Модел. и анализ информ. систем. Т.**13**, №**2** (2006) 4–15

УДК 519.68/.69

Верификация автоматных программ средствами CPN/Tools

Виноградов Р.А., Кузьмин Е.В., Соколов В.А.
Ярославский государственный университет
150 000, Ярославль, Советская, 14

получена 15 июля 2006

Аннотация

В работе предлагается технология спецификации и верификации автоматных программ. Приводятся обнаруженные ошибки в автоматной модели системы управления кофеваркой и вариант их исправления.

1. Введение

Статья посвящена спецификации и верификации моделей программ, построенных на основе автоматного подхода к программированию [1, 2, 3, 4, 5]. Технология автоматного программирования является достаточно эффективной при построении программного обеспечения для «реактивных» 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. Более того, автоматный подход к программированию с точки зрения моделирования и анализа программных систем имеет ряд преимуществ по сравнению с традиционным подходом. При построении модели для программы, написанной традиционным способом, возникает серьезная проблема адекватности этой программной модели исходной программе. Модель может не учитывать ряд программных свойств или порождать несуществующие свойства. При автоматном программировании такая проблема исключена, поскольку набор взаимодействующих автоматов, описывающий логику программы, уже является адекватной моделью, по которой формально и изоморфно строится программный модуль. И это является бесспорным плюсом автоматной технологии. К тому же свойства программной системы в виде автоматов формулируются и специфицируются естественным и понятным образом. 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 задается 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 [6], которая основывается на применении метода model checking [7, 8]. Исследуется возможность спецификации (с точки зрения простоты и адекватности восприятия) структурных и семантических свойств автоматных программ с помощью темпоральной логики CTL [7, 8]. Оценивается удобство и целесообразность применения для анализа корректности автоматных программ выбранного верификатора CPN/Tools [9, 10, 11].

В качестве инструментального средства для написания автоматных программ выбран UniMod [12, 13]. Придуман и реализован механизм трансляции автоматных программ в CPN-модель. Предлагаемая технология спецификации и верификации автоматных программ, построенных по иерархической модели, опробована на примере системы управления кофеваркой [6, 14]. В статье приведены обнаруженные в модели кофеварки ошибки, также предлагается вариант их исправления.

2. Структура Крипке и темпоральная логика CTL

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. С использованием таких простых объектов, как конечная система переходов и формулы темпоральной логики, возможно автоматическим способом выполнение проверки свойств модели, заданных в виде формул.

Темпоральные логики играют важную роль в формальной верификации. Они используются для выражения свойств процессов. Такими свойствами, например, являются свойства «*процесс никогда не достигнет тупикового состояния*» или «*в любом бесконечном исполнении процесса действие b происходит бесконечно часто*».

Важным вопросом спецификации является полнота. Метод проверки модели дает возможность убедиться, что модель проектируемой системы соответствует заданной спецификации, однако определить, охватывает ли заданная спецификация все свойства, которыми должна обладать система, невозможно.

Задача проверки модели – это задача, состоящая в том, чтобы определить выполнимость для процесса, который задается системой переходов (структурой Крипке), свойства, выраженного формулой темпоральной логики. Сформулируем задачу проверки модели.

Задача проверки модели

Дано: конечная система переходов (структура Крипке), начальное состояние s_0 этой системы и формула φ темпоральной логики.

Вопрос: удовлетворяет ли состояние s_0 системы переходов формуле темпоральной логики φ?

В идеальном случае верификация проводится полностью автоматически. Однако на практике она часто требует содействия человека. Одной из сторон деятельности человека является анализ результатов верификации. Если результаты проверки отрицательные, то пользователю нередко предоставляют трассу, содержащую ошибку. Она строится в качестве контрпримера для проверяемого свойства и может помочь проектировщику проследить, где возникает ошибка. В этом случае анализ ошибочной трассы может повлечь за собой модификацию системы и повторное применение алгоритма проверки на модели.

Ошибочная трасса может появиться и вследствие некорректного моделирования системы, а также в результате использования неправильной спецификации (в таких случаях говорят о ложном опровержении). Трасса, демонстрирующая ошибку, может оказаться весьма полезной для выявления этих двух проблем. Наконец, не исключена возможность, что решение задачи верификации не будет завершено из-за большого размера модели, не позволяющего разместить ее в памяти компьютера. В таком случае, возможно, будет необходимо провести верификацию повторно, изменив параметры инструментального средства, реализующего проверку на модели, или упростив саму модель (скажем, за счет более сильной абстракции).

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

- S – конечное множество состояний;
- $s_0 \in S$ – начальное состояние;
- $\to \subseteq S \times S$ – тотальное отношение переходов (тотальность означает, что для каждого состояния $s \in S$ должно существовать состояние $s' \in S$, для которого имеет место $(s, s') \in \to$, т.е. $s \to s$);
- $L : S \to 2^P$ – 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 $s_i \to s_{i+1}$.

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.

Одна из широко используемых темпоральных логик в формальной верификации – логика CTL (branching-time logic или computation tree logic).

Формулы логики CTL для структуры Крипке $\mathbf{S}$ строятся по следующей грамматике:

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid$$

$$AG\varphi \mid EG\varphi \mid A(\varphi \, U \, \varphi) \mid E(\varphi \, U \, \varphi) \mid A(\varphi \, \widetilde{U} \, \varphi) \mid E(\varphi \, \widetilde{U} \, \varphi),$$

где $p \in P$ – элементарное высказывание, определенное над множеством состояний S.

Отношение выполнимости $\models$ для состояния s структуры Крипке $\mathbf{S} = (S, s_0, \rightarrow, L)$ и формулы φ логики CTL индуктивно определяется следующим образом:

- $s \vDash true$ и $s \nvDash false$;
- $s \vDash p$ для $p \in P \Leftrightarrow p \in L(s)$;
- $s \vDash \neg\varphi \Leftrightarrow s \nvDash \varphi$;
- $s \vDash \varphi \wedge \psi \Leftrightarrow s \vDash \varphi$ и $s \vDash \psi$;
- $s \vDash \varphi \vee \psi \Leftrightarrow s \vDash \varphi$ или $s \vDash \psi$;
- $s \vDash EX\varphi \Leftrightarrow \exists s' s \rightarrow s'$ и $s \vDash \varphi$ – существует такое следующее состояние структуры Крипке $\mathbf{S}$, в котором будет выполняться формула φ;
- $s \vDash AX\varphi \Leftrightarrow \forall s' s \rightarrow s'$ следует $s' \vDash \varphi$ – во всех следующих состояниях структуры $\mathbf{S}$ должна выполняться формула φ;
- $s \vDash E(\varphi\, U\, \psi) \Leftrightarrow$ для некоторого пути π, выходящего из состояния $s = \pi(0)$, существует такое $j \geq 0$, что $\pi(j) \vDash \psi$, и при этом для всех i, $0 \leq i < j$, $\pi(i) \vDash \varphi$;
- $s \vDash A(\varphi\, U\, \psi) \Leftrightarrow$ для каждого пути π, выходящего из состояния $s = \pi(0)$, существует такое $j \geq 0$, что $\pi(j) \vDash \psi$, и при этом для всех i, $0 \leq i < j$, $\pi(i) \vDash \varphi$;
- $s \vDash EF\varphi \Leftrightarrow E(true\, U\, \varphi)$ – из s в структуре Крипке $\mathbf{S}$ существует путь, проходящий через состояние, в котором выполняется φ;
- $s \vDash AF\varphi \Leftrightarrow A(true\, U\, \varphi)$ – любой путь из состояния s в структуре Крипке $\mathbf{S}$ обязательно проходит через состояние, в котором выполняется φ;
- $s \vDash E(\varphi\, \widetilde{U}\, \psi) \Leftrightarrow$ для некоторого пути π, выходящего из состояния $s = \pi(0)$, для любого $j \geq 1$ такого, что $\pi(j) \nvDash \psi$, существует i, $0 \leq i < j$, что $\pi(i) \vDash \varphi$ (для некоторого пути формула ψ истинна и должна оставаться истинной до тех пор, пока φ не станет истинной);
- $s \vDash A(\varphi\, \widetilde{U}\, \psi) \Leftrightarrow$ для каждого пути π, выходящего из состояния $s = \pi(0)$, для любого $j \geq 1$ такого, что $\pi(j) \nvDash \psi$, существует i, $0 \leq i < j$, что $\pi(i) \vDash \varphi$ (для каждого пути формула ψ истинна и должна оставаться истинной до тех пор, пока φ не станет истинной);
- $s \vDash EG\varphi \Leftrightarrow E(false\, \widetilde{U}\, \varphi)$ – существует путь из состояния s в структуре Крипке $\mathbf{S}$, на протяжении которого (в каждом состоянии пути) выполняется φ;
- $s \vDash AG\varphi \Leftrightarrow A(false\, \widetilde{U}\, \varphi)$ – на протяжении любого пути (в каждом состоянии каждого пути) из состояния s в структуре Крипке $\mathbf{S}$ выполняется φ;

Кроме введенных логических связок $\wedge$ и $\vee$ традиционно используют связки $\rightarrow$ и $\leftrightarrow$:

- $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$;
- $\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \equiv (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$.

Для темпоральных операторов справедливы (вытекают непосредственно из определений) следующие соотношения:

- $AX\varphi \equiv \neg EX\neg\varphi$;
- $AF\varphi \equiv \neg EG\neg\varphi$;
- $AG\varphi \equiv \neg EF\neg\varphi$;
- $A(\varphi\, \widetilde{U}\, \psi) \equiv \neg E(\neg\varphi\, U\, \neg\psi)$;
- $E(\varphi\, \widetilde{U}\, \psi) \equiv \neg A(\neg\varphi\, U\, \neg\psi)$.

Применяя последние два соотношения, можно отказаться от использования в темпоральных формулах операторов $E\widetilde{U}$ и $A\widetilde{U}$, поскольку их определения довольно сложны, что может привести к трудностям при словесной интерпретации (понимании) формул.

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

Темпоральное свойство, заданное формулой φ темпоральной логики CTL, считается истинным для структуры Крипке $\mathbf{S} = (S, s_0, \rightarrow, L)$ тогда и только тогда, когда выполняется $s_0 \vDash \varphi$.

3. Верификатор CPN/Tools

CPN/Tools [9, 10, 11] – это инструмент, который позволяет визуально редактировать, выполнять и анализировать раскрашенные сети Петри (CPN – Colored Petri Nets). Он разработан в университете Аархуса, Дания, на основе предложенных Куртом Йенсеном в 80-х гг. раскрашенных сетей Петри. Но в CPN/Tools используются не классические раскрашенные сети Петри – добавлено время и используется встроенный язык программирования CPN ML [10] (на основе Standard ML). CPN/Tools используется при разработке и верификации коммуникационных протоколов и распределенных систем.

Перейдем к описанию структуры Крипке, порождаемой верификатором CPN/Tools по раскрашенной сети Петри.

Эта структура Крипке представляет собой граф достижимых разметок – State spaces [11] (occurrence / reachability graphs), узлами которого являются разметки, а дуги – означивания переходов раскрашенной сети Петри. При этом из одного узла в другой может быть несколько дуг (с разными означиваниями), а на дуге только одно означивание перехода (параллельное срабатывание нескольких переходов не разрешается).

Для этой структуры Крипке используется темпоральная логика ASK–CTL [11], которая отличается от CTL тем, что формулы могут быть двух видов: по разметкам и по означиваниям. При этом добавляется еще один темпоральный оператор перехода от разметки к следующему означиванию, от означивания к следующей разметке.

В процессе верификации модели в виде раскрашенной сети Петри средством CPN/Tools сначала по сети строится граф достижимых разметок, по которому можно получить отчет, содержащий информацию о количестве узлов и дуг в графе, тупиковых разметках, живых и мертвых переходах сети. Затем записанную на языке CPN ML [10] темпоральную формулу можно проверить методом model checking (алгоритмы приведены в [11]). Также CPN/Tools позволяет проводить анализ графа достижимых разметок, не используя темпоральных формул – можно писать свои запросы на языке CPN ML.

4. 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) [7, 8].

Рассмотрим для произвольной иерархической программы, построенной на принципах автоматного программирования, ее автоматную модель A, представляющую собой набор взаимодействующих автоматов $(A_0, A_1, \ldots, A_n)$.

Для этой автоматной модели A построим структуру Крипке, которая, с одной стороны, описывает все возможные состояния системы A, а с другой стороны, задает семантику элементарных высказываний, истинных в этих состояниях. Для этого необходимо построить такую модель в виде раскрашенной сети Петри, которая отражала бы поведение автоматной программы.

Рассмотрим произвольный автомат A_k из системы автоматов A. Выделим в 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 (одновременно обратится к объекту управления с запросом значений входных переменных x_k и определит состояния вложенных автоматов y_k для проверки условий переходов) или произведет некоторое выходное воздействие на объект управления или вложенный автомат z_k.

Продемонстрируем идею выделения промежуточных состояний для перехода произвольного автомата A_k (рис. 1).

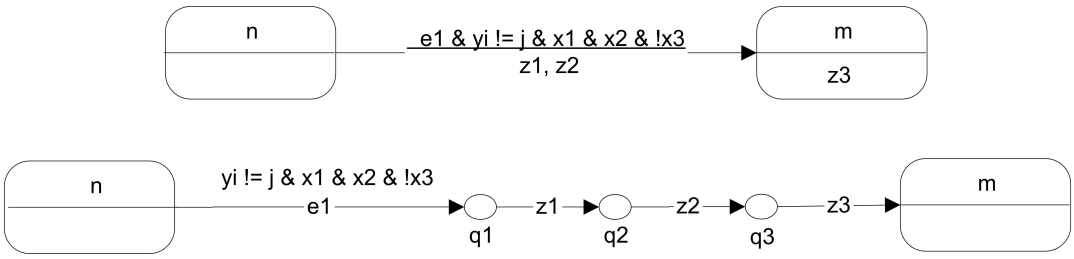

Рис. 1. Выделение промежуточных состояний для автоматного перехода

Внутренний переход с пометкой $e1$ может произойти при наступлении события $e1$, если выполняется условие $[yi \mathrel{!}= j \;\&\; x1 \;\&\; x2 \;\&\; !x3]$. 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 идет переход, соответствующий входному событию, затем переходы, помеченные выходными воздействиями. Таким образом, можно разделить последовательность внутренних состояний на две части, которые идут друг за другом строго в соответствии с указанным порядком. В каждой части метки внутренних переходов помечены в порядке следования соответствующих элементов в выражении на дуге исходного перехода.

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

Каждому автомату A_k из набора A сопоставим переменные $y_k (0 \le k \le n)$. Переменная y_k хранит текущее основное состояние автомата A_k. А для всей системы автоматов A в целом введем стек необработанных действий $GenActList$ системы автоматов A (список еще не выполненных выходных воздействий и вызовов автоматов A_k с конкретными событиями в порядке их поступления).

Тогда состоянием s структуры Крипке S_A автоматной модели A является вектор значений переменных

$$(y_0, \ldots, y_n, GenActList).$$

Вершину стека $GenActList$ обозначим через переменную gen_act. Возможны три варианта ее значений.

1) В стеке нет действий ($gen\_act = []$) – это означает, что сгенерируется одно из возможных событий для основного автомата A_0 (вершина стека получит значение $gen\_act = (A_0, e_k)$).

2) В вершине стека находится событие e_k для автомата A_i ($gen\_act = (A_i, e_k)$) - автомат A_i прореагирует или проигнорирует событие e_k.

3) В вершине стека находится выходное воздействие z_m ($gen\_act = z_m$) - оно будет выполнено (z_m будет удалено из вершины стека).

Переход автомата A_k из состояния $y_k = i$ в состояние $y_k = j$ (или петля в случае $i = j$) может сработать, если ему передано управление - в вершине стека необработанных действий находится вызов A_k с нужным событием e_n, если автомат A_k находится в состоянии $y_k = i$ и истинен предикат от входных переменных и состояний вложенных автоматов. Если этот переход сработал, то текущим состоянием автомата A_k становится $y_k = j$, из вершины стека забирается вызов $A_k(e_n)$, при этом добавляются вызовы выходных воздействий (на дуге и при входе в состояние) и вызовы вложенных автоматов с событием e_n. Если ни

один из переходов по каким-либо причинам сработать не может, то срабатывает специально введенный для каждого состояния автомата A_k пустой переход, действия которого заключаются в удалении из вершины стека вызова $A_k(e_n)$ и добавлении в вершину стека вызовов вложенных автоматов (если такие в этом состоянии есть) с событием e_n.

В начальном состоянии s_0 структуры Крипке S все переменные y_k содержат начальные состояния соответствующих автоматов A_k, а стек необработанных действий пуст ($gen\_act = []$).

Для моделирования работы системы автоматов A раскрашенной сетью Петри необходимо, чтобы построенная сеть обладала описанным выше поведением и порождала описанную выше структуру Крипке. При этом для любого автомата A_k каждому состоянию и переходу A_k должны соответствовать позиции и переходы раскрашенной сети Петри.

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

1) В каждом состоянии автоматной модели существует возможность отслеживать, какое действие произойдет при переходе из текущего состояния в следующее с помощью вершины стека – переменной gen_act.

2) Имеется возможность определения типа еще не произошедшего действия, т.е. произойдет ли генерация входного события, реакция одного из автоматов системы A на поступившее ему событие или выходное воздействие, посредством просмотра вершины стека.

3) Также через стек в каждом состоянии автоматной модели определяется активный в следующий момент времени автомат через выражение $gen\_act = (A_i, e_k)$. Если в состоянии системы A выполняется это условие, то это означает, что активным будет автомат A_i.

4) Для каждого автомата A_i в текущем состоянии системы A можно узнать через выражение $y_i = j$, пребывает ли автомат A_i в основном состоянии с номером j. Но при наличии действий в стеке к моменту их выполнения состояние автомата A_i может измениться, поэтому при записи выражения $y_i = j$ будем предполагать, что стек действий пуст ($gen\_act = []$).

5. Темпоральные свойства автоматных моделей

Рассмотрим несколько примеров темпоральных свойств, которые являются общими для всех конкретных иерархических автоматных моделей (для любой иерархической системы взаимодействующих автоматов).

«Тупиковое состояние». Свойство, описывающее возможность попадания автоматной программы в тупиковое состояние, из которого нельзя выйти, применимо к любой автоматной программе (к модели A любой автоматной программы рассматриваемого типа). Для раскрашенной сети Петри это свойство означает наличие тупиковых разметок, которые появляются только при наличии конечного состояния в автоматной программе. С помощью безусловного перехода из конечного состояния в начальное можно устранить эту проблему. Но будем предполагать, что в автоматной программе отсутствуют конечные состояния – она работает по бесконечному циклу.

«Живость переходов». Для любого перехода автоматной программы (дуги или петли автомата A_k) из любого состояния должен существовать путь в то состояние системы автоматов A, в котором возможно выполнение этого перехода. Это свойство необходимо в предположении, что автоматная программа работает по бесконечному циклу. Но это не означает, что любой переход обязательно сработает на протяжении любого пути в структуре Крипке S_A (существует множество путей из начального состояния, на всем протяжении которых этот переход не сработает). Если хотя бы один из переходов системы A не является живым, то выполнение системы вырождается в бесконечный и малый цикл. Если же существует мертвый переход системы A, то это сигнал того, что при разработке автоматной программы допущена серьезная ошибка. Т.к. дугам и петлям автомата A_k соответствуют переходы в раскрашенной сети Петри, то для дуги или петли выполнено то же свойство живости, что у соответствующего перехода. Также если для пустого перехода автомата A_k, специально введенного для игнорирования поступивших событий для автомата A_k, оказывается, что есть ситуации, когда он срабатывает, то это может быть сигналом того, что при разработке автоматной программы не учтены некоторые пути ее поведения.

«Забытый автомат». Начиная с некоторого момента времени (с некоторого состояния пути структуры Крипке S_A) вложенному автомату A_k никогда не будет передано управление. Таким образом, автомат

A_k с некоторого момента времени навсегда останется в одном из своих основных состояний. Это свойство выражается так:

$$EFAGgen\_act\ != (A_k, e).$$

Далее рассмотрим ряд примеров темпоральных свойств, относящихся к конкретной автоматной модели системы управления кофеваркой [6].

6. Спецификация темпоральных свойств автоматной модели системы управления кофеваркой

Удобство и целесообразность применения введенной структуры Крипке и темпоральной логики CTL для спецификации и верификации свойств автоматных моделей продемонстрируем на примере автоматной модели системы управления кофеваркой.

Рассмотрим и специфицируем ряд свойств для системы управления кофеваркой.

«Определение неисправности». Если произошла поломка нагревателя или одного из клапанов, то кофеварка (основной автомат A_0) обязательно перейдет в состояние «5. Неисправность».

1) Для каждого пути (или исполнения) структуры Крипке рассматриваемой системы взаимодействующих автоматов должно быть выполнено следующее требование. Если система перейдет в состояние $(y_{31} = 4 \mid y_{32} = 4 \mid y_2 = 4)\ \&\ y_0 = 2$, то следующим отличным от состояния $y_0 = 2$ («2. Варит кофе») у автомата A_0 обязательно - рано или поздно, будет состояние $y_0 = 5$ (не существует путь в структуре Крипке, который ведет в другое состояние $(y_0\ != 2\ \&\ y_0\ != 5)$).

2) $AG((y_{31} = 4 \mid y_{32} = 4 \mid y_2 = 4)\ \&\ y_0 = 2 \rightarrow !E(y_0 = 2\ U\ (y_0\ != 2\ \&\ y_0\ != 5)))$.

«Индикация неисправности». Если основной автомат управления кофеваркой A_0 перешел в состояние «5. Неисправность», на дисплее обязательно высвечивается сообщение о типе неисправности (не допускается ситуация, при которой неисправность произошла, но пользователя кофеварка не информирует о типе неисправности).

1) Это свойство целесообразно переписать в негативной форме следующим образом. Существует путь в структуре Крипке, который ведет в состояние $y_0 = 5$, и на протяжении этого пути не будет выводиться на дисплей ни одно из сообщений о поломке клапана или нагревателя – не произойдет ни одного из действий $gen\_act = z_{35}$ или $gen\_act = z_{25}$.

2) $!E((gen\_act\ != z_{35}\ \&\ gen\_act\ != z_{25})\ U\ y_0 = 5)$.

«Сброс дисплея после неисправности». Если автомат A_0 после неисправности находится в состоянии «0. Готов к работе», то на дисплее кофеварки (включая дисплеи бойлера, нагревателя и клапанов) никакие сообщения о поломке не отображаются.

1) Если на дисплей кофеварки (включая дисплеи бойлера, нагревателя и клапанов) было выдано сообщение о неисправности, то до того как автомат A_0 перейдет в начальное состояние $y_0 = 0$, это сообщение должно быть сброшено. В данном случае также целесообразно записать это свойство в негативной форме, ожидая, что указанное условие в некотором пути структуры Крипке нарушается.

2) $!EF((gen\_act = z_{35}\ \&\ E(gen\_act\ != z_{36}\ U\ y_0 = 0))\ \mid$
$(gen\_act = z_{25}\ \&\ E(gen\_act\ != z_{26}\ U\ y_0 = 0))\ \mid$
$(gen\_act = z_{08}\ \&\ E(gen\_act\ != z_{03}\ U\ y_0 = 0)))$.

«Начальное состояние». Если автомат A_0 перешел в начальное состояние «0. Готов к работе», следовательно, все вложенные автоматы уже находятся в своих начальных состояниях.

1) Для всех путей структуры Крипке автоматной модели, если автомат A_0 попал в состояние $y_0 = 0$, то все вложенные автоматы находятся в начальных состояниях $y_1 = 0\ \&\ y_2 = 0\ \&\ y_{31} = 0\ \&\ y_{32} = 0$.

2) $AG(y_0 = 0 \rightarrow y_1 = 0\ \&\ y_2 = 0\ \&\ y_{31} = 0\ \&\ y_{32} = 0)$.

7. Практическая реализация

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).

Однако проблема состоит в том, что каждый верификатор имеет свой формализм для задания модели и свой способ порождения структуры Крипке для этой модели. Более того, верификаторы имеют и свою модификацию (реализацию) темпоральной логики, которая (в ряде случаев) может оказаться менее выразительной, чем, например, рассмотренная темпоральная логика CTL.

Возникает серьёзная задача адекватного задания структуры Крипке автоматной модели средствами уже существующих верификаторов. Под адекватностью понимается порождение верификатором такой структуры Крипке для заданной формальной модели, которая не допускает потерю каких-либо свойств исходной автоматной модели, а также исключает появление несуществующих свойств. Должно быть гарантировано, что после проверки свойств для модели, заданной в рамках программы-верификатора, результат этой проверки будет однозначно применим и к исходным свойствам исходной автоматной модели. Кроме того, при выборе верификатора необходимо руководствоваться и выразительной способностью реализованной в нём темпоральной логики, чтобы иметь возможность выражать описанные выше типы свойств автоматных моделей.

Таким образом, после выбора адекватного и выразительного верификатора задача практической реализации автоматической проверки свойств автоматных программ может быть представлена в виде, показанном на рис. 2.

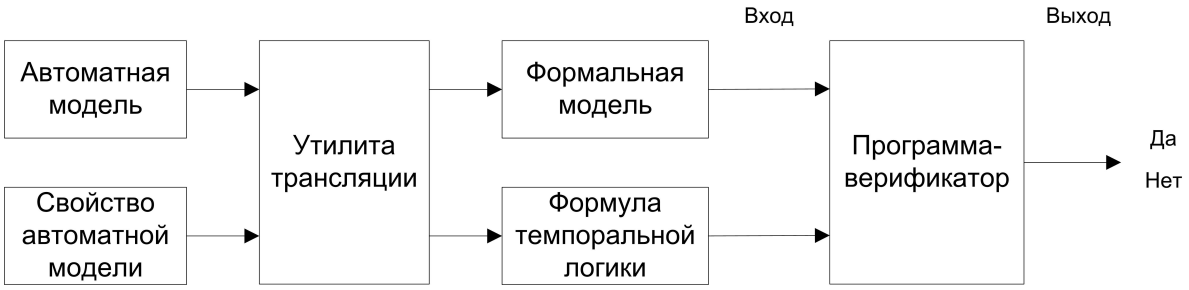

Рис. 2. Применение метода проверки модели для верификации автоматных программ

На рис. 2 важной деталью является утилита адекватной трансляции автоматной модели и её свойства в такие формальную модель и спецификацию, которые допускаются интерфейсом программы-верификатора. В соответствии с требованиями утилита трансляции должна всегда обеспечивать корректное однозначное соответствие между результатом проверки свойства верификатором и истинностью этого свойства автоматной модели.

Реализованная утилита транслирует автоматную модель из unimod-файла формата XML в раскрашенную сеть Петри в cpn-файле формата XML.

Далее приводятся результаты экспериментов, полученные при проверке модели системы управления кофеваркой.

8. Результаты экспериментов

Автоматная модель системы управления кофеваркой [6] была реализована с помощью инструментального средства UniMod [12, 13], транслирована в раскрашенную сеть Петри. Обнаруженные в модели кофеварки ошибки были выявлены путем анализа свойств живости переходов для системы автоматов этой модели.

1) Если основной автомат A_0 находится в состоянии «2. Варит кофе», а автомат A_1 находится в состоянии «2. Готов к варке следующей порции», и происходит событие e_{02} – «нажата кнопка «С»», то основной автомат перейдет в состояние «4. Прервано пользователем» и будет бесконечно долго отсылать своему вложенному автомату A_1 событие e_{12}, а автомат A_1 будет бесконечно долго игнорировать это событие, т.к. в состоянии «2. Готов к варке следующей порции» отсутствует переход в какое-либо состояние по событию e_{12}.

Исправление.

Добавить переход для автомата A_1 из состояния «2. Готов к варке следующей порции» по событию e_{12} в состояние «0. Готов к работе».

2) Автомат A_1 из состояния «1. Прогревает нагреватель» по событию e_12 всегда переходит в состояние «0. Готов к работе» и отсылает своему вложенному автомату A_2 событие e_{22}, по которому A_2 переходит в состояние «0. Готов к работе», в случае если он не находится в состоянии «4. Неисправность», иначе автомат A_2 остается в состоянии «4. Неисправность», автомат A_1 переходит в состояние «0. Готов к работе», при этом основной автомат A_0 переходит в состояние «4. Прервано пользователем», тем самым нарушается свойство «Определение неисправности», а затем основной автомат при поступлении события e_{01} перейдет в состояние «0. Готов к работе», а состояния других автоматов не изменятся, тем самым нарушается свойство «Начальное состояние».

   Исправление.

   Добавить условие y_2 != 4 для перехода в автомате A_1 из состояния «1. Прогревает нагреватель» в состояние «0. Готов к работе», т.е. запись изменится с «$e_{12}/A_2(e_{22})$» на «e_{12} & y_2 != $4/A_2(e_{22})$», что порождает еще одну ошибку, аналогичную двум другим.

3) Основной автомат A_0 находится в состоянии «4. Прервано пользователем» и бесконечно долго посылает событие e_{12} своему вложенному автомату A_1, пытаясь перевести его в состояние «0. Готов к работе». Эта ситуация возможна в трех случаях:

   (a) автомат A_1 находится в состоянии «1. Прогревает нагреватель», а автомат A_2 находится в состоянии «4. Неисправность»,

   (b) автомат A_1 находится в состоянии «3. Набирает воду», а автомат A_{31} не находится в состоянии «0. Закрыт и готов к работе»,

   (c) автомат A_1 находится в состоянии «3. Набирает воду», а автомат A_{32} не находится в состоянии «0. Закрыт и готов к работе».

   Исправление.

   Необходимо в этом случае дать возможность пройти одному из автоматов A_{31} и A_{32} их полный цикл («1. Открывается» → «2. Открыт» → «3. Закрывается» → «0. Закрыт и готов к работе») и определить неисправность автомату A_1, в случае ее возникновения. Это можно сделать, добавив к петле в состоянии «4. Прервано пользователем» автомата A_0 вызов автомата A_1 с событием e_0. В случае попадания автомата A_1 в состояние «7. Неисправность» необходим переход в автомате A_0 из состояния «4. Прервано пользователем» в состояние «5. Неисправность» «e_0 & $y_1 = 7/z_{08}$», а петля на состоянии «4. Прервано пользователем» автомата A_0 примет вид «e_0 & y_1 != 0 & y_1 != $7/A_1(e_0), A_1(e_{12})$».

Автоматная модель системы управления кофеваркой с внесенными исправлениями представлена на рис. 3 и рис. 4. В автоматы A_2 и A_3 изменения не вносились.

Также необходимо внести исправления в выписанные ранее темпоральные свойства автоматной модели системы управления кофеваркой.

«Определение неисправности». Если произошла поломка нагревателя или одного из клапанов, то кофеварка (основной автомат A_0) обязательно перейдет в состояние «5. Неисправность».

1) Для каждого пути (или исполнения) структуры Крипке рассматриваемой системы взаимодействующих автоматов должно быть выполнено следующее требование. Если система перейдёт в состояние $(y_{31} = 4 \mid y_{32} = 4 \mid y_2 = 4)$ & $(y_0 = 2 \mid y_0 = 4)$, то следующим отличным от состояний $y_0 = 2$ («2. Варит кофе») и $y_0 = 4$ («4. Прервано пользователем») у автомата A_0 обязательно - рано или поздно, будет состояние $y_0 = 5$ (не существует путь в структуре Крипке, который ведёт в другое состояние (y_0 != 2 & y_0 != 4 & y_0 != 5)).

2) $AG((y_{31} = 4 \mid y_{32} = 4 \mid y_2 = 4)$ & $(y_0 = 2 \mid y_0 = 4) \rightarrow$
   $!E((y_0 = 2 \mid y_0 = 4)\ U\ (y_0$!= 2 & y_0 != 4 & y_0 != 5))).

Структура Крипке исправленной модели кофеварки содержит 847 состояний и 1170 переходов.

Необходимо отметить, что при использовании описанной структуры Крипке для автоматных моделей оказывается, что в темпоральных свойствах неприменимы связки AU и AF, т.к. нужное состояние структуры Крипке может оказаться для некоторых путей не достижимо. Например, для модели кофеварки

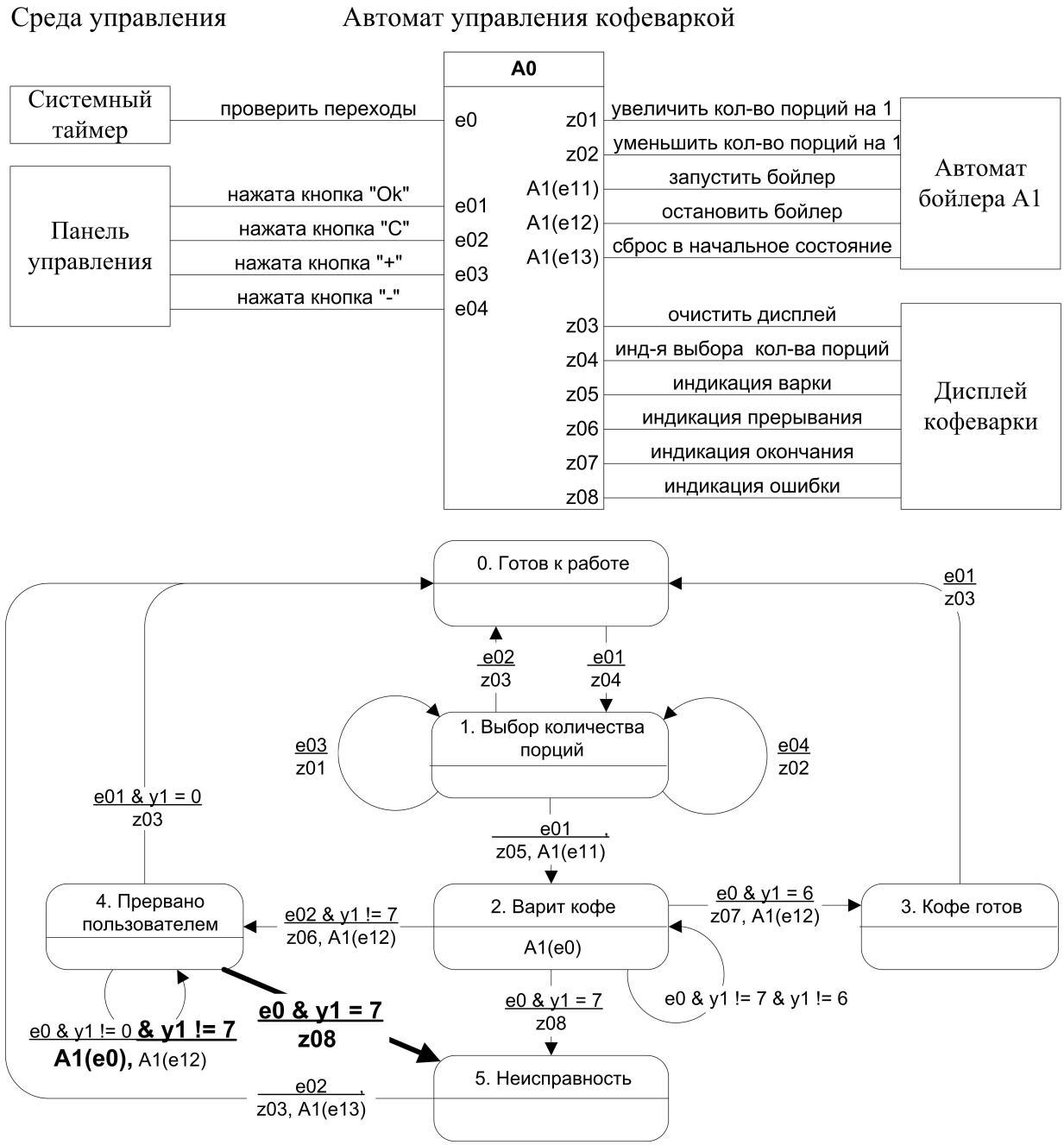

Рис. 3. Схема связей и граф переходов автомата управления кофеваркой A_0 с исправлениями

нельзя сказать, что для любого пути основной автомат A_0 когда-нибудь перейдет в состояние «2. Варит кофе». Это возможно, например, когда бесконечно долго пользователь выбирает количество порций или после выбора постоянно нажимает кнопку «C» (отмена). Но свойства можно записать и без использования этих связок – применяя соотношения для темпоральных операторов.

Также необходимо отметить и недостатки верификатора CPN/Tools. При проверке темпоральных формул в случае отрицательного ответа (свойство не выполняется), контрпример не строится, но его можно получить, вручную анализируя граф достижимых разметок. Ни в одном из темпоральных свойств не использовались значения входных переменных, т.к. в CPN/Tools накладываются значительные ограничения на запись высказываний для означиваний переходов. А при их использовании записанные формулы получаются очень сложными и огромными.

Существенным недостатком CPN/Tools является отсутствие способа выполнить необходимые действия из командной строки – приходится запускать графическую оболочку. Несмотря на продуманный графи-

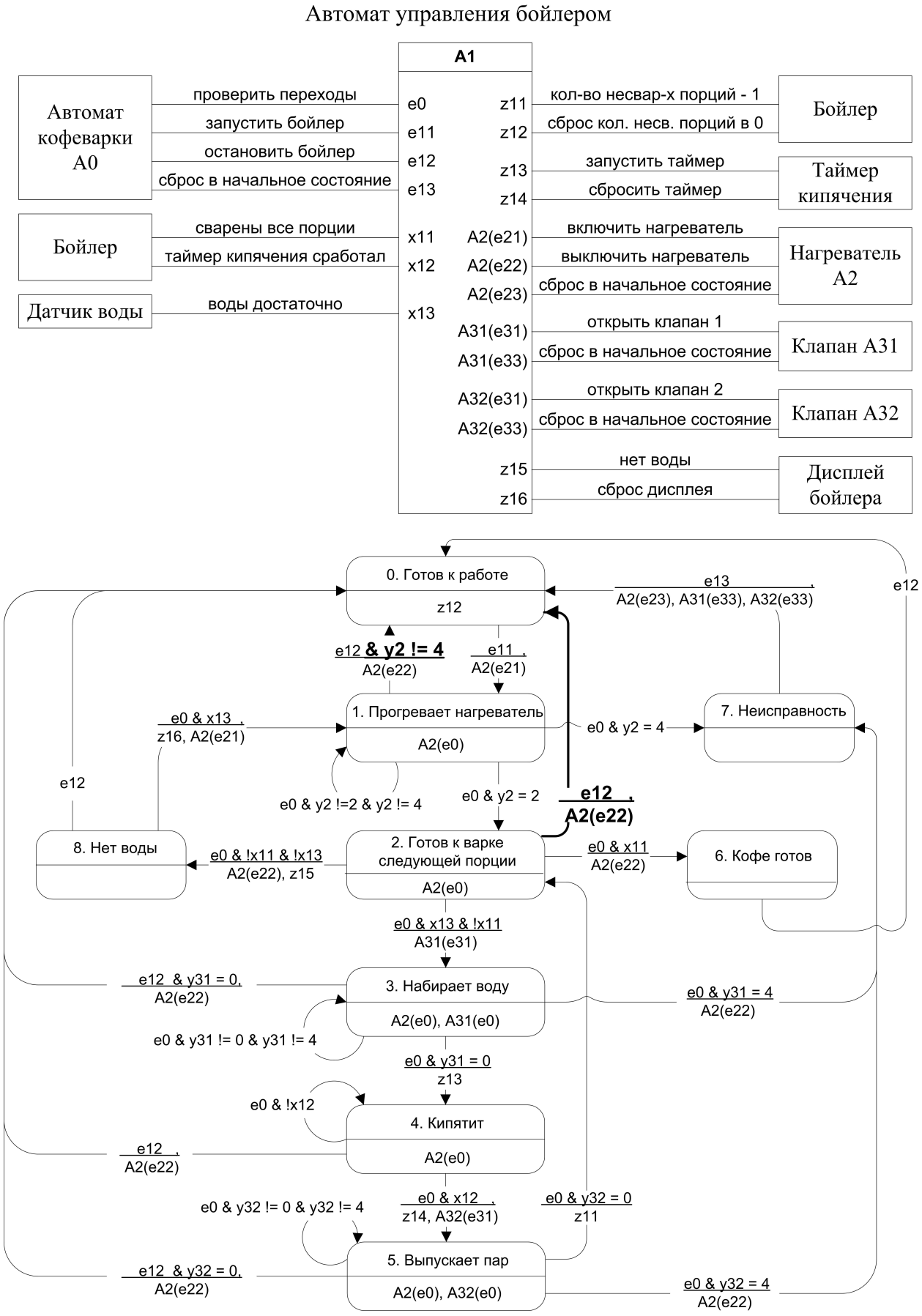

Рис. 4. Схема связей и граф переходов автомата управления бойлером A_1 с исправлениями

ческий интерфейс CPN/Tools, все действия (построить граф достижимых разметок, сохранить отчет, проверить темпоральную формулу) необходимо выбирать вручную. При этом для больших сетей необходимо при запуске долго ожидать завершения процесса компиляции загруженной сети и только после его завершения можно приступать к проверке модели. К тому же нельзя продолжить работу с прерванного места.

В дальнейшем предложенную технологию можно значительно расширить за счет моделирования самих автоматных программ, использования других средств написания автоматных программ, снятия приведенных выше ограничений при использовании CPN/Tools, задания стандарта записи свойства для автоматной модели и реализации механизма трансляции этого свойства в формулу темпоральной логики, применения других верификаторов (SPIN, SMV и др.). Используя технологию автоматного программирования и предложенную технологию, можно верифицировать реальные системы со значительно большим числом состояний, чем в рассмотренном примере.

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/works/app-aplu/1/

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/works/arew/1/

4. *Шалыто А.А., Туккель Н.И.* Программирование с явным выделением состояний // Мир ПК. — 2001. — №8. — С. 116–121; №9. — С. 132–138. — http://is.ifmo.ru/works/mirk/

5. *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/

6. *Кузьмин Е.В.* Иерархическая модель автоматных программ // Моделирование и анализ информационных систем. — Ярославль, 2006. — Т.13, 1. — С. 27–34.

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

8. *Кузьмин Е.В., Соколов В.А.* Вполне структурированные системы помеченных переходов. — М.: ФИЗМАТЛИТ, 2005. — 176 с.

9. CPN Tools home page: http://www.daimi.au.dk/CPNTools/

10. CPN Tools help / online version: http://wiki.daimi.au.dk/cpntools-help/

11. *Allan Cheng, Søren Christensen, Kjeld H. Mortensen.* Model Checking Colored Petri Nets Exploiting Strongly Connected Components / technical report, University of Aarhus, Denmark, 1996. — 14 c.

12. *Гуров В.С., Мазин М.А., Нарвский А.С., Шалыто А.А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. — 2004. — № 6. — С.12–17. — http://is.ifmo.ru/works/uml-switch-eclipse/

13. UniMod home page: http://unimod.sourceforge.net/

14. *Кессель С.В.* Разработка системы управления кофеваркой на основе автоматного подхода // СПбГУ ИТМО, 2003. — http://is.ifmo.ru/projects/coffee2/