для каждого состояния $s \in S$ должно существовать состояние $s' \in S$, для которого имеет место $(s, s') \in \rightarrow$, т. е. $s \rightarrow s'$);
- $L : S \rightarrow 2^P$ - функция, помечающая каждое состояние множеством элементарных высказываний, истинных в этом состоянии.

Путь в структуре Крипке из состояния s_0 — это бесконечная последовательность состояний $\pi = s_0 s_1 s_2 \ldots$ такая, что для всех $i \geq 0$ выполняется $s_i \rightarrow s_{i+1}$. Для некоторого пути $\pi = s_0 s_1 s_2 s_3 \ldots$ обозначим π^i суффикс π, который получается удалением из π первых i состояний - например, $\pi^1 = s_1 s_2 s_3 \ldots$, а $\pi(i)$ будет обозначать i-е состояние пути, $\pi(0) = s_0$, $\pi(1) = s_1$ и т. д.

Рассмотрим для произвольной программы, построенной на принципах автоматного программирования, её иерархическую автоматную модель $\mathcal{A}$, представляющую собой набор взаимодействующих автоматов $(A_0, A_1, \ldots, A_n)$. Для этой автоматной модели $\mathcal{A}$ построим структуру Крипке, которая, с одной стороны, описывает все возможные состояния системы $\mathcal{A}$, а с другой стороны, задаёт семантику элементарных высказываний, истинных в этих состояниях.

Для этого выделим в каждом автомате A_k, помимо основных состояний, множество его промежуточных состояний, в которых автомат пребывает во время перехода из одного основного состояния в другое. Промежуточное состояние перехода автомата будем фиксировать каждый раз, когда автомат совершит одно из элементарных действий, т.е. автомат отреагирует на некоторое событие e_k, обратится к объекту управления с запросом значений входных переменных x_k ($!x_k$) или произведёт некоторое выходное воздействие на объект управления или вложенный автомат z_k.

Продемонстрируем идею выделения промежуточных состояний для перехода произвольного автомата A_k (рис. 5).

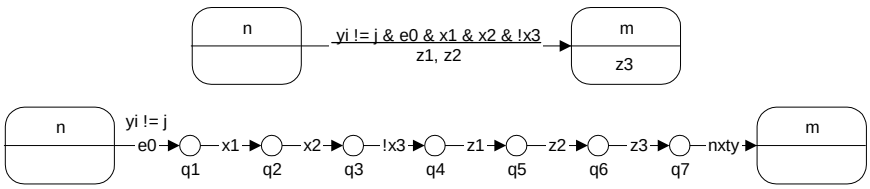

Рис. 5. Выделение промежуточных состояний для автоматного перехода

Промежуточный переход с пометкой $nxty$ вводится для того, чтобы иметь возможность непосредственно отслеживать момент перехода вложенного автомата в следующее основное состояние с одновременной передачей управления главному автомату (в случае промежуточного перехода $nxty$ основного автомата, активным остаётся по-прежнему основной автомат). Внутренний переход с пометкой $e0$ может произойти при наступлении события $e0$, если выполняется условие $yi != j$. Все остальные переходы не имеют условий и являются активными при попадании автоматом в соответствующие промежуточные состояния. В указанной на рисунке последовательности переходов по внутренним состояниям обязательно сначала идёт переход, соответствующий входному событию, далее переходы с пометками входных запросов, а затем переходы, помеченные выходными воздействиями.

Далее для произвольного автомата под переходом в следующее состояние будем предполагать *внутренний* переход в основное или промежуточное состояние.

Каждому автомату A_k из набора $\mathcal{A}$ сопоставим переменные y_k, xm_k, zd_k, ev_k и st_k $(0 \leq k \leq n)$. А для всей системы автоматов $\mathcal{A}$ в целом введём переменные ev, xm, zd, act, $auto$ и $evnt$. Каждая из введённых переменных имеет следующий смысл.

1. Переменная y_k хранит последнее основное состояние автомата A_k.
2. Переменная xm_k содержит последний выполненный запрос параметров объекта управления. Значениями xm_k являются имена входных воздействий в прямой или инверсной форме, т.е. xm_k может содержать либо x, либо $!x$, где x принадлежит множеству X_{A_k} входных запросов автомата A_k.

3. Переменная zd_k используется для хранения имени последнего выполненного выходного воздействия $z \in Z_{A_k}$, где Z_{A_k} — множество выходных воздействий автомата A_k.

4. Переменная ev_k содержит имя последнего обработанного автоматом A_k входного события $e \in E_{A_k}$, т.е. имя такого события, на которое A_k отреагировал; E_{A_k} — множество событий для автомата A_k.

5. Переменная st_k хранит текущее состояние автомата A_k. Значениями st_k являются как основные, так и промежуточные состояния автомата.

6. Переменная $auto$ содержит номер активного в данный момент времени автомата из набора $\mathcal{A}$, т.е. номер последнего автомата, получившего управление.

7. Переменная $evnt$ хранит имя поступившего на обработку входного события для вложенного автомата, которому было передано управление. Значениями переменной являются имена входных событий всех вложенных автоматов из $\mathcal{A}$ и пустое значение 0. Переменная $evnt$ принимает значение 0 тогда, когда входное событие было обработано или же проигнорировано автоматом (если в текущем состоянии автомат не может отреагировать на данное входное событие). В переменную $evnt$ записывается имя некоторого события, если произошла передача управления с этим событием от главного автомата к вложенному.

8. Переменная act используется для хранения имени последнего элементарного действия, произошедшего в системе автоматов $\mathcal{A}$. Значениями переменной act могут быть имена входных событий $e \in E_A$, входных запросов x и $!x$, где $x \in X_A$, выходных воздействий $z \in Z_A$, а также имена специальных действий 0, $nxty$ и end. После перехода активного автомата в новое промежуточное или основное состояние переменная act содержит метку этого перехода. Переменная act принимает значение 0, когда вложенный автомат, находясь в основном состоянии, не может обработать входное событие и игнорирует его. При этом автомат совершает пустое действие 0 и переходит в то же самое основное состояние, в котором находился. Если $act = nxty$, то это означает, что некоторый вложенный автомат перешёл в своё следующее основное состояние с одновременной передачей управления главному автомату. Значение end используется для идентификации тупикового состояния всей системы автоматов. Для того чтобы выполнить условие тотальности отношения переходов для структуры Крипке, устанавливается, что из тупикового состояния (структуры Крипке) есть всего лишь один переход сам в себя и при этом совершается действие $act = end$.

9. Переменные ev, xm и zd используются для хранения имён последних выполненных входных событий, входных запросов и выходных воздействий соответственно в рамках всей системы $\mathcal{A}$ в целом.

Тогда состоянием s структуры Крипке $\mathcal{S}_\mathcal{A}$ автоматной модели $\mathcal{A}$ является вектор значений переменных

$$(act, auto, evnt, ev, xm, zd, ev_0, xm_0, zd_0, y_0, st_0, \ldots, ev_n, xm_n, zd_n, y_n, st_n)$$

В начальном состоянии s_0 структуры Крипке $\mathcal{S}_\mathcal{A}$ все переменные y_i и st_i содержат начальные состояния соответствующих автоматов A_i, переменная $auto = 0$, т. е. активным является основной автомат A_0, $evnt$ не содержит событий и установлена в 0, а всем остальным переменным присваивается начальное значение $intl$, которое вводится специально для инициализации и далее не используется.

Отношение переходов структуры Крипке $\mathcal{S}_\mathcal{A}$ автоматной модели $\mathcal{A}$ определяется в соответствии с поведением системы автоматов $\mathcal{A}$.

1. Переход системы соответствует только одному из внутренних переходов некоторого автомата из $\mathcal{A}$.

2. При выполнении условий перехода некоторый автомат A_k может совершить этот переход, если в данном состоянии он является активным (ему передано управление), что отражает значение переменной $auto = k$.

3. Для активного вложенного автомата A_k переход с пометкой e, где e — некоторое входное событие, может произойти, если переменная $evnt = e$ и выполняется условие для этого перехода — истинен предикат над значениями переменных состояний y_i, сопоставленный переходу. После срабатывания этого перехода изменяются значения переменных act, $evnt$, ev, ev_k и st_k. Переменная $evnt$ принимает нулевое значение 0, означающее, что событие было обработано, а переменным act, ev и ev_k присваивается имя события e. В st_k помещается внутреннее состояние, в которое произошёл переход. Если активным автоматом является основной автомат A_0, то переход с пометкой e происходит при выполнении условия перехода, а $evnt = 0$.

4. Если вложенный автомат A_k получил от главного автомата управление с событием e, которое находится в переменной $evnt$, но не может на него отреагировать — не существует из данного основного состояния перехода с пометкой e или для него не выполнены условия перехода, то происходит пустое действие $act := 0$, переменная $evnt$ обнуляется и управление передаётся главному автомату — в переменную $auto$ заносится номер главного автомата, а все остальные переменные не изменяют своих значений.

5. Если произошёл переход с пометок x, где x — некоторый входной запрос в прямой или инверсной форме (x может иметь вид $!x'$), то происходят изменения следующих переменных: $act := x$, $xm := x$, $xm_k := x$, а st_k получает значение внутреннего состояния, в которое был сделан переход.

6. При срабатывании перехода автомата A_k с пометкой z, где z — выходное воздействие, происходят присваивания $act := z$, $zd := z$, $zd_k := z$, и st_k получает значение внутреннего состояния, в которое был сделан переход. Более того, если z принадлежит к выходным воздействиям второго типа, т.е. $z = A_{k'}(e)$, где $A_{k'}$ — вложенный автомат, а e — передаваемое вместе с управлением входное событие для автомата $A_{k'}$, то $evnt$ получает значение e, и активным становится вложенный автомат $A_{k'}$ за счёт присваивания $auto := k'$.

7. Если произошёл переход с пометкой $nxty$ в основное состояние j во вложенном автомате $A_{k'}$, то происходят присваивания $act := nxty$, $y_{k'} := j$, $st_{k'} := j$ и передача управления главному автомату A_k через $auto := k$. Если переход $nxty$ произошёл в основном автомате A_0, то переменная $auto = 0$ остаётся без изменения.

8. Если из некоторого состояния автомата A_k существует переход, который помечен переменными z, x, $!x$ или $nxty$, то для его срабатывания не требуется дополнительных условий, кроме активности этого автомата $auto = k$.

9. Если система взаимодействующих автоматов $\mathcal{A}$ попала в тупиковое состояние, что равносильно невозможности совершить переход из основного состояния основного автомата A_0 из-за нарушения условий переходов, то происходит специально введённый переход $act := end$, который ведет в то же самое состояние, а значения всех остальных переменных остаются без изменений.

Благодаря построенной структуре Крипке при спецификации и верификации имеется возможность в качестве элементарных высказываний использовать предикаты над значениями введённых переменных, что позволяет выражать следующие свойства состояний автоматных моделей.

1. В каждом состоянии автоматной модели существует возможность отслеживать, какое последнее действие произошло перед попаданием в текущее состояние с помощью переменной act.

2. Имеется возможность определения типа последнего действия, т. е. произошло ли входное событие, входной запрос или выходное воздействие, посредством выражений $act = ev$, $act = xm$ и $act = zd$. Например, если в текущем состоянии автоматной модели выполняется $act = zd$, это означает, что последним действием, которое произошло при переходе в данное состояние, является некоторое выходное воздействие. Уточнить, к какому автомату системы $\mathcal{A}$ принадлежит выходное воздействие, можно с помощью выражений вида $act = zd_i$. Наконец, истинность выражения $act = z$ означает, что последним действием при переходе в текущее состояние было выходное воздействие z.

3. Через переменную $auto$ в каждом состоянии автоматной модели определяется активный в данный момент времени автомат. Например, если в состоянии системы $\mathcal{A}$ выполняется $auto = 0$, это означает, что активным является основной автомат A_0.

4. Для каждого автомата A_i в текущем состоянии системы $\mathcal{A}$ можно узнать через переменную y_i последнее основное состояние, в котором он пребывал. Более того, выражение $y_i = st_i$ означает, что автомат A_i в данном состоянии системы $\mathcal{A}$ находится в своём основном состоянии.

5. С помощью выражения $act = end$ можно отслеживать тупиковые состояния системы автоматов $\mathcal{A}$. А также через $act = nxty$ отслеживается переход одного из автоматов системы в своё новое основное состояние с одновременной передачей управления главному автомату.

5. Темпоральная логика LTL для автоматной модели

Одними из наиболее популярных темпоральных логик для спецификации и верификации свойств программных систем являются логика CTL (branching-time logic или computation tree logic) и логика линейного времени LTL (linear-time logic). Для целей верификации автоматных программ логика LTL заслуживает особого внимания, поскольку любая формула в рамках этой логики, по сути, представляет собой автомат Бюхи, описывающий (принимающий) бесконечные допустимые пути структуры Крипке, которая в свою очередь задаёт поведение (все возможные исполнения) проверяемой на корректность автоматной программы. Это позволяет при верификации и спецификации автоматных программ оперировать в основном таким простым понятием, как «автомат». Таким образом, можно в некотором смысле говорить, что LTL является более естественным средством спецификации свойств автоматных программ.

Формулы логики LTL для структуры Крипке $\mathcal{S}_\mathcal{A}$ автоматной модели $\mathcal{A}$ строятся по следующей грамматике:

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid p \in P \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi\, U\, \varphi \mid \varphi\, \widetilde{U}\, \varphi \mid \langle\,\rangle\, \varphi \mid [\,]\, \varphi,$$

где $p \in P$ — элементарное высказывание, определенное над множеством состояний автоматной модели $\mathcal{A}$.

Формулы темпоральной логики линейного времени интерпретируются через исполнения системы переходов (структуры Крипке). Поведение системы удовлетворяет формуле φ логики LTL, если все пути, выходящие из начального состояния, удовлетворяют данной формуле φ. Для структуры Крипке $\mathcal{S}_\mathcal{A} = (S, s_0, \to, L)$ обозначим $\Omega_\mathcal{S}$ множество всех возможных путей, выходящих из начального состояния s_0, а Ω — множество всех возможных путей из любого состояния.

Отношение выполнимости $\models$ формулы φ логики LTL для некоторого пути $\pi = s_0 s_1 s_2 s_3 \ldots \in \Omega$ структуры Крипке $\mathcal{S}_\mathcal{A}$ индуктивно определяется следующим образом:

- $\pi \models \mathit{true}$ и $\pi \not\models \mathit{false}$;
- $\pi \models p$ для $p \in P \iff p \in L(\pi(0))$;
- $\pi \models \neg\varphi \iff \pi \not\models \varphi$;
- $\pi \models \varphi \wedge \psi \iff \pi \models \varphi$ и $\pi \models \psi$;
- $\pi \models \varphi \vee \psi \iff \pi \models \varphi$ или $\pi \models \psi$;
- $\pi \models X\,\varphi \iff \pi(0) \to \pi(1)$ и $\pi^1 \models \varphi$;
- $\pi \models \varphi\, U\, \psi \iff$ существует такое $j \geq 0$, что $\pi^j \models \psi$ и при этом для всех i, $0 \leq i < j$, $\pi^i \models \varphi$;
- $\pi \models \varphi\, V\, \psi \iff$ для любого $j \geq 1$ такого, что $\pi^j \not\models \psi$, существует i, $0 \leq i < j$, что $\pi^i \models \varphi$;
- $\pi \models \langle\,\rangle\, \varphi \iff \mathit{true}\, U\, \varphi$ — существует такое $j \geq 0$, что $\pi^j \models \varphi$;
- $\pi \models [\,]\, \varphi \iff \mathit{false}\, V\, \varphi$ — на протяжении всего пути π формула φ истинна.

Множество $[\![\varphi]\!]_\mathcal{S}$ представляет собой множество всех путей структуры Крипке $\mathcal{S}_\mathcal{A}$, для которых истинна формула φ, т. е. $[\![\varphi]\!]_\mathcal{S} = \{\pi \mid \pi \in \Omega, \pi \models \varphi\}$.

Структура Крипке $\mathcal{S}_\mathcal{A}$ будет удовлетворять формуле φ логики LTL, если $\Omega_\mathcal{S} \subseteq [\![\varphi]\!]_\mathcal{S}$.

Кроме введенных логических связок $\wedge$ и $\vee$ традиционно используются связки $\to$ и $\leftrightarrow$:

$$\varphi \to \psi \equiv \neg\varphi \vee \psi; \quad \varphi \leftrightarrow \psi \equiv (\varphi \to \psi) \wedge (\psi \to \varphi) \equiv (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi).$$

Далее наряду с символами $\neg$, $\wedge$ и $\vee$ будут также использоваться символы !, && и ||.

6. Темпоральные свойства автоматных моделей

Рассмотрим несколько примеров темпоральных свойств, которые являются общими для любой иерархической системы взаимодействующих автоматов.

«*Тупиковое состояние*». Свойство, описывающее невозможность попадания автоматной программы в тупиковое состояние, из которого нельзя выйти, применимо к модели любой автоматной программы. В рамках описанной структуры Крипке это свойство может быть описано на языке темпоральной логики LTL следующим образом:

`!<>(act == end)` или `[](act != end)`.

Эта формула означает, что на протяжении всех путей (из начального состояния) структуры Крипке не существует перехода с пометкой `end`. По построению структуры Крипке $\mathcal{S}_{\mathcal{A}}$ такой переход происходит тогда, когда нет возможности выйти из основного состояния автомата A_0 из-за нарушения условий переходов или вообще полном отсутствии переходов из данного основного состояния.

«*Тупиковое состояние вложенного автомата*». Предположим, что структура Крипке $\mathcal{S}_{\mathcal{A}}$ перешла в новое состояние посредством перехода некоторого вложенного автомата A_k из $\mathcal{A}$ в своё следующее основное состояние с одновременной передачей управления главному автомату. Далее, пусть на протяжении всех возможных путей из этого состояния структуры Крипке любое входное событие, которое передаётся вместе с управлением для автомата A_k, будет игнорироваться им либо из-за нарушений условий переходов, либо из-за отсутствия переходов, помеченных этим входным событием. Получается, что вложенный автомат попал в некоторое основное состояние, из которого он больше никогда не выйдет, несмотря на то, что регулярно получает управление. И это основное состояние в данный момент времени можно назвать тупиковым для автомата A_k, хотя в другой ситуации оно может и не быть таковым. Свойство, выражающее отсутствие тупикового состояния вложенного автомата A_k, задаётся следующим образом:

```
!<>([](auto == k -> act != evk) && []<>(auto == k)).
```

«*Забытый автомат*». Рассмотрим ещё одну разновидность тупикового состояния вложенного автомата. Начиная с некоторого момента времени вложенному автомату A_k никогда более не будет передано управление. Таким образом, автомат A_k с некоторого момента времени навсегда останется в одном из своих основных состояний. Свойство, выражающее отсутствие таких путей выглядит так (автомат никогда не будет забыт):

```
!<>([](auto != k) или []<>(auto == k).
```

Далее рассмотрим ряд примеров темпоральных свойств, относящихся к конкретной автоматной модели системы управления банкоматом.

«*Корректное завершение*». Если банкомат перешел в состояние снятия денег, то он обязательно вернется в начальное состояние, и до того, как он запросит вставить карту, автомат выдачи денег и автомат захвата карты вернутся в свои начальные состояния. Для любого пути структуры Крипке автоматной модели системы управления банкоматом должно выполняться следующее требование. Если система перейдет в состояние $y0 = 9$, то в будущем, когда произойдёт выходное воздействие $z01$, автомат выдачи денег и автомат захвата карты будут находиться в своих начальных состояниях $st1 = 0$ и $st2 = 0$. Заменив темпоральные связки темпоральными операторами, получим следующую формулу:

```
[](y0 == 9 -> (act != z01 U act == z01 && st1 == 0 && st2 == 0)).
```

«*Сброс в начальное состояние*». Если нажата кнопка «Сброс» до перехода банкомата в состояние снятия денег, то в любом случае до того, как банкомат начнет новый цикл работы (т. е. попросит вставить карту), все автоматы управления вернутся в начальные состояния. Это свойство запишется следующим образом:

```
[](y0 != 9 && y0 != 10 && y0 != 11 && act == e03 ->
(act != z01 U act == z01 && st1 == 0 && st2 == 0)).
```

«*Корректный сброс*». После перехода банкоматом в состояние «9. Снятие денег» вплоть до начального состояния «0. Ожидание карты» нельзя произвести операцию сброса (т. е. завершить транзакцию нажатием кнопки «Сброс»):

```
[](st0 == 9 -> (act != e03 && st0 != 0 U st0 == 0)).
```

«*Обнуление поля суммы*». После того, как банкомат вышел из состояния «5. Выбор суммы» без применения кнопки «Сброс», поле суммы обязательно очищается прежде, чем банкомат опять вернётся в это же состояние:

```
!<>(y0 == 5 && act == ev0 && act != e03 && (act != z040 && st0 != 5 U st0 == 5)).
```

«*Возврат карты*». Если карта была вставлена, то рано или поздно она будет возвращена (т. е. автомат карты $A2$ обязательно перейдёт в состояние «2. Карта на выдаче»), если в процессе работы использовались кнопки «Ввод» и «Сброс»:

```
!<>(st2 == 1 && [](st2 != 2) && []<>(act == e03) && []<>(act == e05)).
```

«*Корректная работа после изъятия*». Если произошло изъятие карты или денег, все автоматы вернуться в свои начальные состояния до того, как будет выведено приглашение вставить карту.

```
[]((act == z25 || act == z24) ->
(act != z01 U act == z01 && st1 == 0 && st2 == 0)).
```

Спецификация свойств в рамках логики LTL осуществляется двумя способами. Либо с помощью формулы логики LTL описываются все допустимые пути структуры Крипке, либо задаётся путь, которого не должно существовать в модели, а затем строится его отрицание.

Интересно заметить, что свойства «Сброс в начальное состояние», «Корректное завершение», «Корректный сброс» и «Корректная работа после изъятия» имеют общую структуру:

```
[](p1 -> (p2 U p3)),
```

где p_i — элементарное высказывание. В связи с этим важным является вопрос о *шаблонах* (структуре) темпоральных свойств, наиболее применимых и адекватных для верификации автоматных программ. Наличие таких шаблонов позволяло бы говорить о *классах* темпоральных свойств автоматных моделей, что, несомненно, облегчало бы построение технологической схемы проверки автоматных программ на корректность относительно спецификации.

7. Верификация автоматных программ в системе SPIN

Определение структуры Крипке иерархической автоматной модели и описанная спецификация свойств позволяют применить для верификации автоматных программ метод проверки модели. Для этого в дальнейшем целесообразно использовать уже существующие пакеты прикладных программ-верификаторов, которые разрабатываются и поддерживаются ведущими научными лабораториями на протяжении довольно длительного времени (более десяти лет). Одним из таких средств верификации является система SPIN [9], разрабатываемая в лаборатории Bell.

SPIN — это система верификации моделей для логики LTL на лету с использованием явного перечисления состояний и редукции частичных порядков, представляющее собой инструментальное средство, которое используется главным образом для верификации асинхронных программных систем и, в частности, коммуникационных протоколов.

Входным языком системы SPIN для описания моделей и спецификации свойств является язык Promela. В состав этого языка входят синтаксические конструкции нескольких различных языков программирования. Логические и арифметические выражения языка Promela унаследованы от языка C. Синтаксис канального взаимодействия процессов ориентирован на CSP (Communicating Sequential Processes) Хоара. Условные операторы и операторы цикла основаны на охраняемых командах Дейкстры; их синтаксис приведён ниже.

```
if                              do
  :: guard1 -> S1                 :: guard1 -> S1
  :: guard2 -> S2                 :: guard2 -> S2
  ...                             ...
  :: else -> Sn                   :: else -> Sn
fi                              od
```

При использовании оператора if и при каждой итерации оператора цикла do недетерминированно выбирается одна из истинных охран guardi и выполняется соответствующая ей команда Si. Если все охраны ложны, то происходит переход к команде Sn по оператору else. Допускается использование этих конструкций и без else.

Рассмотрим ниже, каким образом происходит описание поведения автоматных моделей (задание структуры Крипке) в рамках языка Promela на примере автомата управления механизмом выдачи денег $A1$ системы управления банкоматом.

Далее при использовании конструкции atomic изменение значений переменных внутри фигурных скобок происходит одновременно и расценивается системой SPIN как один шаг, один переход в новое состояние модели.

Заметим, что слово auto в языке Promela является зарезервированным, поэтому мы используем переменную automaton для того, чтобы отслеживать, какой из автоматов в данный момент является активным.

```
proctype A1() { /* описание поведения модели автомата A1 */
 do
  :: automaton == 1 -> /* автомат A1 является активным */
     if  /* основные состояния */
       :: st1==0 -> if
                      :: evnt==e11 -> atomic{act=e11;evnt=0;ev=e11;ev1=e11;st1=2;}
                      :: else -> atomic{act=0; evnt=0; automaton=0;}
                    fi
       :: st1==1 -> if
                      :: evnt==e0 -> atomic{act=e0;evnt=0;ev=e0;ev1=e0;st1=7;}
                      :: else -> atomic{act=0; evnt=0; automaton=0;}
                    fi
        /* дополнительные состояния */
       :: st1==2 -> atomic{act = z13; zd = z13; zd1 = z13; st1 = 3};
       :: st1==3 -> atomic{act = z11; zd = z11; zd1 = z11; st1 = 4};
       :: st1==4 -> atomic{act = nxty; y1 = 1; st1 = 1; automaton = 0};
       :: st1==5 -> atomic{act = nxty; y1 = 0; st1 = 0; automaton = 0};
       :: st1==6 -> atomic{act = z12; zd = z12; zd1 = z12; st1 = 5};
       :: st1==7 -> if
                      :: atomic{act = x11; xm = x11; xm1 = x11; st1 = 6};
                      :: atomic{act=no_x11; xm = no_x11; xm1 = no_x11; st1 = 10};
                    fi
       :: st1==8 -> atomic{act = z12; zd = z12; zd1 = z12; st1 = 5};
       :: st1==9 -> atomic{act = z14; zd = z14; zd1 = z14; st1 = 8};
       :: st1==10 -> atomic{act = x12; xm = x12; xm1 = x12; st1 = 9};
     fi
 od }
```

Проверка истинности темпорального свойства, представленного в виде формулы φ логики LTL, для модели (описанной на языке Promela) происходит следующим образом.

Отрицание проверяемого свойства автоматически преобразуется в автомат Бюхи. Этот автомат описывается при помощи специальной синтаксической конструкции языка Promela, которая называется never claim. Такое название объясняется тем, что автомат полученный в результате трансляции отрицания проверяемого свойства, задаёт такие пути структуры Крипке, которые не должны существовать в поведении модели.

Ниже приводится конструкция never claim для свойства «Корректное завершение» системы управления банкоматом.

```
#define p1 (y0 == 9)
#define p2 (act != z01)
#define p3 (act == z01 && st1 == 0 && st2 == 0)
never {    /* ![](p1 -> (p2 U p3)) */
  T0_init:
           if
             :: (! ((p3)) && (p1)) -> goto accept_S4
             :: (! ((p2)) && ! ((p3)) && (p1)) -> goto accept_all
             :: (1) -> goto T0_init
           fi;
  accept_S4:
           if
             :: (! ((p3))) -> goto accept_S4
             :: (! ((p2)) && ! ((p3))) -> goto accept_all
           fi;
  accept_all: skip }
```

Метка каждой начальной вершины содержит слово init, а метка каждой допускающей вершины включает в себя слово accept.

Система SPIN строит пересечение автомата never claim и автомата (структуры Крипке), выделенного из программы (написанной на Promela). Пересечение строится «на лету», не ожидая полного построения структуры Крипке. Если пересечение не пусто, то выдаётся трасса ошибки.

Необходимо отметить ограничения, которые накладываются на формулы логики LTL в рамках системы SPIN. При выделении структуры Крипке автоматной модели из Promela-программы система SPIN порождает и вспомогательные служебные состояния такие, как вход и выход из конструкций do и if, которые нарушают описанное выше поведение автоматной модели (структуру Крипке). В связи с этим не рекомендуется использовать в формулах логики LTL оператор X (Next), так как на проверку истинности темпорального свойства, выраженного без этого оператора, служебные состояния (встроенные в «чистую» структуру Крипке) никакого влияния не оказывают, в противном случае говорить об адекватной проверке свойства не приходится.

Впрочем, как показывает практика, в большинстве случаев удаётся успешно проводить LTL спецификацию автоматных моделей и без применения оператора X. В частности, истинность всех рассмотренных выше темпоральных свойств системы управления банкоматом была эффективно подтверждена с помощью системы SPIN.

8. Заключение

В работе для верификации автоматных программ предлагается использовать инструментальное средство SPIN. SPIN является бесплатной и сравнительно небольшой программой, дистрибутив которой размером 300 KB легко доступен на сайте [9]. Этот инструмент успешно используется многими известными организациями в различных целях. В перечень клиентов входят NASA, научные центры вооруженных сил разных стран, крупнейшие производители аэрокосмической отрасли. Существует обширное мировое сообщество пользователей SPIN, каждый год (начиная с 1995 г.) проводятся международные конференции пользователей.

Иерархические модели автоматных программ являются очень наглядными и простыми в построении. В дальнейшем предполагается разработка автоматического перевода иерархической системы взаимодействующих автоматов в текст программной модели на языке Promela.

Все вышесказанное дает уверенность в целесообразности использования инструментального средства SPIN для практической реализации верификации программного обеспечения «реактивных» систем и систем логического управления, написанного с применением автоматного подхода к программированию.

Список литературы

1. Шалыто А. А. Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с. (http://is.ifmo.ru/books/switch/1/).

2. 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»).

3. Шалыто А. А. Алгоритмизация и программирование для задач логического управления и «реактивных» систем // Автоматика и телемеханика. Обзоры. 2001. №1. С. 3–39. (http://is.ifmo.ru, «Статьи»).

4. 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/).

5. Кузьмин Е. В. Иерархическая модель автоматных программ // Моделирование и анализ информационных систем. 2006. Т. 13 (1). С. 27–34.

6. Кларк Э. М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. М.: МЦНМО, 2002. 416 с.

7. Кузьмин Е. В., Соколов В. А. Структурированные системы переходов. М.: ФИЗМАТЛИТ, 2006. 178 с.

8. Первушин Е. В., Шалыто А. А. Моделирование банкомата // СПбГУ ИТМО, 2003. http://is.ifmo.ru/projects/.

9. SPIN. http://spinroot.com/spin/whatispin.html.