MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 30, NO. 4, 2023
journal homepage: www.mais-journal.ru

THEORY OF SOFTWARE

LTL-specification for development and verification of control programs

M. V. Neyzov[1], E. V. Kuzmin[2]

DOI: 10.18255/1818-1015-2023-4-308-339

[1]Institute of Automation and Electrometry SB RAS, 1 Academician Koptyug ave., Novosibirsk 630090, Russia.
[2]P.G. Demidov Yaroslavl State University, 14 Sovetskaya str., Yaroslavl 150003, Russia.

MSC2020: 68N30
Research article
Full text in Russian

Received November 6, 2023
After revision November 20, 2023
Accepted November 22, 2023

This work continues the series of articles on development and verification of control programs based on the LTL-specification. The essence of the approach is to describe the behavior of programs using formulas of linear temporal logic LTL of a special form. The developed LTL-specification can be directly verified by using a model checking tool. Next, according to the LTL-specification, the program code in the imperative programming language is unambiguously built. The translation of the specification into the program is carried out using a template.

The novelty of the work consists in the proposal of two LTL-specifications of a new form — declarative and imperative, as well as in a more strict formal justification for this approach to program development and verification. A transition has been made to a more modern verification tool for finite and infinite systems — nuXmv. It is proposed to describe the behavior of control programs in a declarative style. For this purpose, a declarative LTL-specification is intended, which defines a labelled transition system as a formal model of program behavior. This method of describing behavior is quite expressive — the theorem on the Turing completeness of the declarative LTL-specification is proved. Next, to construct program code in an imperative language, the declarative LTL-specification is converted into an equivalent imperative LTL-specification. An equivalence theorem is proved, which guarantees that both specifications specify the same behavior. The imperative LTL-specification is translated into imperative program code according to the presented template. The declarative LTL-specification, which is subject to verification, and the control program built on it are guaranteed to specify the same behavior in the form of a corresponding transition system. Thus, during verification, a model is used that is adequate to the real behavior of the control program.

Keywords: control software; declarative LTL-specification; imperative LTL-specification; model checking; complete transition system; pseudo-complete transition system; Minsky counter machine; nuXmv verifier; PLC program; ST language

INFORMATION ABOUT THE AUTHORS

Maxim V. Neyzov
corresponding author
orcid.org/0009-0000-6893-6137. E-mail: neyzov.max@gmail.com
Researcher.

Egor V. Kuzmin
orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru
Head of the Chair of Theoretical Informatics, Doctor of Science.

Funding: State task IAaE SB RAS, project No. 122031600173-8; Yaroslavl State University (project VIP-016).

For citation: M. V. Neyzov and E. V. Kuzmin, “LTL-specification for development and verification of control programs”, *Modeling and analysis of information systems*, vol. 30, no. 4, pp. 308-339, 2023.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 30, № 4, 2023
сайт журнала: www.mais-journal.ru

THEORY OF SOFTWARE

LTL-спецификация для разработки и верификации управляющих программ

М. В. Нейзов[1], Е. В. Кузьмин[2]

DOI: 10.18255/1818-1015-2023-4-308-339

[1]Институт автоматики и электрометрии СО РАН, просп. Академика Коптюга, д. 1, г. Новосибирск, 630090 Россия.
[2]Ярославский государственный университет им. П.Г. Демидова, ул. Советская, д. 14, г. Ярославль, 150003 Россия.

УДК 004.424+519.683.8
Научная статья
Полный текст на русском языке

Получена 6 ноября 2023 г.
После доработки 20 ноября 2023 г.
Принята к публикации 22 ноября 2023 г.

Настоящая работа продолжает цикл статей по разработке и верификации управляющих программ на основе LTL-спецификации. Суть подхода заключается в описании поведения программ с помощью формул линейной темпоральной логики LTL специального вида. Полученная LTL-спецификация может быть непосредственно верифицирована с помощью инструмента проверки модели. Далее по LTL-спецификации однозначно строится код программы на императивном языке программирования. Перевод спецификации в программу осуществляется по шаблону.

Новизна работы состоит в предложении двух LTL-спецификаций нового вида — декларативной и императивной, а также в более строгом формальном обосновании данного подхода к разработке и верификации программ. Выполнен переход на более современный инструмент верификации конечных и бесконечных систем — nuXmv. Предлагается описывать поведение управляющих программ в декларативном стиле. Для этого предназначена декларативная LTL-спецификация, которая задаёт размеченную систему переходов как формальную модель поведения программы. Данный способ описания поведения является достаточно выразительным — доказана теорема о Тьюринг-полноте декларативной LTL-спецификации. Далее для построения кода программы на императивном языке декларативная LTL-спецификация преобразуется в эквивалентную императивную LTL-спецификацию. Доказана теорема об эквивалентности, которая гарантирует, что обе спецификации задают одно и то же поведение. Императивная LTL-спецификация транслируется в императивный код программы по представленному шаблону. Декларативная LTL-спецификация, которая подвергается верификации, и построенная по ней управляющая программа гарантированно задают одно и то же поведение в виде соответствующей системы переходов. Таким образом, при верификации используется модель, адекватная реальному поведению управляющей программы.

Ключевые слова: управляющее программное обеспечение; декларативная LTL-спецификация; императивная LTL-спецификация; полная система переходов; псевдополная система переходов; счётчиковая машина Минского; проверка модели; верификатор nuXmv; ПЛК-программа; язык ST

ИНФОРМАЦИЯ ОБ АВТОРАХ

Максим Вячеславович Нейзов
автор для корреспонденции
orcid.org/0009-0000-6893-6137. E-mail: neyzov.max@gmail.com
исследователь.

Егор Владимирович Кузьмин
orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru
заведующий кафедрой теоретической информатики, доктор физ.-мат. наук.

Финансирование: Госзадание ИАиЭ СО РАН, проект № 122031600173-8; ЯрГУ (проект VIP-016).

Для цитирования: M. V. Neyzov and E. V. Kuzmin, “LTL-specification for development and verification of control programs”, *Modeling and analysis of information systems*, vol. 30, no. 4, pp. 308-339, 2023.

Введение

При решении задач управления проектируемая система естественным образом разделяется на систему управления и объект управления, оказывающие влияние друг на друга в процессе работы. *Программное обеспечение* (ПО), реализующее процесс управления, является *управляющим* ПО (УПО) и представляет собой систему управления или её часть. Объект управления может иметь различную природу и назначение. Например, в *киберфизической системе* [1, 2] УПО формирует воздействие на физическую среду, взаимодействуя с ней через сенсоры и исполнительные устройства. Так программа *программируемого логического контроллера* (ПЛК) [3] управляет технологическим процессом на производстве. В этом случае УПО является частью аппаратно-программной системы управления. Другой пример — в системе управления базами данных УПО выполняет различные манипуляции с массивом данных, осуществляя управление их выборкой и хранением. Здесь объектом управления является массив данных, который имеет не физическую, а программную природу, как и система управления. При решении же вычислительных задач проектируемая система содержит только программные компоненты. Согласно [4] дискретный преобразователь информации может быть представлен в виде автоматной модели, состоящей из взаимодействующих компонентов — *операционного* и *управляющего* автоматов. Операционный автомат преобразует информацию, а управляющий автомат осуществляет управление действиями по её преобразованию, т. е. описывает процесс управления вычислениями. Выделение задачи управления при разработке ПО присуще направлению, которое получило название *программная кибернетика* [5]. В рамках этого подхода управляющий и операционный автоматы могут быть реализованы в виде программных компонентов на любом уровне абстракции, который определяется решаемой задачей [6]. Таким образом, УПО может разрабатываться как отдельная часть проектируемой системы при решении не только управляющих, но и вычислительных задач.

УПО представляет собой *реагирующую систему* [7, 8], которая постоянно циклически взаимодействует с объектом управления в темпе протекающих в нём процессов. Цикл работы реагирующей системы состоит в получении входных данных и их записи во входные переменные программы, работы программы, которая вычисляет и обновляет внутренние и выходные переменные, выдаче значений выходных переменных на объект управления.

При проектировании систем могут предъявляться высокие требования к их надёжности и безопасности [9], часть из которых неизбежно превращается в требования к *корректности* УПО. Под корректностью понимается соответствие заданному набору поведенческих свойств. Процедура проверки соответствия называется *верификацией* [10]. Высокие гарантии соответствия обеспечивают формальные методы верификации [11]. Одним из таких методов является *проверка модели* (model checking) [12—15], которая требует наличия модели, адекватной реальному поведению УПО.

Настоящая работа продолжает цикл статей по разработке и верификации управляющих программ ПЛК на основе LTL-спецификации [16—32]. Данный подход предлагает разрабатывать LTL-спецификацию поведения программы, которая может быть непосредственно верифицирована методом проверки модели. Более того, спецификация является *конструктивной*, то есть по ней однозначно может быть построена программа на некотором языке программирования (ST [16, 25], LD [28, 29], IL [30], CFC [19, 20]) стандарта МЭК 61131-3 [33], а также модель поведения на входном языке верификатора Cadence SMV (http://www.mcmil.net/smv.html). LTL-спецификация обладает достаточной *выразительностью* для описания любого вычислительного процесса [17, 18]. LTL-спецификации счётчиковых машин представлены в [21, 22]. Также с помощью LTL-спецификации предлагается описывать согласованное поведение датчиков [31, 32], необходимое для проверки ряда программных свойств.

Новизна. В настоящей работе предложена более компактная модификация LTL-спецификации. Дано строгое формальное обоснование её конструктивности и выразительности. Выполнен переход

на более современный инструмент верификации конечных и бесконечных систем — nuXmv (https://nuxmv.fbk.eu).

Содержание работы. В разделе 1 излагается концепция разработки управляющих программ — представлены архитектурные особенности проектируемого УПО. Раздел 2 содержит используемые модели поведения в виде размеченных систем переходов. Вводятся понятия *полной* и *псевдополной* систем переходов. В разделе 3 приводится синтаксис и семантика языка LTL. В разделе 4 вводятся *декларативная* и *императивная* LTL-спецификации. Декларативная предназначена для описания поведения программ, а императивная — для построения кода программ. Доказывается теорема об эквивалентности данных LTL-спецификаций. В разделе 5 проведена оценка выразительности декларативной LTL-спецификации в смысле Тьюринг-полноты. Представлена *счётчиковая машина Минского* как модель алгоритма, эквивалентная по своим вычислительным возможностям *машине Тьюринга*. Приведён пример счётчиковой машины возведения числа в квадрат. Доказывается теорема о полноте по Тьюрингу декларативной LTL-спецификации. В разделе 6 выполняется постановка задачи верификации. Накладывается ограничение на задачу с целью обеспечения её разрешимости. Раздел 7 содержит пример разработки и верификации с помощью инструмента nuXmv декларативной LTL-спецификация ПЛК-программы возведения числа в квадрат. В разделе 8 представлена схема построения ST-программы по императивной LTL-спецификации. Приведён код ST-программы возведения числа в квадрат. Далее делается заключение. Приложение демонстрирует процедуру построения LTL-спецификации, предложенную в теореме о Тьюринг-полноте.

1. Концепция разработки управляющих программ

Управляющая программа работает циклически. В каждом *цикле управления* значение переменной может либо измениться, либо остаться прежним. Мы сосредоточены только на изменениях значений переменных, так как в задачах управления именно они несут в себе ключевую смысловую нагрузку. При управлении всегда известно, почему состояние управляемого объекта нужно изменить, почему, например, то или иное устройство должно быть включено или выключено. Причины изменения значений переменных мы декларируем в виде LTL-спецификации. Также спецификация содержит информацию о том, каким образом происходят эти изменения.

В целом поведение программы — совокупное поведение всех её переменных $V = \{v_1, \ldots, v_n\}$. Данное множество содержит входные и внутренние/выходные переменные. Изначально поведение всех переменных из V не декларировано и является абсолютно недетерминированным. Далее декларированию подлежит поведение только внутренних/выходных переменных. Декларация накладывает некоторые ограничения на их поведение, после чего оно становится строго детерминированным. Оставшийся недетерминизм входных переменных необходим для того, чтобы сохранить всё многообразие в поведении детерминированной программы, так как именно от поведения входных переменных зависит поведение внутренних/выходных переменных. Чтобы не потерять ни один сценарий работы программы, необходимо подавать на её вход любые допустимые значения в любой последовательности, что реализуется абсолютным недетерминизмом поведения входов.

Для анализа и декларирования причин изменения значения переменной часто требуется знать её предыдущее значение. Например, для определения момента нажатия кнопки (переднего фронта сигнала) необходимо располагать информацией о том, была ли она нажата в предыдущем цикле работы. Для этой цели вводится набор вспомогательных переменных $\_V = \{\_v_1, \ldots, \_v_n\}$, которые хранят предыдущие значения соответствующих переменных из V. Поведение вспомогательных переменных не декларируется и всегда известно. При использовании переменных из $\_V$ можно считать, что знак лидирующего подчёркивания «_» является псевдооператором обращения к предыдущему значению соответствующей переменной из V. Сравнивая значения переменных v_i и $\_v_i$ (где $i = 1, \ldots, n$), можно судить о том, произошло ли изменение значения переменной v_i.

Далее декларацию в виде LTL-спецификации можно верифицировать и построить по ней программу. Построенная программа будет обладать двумя особенностями:

1. Значение каждой переменной изменяется не более одного раза за один цикл управления.
2. Значение каждой переменной изменяется только в одном месте программы в некотором простом операторном блоке.

Эти две особенности позволяют иметь прозрачное и наглядное представление о том, каким образом происходит изменение значения той или иной переменной при переходе программы из одного состояния в другое. Для каждой переменной устанавливается четкая зависимость нового её значения от прежних значений переменных, которые были получены при выполнении программы на предыдущем проходе рабочего цикла, и новых значений переменных, уже вычисленных при выполнении программы на текущем проходе рабочего цикла. Условие изменения значения переменной только в одном месте программы облегчает отладку, дает возможность простой оценки степени готовности и объема текста программы. Очевидно, что за один проход рабочего цикла значение любой переменной возрастает, убывает или остается без изменения по отношению к её значению, полученному на предыдущем проходе рабочего цикла. Если вообще не обращаться к переменной с целью присваивания какого-либо значения, то она сохранит своё прежнее значение.

2. Система переходов как модель поведения программы

Пусть программа имеет основные $v_1, \ldots, v_n$ и вспомогательные $\_v_1, \ldots, \_v_n$ переменные, которые принимают значения из соответствующих областей $D_1, \ldots, D_n$. Множество (в общем случае бесконечное) $S = (D_1 \times \ldots \times D_n)^2$ будет являться пространством всех возможных состояний программы. Множество $V = \{v_1, \ldots, v_n\}$ содержит входные и внутренние/выходные переменные. Вектор $(d_1, \ldots, d_n, \_d_1, \ldots, \_d_n) \in S$ значений переменных $(v_1, \ldots, v_n, \_v_1, \ldots, \_v_n)$ описывает конкретное состояние программы и содержит текущие значения входных переменных и вновь вычисленные на их основе значения внутренних/выходных переменных. В переменных $\_v_1, \ldots, \_v_n$ хранятся предыдущие значения переменных $v_1, \ldots, v_n$. Каждому циклу управления соответствует *переход* между состояниями. Если состояние не изменилось, то считается, что произошёл переход по петле в то же самое состояние.

Все возможные переходы формируют поведение программы. В качестве модели поведения программы примем *размеченную систему переходов* (labelled transition system) $LTS = \langle S, S_0, R, P, L \rangle$, где S — множество состояний (в общем случае бесконечное), $S_0 \subseteq S$ — множество начальных состояний, $R \subseteq S \times S$ — *тотальное* отношение переходов, $P = \{p_1, \ldots, p_m\}$ — множество атомарных утверждений относительно значений переменных $v_1, \ldots, v_n$ и $\_v_1, \ldots, \_v_n$, $L : S \to 2^P$ — функция разметки состояний атомарными утверждениями. Тотальность отношения переходов R означает, что из любого состояния $s \in S$ существует переход: $(\forall s \in S)\ (\exists s' \in S)\ (s, s') \in R$.

Граф системы переходов — граф, вершинами которого являются состояния из S, а дугами — переходы из R. *Путь* в системе переходов — бесконечная последовательность $\pi = s_0 s_1 s_2 \ldots$, где $(s_i, s_{i+1}) \in R$, $i \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Путь формирует бесконечный маршрут в графе системы переходов. Система переходов LTS задаёт множество путей Π_{LTS}, исходящих из начальных состояний:

$$\Pi_{LTS} = \{\, \pi \in S^\omega \mid \big(\pi(0) \in S_0\big) \wedge (\forall i \in \mathbb{N}_0)\ \big(\pi(i), \pi(i+1)\big) \in R \,\},$$

где S^ω — множество всех бесконечных слов из алфавита S, $\pi(i)$ — i-ое состояние пути π.

Моделью поведения программы с недетерминированным поведением всех её переменных $v_1, \ldots, v_n, \_v_1, \ldots, \_v_n$ будет *полная* система переходов $cLTS = \langle S, S_0, R_c, P, L \rangle$, где $S_0 = S$. В отличие от LTS система $cLTS$ имеет отношение переходов $R_c = S \times S$, которое задаёт *полный* граф переходов по состояниям. Действительно, при абсолютном недетерминизме всегда возможен переход из любого состояния в любое другое, включая само себя. Заметим, что программа с недетерминированным

поведением всех переменных (*абсолютно недетерминированная* программа) содержит в себе поведение любой другой программы с этим же набором переменных $v_1, \ldots, v_n, \_v_1, \ldots, \_v_n$. Здесь $S_0 = S$ для того, чтобы *cLTS* содержала любой путь, начинающийся с любого состояния.

Если на полную систему переходов *cLTS* наложить ограничение в виде ранее оговорённого поведения переменных $\_v_1, \ldots, \_v_n$, то получим *псевдополную* систему переходов $pLTS = \langle S, S_0, R'_c, P, L\rangle$, где $S_0 = S$, $R'_c = \{\big((a, \_a), (a', a)\big) \in R_c\}$. Псевдополная система переходов *pLTS* содержит в себе поведение любой другой программы, которая сохраняет предыдущие значения переменных $v_1, \ldots, v_n$ в переменных $\_v_1, \ldots, \_v_n$.

При декларировании поведения переменных $v_1, \ldots, v_n$ накладываются определённые ограничения на псевдополную систему переходов *pLTS*. В итоге, модель поведения программы *LTS* получается из псевдополной системы переходов *pLTS* путем устранения переходов, нарушающих заданную декларацию поведения.

3. Язык логики LTL

Линейная темпоральная логика LTL (Linear Temporal Logic), или темпоральная логика линейного времени (Linear-Time Temporal Logic), представляет собой расширение классической логики высказываний с помощью модальных операторов, позволяющих учитывать временной аспект в последовательностях событий [12—15]. Темпоральная логика LTL находит важное применение в области формальной верификации, где она используется для описания требований к аппаратным и программным системам [8, 34]. Мы будем использовать LTL для:

- описания требований к псевдополной системе переходов *pLTS*, чтобы сформировать из неё модель поведения программы *LTS*;
- описания требований к модели поведения программы *LTS*, чтобы провести их верификацию.

Формулы LTL имеют следующую грамматику при $P = \{p_1, \ldots, p_m\}$:

$$\varphi, \psi := \mathit{true} \mid \mathit{false} \mid p_1 \mid \ldots \mid p_m \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid \mathbf{X}\varphi \mid \psi\mathbf{U}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi.$$

Помимо классических булевых операторов в LTL-формуле могут присутствовать и темпоральные: $\mathbf{X}\varphi$ означает, что формула φ должна выполнятся в следующем состоянии, $\mathbf{F}\varphi$ требует, чтобы φ выполнялась в некотором будущем состоянии, $\mathbf{G}\varphi$ гарантирует, что в текущем и во всех будущих состояниях будет выполняться φ. Формула $\psi\mathbf{U}\varphi$ означает, что φ должна выполнятся в текущем или будущем состоянии, а во всех предшествующих состояниях должна выполняться ψ. Операторы **F** и **G** являются двойственными: $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$. При этом сам оператор **F** представляет собой специальный случай применения оператора **U**, а именно $\mathbf{F}\varphi = \mathit{true}\mathbf{U}\varphi$.

Индуктивно определим отношение выполнимости $\vDash$ формулы φ логики LTL для произвольного состояния s_i некоторого пути $\pi = s_0\, s_1\, s_2 \ldots$ системы переходов, где $i \in \mathbb{N}_0$, $p \in P$:

$$\begin{aligned}
&s_i \vDash \mathit{true}; && s_i \nvDash \mathit{false};\\
&s_i \vDash p && \Longleftrightarrow p \in L(s_i);\\
&s_i \vDash \neg\varphi && \Longleftrightarrow s_i \nvDash \varphi;\\
&s_i \vDash \varphi \wedge \psi && \Longleftrightarrow s_i \vDash \varphi \ \text{ и } \ s_i \vDash \psi;\\
&s_i \vDash \varphi \vee \psi && \Longleftrightarrow s_i \vDash \varphi \ \text{ или } \ s_i \vDash \psi;\\
&s_i \vDash \varphi \Rightarrow \psi && \Longleftrightarrow s_i \vDash \neg\varphi \ \text{ или } \ s_i \vDash \psi;\\
&s_i \vDash \mathbf{X}\varphi && \Longleftrightarrow s_{i+1} \vDash \varphi;\\
&s_i \vDash \psi\mathbf{U}\varphi && \Longleftrightarrow (\exists k \geq i)\ s_k \vDash \varphi \ \text{ и } \ (\forall j,\ i \leq j < k)\ s_j \vDash \psi;\\
&s_i \vDash \mathbf{F}\varphi && \Longleftrightarrow (\exists k \geq i)\ s_k \vDash \varphi;\\
&s_i \vDash \mathbf{G}\varphi && \Longleftrightarrow (\forall j \geq i)\ s_j \vDash \varphi.
\end{aligned}$$

Также определим семантику отношения $\vDash$ на путях и системах переходов:

$$\begin{aligned}
\pi \vDash \varphi &\iff \pi(0) \vDash \varphi;\\
\Pi \vDash \varphi &\iff (\forall \pi \in \Pi)\ \pi \vDash \varphi;\\
LTS, s \vDash \varphi &\iff (\forall \pi \in \Pi_{LTS})\ [\pi(0) = s] \Rightarrow [\pi \vDash \varphi];\\
LTS \vDash \varphi &\iff (\forall s \in S_0)\ LTS, s \vDash \varphi.
\end{aligned}$$

Формула φ выполняется на пути π, когда она выполняется в его начальном состоянии, а на множестве путей Π, когда она выполняется на всех его путях. Формула φ выполняется в состоянии $s \in S$ системы переходов LTS, когда φ выполняется для всех путей в LTS, начинающихся с состояния s. Формула φ выполняется на системе переходов LTS, когда φ выполняется для любого начального состояния $s \in S_0$. Обозначим $[\![\varphi]\!]_{LTS}$ множество всех путей в LTS, на которых истинна формула φ, т. е. $[\![\varphi]\!]_{LTS} = \{\, \pi \in \Pi_{LTS} \mid \pi \vDash \varphi \,\}$.

4. LTL-спецификация поведения программ

4.1. Декларативная и императивная LTL-спецификации

Для задания поведения программных переменных $V = \{v_1, \ldots, v_n\}$ используется *декларативная* LTL-спецификация. Для переменной $v \in V$ это пара формул специального вида (1), (2), которая описывает, каким образом происходит изменение значения переменной:

$$\mathbf{GX}\big(\neg(v = \_v) \Rightarrow cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big), \tag{1}$$

$$\mathbf{GX}\big((v = \_v) \Rightarrow \neg\,(cond_1 \vee \ldots \vee cond_k)\big). \tag{2}$$

Здесь $cond_i$ — условие (логическое выражение), выполнимость которого необходима для изменения значения переменной v в соответствии с выражением $expr_i$, $i = 1, \ldots, k$. Выражения $cond_i$ и $expr_i$ строятся над множеством переменных $V \cup \_V$, где $V = \{v_1, \ldots, v_n\}$ и $\_V = \{\_v_1, \ldots, \_v_n\}$, с использованием констант и применением логических и арифметических операторов, а также операторов сравнения. При этом предполагается, что при любых значениях переменных из $V \cup \_V$ для этих выражений выполняются 1) условие изменчивости: $cond_i \Rightarrow \neg\,(\_v = expr_i)$, $\forall i = 1, \ldots, k$, и 2) условие ортогональности: $cond_i \Rightarrow \neg cond_j$, $\forall\, i, j = 1, \ldots, k$ при $i \neq j$.

Первая LTL-формула (1) утверждает, что если значение переменной v изменилось (новое значение v отличается от старого $\_v$), то было верно одно из условий $cond_i$, а переменная v получила значение выражения $expr_i$. Вторая LTL-формула (2) имеет жёсткую конструкцию, составленную из элементов первой формулы (1). Она утверждает, что если значение переменной v не изменилось (осталось равным предыдущему значению $\_v$), то ни одно из условий $cond_i$ не является истинным. Приставка **GX** говорит о том, что утверждение в скобках действует всегда, начиная с первого цикла работы программы, т. е. не захватывает только начальное состояние $s \in S_0$.

Условие изменчивости предохраняет от противоречия в формуле (1), которое может возникнуть при равенстве значения выражения $expr_i$ прежнему значению переменой v, а условие ортогональности предназначено для соблюдения детерминизма в поведении этой переменной.

Требование *изменчивости* без учёта начального состояния программы $s_0 \in S$ может быть представлено в виде LTL-формулы следующим образом:

$$\mathbf{GX}\big(cond_1 \Rightarrow \neg\,(\_v = expr_1)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow \neg\,(\_v = expr_k)\big), \tag{3}$$

т. е. при истинном условии $cond_i$ выражение $expr_i$ должно возвращать значение, отличное от предыдущего значения переменной v.

Аналогично для условия *ортогональности* имеем набор LTL-формул ($\forall\, i, j = 1, \ldots, k$ при $i \neq j$):

$$\mathbf{GX}\big(cond_i \Rightarrow \neg cond_j\big), \tag{4}$$

т. е. истинным может быть не более одного условия $cond_i$.

Конъюнкция множества формул вида (1) и (2) для внутренних/выходных переменных из V при соблюдении условий (3) и (4) образует *декларативную* LTL-спецификацию поведения *неинициализированной* программы φ_{var}.

Для инициализации будем использовать LTL-формулу φ_0 специального вида без темпоральных операторов: $I(v_1, \ldots, v_n) \wedge (\_v_1 = v_1) \wedge \ldots \wedge (\_v_n = v_n)$, где $I(v_1, \ldots, v_n)$ — это предикат, задающий ограничения на основные переменные. Начальные значения вспомогательных переменных $\_v_1, \ldots, \_v_n$ всегда совпадают с начальными значениями соответствующих основных переменных.

Формулу вида $\varphi = \varphi_{var} \wedge \varphi_0$ будем называть *декларативной* LTL-спецификацией поведения некоторой программы. Эта формула на основе псевдополной системы переходов *pLTS* задаёт систему переходов *LTS*. Полученная система переходов *LTS* будет содержать те и только те пути, которые соответствуют возможным сценариям работы рассматриваемой программы.

Продемонстрируем идею спецификации на простом примере. Зададим поведение одной булевой переменной v так, чтобы её значение более не менялось после перехода от 0 к 1. Данное поведение изображено в виде графа (рис. 1). Вершинам графа соответствуют значения переменной v (состояния), а дугам — возможные переходы между состояниями. На графе видно, что запрещено сохранять значение 0 и изменять значение с 1 на 0 (соответствующие дуги перечёркнуты).

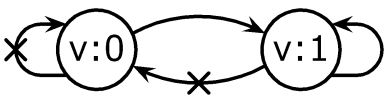

Fig. 1. Variable v behavior

Рис. 1. Поведение переменной v

Программа будет иметь две булевы переменные v и $\_v$. Потенциальное число состояний программы $2^2 = 4$, обозначим их $S = \{s_0, s_1, s_2, s_3\}$. Каждое состояние представляет собой уникальный вектор значений переменных v и $\_v$. В вершине графа значение переменной v указано над чертой, значение переменной $\_v$ — под чертой (рис. 2a).

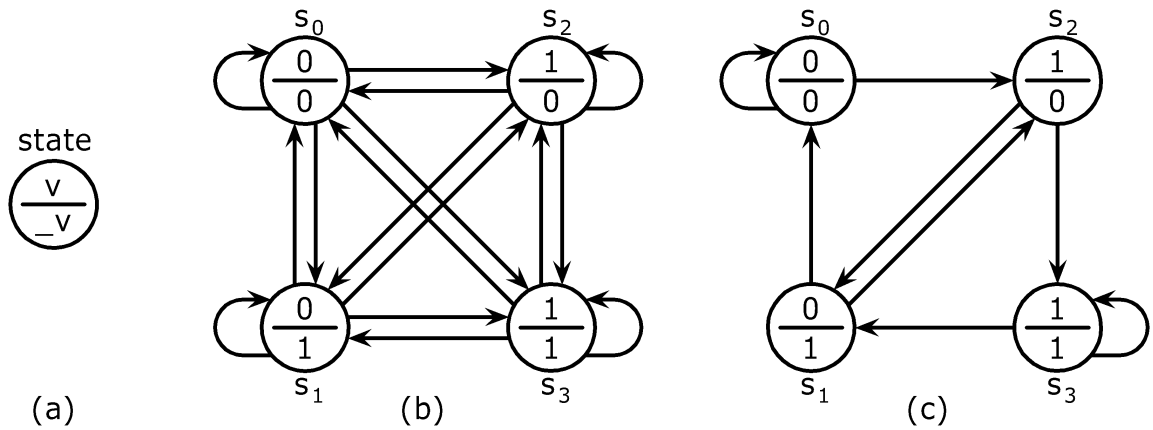

Fig. 2. State notation (a), complete transition system (b), pseudo-complete transition system (c)

Рис. 2. Обозначение состояния (a), полная система переходов (b), псевдополная система переходов (c)

Поведение абсолютно недетерминированной программы с переменными v и $\_v$ описывает полная система переходов *cLTS* (рис. 2b). Здесь возможен переход из любого состояния в любое другое,

включая само себя. Если переменная $\_v$ будет хранить предыдущее значение переменной v, то это накладывает некоторое ограничение на поведение программы. Любое возможное поведение такой программы описывает псевдополная система переходов *pLTS* (рис. 2c). Здесь отсутствуют переходы, которые нарушают условие равенства текущего значения переменной $\_v$ значению переменной v на предыдущем шаге (в предыдущем состоянии). Например, отсутствует переход из s_0 в s_1, так как в состоянии s_1 значение переменной $\_v = 1$, хотя в предыдущем состоянии s_0 значение переменной $v = 0$. То есть предыдущее значение переменной v не сохраняется в переменной $\_v$. И наоборот, присутствует переход из s_1 в s_0, так как предыдущее значение переменной v сохраняется в переменной $\_v$: $v = 0$ в состоянии s_1 и $\_v = 0$ в состоянии s_0.

Опишем поведение программы в виде декларативной LTL-спецификации, которая представляет собой в данном случае всего две формулы:

$$\varphi_1\colon\ \mathbf{GX}\big(\neg(v = \_v) \Rightarrow\ (\_v = 0) \wedge (v = \neg\_v)\big),$$
$$\varphi_2\colon\ \mathbf{GX}\big((v = \_v) \Rightarrow\ \neg\,(\_v = 0)\big).$$

Эти формулы задают именно те требования, которые схематично изображены на рис. 1:

1. Единственная причина изменения значения переменной v — это её нулевое значение (переход из 0 в 1 разрешён, а из 1 в 0 запрещён).
2. При наличии этой причины ($\_v = 0$) изменение значения переменной неизбежно (петля из 0 в 0 запрещена).

Формула φ_1 утверждает, что если изменилось значение переменной v, то её предыдущее значение было равно 0, и изменение заключалось в инверсии этого значения (формализация пункта 1). Формула φ_2 фиксирует, что если значение переменной v не изменилось, то её предыдущее значение не равно 0. Это эквивалентно утверждению, что если предыдущее значение переменной v равно 0, то значение переменной изменилось (формализация пункта 2).

Проследим, как каждая формула корректирует псевдополную систему переходов *pLTS* (рис. 2,c). Формула φ_1 нарушается в состоянии $s \in S$, если из него есть фрагмент пути не нулевой длины в состояние $s' \in S$, в котором истинна левая часть импликации и ложна правая. В нашем случае $s' = s_1$. Здесь v не равно $\_v$ (истинна левая часть импликации) и $\_v$ не равно 0 (ложна правая часть импликации). Поэтому любой фрагмент пути, приводящий в состояние s_1, является контрпримером, нарушающим истинность формулы φ_1. Примеры таких фрагментов: $s_0s_2s_1$, $s_2s_3s_1$. Таким образом, формула φ_1 нарушается на *pLTS* (рис. 2c), т. е. $pLTS \nvDash \varphi_1$. Чтобы этого не происходило, удалим все дуги, входящие в состояние s_1 (рис. 3a). Теперь контрпримеров, нарушающих формулу φ_1, нет.

Аналогично формула φ_2 нарушается в состоянии $s \in S$, если из него есть фрагмент пути не нулевой длины в состояние $s' \in S$, в котором истинна левая часть импликации и ложна правая. В данном случае $s' = s_0$. Здесь v равно $\_v$ (истинна левая часть импликации) и $\_v$ равно 0 (ложна правая часть импликации). Поэтому любой фрагмент пути, приводящий в состояние s_0, является контрпримером, нарушающим истинность формулы φ_2. Примеры таких фрагментов: s_0s_0, s_1s_0. Таким образом, формула φ_2 нарушается на *pLTS*, удовлетворяющей формуле φ_1 (рис. 3a). Чтобы этого не происходило, удалим все дуги, входящие в состояние s_0 (рис. 3b). Теперь контрпримеров, нарушающих формулы φ_1 или φ_2, нет. С помощью представленной LTL-спецификации поведения переменной v из псевдополной системы переходов *pLTS* были отобраны только те пути $[\![\varphi_1 \wedge \varphi_2]\!]_{pLTS}$, на которых выполняются формулы спецификации φ_1 и φ_2.

Зафиксируем одно начальное состояние программы: $S_0 = \{s_0\}$. Для этого добавим следующую LTL-формулу инициализации $\varphi_0 : (v = 0) \wedge (\_v = v)$, которая выполняется только в состоянии s_0 (на рис. 3c отмечено входной стрелкой). Из оставшихся состояний s_1, s_2, s_3 (рис. 3b) пути не могут начинаться, так как это приведёт к нарушению формулы φ_0. Состояния s_2, s_3 достижимы из начального состояния s_0, поэтому с их помощью могут образовываться пути. Состояние s_1 не является

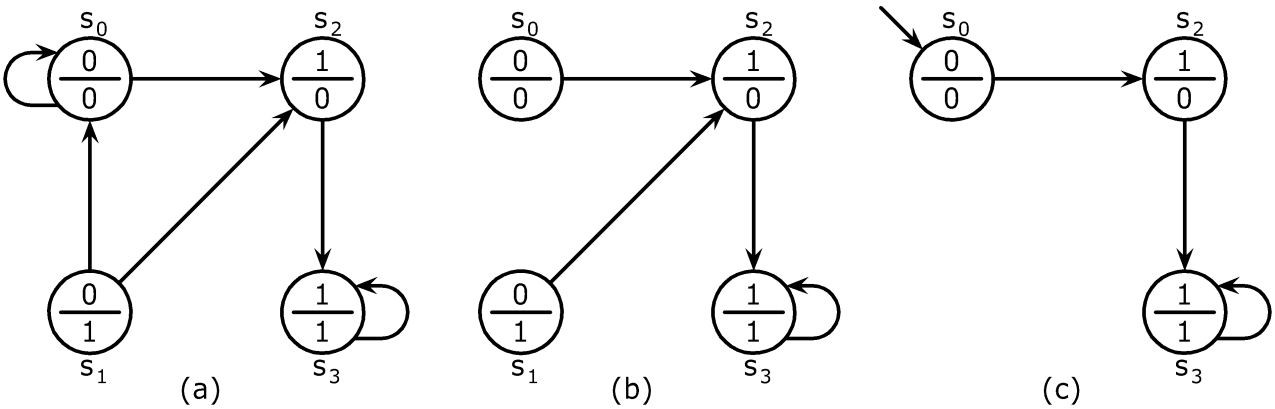

Fig. 3. Transition system: satisfies φ_1 (a), satisfies φ_1 and φ_2 (b), initialized program (c)

Рис. 3. Система переходов: удовлетворяющая φ_1 (a), удовлетворяющая φ_1 и φ_2 (b), инициализированной программы (c)

начальным и недостижимо из состояния s_0, оно не может формировать пути, удовлетворяющие формуле φ_0. Удалим недостижимое состояние s_1 и инцидентные ему дуги — получим систему переходов инициализированной программы (рис. 3c). Данная система переходов задаёт множество путей $[\![\varphi_0 \wedge \varphi_1 \wedge \varphi_2]\!]_{pLTS}$, на котором выполняются все формулы спецификации $\varphi_0, \varphi_1, \varphi_2$.

Проследим поведение переменной v. Изначально (в состоянии s_0) её значение равно 0. Далее значение обязательно меняется на 1 (неизбежно происходит переход в состояние s_2). После чего значение переменной v больше не изменяется (после перехода в состояние s_3, в котором $v = 1$, происходит зацикливание в нём). Таким образом, данная система переходов в точности задаёт поведение переменной v, представленное на рис. 1.

Задание системы переходов с помощью LTL-спецификации. На основе псевдополной системы переходов $pLTS = \langle S_{pLTS}, S_0, R_{pLTS}, P, L\rangle$ с помощью декларативной LTL-спецификации φ задаётся система переходов $LTS = \langle S, S'_0, R_\varphi, P, L'\rangle$, описывающая поведение программы. Отношение переходов R_φ имеет следующий вид:

$$R_\varphi = \{\, (s_1, s_2) \in R_{pLTS} \mid (\exists \pi \in \Pi_{pLTS})\ \pi = \sigma s_1 s_2 \ldots \vDash \varphi \,\},$$

где Π_{pLTS} — множество всех путей системы переходов $pLTS$, σ — конечный фрагмент пути, $|\sigma| \in \mathbb{N}_0$. Таким образом, из псевдополной системы переходов $pLTS$ отбираются только те переходы, которые образуют пути, удовлетворяющие формуле φ.

Множество состояний S можно описать так:

$$S = \{\, s \in S_{pLTS} \mid (\exists s' \in S_{pLTS})\ [(s, s') \in R_\varphi \vee (s', s) \in R_\varphi] \,\},$$

т. е. S содержит только те состояния, которые присутствуют в отобранных переходах. Множество начальных состояний $S'_0 = \{\, s \in S \mid s \vDash \varphi_0 \,\}$, где φ_0 — формула инициализации из спецификации φ. Функция разметки $L' \colon S \to 2^P$ совпадает с L на множестве S, т. е. $(\forall s \in S)\ L'(s) = L(s)$.

Декларативная LTL-спецификация φ отбирает множество путей $[\![\varphi]\!]_{pLTS}$ из псевдополной системы переходов $pLTS$. Формуле φ соответствует построенная выше система переходов LTS, которая задаёт отобранное множество путей $\Pi_{LTS} = [\![\varphi]\!]_{pLTS}$.

Задачи управления конвейером. Рассмотрим простой пример задания поведения программы управления конвейером (рис. 4). Конвейер транспортирует детали (подаются из лотка слева) в тару (находится справа). На конвейере установлено два датчика: датчик наличия и датчик массы детали. Датчик наличия детали устанавливает $d = 1$, когда обнаруживает её наличие, и $d = 0$ при её отсутствии. Показания датчика массы детали w принимают значения из $\mathbb{N}_0$. За работу конвейера отвечает переменная con, которая принимает значения из $\{0, 1\}$.

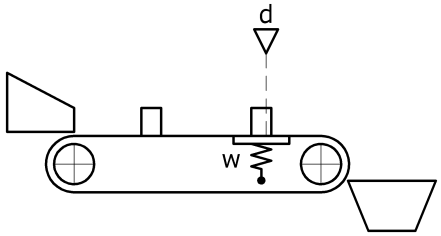

Fig. 4. Conveyor operation

Рис. 4. Работа конвейера

Пусть конвейер изначально включен и пуст: $(con = 1) \wedge (d = 0) \wedge (w = 0)$. Поведение входных переменных (d и w), отражающих значения датчиков, не специфицируем в виде LTL-формул, поэтому оно остаётся абсолютно недетерминированным.

Задача 1: требуется набрать заданное количество деталей $c_1 \in \mathbb{N}_0$ в тару и затем остановить конвейер. Для этого введём счётчик деталей n, который принимает значения из $\mathbb{N}_0$. Сначала опишем поведение программы, которая выполняет подсчёт деталей.

Формула инициализации программы подсчёта деталей имеет вид:

$$\varphi_0 : (d = 0) \wedge (n = 0) \wedge (\_d = d) \wedge (\_n = n).$$

LTL-формула, описывающая поведение счётчика деталей n, следующая:

$$\varphi_n : \mathbf{GX}\big(\neg(n = \_n) \Rightarrow\ (\_d = 0) \wedge (d = 1) \wedge (n = \_n + 1)\big) \wedge$$
$$\mathbf{GX}\big((n = \_n) \Rightarrow\ \neg\,[(\_d = 0) \wedge (d = 1)]\big).$$

Если значение счётчика n изменилось (новое значение отличается от предыдущего), то обнаружен передний фронт сигнала от датчика наличия детали (только новое значение d равно единице), и изменением является увеличение счётчика n на единицу. Вторая строка формулы φ_n утверждает, что если значение счётчика n не изменилось (новое значение равно предыдущему), то не было переднего фронта сигнала от датчика наличия детали.

Рассмотрим систему переходов программы подсчёта деталей (рис. 5b). Обозначение её состояний представлено на рис. 5a. Здесь над чертой указаны значения переменных n и d, под чертой — значения переменных $\_n$ и $\_d$. Значение переменной n выделяется рамкой.

На рис. 5b в начальном состоянии s_0 значения всех переменных равны нулю. Из любого состояния всегда исходят две дуги: одна для перехода при входном сигнале $d = 0$, другая — для перехода

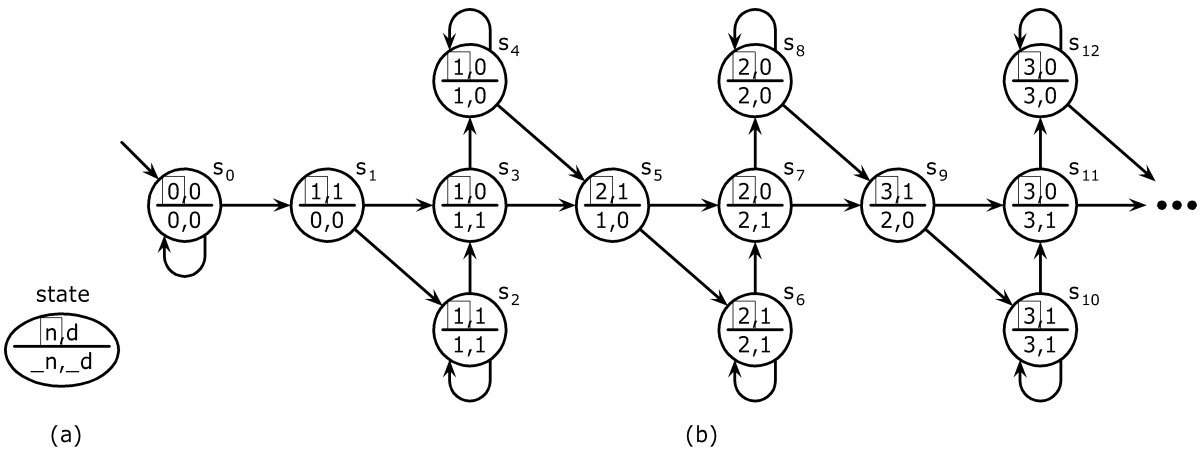

Fig. 5. State notation (a), transition system of the details counting program (b)

Рис. 5. Обозначение состояния (a), система переходов программы подсчёта деталей (b)

при входном сигнале $d = 1$. Видно, что переменные $\_n$ и $\_d$ в любом состоянии (кроме начального) принимают соответственно значения n и d из предыдущего состояния. Система переходов содержит только такие состояния, в которых отражена причина увеличения/сохранения значения переменной n — наличие/отсутствие переднего фронта сигнала от датчика. Других изменений значения переменной n не происходит, так как все прочие состояния удалены из системы переходов. При этом система переходов (рис. 5b) имеет бесконечное число состояний, в которых наблюдается монотонное увеличение значения переменной n при движении слева направо.

Теперь расширим программу, добавив в неё управление конвейером. Формула инициализации расширенной программы будет иметь вид:

$$\varphi_0 : (con = 1) \wedge (d = 0) \wedge (n = 0) \wedge (\_con = con) \wedge (\_d = d) \wedge (\_n = n).$$

После транспортировки заданного количества деталей конвейер должен остановиться. Запишем это с помощью следующей LTL-формулы спецификации поведения переменной *con*:

$$\begin{aligned}\varphi_{con1} : \mathbf{GX}\big(\neg(con = \_con) \implies (\_con = 1) \wedge (n = c_1) \wedge (con = 0)\big) \wedge \\ \mathbf{GX}\big((con = \_con) \implies \neg\,[(\_con = 1) \wedge (n = c_1)]\big).\end{aligned}$$

Если значение переменной *con* изменилось, то конвейер был включён, счётчик n достиг значения c_1, и изменение заключалось в выключении конвейера. Если значение переменной *con* не изменилось, то не возник нужный момент для отключения конвейера. Спецификация поведения всей программы — формула $\varphi = \varphi_0 \wedge \varphi_n \wedge \varphi_{con1}$.

Задача 2: требуется набрать в тару множество деталей с общей массой $c_2 \in \mathbb{N}_0$ и остановить конвейер. Для этого введём счётчик массы m, который принимает значения из $\mathbb{N}_0$. В этом случае формула инициализации программы имеет вид:

$$\varphi_0 : (con = 1) \wedge (d = 0) \wedge (m = 0) \wedge (\_con = con) \wedge (\_d = d) \wedge (\_m = m).$$

Поведение счётчика массы m зададим как

$$\begin{aligned}\varphi_m : \mathbf{GX}\big(\neg(m = \_m) \implies (\_d = 0) \wedge (d = 1) \wedge (m = \_m + w)\big) \wedge \\ \mathbf{GX}\big((m = \_m) \implies \neg\,[(\_d = 0) \wedge (d = 1)]\big).\end{aligned}$$

Если значение счётчика m изменилось, то обнаружен передний фронт сигнала от датчика наличия детали, а изменением является увеличение счётчика m на значение массы детали w, которая сейчас находится на измерительной подложке.

После транспортировки заданной массы деталей конвейер должен остановиться. LTL-формула, описывающая поведение переменной *con*, имеет следующий вид:

$$\begin{aligned}\varphi_{con2} : \mathbf{GX}\big(\neg(con = \_con) \implies (\_con = 1) \wedge (m \geq c_2) \wedge (con = 0)\big) \wedge \\ \mathbf{GX}\big((con = \_con) \implies \neg\,[(\_con = 1) \wedge (m \geq c_2)]\big).\end{aligned}$$

Если значение переменной *con* изменилось, то конвейер был включён, счётчик m достиг или превысил значение c_2, а изменение заключалось в выключении конвейера. Спецификация поведения всей программы — формула $\varphi = \varphi_0 \wedge \varphi_m \wedge \varphi_{con2}$.

Императивная LTL-спецификация. Для построения императивного кода программы по декларативной LTL-спецификации потребуется промежуточная формализация — императивная LTL-спецификация. Данная спецификация эквивалентна декларативной LTL-спецификации (выражает то же самое поведение), но при этом может быть непосредственно преобразована в императивный код программы, написанной, например, на языке ST.

Императивная LTL-спецификация для переменной v с учётом (3) и (4) имеет следующий вид:

$$\mathbf{GX}\big(cond_1 \Rightarrow (v = expr_1)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow (v = expr_k)\big), \tag{5}$$

$$\mathbf{GX}\big(\neg\, cond_1 \wedge \ldots \wedge \neg\, cond_k \Rightarrow (v = \_v)\big). \tag{6}$$

Эта спецификация описывает условия и соответствующие им правила изменения значения переменной v в стиле «если . . . , то . . . ».

4.2. Теорема об эквивалентности LTL-спецификаций

Лемма 1. *Из справедливости декларативной LTL-спецификации следует справедливость императивной LTL-спецификации, т. е. при условиях* (3) *и* (4) *имеем* (1), (2) $\vdash$ (5), (6).

Доказательство. Разобьём доказательство леммы на две части. В первой части покажем выполнимость вывода (4), (1), (2) $\vdash$ (5). Во второй части — справедливость логического вывода (1) $\vdash$ (6).

Докажем первую часть леммы:

Применим закон контрапозиции к формуле (2):

$$\mathbf{GX}\big((cond_1 \vee \ldots \vee cond_k) \Rightarrow \neg\,(v = \_v)\big). \tag{7}$$

Из (7) и (1) по транзитивности импликации получим:

$$\mathbf{GX}\big((cond_1 \vee \ldots \vee cond_k) \Rightarrow cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big) \Longleftrightarrow$$
$$\mathbf{GX}\Big(\neg cond_1 \wedge \ldots \wedge \neg cond_k \vee \big(cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big)\Big).$$

Отнесём выражение в скобках к каждому условию вида $cond_i$:

$$\mathbf{GX}\Big(\big(\neg cond_1 \vee cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big) \wedge \ldots \wedge$$
$$\big(\neg cond_k \vee cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big)\Big).$$

Отсюда следует первый конъюнкт:

$$\mathbf{GX}\big(\neg cond_1 \vee cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big) \Longleftrightarrow$$
$$\mathbf{GX}\big(cond_1 \Rightarrow cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big). \tag{8}$$

Из условия ортогональности (4) следует:

$$\mathbf{GX}\big(cond_1 \Rightarrow \neg cond_2 \wedge \ldots \wedge \neg cond_k\big). \tag{9}$$

Объединим формулы (8) и (9), получим:

$$\mathbf{GX}\Big(cond_1 \Rightarrow \big(cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big) \wedge \big(\neg cond_2 \wedge \ldots \wedge \neg cond_k\big)\Big).$$

Применим выражение в правой скобке к первому дизъюнкту левой скобки:

$$\mathbf{GX}\Big(cond_1 \Rightarrow cond_1 \wedge (v = expr_1) \wedge \neg cond_2 \wedge \ldots \wedge \neg cond_k \vee$$
$$\big(cond_2 \wedge (v = expr_2) \vee \ldots \vee cond_k \wedge (v = expr_k)\big) \wedge \big(\neg cond_2 \wedge \ldots \wedge \neg cond_k\big)\Big).$$

Вторая половина формулы является тождественно ложной. Сократим выражение:

$$\begin{aligned}&\mathbf{GX}\big(cond_1 \Rightarrow (v = expr_1) \wedge \neg cond_2 \wedge \ldots \wedge \neg cond_k\big) \Longleftrightarrow\\ &\mathbf{GX}\Big(\big(cond_1 \Rightarrow (v = expr_1)\big) \wedge \big(cond_1 \Rightarrow \neg cond_2 \wedge \ldots \wedge \neg cond_k\big)\Big) \vdash\\ &\mathbf{GX}\big(cond_1 \Rightarrow (v = expr_1)\big).\end{aligned} \tag{10}$$

Аналогичный вывод имеют формулы этого вида для остальных условий $cond_i$ при $i = 2, \ldots, k$. Взяв в конъюнкцию все эти формулы для всех условий $cond_i$, получим (5). Таким образом, мы доказали справедливость вывода (4), (1), (2) $\vdash$ (5).

Докажем вторую часть леммы: Применим закон контрапозиции к (1), получим:

$$\begin{aligned}&\mathbf{GX}\Big(\neg\big(cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big) \Rightarrow (v = \_v)\Big) \Longleftrightarrow\\ &\mathbf{GX}\Big(\neg\big(cond_1 \wedge (v = expr_1)\big) \wedge \ldots \wedge \neg\big(cond_k \wedge (v = expr_k)\big) \Rightarrow (v = \_v)\Big) \Longleftrightarrow\\ &\mathbf{GX}\Big(\big(\neg cond_1 \vee \neg(v = expr_1)\big) \wedge \ldots \wedge \big(\neg cond_k \vee \neg(v = expr_k)\big) \Rightarrow (v = \_v)\Big).\end{aligned}$$

После раскрытия скобок получится некоторая дизъюнктивная форма:

$$\mathbf{GX}\Big(\big(\neg cond_1 \wedge \ldots \wedge \neg cond_k \vee \ldots \vee \neg(v = expr_1) \wedge \ldots \wedge \neg(v = expr_k)\big) \Rightarrow (v = \_v)\Big).$$

Нас интересует первый дизъюнкт $\neg cond_1 \wedge \ldots \wedge \neg cond_k$, а всё остальное не имеет значения для доказательства, поэтому заменяется абстрактным выражением A. В итоге получим:

$$\begin{aligned}&\mathbf{GX}\Big(\big(\neg cond_1 \wedge \ldots \wedge \neg cond_k \vee A\big) \Rightarrow (v = \_v)\Big) \Longleftrightarrow\\ &\mathbf{GX}\Big(\big(\neg cond_1 \wedge \ldots \wedge \neg cond_k \Rightarrow (v = \_v)\big) \wedge \big(A \Rightarrow (v = \_v)\big)\Big).\end{aligned}$$

Отсюда следует (6). Таким образом, получили, что (1) $\vdash$ (6). Лемма 1 доказана. □

Лемма 2. *Из справедливости императивной LTL-спецификации следует справедливость декларативной LTL-спецификация, т. е. при условиях* (3) *и* (4) *имеем* (5), (6) $\vdash$ (1), (2).

Доказательство. Сначала докажем, что (5), (6) $\vdash$ (1). Преобразуем (5):

$$\mathbf{GX}\big(cond_1 \Rightarrow cond_1 \wedge (v = expr_1)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow cond_k \wedge (v = expr_k)\big). \tag{11}$$

Применим закон контрапозиции к (6), получим:

$$\begin{aligned}&\mathbf{GX}\big(\neg(v = \_v) \Rightarrow cond_1 \vee \ldots \vee cond_k\big) \Longleftrightarrow\\ &\mathbf{GX}\Big(\big(\neg(v = \_v) \Rightarrow cond_1\big) \vee \ldots \vee \big(\neg(v = \_v) \Rightarrow cond_k\big)\Big).\end{aligned} \tag{12}$$

Объединим (12) и (11):

$$\begin{aligned}&\mathbf{GX}\Big(\big[\big(\neg(v = \_v) \Rightarrow cond_1\big) \vee \ldots \vee \big(\neg(v = \_v) \Rightarrow cond_k\big)\big] \wedge\\ &\big[\big(cond_1 \Rightarrow cond_1 \wedge (v = expr_1)\big) \wedge \ldots \wedge \big(cond_k \Rightarrow cond_k \wedge (v = expr_k)\big)\big]\Big).\end{aligned}$$

Раскроем первые квадратные скобки:

$$\mathbf{GX}\Big(\big(\neg(v = \_v) \Rightarrow cond_1\big) \wedge \big[\big(cond_1 \Rightarrow cond_1 \wedge (v = expr_1)\big) \wedge \ldots \wedge \big(cond_k \Rightarrow cond_k \wedge (v = expr_k)\big)\big] \vee \ldots \vee$$
$$\big(\neg(v = \_v) \Rightarrow cond_k\big) \wedge \big[\big(cond_1 \Rightarrow cond_1 \wedge (v = expr_1)\big) \wedge \ldots \wedge \big(cond_k \Rightarrow cond_k \wedge (v = expr_k)\big)\big]\Big) \vdash$$
$$\mathbf{GX}\Big(\big(\neg(v = \_v) \Rightarrow cond_1\big) \wedge \big(cond_1 \Rightarrow cond_1 \wedge (v = expr_1)\big) \vee \ldots \vee$$
$$\big(\neg(v = \_v) \Rightarrow cond_k\big) \wedge \big(cond_k \Rightarrow cond_k \wedge (v = expr_k)\big)\Big) \vdash$$
$$\mathbf{GX}\big(\neg(v = \_v) \Rightarrow cond_1 \wedge (v = expr_1) \vee \ldots \vee \neg(v = \_v) \Rightarrow cond_k \wedge (v = expr_k)\big).$$

Из последней формулы получаем (1). Справедливость вывода (5), (6) $\vdash$ (1) доказана.
Теперь докажем, что (3), (5) $\vdash$ (2). Объединим (5) и (3), получим:

$$\mathbf{GX}\Big(\big(cond_1 \Rightarrow (v = expr_1)\big) \wedge \ldots \wedge \big(cond_k \Rightarrow (v = expr_k)\big) \wedge \ldots \wedge$$
$$\big(cond_1 \Rightarrow \neg(\_v = expr_1)\big) \wedge \ldots \wedge \big(cond_k \Rightarrow \neg(\_v = expr_k)\big)\Big) \Longleftrightarrow$$
$$\mathbf{GX}\Big(\big(cond_1 \Rightarrow (v = expr_1) \wedge \neg(\_v = expr_1)\big) \wedge \ldots \wedge \big(cond_k \Rightarrow (v = expr_k) \wedge \neg(\_v = expr_k)\big)\Big) \vdash$$
$$\mathbf{GX}\Big(\big(cond_1 \Rightarrow \neg(v = \_v)\big) \wedge \ldots \wedge \big(cond_k \Rightarrow \neg(v = \_v)\big)\Big).$$

Применим закон контрапозиции к последней формуле:

$$\mathbf{GX}\Big(\big((v = \_v) \Rightarrow \neg cond_1\big) \wedge \ldots \wedge \big((v = \_v) \Rightarrow \neg cond_k\big)\Big) \Longleftrightarrow$$
$$\mathbf{GX}\big((v = \_v) \Rightarrow \neg cond_1 \wedge \ldots \wedge \neg cond_k\big) \Longleftrightarrow \mathbf{GX}\big((v = \_v) \Rightarrow \neg(cond_1 \vee \ldots \vee cond_k)\big).$$

Получили формулу (2). Следовательно, вывод (3), (5) $\vdash$ (2) является справедливым.
Лемма 2 доказана. □

Теорема 1 (Об эквивалентности спецификаций)**.** *Декларативная* (1), (2) *и императивная* (5), (6) *LTL-спецификации эквивалентны при соблюдении условий изменчивости* (3) *и ортогональности* (4)*.*

Доказательство. Следует из Лемм 1 и 2. □

5. Выразительность декларативной LTL-спецификации

Оценим выразительные возможности декларативной LTL-спецификации в смысле Тьюринг-полноты. Для доказательства теоремы о Тьюринг-полноте декларативной LTL-спецификации в качестве формальной модели алгоритма выберем счётчиковую машину Минского [35—37]. Она эквивалентна по вычислительным возможностям машине Тьюринга, но представляется более удобной для описания поведения программ, так как имеет более наглядный программный вид, т. е. вид компьютерной программы, написанной на языке высокого уровня.

Счётчиковая машина Минского M представляет собой набор $\langle Q, q_1, q_n, Y, \Delta\rangle$, где $Q = \{q_1, \ldots, q_n\}$ — конечное непустое множество управляющих состояний машины; $q_1 \in Q$ — начальное управляющее состояние; $q_n \in Q$ — заключительное, или финальное, управляющее состояние; $Y = \{y_1, \ldots, y_m\}$ — конечное непустое множество счётчиков, которые могут принимать значения из $\mathbb{N}_0$; $\Delta = \{\delta_1, \ldots, \delta_{n-1}\}$ — набор правил переходов по управляющим состояниям машины; δ_i — правило переходов для управляющего состояния q_i, где $i = 1, \ldots, n-1$.

Состояния q_i, $1 \leqslant i \leqslant n-1$, подразделяются на два типа. Управляющие состояния первого типа имеют правила переходов вида:

$$(\delta_i)\ q_i\colon\ y := y + 1;\ \textbf{goto}\ q_k, \tag{13}$$

где $y \in Y$, $q_k \in Q$. Для управляющих состояний второго типа определяются правила переходов вида:

$$(\delta_i)\ q_i\colon \textbf{if}\ y > 0\ \textbf{then}\ (y := y - 1;\ \textbf{goto}\ q_k)\ \textbf{else goto}\ q_l, \tag{14}$$

где $q_k, q_l \in Q$. Для финального состояния q_n правил переходов не предусмотрено. При попадании в финальное состояние q_n машина завершает свою работу.

Состояние (*конфигурация*) счётчиковой машины — это набор $(q, c_1, \ldots, c_m)$, где $q \in Q$, $c_i \in \mathbb{N}_0$, $i = 1, \ldots, m$; здесь $c_1, \ldots, c_m$ являются значениями соответствующих счётчиков $y_1, \ldots, y_m$.

Исполнением машины Минского называется последовательность состояний $s_0\, s_1\, s_2\, \ldots$, индуктивно определяемая в соответствии с правилами переходов. Счётчиковая машина имеет одно исполнение из начального состояния s_0, так как для каждого управляющего состояния предусмотрено не более одного правила переходов.

Машина, получив на вход некоторый набор значений счетчиков, стартует из управляющего состояния q_1 и либо останавливается в управляющем состоянии q_n с выходным набором значений счетчиков, либо зацикливается, реализуя тем самым частичную числовую функцию.

Рассмотрим в качестве примера трехсчётчиковую машину Минского *3cM*, реализующую функцию возведения числа n в квадрат, где $n \in \mathbb{N}_0$. Правила переходов по управляющим состояниям машины *3cM* представлены ниже [36]:

$$\begin{aligned}
&(\delta_1)\ q_1\colon \textbf{if}\ a > 0\ \textbf{then}\ (a := a - 1;\ \textbf{goto}\ q_2)\ \textbf{else goto}\ q_8;\\
&(\delta_2)\ q_2\colon\ c := c + 1;\ \textbf{goto}\ q_3;\\
&(\delta_3)\ q_3\colon \textbf{if}\ a > 0\ \textbf{then}\ (a := a - 1;\ \textbf{goto}\ q_4)\ \textbf{else goto}\ q_6;\\
&(\delta_4)\ q_4\colon\ b := b + 1;\ \textbf{goto}\ q_5;\\
&(\delta_5)\ q_5\colon\ c := c + 1;\ \textbf{goto}\ q_2;\\
&(\delta_6)\ q_6\colon \textbf{if}\ b > 0\ \textbf{then}\ (b := b - 1;\ \textbf{goto}\ q_7)\ \textbf{else goto}\ q_1;\\
&(\delta_7)\ q_7\colon\ a := a + 1;\ \textbf{goto}\ q_6.
\end{aligned} \tag{15}$$

Счётчиковая машина *3cM* имеет восемь управляющих состояний $q_1, \ldots, q_8$, где q_1 является начальным управляющим состоянием, а q_8 — финальным. Множество счетчиков $Y = \{a, b, c\}$. Предполагается, что в начальном состоянии счетчик a получает значение n, а начальные значения двух других счетчиков b и c равны нулю. В финальном состоянии результат вычисления будет содержаться в счетчике c при нулевых значениях счетчиков a и b.

Правила переходов по управляющим состояниям машины *3cM* представлены в графическом виде на рис. 6, где для счётчика $y \in Y$ обозначение «$y+$» соответствует его увеличению на единицу, а «$y-$» используется для обозначения условного вычитания единицы с переходом в другое управляющее состояние по правосторонней стрелке в случае нулевого значения счётчика y.

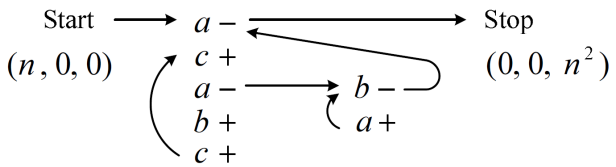

Fig. 6. Graphical representation of the counter machine *3cM* of squaring a number

Рис. 6. Графическое представление счётчиковой машины *3cM* возведения числа в квадрат

Теорема 2 (О Тьюринг-полноте LTL-спецификации)**.** *Для любой счётчиковой машины Минского её поведение, т. е. множество всех возможных исполнений, может быть задано с помощью декларативной LTL-спецификации.*

Доказательство. В качестве доказательства приведём общую процедуру построения декларативной LTL-спецификации поведения программы, которая моделирует работу произвольной счётчиковой машины Минского.

Пусть имеется некоторая m-счётчиковая машина Минского, представленная в виде набора правил переходов. Программа будет содержать основные $q, y_1, \ldots, y_m$ и вспомогательные $\_q, \_y_1, \ldots, \_y_m$ переменные. Переменная q предназначена для хранения номера текущего управляющего состояния, переменные $y_1, \ldots, y_m$ — для хранения значений счётчиков машины. Таким образом, вектор $(q, y_1, \ldots, y_m, \_q, \_y_1, \ldots, \_y_m)$ описывает текущее состояние машины. Все переменные программы принимают значения из $\mathbb{N}_0$. Любому переходу (согласно правилам переходов) машины Минского соответствует переход между состояниями программы. Если для текущего состояния машины ни одно из правил переходов не выполняется, то машина остаётся в прежнем состоянии. При этом в программе происходит переход по петле в то же самое программное состояние.

Формула инициализации программы, описывающая множество начальных состояний, будет иметь следующий вид:

$$q = 1 \wedge y_1 \geqslant 0 \wedge \cdots \wedge y_m \geqslant 0 \wedge \_q = q \wedge \_y_1 = y_1 \wedge \cdots \wedge \_y_m = y_m.$$

Далее, выполним LTL-формализацию правил переходов для управляющих состояний $q_i \in Q$ первого типа следующим образом:

$$\mathbf{GX}\big((\_q = i) \Rightarrow (y = \_y + 1) \wedge (q = k)\big). \tag{16}$$

LTL-формализация правил переходов для состояний $q_i \in Q$ второго типа будет иметь вид:

$$\mathbf{GX}\Big(\big[(\_q = i) \wedge (\_y > 0) \Rightarrow (y = \_y - 1) \wedge (q = k)\big] \wedge \big[(\_q = i) \wedge \neg(\_y > 0) \Rightarrow (q = l)\big]\Big). \tag{17}$$

Каждую формулу вида (16) разобьём на две LTL-формулы:

$$\begin{aligned} &\mathbf{GX}\big((\_q = i) \Rightarrow (y = \_y + 1)\big), \\ &\mathbf{GX}\big((\_q = i) \Rightarrow (q = k)\big). \end{aligned} \tag{18}$$

Фрагментация формул (17) будет такой:

$$\begin{aligned} &\mathbf{GX}\big((\_q = i) \wedge (\_y > 0) \Rightarrow (y = \_y - 1)\big), \\ &\mathbf{GX}\big((\_q = i) \wedge (\_y > 0) \Rightarrow (q = k)\big), \\ &\mathbf{GX}\big((\_q = i) \wedge \neg(\_y > 0) \Rightarrow (q = l)\big). \end{aligned} \tag{19}$$

Все счётчики имеют одну и ту же процедуру построения спецификации. Покажем, как построить спецификацию для произвольного счётчика $y \in Y$. Сгруппируем все формулы, в которых фигурирует счётчик y, получим:

$$\begin{aligned} &\mathbf{GX}\big((\_q = e_{11}) \Rightarrow (y = \_y + 1)\big), \ldots, \\ &\mathbf{GX}\big((\_q = e_{1s}) \Rightarrow (y = \_y + 1)\big), \\ &\mathbf{GX}\big((\_q = e_{21}) \wedge (\_y > 0) \Rightarrow (y = \_y - 1)\big), \ldots, \\ &\mathbf{GX}\big((\_q = e_{2p}) \wedge (\_y > 0) \Rightarrow (y = \_y - 1)\big). \end{aligned} \tag{20}$$

Конъюнктивно объединим формулы в группе (20), а также добавим формулу, описывающую отсутствие изменений значения переменной. Получим императивную LTL-спецификацию для y:

$$\begin{aligned}
&\mathbf{GX}\big((_q = e_{11}) \Rightarrow (y = _y + 1)\big) \wedge \ldots \wedge \\
&\mathbf{GX}\big((_q = e_{1s}) \Rightarrow (y = _y + 1)\big) \wedge \\
&\mathbf{GX}\big((_q = e_{21}) \wedge (_y > 0) \Rightarrow (y = _y - 1)\big) \wedge \ldots \wedge \\
&\mathbf{GX}\big((_q = e_{2p}) \wedge (_y > 0) \Rightarrow (y = _y - 1)\big), \\
&\mathbf{GX}\Big(\neg\big(_q = e_{11}\big) \wedge \ldots \wedge \neg\big(_q = e_{1s}\big) \wedge \\
&\qquad \neg\big((_q = e_{21}) \wedge (_y > 0)\big) \wedge \ldots \wedge \neg\big((_q = e_{2p}) \wedge (_y > 0)\big) \Rightarrow (y = _y)\Big).
\end{aligned} \tag{21}$$

Также покажем, как построить спецификацию для переменной q. Сгруппируем все формулы, в которых она фигурирует, получим:

$$\begin{aligned}
&\mathbf{GX}\big((_q = d_{11}) \Rightarrow (q = g_{11})\big), \ldots, \\
&\mathbf{GX}\big((_q = d_{1h}) \Rightarrow (q = g_{1h})\big), \\
&\mathbf{GX}\big((_q = d_{21}) \wedge (_y > 0) \Rightarrow (q = g_{21})\big), \ldots, \\
&\mathbf{GX}\big((_q = d_{2t}) \wedge (_y > 0) \Rightarrow (q = g_{2t})\big), \\
&\mathbf{GX}\big((_q = d_{31}) \wedge \neg(_y > 0) \Rightarrow (q = g_{31})\big), \ldots, \\
&\mathbf{GX}\big((_q = d_{3z}) \wedge \neg(_y > 0) \Rightarrow (q = g_{3z})\big).
\end{aligned} \tag{22}$$

Удалим из группы (22) формулы, в которых $d_i = g_i$, где $i = 11\ldots1h, 21\ldots2t, 31\ldots3z$. Оставшиеся формулы конъюнктивно объединим, а также добавим формулу, описывающую отсутствие изменений значения переменной. Получим императивную LTL-спецификацию для q:

$$\begin{aligned}
&\mathbf{GX}\big((_q = d_{11}) \Rightarrow (q = g_{11})\big) \wedge \ldots \wedge \\
&\mathbf{GX}\big((_q = d_{1h}) \Rightarrow (q = g_{1h})\big) \wedge \\
&\mathbf{GX}\big((_q = d_{21}) \wedge (_y > 0) \Rightarrow (q = g_{21})\big) \wedge \ldots \wedge \\
&\mathbf{GX}\big((_q = d_{2t}) \wedge (_y > 0) \Rightarrow (q = g_{2t})\big) \wedge \\
&\mathbf{GX}\big((_q = d_{31}) \wedge \neg(_y > 0) \Rightarrow (q = g_{31})\big) \wedge \ldots \wedge \\
&\mathbf{GX}\big((_q = d_{3z}) \wedge \neg(_y > 0) \Rightarrow (q = g_{3z})\big), \\
&\mathbf{GX}\Big(\neg\big(_q = d_{11}\big) \wedge \ldots \wedge \neg\big(_q = d_{1h}\big) \wedge \\
&\qquad \neg\big((_q = d_{21}) \wedge (_y > 0)\big) \wedge \ldots \wedge \neg\big((_q = d_{2t}) \wedge (_y > 0)\big) \wedge \\
&\qquad \neg\big((_q = d_{31}) \wedge \neg(_y > 0)\big) \wedge \ldots \wedge \neg\big((_q = d_{3z}) \wedge \neg(_y > 0)\big) \Rightarrow (q = _q)\Big).
\end{aligned} \tag{23}$$

На основании императивных LTL-спецификаций (21) и (23) построим эквивалентные декларативные LTL-спецификации для y и q соответственно:

$$\begin{aligned}
\mathbf{GX}\big(&\neg(y = _y) \Rightarrow (_q = e_{11}) \wedge (y = _y + 1) \vee \ldots \vee (_q = e_{1s}) \wedge (y = _y + 1) \vee \\
&(_q = e_{21}) \wedge (_y > 0) \wedge (y = _y - 1) \vee \ldots \vee (_q = e_{2p}) \wedge (_y > 0) \wedge (y = _y - 1)\big), \\
\mathbf{GX}\big(&(y = _y) \Rightarrow \neg\big((_q = e_{11}) \vee \ldots \vee (_q = e_{1s}) \vee \\
&(_q = e_{21}) \wedge (_y > 0) \vee \ldots \vee (_q = e_{2p}) \wedge (_y > 0)\big)\big).
\end{aligned} \tag{24}$$

$$\begin{aligned}
&\mathbf{GX}\big(\neg(q = _q) \Rightarrow (_q = d_{11}) \wedge (q = g_{11}) \vee \ldots \vee (_q = d_{1h}) \wedge (q = g_{1h}) \vee \\
&\qquad (_q = d_{21}) \wedge (_y > 0) \wedge (q = g_{21}) \vee \ldots \vee (_q = d_{2t}) \wedge (_y > 0) \wedge (q = g_{2t}) \vee \\
&\qquad (_q = d_{31}) \wedge \neg(_y > 0) \wedge (q = g_{31}) \vee \ldots \vee (_q = d_{3z}) \wedge \neg(_y > 0) \wedge (q = g_{3z})\big), \\
&\mathbf{GX}\big((q = _q) \Rightarrow \neg\big((_q = d_{11}) \vee \ldots \vee (_q = d_{1h}) \vee \\
&\qquad (_q = d_{21}) \wedge (_y > 0) \vee \ldots \vee (_q = d_{2t}) \wedge (_y > 0) \vee \\
&\qquad (_q = d_{31}) \wedge \neg(_y > 0) \vee \ldots \vee (_q = d_{3z}) \wedge \neg(_y > 0)\big)\big).
\end{aligned} \tag{25}$$

Декларативная LTL-спецификация поведения программы, моделирующей работу счётчиковой машины, — формула инициализации и набор формул, описывающих поведение переменной q (25), а также всех переменных $y_1, \ldots, y_m$ в виде (24). Формулы (25) и (24) удовлетворяют условиям изменчивости (3) и ортогональности (4). □

Процедура построения декларативной LTL-спецификации φ_{3cM} счётчиковой машины *3cM* представлена в приложении. Сама спецификация φ_{3cM} имеет следующий вид:

$$\mathrm{S0}: \quad (q = 1) \wedge (a \geqslant 0) \wedge (b \geqslant 0) \wedge (c \geqslant 0) \wedge (\_q = q) \wedge (\_a = a) \wedge (\_b = b) \wedge (\_c = c);$$

$$\begin{aligned}
\mathrm{S}a: \quad &\mathbf{GX}\big(\neg(a = _a) \Rightarrow (_q = 1) \wedge (_a > 0) \wedge (a = _a - 1) \vee \\
&\qquad (_q = 3) \wedge (_a > 0) \wedge (a = _a - 1) \vee \\
&\qquad (_q = 7) \qquad\quad \wedge (a = _a + 1) \quad \big) \wedge \\
&\mathbf{GX}\big((a = _a) \Rightarrow \neg\big((_q = 1) \wedge (_a > 0) \vee (_q = 3) \wedge (_a > 0) \vee (_q = 7)\big)\big);
\end{aligned}$$

$$\begin{aligned}
\mathrm{S}b: \quad &\mathbf{GX}\big(\neg(b = _b) \Rightarrow (_q = 4) \qquad\quad \wedge (b = _b + 1) \vee \\
&\qquad (_q = 6) \wedge (_b > 0) \wedge (b = _b - 1) \quad \big) \wedge \\
&\mathbf{GX}\big((b = _b) \Rightarrow \neg\big((_q = 4) \vee (_q = 6) \wedge (_b > 0)\big)\big);
\end{aligned}$$

$$\begin{aligned}
\mathrm{S}c: \quad &\mathbf{GX}\big(\neg(c = _c) \Rightarrow (_q = 2) \wedge (c = _c + 1) \vee \\
&\qquad (_q = 5) \wedge (c = _c + 1) \quad \big) \wedge \\
&\mathbf{GX}\big((c = _c) \Rightarrow \neg\big((_q = 2) \vee (_q = 5)\big)\big);
\end{aligned}$$

$$\begin{aligned}
\mathrm{S}q: \quad &\mathbf{GX}\big(\neg(q = _q) \Rightarrow (_q = 1) \wedge \quad (_a > 0) \wedge (q = 2) \vee \\
&\qquad (_q = 1) \wedge \neg(_a > 0) \wedge (q = 8) \vee \\
&\qquad (_q = 2) \qquad\quad \wedge (q = 3) \vee \\
&\qquad (_q = 3) \wedge \quad (_a > 0) \wedge (q = 4) \vee \\
&\qquad (_q = 3) \wedge \neg(_a > 0) \wedge (q = 6) \vee \\
&\qquad (_q = 4) \qquad\quad \wedge (q = 5) \vee \\
&\qquad (_q = 5) \qquad\quad \wedge (q = 2) \vee \\
&\qquad (_q = 6) \wedge \quad (_b > 0) \wedge (q = 7) \vee \\
&\qquad (_q = 6) \wedge \neg(_b > 0) \wedge (q = 1) \vee \\
&\qquad (_q = 7) \qquad\quad \wedge (q = 6) \quad \big) \wedge \\
&\mathbf{GX}\big((q = _q) \Rightarrow \neg\big((_q = 1) \wedge (_a > 0) \vee (_q = 1) \wedge \neg(_a > 0) \vee (_q = 2) \vee \\
&\qquad (_q = 3) \wedge (_a > 0) \vee (_q = 3) \wedge \neg(_a > 0) \vee (_q = 4) \vee (_q = 5) \vee \\
&\qquad (_q = 6) \wedge (_b > 0) \vee (_q = 6) \wedge \neg(_b > 0) \vee (_q = 7)\big)\big).
\end{aligned}$$

Заметим, что последнюю формулу LTL-спецификации можно упростить следующим образом:

$$\mathbf{GX}\big((q = \_q) \Rightarrow \neg\big((\_q = 1) \vee (\_q = 2) \vee (\_q = 3) \vee (\_q = 4) \vee (\_q = 5) \vee (\_q = 6) \vee (\_q = 7)\big)\big).$$

Счётчиковая машина *3cM* имеет всего восемь управляющих состояний $q_1, \ldots, q_8$, поэтому переменная $q = 1, \ldots, 8$. Из упрощённой формулы видно, что если $\_q = 1, \ldots, 7$, то значение q изменяется.

Это говорит о том, что машина *3cM* всегда переходит в другое управляющее состояние. Исключением является только финальное управляющее состояние q_8, в котором машина останавливается.

6. Верификация декларативной LTL-спецификации

6.1. Постановка задачи

Исходная задача верификации декларативной LTL-спецификации

$$[[\varphi]]_{pLTS} \vDash \psi \tag{26}$$

является задачей проверки модели, т. е. проверки того, является ли интерпретация $[[\varphi]]_{pLTS}$ моделью формулы ψ, где φ — декларативная LTL-спецификация, задающая множество путей в псевдополной системе переходов *pLTS*, а ψ — проверяемое LTL-свойство.

Мы намерены решать данную задачу с помощью программного средства nuXmv — инструмента символьной *проверки модели* (model checking) [12—15], который позволяет работать с моделями с конечным и бесконечным числом состояний.

Важно отметить, что задача (26) в общем случае неразрешима, т. е. не существует алгоритма проверки выполнимости формулы ψ на интерпретации $[[\varphi]]_{pLTS}$. Если бы такая процедура существовала, то можно было бы выяснить для произвольной счётчиковой машины *cM* справедливость $[[\varphi_{cM}]]_{pLTS} \vDash \mathbf{F}\,\mathbf{G}\,(q = q_n)$, где φ_{cM} — декларативная LTL-спецификация поведения этой счётчиковой машины, а q_n — её финальное состояние. И мы бы имели решение проблемы *тотальности* для счётчиковых машин Минского, т. е. имели бы алгоритм, который для любой счётчиковой машины определял, будет ли она останавливаться при всех возможных начальных значениях счётчиков. Но известно, что проблема тотальности не является даже частично разрешимой уже для двухсчётиковых машин Минского [37].

Для разрешимости задачи (26) потребуем, чтобы множество состояний $S = (D_1 \times \ldots \times D_n)^2$ псевдополной системы переходов $pLTS = \langle S, S_0, R'_c, P, L\rangle$ было конечным. Для этого необходимо ограничить область значений D_i, где $i = 1, \ldots, n$, каждой переменной из $V = \{v_1, \ldots, v_n\}$. В этом случае конечная система переходов *pLTS* становится *структурой Крипке* [12—15].

Для корректного описания поведения конечных программ декларативная LTL-спецификация должна удовлетворять дополнительному условию *ограниченности* для каждой переменной $v_i \in V$:

$$\mathbf{GX}\big(cond_1 \Rightarrow (val(expr_1) \in D_i)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow (val(expr_k) \in D_i)\big), \tag{27}$$

т. е. если условие $cond_j$ истинно ($j = 1, \ldots, k$), то выражение $expr_j$ возвращает значение $val(expr_j)$, которое принадлежит области значений данной переменной. А также потребуем, чтобы в формуле инициализации содержались выражения, обеспечивающие попадание начального значения каждой переменной в допустимую область значений.

Декларативная LTL-спецификация с ограничением (27) позволяет, например, задавать поведение *конечной* счётчиковой машины Минского, которая представляет собой счётчиковую машину с ограничениями на значения счётчиков и модифицированным правилом переходов для управляющих состояний первого типа. Ограничения вводятся через отображение $\mathrm{bnd} : Y \to \mathbb{N}_0$, устанавливающее верхнее предельное значение $\mathrm{bnd}y \in \mathbb{N}_0$ для каждого счётчика $y \in Y$. А модифицированное правило переходов для управляющих состояний q_i первого типа имеет вид:

$$(\delta_i)\ q_i\text{: } \mathbf{if}\ y < \mathrm{bnd}y\ \mathbf{then}\ (y := y + 1;\ \mathbf{goto}\ q_k)\ \mathbf{else\ goto}\ q_l.$$

В качестве примера в следующем параграфе данного раздела будет рассмотрена конечная счётчиковая машина *3cM′* возведения числа в квадрат.

Для решения задачи проверки модели (26) с помощью инструмента nuXmv необходимо задать множество путей $[[\varphi]]_{pLTS}$ в некотором приемлемом для этого инструмента виде. Так как φ и ψ

уже являются LTL-формулами, которые поддерживает nuXmv, то самым простым решением будет не преобразовывать их в другую форму, а оставить как есть. При этом в соответствии с определением логики LTL задача (26) может быть сведена к следующей задаче:

$$pLTS \vDash (\varphi \Rightarrow \psi). \tag{28}$$

Здесь проверяется выполнимость импликации $\varphi \Rightarrow \psi$ на псевдополной системе переходов $pLTS$. Верификатор в данном случае будет обходить множество всех возможных путей в $pLTS$ и проверять истинность свойства ψ только на тех путях, на которых истинна спецификация поведения φ.

В рамках такой постановки задачи верификации потребуется задать псевдополную систему переходов $pLTS$ на входном языке верификатора nuXmv. Для этого нужно объявить два набора переменных $V = \{v_1, \ldots, v_n\}$ и $\_V = \{\_v_1, \ldots, \_v_n\}$, а затем описать поведение каждой вспомогательной переменной $\_v \in \_V$ с помощью конструкции **next** $(\_v) := v$, обеспечивающей сохранение в $\_v$ предыдущего значения соответствующей переменной $v \in V$. По умолчанию в nuXmv поведение переменных из V является абсолютно недетерминированным. После задания $pLTS$ можно запускать инструмент nuXmv для проверки выполнимости формулы $\varphi \Rightarrow \psi$. Если эта формула является истинной, значит, спецификация поведения φ удовлетворяет свойству ψ, в противном случае спецификация φ не соответствует свойству ψ.

6.2. Конечная счётчиковая машина возведения числа в квадрат

Построим конечную счётчиковую машину $3cM'$ возведения числа в квадрат при ограничениях $\mathrm{bnd}a$, $\mathrm{bnd}b$ и $\mathrm{bnd}c$ на счётчики a, b и c соответственно:

(δ_1) q_1: **if** $a > 0$ **then** $(a := a - 1;$ **goto** $q_2)$ **else goto** q_8;
(δ_2) q_2: **if** $c < \mathrm{bnd}c$ **then** $(c := c + 1;$ **goto** $q_3)$ **else goto** q_3;
(δ_3) q_3: **if** $a > 0$ **then** $(a := a - 1;$ **goto** $q_4)$ **else goto** q_6;
(δ_4) q_4: **if** $b < \mathrm{bnd}b$ **then** $(b := b + 1;$ **goto** $q_5)$ **else goto** q_5;
(δ_5) q_5: **if** $c < \mathrm{bnd}c$ **then** $(c := c + 1;$ **goto** $q_2)$ **else goto** q_2;
(δ_6) q_6: **if** $b > 0$ **then** $(b := b - 1;$ **goto** $q_7)$ **else goto** q_1;
(δ_7) q_7: **if** $a < \mathrm{bnd}a$ **then** $(a := a + 1;$ **goto** $q_6)$ **else goto** q_6.

Приведём декларативную LTL-спецификацию поведения конечной счётчиковой машины $3cM'$ в той её части, которая отличается от декларативной спецификации исходной машины $3cM$ возведения числа в квадрат. Изменятся только формулы S0, Sa, Sb и Sc:

$$\mathrm{S0}: \quad (q = 1) \wedge (a \geqslant 0) \wedge (a \leqslant \mathrm{bnd}a) \wedge (b \geqslant 0) \wedge (b \leqslant \mathrm{bnd}b) \wedge (c \geqslant 0) \wedge (c \leqslant \mathrm{bnd}c) \wedge (\_q = q) \wedge (\_a = a) \wedge (\_b = b) \wedge (\_c = c);$$

$$\begin{aligned}\mathrm{S}a: \quad & \mathbf{GX}\big(\neg(a = \_a) \Rightarrow (\_q = 1) \wedge (\_a > 0) \quad \wedge (a = \_a - 1) \vee \\ & \qquad (\_q = 3) \wedge (\_a > 0) \quad \wedge (a = \_a - 1) \vee \\ & \qquad (\_q = 7) \wedge (\_a < \mathrm{bnd}a) \wedge (a = \_a + 1) \quad \big) \wedge \\ & \mathbf{GX}\big((a = \_a) \Rightarrow \neg\big((\_q = 1) \wedge (\_a > 0) \vee (\_q = 3) \wedge (\_a > 0) \vee (\_q = 7) \wedge (\_a < \mathrm{bnd}a)\big)\big);\end{aligned}$$

$$\begin{aligned}\mathrm{S}b: \quad & \mathbf{GX}\big(\neg(b = \_b) \Rightarrow (\_q = 4) \wedge (\_b < \mathrm{bnd}b) \wedge (b = \_b + 1) \vee \\ & \qquad (\_q = 6) \wedge (\_b > 0) \quad \wedge (b = \_b - 1) \quad \big) \wedge \\ & \mathbf{GX}\big((b = \_b) \Rightarrow \neg\big((\_q = 4) \wedge (\_b < \mathrm{bnd}b) \vee (\_q = 6) \wedge (\_b > 0)\big)\big);\end{aligned}$$

$$\begin{aligned}\mathrm{S}c: \quad & \mathbf{GX}\big(\neg(c = \_c) \Rightarrow (\_q = 2) \wedge (\_c < \mathrm{bnd}c) \wedge (c = \_c + 1) \vee \\ & \qquad (\_q = 5) \wedge (\_c < \mathrm{bnd}c) \wedge (c = \_c + 1) \quad \big) \wedge \\ & \mathbf{GX}\big((c = \_c) \Rightarrow \neg\big((\_q = 2) \wedge (\_c < \mathrm{bnd}c) \vee (\_q = 5) \wedge (\_c < \mathrm{bnd}c)\big)\big).\end{aligned}$$

Формула Sq останется неизменной, так как в каждом правиле переходов первого типа для *3cM′* обе ветки условия ведут в одно и то же управляющее состояние. Поэтому для переменной q формула описания её поведения упрощается до прежнего варианта Sq.

Рассмотрим некоторые LTL-свойства счётчиковой машины *3cM′*, которые обязательно должны выполняться для любого её исполнения.

$$(a = n \wedge b = 0 \wedge c = 0 \wedge n \geqslant 0 \wedge \mathbf{G}(n = \_n)) \Rightarrow \mathbf{G}(\ q = 8 \ \Rightarrow \ c = n^2 \wedge a = 0 \wedge b = 0\ ).$$

Это свойство означает, что если счётчиковая машина *3cM′*, стартуя при $a = n$, $b = 0$ и $c = 0$, оказывается в финальном управляющем состоянии q_8, то значения счётчиков a и b будут равны 0, а счётчик c будет содержать искомый результат n^2.

$$(a = n \wedge b = 0 \wedge c = 0 \wedge n \geqslant 0 \wedge \mathbf{G}(n = \_n)) \Rightarrow \mathbf{G}(\ a + b \leqslant n\ ).$$

Это свойство требует, чтобы суммарное значение счётчиков a и b было ограничено константой n на протяжении всего исполнения счётчиковой машины *3cM′*.

$$(a = n \wedge b = 0 \wedge c = 0 \wedge n \geqslant 0 \wedge \mathbf{G}(n = \_n)) \Rightarrow \mathbf{G}(\ c \leqslant n^2\ ).$$

Требуется, чтобы значение счётчика c на протяжении всего исполнения машины *3cM′* не выходило за пределы n^2.

$$\mathbf{G}\,\mathbf{F}\,(q = 8).$$

Машина *3cM′* из любого управляющего состояния (в том числе и из начального q_1) всегда рано или поздно переходит в финальное управляющее состояние q_8.

$$\mathbf{G}\,(\ q = 8 \ \Rightarrow \ \mathbf{X}(q = 8)\ ).$$

Если счётчиковая машина *3cM′* попадает в финальное управляющее состояние q_8, то она навсегда остаётся в этом состоянии.

$$\mathbf{G}\,(\ q \geqslant 1 \wedge q \leqslant 8\ ).$$

Управляющее состояние q всегда принимает значения из диапазона от 1 до 8.

Теперь рассмотрим LTL-свойство, которое нарушается на некотором исполнении машины *3cM′*.

$$(a = n \wedge b = 0 \wedge c = 0 \wedge n > 0 \wedge \mathbf{G}(n = \_n)) \Rightarrow \mathbf{G}(\ q = 8 \ \Rightarrow \ \neg(c = 2n)\ ).$$

Это свойство утверждает, что в каждом возможном исполнении машины *3cM′* из определённых начальных состояний итоговый результат её работы в управляющем состоянии q_8 отличен от $c = 2n$ при $n > 0$. Контрпример соответствует исполнению машины при $n = 2$.

Заметим, что при спецификации арифметических свойств машины *3cM′*, вычисляющей значение n^2, потребовалось ввести входную переменную n и соответствующую ей вспомогательную переменную $\_n$, которых не было в правилах переходов машины *3cM′*. Для проверки справедливости свойств такого рода необходимо расширить псевдополную систему переходов *pLTS* с помощью введения в её состояния значения этих переменных n и $\_n$. Имитируя входное число, подлежащее возведению в квадрат, переменная n должна сохранять своё значение на протяжении всего процесса вычисления. Это достигается с помощью специальной подформулы $\mathbf{G}(n = \_n)$, включённой в сами свойства. Выражение $\mathbf{G}(n = \_n)$ требует, чтобы текущее значение переменной n всегда было равно своему предыдущему значению, т. е. не изменялось.

7. Разработка и верификация LTL-спецификации программы ПЛК

Реализуем реагирующую систему в виде программы ПЛК на базе счётчиковой машины возведения числа в квадрат *3cM′*. Мотивация данного примера — продемонстрировать некоторое абстрактное поведение автомата управления без привязки к реальному оборудованию и технологическому процессу. Технологическим процессом здесь можно считать процесс вычисления квадрата числа: входное число n выступает в роли исходной заготовки, а n^2 является готовым изделием. Схема системы представлена на рис. 7. ПЛК реагирует на входные сигналы от кнопок (*Buttons*), выдавая выходные сигналы на индикаторы (*Indicators*). Управляющий автомат (*Control automaton*) с состоянием q является системой управления процессом вычисления квадрата числа. Объектом управления является набор из четырёх переменных n, a, b, c. Автомат управления оказывает управляющее воздействие на данный набор переменных путем увеличения (*inc*) и уменьшения (*dec*) их значений. Также управляющий автомат может считывать их значения в любом цикле работы ПЛК.

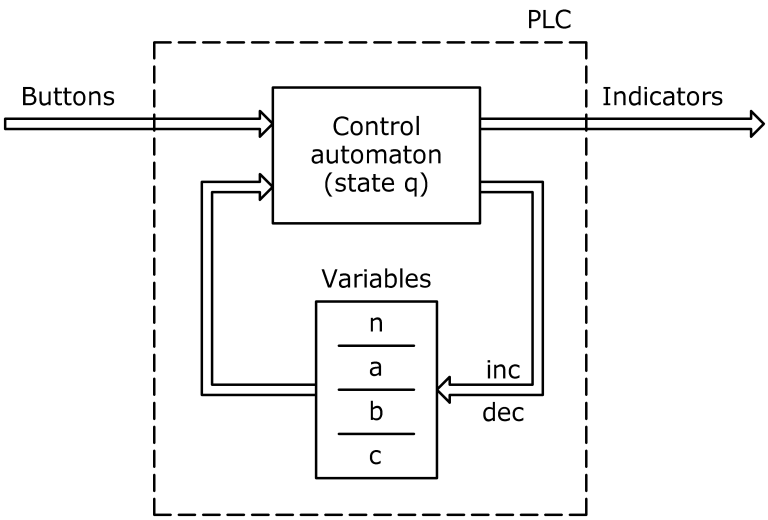

Fig. 7. Reactive system in the form of a PLC program

Рис. 7. Реагирующая система в виде программы ПЛК

Пусть имеется панель управления (см. рис. 8), на которой расположены четыре кнопки «Start», «Reset», «+1» и «-1», девять ламп состояний «q0», . . . , «q8», а также четыре индикатора для отображения значений счётчиков a, b, c и входного числа n. С помощью кнопок «+1» и «-1» происходит выставление числа n, которое будет возводиться в квадрат. Нажатие на кнопку «+1» приводит к увеличению значения n на 1, а нажатие на кнопку «-1» — к уменьшению на 1. Обе кнопки будут срабатывать только тогда, когда управляющий автомат будет находиться в начальном состоянии q_0. Остальные состояния автомата $q_1, \ldots, q_8$ унаследованы от счётчиковой машины *3cM′*. Кнопка «Start» запускает вычислительный процесс для текущего n при условии, что автомат находится в состоянии q_0. При запуске значение переменной n помещается в переменную a, значения b и c равны нулю. Кнопка «Reset» переводит автомат из финального состояния q_8, в которое он попадает после завершения вычисления числа n^2, в стартовое состояние q_0. При этом счётчик c сбрасывается в 0. С помощью ламп «q0», . . . , «q8» отображается текущее состояния управляющего автомата (номер активного состояния хранится в переменной q).

Интерфейс ПЛК представлен на рис. 9. Здесь кнопкам «Start», «Reset», «+1» и «-1» сопоставлены булевы переменные PBStart, PBReset, PBPls и PBMns соответственно. Программные переменные q, a, b, c, n играют роль выходов ПЛК на соответствующие лампы и индикаторы.

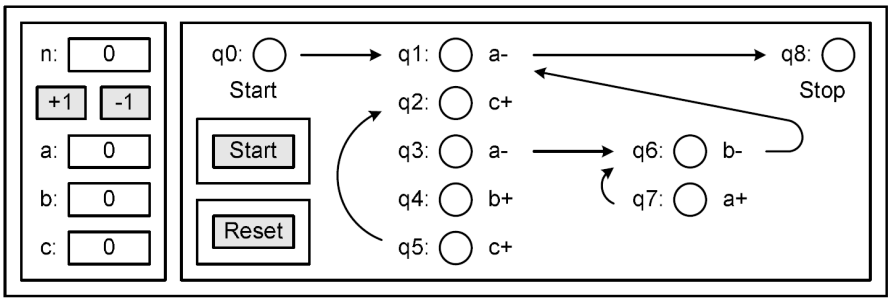

Fig. 8. PLC control panel

Рис. 8. Панель управления ПЛК

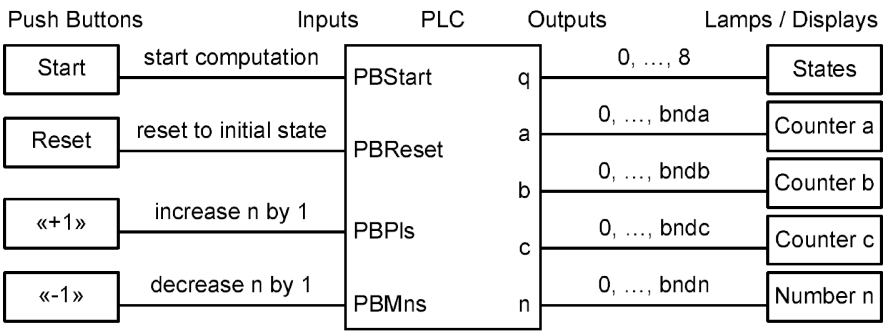

Fig. 9. PLC interface

Рис. 9. Интерфейс ПЛК

Так как верификация выполняется с помощью nuXmv, то спецификацию и её свойства будем записывать на языке этого верификатора. Декларативная LTL-спецификация ПЛК-программы возведения числа в квадрат в синтаксисе nuXmv имеет следующий вид:

```
-- S0:
   q=0 & n=0 & a=0 & b=0 & c=0 & _q=q & _n=n & _a=a & _b=b & _c=c &
   !PBStart & !PBReset & !PBPls & !PBMns & !_PBStart & !_PBReset & !_PBPls & !_PBMns &
-- Sn:
   G X( !(n = _n) ->   _q=0 & _n<bndn  &
                       !_PBPls & PBPls & !PBMns & !PBStart & (n = _n + 1) |
                       _q=0 & _n>0     &
                       !_PBMns & PBMns & !PBPls & !PBStart & (n = _n - 1)  ) &
   G X(  (n = _n) -> !(_q=0 & _n<bndn & !_PBPls & PBPls & !PBMns & !PBStart |
                       _q=0 & _n>0    & !_PBMns & PBMns & !PBPls & !PBStart) ) &
-- Sa:
   G X( !(a = _a) ->   _q=0 & PBStart & !(_a=n) & (a = n)      |
                       _q=7 & _a<bnda           & (a = _a + 1) |
                       _q=1 & _a>0              & (a = _a - 1) |
                       _q=3 & _a>0              & (a = _a - 1) ) &
   G X(  (a = _a) -> !(_q=0 & PBStart & !(_a=n) | _q=7 & _a<bnda |
                       _q=1 & _a>0              | _q=3 & _a>0     ) ) &
-- Sb:
   G X( !(b = _b) ->   _q=4 & _b<bndb & (b = _b + 1) |
                       _q=6 & _b>0    & (b = _b - 1)  ) &
   G X(  (b = _b) -> !(_q=4 & _b<bndb | _q=6 & _b>0)  ) &
-- Sc:
   G X( !(c = _c) ->   _q=2 & _c<bndc        & (c = _c + 1) |
                       _q=5 & _c<bndc        & (c = _c + 1) |
                       _q=8 & _c>0 & PBReset & (c = 0)       ) &
```

```
   G X(  (c = _c) -> !(_q=2 & _c<bndc | _q=5 & _c<bndc | _q=8 & _c>0 & PBReset) ) &
-- Sq:
   G X( !(q = _q) ->   _q=1 &  (_a>0) & (q=2) |
                       _q=1 & !(_a>0) & (q=8) |
                       _q=2           & (q=3) |
                       _q=3 &  (_a>0) & (q=4) |
                       _q=3 & !(_a>0) & (q=6) |
                       _q=4           & (q=5) |
                       _q=5           & (q=2) |
                       _q=6 &  (_b>0) & (q=7) |
                       _q=6 & !(_b>0) & (q=1) |
                       _q=7           & (q=6) |
                       _q=8 & PBReset & (q=0) |
                       _q=0 & PBStart & (q=1)  ) &
   G X(  (q = _q) -> !(_q=1 &  (_a>0) | _q=1 & !(_a>0) | _q=2 | _q=3 & (_a>0) |
                       _q=3 & !(_a>0) | _q=4 | _q=5 | _q=6 & (_b>0) |
                       _q=6 & !(_b>0) | _q=7 | _q=8 & PBReset | _q=0 & PBStart) )
```

На языке верификатора nuXmv символы «&», «|», «!» и «->» означают логические операторы «∧», «∨», «¬» и «⇒» соответственно.

Отметим, что спецификация поведения входных переменных PBStart, PBReset, PBPls и PBMns не производится, так как значения этих переменных могут быть любыми на каждом новом шаге рабочего цикла ПЛК. Ограничения для переменных a, b, c, n задаются с помощью соответствующих констант bnd$a$, bnd$b$, bnd$c$, bnd$n$.

Как можно видеть, спецификация поведения программы записана в виде одной большой LTL-формулы. Обозначим её буквой $\varphi = \mathrm{S0} \wedge \mathrm{S}n \wedge \mathrm{S}a \wedge \mathrm{S}b \wedge \mathrm{S}c \wedge \mathrm{S}q$.

Зададим в синтаксисе nuXmv спецификацию свойств ПЛК-программы возведения числа в квадрат в виде набора LTL-формул P1, . . . , P7:

```
G( q=8 -> c=n*n & a=0 & b=0 )                                    -- P1;
G( a+b <= n )                                                    -- P2;
G( c <= n*n )                                                    -- P3;
G( !(q=0) -> F(q=8) )                                            -- P4;
G( q=0 & X(PBStart) -> F(q=8) )                                  -- P5;
G( q=8 -> X(q=8 | PBReset & q=0 & a=0 & b=0 & c=0) )             -- P6;
G( ((q=2 | q=5) -> c<bndc) & (q=4 -> b<bndb) & (q=7 -> a<bnda) )  -- P7.
```

Свойство P1 утверждает, что всегда в заключительном состоянии $q = 8$ результат вычисления n^2 находится в переменной c, а переменные a и b равны нулю. Свойство P2 — всегда сумма a и b не превышает n. Свойство P3 — значение переменной c в процессе вычисления никогда не превышает n^2. Свойство P4 — если процесс вычисления начался (автомат не находится в начальном состоянии $q = 0$), то он обязательно закончится (в будущем автомат перейдёт в финальное состояние $q = 8$). Свойство P5 — если в начальном состоянии $q = 0$ нажата кнопка «Start», то процесс вычисления будет запущен и завершится в своём финальном состоянии $q = 8$. Свойство P6 — если процесс вычисления достиг финального состояния $q = 8$, то он либо продолжает оставаться в данном состоянии, либо по нажатию кнопки «Reset» переходит в начальное состояние $q = 0$ при $a = 0, b = 0, c = 0$. Свойство P7 проверяет ограничение переменных: в состоянии $q = 2$ и $q = 5$ значение переменной c меньше bnd$c$, в состоянии $q = 4$ значение переменной b меньше bnd$b$, в состоянии $q = 7$ значение переменной a меньше bnd$a$.

Каждое свойство $p \in \{\mathrm{P1}, \ldots, \mathrm{P7}\}$ проверяется по отдельности — выполняется проверка выполнимости $pLTS \vDash (\varphi \Rightarrow p)$, где φ — спецификация поведения программы возведения числа в квадрат.

Опишем псевдополную систему переходов *pLTS* и проверяемую формулу в синтаксисе nuXmv:

```
MODULE main
VAR
    q : 0..8;    _q : 0..8;                  -- State q
    n : 0..15;   _n : 0..15;                 -- Number n
    a : 0..15;   _a : 0..15;                 -- Counter a
    b : 0..15;   _b : 0..15;                 -- Counter b
    c : 0..255;  _c : 0..255;                -- Counter c
    PBStart : boolean;  _PBStart : boolean;  -- Push Button "Start"
    PBReset : boolean;  _PBReset : boolean;  -- Push Button "Reset"
    PBPls   : boolean;  _PBPls   : boolean;  -- Push Button "+1"
    PBMns   : boolean;  _PBMns   : boolean;  -- Push Button "-1"
DEFINE
    bndn := 15; bnda := 15; bndb := 15; bndc := 255;  -- Bounds
ASSIGN -- pLTS
    next(_q) := q;  next(_n) := n;  next(_a) := a;  next(_b) := b;  next(_c) := c;
    next(_PBStart) := PBStart;  next(_PBPls) := PBPls;
    next(_PBReset) := PBReset;  next(_PBMns) := PBMns;
LTLSPEC (Spec -> Prop)
```

В последней строке после ключевого слова LTLSPEC приводится формула, для которой необходимо провести проверку на выполнимость относительно псевдополной системы переходов *pLTS*. В этой строке Spec — это LTL-спецификация φ программы возведения числа в квадрат, а Prop — любое свойство из P1, . . . , P7.

Проверка выполнимости свойств P1, . . . , P7 проводилась на персональном компьютере с процессором Intel Core i5-3570 3.4 ГГц и 8 ГБ оперативной памяти. Свойства P4 и P5 проверялись около двух минут, остальные — несколько секунд.

8. Построение программы ПЛК по LTL-спецификации

По декларативной LTL-спецификации φ из раздела 7 построим код программы возведения числа в квадрат на языке ST. Для этого потребуется перевести декларативную LTL-спецификацию φ в императивную. В свою очередь императивная LTL-спецификация (5), (6) для переменной v из множества $\{v_1, \ldots, v_n\}$ имеет следующую схему построения ST-кода:

IF $cond_1$ **THEN** $v := expr_1$;
ELSIF $cond_2$ **THEN** $v := expr_2$;
. . .
ELSIF $cond_k$ **THEN** $v := expr_k$;
END_IF;

ST-программа, построенная в среде CoDeSys (https://codesys.com) на основе этой схемы, имеет вид:

```
PROGRAM PLC_PRG
VAR_INPUT
    PBStart, PBReset, PBPls, PBMns : BOOL := FALSE;
END_VAR
VAR_OUTPUT
    q, n, a, b, c : BYTE := 0;
END_VAR
```

```
VAR
    _q, _n, _a, _b, _c : BYTE := 0;
    _PBStart, _PBReset, _PBPls, _PBMns : BOOL := FALSE;
END_VAR
VAR CONSTANT
    bndn, bnda, bndb : BYTE := 15;
    bndc : BYTE := 255;
END_VAR
IF    _q=0 AND _n<bndn AND PBPls AND NOT _PBPls AND NOT PBMns AND NOT PBStart THEN n:=_n+1;
ELSIF _q=0 AND _n>0    AND PBMns AND NOT _PBMns AND NOT PBPls AND NOT PBStart THEN n:=_n-1;
END_IF;
IF    _q=0 AND PBStart AND NOT(_a=n) THEN a:=n;
ELSIF _q=7 AND _a<bnda               THEN a:=_a+1;
ELSIF _q=1 AND _a>0                  THEN a:=_a-1;
ELSIF _q=3 AND _a>0                  THEN a:=_a-1;
END_IF;
IF    _q=4 AND _b<bndb          THEN b:=_b+1;
ELSIF _q=6 AND _b>0             THEN b:=_b-1;
END_IF;
IF    _q=2 AND _c<bndc          THEN c:=_c+1;
ELSIF _q=5 AND _c<bndc          THEN c:=_c+1;
ELSIF _q=8 AND _c>0 AND PBReset THEN c:=0;
END_IF;
IF    _q=1 AND     _a>0  THEN q:=2;
ELSIF _q=1 AND NOT(_a>0) THEN q:=8;
ELSIF _q=2               THEN q:=3;
ELSIF _q=3 AND     _a>0  THEN q:=4;
ELSIF _q=3 AND NOT(_a>0) THEN q:=6;
ELSIF _q=4               THEN q:=5;
ELSIF _q=5               THEN q:=2;
ELSIF _q=6 AND     _b>0  THEN q:=7;
ELSIF _q=6 AND NOT(_b>0) THEN q:=1;
ELSIF _q=7               THEN q:=6;
ELSIF _q=8 AND PBReset   THEN q:=0;
ELSIF _q=0 AND PBStart   THEN q:=1;
END_IF;
(* Pseudo operator section: saving previous values of variables *)
_q:=q; _a:=a; _b:=b; _c:=c; _n:=n;
_PBStart:=PBStart; _PBReset:=PBReset; _PBPls:=PBPls; _PBMns:=PBMns;
```

При генерации программного кода по LTL-спецификации порядок размещения блоков IF-ELSIF должен подчиняться следующему правилу: некоторая переменная без псевдооператора «_» может быть задействована в IF-ELSIF-блоке другой переменной только в том случае, если её IF-ELSIF-блок уже находится выше по тексту.

Например, в представленной программе IF-ELSIF-блок переменной *n* должен идти раньше IF-ELSIF-блока переменной *a*, поскольку в правилах формирования нового значения *a* переменная *n* участвует без применения псевдооператора «_», т. е. новое значение *a* формируется на основе нового значения *n*.

При построении программы ПЛК каждая переменная из LTL-спецификации должна быть определена в подходящем разделе описания переменных («входы», «выходы», «локальные переменные») и проинициализирована в соответствии со спецификацией, а конкретнее — в соответствии с LTL-формулой инициализации вида S0. Входы ПЛК описываются в разделе VAR_INPUT, выходы —

в разделе VAR_OUTPUT, остальные переменные — в разделе VAR. Инициализация осуществляется в разделах объявления переменных.

Кроме того, для псевдополной системы переходов необходимо реализовать идею псевдооператора лидирующего подчеркивания «_». Для этого в самом конце ПЛК-программы выделяется место для псевдооператорного раздела, куда добавляются присваивания $\_v_i := v_i$, где $i = 1, \ldots, n$. При этом $\_v_i$ также необходимо определить в разделе описания переменных с таким же начальным значением, что и для переменной v_i.

В рамках модели поведения *LTS* переход из одного состояния в другое будет соответствовать выполнению программы ПЛК вплоть до псевдооператорного раздела, т. е. состояние будет представлять собой вектор значений всех программных переменных, который был получен до выполнения присваиваний, реализующих псевдооператор «_». Начальное состояние — состояние программы ПЛК после инициализации. Построение нового состояния ПЛК-программы происходит в следующем порядке: 1) выполняется псевдооператорный раздел (кроме первого раза), 2) входные переменные (входы ПЛК) получают новые значения, 3) программа ПЛК выполняется с самого начала до псевдооператорного раздела. При этом поведение ST-программы гарантированно соответствует исходной декларативной LTL-спецификации.

Заключение

В работе представлена декларативная LTL-спецификация, которую предлагается использовать для описания поведения управляющих программ. Данная спецификация позволяет описывать причины и правила изменения значений программных переменных. Предложенный способ декларирования поведения представляется естественным и удобным для управляющих систем.

В качестве модели программы используется размеченная система переходов *LTS*. Псевдополная система переходов *pLTS* содержит в себе поведение всех программ с заданным набором переменных. Декларативная LTL-спецификация формирует модель поведения программы *LTS* путем отбора из псевдополной системы переходов *pLTS* только необходимых путей.

Декларативная LTL-спецификация программы может быть непосредственно верифицирована методом проверки модели с помощью инструмента nuXmv. Для этого необходимо задать nuXmv-модель псевдополной системы переходов. Задача верификации для бесконечных моделей в общем случае неразрешима. Для разрешимости предлагается ограничить модель — сделать множество состояний конечным. В этом случае система переходов становится структурой Крипке.

Декларативная LTL-спецификация является конструктивной в том смысле, что по ней может быть построена управляющая программа. С этой целью в работе представлена вспомогательная императивная LTL-спецификация, для которой затем доказывается её эквивалентность декларативной LTL-спецификации. Императивная LTL-спецификация описывает поведение программы в императивном стиле, соответствующем, например, стилю языка программирования ST. Благодаря этому, становится возможной непосредственная трансляция императивной LTL-спецификации в ST-программу. Подход к трансляции описывается схематично в виде шаблона. Приведённая схема трансляции обеспечивает полное соответствие поведения полученной ST-программы исходной декларативной LTL-спецификации.

Декларативная LTL-спецификация является полной по Тьюрингу. Для доказательства этого факта применяется приём, позволяющий построить декларативную LTL-спецификацию для произвольной счётчиковой машины Минского, представляющей собой формализм, равномощный машине Тьюринга. Полнота по Тьюрингу даёт гарантию возможности описания абсолютно любого вычислительного алгоритма с помощью декларативной LTL-спецификации. Способ построения LTL-спецификации демонстрируется на примере счётчиковой машины возведения числа в квадрат.

Полученные в работе результаты подтверждают теоретическую состоятельность подхода к разработке и верификации управляющих программ на основе LTL-спецификации.

References

[1] S. Oks, M. Jalowski, M. Lechner, *et al.*, "Cyber-Physical Systems in the Context of Industry 4.0: A Review, Categorization and Outlook", *Inf. Systems Frontiers*, 2022. DOI: 10.1007/s10796-022-10252-x.

[2] E. Lee and S. Seshia, *Introduction to Embedded Systems – A Cyber-Physical Systems Approach*, 2nd ed. MIT Press, 2017, 561 pp.

[3] E. A. Parr, *Programmable Controllers. An Engineer's Guide*, 3rd ed. Newnes, 2003, 448 pp.

[4] V. Lifshic and L. Sadovskii, "Algebraic Models of Computing Machines", *UMN*, vol. 27, no. 3(165), pp. 79–125, 1972, in Russian.

[5] K.-Y. Cai, T. Chen, and T. Tse, "Towards Research on Software Cybernetics", in *Proceedings of 7th IEEE International on High-assurance Systems Engineering (HASE 2002)*, IEEE Computer Society Press, 2002, pp. 240–241.

[6] N. Polikarpova and A. Shalyto, *Automata-Based Programming*, 2nd ed. Spb.: Piter, 2011, 176 pp., in Russian.

[7] D. Harel and A. Pnueli, "On the Development of Reactive Systems", in *Logics and Models of Concurrent Systems*, vol. 13, 1985, pp. 477–498.

[8] A. Pnueli, "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends", in *Current Trends in Concurrency*, vol. 224, Springer, 1986, pp. 510–584.

[9] A. Maurya and D. Kumar, "Reliability of safety-critical systems: A state-of-the-art review", *Quality and Reliability Engineering International*, vol. 36, no. 7, pp. 2547–2568, 2020.

[10] N. Rajabli, F. Flammini, R. Nardone, and V. Vittorini, "Software Verification and Validation of Safe Autonomous Cars: A Systematic Literature Review", in *IEEE Access*, vol. 9, 2021, pp. 4797–4819.

[11] V. D'Silva, D. Kroening, and G. Weissenbacher, "A Survey of Automated Techniques for Formal Software Verification", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, 2008, pp. 1165–1178.

[12] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*, 1st ed. Springer Publishing Company, Incorporated, 2018, 1212 pp.

[13] Y. G. Karpov, *Model checking. Verification of Parallel and Distributed Program Systems*. BHV-Peterburg, 2010, 560 pp., in Russian.

[14] S. Velder, M. Lukin, A. Shalyto, and B. Yaminov, *Verification of Automata-Based Programs*. Spb Science, 2011, 244 pp., in Russian.

[15] E. M. Clarke, O. Grumberg, and D. Peled, *Verification of Program Models: Model Checking*. MCNMO, 2002, 416 pp., Transl. from English to Russian.

[16] E. Kuzmin and V. Sokolov, "Modeling, Specification and Construction of PLC-programs", *Modeling and Analysis of Information Systems*, vol. 20, no. 2, pp. 104–120, 2013, in Russian.

[17] E. Kuzmin, D. Ryabukhin, and V. Sokolov, "On the Expressiveness of the Approach to Constructing PLC-programs by LTL-Specification", *Modeling and Analysis of Information Systems*, vol. 22, no. 4, pp. 507–520, 2015, in Russian.

[18] E. Kuzmin, D. Ryabukhin, and V. Sokolov, "On the Expressiveness of the Approach to Constructing PLC-programs by LTL-Specification", *Automatic Control and Computer Sciences*, vol. 50, pp. 510–519, 2016.

[19] D. Ryabukhin, E. Kuzmin, and V. Sokolov, "Construction of CFC-programs by LTL-specification", *Modeling and Analysis of Information Systems*, vol. 23, no. 2, pp. 173–184, 2016, in Russian.

[20] D. Ryabukhin, E. Kuzmin, and V. Sokolov, “Construction of CFC-Programs by LTL-Specification”, *Automatic Control and Computer Sciences*, vol. 51, pp. 567–575, 2017.

[21] E. Kuzmin, “LTL-Specification of Counter Machines”, *Modeling and Analysis of Information Systems*, vol. 28, no. 1, pp. 104–119, 2021, in Russian.

[22] E. Kuzmin, “LTL-Specification of Bounded Counter Machines”, *Modeling and Analysis of Information Systems*, vol. 29, no. 1, pp. 44–59, 2022, in Russian.

[23] E. Kuzmin and V. Sokolov, “On Construction and Verification of PLC-Programs”, *Modeling and Analysis of Information Systems*, vol. 19, no. 4, pp. 25–36, 2012, in Russian.

[24] E. Kuzmin and V. Sokolov, “On Construction and Verification of PLC Programs”, *Automatic Control and Computer Sciences*, vol. 47, no. 7, pp. 443–451, 2013.

[25] E. Kuzmin and V. Sokolov, “Modeling, Specification and Construction of PLC-programs”, *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 554–563, 2014.

[26] E. Kuzmin, V. Sokolov, and D. Ryabukhin, “Construction and Verification of PLC-programs by LTL-specification”, *Modeling and Analysis of Information Systems*, vol. 20, no. 4, pp. 5–22, 2013, in Russian.

[27] E. Kuzmin, V. Sokolov, and D. Ryabukhin, “Construction and Verification of PLC-Programs by LTL-Specification”, *Automatic Control and Computer Sciences*, vol. 49, no. 7, pp. 453–465, 2015.

[28] E. Kuzmin, V. Sokolov, and D. Ryabukhin, “Construction and Verification of PLC LD-programs by LTL-specification”, *Modeling and Analysis of Information Systems*, vol. 20, no. 6, pp. 78–94, 2013, in Russian.

[29] E. Kuzmin, V. Sokolov, and D. Ryabukhin, “Construction and Verification of PLC LD Programs by the LTL Specification”, *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 424–436, 2014.

[30] D. Ryabukhin, E. Kuzmin, and V. Sokolov, “Construction of PLC IL-programs by LTL-specification”, *Modeling and Analysis of Information Systems*, vol. 21, no. 2, pp. 26–38, 2014, in Russian.

[31] E. Kuzmin, D. Ryabukhin, and V. Sokolov, “Modeling a Consistent Behavior of PLC-Sensors”, *Modeling and Analysis of Information Systems*, vol. 21, no. 4, pp. 75–90, 2014, in Russian.

[32] E. Kuzmin, D. Ryabukhin, and V. Sokolov, “Modeling a Consistent Behavior of PLC-Sensors”, *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 602–614, 2014.

[33] *IEC 61131-3:2013 Programmable controllers – Part 3: Programming languages*, International Standard. [Online]. Available: https://webstore.iec.ch/publication/4552.

[34] A. Pnueli, “The Temporal Logic of Programs”, in *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, IEEE Computer Society Press, 1977, pp. 46–57.

[35] M. Minsky, *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967, 317 pp.

[36] R. Schroeppel, “A Two Counter Machine Cannot Calculate 2^N”, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Artificial Intelligence Memo #257, 1972, 32 pp.

[37] E. V. Kuzmin, *Counter Machines*. Yaroslavl: Yaroslavl State University, 2010, 127 pp., in Russian.

Приложение. Построение LTL-спецификации счётчиковой машины 3cM

Продемонстрируем процедуру построения LTL-спецификации из теоремы 2 на примере счётчиковой машины *3cM* возведения числа в квадрат (раздел 5).

Выполним LTL-формализацию правил (15):

$$
\begin{aligned}
&(\delta_1)\ \mathbf{GX}\Big(\big[(\_q = 1) \wedge (\_a > 0) \Rightarrow (a = \_a - 1) \wedge (q = 2)\big] \wedge \big[(\_q = 1) \wedge \neg(\_a > 0) \Rightarrow (q = 8)\big]\Big),\\
&(\delta_2)\ \mathbf{GX}\big((\_q = 2) \Rightarrow (c = \_c + 1) \wedge (q = 3)\big),\\
&(\delta_3)\ \mathbf{GX}\Big(\big[(\_q = 3) \wedge (\_a > 0) \Rightarrow (a = \_a - 1) \wedge (q = 4)\big] \wedge \big[(\_q = 3) \wedge \neg(\_a > 0) \Rightarrow (q = 6)\big]\Big),\\
&(\delta_4)\ \mathbf{GX}\big((\_q = 4) \Rightarrow (b = \_b + 1) \wedge (q = 5)\big),\\
&(\delta_5)\ \mathbf{GX}\big((\_q = 5) \Rightarrow (c = \_c + 1) \wedge (q = 2)\big),\\
&(\delta_6)\ \mathbf{GX}\Big(\big[(\_q = 6) \wedge (\_b > 0) \Rightarrow (b = \_b - 1) \wedge (q = 7)\big] \wedge \big[(\_q = 6) \wedge \neg(\_b > 0) \Rightarrow (q = 1)\big]\Big),\\
&(\delta_7)\ \mathbf{GX}\big((\_q = 7) \Rightarrow (a = \_a + 1) \wedge (q = 6)\big).
\end{aligned}
\tag{29}
$$

Осуществим фрагментацию формул (29) (результат представлен в левой колонке таблицы 1), и сгруппируем эти формулы по переменным (результат приведён в правой колонке таблицы 1).

Table 1. Fragmented (left) and grouped by variables (right) LTL-formulas

(δ_1)	$\mathbf{GX}\big((\_q = 1) \wedge (\_a > 0) \Rightarrow (a = \_a - 1)\big)$,
	$\mathbf{GX}\big((\_q = 1) \wedge (\_a > 0) \Rightarrow (q = 2)\big)$,
	$\mathbf{GX}\big((\_q = 1) \wedge \neg(\_a > 0) \Rightarrow (q = 8)\big)$,
(δ_2)	$\mathbf{GX}\big((\_q = 2) \Rightarrow (c = \_c + 1)\big)$,
	$\mathbf{GX}\big((\_q = 2) \Rightarrow (q = 3)\big)$,
(δ_3)	$\mathbf{GX}\big((\_q = 3) \wedge (\_a > 0) \Rightarrow (a = \_a - 1)\big)$,
	$\mathbf{GX}\big((\_q = 3) \wedge (\_a > 0) \Rightarrow (q = 4)\big)$,
	$\mathbf{GX}\big((\_q = 3) \wedge \neg(\_a > 0) \Rightarrow (q = 6)\big)$,
(δ_4)	$\mathbf{GX}\big((\_q = 4) \Rightarrow (b = \_b + 1)\big)$,
	$\mathbf{GX}\big((\_q = 4) \Rightarrow (q = 5)\big)$,
(δ_5)	$\mathbf{GX}\big((\_q = 5) \Rightarrow (c = \_c + 1)\big)$,
	$\mathbf{GX}\big((\_q = 5) \Rightarrow (q = 2)\big)$,
(δ_6)	$\mathbf{GX}\big((\_q = 6) \wedge (\_b > 0) \Rightarrow (b = \_b - 1)\big)$,
	$\mathbf{GX}\big((\_q = 6) \wedge (\_b > 0) \Rightarrow (q = 7)\big)$,
	$\mathbf{GX}\big((\_q = 6) \wedge \neg(\_b > 0) \Rightarrow (q = 1)\big)$,
(δ_7)	$\mathbf{GX}\big((\_q = 7) \Rightarrow (a = \_a + 1)\big)$,
	$\mathbf{GX}\big((\_q = 7) \Rightarrow (q = 6)\big)$.

Таблица 1. Фрагментированные (слева) и сгруппированные по переменным (справа) LTL-формулы

(a)	$\mathbf{GX}\big((\_q = 1) \wedge (\_a > 0) \Rightarrow (a = \_a - 1)\big)$,
	$\mathbf{GX}\big((\_q = 3) \wedge (\_a > 0) \Rightarrow (a = \_a - 1)\big)$,
	$\mathbf{GX}\big((\_q = 7) \Rightarrow (a = \_a + 1)\big)$,
(b)	$\mathbf{GX}\big((\_q = 4) \Rightarrow (b = \_b + 1)\big)$,
	$\mathbf{GX}\big((\_q = 6) \wedge (\_b > 0) \Rightarrow (b = \_b - 1)\big)$,
(c)	$\mathbf{GX}\big((\_q = 2) \Rightarrow (c = \_c + 1)\big)$,
	$\mathbf{GX}\big((\_q = 5) \Rightarrow (c = \_c + 1)\big)$,
(q)	$\mathbf{GX}\big((\_q = 1) \wedge (\_a > 0) \Rightarrow (q = 2)\big)$,
	$\mathbf{GX}\big((\_q = 1) \wedge \neg(\_a > 0) \Rightarrow (q = 8)\big)$,
	$\mathbf{GX}\big((\_q = 2) \Rightarrow (q = 3)\big)$,
	$\mathbf{GX}\big((\_q = 3) \wedge (\_a > 0) \Rightarrow (q = 4)\big)$,
	$\mathbf{GX}\big((\_q = 3) \wedge \neg(\_a > 0) \Rightarrow (q = 6)\big)$,
	$\mathbf{GX}\big((\_q = 4) \Rightarrow (q = 5)\big)$,
	$\mathbf{GX}\big((\_q = 5) \Rightarrow (q = 2)\big)$,
	$\mathbf{GX}\big((\_q = 6) \wedge (\_b > 0) \Rightarrow (q = 7)\big)$,
	$\mathbf{GX}\big((\_q = 6) \wedge \neg(\_b > 0) \Rightarrow (q = 1)\big)$,
	$\mathbf{GX}\big((\_q = 7) \Rightarrow (q = 6)\big)$.

Конъюнктивно объединим формулы в каждой группе из правой колонки таблицы 1, а также добавим формулы, описывающие отсутствие изменений значения переменной. Получим императивную LTL-спецификацию:

$$
\begin{aligned}
(a)\ & \mathbf{GX}\big((\_q = 1) \wedge (\_a > 0) \Rightarrow (a = \_a - 1)\big) \wedge \\
& \mathbf{GX}\big((\_q = 3) \wedge (\_a > 0) \Rightarrow (a = \_a - 1)\big) \wedge \\
& \mathbf{GX}\big((\_q = 7) \Rightarrow (a = \_a + 1)\big), \\
& \mathbf{GX}\Big(\neg\big((\_q = 1) \wedge (\_a > 0)\big) \wedge \neg\big((\_q = 3) \wedge (\_a > 0)\big) \wedge \neg\big((\_q = 7)\big) \Rightarrow (a = \_a)\Big), \\
(b)\ & \mathbf{GX}\big((\_q = 4) \Rightarrow (b = \_b + 1)\big) \wedge \\
& \mathbf{GX}\big((\_q = 6) \wedge (\_b > 0) \Rightarrow (b = \_b - 1)\big), \\
& \mathbf{GX}\Big(\neg\big((\_q = 4)\big) \wedge \neg\big((\_q = 6) \wedge (\_b > 0)\big) \Rightarrow (b = \_b)\Big), \\
(c)\ & \mathbf{GX}\big((\_q = 2) \Rightarrow (c = \_c + 1)\big) \wedge \\
& \mathbf{GX}\big((\_q = 5) \Rightarrow (c = \_c + 1)\big), \\
& \mathbf{GX}\Big(\neg\big((\_q = 2)\big) \wedge \neg\big((\_q = 5)\big) \Rightarrow (c = \_c)\Big), \\
(q)\ & \mathbf{GX}\big((\_q = 1) \wedge (\_a > 0) \Rightarrow (q = 2)\big) \wedge \\
& \mathbf{GX}\big((\_q = 1) \wedge \neg(\_a > 0) \Rightarrow (q = 8)\big) \wedge \\
& \mathbf{GX}\big((\_q = 2) \Rightarrow (q = 3)\big) \wedge \\
& \mathbf{GX}\big((\_q = 3) \wedge (\_a > 0) \Rightarrow (q = 4)\big) \wedge \\
& \mathbf{GX}\big((\_q = 3) \wedge \neg(\_a > 0) \Rightarrow (q = 6)\big) \wedge \\
& \mathbf{GX}\big((\_q = 4) \Rightarrow (q = 5)\big) \wedge \\
& \mathbf{GX}\big((\_q = 5) \Rightarrow (q = 2)\big) \wedge \\
& \mathbf{GX}\big((\_q = 6) \wedge (\_b > 0) \Rightarrow (q = 7)\big) \wedge \\
& \mathbf{GX}\big((\_q = 6) \wedge \neg(\_b > 0) \Rightarrow (q = 1)\big) \wedge \\
& \mathbf{GX}\big((\_q = 7) \Rightarrow (q = 6)\big), \\
& \mathbf{GX}\Big(\neg\big((\_q = 1) \wedge (\_a > 0)\big) \wedge \neg\big((\_q = 1) \wedge \neg(\_a > 0)\big) \wedge \neg\big((\_q = 2)\big) \wedge \\
& \qquad \neg\big((\_q = 3) \wedge (\_a > 0)\big) \wedge \neg\big((\_q = 3) \wedge \neg(\_a > 0)\big) \wedge \neg\big((\_q = 4)\big) \wedge \\
& \qquad \neg\big((\_q = 5)\big) \wedge \neg\big((\_q = 6) \wedge (\_b > 0)\big) \wedge \neg\big((\_q = 6) \wedge \neg(\_b > 0)\big) \wedge \\
& \qquad \neg\big((\_q = 7)\big) \Rightarrow (q = \_q)\Big).
\end{aligned}
$$

Далее по этой императивной LTL-спецификации строится декларативная LTL-спецификация счётчиковой машины *3cM* из раздела 5.