MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 31, NO. 2, 2024
journal homepage: www.mais-journal.ru

Verification of Declarative LTL-specification of Control Programs Behavior

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

DOI: 10.18255/1818-1015-2024-2-120-141

[1]Institute of Automation and Electrometry SB RAS, Novosibirsk, Russia
[2]P.G. Demidov Yaroslavl State University, Yaroslavl, Russia

MSC2020: 68N30
Research article
Full text in Russian

Received May 3, 2024
Revised May 17, 2024
Accepted May 22, 2024

The article continues the series of works on development and verification of control programs based on LTL-specifications of a special type. Previously, it was proposed a declarative LTL-specification, which allows describing the behavior of control programs and building program code based on it in the imperative ST-language for programmable logic controllers. The LTL-specification can be directly verified for compliance with specified temporal properties by the model checking method using the nuXmv symbolic verification tool. In general, it is not required translating LTL-formulas of the specification into another formalism — an SMV-specification (code in the input language of the nuXmv tool).

The purpose of this work is to explore alternative ways of representing a program behavior model corresponding to the declarative LTL-specification during its verification within the nuXmv tool.

In the article, we transform the declarative LTL-specification into various SMV-specifications with accompanying changes of formulation of the verification problem, what leads to a significant reduction in time costs when checking temporal properties by using the nuXmv tool. The acceleration of verification is due to the reduction of the state space of a model being verified. The SMV-specifications obtained as a result of the proposed transformations specify identical or bisimulationally equivalent transition systems. It is ensuring the same verification results when replacing one SMV-specification with another.

Keywords: control software; PLC program; declarative LTL-specification; imperative LTL-specification; complete transition system; pseudo-complete transition system; state space; model checking; nuXmv verifier; SMV-specification; bisimulation equivalence; bisimulation

INFORMATION ABOUT THE AUTHORS

Neyzov, Maxim V. (corresponding author) | ORCID iD: 0009-0000-6893-6137. E-mail: neyzov.max@gmail.com
Researcher

Kuzmin, Egor V. | ORCID iD: 0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru
Head of the Chair of Theoretical Informatics, Dr. Sc.

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, "Verification of declarative LTL-specification of control programs behavior", *Modeling and Analysis of Information Systems*, vol. 31, no. 2, pp. 120–141, 2024. DOI: 10.18255/1818-1015-2024-2-120-141.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 31, № 2, 2024
сайт журнала: www.mais-journal.ru

THEORY OF SOFTWARE

Верификация декларативной LTL-спецификации поведения управляющих программ

М. В. Нейзов[1], Е. В. Кузьмин[2]

DOI: 10.18255/1818-1015-2024-2-120-141

[1]Институт автоматики и электрометрии СО РАН, Новосибирск, Россия
[2]Ярославский государственный университет им. П.Г. Демидова, Ярославль, Россия

УДК 004.424+519.683.8
Научная статья
Полный текст на русском языке

Получена 3 мая 2024 г.
После доработки 17 мая 2024 г.
Принята к публикации 22 мая 2024 г.

Статья продолжает цикл трудов по разработке и верификации управляющих программ на основе LTL-спецификаций специального вида. Ранее была предложена декларативная LTL-спецификация, позволяющая описывать поведение управляющих программ и выполнять построение по ней программного кода на императивном языке ST для программируемых логических контроллеров. Данная LTL-спецификация может быть непосредственно верифицирована на предмет соответствия заданным темпоральным свойствам методом проверки модели (model checking) с помощью инструмента символьной верификации nuXmv. При этом не требуется переводить LTL-формулы спецификации в другой формализм — SMV-спецификацию (код на входном языке инструмента nuXmv).

Цель настоящей работы состоит в исследовании альтернативных способов представления модели поведения программы, соответствующей декларативной LTL-спецификации, при её верификации в рамках инструментального средства nuXmv.

В статье выполняются преобразования декларативной LTL-спецификации в различные SMV-спецификации с сопутствующими изменениями постановки задачи верификации, что приводит к значительному снижению временных затрат при проверке темпоральных свойств с использованием инструмента nuXmv. Ускорение верификации обусловлено сокращением пространства состояний проверяемой модели. Полученные в результате предложенных преобразований SMV-спецификации задают одинаковые или бисимуляционно эквивалентные системы переходов, обеспечивая неизменность результатов верификации при замене одной SMV-спецификации на другую.

Ключевые слова: управляющее программное обеспечение; ПЛК-программа; декларативная LTL-спецификация; императивная LTL-спецификация; полная система переходов; псевдополная система переходов; пространство состояний; проверка модели; верификатор nuXmv; SMV-спецификация; бисимуляционная эквивалентность; бисимуляция

ИНФОРМАЦИЯ ОБ АВТОРАХ

Нейзов, Максим Вячеславович (автор для корреспонденции)	ORCID iD: 0009-0000-6893-6137. E-mail: neyzov.max@gmail.com Исследователь
Кузьмин, Егор Владимирович	ORCID iD: 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, “Verification of declarative LTL-specification of control programs behavior”, *Modeling and Analysis of Information Systems*, vol. 31, no. 2, pp. 120–141, 2024. DOI: 10.18255/1818-1015-2024-2-120-141.

Введение

В связи с ростом внедрений ответственных программно-управляемых технических систем [1] повышается актуальность задачи разработки и верификации [2] *управляющего программного обеспечения* (УПО). Работа УПО выполняется циклически. *Цикл управления* состоит в получении информации об управляемом объекте (информация фиксируется во входных переменных), непосредственной работе программы (вычислении новых значений внутренних/выходных переменных) и передаче команд на объект управления. Объектом управления может выступать как программный, так и физический объект. УПО при этом является частью программной или *киберфизической системы* [3, 4] соответственно.

В работе [5] была предложена LTL-спецификация специального вида, предназначенная для описания поведения УПО *трансформационных* и *реактивных систем* [6, 7]. Эта LTL-спецификация является конструктивной в том смысле, что по ней может быть построен программный код УПО на некотором стандартном [8] языке программирования. Предварительно LTL-спецификация может быть непосредственно верифицирована на предмет соответствия заданным *темпоральным* свойствам методом *проверки модели* (model checking) [9—11] с помощью инструмента символьной верификации nuXmv [12]. При этом не требуется переводить LTL-формулы спецификации в другой формализм — SMV-спецификацию (код на входном языке инструмента nuXmv).

Цель настоящей работы — исследовать альтернативные способы представления модели поведения программы, соответствующей предложенной LTL-спецификации, для её верификации в рамках инструментального средства nuXmv. Замена декларативной LTL-спецификации на SMV-спецификацию и изменение постановки задачи верификации могут привести к значительному сокращению времени при проверке темпоральных свойств программы.

Содержание работы. Раздел 1 представляет собой краткое содержание работы [5]. В разделе 2 приведены примеры декларативной и императивной LTL-спецификаций поведения программы возведения числа в квадрат для *программируемого логического контроллера* (ПЛК) [13]. В разделе 3 представлены LTL-свойства и проведена верификация данных LTL-спецификаций согласно схеме, предложенной в работе [5]. Раздел 4 содержит процедуры преобразования LTL-спецификаций в соответствующие декларативную и императивную SMV-спецификации. Проводится их верификация. Раздел 5 содержит дополнительные процедуры преобразования SMV-спецификаций. Доказывается, что данные процедуры сохраняют бисимуляционную эквивалентность между системами переходов, которые задаются этими SMV-спецификациями. Проводится верификация новых SMV-спецификаций. Демонстрируется, что представленные в статье преобразования приводят к снижению времени верификации. В заключительном разделе подводятся итоги.

1. LTL-спецификация поведения управляющих программ

LTL-спецификация предназначена для описания и верификации модели поведения УПО. В работе [5] представлены следующие базовые положения. Программа содержит основные $V = \{v_1, \ldots, v_n\}$ и вспомогательные $\_V = \{\_v_1, \ldots, \_v_n\}$ переменные. Основные переменные из V предназначены для хранения всей необходимой информации для построения программы — это могут быть входные, внутренние и выходные переменные. Вспомогательные переменные из $\_V$ всегда хранят значения соответствующих переменных из V, полученные на предыдущем цикле управления. В LTL-спецификации используются все переменные $V \cup \_V$. Вспомогательные переменные из $\_V$ позволяют детектировать изменения значений основных переменных из V: если значение v_i не равно значению $\_v_i$, где $i = 1, \ldots, n$, то значение переменной v_i изменилось. Также вспомогательные переменные множества $\_V$ могут использоваться для определения значений основных переменных из V. Например, выражение $v_1 = \_v_1 + 1$ утверждает, что значение переменной v_1 увеличивается на единицу.

Пусть переменные $v_1, \ldots, v_n$ и $\_v_1, \ldots, \_v_n$ принимают значения из соответствующих бесконечных множеств $D_1, \ldots, D_n$. Тогда множество $S = (D_1 \times \ldots \times D_n)^2$ является пространством возможных состояний программы. Состояние $s = (\boldsymbol{a}, \_\boldsymbol{a}) \in S$, где $\boldsymbol{a}$ и $\_\boldsymbol{a}$ — значения соответствующих векторов $\mathrm{v} = (v_1, \ldots, v_n)$ и $\_\mathrm{v} = (\_v_1, \ldots, \_v_n)$. Каждый цикл управления приводит к *переходу* между состояниями из S. Если состояние в результате цикла управления не изменилось, то происходит переход в это же самое состояние. Совокупность всех возможных переходов образует поведение программы. Моделью поведения в [5] является *размеченная система переходов* (labelled transition system) $LTS = \langle S, S_0, R, P, L \rangle$, где S — множество состояний, $S_0 \subseteq S$ — множество начальных состояний, $R \subseteq S \times S$ — *тотальное* отношение переходов, $P = \{p_i \mid i \in \mathbb{N}\}$ — множество произвольных атомарных утверждений относительно значений переменных из $V \cup \_V$, $L\colon S \to 2^P$ — функция разметки состояний атомарными утверждениями, истинными в этих состояниях. Свойство тотальности отношения переходов R имеет вид: $(\forall s \in S)\ (\exists s' \in S)\ (s, s') \in R$, т. е. из любого состояния $s \in S$ существует переход из R.

Обозначим S^ω множество всех бесконечных слов в алфавите S. *Путь* — бесконечная последовательность $\pi \in S^\omega$. Система переходов LTS задаёт множество всех возможных в ней путей Π_{LTS} — путей, начинающихся в начальных состояниях:

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

где $\pi(i)$ — i-ое состояние пути π, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

Абсолютно недетерминированной программой назовём программу с недетерминированным поведением всех её переменных из $V \cup \_V$. Моделью поведения такой программы будет *полная* [5] система переходов $cLTS = \langle S, S_0, R_c, P, L \rangle$, где $S_0 = S$, отношение переходов $R_c = S \times S$, т. е. в $cLTS$ возможен переход из любого состояния $s \in S$ в любое состояние $s' \in S$, что соответствует абсолютно недетерминированному поведению.

Наложим на полную систему переходов $cLTS$ ранее оговорённое ограничение на поведение всех её вспомогательных переменных из $\_V$, которое заключается в том, что их значения всегда равны соответствущим значениям переменных из V на предыдущем цикле управления. В итоге получим *псевдополную* [5] систему переходов $pLTS = \langle S, S_0, R'_c, P, L \rangle$, где $S_0 = S$, $R'_c = \{\big((\boldsymbol{a}, \_\boldsymbol{a}), (\boldsymbol{a}', \boldsymbol{a})\big) \in R_c\}$, векторы $(\boldsymbol{a}, \_\boldsymbol{a}) \in S$ и $(\boldsymbol{a}', \boldsymbol{a}) \in S$. Псевдополная система переходов $pLTS$ описывает поведение абсолютно недетерминированной программы, в которой предыдущие значения переменных $v_1, \ldots, v_n$ сохраняются в переменных $\_v_1, \ldots, \_v_n$.

Определим синтаксис и семантику формул линейной темпоральной логики LTL (Linear Temporal Logic). Пусть $p \in P$, тогда с учётом этого формулы LTL имеют следующую грамматику:

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

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

$$\begin{array}{ll}
s_i \models \mathit{true}; & s_i \not\models \mathit{false}; \\
s_i \models p & \Longleftrightarrow p \in L(s_i); \\
s_i \models \neg\, \varphi & \Longleftrightarrow s_i \not\models \varphi; \\
s_i \models \varphi \wedge \psi & \Longleftrightarrow s_i \models \varphi \ \text{ и } \ s_i \models \psi; \\
s_i \models \varphi \vee \psi & \Longleftrightarrow s_i \models \varphi \ \text{ или } \ s_i \models \psi; \\
s_i \models \varphi \Rightarrow \psi & \Longleftrightarrow s_i \models \neg\varphi \ \text{ или } \ s_i \models \psi; \\
s_i \models \mathbf{X}\varphi & \Longleftrightarrow s_{i+1} \models \varphi; \\
s_i \models \psi \mathbf{U} \varphi & \Longleftrightarrow (\exists k \geq i)\ s_k \models \varphi \ \text{ и } \ (\forall j,\ i\ \leq j < k)\ s_j \models \psi;
\end{array}$$

$$s_i \models \mathbf{F}\varphi \iff (\exists k \geq i)\ s_k \models \varphi;$$
$$s_i \models \mathbf{G}\varphi \iff (\forall j \geq i)\ s_j \models \varphi.$$

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

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

В работе [5] LTL-формулы используются для:

1. Задания ограничений для псевдополной системы переходов *pLTS*. Таким образом, на основе *pLTS* формируется модель поведения программы *LTS*.
2. Формализации требуемых свойств модели поведения программы *LTS*.

LTL-спецификация накладывает ограничения на псевдополную систему переходов *pLTS* путём индивидуального ограничения поведения переменных из V. *Декларативная* LTL-спецификация поведения переменной $v \in V$ имеет следующий вид [5]:

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

Данная формула описывает, каким образом происходит изменение значения переменной $v \in V$. Логическое выражение $cond_i$ является необходимым и достаточным условием для изменения значения переменной v согласно выражению $expr_i$, где $i = 1, \ldots, k$. В выражениях $cond_i$ и $expr_i$ могут использоваться любые переменные из $(V \cup \_V) \setminus \{v\}$, константы, логические и арифметические операторы, а также операторы сравнения. Отметим, что в (1) выражения вида $v = \_v$ и $v = expr_i$ являются элементарными высказываниями из множества P, а выражения вида $cond_i$ — пропозициональными формулами (LTL-формулами без темпоральных операторов).

Декларативная LTL-спецификация (1) поведения переменной $v \in V$ должна удовлетворять условию *изменчивости* значения переменной [5]:

$$\mathbf{GX}\big(cond_1 \Rightarrow \neg\,(\_v = expr_1)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow \neg\,(\_v = expr_k)\big), \tag{2}$$

т. е. при истинном условии $cond_i$ выражение $expr_i$ должно возвращать значение, отличное от прошлого значения переменной v. Также для (1) должна выполняться *ортогональность* условий изменения значения переменной для любых $i, j = 1, \ldots, k$ при $i \neq j$ [5]:

$$\mathbf{GX}\big(cond_i \Rightarrow \neg cond_j\big), \tag{3}$$

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

Изначально поведение всех переменных из V недетерминировано — моделью поведения такой программы является псевдополная система переходов *pLTS*. Далее на псевдополную систему переходов *pLTS* накладывается ряд ограничений в виде декларативных LTL-спецификаций поведения переменных из V. Задекларированное поведение переменной становится строго детерминированным. В конечном итоге должно быть задекларировано поведение всех переменных из V, кроме входных. Сценарии работы программы зависят от последовательностей значений входных переменных. Поэтому для сохранения всего многообразия сценариев работы поведение всех входных переменных должно оставаться абсолютно недетерминированным. *Декларативная* LTL-спецификация поведения *неинициализированной* программы φ_{var} — конъюнкция формул вида (1) для переменных из V при соблюдении условий (2) и (3).

LTL-формула φ_0 специального вида $I(v_1, \ldots, v_n) \wedge (\_v_1 = v_1) \wedge \ldots \wedge (\_v_n = v_n)$ без темпоральных операторов используется для инициализации модели программы. Здесь $I(v_1, \ldots, v_n)$ — предикат, ограничивающий значения переменных из V. Начальные значения переменных из $\_V$ всегда равны начальным значениям соответствующих переменных из V. Формула $\varphi = \varphi_{var} \wedge \varphi_0$ является *декларативной* LTL-спецификацией (обозначим *dcl_LTL*) поведения программы.

Если в декларативной LTL-спецификации (1) $cond_i$ или $expr_i$, где $i = 1, \ldots, k$, содержит переменную $v' \in V$, то переменная v непосредственно зависит от переменной v'. Обозначим $(v, v') \in Dep$, где $Dep \subseteq V \times V$, бинарное отношение непосредственной зависимости переменных (Dep от англ. *Dependency*). Обозначим Dep^T транзитивное замыкание отношения Dep:

$$Dep^T = \{(v \in V, v' \in V) \mid \exists\, (v_1, \ldots, v_n)\, [(v_1 = v) \wedge (v_n = v') \wedge \forall i \ \in \{1, \ldots, n-1\}\, (v_i, v_{i+1}) \in Dep]\},$$

т. е. $(v, v') \in Dep^T$ тогда и только тогда, когда v непосредственно зависит от v' или существует цепочка зависимостей между v и v'. Таким образом, $Dep^T \subseteq V \times V$ — бинарное отношение непосредственной и опосредованной зависимости переменных.

При построении программ никакие переменные не должны зависеть сами от себя ни непосредственно, ни опосредованно через другие переменные. Поэтому декларативная LTL-спецификация φ поведения программы должна удовлетворять условию *запрета циклических зависимостей*:

$$\forall v \in V\ [(v, v) \notin Dep^T]. \tag{4}$$

С помощью декларативной LTL-спецификации φ на основе псевдополной системы переходов *pLTS* определяется система переходов *LTS*, которая является моделью поведения программы. Пусть $pLTS = \langle S_{pLTS}, S_0, R_{pLTS}, P, L\rangle$, а $LTS = \langle S, S'_0, R_\varphi, P, L'\rangle$, тогда отношение переходов R_φ имеет вид:

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

где Π_{pLTS} — множество всех путей системы переходов *pLTS*, σ — конечный фрагмент пути, $|\sigma| \in \mathbb{N}_0$. Таким образом, система переходов *LTS* содержит только те пути из *pLTS*, которые удовлетворяют декларативной LTL-спецификации φ. По определению R_φ является тотальным отношением переходов, так как содержит переходы, которые являются частью бесконечной последовательности состояний.

Множество состояний S определяется следующим образом:

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

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

В итоге полученная система переходов *LTS* задаёт то же самое множество путей, что и декларативная LTL-спецификация φ на псевдополной системе переходов: $\Pi_{LTS} = [\![\varphi]\!]_{pLTS}$. Нотация $[\![\varphi]\!]_{LTS}$ означает множество всех путей в *LTS*, на которых истинна формула φ, т. е. $[\![\varphi]\!]_{LTS} = \{\pi \in \Pi_{LTS} \mid \pi \models \varphi\}$.

Императивная LTL-спецификация поведения переменной $v \in V$ [5]:

$$\begin{aligned} &\mathbf{GX}\big(cond_1 \Rightarrow (v = expr_1)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow (v = expr_k)\big) \wedge \\ &\qquad \mathbf{GX}\big(\neg\, cond_1 \wedge \ldots \wedge \neg\, cond_k \Rightarrow (v = \_v)\big). \end{aligned} \tag{5}$$

Данная спецификация описывает поведение переменной в императивном стиле «если…, то…». Императивная LTL-спецификация (обозначим *imp_LTL*) поведения программы является *конструктивной*, т. е. по ней непосредственно может быть построен код программы/модели на императивном языке программирования/моделирования. Декларативная и императивная LTL-спецификации

задают одно и то же поведение — в работе [5] доказана их эквивалентность. Проведена оценка выразительности этих спецификаций — обе спецификации являются Тьюринг-полными [5] при определении логики LTL над бесконечным множеством элементарных высказываний $P = \{p_i \mid i \in \mathbb{N}\}$. Заметим, что Тьюринг-полнота LTL-спецификаций позволяет с их помощью задавать поведение любой машины Тьюринга.

В работе [5] для разрешимости в общем случае задачи проверки модели (model checking) система переходов *cLTS* была ограничена — множества значений переменных $D_1, \ldots, D_n$ имеют конечное число элементов. В этом случае пространство состояний программы S также является конечным. Для соблюдения этого ограничения LTL-спецификация поведения переменной $v_i \in V$ должна удовлетворять условию *ограниченности*:

$$\mathbf{GX}\big(cond_1 \Rightarrow (val(expr_1) \in D_i)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow (val(expr_k) \in D_i)\big), \tag{6}$$

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. Формула инициализации φ_0 также должна задавать начальные значения переменных, попадающие в допустимую область значений.

В данной статье в качестве области значений некоторой переменной рассматривается либо булево множество $\mathbb{B} = \{0, 1\}$, либо конечное подмножество множества натуральных чисел $D \subset \mathbb{N}_0$.

При задании LTL-спецификации φ поведения конкретной программы и свойств $\Psi = \{\psi_1, \ldots, \psi_h\}$ для её проверки из бесконечного множества атомарных высказываний $\{p_i \mid i \in \mathbb{N}\}$ отбирается конечное множество высказываний $P = \{p_1, \ldots, p_m\}$, необходимых для построения φ и Ψ.

2. LTL-спецификация поведения программы возведения числа в квадрат

В работе [5] была разработана декларативная LTL-спецификация поведения ПЛК-программы возведения числа n в квадрат. Спецификация содержит основные $V = \{n, a, b, c, q, PBStart, PBReset, PBPls, PBMns\}$ и вспомогательные $\_V = \{\_n, \_a, \_b, \_c, \_q, \_PBStart, \_PBReset, \_PBPls, \_PBMns\}$ переменные. Переменная n хранит входное значение, переменные a, b, c — результаты промежуточных вычислений, q — управляющее состояние. При завершении работы алгоритма результат содержится в переменной c. Булевы переменные *PBStart, PBReset, PBPls, PBMns* предназначены соответственно для запуска вычисления, перевода алгоритма в начальное управляющее состояние, увеличения и уменьшения на единицу значения переменной n.

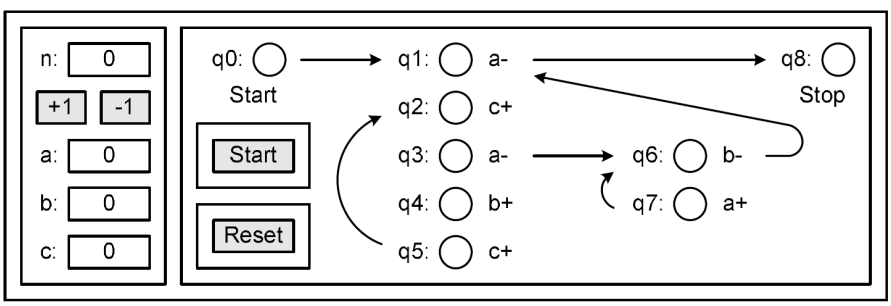


Fig. 1. PLC control panel

Рис. 1. Панель управления ПЛК

К ПЛК подключена панель управления (рис. 1). Интерфейс ПЛК для её подключения представлен на рис. 2. Индикаторы «n», «a», «b», «c» отображают текущие значения одноименных переменных, лампы «q0», ..., «q8» — текущее значение переменной q. Кнопки «Start», «Reset», «+1» и «-1» связаны с булевыми переменными *PBStart, PBReset, PBPls, PBMns* соответственно. Нажатие кнопки

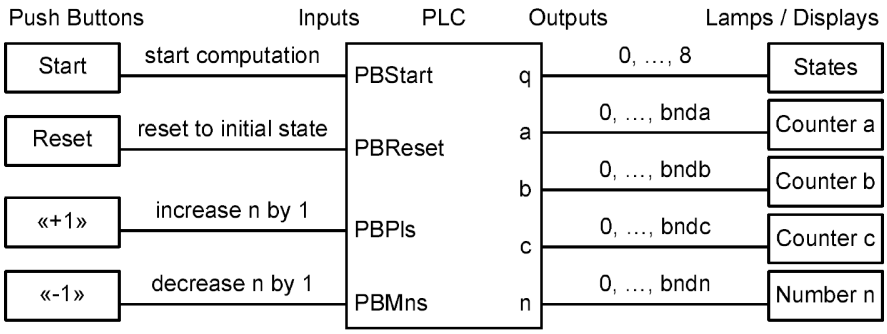


Fig. 2. PLC interface

Рис. 2. Интерфейс ПЛК

выставляет для булевой переменной значение «истина». Спецификация поведения была построена на базе счётчиковой машины, поэтому панель управления также содержит графическое представление её работы. Видно, что $q0$ — начальное управляющее состояние, $q8$ — финальное.

Декларативная LTL-спецификация ПЛК-программы возведения числа в квадрат в синтаксисе инструмента nuXmv представлена в листинге 1, императивная LTL-спецификация — в листинге 2.

Листинг 1 (dcl_LTL-спецификация ПЛК-программы)**:**

```
-- 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) )
```

Листинг 2 (imp_LTL-спецификация ПЛК-программы)**:**

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

Символы «&», «|», «!» и «->» означают логические операторы «$\wedge$», «$\vee$», «$\neg$» и «$\Rightarrow$» соответственно. Здесь спецификация поведения программы — формула $\varphi = \mathrm{S0} \wedge \mathrm{S}n \wedge \mathrm{S}a \wedge \mathrm{S}b \wedge \mathrm{S}c \wedge \mathrm{S}q$.

3. Непосредственная верификация LTL-спецификаций

Декларативная или императивная LTL-спецификация φ может быть непосредственно верифицирована с помощью инструмента символьной проверки модели nuXmv. Для этого в работе [5] постановка задачи верификации имеет следующий вид:

$$pLTS \models (\varphi \Rightarrow \psi). \tag{7}$$

Здесь верификатор проверяет выполнимость импликации $\varphi \Rightarrow \psi$ на псевдополной системе переходов $pLTS$, т. е. проверяется истинность свойства ψ только на тех путях из множества Π_{pLTS}, на которых истинна спецификация поведения φ. Если формула (7) верна, то спецификация поведения φ удовлетворяет свойству ψ, иначе спецификация φ нарушает свойство ψ.

Проведём верификацию декларативной (Листинг 1) и императивной (Листинг 2) LTL-спецификаций ПЛК-программы возведения числа в квадрат согласно (7). Будем проверять набор из девяти LTL-свойств $P1, \ldots, P9$, которые в синтаксисе nuXmv представлены в листинге 3.

Листинг 3 (LTL-свойства ПЛК-программы)**:**

```
G( q=8 -> c=n*n & a=0 & b=0 )                                  -- P1;
G( a+b <= n )                                                  -- P2;
G( c <= n*n )                                                  -- P3;
G( q=8 -> X(q=8 | PBReset & q=0 & a=0 & b=0 & c=0) )           -- P4;
G( ((q=2 | q=5) -> c<bndc) & (q=4 -> b<bndb) & (q=7 -> a<bnda) )  -- P5;
G( !(q=0) -> F(q=8) )                                          -- P6;
G( q=0 & X(PBStart) -> F(q=8) )                                -- P7;
F G(PBReset & PBStart) -> (G F q=0) & (G F q=8)                 -- P8;
(G F PBStart) & (G F PBReset) -> (G F q=8) & (G F q=0)          -- P9.
```

Свойство *P1* утверждает, что всегда в заключительном управляющем состоянии переменная c содержит результат возведения числа n в квадрат, а переменные a и b равны нулю. Свойство *P2* задаёт инвариант — сумма значений переменных a и b никогда не превышает n. Свойство *P3* задаёт другой инвариант — значение переменной c не превышает n^2. Свойство *P4* — после достижения финального состояния его значение больше не изменяется, либо значения всех переменных обнуляются при нажатии кнопки сброса. Свойство *P5* — значения переменных a, b, c ограничены при заданных значениях переменной q. Свойство *P6* утверждает, что всегда если вычисление началось (управляющее состояние q отлично от нуля), то оно гарантированно завершится (управляющее состояние q станет заключительным). Свойство *P7* — если в начальном управляющем состоянии будет нажата кнопка запуска, то вычисление в будущем гарантированно завершится. Свойство *P8* — если когда-то в будущем навсегда зажать кнопки сброса и старта, то бесконечно часто будут происходить вычисления: всегда в будущем будет встречаться начальное ($q = 0$) и финальное ($q = 8$) состояния. Свойство *P9* — если всегда в будущем нажимать на кнопки старта и сброса (по отдельности или вместе), то также бесконечно часто будут происходить вычисления.

Свойства $P1, \ldots, P5$ — свойства *безопасности (Safety)*, $P6, \ldots, P9$ — свойства *живости (Liveness)* [14]. Более точно, *P6*, *P7*, *P8* — свойства *рекуррентности (Recurrence)* [15], *P9* — свойство *реактивности (Reactivity)* [15]. Классификация осуществлялась с помощью программного средства Spot [16].

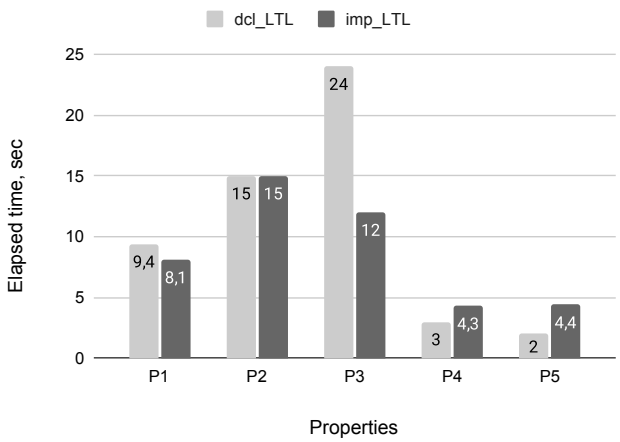
Для непосредственной верификации декларативной/императивной LTL-спецификации φ согласно (7) необходимо задать псевдополную систему переходов $pLTS$ и проверяемое свойство ψ. В синтаксисе nuXmv это было сделано в работе [5]. Результат представлен в листинге 4.

В разделе VAR объявлены основные и вспомогательные переменные. Раздел DEFINE содержит ограничения значений переменных. В разделе ASSIGN задана псевдополная система переходов. Ключевое слово LTLSPEC задаёт проверяемую LTL-формулу. В ней Spec — это dcl_LTL-спецификация φ программы возведения числа в квадрат, а Prop — любое свойство $P1, \ldots, P9$.

Листинг 4 (nuXmv-модель для верификации dcl_LTL-спецификации ПЛК-программы):

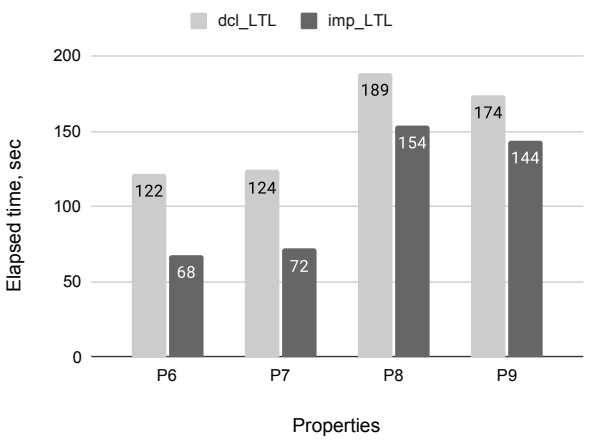
```
MODULE main
VAR
    q : 0..8;    _q : 0..8;                  -- State q
    n : 0..15;   _n : 0..15;                 -- Number n
    a : 0..15;   _a : 0..15;                 -- Counter a
    b : 0..15;   _b : 0..15;                 -- Counter b
    c : 0..255;  _c : 0..255;                -- Counter 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)
```

Свойства $P1, P2, \ldots, P9$ выполняются для обеих LTL-спецификаций. Верификация проводилась на персональном компьютере с процессором Intel Core i5-3570 3.4 ГГц и 8 ГБ оперативной памяти. Время, затраченное на проверку свойств, представлено на рис. 3. Рис. 3,a содержит свойства безопасности (Safety), а рис. 3,b — свойства живости (Liveness). По горизонтальной оси расположены свойства (Properties), по вертикальной оси откладывается время в секундах, затраченное на верификацию (Elapsed time). Первый столбец отражает длительность верификации dcl_LTL-спецификации, второй — imp_LTL-спецификации.



(a)

Fig. 3. Experiment 1. Elapsed time for properties verification: safety (a), liveness (b)



(b)

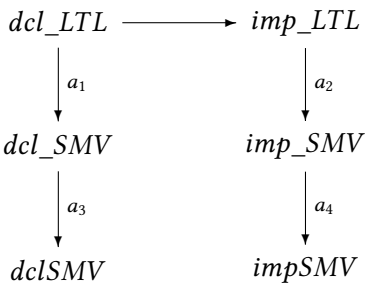
Рис. 3. Эксперимент 1. Затраченное время на верификацию свойств: безопасности (a), живости (b)

Из рис. 3 видно, что время, затраченное на проверку свойства $P2$, совпадает для обеих спецификаций, часть свойств ($P1, P3, P6, \ldots, P9$) проверяются немного быстрее на imp_LTL-спецификации, другая часть ($P4, P5$) наоборот, немного медленнее. В итоге можно считать, что время, затраченное на верификацию декларативной LTL-спецификации, сопоставимо со временем верификации

императивной LTL-спецификации, т. е. ни одна из этих спецификаций не обладает существенным преимуществом в скорости проверки свойств.

4. Преобразование LTL-спецификаций в SMV-код

В разделе 2 по декларативной LTL-спецификации (dcl_LTL) была построена эквивалентная ей императивная LTL-спецификация (imp_LTL) [5]. Выполним перевод обеих LTL-спецификаций в SMV-код. Под SMV-кодом будем понимать код на входном языке инструмента nuXmv. В результате преобразования a_1 получим декларативную SMV-спецификацию (обозначим dcl_SMV), в результате преобразования a_2 — императивную SMV-спецификацию (обозначим imp_SMV). Вышеуказанные преобразования схематично изображены на диаграмме (8).



(8)

Преобразование a_1. В декларативной LTL-спецификации (1) поведения переменной $v \in V$ заменим темпоральный оператор «**G**» на ключевое слово «**TRANS**», которое описывает отношение переходов модели [17]. Полученное отношение переходов содержит только те переходы, которые удовлетворяют формуле, находящейся под действием ключевого слова TRANS. Далее заменим темпоральный LTL-оператор «**X**» на SMV-оператор «**next**», который определяет значение выражения в следующий момент времени. Удалим знак конъюнкции между формулами. В итоге получим следующий (эквивалентный исходной LTL-спецификации) шаблон для перевода:

$$\begin{aligned}&\mathbf{TRANS}\big(\mathbf{next}\big(\neg(v = \_v) \Rightarrow cond_1 \wedge (v = expr_1) \vee \ldots \vee cond_k \wedge (v = expr_k)\big)\big)\\ &\mathbf{TRANS}\big(\mathbf{next}\big(\ \ (v = \_v) \Rightarrow \neg\,(cond_1 \vee \ldots \vee cond_k)\big)\big)\end{aligned} \tag{9}$$

Формулу инициализации поместим под действие ключевого слова INIT. Преобразуем согласно шаблону (9) декларативную LTL-спецификацию поведения программы возведения числа в квадрат (листинг 1). Результирующий SMV-код представлен в листинге 5. Для работы верификатора данный код необходимо перенести из раздела LTLSPEC в раздел ASSIGN.

Листинг 5 (dcl_SMV-спецификация ПЛК-программы)**:**

```
-- S0:
INIT(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:
TRANS(next(!(n = _n) -> _q=0 & _n<bndn & !_PBPls & PBPls & !PBMns & !PBStart & (n = _n + 1) |
                        _q=0 & _n>0 & !_PBMns & PBMns & !PBPls & !PBStart & (n = _n - 1) ))
TRANS(next( (n = _n) -> !(_q=0 & _n<bndn & !_PBPls & PBPls & !PBMns & !PBStart |
                          _q=0 & _n>0 & !_PBMns & PBMns & !PBPls & !PBStart) ))
-- Sa:
TRANS(next( !(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) ))
```

```
TRANS(next(  (a = _a) -> !(_q=0 & PBStart & !(_a=n) | _q=7 & _a<bnda |
                           _q=1 & _a>0 | _q=3 & _a>0) ))
-- Sb:
TRANS(next( !(b = _b) -> _q=4 & _b<bndb & (b = _b + 1) |
                         _q=6 & _b>0    & (b = _b - 1) ))
TRANS(next(  (b = _b) -> !(_q=4 & _b<bndb | _q=6 & _b>0) ))
-- Sc:
TRANS(next( !(c = _c) -> _q=2 & _c<bndc        & (c = _c + 1) |
                         _q=5 & _c<bndc        & (c = _c + 1) |
                         _q=8 & PBReset & _c>0 & (c = 0) ))
TRANS(next(  (c = _c) -> !(_q=2 & _c<bndc | _q=5 & _c<bndc | _q=8 & PBReset & _c>0) ))
-- Sq:
TRANS(next( !(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) ))
TRANS(next(  (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 ) ))
```

Преобразование a_2**.** Декларативной LTL-спецификации (1) поведения переменной $v \in V$ соответствует эквивалентная ей императивная LTL-спецификация (5). Преобразуем формулу (5) — внесём оператор «**X**» внутрь скобки, получим:

$$\mathbf{G}\big(\mathbf{X}(cond_1) \Rightarrow \mathbf{X}(v = expr_1)\big) \wedge \ldots \wedge \mathbf{G}\big(\mathbf{X}(cond_k) \Rightarrow \mathbf{X}(v = expr_k)\big) \wedge \\ \mathbf{G}\big(\neg\,\mathbf{X}(cond_1) \wedge \ldots \wedge \neg\,\mathbf{X}(cond_k) \Rightarrow \mathbf{X}(v = \_v)\big). \tag{10}$$

В формуле (10) заменим темпоральный LTL-оператор «**X**» на SMV-оператор «**next**», раскроем скобки, где присутствует переменная v, и получим следующий императивный шаблон SMV-кода:

$$\begin{aligned} &\mathbf{next}\,(v) := \mathbf{case} \\ &\quad \mathbf{next}(cond_1) : \mathbf{next}(expr_1); \\ &\quad \ldots \\ &\quad \mathbf{next}(cond_k) : \mathbf{next}(expr_k); \\ &\quad \text{TRUE} \qquad\quad : \mathbf{next}(\_v); \\ &\mathbf{esac;} \end{aligned} \tag{11}$$

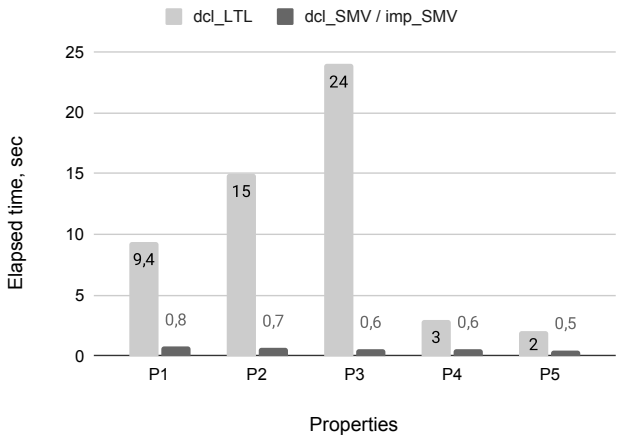
Из декларативной LTL-спецификации поведения программы возведения числа в квадрат (листинг 1) получим императивную LTL-спецификацию и выполним преобразование согласно шаблону (11). Инициализацию выполним для каждой переменной с помощью ключевого слова **init**. Результирующий SMV-код представлен в листинге 6. Данный код необходимо перенести из раздела LTLSPEC в раздел ASSIGN.

Листинг 6 (imp_SMV-спецификация ПЛК-программы)**:**

```
-- S0:
   init(q) := 0; init(_q) := q; init(n) := 0; init(_n) := n;
   init(a) := 0; init(_a) := a; init(b) := 0; init(_b) := b; init(c) := 0; init(_c) := c;
```

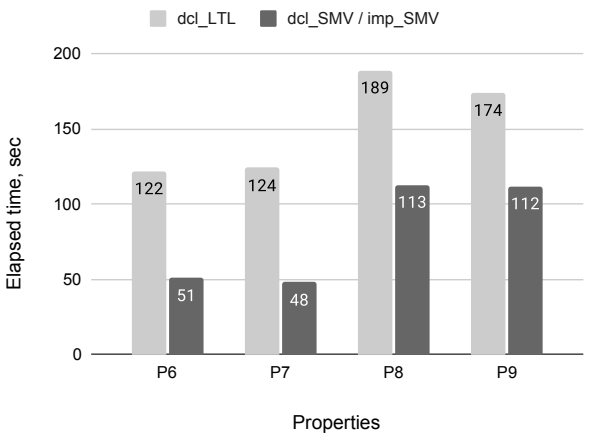
```
    init(PBStart) := FALSE;  init(_PBStart) := PBStart;
    init(PBReset) := FALSE;  init(_PBReset) := PBReset;
    init(PBPls)   := FALSE;  init(_PBPls)   := PBPls;
    init(PBMns)   := FALSE;  init(_PBMns)   := PBMns;
 -- Sn:
    next(n) := case
        next(_q=0 & _n<bndn & !_PBPls & PBPls & !PBMns & !PBStart) : next(_n + 1);
        next(_q=0 & _n>0 & !_PBMns & PBMns & !PBPls & !PBStart)    : next(_n - 1);
        TRUE                                                       : next(_n);
    esac;
 -- Sa:
    next(a) := case
        next(_q=0 & PBStart & !(_a=n)) : next(n);
        next(_q=7 & _a<bnda)           : next(_a + 1);
        next(_q=1 & _a>0)              : next(_a - 1);
        next(_q=3 & _a>0)              : next(_a - 1);
        TRUE                           : next(_a);
    esac;
 -- Sb:
    next(b) := case
        next(_q=4 & _b<bndb) : next(_b + 1);
        next(_q=6 & _b>0)    : next(_b - 1);
        TRUE                 : next(_b);
    esac;
 -- Sc:
    next(c) := case
        next(_q=2 & _c<bndc)        : next(_c + 1);
        next(_q=5 & _c<bndc)        : next(_c + 1);
        next(_q=8 & PBReset & _c>0) : 0;
        TRUE                        : next(_c);
    esac;
 -- Sq:
    next(q) := case
        next(_q=1 &  (_a>0)) : 2;
        next(_q=1 & !(_a>0)) : 8;
        next(_q=2          ) : 3;
        next(_q=3 &  (_a>0)) : 4;
        next(_q=3 & !(_a>0)) : 6;
        next(_q=4          ) : 5;
        next(_q=5          ) : 2;
        next(_q=6 &  (_b>0)) : 7;
        next(_q=6 & !(_b>0)) : 1;
        next(_q=7          ) : 6;
        next(_q=8 & PBReset) : 0;
        next(_q=0 & PBStart) : 1;
        TRUE                 : next(_q);
    esac;
```

С помощью инструмента nuXmv выполним верификацию dcl_SMV-спецификации и imp_SMV-спецификации. Сравним время их верификации со временем верификации dcl_LTL-спецификации. Результаты проверки свойств *P1*, . . . , *P9* представлены на рис. 4. Первый столбец показывает время верификации dcl_LTL-спецификации. Верификация спецификаций dcl_SMV и imp_SMV занимает одинаковое время, которое представлено во втором столбце.



(a)

Fig. 4. Experiment 2. Elapsed time for properties verification: safety (a), liveness (b)



(b)

Рис. 4. Эксперимент 2. Затраченное время на верификацию свойств: безопасности (a), живости (b)

Из рис. 4 видно, что время проверки SMV-спецификаций меньше времени проверки LTL-спецификации. Имеем следующий вектор коэффициентов сокращения времени верификации:

$$(xP_1 = 12;\ xP_2 = 21;\ xP_3 = 40;\ xP_4 = 5;\ xP_5 = 4;\ xP_6 = 2.4;\ xP_7 = 2.6;\ xP_8 = 1.6;\ xP_9 = 1.5),$$

где xP_i — коэффициент сокращения времени верификации свойства Pi, $i = 1, \ldots, 9$. Время проверки свойства $P1$ сократилось в двенадцать раз, свойства $P2$ — в двадцать один раз, $P3$ — в сорок раз, $P4$ — в пять раз, $P5$ — в четыре раза, т. е. время верификации свойств безопасности сократилось до нескольких десятков раз, что является достаточно хорошим показателем. Время проверки свойств $P6$ и $P7$ сократилось примерно в два с половиной раза, $P8$ и $P9$ — примерно в полтора раза. Свойства живости $P6, \ldots, P9$ имеют относительно небольшой коэффициент сокращения времени.

Также заметим, что свойства безопасности $P1, \ldots, P5$ проверяются значительно быстрее свойств живости $P6, \ldots, P9$. У свойств безопасности наблюдается следующая тенденция — при увеличении времени верификации dcl_LTL-спецификации также увеличивается и коэффициент сокращения времени для SMV-спецификаций, что положительно сказывается при проверке свойств безопасности, занимающих длительное время.

Причиной ускорения верификации является сокращение пространства достижимых состояний проверяемой модели. При верификации декларативной LTL-спецификации (dcl_LTL — листинг 1) согласно (7) выполняется анализ псевдополной системы переходов $pLTS$, которая имеет $2.27995 \cdot 10^{16}$ $(2^{54.3399})$ достижимых состояний. Оценку количества достижимых состояний выполняет инструментальное средство nuXmv при верификации.

Задача верификации SMV-спецификаций выглядит следующим образом:

$$LTS \models \psi, \tag{12}$$

где LTS — система переходов, описывающая то же самое поведение, что и декларативная LTL-спецификация φ, т. е. $\Pi_{LTS} = [\![\varphi]\!]_{pLTS}$, ψ — проверяемое свойство. SMV-спецификация непосредственно задаёт систему переходов LTS. При верификации SMV-спецификаций (dcl_SMV — листинг 5, imp_SMV — листинг 6) согласно (12) выполняется анализ системы переходов LTS, которая имеет $9.92256 \cdot 10^5$ $(2^{19.9204})$ достижимых состояний. Таким образом, пространство состояний сократилось примерно в $2.4 \cdot 10^{10}$ (2^{35}) раз.

5. Бисимуляционное преобразование SMV-спецификаций

Из листинга 3 видно, что свойства $P1, \ldots, P9$ формулируются только относительно переменных из $V = \{v_1, \ldots, v_n\}$, переменные из $\_V = \{\_v_1, \ldots, \_v_n\}$ отсутствуют, — данный набор свойств сосредоточен на поведении только основных переменных V. Далее будет показано, что в этом случае переменные множества $\_V$ могут быть удалены из SMV-спецификации.

Выполним преобразование полученных в разделе 4 SMV-спецификаций (dcl_SMV, imp_SMV) согласно диаграмме (8). В результате преобразования a_3 получим новую декларативную SMV-спецификацию (обозначим dclSMV), в результате преобразования a_4 — новую императивную SMV-спецификацию (обозначим impSMV). В обозначениях новых спецификаций отсутствует знак лидирующего подчёркивания «_», что указывает на отсутствие в спецификациях переменных из множества $\_V = \{\_v_1, \ldots, \_v_n\}$.

Преобразование a_3. Выполним эквивалентное преобразование — в шаблоне (9) перенесём оператор **next** внутрь всех скобок и выражений $cond_i$, $expr_i$, где $i = 1, \ldots, k$. В итоге оператор **next** будет применён к каждой переменной. Согласно псевдополной системе переходов $\mathbf{next}(\_v) = v$. Выполним данную подстановку — таким образом, удалим все переменные множества $\_V = \{\_v_1, \ldots, \_v_n\}$ из спецификации. Получим следующий шаблон:

$$\begin{aligned}
&\mathbf{TRANS}\big(\neg(\mathbf{next}(v) = v) \Rightarrow Xcond_1 \wedge (\mathbf{next}(v) = Xexpr_1) \vee \ldots \vee Xcond_k \wedge (\mathbf{next}(v) = Xexpr_k)\big)\\
&\mathbf{TRANS}\big(\ \ (\mathbf{next}(v) = v) \Rightarrow \neg\,(Xcond_1 \vee \ldots \vee Xcond_k)\big)
\end{aligned} \tag{13}$$

В этом шаблоне выражения $Xcond_i$ и $Xexpr_i$ представляют собой исходные выражения $cond_i$ и $expr_i$ соответственно ($i = 1, \ldots, k$), в которых каждая переменная v без лидирующего подчёркивания заменяется на $\mathbf{next}(v)$, а вместо переменных вида $\_v$ подставляются соответствующие переменные вида v. Преобразуем согласно шаблону (13) декларативную SMV-спецификацию поведения программы возведения числа в квадрат (dcl_SMV — листинг 5). В конструкции INIT удалим инициализацию переменных из $\_V = \{\_v_1, \ldots, \_v_n\}$. Результирующий SMV-код представлен в листинге 7. Из раздела ASSIGN исключим выражения вида $\mathbf{next}(\_v) = v$. Теперь переменные из $\_V = \{\_v_1, \ldots, \_v_n\}$ больше нигде не используются, поэтому удалим их из раздела объявления переменных VAR. После чего можно проводить верификацию.

Листинг 7 (dclSMV-спецификация ПЛК-программы)**:**

```
-- S0:
INIT(q=0 & n=0 & a=0 & b=0 & c=0 & !PBStart & !PBReset & !PBPls & !PBMns)
-- Sn:
TRANS(!(next(n) = n) ->
  q=0 & n<bndn & !PBPls & next(PBPls) & !next(PBMns) & !next(PBStart) & (next(n) = n + 1) |
  q=0 & n>0    & !PBMns & next(PBMns) & !next(PBPls) & !next(PBStart) & (next(n) = n - 1) )
TRANS( (next(n) = n) ->
       !(q=0 & n<bndn & !PBPls & next(PBPls) & !next(PBMns) & !next(PBStart) |
         q=0 & n>0    & !PBMns & next(PBMns) & !next(PBPls) & !next(PBStart)) )
-- Sa:
TRANS(!(next(a) = a) -> q=0 & next(PBStart) & !(a=next(n)) & (next(a) = next(n)) |
                        q=7 & a<bnda                       & (next(a) = a + 1) |
                        q=1 & a>0                          & (next(a) = a - 1) |
                        q=3 & a>0                          & (next(a) = a - 1) )
TRANS( (next(a) = a) -> !(q=0 & next(PBStart) & !(a=next(n)) |
                          q=7 & a<bnda | q=1 & a>0 | q=3 & a>0) )
-- Sb:
TRANS(!(next(b) = b) -> q=4 & b<bndb & (next(b) = b + 1) | q=6 & b>0 & (next(b) = b - 1) )
TRANS( (next(b) = b) -> !(q=4 & b<bndb | q=6 & b>0) )
```

```
-- Sc:
TRANS(!(next(c) = c) -> q=2 & c<bndc              & (next(c) = c + 1) |
                        q=5 & c<bndc              & (next(c) = c + 1) |
                        q=8 & next(PBReset) & c>0 & (next(c) = 0) )
TRANS( (next(c) = c) -> !(q=2 & c<bndc | q=5 & c<bndc | q=8 & next(PBReset) & c>0) )
-- Sq:
TRANS(!(next(q) = q) -> q=1 &  (a>0)        & next(q)=2 |
                        q=1 & !(a>0)        & next(q)=8 |
                        q=2                 & next(q)=3 |
                        q=3 &  (a>0)        & next(q)=4 |
                        q=3 & !(a>0)        & next(q)=6 |
                        q=4                 & next(q)=5 |
                        q=5                 & next(q)=2 |
                        q=6 &  (b>0)        & next(q)=7 |
                        q=6 & !(b>0)        & next(q)=1 |
                        q=7                 & next(q)=6 |
                        q=8 & next(PBReset) & next(q)=0 |
                        q=0 & next(PBStart) & next(q)=1 )
TRANS( (next(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 & next(PBReset) | q=0 & next(PBStart) ) )
```

Преобразование a$_4$. Аналогично преобразованию a_3 в шаблоне (11) перенесём оператор **next** внутрь всех выражений $cond_i$, $expr_i$, получим выражения $Xcond_i$ и $Xexpr_i$ соответственно, где $i = 1, \ldots, k$. С учётом **next**$(\_v) = v$ получим шаблон:

$$
\begin{aligned}
&\textbf{next}\,(v) := \textbf{case}\\
&\quad Xcond_1 : Xexpr_1;\\
&\quad \ldots\\
&\quad Xcond_k : Xexpr_k;\\
&\quad \text{TRUE} \quad : v;\\
&\textbf{esac;}
\end{aligned}
\tag{14}
$$

Преобразуем по шаблону (14) императивную SMV-спецификацию поведения программы возведения числа в квадрат (imp_SMV — листинг 6). Из раздела ASSIGN исключим выражения вида **init**$(\_v) = v$ и **next**$(\_v) = v$. И удалим все переменные множества $\_V = \{\_v_1, \ldots, \_v_n\}$ из раздела объявления переменных VAR. Результирующий SMV-код представлен в листинге 8.

Листинг 8 (impSMV-спецификация ПЛК-программы)**:**

```
-- S0:
   init(q) := 0;   init(n) := 0;   init(a) := 0;   init(b) := 0;   init(c) := 0;
   init(PBStart) := FALSE;     init(PBPls) := FALSE;
   init(PBReset) := FALSE;     init(PBMns) := FALSE;
-- Sn:
   next(n) := case
       q=0 & n<bndn & !PBPls & next(PBPls) & !next(PBMns) & !next(PBStart) : n + 1;
       q=0 & n>0    & !PBMns & next(PBMns) & !next(PBPls) & !next(PBStart) : n - 1;
       TRUE                                                                : n;
   esac;
-- Sa:
   next(a) := case
       q=0 & next(PBStart) & !(a=next(n)) : next(n);
       q=7 & a<bnda                       : a + 1;
```

```
        q=1 & a>0                          : a - 1;
        q=3 & a>0                          : a - 1;
        TRUE                               : a;
    esac;
-- Sb:
    next(b) := case
        q=4 & b<bndb : b + 1;
        q=6 & b>0    : b - 1;
        TRUE         : b;
    esac;
-- Sc:
    next(c) := case
        q=2 & c<bndc               : c + 1;
        q=5 & c<bndc               : c + 1;
        q=8 & next(PBReset) & c>0 : 0;
        TRUE                       : c;
    esac;
-- Sq:
    next(q) := case
        q=1 &  (a>0)        : 2;
        q=1 & !(a>0)        : 8;
        q=2                 : 3;
        q=3 &  (a>0)        : 4;
        q=3 & !(a>0)        : 6;
        q=4                 : 5;
        q=5                 : 2;
        q=6 &  (b>0)        : 7;
        q=6 & !(b>0)        : 1;
        q=7                 : 6;
        q=8 & next(PBReset) : 0;
        q=0 & next(PBStart) : 1;
        TRUE                : q;
    esac;
```

С помощью инструмента nuXmv выполним верификацию полученных спецификаций: dclSMV и impSMV. На рис. 5 представлены результаты верификации свойств *P1*, . . . , *P9*. Первый столбец показывает время верификации спецификаций dcl_SMV и imp_SMV. Верификация спецификаций dclSMV и impSMV занимает одинаковое время, которое представлено во втором столбце.

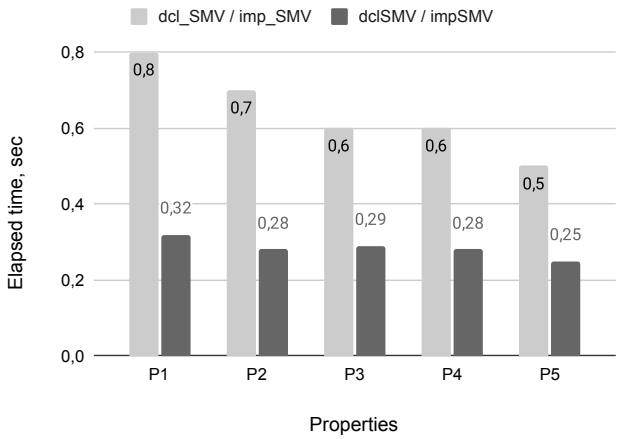
Из рис. 5 видно, что время проверки новых SMV-спецификаций меньше времени проверки предыдущих SMV-спецификаций. Имеем следующий вектор коэффициентов сокращения времени верификации:

$$(xP_1 = 2.5;\ xP_2 = 2.5;\ xP_3 = 2;\ xP_4 = 2.1;\ xP_5 = 2;\ xP_6 = 4.6;\ xP_7 = 4.8;\ xP_8 = 5.4;\ xP_9 = 8.6),$$

где xP_i — коэффициент сокращения времени верификации свойства *Pi*, $i = 1, \ldots, 9$. Время верификации свойств безопасности *P1*, . . . , *P5* сократилось примерно в 2–2.5 раза, время верификации свойств живости *P6*, . . . , *P9* — примерно в 4.5–8.5 раз. На этот раз свойства живости имеют более высокий коэффициент сокращения времени верификации.

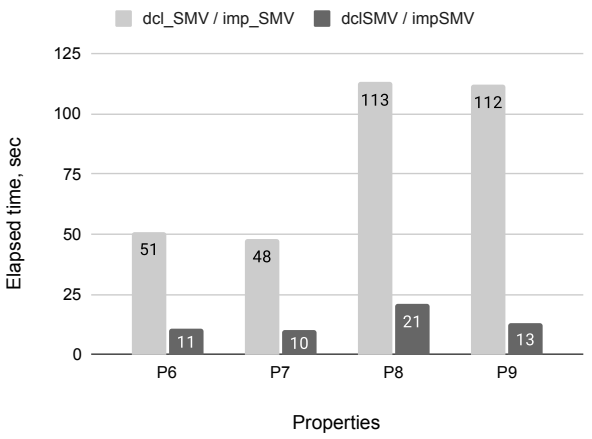
При сравнении времени верификации исходной dcl_LTL-спецификации со временем верификации спецификаций dclSMV и impSMV получим следующий вектор коэффициентов:

$$(xP_1 = 29;\ xP_2 = 53;\ xP_3 = 82;\ xP_4 = 10;\ xP_5 = 8;\ xP_6 = 11;\ xP_7 = 12;\ xP_8 = 9;\ xP_9 = 13),$$



(a)

Fig. 5. Experiment 3. Elapsed time for properties verification: safety (a), liveness (b)



(b)

Рис. 5. Эксперимент 3. Затраченное время на верификацию свойств: безопасности (a), живости (b)

где xP_i — коэффициент сокращения времени верификации свойства Pi, $i = 1, \ldots, 9$. Итоговое время верификации свойств безопасности $P1, \ldots, P5$ сократилось от 8 до 82 раз, время верификации свойств живости $P6, \ldots, P9$ — от 9 до 13 раз.

Причиной ускорения верификации также является сокращение пространства достижимых состояний проверяемой модели. Полученные спецификации (dclSMV — листинг 7, impSMV — листинг 8) задают систему переходов LTS', которая описывает поведение только основных переменных $v_1, \ldots, v_n$. Система переходов LTS имеет $9.92256 \cdot 10^5$ $(2^{19.9204})$ достижимых состояний, а система переходов LTS' $6.2016 \cdot 10^4$ $(2^{15.9204})$ достижимых состояний. Таким образом, пространство состояний сократилось в 16 (2^4) раз. Если сравнивать с псевдополной системой переходов $pLTS$, которая имеет $2^{54.3399}$ достижимых состояний, то пространство состояний сократилось в $2^{38.4195}$ раз.

Удаление вспомогательных переменных $\_v_1, \ldots, \_v_n$ из спецификации привело к изменению системы переходов — как минимум уменьшилось число достижимых состояний, что сократило отношение переходов. Убедимся, что такое изменение модели поведения программы не повлияет на результаты верификации. Только в этом случае можно заменить LTS на LTS' при проверке свойств $P1, \ldots, P9$. Рассмотрим преобразования a_3, a_4 на уровне систем переходов.

Процедура преобразования систем переходов. Пусть дана исходная система переходов $LTS = \langle S, S_0, R, P, L \rangle$, где $S = (D_1 \times \ldots \times D_n)^2$. Вектор состояния $s = (\boldsymbol{a}, \boldsymbol{\_a})$ принимает значения из S, где $\boldsymbol{a}$ и $\boldsymbol{\_a}$ — значения соответствующих векторов $\mathrm{v} = (v_1, \ldots, v_n)$ и $\mathrm{\_v} = (\_v_1, \ldots, \_v_n)$. Преобразуем исходную систему переходов LTS в систему переходов $LTS' = \langle S', S'_0, R', P', L' \rangle$, где $S' = D_1 \times \ldots \times D_n$, с помощью отображения $f : S \to S'$ следующего вида:

$$f(\mathrm{v}, \mathrm{\_v}) = \mathrm{v}, \tag{15}$$

т. е. берётся проекция вектора $(\mathrm{v}, \mathrm{\_v})$, в которой присутствуют только компоненты $v_1, \ldots, v_n$ и отсутствуют компоненты $\_v_1, \ldots, \_v_n$. Функция (15) формализует удаление вспомогательных переменных, что было произведено на уровне SMV-спецификации.

Множество начальных состояний S'_0 также определяется с помощью f на основании S_0, а именно:

$$S'_0 = \{f(s) \in S' \mid s \in S_0\}, \tag{16}$$

т. е. все начальные состояния из LTS и только они преобразуются в начальные состояния в LTS'.

Все переходы в R удовлетворяют спецификациям dcl_SMV и imp_SMV. Так как преобразования a_3 и a_4 сохраняют эквивалентность между исходным и результирующим TRANS-выражениями, то полученные спецификации dclSMV и impSMV задают то же поведение основных переменных $v_1, \ldots, v_n$, несмотря на то, что они уже не зависят от вспомогательных переменных $\_v_1, \ldots, \_v_n$. Таким образом, отношение переходов R' сохраняет переходы из R — для каждого перехода из R с помощью f строится соответствующий ему переход в R' согласно диаграмме:

$$\begin{array}{ccc} s = (\boldsymbol{a}, \_\boldsymbol{a}) & \xrightarrow{\quad f \quad} & s' = \boldsymbol{a} \\ {\scriptstyle \in R}\big\downarrow & & \big\downarrow{\scriptstyle \in R'} \\ s_1 = (\boldsymbol{a}_1, \boldsymbol{a}) & \xrightarrow{\quad f \quad} & s'_1 = \boldsymbol{a}_1 \end{array}$$

Здесь $\boldsymbol{a}, \boldsymbol{a}_1$ — значения вектора $(v_1, \ldots, v_n)$, $\_\boldsymbol{a}$ — вектора $(\_v_1, \ldots, \_v_n)$. В итоге отношение переходов R' определяется следующим образом:

$$R' = \{\big(f(s), f(s_1)\big) \in (S')^2 \mid (s, s_1) \in R\}. \tag{17}$$

Компонент P — множество атомарных утверждений над переменными множества $V \cup \_V$. Здесь $P = P_\varphi \cup P_\psi$, где P_φ — утверждения, используемые при задании LTL-спецификации φ поведения программы, а P_ψ — при задании LTL-спецификации проверяемых свойств $\psi_1, \ldots, \psi_h$. Множество $P_\psi \subset P$ содержит утверждения только относительно переменных из V. В LTS' компонент $P' = P_\psi$.

Разметка состояний из LTS сохраняется и в LTS' для множества P' — если утверждение $p' \in P'$ истинно в $s \in S$, то оно истинно и в $f(s)$:

$$(\forall p' \in P')\ (\forall s \in S)\ [p' \in L(s) \Rightarrow p' \in L'\big(f(s)\big)]. \tag{18}$$

Определение бисимуляции. Системы переходов $LTS = \langle S, S_0, R, P', L\rangle$ и $LTS' = \langle S', S'_0, R', P', L'\rangle$ находятся в отношении *бисимуляции* $B \subseteq S \times S'$, если выполняются следующие три условия [10]:

$$(\forall s \in S)\ (\forall s' \in S')\ [B(s, s') \Rightarrow L(s) = L'(s')], \tag{19}$$

$$(\forall s, s_1 \in S)\ (\forall s' \in S')\ [B(s, s') \wedge R(s, s_1) \Rightarrow (\exists s'_1 \in S')\ R'(s', s'_1) \wedge B(s_1, s'_1)], \tag{20}$$

$$(\forall s \in S)\ (\forall s', s'_1 \in S')\ [B(s, s') \wedge R'(s', s'_1) \Rightarrow (\exists s_1 \in S)\ R(s, s_1) \wedge B(s_1, s'_1)]. \tag{21}$$

Условие (19) является условием *совпадения разметки* соответствующих состояний. Условие (20) утверждает, что если существует переход в LTS, то существует и соответствующий ему переход в LTS'. То же верно и в обратном направлении — условие (21).

Системы переходов LTS и LTS' *бисимуляционно эквивалентны* [10, 18] (обозначим $LTS \equiv LTS'$), если существует такое отношение бисимуляции B, что:

$$(\forall s_0 \in S_0)\ (\exists s'_0 \in S'_0)\ B(s_0, s'_0), \tag{22}$$

$$(\forall s'_0 \in S'_0)\ (\exists s_0 \in S_0)\ B(s_0, s'_0), \tag{23}$$

т. е. для любого начального состояния в LTS найдётся (в соответствии с отношением бисимуляции B) начальное состояние в LTS', и наоборот.

Известна теорема [10], которая утверждает, что если системы переходов бисимуляционно эквивалентны $LTS \equiv LTS'$, то для любой формулы ψ логики CTL* верно:

$$(LTS \models \psi) \Leftrightarrow (LTS' \models \psi), \tag{24}$$

т. е. если свойство ψ выполняется на LTS, то оно выполняется и на LTS', и если свойство ψ нарушается на LTS, то оно нарушается и на LTS'. Так как логика LTL является подмножеством логики CTL*, то данная теорема верна и для любой LTL-формулы.

Утверждение 1 (О сохранении бисимуляционной эквивалентности)**.** *Преобразование систем переходов с помощью отображения* $f : S \to S'$ *вида* $f(\mathrm{v}, \_\mathrm{v}) = \mathrm{v}$, *удовлетворяющее условиям* (16), (17), (18), *сохраняет бисимуляционную эквивалентность, т. е. имеем* $LTS = \langle S, S_0, R, P', L\rangle \equiv LTS' = \langle S', S'_0, R', P', L'\rangle$.

Доказательство. Формально сохранение бисимуляционной эквивалентности — это утверждение (15), (16), (17), (18) $\vdash$ (19), (20), (21), (22), (23).

Согласно (18) для P' при преобразовании разметка состояний сохраняется. Так как истинность/ложность любого атомарного утверждения $p' \in P'$ зависит только от v и не зависит от _v, а состояния $s \in S$ и $f(s) \in S'$ всегда имеют одно и то же значение вектора v согласно (15), то все прообразы для $f(s) \in S'$ всегда имеют одинаковую разметку, т. е. верно утверждение:

$$(\forall p' \in P')\,(\forall s \in S)\,[p' \in L'\big(f(s)\big) \Rightarrow p' \in L(s)]. \tag{25}$$

Для множества P' оба утверждения (18) и (25) образуют условие *равенства разметки* состояний:

$$(\forall p' \in P')\,(\forall s \in S)\,[L(s) = L'\big(f(s)\big)]. \tag{26}$$

Из (26) следует (19) при $s' = f(s)$.

Согласно (17) переходы в R' образуются только из переходов из R, поэтому по построению любому переходу из R соответствует ровно один переход в R', и любому переходу из R' соответствует хотя бы один переход в R, т. е. верны утверждения (20) и (21).

В соответствии с (16) все начальные состояния из LTS и только они преобразуются в начальные состояния в LTS', поэтому по построению любому начальному состоянию из S_0 соответствует ровно одно состояние в S'_0, и любому начальному состоянию из S'_0 соответствует хотя бы одно начальное состояние в S_0, т. е. верны утверждения (22) и (23). Таким образом, предложенное преобразование систем переходов сохраняет бисимуляционную эквивалентность относительно множества P'. □

Бисимуляционная эквивалентность $LTS \equiv LTS'$ согласно теореме (24) позволяет заменить систему переходов LTS на систему переходов LTS' без изменения результатов проверки LTL-свойств.

Заключение

В настоящей работе предложены преобразования декларативной (dcl_LTL) и императивной (imp_LTL) LTL-спецификаций [5] в декларативную (dcl_SMV) и императивную (imp_SMV) SMV-спецификации соответственно. Данные SMV-спецификации задают систему переходов LTS, которая по сравнению с псевдополной системой переходов $pLTS$ имеет меньший размер пространства состояний. Система переходов $pLTS$ по определению содержит состояния и переходы, которые не удовлетворяют LTL-спецификации поведения программы. В системе переходов LTS исключены такие состояния и переходы. Уменьшение размера пространства состояний приводит к сокращению времени верификации модели программы.

Если проверяемые LTL-свойства не содержат вспомогательных переменных, то могут быть выполнены дальнейшие преобразования SMV-спецификаций: перевод dcl_SMV и imp_SMV в спецификации dclSMV и impSMV соответственно. Вновь полученные SMV-спецификации задают систему переходов LTS', которая по сравнению с LTS имеет меньший размер пространства состояний. Это также приводит к сокращению времени верификации. Система переходов LTS описывает поведение основных и вспомогательных переменных. Система переходов LTS' описывает поведение только основных переменных. Предложенное преобразование SMV-спецификаций сохраняет бисимуляционную эквивалентность систем переходов: $LTS \equiv LTS'$. Это позволяет заменить одну SMV-спецификацию на другую без каких-либо изменений результатов верификации.

Из dcl_LTL-спецификации проще получить dclSMV-спецификацию, чем impSMV-спецификацию. Преимуществом impSMV-спецификации является её компактность по сравнению с dclSMV-спецификацией. Выбор декларативности или императивности спецификации зависит от целей и удобства

её использования. Для описания требуемого поведения переменных УПО больше подходит декларативный стиль, для построения кода программы — императивный.

Таким образом, в настоящей работе были предложены четыре новые SMV-спецификации, которые позволяют значительно сократить время верификации декларативной LTL-спецификации.

References

[1] D. J. Smith and K. G. Simpson, *The Safety Critical Systems Handbook*, 5th. Butterworth-Heinemann, 2020, ISBN: 978-0-12-820700-0.

[2] 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. DOI: 10.1109/TCAD.2008.923410.

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

[4] E. Lee and S. Seshia, *Introduction to Embedded Systems – A Cyber-Physical Systems Approach*, 2nd. MIT Press, 2017, ISBN: 978-0-262-53381-2.

[5] M. Neyzov and E. Kuzmin, "LTL-specification for Development and Verification of Control Programs", *Modeling and Analysis of Information Systems*, vol. 30, no. 4, pp. 308–339, 2023, in Russian. DOI: 10.18255/1818-1015-2023-4-308-339.

[6] D. Harel and A. Pnueli, "On the Development of Reactive Systems", in *Logics and Models of Concurrent Systems*, vol. 13, 1985, pp. 477–498. DOI: 10.1007/978-3-642-82453-1_17.

[7] 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. DOI: 10.1007/BFb0027047.

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

[9] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*, 1st. Springer Publishing Company, Incorporated, 2018, ISBN: 3319105744.

[10] E. M. Clarke, O. Grumberg, and D. Peled, *Verification of Program Models: Model Checking*. MCNMO, 2002, p. 416, translated from English to Russian, ISBN: 5-94057-054-2.

[11] D. Beyer and A. Podelski, "Software model checking: 20 years and beyond", in *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, Springer, 2022, pp. 554–582, ISBN: 978-3-031-22337-2. DOI: 10.1007/978-3-031-22337-2_27.

[12] *nuXmv home.* [Online]. Available: https://nuxmv.fbk.eu/.

[13] *IEC 61131-1:2003 Programmable controllers – Part 1: General information.* [Online]. Available: https://webstore.iec.ch/publication/4550.

[14] B. Alpern and F. B. Schneider, "Defining Liveness", *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, 1985. DOI: 10.1016/0020-0190(85)90056-0.

[15] Z. Manna and A. Pnueli, "A Hierarchy of Temporal Properties", in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, 1990, pp. 377–410. DOI: 10.1145/93385.93442.

[16] *Spot home.* [Online]. Available: https://spot.lre.epita.fr/.

[17] *nuXmv User Manual.* [Online]. Available: https://nuxmv.fbk.eu/downloads/nuxmv-user-manual.pdf.

[18] D. Park, "Concurrency and Automata on Infinite Sequences", in *Theoretical Computer Science: 5th GI-Conference Karlsruhe*, vol. 104, 1981, pp. 167–183. DOI: 10.1007/BFb0017309.