MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 31, NO. 3, 2024
journal homepage: www.mais-journal.ru

THEORY OF SOFTWARE

LTL-Specification for Development and Verification of Logical Control Programs in Feedback Systems

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

DOI: 10.18255/1818-1015-2024-3-240-279

[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 August 6, 2024
Revised August 22, 2024
Accepted August 28, 2024

The article continues the series of publications on the development and verification of control programs based on LTL-specifications of a special type. Earlier, a declarative LTL-specification was proposed to describe the strictly deterministic behavior of programs, ways of its verification and translation were worked out: for verification, the model checking tool nuXmv is used, and the translation is carried out into an imperative programming language ST for programmable logic controllers. When verifying the declarative LTL-specification of the behavior of programs, there may be a need to simulate the behavior of its environment. In general, it is required to ensure the possibility of constructing closed-loop systems "program-environment". In this work, an LTL-specification of constraintly nondeterministic behavior of a Boolean variable is proposed to describe the behavior of the environment of logical control programs. This specification allows defining the behavior of Boolean feedback signals, as well as fairness conditions to exclude unrealistic scenarios of behavior. The article proposes an approach to the development and verification of logical control programs, within which the behavior model of the program environment is described in the form of constraints on the behavior of its input signals, what allows avoiding a separate detailed representation of the processes of the environment operation. As a result, the obtained behavior model of the closed-loop system "program-environment" provides a number of advantages: simplification of the modeling process, reduction of the state space of the verified model, and reduction of verification time. If it is impossible to reduce the behavior of the environment to the behavior of existing input signals, this approach suggests using "imaginary" sensors — additional Boolean variables that are used as an auxiliary means for describing the behavior of input signals. The purpose of introducing imaginary sensors is to compensate for missing sensors to track the specific behavior of some elements of the environment that needs to be taken into account when defining realistic behavior of the inputs of a logical control program. The proposed approach to the development and verification of programs taking into account the behavior of the environment (a control object) is demonstrated by the example of an industrial plastic molding plant.

Keywords: control software; PLC program; declarative LTL-specification; LTL-specification of constraintly nondeterministic behavior; fairness conditions; closed-loop systems verification; plant behavior model; imaginary sensor; temporal properties; model checking; nuXmv verifier; SMV-specification

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, "LTL-specification for development and verification of logical control programs in feedback systems", *Modeling and Analysis of Information Systems*, vol. 31, no. 3, pp. 240–279, 2024. DOI: 10.18255/1818-1015-2024-3-240-279.

© Neyzov M. V., Kuzmin E. V., 2024
This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 31, № 3, 2024
сайт журнала: www.mais-journal.ru

LTL-спецификация для разработки и верификации программ логического управления в системах с обратной связью

М. В. Нейзов[1], Е. В. Кузьмин[2]

DOI: 10.18255/1818-1015-2024-3-240-279

[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 6 августа 2024 г.
После доработки 22 августа 2024 г.
Принята к публикации 28 августа 2024 г.

Статья продолжает цикл публикаций по разработке и верификации управляющих программ на основе LTL-спецификаций специального вида. Ранее для описания строго детерминированного поведения программ была предложена декларативная LTL-спецификация, проработаны способы её верификации и трансляции: для верификации используется инструмент проверки модели nuXmv, трансляция осуществляется в императивный язык программирования ST для программируемых логических контроллеров. При верификации декларативной LTL-спецификации поведения программ может возникнуть необходимость в моделировании поведения её окружения. В общем случае требуется обеспечить возможность построения замкнутых систем «программа-окружение». В настоящей работе для описания поведения окружения программ логического управления предложена LTL-спецификация ограниченно недетерминированного поведения булевой переменной. Данная спецификация позволяет задавать поведение булевых сигналов обратной связи, а также условия справедливости для исключения нереалистичных сценариев поведения. В статье предлагается подход к разработке и верификации программ логического управления, в рамках которого модель поведения окружения программы описывается в виде ограничений на поведение её входных сигналов, что позволяет избежать отдельного детального представления процессов функционирования окружения. В результате полученная модель поведения замкнутой системы «программа-окружение» даёт ряд преимуществ: упрощение процесса моделирования, сокращение пространства состояний проверяемой модели, снижение времени верификации. При невозможности сведения поведения окружения к поведению имеющихся входных сигналов данный подход предполагает применение «мнимых» датчиков — дополнительных булевых переменных, использующихся как вспомогательное средство для описания поведения входных сигналов. Цель введения мнимых датчиков состоит в компенсации недостающих датчиков для отслеживания специфического поведения отдельных элементов окружения, которое необходимо учесть при задании реалистичного поведения входов программы логического управления. Предложенный подход к разработке и верификации программ с учётом поведения окружения (объекта управления) демонстрируется на примере промышленной установки для литья пластмасс.

Ключевые слова: управляющее программное обеспечение; ПЛК-программа; декларативная LTL-спецификация; LTL-спецификация ограниченно недетерминированного поведения; условия справедливости; верификация замкнутых систем; модель поведения установки; мнимый датчик; темпоральные свойства; проверка модели; верификатор nuXmv; SMV-спецификация

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, “LTL-specification for development and verification of logical control programs in feedback systems”, *Modeling and Analysis of Information Systems*, vol. 31, no. 3, pp. 240–279, 2024. DOI: 10.18255/1818-1015-2024-3-240-279.

© Нейзов М. В., Кузьмин Е. В., 2024
Эта статья открытого доступа под лицензией CC BY license (https://creativecommons.org/licenses/by/4.0/).

Введение

Промышленные *киберфизические системы* (КФС) [1—3] используются для автоматизации производства [4]. *Управляющее программное обеспечение* (УПО) является неотъемлемой частью КФС и зависит от её назначения [5]. КФС условно можно разделить на УПО и его окружение. К окружению могут относиться:

- программные компоненты, не входящие в УПО (служба времени, сторонние сервисы);
- аппаратные компоненты системы управления (датчики, исполнительные устройства);
- компоненты технологического оборудования (механизмы, резервуары, камеры нагрева).

Окружение (или его часть) подвергается управлению со стороны УПО. Управляемая часть окружения является *объектом управления* (ОУ). Технологическое оборудование промышленной установки представляет собой *технологический объект управления* (ТОУ). *Программируемый логический контроллер* (ПЛК) [6] является наиболее распространённой вычислительной платформой для УПО промышленных КФС [4, 7]. К ПЛК подключаются датчики и исполнительные устройства, посредством которых он взаимодействует с ТОУ.

УПО работает циклически: считывает входные данные из окружения, выполняет вычисления, обновляет выходные данные для окружения. Таким образом, УПО КФС представляет собой *реагирующую систему* [8], так как постоянно взаимодействует со своим окружением. Совокупность всех входных и выходных сигналов УПО образует его *интерфейс*. Любой выходной сигнал УПО является сигналом *прямой связи* (feedforward) с ОУ. Входной сигнал УПО, который информирует о реакции ОУ на некоторое управляющее воздействие, является сигналом *обратной связи* (feedback). Система с прямой и обратной связями образует *замкнутый контур* (closed-loop), поэтому такие системы называются *замкнутыми*.

К промышленным КФС предъявляются высокие требования надежности и безопасности [9, 10]. Функционирование КФС напрямую зависит от УПО, поэтому его *корректность* является критическим показателем [11, 12]. Для повышения уровня доверия к УПО и гарантий его корректности используют *формальные методы верификации* [13, 14]. *Проверка модели* (model checking) [15—17] является наиболее распространённым формальным методом верификации реагирующих систем. Проверке подлежат *темпоральные свойства* [18—20], которые чаще всего выражаются в виде формул *линейной темпоральной логики* LTL (Linear Temporal Logic) или *логики деревьев вычислений* CTL (Computation Tree Logic) [15]. Фактическое поведение реагирующей системы (УПО) задаётся в виде конечной системы переходов — *структуры Крипке* [15—17]. Верификация осуществляется с помощью инструментальных средств, которые проверяют соответствие модели фактического поведения заданной *спецификации* — набору требуемых свойств. Для проверки некоторых свойств недостаточно модели поведения УПО, может потребоваться модель поведения замкнутой системы «УПО-ОУ» [21].

Согласно [22] можно выделить три подхода к формальной верификации УПО ПЛК:

1. Подход, *основанный на модели* (model based) поведения ОУ. Верификации подлежит модель поведения замкнутой системы, состоящей из УПО и ОУ.
2. Подход, *не основанный на модели* (non model based) поведения ОУ. Проводится верификация только модели поведения УПО. Входы УПО ведут себя абсолютно недетерминировано.
3. Подход, *основанный на ограничениях* (constrained based) поведения входов УПО. На входы УПО накладываются ограничения, основанные на знаниях о невозможном поведении окружения.

Настоящая работа имеет отношение к первому и третьему подходам: к первому — так как даёт возможность описания поведения элементов ОУ, к третьему — поскольку нацелена на описание ограничений, накладываемых на поведение входных сигналов УПО.

Окружение воздействует на входы УПО и определяет их поведение. Таким образом, поведение входов УПО полностью или частично отражает поведение окружения. Этот факт даёт возможность задания модели поведения окружения в виде модели поведения входов УПО с некоторым дополне-

нием в случае необходимости. При описании поведения входных сигналов УПО трудность обычно вызывают сигналы обратной связи, так как они зависят от поведения ОУ, которое, в свою очередь, зависит от выходных сигналов УПО — все эти особенности влияют на поведение сигналов обратной связи. Данная модель поведения по сути является некоторым недетерминированным дискретным аналогом *передаточной функции* (transfer function) ОУ из теории автоматического управления.

В случае трудности или невозможности сведения модели поведения окружения к модели поведения входов УПО предлагается применять подход, основанный на модели: использовать дополнительные переменные, отражающие состояние компонентов из окружения. В этом случае можно считать, что дополнительная переменная представляет собой *мнимый* датчик, показания которого предназначены для описания поведения входов УПО. При этом получается, что в общем случае модель поведения окружения представляет собой модель поведения входов УПО и мнимых датчиков.

В итоге при верификации модели поведения замкнутого контура «УПО-ОУ» нет необходимости в детальном описании поведения ОУ, которое приводит к повышению трудозатрат и требует учёта всех его возможных сценариев поведения. При описании ограничений входов необходимо лишь учесть все запрещённые сценарии, что по нашему мнению сделать проще. Также наличие модели поведения ОУ в контуре увеличивает пространство состояний результирующей модели «УПО-ОУ» и, как следствие, время верификации, а введение ограничений на входы УПО, наоборот, сокращает пространство достижимых состояний [23].

Таким образом, при построении модели замкнутого контура «УПО-ОУ» помимо описания модели поведения УПО основной целью данной статьи является задание ограничений на поведение сигналов обратной связи без определения деталей в поведении ОУ. К детализации в поведении ОУ следует прибегать в крайнем случае, когда поведение сигналов обратной связи сложно или невозможно описать без некоторых характеристик ОУ.

УПО может решать большой спектр задач. Настоящая работа сфокусирована на задачах *логического управления* (ЛУ) [24]. Данный вид управления использует только булевы входные и выходные сигналы. УПО или его часть с таким интерфейсом назовём программой ЛУ (ПЛУ). Отметим, что в общем случае окружение ПЛУ содержит в себе окружение УПО, а также ту часть УПО, которая не входит в ПЛУ. Таким образом, при выделении ПЛУ из УПО происходит расширение окружения/ОУ. Пусть, например, УПО содержит счётчик и компаратор. Счётчик фиксирует количество нажатий кнопки и выдаёт сигнал, когда их число достигнет заданного значения. Компаратор срабатывает, когда фактическое значение температуры выше уставки. Данные компоненты входят в окружение ПЛУ. Более того, если счётчик имеет вход для сброса, то он является и ОУ для ПЛУ.

В настоящей работе предлагается описывать поведение окружения ПЛУ с помощью LTL-спецификации специального вида, которая основана на идее из публикаций по согласованному поведению датчиков [25, 26]. Данная LTL-спецификация в общем случае задаёт недетерминированное, но ограниченное поведение булевых переменных. При этом поведение ПЛУ является абсолютно детерминированным и описывается с помощью *декларативной* [27] LTL-спецификации. В [28] для ускорения верификации с помощью инструмента символьной проверки модели nuXmv [29] декларативная LTL-спецификация преобразуется в SMV-спецификацию — код на входном языке инструмента nuXmv. Настоящая работа также для ускорения верификации предлагает преобразование LTL-спецификации ограниченно недетерминированного поведения окружения ПЛУ в SMV-спецификацию. Далее SMV-спецификации ПЛУ и её окружения объединяются и образуют SMV-спецификацию КФС, которая подлежит верификации.

Настоящее исследование является продолжением цикла статей по разработке и верификации УПО на основе LTL-спецификаций специального вида. В качестве языка спецификации поведения был выбран формализм LTL, а не входной язык инструмента проверки модели. Это сделано по двум причинам:

1. Отсутствует привязка к конкретному инструменту проверки модели. LTL-спецификацию можно использовать как инструментально независимый формализм, который всегда можно перевести во входной язык используемого в данный момент инструмента. Это обусловлено тем, что LTL-спецификация не навязывает применение конкретной модели, в отличие от инструментов верификации [30].
2. LTL-спецификация является математическим объектом, что позволяет работать с ней, используя соответствующий математический аппарат и любые инструментальные средства, поддерживающие язык LTL. Например, можно сравнить две LTL-формулы с помощью программного средства Spot [31]. Это позволяет легко выполнять эквивалентные преобразования LTL-спецификаций, определять их избыточность или противоречивость. Сравнение выражений (например определение эквивалентности) на входном языке инструмента проверки модели может быть затруднительно.

Содержание работы. Раздел 1 представляет обзор связанных работ. Раздел 2 содержит предварительные сведения об LTL-спецификации поведения программ. Частный случай декларативной LTL-спецификации для булевой переменной представлен в разделе 3. В разделе 4 определена LTL-спецификация ограниченно недетерминированного поведения булевой переменной для описания окружения программ. В разделе 5 приведён пример LTL-спецификации поведения установки для литья пластмасс, а в разделе 6 — её LTL-спецификация свойств. В разделе 7 осуществляется перевод LTL-спецификации поведения установки в SMV-спецификацию для ускорения верификации. Выполнена стандартизация предложенных шаблонов условий справедливости. В разделе 8 на основе LTL-спецификации поведения ПЛУ построена ST-программа для ПЛК. В заключительном разделе подводятся итоги работы. Приложения 1–6 содержат листинги спецификаций и программного кода, а также доказательства утверждений.

1. Обзор связанных работ

Относительно предлагаемого в настоящей статье подхода к верификации рассмотрим краткий обзор работ, связанных с построением моделей поведения окружения УПО для верификации замкнутого контура «УПО-окружение» методом проверки модели. В обозреваемых работах ТОУ чаще всего называется установкой (plant), поэтому в данном разделе будем использовать этот термин.

В работах [32—44] для описания поведения установки используют формализмы на основе *сетей Петри* и *автоматов* специального вида.

Системы сетевых условий/событий (Net Condition/Event Systems, NCES) [32] — расширение сетей Петри, которое получило достаточно широкое распространение в области моделирования поведения замкнутых систем. В [37—39] разработка и верификация NCES-модели осуществляется с помощью инструмента ViVe/SESA. В работах [40, 41] используют дальнейшее расширение формализма NCES — Timed NCES (TNCES).

Также применяются формализмы, которые являются расширениями автоматов. В работе [42] *временные автоматы* (Timed Automata, TA) переводятся во входной язык верификатора NuSMV. В [43] TA-модель проверяется с помощью верификатора UPPAAL. Работа [44] нацелена на построение моделей, описывающих реалистичное поведение установок. Для этого используется формализм TA *с дискретными данными* (TA with Discrete Data, TADD). Авторы не выбирают конкретный верификатор для проверки TADD-моделей.

В работах [33—36] предлагается строить дискретную модель поведения установки не с нуля, а на основе уже имеющихся артефактов. В [33] TNCES-модель строится на основе *гибридного автомата* (hybrid automaton), заданного в виде модели Stateflow. В [34, 35] TNCES-модель установки транслируется из UML-диаграмм. Для верификации модели замкнутой системы применяется инструмент проверки модели SESA. В [36] предлагается на основе CAD-модели установки строить

имитационную модель, далее по ней формируется TNCES-модель. Процедура преобразования частично автоматизирована.

Отметим, что при использовании вышеуказанных формализмов из работ [32—44] получается достаточно громоздкое описание модели поведения установки. В работе [45] вообще утверждается, что нет удобного систематического способа описания таких моделей. Кроме того, требуются специализированные инструменты для работы с их графическим представлением. Напротив, LTL-спецификация представляет собой более компактное текстовое описание.

В работах [46—51] применяются специализированные нотации [52—55], которые используются и для разработки УПО, и для описания моделей поведения установки/окружения.

В [46] для описания модели поведения замкнутой системы, и установки в частности, предлагается использовать функциональные блоки стандарта МЭК 61499 [52]. В [47—49] данные блоки автоматически переводятся во входной язык инструмента NuSMV. Недетерминизм поведения блоков обеспечивается с помощью специальных событий NDT, инициирующих недетерминированные переходы в модели поведения установки.

В [50] для описания УПО и его модели окружения используется язык poST [53] — процесс-ориентированное расширение языка ST стандарта МЭК 61131-3. Недетерминизм окружения обеспечивается за счёт переменных с абсолютно недетерминированным поведением. Разработан транслятор с языка poST на Promela — входной язык верификатора Spin.

Синхронный язык [56] программирования Lustre [54] для реактивных систем можно рассматривать как подмножество темпоральной логики [57]. В работе [51] предлагается использовать Lustre для разработки и верификации критичных систем реального времени: программа, её окружение и проверяемые свойства задаются на одном языке. Поведение окружения описывается с помощью механизма *утверждений* (assertion). Согласно [57] любое свойство *безопасности* может быть выражено на Lustre. Для верификации используется инструмент Lesar [58].

Программы на расширении языка Lustre могут быть верифицированы с помощью инструмента проверки модели Kind 2 [59]. Поведение окружения и проверяемые свойства для компонентов Lustre-программы задаются с помощью контрактов в стиле *«предположение-гарантия»* (assume-guarantee). Контракты записываются на этом же языке. Язык CoCoSpec [55] является расширением языка Lustre и предназначен для спецификации контрактов реактивных систем с учетом режимов.

Отметим, что использование специализированных и стандартизированных нотаций [52—55] делает спецификации удобочитаемыми, однако их выразительные возможности ограничены: могут возникнуть трудности (в том числе непреодолимые) при описании *мгновенных циклических зависимостей* (instantaneous cyclic dependency) [60], а также ограничений *справедливости* [16, 57].

В работе [61] подчеркивается, что при верификации КФС необходимо моделирование её физических аспектов. Авторы предлагают разделение КФС на кибер- и физическую части с фиксацией интерфейса между ними. Модель поведения определяется только для киберчасти, поведение физической части сводится к описанию поведения интерфейса между ними. Также отмечается, что при наличии определенных знаний о поведении среды требуется добавить дополнительный компонент для её моделирования. Таким образом, данная работа предлагает гибридный подход к верификации, который основан на модели поведения окружения и на ограничениях входов киберчасти. В этом заключается сходство с нашей работой.

Для описания киберчасти используется Lingua Franca (LF) — язык программирования КФС, основанный на *реакторах*. LF-программа переводится на язык реактивных объектов Rebeca — входной язык инструмента проверки модели Afra. Внешние недетерминированные воздействия моделируются через серверы сообщений.

В [62] выполняется автоматический перевод LF-программ в формальную аксиоматическую модель, которая подлежит верификации методом *ограниченной проверки модели* (bounded model

checking) с помощью инструмента Uclid5. Требуемые свойства модели задаются в виде формул Safety Metric Temporal Logic (Safety MTL).

В работе [61] представлен простой пример контроллера дверей поезда, в [62] — системы помощи водителю автомобиля. Данные примеры демонстрируют незамкнутые КФС. В [61, 62] авторы не раскрывают особенностей разработки и верификации систем с замкнутым контуром, поэтому трудно оценить достоинства и недостатки их работ в этом отношении.

Lutin [63] является специализированным языком спецификации недетерминированного поведения. Язык описывает ограничения, накладываемые на хаотическое поведение. Основная цель языка — моделирование (выполнение) сценариев: по Lutin-спецификации генерируются тестовые последовательности для инструмента Lurette. Также язык может использоваться для прототипирования — описания поведения незавершенных компонентов реактивной системы и её окружения. Lutin не ориентирован на верификацию методом проверки модели, однако имеет такой же принцип описания поведения, как и в настоящей работе, который заключается в наложении ограничений на абсолютный недетерминизм.

В синтезе реактивных систем [15, 64] GR(1) [15, 65] является широко используемым фрагментом логики LTL, который ограничивает вид LTL-формул, описывающих предположения (assumptions) относительно окружения синтезируемой системы. В GR(1) для описания эволюции системы используется формула вида $\mathbf{G}(\varphi \Rightarrow \mathbf{X}\psi)$, а для описания целей, которые необходимо достигать бесконечно часто, — формула вида $\mathbf{GF}\gamma$, где φ, ψ и γ — формулы состояния. В настоящей работе также предлагается описывать поведение окружения УПО в виде LTL-формул специального вида, однако есть некоторые отличия. В GR(1) описание допустимых переходов выглядит следующим образом: для некоторого множества исходных состояний (удовлетворяющих φ) задаётся ограничение (в виде ψ) на состояния, которые являются их непосредственными потомками. В настоящей работе на заданное множество переходов для конкретной булевой переменной накладываются ограничения на состояния, образующие эти переходы. Другое отличие: в спецификации GR(1) допускается только *безусловная справедливость* [16], в LTL-спецификации ограниченного недетерминизма — более сложные формы справедливости.

Ряд работ [66—71] направлен на синтез моделей поведения. Работа [66] освещает вопрос идентификации модели поведения замкнутой системы «контроллер-установка» на основе сигналов их интерфейса. Результатом идентификации является недетерминированный автономный автомат с выходом (non deterministic autonomous automaton with output).

В работе [67] представлен автоматический вывод модели поведения установки на основе трассировок выполнения и их LTL-свойств. Трассировки могут быть получены в результате симуляции или реальной работы установки. Вывод модели в виде недетерминированного *автомата Мура* сводится к задаче *выполнимости* (satisfiability, SAT). Проверка CTL-свойств модели замкнутой системы осуществляется с помощью инструмента NuSMV. В подобной работе [68] предлагается вывод модели только на основе трассировок, полученных в результате симуляции. Более масштабируемые методы генерации для моделей большой размерности были представлены в [69].

В [70] на основе трассировок выполнения, полученных от цифрового двойника установки, генерируется её SMV-спецификация. УПО в виде функциональных блоков стандарта МЭК 61499 также транслируется в SMV-код. Далее полученные SMV-спецификации объединяются и результирующая модель замкнутой системы проверяется с помощью инструмента NuSMV.

В [71] при симуляции работы установки формируется журнал событий, на основании которого алгоритм альфа-обнаружения процессов формирует модель поведения в виде сети Петри. Далее сеть Петри преобразуется в функциональный блок стандарта МЭК 61499. В результате его трансляции получается SMV-код для дальнейшей верификации в NuSMV.

Для [66—71] отметим, что в результате синтеза на основе трассировок выполнения может получиться неточная модель поведения установки, что недопустимо при формальной верификации, так как это влияет на достоверность полученных результатов. Синтезированные модели поведения установок могут применяться для поиска ошибок в УПО, но не для доказательства их отсутствия.

2. Предварительные сведения об LTL-спецификации поведения программ

В работе [27] для описания поведения управляющих программ представлена LTL-спецификация, которая содержит *основные* $V = \{v_1, \ldots, v_n\}$ и *вспомогательные* $\_V = \{\_v_1, \ldots, \_v_n\}$ переменные, принимающие значения из соответствующих множеств $D_1, \ldots, D_n$. Основные переменные $v_1, \ldots, v_n$ хранят входные и выходные данные, а также внутреннее состояние программы. Вспомогательные переменные $\_v_1, \ldots, \_v_n$ предназначены для возможности обращения к значениям соответствующих основных переменных в предыдущий момент времени. Множество $S = (D_1 \times \ldots \times D_n)^2$ представляет собой пространство возможных состояний программы. Каждый цикл управления приводит к переходу из состояния программы $s \in S$ в состояние $s' \in S$.

В качестве модели поведения программы используется *система переходов* (transition system) $TS = \langle S, S_0, R, P, L\rangle$, где S — множество состояний, $S_0 \subseteq S$ — множество начальных состояний, $R \subseteq S \times S$ — *тотальное* отношение переходов, $P = \{p_1, \ldots, p_m\}$ — множество атомарных утверждений относительно значений переменных из $V \cup \_V$, $L\colon S \to 2^P$ — функция разметки состояний атомарными утверждениями, истинными в этих состояниях. *Полная* система переходов $cTS = \langle S, S_0, R_c, P, L\rangle$, где отношение переходов $R_c = S \times S$, является моделью поведения программы с абсолютно недетерминированным поведением всех её переменных из $V \cup \_V$. Отношение R_c допускает переходы между любыми двумя состояниями $s, s' \in S$.

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

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

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

Модель поведения программы TS получается в результате ограничений, накладываемых на полную систему переходов cTS. Ограничения задаются с помощью линейной темпоральной логики LTL (Linear Temporal Logic). Пусть $p \in P$, тогда синтаксис LTL-формул определяется грамматикой:

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

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:

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

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

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

Декларативная LTL-спецификация поведения переменной $v \in V$ имеет следующий вид [27]:

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

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. Выражения вида $v = \_v$ и $v = expr_i$ являются элементарными высказываниями из множества P, а выражения вида $cond_i$ — пропозициональными формулами.

Декларативная LTL-спецификация (1) поведения переменной $v \in V$ должна удовлетворять условию *изменчивости* значения переменной [27]:

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

т. е. при истинном условии $cond_i$ выражение $expr_i$ должно возвращать значение, отличное от предыдущего значения переменной v. Также для (1) должна выполняться *ортогональность* условий изменения значения переменной для любых $i, j = 1, \ldots, k$ при $i \neq j$ [27]:

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

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

При задании выражений $expr_1, \ldots, expr_k$ LTL-спецификация поведения переменной $v_i \in V$ должна удовлетворять условию *ограниченности*:

$$\mathbf{GX}\big(cond_1 \Rightarrow (val(expr_1) \in D_i)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow (val(expr_k) \in D_i)\big), \tag{4}$$

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.

Декларативная LTL-спецификация (1) задаёт строго детерминированное поведение переменной $v \in V$. Декларативная LTL-спецификация поведения программы — формула $\varphi = \varphi_{var} \wedge \varphi_0$, где φ_{var} — конъюнкция формул вида (1) для всех переменных из V (кроме входных) при соблюдении условий (2), (3) и (4), φ_0 — формула инициализации. LTL-спецификация φ должна удовлетворять условию *запрета мгновенных циклических зависимостей* [28]:

$$\forall v \in V\ [(v, v) \notin Dep^T], \tag{5}$$

где $Dep \subseteq V \times V$ — отношение непосредственной зависимости переменных, его транзитивное замыкание Dep^T задаёт отношение непосредственной и опосредованной зависимости переменных. Переменная v непосредственно зависит от переменной v', т. е. $(v, v') \in Dep$, если в декларативной LTL-спецификации (1) $cond_i$ или $expr_i$, где $i = 1, \ldots, k$, содержит переменную $v' \in V$.

Императивная LTL-спецификация поведения переменной $v \in V$ [27]:

$$\mathbf{GX}\big(cond_1 \Rightarrow (v = expr_1)\big) \wedge \ldots \wedge \mathbf{GX}\big(cond_k \Rightarrow (v = expr_k)\big) \wedge \\ \mathbf{GX}\big(\neg\, cond_1 \wedge \ldots \wedge \neg\, cond_k \Rightarrow (v = \_v)\big). \tag{6}$$

Данная спецификация описывает поведение переменной в императивном стиле «если…, то…» и является *конструктивной*, т. е. по ней непосредственно может быть построен код программы на императивном языке программирования. Декларативная и императивная LTL-спецификации эквивалентны [27], т. е. задают одно и то же программное поведение.

3. LTL-спецификация поведения ПЛУ

ПЛУ имеют булевы входные и выходные переменные, принимающие значения из множества $\{0, 1\}$. Рассмотрим частный случай декларативной LTL-спецификации (1). Пусть v — булева переменная, $expr_1 \equiv 1$ и $expr_2 \equiv 0$, тогда формула

$$\mathbf{GX}\big(\neg(v = \_v) \Rightarrow cond_1 \wedge (v = 1) \vee cond_2 \wedge (v = 0)\big) \wedge \\ \mathbf{GX}\big(\ \ (v = \_v) \Rightarrow \neg\,(cond_1 \vee cond_2)\ \ \big)$$

может быть переписана следующим образом:

$$\mathbf{GX}\big(\neg(\_v \wedge v \vee \neg\_v \wedge \neg v) \Rightarrow cond_1 \wedge v \vee cond_2 \wedge \neg v\big) \wedge \\ \mathbf{GX}\big(\ \ (\_v \wedge v \vee \neg\_v \wedge \neg v) \Rightarrow \neg\,(cond_1 \vee cond_2)\ \ \big).$$

Поскольку для логических выражений $cond_1$ и $cond_2$ должны выполняться условия изменчивости $cond_1 \Rightarrow \neg(\_v = 1)$ и $cond_2 \Rightarrow \neg(\_v = 0)$, рассмотрим эти выражения в виде $\neg\_v \wedge cnd_1$ и $\_v \wedge cnd_2$ соответственно, где cnd_1 и cnd_2 — логические выражения, выполнимость которых необходима для изменения переменной v без наложения на них условий изменчивости. Также выражения $\neg\_v \wedge cnd_1$ и $\_v \wedge cnd_2$ обеспечивают выполнение ортогональности (3), так как имеют вхождения переменной $\_v$ со знаком отрицания и без него соответственно. Получим LTL-формулу

$$\mathbf{GX}\big(\neg(\_v \wedge v \vee \neg\_v \wedge \neg v) \Rightarrow \neg\_v \wedge cnd_1 \wedge v \vee \_v \wedge cnd_2 \wedge \neg v\big) \wedge \\ \mathbf{GX}\big(\ \ (\_v \wedge v \vee \neg\_v \wedge \neg v) \Rightarrow \neg\,(\neg\_v \wedge cnd_1 \vee \_v \wedge cnd_2)\ \ \big).$$

Вынесем **GX** за скобки и избавимся от знака импликации:

$$\mathbf{GX}\Big(\big((\_v \wedge v) \vee (\neg\_v \wedge \neg v) \vee (\neg\_v \wedge v \wedge cnd_1) \vee (\_v \wedge \neg v \wedge cnd_2)\big) \wedge \\ \big((\_v \wedge \neg v) \vee (\neg\_v \wedge v) \vee (\neg\_v \wedge \neg cnd_1) \vee (\_v \wedge \neg cnd_2)\big)\Big).$$

Раскрыв скобки, получим

$$\mathbf{GX}\big((\neg\_v \wedge v \wedge cnd_1) \vee (\_v \wedge \neg v \wedge cnd_2) \vee (\neg\_v \wedge \neg v \wedge \neg cnd_1) \vee (\_v \wedge v \wedge \neg cnd_2)\big),$$

что эквивалентно формуле

$$\mathbf{GX}\big((\neg\_v \wedge v \Rightarrow cnd_1) \wedge (\_v \wedge \neg v \Rightarrow cnd_2) \wedge (\neg\_v \wedge \neg v \Rightarrow \neg cnd_1) \wedge (\_v \wedge v \Rightarrow \neg cnd_2)\big).$$

Внесём **GX** внутрь скобок, получим:

$$\mathbf{GX}\big(\neg\_v \wedge \ \ v \Rightarrow \ \ cnd_1\big) \wedge \\ \mathbf{GX}\big(\neg\_v \wedge \neg v \Rightarrow \neg cnd_1\big) \wedge \\ \mathbf{GX}\big(\ \ \_v \wedge \neg v \Rightarrow \ \ cnd_2\big) \wedge \\ \mathbf{GX}\big(\ \ \_v \wedge \ \ v \Rightarrow \neg cnd_2\big).$$

Внесём $\mathbf{X}$ внутрь скобок, учитывая, что $\mathbf{X}(\_v) = v$, получим декларативную LTL-спецификацию поведения булевой переменной v:

$$\begin{aligned}
&\mathbf{G}\big(\neg v \wedge \ \ \mathbf{X}(v) \Rightarrow \ \ \mathbf{X}(cnd_1)\big) \wedge \\
&\mathbf{G}\big(\neg v \wedge \neg\mathbf{X}(v) \Rightarrow \neg\mathbf{X}(cnd_1)\big) \wedge \\
&\mathbf{G}\big(\ \ v \wedge \neg\mathbf{X}(v) \Rightarrow \ \ \mathbf{X}(cnd_2)\big) \wedge \\
&\mathbf{G}\big(\ \ v \wedge \ \ \mathbf{X}(v) \Rightarrow \neg\mathbf{X}(cnd_2)\big).
\end{aligned} \tag{7}$$

Из (7) видно, что cnd_1 является *необходимым* условием для изменения значения переменной v с *False* на *True* (строка 1). Также данное условие является *достаточным*, что подтверждается формулой $\mathbf{G}\big(\neg v \wedge \mathbf{X}(cnd_1) \Rightarrow \mathbf{X}(v)\big)$, которая получена в результате контрапозиции во второй строке. Аналогично cnd_2 является необходимым и достаточным условием для обратного изменения значения переменной v с *True* на *False*. Таким образом, набор ограничений в (7) допускает только строго детерминированное поведение.

Отметим, что если $cnd_2 = \neg cnd_1$, то LTL-спецификацию (7) можно сократить. Получим *сокращённую* декларативную LTL-спецификацию поведения булевой переменной v:

$$\mathbf{G}\big(\ \mathbf{X}(v) \Leftrightarrow \mathbf{X}(cnd_1)\ \big). \tag{8}$$

Поведение всех булевых переменных ПЛУ (кроме входных) будем описывать с помощью декларативной LTL-спецификации — формулы (7) или (8).

Выполним контрапозицию в (7), получим:

$$\begin{aligned}
&\mathbf{G}\big(\neg v \wedge \neg\mathbf{X}(cnd_1) \Rightarrow \neg\mathbf{X}(v)\big) \wedge \\
&\mathbf{G}\big(\neg v \wedge \ \ \mathbf{X}(cnd_1) \Rightarrow \ \ \mathbf{X}(v)\big) \wedge \\
&\mathbf{G}\big(\ \ v \wedge \neg\mathbf{X}(cnd_2) \Rightarrow \ \ \mathbf{X}(v)\big) \wedge \\
&\mathbf{G}\big(\ \ v \wedge \ \ \mathbf{X}(cnd_2) \Rightarrow \neg\mathbf{X}(v)\big).
\end{aligned}$$

Сгруппируем отдельно подформулы в первой и третьей строках:

$$\begin{aligned}
&\mathbf{G}\big(\neg v \wedge \mathbf{X}(cnd_1) \Rightarrow \ \ \mathbf{X}(v)\big) \wedge \\
&\mathbf{G}\big(\ \ v \wedge \mathbf{X}(cnd_2) \Rightarrow \neg\mathbf{X}(v)\big) \wedge \\
&\mathbf{G}\big(\neg v \wedge \neg\mathbf{X}(cnd_1) \Rightarrow \neg\mathbf{X}(v)\big) \wedge \mathbf{G}\big(v \wedge \neg\mathbf{X}(cnd_2) \Rightarrow \mathbf{X}(v)\big).
\end{aligned}$$

После преобразования получим императивную LTL-спецификацию поведения булевой переменной:

$$\begin{aligned}
&\mathbf{G}\big(\neg v \wedge \mathbf{X}(cnd_1) \Rightarrow \ \ \mathbf{X}(v)\big) \wedge \\
&\mathbf{G}\big(\ \ v \wedge \mathbf{X}(cnd_2) \Rightarrow \neg\mathbf{X}(v)\big) \wedge \\
&\mathbf{G}\big(\neg[\neg v \wedge \mathbf{X}(cnd_1)] \wedge \neg[v \wedge \mathbf{X}(cnd_2)] \Rightarrow [v \Leftrightarrow \mathbf{X}(v)]\big).
\end{aligned} \tag{9}$$

Далее формулу (9) будем использовать для построения императивного кода программы.

Отметим, что внутренние переменные ПЛУ могут иметь любой тип — не только булев. В этом случае их поведение описывается с помощью декларативной LTL-спецификации общего вида (1).

4. LTL-спецификация поведения окружения ПЛУ

Согласно предлагаемому в данной работе подходу поведение окружения представляет собой поведение входных переменных ПЛУ и мнимых датчиков. Под мнимыми датчиками понимаются дополнительные булевы переменные, описывающие состояния компонентов из окружения. Цель

введения мнимых датчиков состоит в компенсации недостающих датчиков для отслеживания специфического поведения, которое необходимо учесть при задании реалистичного поведения входов ПЛУ. Как уже было отмечено, в настоящей работе мы ограничиваемся рассмотрением только ПЛУ — программ с булевыми входными и выходными переменными. Поведение входных переменных ПЛУ и мнимых датчиков является абсолютно недетерминированным, так как не описывается в декларативной LTL-спецификации. Их поведение будем задавать путем наложения ограничений на абсолютный недетерминизм.

Автоматная модель *абсолютного недетерминизма* булевой переменной представлена на рис. 1a. Из графа переходов видно, что переменная может принимать любые значения независимо от внешних условий. Автоматная модель *ограниченного недетерминизма* булевой переменной представлена на рис. 1b. Каждый из четырёх возможных переходов имеет ограничение: переход возможен только при выполнении указанного на нём условия $cond_{ij}$, выраженного в виде пропозициональной формулы, где i и j — исходное и новое значения булевой переменной соответственно. Например, если переменная изменила своё значение с *False* на *True*, то обязательно выполняется условие $cond_{01}$ в тот момент, когда значение переменной равно *True*. Таким образом, $cond_{ij}$ является только *необходимым* (не является *достаточным*) условием для совершения перехода.

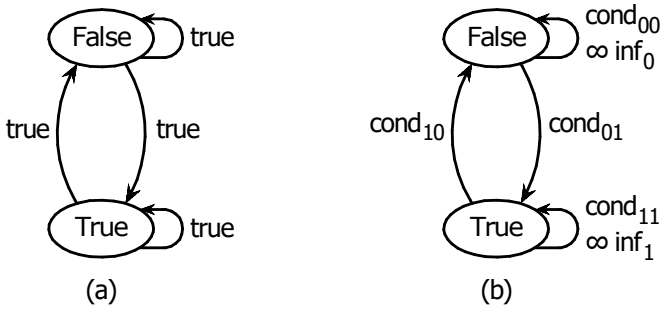

Fig. 1. Nondeterminism of a boolean variable: absolute (a), constrained (b)

Рис. 1. Недетерминизм булевой переменной: абсолютный (a), ограниченный (b)

На рис. 1b каждая петля имеет дополнительное условие inf_x, выраженное в виде LTL-формулы, где x — значение булевой переменной при совершении данного перехода. «Залипание» в состоянии x (бесконечное совершение перехода в это состояние по петле) разрешено только при выполнении условия inf_x. В связи с этим в графической нотации перед условием inf_x стоит знак бесконечности «∞». Например, если переменная изменила своё значение с *False* на *True* и начиная с этого момента истинна LTL-формула inf_1, то «залипание» в состоянии *True* разрешено. Таким образом, inf_x является *необходимым* условием для «залипания».

Формализуем вышеприведённое описание — LTL-спецификация *ограниченно недетерминированного* поведения булевой переменной имеет вид:

$$\begin{aligned}
&\mathbf{G}\big(\neg v \wedge \ \ \mathbf{X}(v) \Rightarrow \ \mathbf{X}(cond_{01})\big) \wedge \\
&\mathbf{G}\big(\neg v \wedge \neg\mathbf{X}(v) \Rightarrow \ \mathbf{X}(cond_{00})\big) \wedge \\
&\mathbf{G}\big(\ \ v \wedge \neg\mathbf{X}(v) \Rightarrow \ \mathbf{X}(cond_{10})\big) \wedge \\
&\mathbf{G}\big(\ \ v \wedge \ \ \mathbf{X}(v) \Rightarrow \ \mathbf{X}(cond_{11})\big) \wedge \\
&\quad \mathbf{G}\big(\mathbf{G}(\neg v) \Rightarrow \ inf_0\big) \wedge \\
&\quad \mathbf{G}\big(\mathbf{G}(\ \ v) \Rightarrow \ inf_1\big).
\end{aligned} \tag{10}$$

Первые четыре строки формулы описывают необходимые условия при совершении каждого из четырёх переходов. Отметим, что если, например, $cond_{01} \equiv 1$, то переход из *False* в *True* всегда разрешён,

если $cond_{01} \equiv 0$, то — всегда запрещён. Последние две строки формулы описывают необходимые условия при «залипании» в каждом из состояний. Например, последняя строка формулы утверждает, что всегда, если произошло «залипание» в состоянии *True* (всегда $v = True$), то LTL-формула inf_1 выполняется в начальный момент этого «залипания».

С помощью формулы (10) может быть описано как недетерминированное, так и детерминированное поведение булевой переменной. Если установить следующие ограничения: $cond_{00} \equiv \neg cond_{01}$, $cond_{11} \equiv \neg cond_{10}$, $inf_0 \equiv 1$, $inf_1 \equiv 1$, то получим декларативную LTL-спецификацию поведения булевой переменной (7), где $cnd_1 = cond_{01}$ и $cnd_2 = cond_{10}$. Таким образом, декларативная LTL-спецификацию поведения булевой переменной (7) является частным случаем LTL-спецификации ограниченно недетерминированного поведения (10).

Далее формулу (10) будем использовать для описания поведения окружения ПЛУ. При этом рекомендуется условия $cond_{00}$, $cond_{01}$, $cond_{10}$, $cond_{11}$, inf_0, inf_1 задавать максимально *слабыми*, чтобы LTL-спецификация допускала как можно больше возможных путей, пусть даже не существующих. В этом случае поведение КФС будет иметь больше путей, чем в действительности. Если LTL-спецификация КФС задаёт множество путей Π^+, а фактическое множество путей $\Pi \subset \Pi^+$, то из $\Pi^+ \models \psi$ следует $\Pi \models \psi$, где ψ — проверяемое свойство. Таким образом, не нужно стремиться описывать максимально реалистичную модель, нужно вводить только самые очевидные ограничения, чтобы захватить как можно больше путей и отбросить как можно меньше. А при попытке описания максимально реалистичной модели поведения возникает риск отбросить пути, входящие в Π — получим множество $\Pi^- \subset \Pi$. В этом случае верификация будет проводиться на некорректной модели, что ставит под сомнения полученный результат, так как из $\Pi^- \models \psi$ не следует $\Pi \models \psi$.

5. LTL-спецификация поведения установки для литья пластмасс

5.1. Описание установки для литья пластмасс

В качестве примера рассмотрим LTL-спецификацию поведения замкнутой системы «ПЛУ-ОУ» — установки для литья пластмасс, схема которой представлена на рис. 2. Гранулированный пластик хранится в бункере (Storage Bunker for Granular Plastic) и с помощью шнекового механизма подачи (Feed Mechanism) транспортируется в дозатор (Dosing Unit) при открытой крышке (Lid). Дозатор имеет электрический нагреватель (*Heater*) для плавления гранул пластика и весоизмерительную платформу (Weighing Platform) для определения собственного веса. При открытом клапане (*Valve*) расплавленный пластик из дозатора по сливному трубопроводу подаётся в литейную форму (Form). С помощью конвейера (Conveyor) осуществляется транспортировка формы справа налево.

Система управления установкой построена на базе ПЛК — его УПО координирует работу оборудования. Оператор взаимодействует с установкой посредством кнопок и ламп, расположенных на панели. В составе УПО выделим ПЛУ, интерфейс которой представлен на рис. 3.

Интерфейс взаимодействия с панелью оператора. Вход *PBStart* принимает команду запуска установки (Plant Start Command) от кнопки (Button), расположенной на панели (Panel), вход *PBStop* — команду немедленного останова установки (Plant Stop Command), вход *PBCompl* — команду плавного завершения работы установки (Plant Complete Work Command), вход *PBConvr* — команду включения конвейера (Conveyor Turn On Command) в ручном режиме.

Также панель оператора содержит ряд ламп (Lamps) для индикации текущих режимов работы и состояний. Выход *SysOn* сигнализирует о включении системы (System is On Mode) в автоматическом режиме, выход *Compl* — о процессе плавного завершения работы (Completion Mode), выход *FErr* — об ошибке подачи пластика в дозатор (Feed Error), выход *CErr* — об ошибке работы конвейера (Conveyor Error), выход *HErr* — об ошибке работы нагревателя (Heater Error), выход *Disch* — о режиме разгрузки пластика из дозатора (Plastic Discharge Mode), выход *Mltng* — о режиме плавлении пластика (Plastic Melting Mode), выход *Mlted* — о завершении плавления пластика (Plastic is Melted).

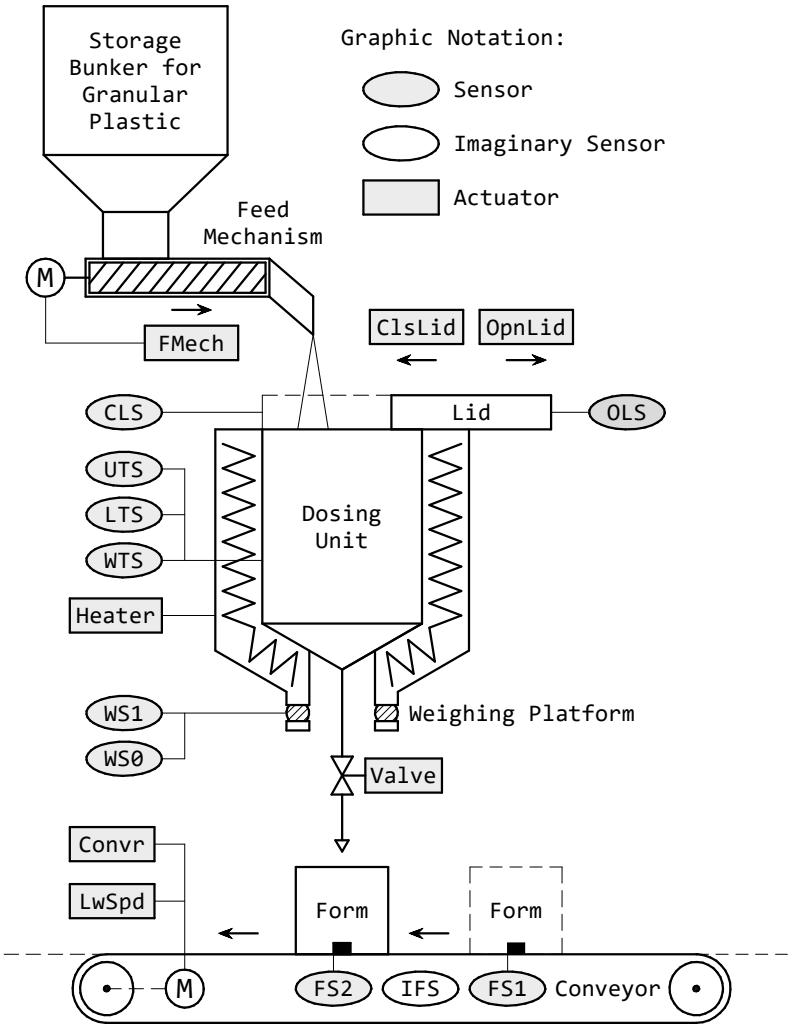

Fig. 2. Plastic molding plant diagram

Рис. 2. Схема установки для литья пластмасс

Интерфейс взаимодействия с установкой. Установка (Plant) оснащена датчиками (Sensors) и исполнительными устройствами (Actuators). Их расположение изображено на рис. 2, а названия совпадают с соответствующими входными/выходными сигналами ПЛУ. Датчики изображены в виде серых овалов, исполнительные устройства — в виде серых прямоугольников. Вход ПЛУ *FS1* принимает сигнал от датчика наличия формы на первой позиции (Form on Position 1) конвейера, вход *FS2* — от датчика наличия формы на второй позиции (Form on Position 2) конвейера, вход *OLS* — от датчика открытой крышки (Lid is Opened), вход *CLS* — от датчика закрытой крышки (Lid is Closed), вход *WS0* — от датчика веса о том, что дозатор не пуст (Dosing Unit is not Empty), вход *WS1* — от датчика веса о том, что дозатор заполнен (Dosing Unit is Full), вход *UTS* — от датчика высокой температуры дозатора (Dosing Unit Temperature at the Up Level), вход *LTS* — от датчика низкой температуры дозатора (Dosing Unit Temperature at the Low Level), вход *WTS* — от датчика рабочей температуры дозатора (Dosing Unit Temperature at the Working Level). Пластик плавится при рабочей температуре. Низкая и высокая температуры выше рабочей и предназначены для управления нагревателем.

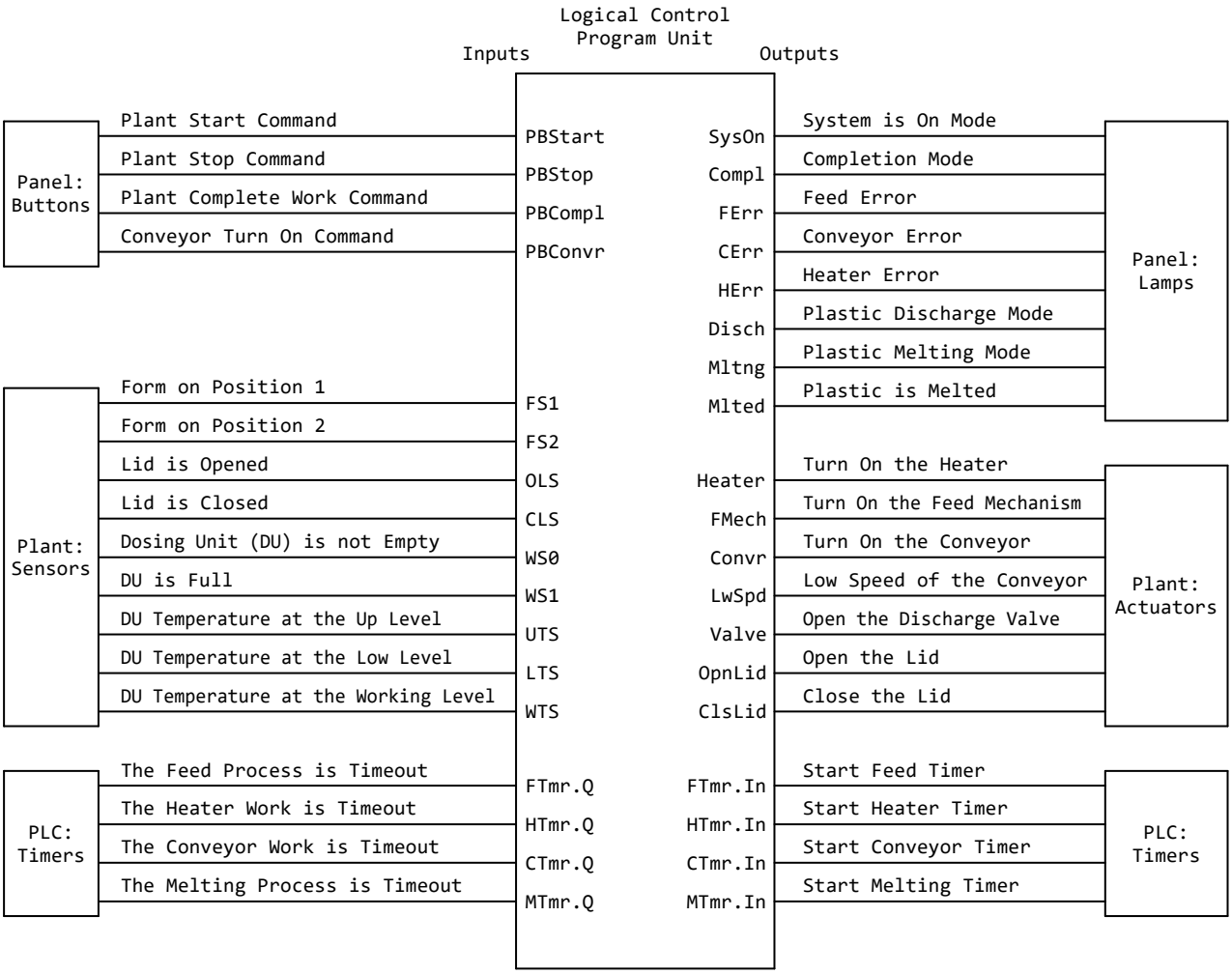

Fig. 3. Logical control program interface

Рис. 3. Интерфейс программы логического управления

Выход ПЛУ *Heater* подаёт сигнал включения нагревателя (Turn On the Heater), выход *FMech* — сигнал включения механизма подачи (Turn On the Feed Mechanism) пластика, выход *Convr* — сигнал включения конвейера (Turn On the Conveyor), выход *LwSpd* — сигнал установки пониженной скорости конвейера (Low Speed of the Conveyor), выход *Valve* — сигнал открытия выпускного клапана (Open the Discharge Valve), выход *OpnLid* — сигнал открытия крышки (Open the Lid) дозатора, выход *ClsLid* — сигнал закрытия крышки (Close the Lid) дозатора.

Интерфейс взаимодействия с таймерами. Для работы с временными интервалами УПО ПЛК (PLC) содержит таймеры (Timers). Вход ПЛУ *FTmr.Q* принимает сигнал от таймера об истечении времени процесса подачи (The Feed Process is Timeout), вход *HTmr.Q* — сигнал об истечении времени прогрева нагревателя (The Heater Work is Timeout), вход *CTmr.Q* — сигнал об истечении времени ожидания подачи формы конвейером (The Conveyor Work is Timeout), вход *MTmr.Q* — сигнал об истечении времени процесса плавления (The Melting Process is Timeout). Выходы ПЛУ *FTmr.In*, *HTmr.In*, *CTmr.In* и *MTmr.In* предназначены для запуска вышеуказанных таймеров. Таким образом, ПЛУ получает сигналы обратной связи от датчиков установки и таймеров ПЛК.

Работа установки. При получении команды запуска установки (*PBStart*) включается регулирование температуры и начинается работа оборудования по циклу. Регулятор температуры работает вне цикла и поддерживает рабочую температуру (*WTS*): включает нагреватель (*Heater*) при понижении температуры ($\neg LTS$), выключает ($\neg Heater$) — при повышении (*UTS*), где $\neg LTS$ означает $LTS = False$. Если работа нагревателя не позволяет достичь рабочей температуры (*WTS*) в течение заданного времени или произошло снижение температуры ($\neg WTS$) при включенном нагревателе, то устанавливается ошибка его работы (*HErr*).

Рассмотрим цикл работы оборудования установки. Изначально включается механизм подачи (*FMech*) и поддерживается его работа до тех пор, пока не сработает датчик веса (*WS1*), сигнализирующий о заполнении дозатора. Если время процесса подачи вышло ($FTmr.Q$) и дозатор не заполнен, то выдается сигнал ошибки (*FErr*) и работа останавливается. Данная ошибка может возникнуть из-за пустого бункера или поломки механизма подачи.

Параллельно с механизмом подачи включается конвейер (*Convr*), который транспортирует форму в направлении справа налево к месту для литья. Каждая форма оснащена меткой (чёрный прямоугольник на рис. 2), на которую реагируют датчики *FS1* и *FS2*. При достижении формой (выделена штриховой линией) первой позиции (*FS1*) конвейер снижает скорость (*LwSpd*). Далее транспортировка продолжается на пониженной скорости для повышения точности координаты остановки. Конвейер останавливается ($\neg Convr$), когда форма (выделена основной линией) достигнет второй позиции (*FS2*). Если время работы конвейера вышло ($CTmr.Q$) и форма не на второй позиции ($\neg FS2$), то выдается сигнал ошибки (*CErr*) и работа установки останавливается. Предполагается, что конвейер всегда работает исправно и причиной ошибки *CErr* может быть только отсутствие формы на ленте конвейера.

После заполнения дозатора его крышка закрывается (*ClsLid*) и активируется разгрузочный режим (*Disch*). Если крышка закрыта (*CLS*) и дозатор имеет рабочую температуру (*WTS*), то запускается таймер отсчёта времени плавления ($MTmr.In$). По истечении данного времени ($MTmr.Q$) если форма находится на рабочей позиции (*FS2*), то открывается клапан (*Valve*) для её заполнения. После опустошения дозатора ($\neg WS0$) клапан закрывается ($\neg Valve$), на нормальной скорости ($\neg LwSpd$) включается конвейер (*Convr*) и освобождает место для новой формы. Также закрытие клапана ($\neg Valve$) влечёт открытие крышки (*OpnLid*) дозатора. После её открытия (*OLS*) цикл работы повторяется.

О работе установки в автоматическом режиме свидетельствует сигнал *SysOn*. При наличии любой ошибки (*FErr*, *CErr*, *HErr*) или нажатии кнопки останова (*PBStop*) работа установки немедленно прекращается. Повторное нажатие кнопки *PBStop* приводит к сбросу сигналов об ошибках. Нажатие кнопки включения конвейера (*PBConvr*) запускает его (*Convr*) в ручном режиме и отключает автоматический режим работы установки ($\neg SysOn$). Кнопка плавного завершения работы (*PBCompl*) устанавливает специальный режим (*Compl*), который завершает текущий цикл и не начинает новый.

5.2. LTL-спецификация поведения ПЛУ

Рассмотрим декларативную LTL-спецификацию поведения переменной *Heater*, которая управляет работой нагревателя:

$$\begin{aligned}
&\mathbf{G}\big(\neg Heater \wedge \ \ \mathbf{X}(Heater) \Rightarrow \ \ (\mathbf{X}(SysOn) \wedge \neg\mathbf{X}(LTS))\big) \wedge \\
&\mathbf{G}\big(\neg Heater \wedge \neg\mathbf{X}(Heater) \Rightarrow \ \neg(\mathbf{X}(SysOn) \wedge \neg\mathbf{X}(LTS))\big) \wedge \\
&\mathbf{G}\big(\ \ Heater \wedge \neg\mathbf{X}(Heater) \Rightarrow \ \ (\neg\mathbf{X}(SysOn) \vee \mathbf{X}(UTS))\big) \wedge \\
&\mathbf{G}\big(\ \ Heater \wedge \ \ \mathbf{X}(Heater) \Rightarrow \ \neg(\neg\mathbf{X}(SysOn) \vee \mathbf{X}(UTS))\big).
\end{aligned} \tag{11}$$

Включение нагревателя ($\neg Heater \wedge \mathbf{X}(Heater)$) происходит при работе системы (*SysOn*) и относительно низком уровне температуры (отсутствует сигнал *LTS*). Нагреватель выключается ($Heater \wedge$

$\neg\mathbf{X}(Heater)$) при отключении системы (*SysOn*) или при достижении высокого уровня температуры (присутствует сигнал *UTS*).

Аналогичным образом зададим поведение всех переменных программы в виде декларативной LTL-спецификации в синтаксисе инструмента nuXmv (см. приложение 1). Отметим, что для описания поведения программы была введена одна дополнительная внутренняя булева переменная *Fin*, которая сигнализирует о необходимости завершения процесса работы. Она получает истинное значение, если либо был активен режим плавного завершения (*Compl*) при открытой заслонке (*OLS*) и опустошённом дозаторе ($\neg$*WS0*), либо выставлена хотя бы одна из ошибок *FErr*, *HErr*, *CErr*, либо нажата кнопка *PBStop* или кнопка *PBConvr*. В противном случае сигнал *Fin* сбрасывается.

5.3. LTL-спецификация поведения окружения ПЛУ

Рассмотрим построение LTL-спецификации поведения для датчиков конвейера. Предполагается, что поведение датчиков положения формы *FS1* и *FS2* взаимосвязано: датчик *FS2* не может сработать, если до этого не сработал датчик *FS1*. Из рис. 2 видно, что форма может находиться в промежуточном положении между двумя датчиками, когда ни один из них не выдает сигнала о наличии формы. Таким образом, для описания поведения датчика *FS2* невозможно использовать только текущие показания датчика *FS1* — этого недостаточно, нужно запоминать, что датчик *FS1* сработал в прошлом до момента срабатывания датчика *FS2*. Для этой цели введём мнимый датчик наличия формы *IFS* (Imaginary Form Sensor) между датчиками *FS1* и *FS2*. Мнимый датчик *IFS* изображён на рис. 2 в виде овала с незакрашенным фоном. Данный датчик будет использоваться при описании поведения датчика *FS2*: датчик *FS2* не может сработать, если в этот момент не исчезает сигнал датчика *IFS*. Формально опишем поведение датчика *FS2* согласно (10), получим:

$$
\begin{aligned}
1:&\quad \mathbf{G}\big(\neg FS2 \wedge \ \mathbf{X}(FS2) \Rightarrow\ IFS \wedge \neg\mathbf{X}(IFS) \wedge \neg FS1 \wedge \neg\mathbf{X}(FS1) \wedge Convr\big) \wedge\\
2:&\quad \mathbf{G}\big(\neg FS2 \wedge \neg\mathbf{X}(FS2) \Rightarrow\ true\big) \wedge\\
3:&\quad \mathbf{G}\big(\ FS2 \wedge \neg\mathbf{X}(FS2) \Rightarrow\ \neg IFS \wedge \neg\mathbf{X}(IFS) \wedge \neg FS1 \wedge \neg\mathbf{X}(FS1) \wedge Convr\big) \wedge\\
4:&\quad \mathbf{G}\big(\ FS2 \wedge \ \mathbf{X}(FS2) \Rightarrow\ \neg IFS \wedge \neg\mathbf{X}(IFS) \wedge \neg FS1 \wedge \neg\mathbf{X}(FS1)\big) \wedge\\
5:&\quad \mathbf{G}\big(\mathbf{G}(\neg FS2) \Rightarrow\ true\big) \wedge\\
6:&\quad \mathbf{G}\big(\mathbf{G}(\ FS2) \Rightarrow\ \mathbf{F}(\mathbf{G}(\neg Convr))\big).
\end{aligned}
\tag{12}
$$

Первая строка формулы (12) описывает необходимые условия для включения датчика *FS2*: датчик *IFS* был включен, а сейчас отключён, датчик *FS1* был отключён и сейчас отключён, конвейер (*Convr*) был включен в предыдущий момент времени. При выполнении данных условий датчик *FS2* может включиться, а может и остаться в выключенном состоянии, так как непосредственно на этот переход не накладывается никаких ограничений (строка 2). При отключении датчика *FS2* также обязательно должен работать конвейер в предыдущий момент времени (строка 3). Это отражает тот факт, что работа конвейера является причиной, а смена состояния датчика следствием. Причина и следствие не могут быть одновременны — следствие проявляется после причины.

«Залипание» датчика *FS2* в состоянии *False* допустимо при любых условиях (строка 5). Оно может произойти из-за отсутствия формы. Таким образом, форма никогда не достигнет положения для срабатывания датчика *FS2*. «Залипание» датчика *FS2* в состоянии *True* разрешено, только если конвейер в какой-то момент остановится и больше никогда не включится (строка 6). Таким образом, форма никогда не сможет покинуть положение срабатывания датчика *FS2*. В итоге формула (12) описывает возможное поведение сигнала обратной связи *FS2* при управляющем воздействии на конвейер.

Поведение мнимого датчика *IFS* имеет вид:

$$
\begin{aligned}
1:&\quad \mathbf{G}\big(\neg IFS \wedge \ \mathbf{X}(IFS) \Rightarrow\ \neg FS2 \wedge \neg\mathbf{X}(FS2) \wedge FS1 \wedge \neg\mathbf{X}(FS1) \wedge Convr\big) \wedge\\
2:&\quad \mathbf{G}\big(\neg IFS \wedge \neg\mathbf{X}(IFS) \Rightarrow\ true\big) \wedge\\
3:&\quad \mathbf{G}\big(\ \ IFS \wedge \neg\mathbf{X}(IFS) \Rightarrow\ \neg FS2 \wedge\ \ \mathbf{X}(FS2) \wedge \neg FS1 \wedge \neg\mathbf{X}(FS1) \wedge Convr\big) \wedge\\
4:&\quad \mathbf{G}\big(\ \ IFS \wedge\ \ \mathbf{X}(IFS) \Rightarrow\ \neg FS2 \wedge \neg\mathbf{X}(FS2) \wedge \neg FS1 \wedge \neg\mathbf{X}(FS1)\big) \wedge\\
5:&\quad \mathbf{G}\big(\mathbf{G}(\neg IFS) \Rightarrow\ true\big) \wedge\\
6:&\quad \mathbf{G}\big(\mathbf{G}(\ \ IFS) \Rightarrow\ \mathbf{F}(\mathbf{G}(\neg Convr))\big).
\end{aligned}
\tag{13}
$$

При включении датчика *IFS* происходит отключение датчика *FS1*, датчик *FS2* не меняет своего состояния — остается отключённым (строка 1). Отключение датчика *IFS* сопровождается включением датчика *FS2*, датчик *FS1* остается отключённым (строка 3). Условия для «залипаний» датчиков *IFS* и *FS2* совпадают (строки 5 и 6).

Поведение оставшегося датчика конвейера *FS1* имеет вид:

$$
\begin{aligned}
1:&\quad \mathbf{G}\big(\neg FS1 \wedge \ \mathbf{X}(FS1) \Rightarrow\ \neg FS2 \wedge \neg\mathbf{X}(FS2) \wedge \neg IFS \wedge \neg\mathbf{X}(IFS) \wedge Convr\big) \wedge\\
2:&\quad \mathbf{G}\big(\neg FS1 \wedge \neg\mathbf{X}(FS1) \Rightarrow\ true\big) \wedge\\
3:&\quad \mathbf{G}\big(\ \ FS1 \wedge \neg\mathbf{X}(FS1) \Rightarrow\ \neg FS2 \wedge \neg\mathbf{X}(FS2) \wedge \neg IFS \wedge\ \ \mathbf{X}(IFS) \wedge Convr\big) \wedge\\
4:&\quad \mathbf{G}\big(\ \ FS1 \wedge\ \ \mathbf{X}(FS1) \Rightarrow\ \neg FS2 \wedge \neg\mathbf{X}(FS2) \wedge \neg IFS \wedge \neg\mathbf{X}(IFS)\big) \wedge\\
5:&\quad \mathbf{G}\big(\mathbf{G}(\neg FS1) \Rightarrow\ true\big) \wedge\\
6:&\quad \mathbf{G}\big(\mathbf{G}(\ \ FS1) \Rightarrow\ \mathbf{F}(\mathbf{G}(\neg Convr))\big).
\end{aligned}
\tag{14}
$$

При включении датчика *FS1* конвейер работал в предыдущий момент времени, датчики *FS2* и *IFS* остаются в выключенном состоянии (строка 1). При отключении датчика *FS1* также работал конвейер, датчик *FS2* остаётся в выключенном состоянии, датчик *IFS* включается (строка 3).

Отметим, что заданная LTL-спецификация поведения группы датчиков *FS1*, *FS2*, *IFS* задаёт мгновенные циклические зависимости между ними, что запрещено в декларативной LTL-спецификации (1) и её частном случае (7). Например, из (14) в строке 1 видно, что включение датчика *FS1* зависит от текущего состояния датчика *FS2*, который должен быть выключен ($\neg\mathbf{X}(FS2)$). А из (12) в строке 1 видно, что включение датчика *FS2* также зависит от текущего состояния датчика *FS1*, который тоже должен быть выключен ($\neg\mathbf{X}(FS1)$). Таким образом, формулы (14) и (12) задают мгновенную циклическую зависимость между датчиками *FS1* и *FS2*. LTL-спецификация ограниченно недетерминированного поведения (10) допускает такой способ описания зависимостей — это обеспечивает дополнительные выразительные возможности при задании поведения окружения. Запрет на мгновенные циклические зависимости (5) в декларативной LTL-спецификации введён для возможности её трансляции в последовательно исполняемый код на императивном языке программирования.

Поведение трёх датчиков конвейера было описано согласно шаблону (10), который детально отражает все условия для переходов и «залипаний». Однако при задании поведения взаимосвязанной группы переменных может возникнуть противоречие или избыточность. Для исключения данных проблем предлагается в процессе разработки LTL-спецификации выполнять анализ связанной группы переменных с помощью инструмента для работы с LTL-формулами Spot. Все LTL-формулы в (10), которые имеют значение *True* в правой части импликации, избыточны, так как являются тождественно истинными и не накладывают никаких ограничений на поведение, поэтому могут быть исключены из LTL-спецификации — это строки 2 и 5 формул (12), (13), (14). После их удаления получим эквивалентную LTL-спецификацию поведения φ_{sens} группы датчиков *FS1*, *FS2*, *IFS*. Далее с помощью инструмента Spot было выявлено, что строка 4 в (12) является логическим следствием

φ_{sens} при заданных начальных условиях: $\neg FS1 \wedge \neg FS2 \wedge \neg IFS$. Таким образом, строку 4 можно исключить из LTL-спецификации φ_{sens}. Аналогичным образом строка 4 формул (13) и (14) может быть исключена из φ_{sens}.

В итоге формулы (12), (13), (14) содержат полную (избыточную) информацию по каждому датчику *FS2*, *IFS* и *FS1* соответственно, а сокращённая спецификация φ_{sens} — только необходимую информацию для всей группы датчиков. Модификация спецификации поведения отдельно взятого датчика может привести к кардинальному изменению сокращённой спецификации φ_{sens}. Поэтому рекомендуется разрабатывать и хранить полную спецификацию, а сокращённую использовать при верификации.

Зададим LTL-спецификацию ограниченно недетерминированного поведения окружения программы в синтаксисе инструмента nuXmv (см. приложение 2). Отметим, что на поведение кнопок не накладывается никаких ограничений, поэтому выполняется только их инициализация. Поведение всех датчиков и таймеров описано вместе с их инициализацией.

Рассмотрим поведение таймера $MTmr.Q$ из данной спецификации:

$$
\begin{aligned}
0:&\quad \neg MTmr.Q\\
1:&\quad \mathbf{G}\big(\neg MTmr.Q \wedge \mathbf{X}(MTmr.Q) \Rightarrow MTmr.In\big) \wedge\\
2:&\quad \mathbf{G}\big(\neg MTmr.Q \wedge \neg\mathbf{X}(MTmr.Q) \Rightarrow true\big) \wedge\\
3:&\quad \mathbf{G}\big(MTmr.Q \wedge \neg\mathbf{X}(MTmr.Q) \Rightarrow \neg MTmr.In\big) \wedge\\
4:&\quad \mathbf{G}\big(MTmr.Q \wedge \mathbf{X}(MTmr.Q) \Rightarrow MTmr.In\big) \wedge\\
5:&\quad \mathbf{G}\big(\mathbf{G}(\neg MTmr.Q) \Rightarrow \mathbf{F}(\mathbf{G}(\neg MTmr.In)) \vee \mathbf{G}(MTmr.In \Rightarrow \mathbf{F}(\neg MTmr.In))\big) \wedge\\
6:&\quad \mathbf{G}\big(\mathbf{G}(MTmr.Q) \Rightarrow \mathbf{G}(MTmr.In)\big).
\end{aligned}
\tag{15}
$$

Изначально выход таймера выключен (строка 0). При включении таймера его вход должен быть установлен в предыдущий момент времени (строка 1). Сигнал $MTmr.In$ является управляющим воздействием на таймер, а $MTmr.Q$ — сигналом обратной связи. Таким образом, таймер — такой же объект управления для ПЛУ, как и технологическое оборудование. При наличии входного сигнала таймер может и не включиться (строка 2). Вместе строки 1 и 2 задают недетерминированное включение таймера. При отключении таймера его вход должен быть сброшен в предыдущий момент времени (строка 3), в противном случае таймер продолжает оставаться включённым (строка 4). Строки 3 и 4 задают строго детерминированное отключение таймера.

«Залипание» таймера в отключенном состоянии возможно, если начиная с какого-то момента времени в будущем вход таймера больше никогда не устанавливается или всегда после его установки в будущем следует сброс (строка 5). Таким образом, таймер либо больше не работает, либо не успевает выдержать заданный интервал времени, поэтому может никогда не установить свой выход. Разрешим «залипание» таймера во включенном состоянии при «залипании» его входного сигнала (строка 6). На самом деле данное условие является лишним — с помощью инструмента Spot было проверено, что строка 6 является логическим следствие строки 4. В итоге строки 2 и 6 исключаются при построении сокращённой LTL-спецификации поведения таймера.

6. LTL-спецификация свойств

Зададим требуемый набор свойств установки для литья пластмасс. Свойства $P1, \ldots, P15$ и их LTL-формализация представлены в таблице 1, свойства $P16, \ldots, P28$ — в таблице 2. В целом спецификация свойств $\mathbb{P} = \{P1, \ldots, P28\}$ построена на основе шаблонов: (1) устанавливающих безопасные состояния группы устройств, (2) нацеленных на проверку возможности изменения значений переменных, (3) накладывающих ограничения на порядок работы устройств в группе, (4) определяющих поведение устройств в некотором режиме. Модель поведения ПЛУ установки должна удовлетворять спецификации свойств $\mathbb{P}$.

Table 1. Properties $P1 \ldots P15$ **Таблица 1.** Свойства $P1 \ldots P15$

Св-во	Описание и формализация
$P1$	$\mathbf{G}(SysOn \Rightarrow \neg(PBStop \vee PBConvr \vee FErr \vee CErr \vee HErr))$
	Всегда, если система включена, то неверно, что нажата кнопка экстренного останова, кнопка запуска конвейера или есть ошибки (подачи, конвейера или нагревателя).
$P2$	$\mathbf{G}(\neg SysOn \Rightarrow \neg(Valve \vee FMech \vee Heater \vee OpnLid \vee ClsLid \vee Compl) \wedge (Convr \Rightarrow PBConvr))$
	Всегда при выключенной системе не работает ни одно из устройств (клапан, механизм подачи, нагреватель, привод крышки дозатора), кроме конвейера, а также не активен режим плавного завершения; работа конвейера допускается только по команде оператора.
$P3$	$\mathbf{G}(FMech \Rightarrow \neg Valve \wedge \neg OpnLid \wedge \neg ClsLid) \wedge \mathbf{G}(Valve \Rightarrow \neg FMech \wedge \neg OpnLid \wedge \neg ClsLid) \wedge \mathbf{G}(OpnLid \Rightarrow \neg FMech \wedge \neg Valve \wedge \neg ClsLid) \wedge \mathbf{G}(ClsLid \Rightarrow \neg FMech \wedge \neg Valve \wedge \neg OpnLid)$
	Взаимное исключение работы устройств: всегда, если работает указанное устройство, то все остальные устройства группы не работают.
$P4$	$\boxed{\mathbf{G}\neg(Convr \wedge Valve)}$ Никогда форма не заливается во время её перемещении: конвейер и разливной клапан никогда не работают одновременно.
$P5$	$\mathbf{G}(\neg(OpnLid \vee ClsLid) \Rightarrow (CLS \vee OLS \vee \neg SysOn))$
	Всегда, если крышка дозатора не перемещается (открывается или закрывается), то она находится в крайнем положении (открыта или закрыта) или система отключена.
$P6$	$\boxed{\mathbf{G}(WS1 \Rightarrow \neg FMech)}$ Дозатор никогда не переполняется: всегда, если дозатор заполнен, то подача пластика отключена.
$P7$	$\boxed{\mathbf{G}(Valve \Rightarrow (FS2 \wedge \neg Convr))}$ Пластик никогда не разливается на конвейер: всегда, если разливной клапан открыт, то форма на позиции и конвейер остановлен.
$P8$	$\mathbf{G}([FErr \wedge \neg\mathbf{X}(FErr) \vee HErr \wedge \neg\mathbf{X}(HErr) \vee CErr \wedge \neg\mathbf{X}(CErr)] \Rightarrow \mathbf{X}(PBStop) \wedge \neg\mathbf{X}(SysOn))$
	Организация сброса ошибок в программе: всегда, если ошибка сброшена, то была нажата кнопка останова и система отключена.
$P9$	$\mathbf{G}(\neg Valve \wedge FS2 \wedge WS0 \Rightarrow \neg[(FS2 \wedge WS0)\ \mathbf{U}\ (Convr \wedge \neg PBConvr \wedge FS2 \wedge WS0)])$
	Весь пластик дозатора заливается в одну форму: всегда, если заливка формы была прервана (дозатор ещё не пуст), то до включения конвейера дозатор будет опустошён.
$P10$	$\mathbf{G}(Valve \wedge \mathbf{X}(\neg Valve) \wedge \mathbf{X}(FS2) \wedge \neg\mathbf{X}(WS0) \Rightarrow \neg\mathbf{X}(FS2\ \mathbf{U}\ (Valve \wedge FS2)))$
	Никогда не происходит переполнения формы: всегда, если клапан закрывается, разгрузив пластик из дозатора в форму, то он не откроется повторно при наличии данной формы.
$P11$	$\mathbf{G}(Valve \wedge \mathbf{X}(\neg Valve) \wedge \mathbf{X}(FS2) \wedge \neg\mathbf{X}(WS0) \Rightarrow \neg\mathbf{X}(\neg Convr\ \mathbf{U}\ Valve))$
	Транспортировка формы конвейером обязательна между двумя заливками формы: всегда, если клапан закрывается, разгрузив пластик из дозатора в форму, то неверно, что конвейер будет выключен до следующего открытия клапана.
$P12$	$\mathbf{G}(\neg Valve \wedge \neg WS1 \wedge OLS \Rightarrow \neg[\neg WS1\ \mathbf{U}\ (Valve \wedge \neg WS1)])$
	Всегда, если дозатор не заполнен, то разливной клапан не откроется до его заполнения.
$P13$	$\boxed{\mathbf{G}(\neg FS2 \wedge \mathbf{X}(FS2) \Rightarrow LwSpd)}$ Всегда, если сработал датчик наличия формы на второй позиции, то конвейер транспортировал форму на пониженной скорости.
$P14$	$\boxed{\mathbf{G}(\neg FS1 \wedge \mathbf{X}(FS1) \Rightarrow \neg LwSpd)}$ Всегда, если сработал датчик наличия формы на первой позиции, то конвейер транспортировал форму на нормальной (не пониженной) скорости.
$P15$	$\mathbf{G}(OpnLid \wedge \neg\mathbf{X}(OpnLid) \wedge \mathbf{X}(SysOn) \Rightarrow \neg FMech \wedge \mathbf{X}(FMech)) \wedge \mathbf{G}(FMech \wedge \neg\mathbf{X}(FMech) \wedge \mathbf{X}(SysOn) \Rightarrow \neg ClsLid \wedge \mathbf{X}(ClsLid)) \wedge \mathbf{G}(Valve \wedge \neg\mathbf{X}(Valve) \wedge \mathbf{X}(SysOn) \Rightarrow \neg Convr \wedge \mathbf{X}(Convr))$
	Последовательность работы устройств: всегда при включённой системе после открытия заслонки включается механизм подачи, после механизма подачи заслонка закрывается, после закрытия клапана включается конвейер.

Table 2. Properties $P16 \ldots P28$ **Таблица 2.** Свойства $P16 \ldots P28$

Св-во	Описание и формализация
$P16$	$\mathbf{G}(FTmr.Q \Rightarrow \mathbf{X}(\neg FTmr.Q)) \wedge \mathbf{G}(HTmr.Q \Rightarrow \mathbf{X}(\neg HTmr.Q)) \wedge \mathbf{G}(CTmr.Q \Rightarrow \mathbf{X}(\neg CTmr.Q))$
	Организация работы с таймерами в программе: всегда, если указанный таймер сработал, то в следующий момент времени он отключен.
$P17$	$\mathbf{G}(Convr \wedge \neg FS2 \wedge \mathbf{X}(FS2) \Rightarrow \mathbf{X}(\neg PBConvr \Rightarrow \neg Convr))$
	Всегда, если при работе конвейера срабатывает датчик наличия формы на второй позиции, то в этот момент конвейер отключается, если не нажата кнопка запуска конвейера.
$P18$	$\mathbf{G}(\neg FS2 \wedge \mathbf{X}(FS2) \wedge \mathbf{X}(SysOn) \Rightarrow \neg\mathbf{X}((SysOn \wedge \neg Valve)\ \mathbf{U}\ (SysOn \wedge Convr)))$
	В каждую остановленную форму подаётся пластик: при постоянно работающей системе всегда, если форма появилась под дозатором, то будет открыт разливной клапан до включения конвейера (не верно, что клапан будет закрыт до включения конвейера).
$P19$	$\mathbf{G}(Valve \wedge \neg\mathbf{X}(Valve) \wedge \mathbf{X}(SysOn) \Rightarrow \neg\mathbf{X}((SysOn \wedge \neg FMech)\ \mathbf{U}\ (SysOn \wedge Valve)))$
	Между литьём форм происходит заполнение дозатора: при постоянно работающей системе всегда, если разливной клапан закрылся, то неверно, что механизм подачи пластика в дозатор будет выключен до следующего открытия клапана.
$P20$	$\mathbf{G}(ClsLid \wedge \neg\mathbf{X}(ClsLid) \wedge \mathbf{X}(SysOn) \Rightarrow \mathbf{X}(\neg OpnLid\ \mathbf{U}\ (Valve \vee \neg SysOn))) \wedge$ $\mathbf{G}(Valve \wedge \neg\mathbf{X}(Valve) \wedge \mathbf{X}(SysOn) \Rightarrow \mathbf{X}(\neg Valve\ \mathbf{U}\ (OpnLid \vee \neg SysOn)))$
	Ограничения на повторную работу устройств: всегда при включённой системе после закрытия крышки она не открывается до тех пор, пока не откроется разливной клапан, после закрытия клапана он повторно не открывается до тех пор, пока не откроется крышка.
$P21$	$\mathbf{G}(\neg OpnLid \Rightarrow \mathbf{F}(OpnLid \vee \neg SysOn)) \wedge \mathbf{G}(\neg ClsLid \Rightarrow \mathbf{F}(ClsLid \vee \neg SysOn))$
	Бесконечная работа устройств по циклу: всегда, если работа заслонки прекратилась, то в будущем она обязательно возобновится. Исключение — когда система отключена.
$P22$	$\mathbf{G}(\neg FMech \Rightarrow \mathbf{F}(FMech \vee \neg SysOn)) \wedge \mathbf{G}(\neg Valve \Rightarrow \mathbf{F}(Valve \vee \neg SysOn)) \wedge$ $\mathbf{G}(\neg Convr \Rightarrow \mathbf{F}(Convr \vee \neg SysOn)) \wedge \mathbf{G}(\neg Heater \Rightarrow \mathbf{F}(Heater \vee \neg SysOn))$
	Бесконечная работа устройств по циклу: всегда, если работа устройства прекратилась, то в будущем она обязательно возобновится. Исключение — когда система отключена.
$P23$	$\mathbf{G}(FMech \Rightarrow \mathbf{F}(\neg FMech)) \wedge \mathbf{G}(Valve \Rightarrow \mathbf{F}(\neg Valve)) \wedge \mathbf{G}(OpnLid \Rightarrow \mathbf{F}(\neg OpnLid)) \wedge \mathbf{G}(ClsLid \Rightarrow \mathbf{F}(\neg ClsLid)) \wedge \mathbf{G}(Convr \Rightarrow \mathbf{F}(\neg Convr \vee PBConvr)) \wedge \mathbf{G}(Heater \Rightarrow (\mathbf{F}(\neg Heater) \vee \mathbf{G}(\neg UTS)))$
	Конечное время работы устройств: всегда, если указанное устройство включилось, то в будущем оно обязательно отключится. Исключения: для конвейера — удержание кнопки его работы, для нагревателя — невозможность достигнуть температуры его отключения.
$P24$	$\boxed{\mathbf{G}(Compl \Rightarrow \mathbf{F}(\neg Compl))}$ Результативность режима плавного завершения работы: всегда, если началось плавное завершение работы, то оно обязательно закончится.
$P25$	$\boxed{\mathbf{G}(MTmr.Q \Rightarrow \mathbf{F}(\neg MTmr.Q))}$ Организация работы с таймерами в программе: всегда, если таймер плавления сработал, то в будущем он будет отключен.
$P26$	$\boxed{\mathbf{G}(Convr \Rightarrow (Convr\ \mathbf{U}\ (FS2 \vee \neg SysOn)))}$ Непрерывность работы конвейера: всегда, если конвейер включился, то он работает до появления формы или отключения системы.
$P27$	$\mathbf{G}[FMech \Rightarrow (FMech\ \mathbf{U}\ (WS1 \vee \neg SysOn))]$
	Непрерывность загрузки дозатора: всегда, если включился механизм подачи, то он будет работать до заполнения дозатора или отключения системы.
$P28$	$\mathbf{G}[Valve \Rightarrow (Valve\ \mathbf{U}\ (\neg WS0 \vee \neg SysOn))]$
	Непрерывность литья: всегда, если открылся разливной клапан, то он будет открыт до опустошения дозатора или отключения системы.

7. Построение и верификация SMV-спецификации

В LTL-спецификации поведения булевых переменных не используются вспомогательные переменные из $\_V$, поэтому задача непосредственной верификации имеет вид $cTS \models (\varphi \Rightarrow \psi)$, где φ — LTL-спецификация поведения ПЛУ и её окружения, а ψ — LTL-спецификация свойств. Непосредственная верификация декларативной LTL-спецификации поведения УПО с помощью инструмента nuXmv занимает длительное время [28]. Для ускорения процесса верификации в [28] было предложено преобразование декларативной LTL-спецификации (dcl_LTL) в декларативную SMV-спецификацию (dclSMV). В настоящей работе воспользуемся этим преобразованием для LTL-спецификаций поведения ПЛУ (приложение 1) и её окружения (приложение 2).

При преобразовании LTL-спецификации поведения окружения ПЛУ в SMV-спецификацию есть некоторые особенности:

1. Преобразованию подлежит не полная (приложение 2), а сокращённая LTL-спецификация (см. параграф 5.3).
2. В (10) последние две формулы представляют собой условия *справедливости* и задаются в SMV-спецификации специальным стандартным образом.
3. Не все условия справедливости могут потребоваться для проверки свойств.

Стандартизация условий справедливости. Инструмент проверки модели nuXmv поддерживает две формы справедливости [72]:

1. *Безусловная справедливость* [16]: p выполняется бесконечно часто — $\mathbf{GF}(p)$. В SMV-спецификации это записывается **FAIRNESS**(p).
2. *Сильная справедливость* [16]: p выполняется бесконечно часто, если q также выполняется бесконечно часто — $\mathbf{GF}(q) \Rightarrow \mathbf{GF}(p)$. В SMV-спецификации это записывается **COMPASSION**(q, p).

Приведём каждое условие справедливости из LTL-спецификации поведения окружения ПЛУ (приложение 2) к одной из двух вышеуказанных форм. Для этого определим набор шаблонов, который будет иметь три булевы переменные: act_1 — действие, приводящее к реакции *rea*, act_0 — противодействие действию act_1, т. е. act_0 приводит к отсутствию реакции *rea*.

Общий шаблон (шаблон № 1) имеет вид

$$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(act_0))\,] \tag{16}$$

и утверждает, что всегда, если реакция постоянно отсутствует, то с некоторого момента времени всегда нет действия act_1 или любое действие act_1 в будущем сопровождается противодействием act_0. Данному шаблону соответствуют четыре условия справедливости из LTL-спецификации поведения окружения ПЛУ (см. таблицу 3). Для первого условия справедливости $act_1 = OpnLid$, $act_0 = ClsLid$, $rea = \neg CLS$. Действие *OpnLid* (открытие) приводит к реакции $\neg CLS$ (крышка не закрыта), действие *ClsLid* (закрытие) имеет обратный эффект.

Для (16) при постоянном отсутствии противодействия ($act_0 = False$) получим частный случай:

$$\mathbf{G}(\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1)\,). \tag{17}$$

Доказательство см. в приложении 3. Полученному шаблону № 2 соответствуют три условия справедливости (см. таблицу 3). Первое условие утверждает, что всегда, если датчик *FS1* «залип» в состоянии *True*, то с некоторого момента времени конвейер больше никогда не включается. У действия включения конвейера ($act_1 = Convr$) нет противодействия, так как привод конвейера не может вращаться в обратную сторону.

Если отсутствие действия является противодействием ($act_0 = \neg act_1$), то для (16) получим другой частный случай (шаблон № 3):

$$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(\neg act_1))\,]. \tag{18}$$

Table 3. Fairness patterns

Таблица 3. Шаблоны справедливости

№	Шаблон справедливости	Условие справедливости
1	$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(act_0))\,]$	$\mathbf{G}[\,\mathbf{G}(CLS) \Rightarrow \mathbf{FG}(\neg OpnLid) \vee \mathbf{G}(OpnLid \Rightarrow \mathbf{F}(ClsLid))\,]$
		$\mathbf{G}[\,\mathbf{G}(OLS) \Rightarrow \mathbf{FG}(\neg ClsLid) \vee \mathbf{G}(ClsLid \Rightarrow \mathbf{F}(OpnLid))\,]$
		$\mathbf{G}[\,\mathbf{G}(WS0) \Rightarrow \mathbf{FG}(\neg(Valve \wedge WTS)) \vee \mathbf{G}(Valve \wedge WTS \Rightarrow \mathbf{F}(FMech \wedge \neg CLS))\,]$
		$\mathbf{G}[\,\mathbf{G}(WS1) \Rightarrow \mathbf{FG}(\neg(Valve \wedge WTS)) \vee \mathbf{G}(Valve \wedge WTS \Rightarrow \mathbf{F}(FMech \wedge \neg CLS))\,]$
2	$\mathbf{G}(\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1)\,)$	$\mathbf{G}(\,\mathbf{G}(FS1) \Rightarrow \mathbf{FG}(\neg Convr)\,)$
		$\mathbf{G}(\,\mathbf{G}(IFS) \Rightarrow \mathbf{FG}(\neg Convr)\,)$
		$\mathbf{G}(\,\mathbf{G}(FS2) \Rightarrow \mathbf{FG}(\neg Convr)\,)$
3	$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(\neg act_1))\,]$	$\mathbf{G}[\,\mathbf{G}(UTS) \Rightarrow \mathbf{FG}(Heater) \vee \mathbf{G}(\neg Heater \Rightarrow \mathbf{F}(Heater))\,]$
		$\mathbf{G}[\,\mathbf{G}(LTS) \Rightarrow \mathbf{FG}(Heater) \vee \mathbf{G}(\neg Heater \Rightarrow \mathbf{F}(Heater))\,]$
		$\mathbf{G}[\,\mathbf{G}(WTS) \Rightarrow \mathbf{FG}(Heater) \vee \mathbf{G}(\neg Heater \Rightarrow \mathbf{F}(Heater))\,]$
		$\mathbf{G}[\,\mathbf{G}(\neg MTmr.Q) \Rightarrow \mathbf{FG}(\neg MTmr.In) \vee \mathbf{G}(MTmr.In \Rightarrow \mathbf{F}(\neg MTmr.In))\,]$

Шаблону (18) соответствуют оставшиеся условия справедливости из LTL-спецификации поведения окружения ПЛУ (см. таблицу 3). Первое условие утверждает, что всегда, если датчик высокой температуры *UTS* «залип» в состоянии *True*, то с некоторого момента нагреватель больше никогда не выключается или после выключения обязательно следует его включение. Здесь $act_1 = \neg Heater$, $act_0 = Heater$, $rea = \neg UTS$. Чтобы снизить температуру ($\neg UTS$), необходимо выключить нагреватель ($\neg Heater$) — его включение ($Heater$) является противодействием. Четвертое условие накладывает ограничение на справедливое включение таймера $MTmr.Q$. Формулы для таймеров $HTmr.Q$, $FTmr.Q$, $CTmr.Q$ аналогичны формуле для таймера $MTmr.Q$.

Общий шаблон (16) — шаблон № 1 — представляет собой сильную справедливость:

$$\mathbf{GF}(act_1) \Rightarrow \mathbf{GF}(act_0 \vee rea). \tag{19}$$

Доказательство представлено в приложении 3. В (19) подставим $act_0 = False$, получим стандартную форму для шаблона № 2:

$$\mathbf{GF}(act_1) \Rightarrow \mathbf{GF}(rea). \tag{20}$$

Стандартную форму для шаблона № 3 получим путём подстановки $act_0 = \neg act_1$ в (19) и дальнейшего преобразования (см. приложение 3). В результате получим:

$$\mathbf{GF}(act_1 \Rightarrow rea). \tag{21}$$

Соответствие шаблонов условий справедливости (16), (17), (18) и их стандартных форм (в том числе в SMV-формате) представлено в таблице 4. В итоге, шаблоны № 1 и № 2 выражают сильную справедливость (COMPASSION), а шаблон № 3 — безусловную (FAIRNESS).

Table 4. Patterns standard form **Таблица 4.** Стандартная форма шаблонов

№	Шаблон справедливости	Стандартная форма	SMV-формат
1	$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(act_0))\,]$	$\mathbf{GF}(act_1) \Rightarrow \mathbf{GF}(act_0 \vee rea)$	COMPASSION(act_1, $act_0 \mid rea$)
2	$\mathbf{G}(\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1)\,)$	$\mathbf{GF}(act_1) \Rightarrow \mathbf{GF}(rea)$	COMPASSION(act_1, rea)
3	$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(\neg act_1))\,]$	$\mathbf{GF}(act_1 \Rightarrow rea)$	FAIRNESS(act_1 -> rea)

Результат преобразования LTL-спецификаций поведения ПЛУ (приложение 1) и её окружения (приложение 2) представляет собой SMV-спецификацию поведения КФС, которая находится в приложении 4. SMV-спецификация описывает поведение всех переменных ПЛУ и её окружения. Так как поведение всех таймеров однотипно, оно было описано в виде модуля *Timer* (см. листинг 1). Данный листинг является фрагментом SMV-спецификации поведения КФС.

Листинг 1 (SMV-спецификация модуля Timer)**:**

```
MODULE Timer
VAR
    In : boolean; -- Input  (Logical Control Program Unit Output)
    Q  : boolean; -- Output (Logical Control Program Unit Input)
ASSIGN
    INIT ( !Q & !In )
    TRANS( !Q &  next(Q) ->  In )
    TRANS(  Q & !next(Q) -> !In )
    TRANS(  Q &  next(Q) ->  In )
    FAIRNESS (In -> Q) -- Timer Fair Turn On.
```

Модуль содержит объявление и инициализацию входной (In) и выходной (Q) переменных таймера, поведение переменной Q, условие честного срабатывания. Все таймеры УПО являются экземплярами данного модуля.

При верификации условия справедливости добавляются в SMV-спецификацию поведения КФС только в случае необходимости. Мы используем подход уточнения абстракции (модели поведения КФС) на основе контрпримера (CounterExample-Guided Abstraction Refinement, CEGAR) [15]. Если обнаружен ложный контрпример, демонстрирующий нарушение свойства из-за несоблюдения некоторого условия справедливости, то данное условие добавляется в SMV-спецификацию. Для проверки всей совокупности свойств помимо условия справедливости для модуля таймера из листинга 1 потребовалось добавить ещё пять условий справедливости (см. листинг 2).

Листинг 2 (SMV-спецификация добавленных условий справедливости)**:**

```
COMPASSION (Convr, !FS2)              -- Fair sticking of FS2 sensor in True  state.
COMPASSION (OpnLid, ClsLid | OLS)                -- ... OLS sensor in False state.
COMPASSION (ClsLid, OpnLid | CLS)                -- ... CLS sensor in False state.
COMPASSION (Valve & WTS, (FMech & !CLS) | !WS0)  -- ... WS0 sensor in True  state.
FAIRNESS   (!Heater -> !WTS)                     -- ... WTS sensor in True  state.
```

Результаты верификации. Потенциально возможное число состояний модели поведения замкнутой системы (установки для литья пластмасс) составляет $2.74878 \cdot 10^{11}$ (2^{38}), так как SMV-спецификация установки содержит 38 булевых переменных. Верификация проводилась на персональном компьютере с процессором Intel Core i5-3570 3.4 ГГц и 8 ГБ оперативной памяти. Результаты представлены в таблице 5.

Table 5. Verification results **Таблица 5.** Результаты верификации

Свойство	Тип свойства: безопасность (Saf), живость (Liv)	Недетерминизм окружения		
		Абсолютный	Ограниченный	
			Без справедлив.	Со справедлив.
Столбец 1	Столбец 2	Столбец 3	Столбец 4	Столбец 5
$P1$	Saf	**1**	**1**	**1**
$P2$	Saf	**1**	**1**	**1**
$P3$	Saf	**1**	**1**	**1**
$P4$	Saf	**1**	**1**	**1**
$P5$	Saf	**1**	**1**	**1**
$P6$	Saf	**1**	**1**	**1**
$P7$	Saf	**1**	**1**	**1**
$P8$	Saf	**1**	**1**	**1**
$P9$	Saf	**1**	**1**	**1**
$P10$	Saf	0	**1**	**1**
$P11$	Saf	0	**1**	**1**
$P12$	Saf	0	**1**	**1**
$P13$	Saf	0	**1**	**1**
$P14$	Saf	0	**1**	**1**
$P15$	Saf	0	**1**	**1**
$P16$	Saf	0	**1**	**1**
$P17$	Saf	0	**1**	**1**
$P18$	Liv	0	**1**	**1**
$P19$	Liv	0	**1**	**1**
$P20$	Liv	0	0	**1**
$P21$	Liv	0	0	**1**
$P22$	Liv	0	0	**1**
$P23$	Liv	0	0	**1**
$P24$	Liv	0	0	**1**
$P25$	Liv	0	0	**1**
$P26$	Liv	0	0	**1**
$P27$	Liv	0	0	**1**
$P28$	Liv	0	0	**1**
Количество состояний:		1381496 ($2^{20.3978}$)	16150 ($2^{13.9792}$)	

Столбец 1 содержит идентификаторы проверяемых свойств $P1, \ldots, P28$, столбец 2 — тип свойства: $P1, \ldots, P17$ являются свойствами *безопасности*, $P18, \ldots, P28$ — свойствами *живости*. Столбцы 3, 4 и 5 отражают результаты верификации свойств при различных условиях. Наличие «1» в ячейке таблицы означает, что соответствующее свойство выполняется, а «0» — нарушается. Столбец 3 содержит результаты верификации при абсолютном недетерминизме окружения ПЛУ, столбец 4 — при ограниченном недетерминизме без условий справедливости, столбец 5 — при ограниченном недетерминизме с условиями справедливости.

Из таблицы 5 видно, что свойства $P1, \ldots, P9$ могут быть проверены при любых условиях (столбцы 3, 4 и 5 содержат «1»). При абсолютно недетерминированном поведении окружения ПЛУ система переходов имеет $1.38146 \cdot 10^6$ ($2^{20.3978}$) достижимых состояний (нижняя часть столбца 3), а при огра-

ниченно недетерминированном — 16150 ($2^{13.9792}$) состояний (нижняя часть столбцов 4 и 5). Таким образом, ограничение недетерминизма привело к сокращению пространства состояний примерно в 85 раз. Это ускоряет верификацию: время проверки свойств $P1, \ldots, P9$ при абсолютном недетерминизме составляет 6,3 секунды, а ограниченном (без условий справедливости) — 1,6 секунды. Таким образом, время верификации сократилось примерно в четыре раза. Также данное ограничение недетерминизма позволяет проверять дополнительные свойства $P10, \ldots, P19$ (см. столбец 4), которые требуют наличия реалистичных ограничений в поведении окружения.

Для проверки свойств $P20, \ldots, P28$ в модель поведения окружения необходимо добавить условия справедливости (см. столбец 5). Однако добавление условий справедливости увеличивает время верификации: время проверки свойств $P1, \ldots, P9$ при ограниченном недетерминизме с условиями справедливости составляет 74 секунды, что многократно превышает время верификации, равное 6,3 секунды и 1,6 секунды в двух предыдущих случаях. Проверка с помощью инструмента nuXmv всех свойств $P1, \ldots, P28$ с учётом ограничений и условий справедливости заняла около 4 минут.

В итоге описание поведения сигналов обратной связи (без условий справедливости) позволило проверить свойства безопасности $P10, \ldots, P17$, а также свойства живости $P18, P19$. Использование условий справедливости добавило возможность проверки свойств живости $P20, \ldots, P28$.

8. Построение ST-программы для ПЛК

Согласно [27] на основе декларативной (приложение 1) была построена императивная LTL-спецификация (приложение 5), которая далее переводится в программу на языке ST (приложение 6). ST-программа (PLC_PRG) имеет разделы для объявления входных (VAR_INPUT), выходных (VAR_OUTPUT) и внутренних (VAR) переменных. Таймеры являются внутренними объектами УПО и внешними по отношению к ПЛУ.

Заключение

В работе предложен специальный вид LTL-спецификации поведения программ логического управления для систем с обратной связью. Детерминированное поведение внутренних и выходных булевых переменных программы описывается с помощью частного случая декларативной LTL-спецификации, а недетерминированное поведение входных булевых переменных — с помощью LTL-спецификации ограниченно недетерминированного поведения. Последняя LTL-спецификация позволяет описывать поведение сигналов обратной связи и мнимых датчиков (дополнительных булевых переменных для описания состояния компонентов из окружения программы). В ней задаются условия для переходов между состояниями (*False* и *True*) каждой булевой переменной, а также условия, необходимые при «залипании» в этих состояниях, — условия справедливости. Предложенный способ LTL-спецификации позволяет строить достаточно простые модели поведения при верификации замкнутых систем, что уменьшает трудозатраты на разработку программ. Результаты работы продемонстрированы на примере промышленной установки для литья пластмасс.

Для ускорения процесса верификации с помощью инструмента проверки модели nuXmv LTL-спецификация переводится в SMV-спецификацию — входной язык инструмента nuXmv. Для представленного в работе примера установки определён набор шаблонов условий справедливости и связей между ними. Доказано, что каждый из шаблонов является условием сильной или безусловной справедливости, которые могут быть выражены с помощью стандартных средств инструмента nuXmv. Выполнена проверка SMV-спецификации поведения установки для литья пластмасс на предмет соответствия заданному набору LTL-свойств. Продемонстрировано, что только часть свойств может быть проверена без моделирования поведения окружения программы. Для проверки другой части свойств требуется такая модель. Отсутствие условий справедливости в данной модели также не позволяет проверить ряд свойств. Таким образом, LTL-спецификация ограниченно недетерминированного поведения окружения программы позволяет задавать всю необходимую

информацию для проверки всего набора свойств. Кроме того, задание ограничений в поведении сигналов обратной связи сокращает пространство состояний модели при верификации.

Декларативной LTL-спецификации поведения программы соответствует эквивалентная императивная LTL-спецификация, по которой построена ST-программа для ПЛК. Поведение данной программы гарантированно соответствует поведению исходной декларативной LTL-спецификации.

Будущие исследования могут быть направлены на автоматизацию синтеза ST-программ и SMV-спецификаций по исходной LTL-спецификации КФС.

References

[1] S. Oks and et al., "Cyber-Physical Systems in the Context of Industry 4.0: A Review, Categorization and Outlook", *Information Systems Frontiers*, pp. 1–42, 2022. DOI: 10.1007/s10796-022-10252-x.

[2] K. Zhang, Y. Shi, S. Karnouskos, T. Sauter, H. Fang, and A. W. Colombo, "Advancements in Industrial Cyber-Physical Systems: An Overview and Perspectives", *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 716–729, 2023. DOI: 10.1109/TII.2022.3199481.

[3] S. J. Oks, *Industrial Cyber-Physical Systems: Advancing Industry 4.0 from Vision to Application* (MAU), 1st ed. Springer, 2024, ISBN: 978-3-658-44416-7. DOI: 10.1007/978-3-658-44417-4.

[4] C. Dey and S. K. Sen, *Industrial Automation Technologies*, 1st ed. CRC Press, 2020, ISBN: 9780429299346. DOI: 10.1201/9780429299346.

[5] K. Thramboulidis, "A Cyber–Physical System-Based Approach for Industrial Automation Systems", *Computers in Industry*, vol. 72, pp. 92–102, 2015. DOI: 10.1016/j.compind.2015.04.006.

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

[7] R. Langmann and L. F. Rojas-Peña, "A PLC as an Industry 4.0 Component", in *Remote Engineering and Virtual Instrumentation*, 2016, pp. 10–15. DOI: 10.1109/REV.2016.7444433.

[8] 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.

[9] A. Maurya and D. Kumar, "Reliability of safety-critical systems: A state-of-the-art review", *Quality and Reliability Engineering International*, vol. 36, Aug. 2020. DOI: 10.1002/qre.2715.

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

[11] V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review", *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013. DOI: 10.1109/TII.2013.2258165.

[12] S. Mitra, *Verifying Cyber-Physical Systems: A Path to Safe Autonomy* (Cyber Physical Systems Series). MIT Press, 2021, ISBN: 978-0262044806.

[13] 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.

[14] R. Sinha, S. Patil, L. Gomes, and V. Vyatkin, "A Survey of Static Formal Methods for Building Dependable Industrial Automation Systems", *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 3772–3783, 2019. DOI: 10.1109/TII.2019.2908665.

[15] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*, 1st. Springer, 2018, vol. 10, ISBN: 3319105744. DOI: 10.1007/978-3-319-10575-8.

[16] Y. G. Karpov, *MODEL CHECKING. Verification of Parallel and Distributed Program Systems.* BHV-Peterburg, 2010, p. 560, in Russian, ISBN: 978-5-9775-0404-1.

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

[18] 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.

[19] K. Schneider, J. Shabolt, and J. G. Taylor, *Verification of reactive systems: formal methods and algorithms* (EATCS), 1st ed. Springer, 2004, ISBN: 978-3-540-00296-3. DOI: 10.1007/978-3-662-10778-2.

[20] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*, 1st ed. Springer, 2012, ISBN: 978-0-387-94459-3. DOI: 10.1007/978-1-4612-4222-2.

[21] J. Galvão, C. Oliveira, and et al., "Formal Verification: Focused on the Verification Using a Plant Model", in *Innovation, Engineering and Entrepreneurship*, Springer, 2019, pp. 124–131. DOI: 10.1007/978-3-319-91334-6_18.

[22] G. Frey and L. Litz, "Formal Methods in PLC Programming", in *IEEE International Conference On Systems, Man and Cybernetics*, vol. 4, 2000, pp. 2431–2436. DOI: 10.1109/ICSMC.2000.884356.

[23] J. M. Machado, B. Denis, and et al., "Logic Controllers Dependability Verification Using a Plant Model", *IFAC Proceedings Volumes*, vol. 39, no. 17, pp. 37–42, 2006. DOI: 10.3182/20060926-3-PL-4904.00007.

[24] A. A. Shalyto, "Logic Control and "Reactive" Systems: Algorithmization and Programming", *Automation and Remote Control*, vol. 62, no. 1, pp. 1–29, 2001. DOI: 10.1023/A:1002837232103.

[25] 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. DOI: 10.18255/1818-1015-2014-4-75-90.

[26] 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. DOI: 10.3103/S0146411614070256.

[27] 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.

[28] M. Neyzov and E. Kuzmin, "Verification of Declarative LTL-specification of Control Programs Behavior", *Modeling and Analysis of Information Systems*, vol. 31, no. 2, pp. 120–141, 2024, in Russian. DOI: 10.18255/1818-1015-2024-2-120-141.

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

[30] M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, and M. Ouenzar, "Comparison of Model Checking Tools for Information Systems", in *Formal Methods and Software Engineering*, ser. LNCS, vol. 6447, Springer, 2010, pp. 581–596, ISBN: 978-3-642-16901-4. DOI: 10.1007/978-3-642-16901-4_38.

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

[32] M. Xavier, S. Patil, V. Dubinin, and V. Vyatkin, "Formal Modelling, Analysis, and Synthesis of Modular Industrial Systems Inspired by Net Condition/Event Systems", in *Applications and Theory of Petri Nets and Concurrency*, 2023, pp. 16–33. DOI: 10.1007/978-3-031-33620-1_2.

[33] V. Vyatkin, H.-M. Hanisch, C. Pang, and C.-H. Yang, "Closed-Loop Modeling in Future Automation System Engineering and Validation", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 1, pp. 17–28, 2009. DOI: 10.1109/TSMCC.2008.2005785.

[34] A. Lobov, J. Lastra, and R. Tuokko, "Application of UML in Plant Modeling for Model-Based Verification: UML Translation to TNCES", in *IEEE Industrial Informatics*, 2005, pp. 495–501. DOI: 10.1109/INDIN.2005.1560426.

[35] A. Lobov, J. Lastra, and R. Tuokko, "On Controller and Plant Modeling for Model-Based Formal Verification", in *IEEE Emerging Technologies and Factory Automation*, vol. 1, 2005, pp. 121–128. DOI: 10.1109/ETFA.2005.1612510.

[36] S. Preuße, *Technologies for Engineering Manufacturing Systems Control in Closed Loop*, 10th ed. Logos Verlag Berlin GmbH, 2013, ISBN: 978-3-8325-3600-8.

[37] S. Patil, S. Bhadra, and V. Vyatkin, "Closed-Loop Formal Verification Framework with Non-Determinism, Configurable by Meta-Modelling", in *IEEE Industrial Electronics Society*, 2011, pp. 3770–3775. DOI: 10.1109/IECON.2011.6119923.

[38] S. Patil, V. Vyatkin, and M. Sorouri, "Formal Verification of Intelligent Mechatronic Systems with Decentralized Control Logic", in *IEEE Emerging Technologies & Factory Automation*, 2012, pp. 1–7. DOI: 10.1109/ETFA.2012.6489678.

[39] S. Patil, V. Vyatkin, and C. Pang, "Counterexample-Guided Simulation Framework for Formal Verification of Flexible Automation Systems", in *IEEE Industrial Informatics*, 2015, pp. 1192–1197. DOI: 10.1109/INDIN.2015.7281905.

[40] C. Gerber, S. Preuße, and H.-M. Hanisch, "A Complete Framework for Controller Verification in Manufacturing", in *IEEE Emerging Technologies & Factory Automation*, 2010, pp. 1–9. DOI: 10.1109/ETFA.2010.5641220.

[41] S. Preuße, H.-C. Lapp, and H.-M. Hanisch, "Closed-Loop System Modeling, Validation, and Verification", in *IEEE Emerging Technologies & Factory Automation*, 2012, pp. 1–8. DOI: 10.1109/ETFA.2012.6489679.

[42] J. Machado, B. Denis, and J.-J. Lesage, "A Generic Approach to Build Plant Models for DES Verification Purposes", in *International Workshop on Discrete Event Systems*, 2006, pp. 407–412. DOI: 10.1109/WODES.2006.382508.

[43] J. Machado, E. Seabra, and et al., "Safe Controllers Design for Industrial Automation Systems", *Computers & Industrial Engineering*, vol. 60, no. 4, pp. 635–653, 2011. DOI: 10.1016/j.cie.2010.12.020.

[44] M. Perin and J.-M. Faure, "Building Meaningful Timed Models of Closed-Loop DES for Verification Purposes", *Control Engineering Practice*, vol. 21, no. 11, pp. 1620–1639, 2013. DOI: 10.1016/j.conengprac.2012.05.002.

[45] H.-M. Hanisch, "Closed-Loop Modeling and Related Problems of Embedded Control Systems in Engineering", in *Abstract State Machines 2004. Advances in Theory and Practice*, Springer, 2004, pp. 6–19. DOI: 10.1007/978-3-540-24773-9_2.

[46] C. Pang and V. Vyatkin, "Systematic Closed-Loop Modelling in IEC 61499 Function Blocks: A Case Study", *IFAC Proceedings Volumes*, vol. 42, pp. 199–204, 2009. DOI: 10.3182/20090603-3-RU-2001.0264.

[47] D. Drozdov, S. Patil, V. Dubinin, and V. Vyatkin, "Formal Verification of Cyber-Physical Automation Systems Modelled with Timed Block Diagrams", in *IEEE Industrial Electronics*, 2016, pp. 316–321. DOI: 10.1109/ISIE.2016.7744910.

[48] M. Xavier, S. Patil, and V. Vyatkin, "Cyber-Physical Automation Systems Modelling with IEC 61499 for their Formal Verification", in *IEEE Industrial Informatics*, 2021, pp. 1–6. DOI: 10.1109/INDIN45523.2021.9557416.

[49] G. Lilli *et al.*, "Formal Verification of the Control Software of a Radioactive Material Remote Handling System, Based on IEC 61499", *IEEE Open Journal of the Industrial Electronics Society*, vol. 4, pp. 417–431, 2023. DOI: 10.1109/OJIES.2023.3321084.

[50] N. Garanina, S. Staroletov, V. Zyubin, and I. Anureev, "Model Checking Programs in Process-Oriented IEC 61131-3 Structured Text", *Modeling and Analysis of Information Systems*, vol. 31, no. 1, pp. 32–53, 2024, in Russian. DOI: 10.18255/1818-1015-2024-1-32-53.

[51] N. Halbwachs, F. Lagnier, and C. Ratel, "Programming and Verifying Real-Time Systems by Means of the Synchronous Data-Flow Language LUSTRE", *IEEE Transactions on Software Engineering*, vol. 18, no. 9, pp. 785–793, 1992. DOI: 10.1109/32.159839.

[52] *IEC 61499-1:2012 Function blocks – Part 1: Architecture*. [Online]. Available: https://webstore.iec.ch/publication/5506.

[53] V. E. Zyubin, A. S. Rozov, I. S. Anureev, N. O. Garanina, and V. Vyatkin, "poST: A Process-Oriented Extension of the IEC 61131-3 Structured Text Language", *IEEE Access*, vol. 10, pp. 35 238–35 250, 2022. DOI: 10.1109/ACCESS.2022.3157601.

[54] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The Synchronous Data Flow Programming Language LUSTRE", *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, 1991. DOI: 10.1109/5.97300.

[55] A. Champion, A. Gurfinkel, and et al., "CoCoSpec: A Mode-Aware Contract Language for Reactive Systems", in *Software Engineering and Formal Methods*, ser. LNTCS, vol. 9763, Springer, 2016, pp. 347–366. DOI: 10.1007/978-3-319-41591-8_24.

[56] A. Benveniste, P. Caspi, and et al., "The Synchronous Languages 12 Years Later", *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003. DOI: 10.1109/JPROC.2002.805826.

[57] A. Bouajjani, J. Fernandez, and N. Halbwachs, "On the Verification of Safety Properties", *Rapport Technique SPECTRE L12*, 1990.

[58] P. Raymond, "Synchronous Program Verification with Lustre/Lesar", in *Modeling and Verification of Real-Time Systems*, John Wiley & Sons, 2008, ch. 6, pp. 171–206, ISBN: 9780470611012. DOI: 10.1002/9780470611012.ch6.

[59] A. Champion, A. Mebsout, C. Sticksel, and C. Tinelli, "The Kind 2 Model Checker", in *Computer Aided Verification*, ser. LNTCS, vol. 9780, Springer, 2016, pp. 510–517. DOI: 10.1007/978-3-319-41540-6_29.

[60] N. Halbwachs, *Synchronous Programming of Reactive Systems*. Springer, 1993, ISBN: 978-0792393115.

[61] M. Sirjani, E. A. Lee, and E. Khamespanah, "Verification of Cyberphysical Systems", *Mathematics*, vol. 8, no. 7, 2020. DOI: 10.3390/math8071068.

[62] S. Lin *et al.*, "Towards Building Verifiable CPS using Lingua Franca", *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 1–24, 2023. DOI: 10.1145/3609134.

[63] P. Raymond, Y. Roux, and E. Jahier, "Lutin: A Language for Specifying and Executing Reactive Scenarios", *EURASIP Journal on Embedded Systems*, vol. 2008, pp. 1–11, 2008. DOI: 10.1155/2008/753821.

[64] B. Finkbeiner, "Synthesis of Reactive Systems", *Dependable Software Systems Engineering*, vol. 45, pp. 72–98, 2016. DOI: 10.3233/978-1-61499-627-9-72.

[65] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive (1) Designs", in *Verification, Model Checking, and Abstract Interpretation*, vol. 3855, Springer, 2006, pp. 364–380. DOI: 10.1007/11609773_24.

[66] M. Roth, L. Litz, and J.-J. Lesage, "Identification of Discrete Event Systems: Implementation Issues and Model Completeness", in *Informatics in Control, Automation and Robotics*, vol. 3, 2010, pp. 73–80.

[67] I. Buzhinsky and V. Vyatkin, "Automatic Inference of Finite-State Plant Models From Traces and Temporal Properties", *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1521–1530, 2017. DOI: 10.1109/TII.2017.2670146.

[68] P. Ovsiannikova, D. Chivilikhin, V. Ulyantsev, and A. Shalyto, "Closed-Loop Verification of a Compensating Group Drive Model Using Synthesized Formal Plant Model", in *IEEE Emerging Technologies and Factory Automation*, 2017, pp. 1–4. DOI: 10.1109/ETFA.2017.8247714.

[69] I. Buzhinsky, A. Pakonen, and V. Vyatkin, "Scalable Methods of Discrete Plant Model Generation for Closed-Loop Model Checking", in *IEEE Industrial Electronics Society*, 2017, pp. 5483–5488. DOI: 10.1109/IECON.2017.8216949.

[70] M. Xavier, J. Håkansson, S. Patil, and V. Vyatkin, "Plant Model Generator from Digital Twin for Purpose of Formal Verification", in *IEEE Emerging Technologies and Factory Automation*, 2021, pp. 1–4. DOI: 10.1109/ETFA45728.2021.9613704.

[71] M. Xavier, V. Dubinin, S. Patil, and V. Vyatkin, "Plant Model Generation From Event Log Using ProM for Formal Verification of CPS", *arXiv preprint arXiv:2211.03681*, 2022. DOI: 10.48550/arXiv.2211.03681.

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

Приложение 1. Декларативная LTL-спецификация поведения ПЛУ

```
!FErr &
G( !FErr &  X(FErr) ->  (X(FTmr.Q) & !X(WS1)) ) &
G( !FErr & !X(FErr) -> !(X(FTmr.Q) & !X(WS1)) ) &
G(  FErr & !X(FErr) ->  X(PBStop) ) &
G(  FErr &  X(FErr) -> !X(PBStop) ) &
!CErr &
G( !CErr &  X(CErr) ->  (X(CTmr.Q) & !X(FS2)) ) &
G( !CErr & !X(CErr) -> !(X(CTmr.Q) & !X(FS2)) ) &
G(  CErr & !X(CErr) ->  X(PBStop) ) &
G(  CErr &  X(CErr) -> !X(PBStop) ) &
!HErr &
G( !HErr &  X(HErr) ->  (!X(WTS) & (X(HTmr.Q) | Heater & WTS)) ) &
G( !HErr & !X(HErr) -> !(!X(WTS) & (X(HTmr.Q) | Heater & WTS)) ) &
G(  HErr & !X(HErr) ->  X(PBStop) ) &
G(  HErr &  X(HErr) -> !X(PBStop) ) &
!Fin &
G( X(Fin) <-> (Compl & X(OLS) & !X(WS0) | X(FErr) | X(HErr) | X(CErr) |
               X(PBStop) | X(PBConvr)) ) &
!SysOn &
G( !SysOn &  X(SysOn) ->  (X(PBStart) & !X(Fin)) ) &
G( !SysOn & !X(SysOn) -> !(X(PBStart) & !X(Fin)) ) &
G(  SysOn & !X(SysOn) ->  X(Fin) ) &
G(  SysOn &  X(SysOn) -> !X(Fin) ) &
!Compl &
G( !Compl &  X(Compl) ->  (X(SysOn) & X(PBCompl)) ) &
G( !Compl & !X(Compl) -> !(X(SysOn) & X(PBCompl)) ) &
G(  Compl & !X(Compl) -> !X(SysOn) ) &
G(  Compl &  X(Compl) ->  X(SysOn) ) &
!Mltng &
G( X(Mltng) <-> X(SysOn) & X(CLS) & X(Disch) & X(WS0) ) &
!Mlted &
G( !Mlted &  X(Mlted) ->  (X(MTmr.Q) & X(WTS) & X(WS0) & X(CLS)) ) &
G( !Mlted & !X(Mlted) -> !(X(MTmr.Q) & X(WTS) & X(WS0) & X(CLS)) ) &
G(  Mlted & !X(Mlted) ->  (!X(WTS) | !X(CLS) | !X(WS0)) ) &
G(  Mlted &  X(Mlted) -> !(!X(WTS) | !X(CLS) | !X(WS0)) ) &
!Disch &
G( !Disch &  X(Disch) ->  (X(OLS) & X(WS1)) ) &
G( !Disch & !X(Disch) -> !(X(OLS) & X(WS1)) ) &
G(  Disch & !X(Disch) ->  (!X(FS2) & !X(WS1)) ) &
G(  Disch &  X(Disch) -> !(!X(FS2) & !X(WS1)) ) &
-- Actuators:
!Heater &
G( !Heater &  X(Heater) ->  (X(SysOn) & !X(LTS)) ) &
G( !Heater & !X(Heater) -> !(X(SysOn) & !X(LTS)) ) &
G(  Heater & !X(Heater) ->  (!X(SysOn) | X(UTS)) ) &
G(  Heater &  X(Heater) -> !(!X(SysOn) | X(UTS)) ) &
!ClsLid &
G( X(ClsLid) <-> X(SysOn) & !X(CLS) & X(Disch) ) &
!OpnLid &
G( X(OpnLid) <-> X(SysOn) & !X(OLS) & !X(Disch) ) &
!Convr &
G( X(Convr) <-> (X(SysOn) & (X(FS2) & X(Disch) & !X(WS0) | !X(FS2)) | X(PBConvr)) ) &
```

```
!LwSpd &
G( !LwSpd &  X(LwSpd) ->  X(FS1) ) &
G( !LwSpd & !X(LwSpd) -> !X(FS1) ) &
G(  LwSpd & !X(LwSpd) ->  X(FS2) ) &
G(  LwSpd &  X(LwSpd) -> !X(FS2) ) &
!FMech &
G( X(FMech) <-> X(SysOn) & !X(Disch) & X(OLS) & !X(WS1) ) &
!Valve &
G( X(Valve) <-> X(SysOn) & X(Disch) & X(Mlted) & X(FS2) ) &
-- Impact_On_Timers:
G( X(FTmr.In) <-> X(FMech)  & !X(WS1) )                    & !FTmr.In &
G( X(HTmr.In) <-> X(Heater) & !X(WTS) )                    & !HTmr.In &
G( X(CTmr.In) <-> X(Convr)  & !X(FS2) & X(SysOn) )         & !CTmr.In &
G( X(MTmr.In) <-> X(Mltng)  &  X(WTS) & X(WS0) & X(CLS) ) & !MTmr.In
```

Приложение 2. LTL-спецификация поведения окружения ПЛУ

```
-- Buttons:
!PBStart & !PBStop & !PBCompl & !PBConvr &
-- Sensors:
!FS1 &
G( !FS1 &  X(FS1) ->  !FS2 & !X(FS2) & !IFS & !X(IFS) & Convr ) &
G( !FS1 & !X(FS1) ->  true ) &
G(  FS1 & !X(FS1) ->  !FS2 & !X(FS2) & !IFS &  X(IFS) & Convr ) &
G(  FS1 &  X(FS1) ->  !FS2 & !X(FS2) & !IFS & !X(IFS)         ) &
G( G(!FS1) ->  true         ) &
G( G( FS1) ->  F(G(!Convr)) ) &
!IFS &
G( !IFS &  X(IFS) ->  !FS2 & !X(FS2) &  FS1 & !X(FS1) & Convr ) &
G( !IFS & !X(IFS) ->  true ) &
G(  IFS & !X(IFS) ->  !FS2 &  X(FS2) & !FS1 & !X(FS1) & Convr ) &
G(  IFS &  X(IFS) ->  !FS2 & !X(FS2) & !FS1 & !X(FS1)         ) &
G( G(!IFS) ->  true         ) &
G( G( IFS) ->  F(G(!Convr)) ) &
!FS2 &
G( !FS2 &  X(FS2) ->   IFS & !X(IFS) & !FS1 & !X(FS1) & Convr ) &
G( !FS2 & !X(FS2) ->  true ) &
G(  FS2 & !X(FS2) ->  !IFS & !X(IFS) & !FS1 & !X(FS1) & Convr ) &
G(  FS2 &  X(FS2) ->  !IFS & !X(IFS) & !FS1 & !X(FS1)         ) &
G( G(!FS2) ->  true         ) &
G( G( FS2) ->  F(G(!Convr)) ) &
!CLS &
G( !CLS &  X(CLS) ->  !OLS & !X(OLS) & ClsLid ) &
G( !CLS & !X(CLS) ->  true ) &
G(  CLS & !X(CLS) ->  !OLS & !X(OLS) & OpnLid ) &
G(  CLS &  X(CLS) ->  !OLS & !X(OLS) ) &
G( G(!CLS) ->  F(G(!ClsLid)) | G(ClsLid -> F(OpnLid)) ) &
G( G( CLS) ->  F(G(!OpnLid)) | G(OpnLid -> F(ClsLid)) ) &
!OLS &
G( !OLS &  X(OLS) ->  !CLS & !X(CLS) & OpnLid ) &
G( !OLS & !X(OLS) ->  true ) &
G(  OLS & !X(OLS) ->  !CLS & !X(CLS) & ClsLid ) &
G(  OLS &  X(OLS) ->  !CLS & !X(CLS) ) &
G( G(!OLS) ->  F(G(!OpnLid)) | G(OpnLid -> F(ClsLid)) ) &
```

```
G( G( OLS) ->  F(G(!ClsLid)) | G(ClsLid -> F(OpnLid)) ) &
!WS0 &
G( !WS0 &  X(WS0) ->  !WS1 & !X(WS1) & FMech & !CLS ) &
G( !WS0 & !X(WS0) ->  !WS1 & !X(WS1) ) &
G(  WS0 & !X(WS0) ->  !WS1 & !X(WS1) & Valve &  WTS ) &
G(  WS0 &  X(WS0) ->  true ) &
G( G(!WS0) ->  true ) &
G( G( WS0) ->  F(G(!Valve | !WTS)) | G(Valve & WTS -> F(FMech & !CLS)) ) &
!WS1 &
G( !WS1 &  X(WS1) ->  WS0 & X(WS0) & FMech & !CLS ) &
G( !WS1 & !X(WS1) ->  true ) &
G(  WS1 & !X(WS1) ->  WS0 & X(WS0) & Valve &  WTS ) &
G(  WS1 &  X(WS1) ->  WS0 & X(WS0) ) &
G( G(!WS1) ->  true ) &
G( G( WS1) ->  F(G(!Valve | !WTS)) | G(Valve & WTS -> F(FMech & !CLS)) ) &
!UTS &
G( !UTS &  X(UTS) ->  WTS & X(WTS) & LTS & X(LTS) & Heater ) &
G( !UTS & !X(UTS) ->  true ) &
G(  UTS & !X(UTS) ->  WTS & X(WTS) & LTS & X(LTS) ) &
G(  UTS &  X(UTS) ->  WTS & X(WTS) & LTS & X(LTS) ) &
G( G(!UTS) ->  true ) &
G( G( UTS) ->  F(G(Heater)) | G(!Heater -> F(Heater)) ) &
!LTS &
G( !LTS &  X(LTS) ->  WTS &  X(WTS) & !UTS & !X(UTS) & Heater ) &
G( !LTS & !X(LTS) -> !UTS & !X(UTS) ) &
G(  LTS & !X(LTS) ->  WTS &  X(WTS) & !UTS & !X(UTS) ) &
G(  LTS &  X(LTS) ->  WTS &  X(WTS) ) &
G( G(!LTS) ->  true ) &
G( G( LTS) ->  F(G(Heater)) | G(!Heater -> F(Heater)) ) &
!WTS &
G( !WTS &  X(WTS) ->  !LTS & !X(LTS) & !UTS & !X(UTS) & Heater ) &
G( !WTS & !X(WTS) ->  !LTS & !X(LTS) & !UTS & !X(UTS) ) &
G(  WTS & !X(WTS) ->  !LTS & !X(LTS) & !UTS & !X(UTS) ) &
G(  WTS &  X(WTS) ->  true ) &
G( G(!WTS) ->  true ) &
G( G( WTS) ->  F(G(Heater)) | G(!Heater -> F(Heater)) ) &
-- Timers:
!MTmr.Q &
G( !MTmr.Q &  X(MTmr.Q) ->  MTmr.In ) &
G( !MTmr.Q & !X(MTmr.Q) ->  true    ) &
G(  MTmr.Q & !X(MTmr.Q) -> !MTmr.In ) &
G(  MTmr.Q &  X(MTmr.Q) ->  MTmr.In ) &
G( G(!MTmr.Q) ->  F(G(!MTmr.In)) | G(MTmr.In -> F(!MTmr.In)) ) &
G( G( MTmr.Q) ->  G(MTmr.In) ) &
!HTmr.Q &
G( !HTmr.Q &  X(HTmr.Q) ->  HTmr.In ) &
G( !HTmr.Q & !X(HTmr.Q) ->  true    ) &
G(  HTmr.Q & !X(HTmr.Q) -> !HTmr.In ) &
G(  HTmr.Q &  X(HTmr.Q) ->  HTmr.In ) &
G( G(!HTmr.Q) ->  F(G(!HTmr.In)) | G(HTmr.In -> F(!HTmr.In)) ) &
G( G( HTmr.Q) ->  G(HTmr.In) ) &
!FTmr.Q &
G( !FTmr.Q &  X(FTmr.Q) ->  FTmr.In ) &
G( !FTmr.Q & !X(FTmr.Q) ->  true    ) &
```

```
G(  FTmr.Q & !X(FTmr.Q) -> !FTmr.In ) &
G(  FTmr.Q &  X(FTmr.Q) ->  FTmr.In ) &
G( G(!FTmr.Q) ->  F(G(!FTmr.In)) | G(FTmr.In -> F(!FTmr.In)) ) &
G( G( FTmr.Q) ->  G(FTmr.In) ) &
!CTmr.Q &
G( !CTmr.Q &  X(CTmr.Q) ->  CTmr.In ) &
G( !CTmr.Q & !X(CTmr.Q) ->  true    ) &
G(  CTmr.Q & !X(CTmr.Q) -> !CTmr.In ) &
G(  CTmr.Q &  X(CTmr.Q) ->  CTmr.In ) &
G( G(!CTmr.Q) ->  F(G(!CTmr.In)) | G(CTmr.In -> F(!CTmr.In)) ) &
G( G( CTmr.Q) ->  G(CTmr.In) )
```

Приложение 3. Доказательства

Утверждение 1 (Об эквивалентности). $\mathbf{GF}(a) \wedge \mathbf{G}(a \Rightarrow \mathbf{F}(b)) = \mathbf{GF}(a) \wedge \mathbf{GF}(b)$.

Доказательство. Представим доказательство в виде двух шагов.

Шаг 1. $\mathbf{GF}(a) \wedge \mathbf{GF}(b) \vdash \mathbf{GF}(a) \wedge \mathbf{G}(a \Rightarrow \mathbf{F}(b))$ очевидно, так как $\mathbf{GF}(b) \vdash \mathbf{G}(\neg a \vee \mathbf{F}(b))$.

Шаг 2. Докажем $\mathbf{GF}(a) \wedge \mathbf{G}(a \Rightarrow \mathbf{F}(b)) \vdash \mathbf{GF}(a) \wedge \mathbf{GF}(b)$ методом от противного. Пусть существует такой путь, что на нём имеем справедливость формул $\mathbf{GF}(a) \wedge \mathbf{G}(a \Rightarrow \mathbf{F}(b))$ и $\neg(\mathbf{GF}(a) \wedge \mathbf{GF}(b)) = \mathbf{FG}(\neg a) \vee \mathbf{FG}(\neg b)$. Тогда достаточно рассмотреть следующие два варианта: 1) истинность $\mathbf{FG}(\neg a) = \neg\mathbf{GF}(a)$ противоречит подформуле $\mathbf{GF}(a)$ из посылки, 2) если верна $\mathbf{FG}(\neg b)$, то $\mathbf{G}(a \Rightarrow \mathbf{F}(b)) = \mathbf{G}(\neg a \vee \mathbf{F}(b))$ будет выполняться на этом пути только в виде $\mathbf{FG}(\neg a)$, что соответствует предыдущему варианту 1. Пришли к противоречию. Следовательно, верно обратное, т. е. утверждение на шаге 2 доказано. □

Обозначим доказанную эквивалентность (утверждение 1) как правило $R1$. Перечислим ещё ряд правил, которые будут использоваться для доказательства следующих утверждений:

$R1:$	$\mathbf{GF}(a) \wedge \mathbf{G}(a \Rightarrow \mathbf{F}(b)) = \mathbf{GF}(a) \wedge \mathbf{GF}(b)$	$R10:$	$\mathbf{GFG}(p) = \mathbf{FG}(p)$
$R2:$	$p \Rightarrow q = \neg p \vee q$	$R11:$	$\mathbf{FGF}(p) = \mathbf{GF}(p)$
$R3:$	$p \vee q = p \vee q \wedge \neg p$	$R12:$	$\mathbf{GF}(p \vee q) = \mathbf{GF}(p) \vee \mathbf{GF}(q)$
$R4:$	$\neg\neg p = p$	$R13:$	$\mathbf{F}(\mathit{False}) = \mathit{False}$
$R5:$	$\neg\mathbf{G}(p) = \mathbf{F}(\neg p)$	$R14:$	$p \vee \mathit{False} = p$
$R6:$	$\neg\mathbf{F}(p) = \mathbf{G}(\neg p)$	$R15:$	$p \vee q = p$, если $q \vdash p$
$R7:$	$\neg\mathbf{FG}(p) = \mathbf{GF}(\neg p)$	$R16:$	$p \wedge q = p$, если $p \vdash q$
$R8:$	$\neg\mathbf{GF}(p) = \mathbf{FG}(\neg p)$	$R17:$	$\neg(p \wedge q) = (\neg p \vee \neg q)$
$R9:$	$\mathbf{F}(p \vee q) = \mathbf{F}(p) \vee \mathbf{F}(q)$	$R18:$	$\neg\mathbf{X}(p) = \mathbf{X}(\neg p)$

Правила $R2, \ldots, R18$ — хорошо известные эквивалентные преобразования или следствия из семантики языка LTL, которые легко могут быть доказаны, как, например, это было сделано для $R1$.

Утверждение 2 (Об эквивалентности). *Шаблон справедливости № 1 представляет собой сильную справедливость:* $\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(act_0))\,] = \mathbf{GF}(act_1) \Rightarrow \mathbf{GF}(act_0 \vee rea)$.

Доказательство.

$$\mathbf{G}(\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(act_0))\,) = \qquad [R2, R5, R4]$$

$$\mathbf{G}(\,\mathbf{F}(rea) \vee \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(act_0))\,) = \qquad [R3]$$

$$\mathbf{G}(\,\mathbf{F}(rea) \vee \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(act_0)) \wedge \neg\mathbf{FG}(\neg act_1)\,) = \qquad [R7, R4]$$

$$\mathbf{G}(\,\mathbf{F}(rea) \vee \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(act_0)) \wedge \ \mathbf{GF}(\ act_1)\,) = \qquad [R1]$$

$$\mathbf{G}(\,\mathbf{F}(rea) \vee \mathbf{FG}(\neg act_1) \vee \mathbf{GF}(act_0) \wedge \mathbf{GF}(act_1)\,) = \qquad [R7, R4]$$

$$\mathbf{G}(\,\mathbf{F}(rea) \vee \mathbf{FG}(\neg act_1) \vee \mathbf{GF}(act_0) \wedge \neg\mathbf{FG}(\neg act_1)\,) = \quad [R3]$$
$$\mathbf{G}(\,\mathbf{F}(rea) \vee \mathbf{FG}(\neg act_1) \vee \mathbf{GF}(act_0)\,) = \quad [R11]$$
$$\mathbf{G}(\,\mathbf{F}(rea) \vee \mathbf{FG}(\neg act_1) \vee \mathbf{FGF}(act_0)\,) = \quad [R9]$$
$$\mathbf{GF}(\,rea \vee \mathbf{G}(\neg act_1) \vee \mathbf{GF}(act_0)\,) = \quad [R12]$$
$$\mathbf{GF}(rea) \vee \mathbf{GFG}(\neg act_1) \vee \mathbf{GFGF}(act_0) = \quad [R10, R11]$$
$$\mathbf{GF}(rea) \vee \mathbf{FG}(\neg act_1) \vee \mathbf{GF}(act_0) = \quad [R12]$$
$$\mathbf{FG}(\neg act_1) \vee \mathbf{GF}(act_0 \vee rea) = \quad [R2, R7, R4]$$
$$\mathbf{GF}(act_1) \Rightarrow \mathbf{GF}(act_0 \vee rea). \quad \square$$

Утверждение 3 (О частном случае). *Формула* $\mathbf{G}(\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1)\,)$ *является частным случаем формулы* (16) *при* $act_0 = False$.

Доказательство.

$$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow \mathbf{F}(False))\,] = \quad [R13]$$
$$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(act_1 \Rightarrow False)\,] = \quad [R2]$$
$$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(\neg act_1 \vee False)\,] = \quad [R14]$$
$$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1) \vee \mathbf{G}(\neg act_1)\,] = \quad [R15]$$
$$\mathbf{G}[\,\mathbf{G}(\neg rea) \Rightarrow \mathbf{FG}(\neg act_1)\,]. \quad \square$$

Утверждение 4 (О частном случае). *Формула* $\mathbf{GF}(act_1 \Rightarrow rea)$ *является частным случаем формулы* (19) *при* $act_0 = \neg act_1$.

Доказательство.

$$\mathbf{GF}(act_1) \Rightarrow \mathbf{GF}(\neg act_1 \vee rea) = \quad [R2]$$
$$\neg\mathbf{GF}(act_1) \vee \mathbf{GF}(\neg act_1 \vee rea) = \quad [R12]$$
$$\neg\mathbf{GF}(act_1) \vee \mathbf{GF}(\neg act_1) \vee \mathbf{GF}(rea) = \quad [R7]$$
$$\neg\mathbf{GF}(act_1) \vee \neg\mathbf{FG}(act_1) \vee \mathbf{GF}(rea) = \quad [R17]$$
$$\neg(\mathbf{GF}(act_1) \wedge \mathbf{FG}(act_1)) \vee \mathbf{GF}(rea) = \quad [R16]$$
$$\neg\mathbf{FG}(\ \ act_1) \vee \mathbf{GF}(rea) = \quad [R7]$$
$$\mathbf{GF}(\neg act_1) \vee \mathbf{GF}(rea) = \quad [R12]$$
$$\mathbf{GF}(\neg act_1 \vee rea) = \quad [R2]$$
$$\mathbf{GF}(act_1 \Rightarrow rea). \quad \square$$

Приложение 4. SMV-спецификация поведения КФС: ПЛУ и её окружения

```
MODULE Timer
VAR
    In : boolean; -- Input  (Logical Control Program Unit Output)
    Q  : boolean; -- Output (Logical Control Program Unit Input)
ASSIGN
    INIT ( !Q & !In )
    TRANS( !Q &  next(Q) ->  In )
    TRANS(  Q & !next(Q) -> !In )
    TRANS(  Q &  next(Q) ->  In )
    FAIRNESS (In -> Q) -- Timer Fair Turn On.
```

```
MODULE main
VAR
-- Buttons:
PBStart : boolean;  PBStop : boolean;  PBCompl : boolean;  PBConvr : boolean;
-- Imaginary Sensors:
IFS : boolean;
-- Sensors:
FS1 : boolean;  FS2 : boolean;  OLS : boolean;  CLS : boolean;  WS0 : boolean;
WS1 : boolean;  UTS : boolean;  LTS : boolean;  WTS : boolean;
-- Timers:
FTmr : Timer;   HTmr : Timer;   CTmr : Timer;   MTmr : Timer;
-- Variables:
SysOn : boolean;  Compl : boolean;  FErr : boolean;  CErr : boolean;  HErr : boolean;
Disch : boolean;  Mlted : boolean;  Mltng : boolean;  Fin : boolean;
-- Actuators:
Heater : boolean;   FMech : boolean;   Convr : boolean;  LwSpd : boolean;
Valve  : boolean;  OpnLid : boolean;  ClsLid : boolean;
-- Behavior
-- Buttons:
INIT ( !PBStart & !PBStop & !PBCompl & !PBConvr )
-- Sensors:
INIT ( !FS1 )
TRANS( !FS1 &  next(FS1) ->  !FS2 & !next(FS2) & !IFS & !next(IFS) & Convr )
TRANS(  FS1 & !next(FS1) ->  !FS2 & !next(FS2) & !IFS &  next(IFS) & Convr )
INIT ( !IFS )
TRANS( !IFS &  next(IFS) ->  !FS2 & !next(FS2) &  FS1 & !next(FS1) & Convr )
TRANS(  IFS & !next(IFS) ->  !FS2 &  next(FS2) & !FS1 & !next(FS1) & Convr )
INIT ( !FS2 )
TRANS( !FS2 &  next(FS2) ->   IFS & !next(IFS) & !FS1 & !next(FS1) & Convr )
TRANS(  FS2 & !next(FS2) ->  !IFS & !next(IFS) & !FS1 & !next(FS1) & Convr )
INIT ( !OLS )
TRANS( !OLS &  next(OLS) ->  !CLS & !next(CLS) & OpnLid )
TRANS(  OLS & !next(OLS) ->  !CLS & !next(CLS) & ClsLid )
INIT ( !CLS )
TRANS( !CLS &  next(CLS) ->  !OLS & !next(OLS) & ClsLid )
TRANS(  CLS & !next(CLS) ->  !OLS & !next(OLS) & OpnLid )
INIT ( !WS0 )
TRANS( !WS0 &  next(WS0) ->  !WS1 & !next(WS1) & FMech & !CLS )
TRANS(  WS0 & !next(WS0) ->  !WS1 & !next(WS1) & Valve &  WTS )
INIT ( !WS1 )
TRANS( !WS1 &  next(WS1) ->  WS0 & next(WS0) & FMech & !CLS )
TRANS(  WS1 & !next(WS1) ->  WS0 & next(WS0) & Valve &  WTS )
INIT ( !UTS )
TRANS( !UTS &  next(UTS) ->  WTS & next(WTS) & LTS & next(LTS) & Heater )
TRANS(  UTS & !next(UTS) ->  WTS & next(WTS) & LTS & next(LTS)          )
INIT ( !LTS )
TRANS( !LTS &  next(LTS) ->  WTS & next(WTS) & !UTS & !next(UTS) & Heater )
TRANS(  LTS & !next(LTS) ->  WTS & next(WTS) & !UTS & !next(UTS)          )
INIT ( !WTS )
TRANS( !WTS &  next(WTS) ->  !LTS & !next(LTS) & !UTS & !next(UTS) & Heater )
TRANS(  WTS & !next(WTS) ->  !LTS & !next(LTS) & !UTS & !next(UTS)          )
-- Control Software
-- Variables:
INIT ( !FErr )
```

```
TRANS( !FErr &  next(FErr) ->  (next(FTmr.Q) & !next(WS1)) )
TRANS( !FErr & !next(FErr) -> !(next(FTmr.Q) & !next(WS1)) )
TRANS(  FErr & !next(FErr) ->  next(PBStop) )
TRANS(  FErr &  next(FErr) -> !next(PBStop) )
INIT ( !CErr )
TRANS( !CErr &  next(CErr) ->  (next(CTmr.Q) & !next(FS2)) )
TRANS( !CErr & !next(CErr) -> !(next(CTmr.Q) & !next(FS2)) )
TRANS(  CErr & !next(CErr) ->  next(PBStop) )
TRANS(  CErr &  next(CErr) -> !next(PBStop) )
INIT ( !HErr )
TRANS( !HErr &  next(HErr) ->  (!next(WTS) & (next(HTmr.Q) | Heater & WTS)) )
TRANS( !HErr & !next(HErr) -> !(!next(WTS) & (next(HTmr.Q) | Heater & WTS)) )
TRANS(  HErr & !next(HErr) ->  next(PBStop) )
TRANS(  HErr &  next(HErr) -> !next(PBStop) )
INIT ( !Fin )
TRANS( next(Fin) <-> (Compl & next(OLS) & !next(WS0) | next(FErr) | next(HErr) |
                      next(CErr) | next(PBStop) | next(PBConvr)) )
INIT ( !SysOn )
TRANS( !SysOn &  next(SysOn) ->  (next(PBStart) & !next(Fin)) )
TRANS( !SysOn & !next(SysOn) -> !(next(PBStart) & !next(Fin)) )
TRANS(  SysOn & !next(SysOn) ->  next(Fin) )
TRANS(  SysOn &  next(SysOn) -> !next(Fin) )
INIT ( !Compl )
TRANS( !Compl &  next(Compl) ->  (next(SysOn) & next(PBCompl)) )
TRANS( !Compl & !next(Compl) -> !(next(SysOn) & next(PBCompl)) )
TRANS(  Compl & !next(Compl) -> !next(SysOn) )
TRANS(  Compl &  next(Compl) ->  next(SysOn) )
INIT ( !Mltng )
TRANS( next(Mltng) <-> next(SysOn) & next(CLS) & next(Disch) & next(WS0) )
INIT ( !Mlted )
TRANS( !Mlted &  next(Mlted) ->  (next(MTmr.Q) & next(WTS) & next(WS0) & next(CLS)) )
TRANS( !Mlted & !next(Mlted) -> !(next(MTmr.Q) & next(WTS) & next(WS0) & next(CLS)) )
TRANS(  Mlted & !next(Mlted) ->  (!next(WTS) | !next(CLS) | !next(WS0)) )
TRANS(  Mlted &  next(Mlted) -> !(!next(WTS) | !next(CLS) | !next(WS0)) )
INIT ( !Disch )
TRANS( !Disch &  next(Disch) ->  (next(OLS) & next(WS1)) )
TRANS( !Disch & !next(Disch) -> !(next(OLS) & next(WS1)) )
TRANS(  Disch & !next(Disch) ->  (!next(FS2) & !next(WS1)) )
TRANS(  Disch &  next(Disch) -> !(!next(FS2) & !next(WS1)) )
-- Actuators:
INIT ( !Heater )
TRANS( !Heater &  next(Heater) ->  (next(SysOn) & !next(LTS)) )
TRANS( !Heater & !next(Heater) -> !(next(SysOn) & !next(LTS)) )
TRANS(  Heater & !next(Heater) ->  (!next(SysOn) | next(UTS)) )
TRANS(  Heater &  next(Heater) -> !(!next(SysOn) | next(UTS)) )
INIT ( !ClsLid )
TRANS( next(ClsLid) <-> next(SysOn) & !next(CLS) & next(Disch) )
INIT ( !OpnLid )
TRANS( next(OpnLid) <-> next(SysOn) & !next(OLS) & !next(Disch) )
INIT ( !Convr )
TRANS( next(Convr) <-> (next(SysOn) & (next(FS2) & next(Disch) & !next(WS0) |
                        !next(FS2)) | next(PBConvr)) )
INIT ( !LwSpd )
TRANS( !LwSpd &  next(LwSpd) ->  next(FS1) )
```

```
TRANS( !LwSpd & !next(LwSpd) -> !next(FS1) )
TRANS(  LwSpd & !next(LwSpd) ->  next(FS2) )
TRANS(  LwSpd &  next(LwSpd) -> !next(FS2) )
INIT ( !FMech )
TRANS( next(FMech) <-> next(SysOn) & !next(Disch) & next(OLS) & !next(WS1) )
INIT ( !Valve )
TRANS( next(Valve) <-> next(SysOn) & next(Disch) & next(Mlted) & next(FS2) )
-- Impact_On_Timers:
TRANS( next(FTmr.In) <-> next(FMech)  & !next(WS1) )
TRANS( next(HTmr.In) <-> next(Heater) & !next(WTS) )
TRANS( next(CTmr.In) <-> next(Convr)  & !next(FS2) & next(SysOn) )
TRANS( next(MTmr.In) <-> next(Mltng)  &  next(WTS) & next(WS0) & next(CLS) )
-- Fairness Conditions:
COMPASSION (Convr, !FS2)
COMPASSION (OpnLid, ClsLid | OLS)
COMPASSION (ClsLid, OpnLid | CLS)
COMPASSION (Valve & WTS, (FMech & !CLS) | !WS0)
FAIRNESS   (!Heater -> !WTS)
-- [no use] COMPASSION (ClsLid, OpnLid | !OLS)
-- [no use] COMPASSION (OpnLid, ClsLid | !CLS)
-- [no use] COMPASSION (Valve & WTS, (FMech & !CLS) | !WS1)
-- [no use] FAIRNESS (!Heater -> !UTS)
-- [no use] FAIRNESS (!Heater -> !LTS)
```

Приложение 5. Императивная LTL-спецификация поведения ПЛУ

```
!FErr &
G( !FErr & X(FTmr.Q) & !X(WS1) ->  X(FErr) ) &
G(  FErr & X(PBStop)           -> !X(FErr) ) &
G( !(!FErr & X(FTmr.Q) & !X(WS1)) & !(FErr & X(PBStop)) -> (X(FErr) <-> FErr) ) &
!CErr &
G( !CErr & X(CTmr.Q) & !X(FS2) ->  X(CErr) ) &
G(  CErr & X(PBStop)           -> !X(CErr) ) &
G( !(!CErr & X(CTmr.Q) & !X(FS2)) & !(CErr & X(PBStop)) -> (X(CErr) <-> CErr) ) &
!HErr &
G( !HErr & (!X(WTS) & (X(HTmr.Q) | Heater & WTS)) ->  X(HErr) ) &
G(  HErr & X(PBStop)                              -> !X(HErr) ) &
G( !(!HErr & (!X(WTS) & (X(HTmr.Q) | Heater & WTS)) & !(HErr & X(PBStop)) ->
                                                     (X(HErr) <-> HErr) ) &
!Fin &
G( X(Fin) <-> (Compl & X(OLS) & !X(WS0) | X(fErr) | X(HErr) | X(CErr) |
                                          X(PBStop) | X(PBConvr)) ) &
!SysOn &
G( !SysOn & X(PBStart) & !X(Fin) ->  X(SysOn) ) &
G(  SysOn & X(Fin)               -> !X(SysOn) ) &
G( !(!SysOn & X(PBStart) & !X(Fin)) & !(SysOn & X(Fin)) -> (X(SysOn) <-> SysOn) ) &
!Compl &
G( !Compl &  X(SysOn) & X(PBCompl) ->  X(Compl) ) &
G(  Compl & !X(SysOn)              -> !X(Compl) ) &
G( !(!Compl & X(SysOn) & X(PBCompl)) & !(Compl & !X(SysOn)) -> (X(Compl)<->Compl) ) &
!Mltng &
G( X(Mltng) <-> X(SysOn) & X(CLS) & X(Disch) & X(WS0) ) &
!Mlted &
G( !Mlted & X(MTmr.Q) & X(WTS) &  X(WS0) & X(CLS) ->  X(Mlted) ) &
```

```
    G(  Mlted & (!X(WTS) | !X(CLS) | !X(WS0))         -> !X(Mlted) ) &
    G( !(!Mlted & X(MTmr.Q) & X(WTS) & X(WS0) & X(CLS)) &
       !(Mlted & (!X(WTS) | !X(CLS) | !X(WS0))) -> (X(Mlted) <-> Mlted) ) &
    !Disch &
    G( !Disch &  X(OLS) &  X(WS1) ->  X(Disch) ) &
    G(  Disch & !X(FS2) & !X(WS1) -> !X(Disch) ) &
    G( !(!Disch & X(OLS) & X(WS1)) & !(Disch & !X(FS2) & !X(WS1)) ->
                                          (X(Disch) <-> Disch) ) &
    -- Actuators:
    !Heater &
    G( !Heater & ( X(SysOn) & !X(LTS)) ->  X(Heater) ) &
    G(  Heater & (!X(SysOn) |  X(UTS)) -> !X(Heater) ) &
    G( !(!Heater & (X(SysOn) & !X(LTS))) & !(Heater & (!X(SysOn) |  X(UTS))) ->
                                                  (X(Heater) <-> Heater) ) &
    !ClsLid &
    G( X(ClsLid) <-> X(SysOn) & !X(CLS) & X(Disch) ) &
    !OpnLid &
    G( X(OpnLid) <-> X(SysOn) & !X(OLS) & !X(Disch) ) &
    !Convr &
    G( X(Convr) <-> (X(SysOn) & (X(FS2) & X(Disch) & !X(WS0) | !X(FS2)) | X(PBConvr)) ) &
    !LwSpd &
    G( !LwSpd & X(FS1) ->  X(LwSpd) ) &
    G(  LwSpd & X(FS2) -> !X(LwSpd) ) &
    G( !(!LwSpd & X(FS1)) & !(LwSpd & X(FS2)) -> (X(LwSpd) <-> LwSpd))
    G( X(FMech) <-> X(SysOn) & !X(Disch) & X(OLS) & !X(WS1) ) & !FMech &
    G( X(Valve) <-> X(SysOn) & X(Disch) & X(Mlted) & X(FS2) ) & !Valve &
    -- Impact_On_Timers:
    G( X(FTmr.In) <-> X(FMech)  & !X(WS1) )                  & !FTmr.In &
    G( X(HTmr.In) <-> X(Heater) & !X(WTS) )                  & !HTmr.In &
    G( X(CTmr.In) <-> X(Convr)  & !X(FS2) & X(SysOn) )       & !CTmr.In &
    G( X(MTmr.In) <-> X(Mltng)  &  X(WTS) & X(WS0) & X(CLS) ) & !MTmr.In
```

Приложение 6. ST-программа управления установкой для литья пластмасс

```
PROGRAM PLC_PRG
VAR_INPUT
    PBStart, PBStop, PBCompl, PBConvr: BOOL := FALSE;           (* Buttons *)
    FS1, FS2, OLS, CLS, WS0, WS1, UTS, LTS, WTS: BOOL := FALSE; (* Sensors *)
END_VAR
VAR_OUTPUT
    SysOn, Compl, FErr, CErr, HErr, Disch, Mltng, Mlted: BOOL := FALSE; (* Lamps     *)
    Heater, FMech, Convr, LwSpd, Valve, OpnLid, ClsLid: BOOL := FALSE;  (* Actuators *)
END_VAR
VAR
    (* Timers *)
    MTmr: TON := (In := FALSE, Q := FALSE, PT := T#6s );
    HTmr: TON := (In := FALSE, Q := FALSE, PT := T#12s);
    FTmr: TON := (In := FALSE, Q := FALSE, PT := T#10s);
    CTmr: TON := (In := FALSE, Q := FALSE, PT := T#18s);
    Fin: BOOL := FALSE;     (* Internal variable *)
    (* Auxiliary variables *)
    _SysOn, _Compl, _Mlted, _Disch, _Heater : BOOL := FALSE;
    _LwSpd, _CErr, _FErr, _HErr, _WTS : BOOL := FALSE;
END_VAR
```

```
(* Calling timers *)
MTmr(); HTmr(); FTmr(); CTmr();
(* Variables *)
IF NOT _FErr AND FTmr.Q AND NOT WS1 THEN FErr:=1;  (* FErr *)
ELSIF  _FErr AND PBStop             THEN FErr:=0;
END_IF;
IF NOT _CErr AND CTmr.Q AND NOT FS2 THEN CErr:=1;  (* CErr *)
ELSIF  _CErr AND PBStop             THEN CErr:=0;
END_IF;
IF NOT _HErr AND NOT WTS AND
   (HTmr.Q OR _Heater AND _WTS) THEN HErr:=1;  (* HErr *)
ELSIF  _HErr AND PBStop         THEN HErr:=0;
END_IF;
Fin:= _Compl AND OLS AND NOT WS0 OR FErr OR HErr OR CErr OR
       PBStop OR PBConvr;  (* Fin *)
IF NOT _SysOn AND PBStart AND NOT Fin THEN SysOn:=1;  (* SysOn *)
ELSIF  _SysOn AND                 Fin THEN SysOn:=0;
END_IF;
IF NOT _Compl AND     SysOn AND PBCompl THEN Compl:=1;  (* Compl *)
ELSIF  _Compl AND NOT SysOn             THEN Compl:=0;
END_IF;
Mltng := SysOn AND CLS AND Disch AND WS0;  (* Mltng *)
IF NOT _Mlted AND MTmr.Q AND WTS AND WS0 AND CLS  THEN Mlted:=1;  (* Mlted *)
ELSIF  _Mlted AND (NOT WTS OR NOT CLS OR NOT WS0) THEN Mlted:=0;
END_IF;
IF NOT _Disch AND OLS     AND     WS1 THEN Disch:=1;  (* Disch *)
ELSIF  _Disch AND NOT FS2 AND NOT WS1 THEN Disch:=0;
END_IF;
(* Actuators *)
IF NOT _Heater AND      SysOn AND NOT LTS  THEN Heater:=1;  (* Heater *)
ELSIF  _Heater AND (NOT SysOn OR      UTS) THEN Heater:=0;
END_IF;
ClsLid:= SysOn AND NOT CLS AND     Disch;                              (* ClsLid *)
OpnLid:= SysOn AND NOT OLS AND NOT Disch;                              (* OpnLid *)
Convr := SysOn AND (FS2 AND Disch AND NOT WS0 OR NOT FS2) OR PBConvr;  (* Convr  *)
IF NOT _LwSpd AND FS1 THEN LwSpd:=1;                                   (* LwSpd  *)
ELSIF  _LwSpd AND FS2 THEN LwSpd:=0;
END_IF;
FMech := SysOn AND NOT Disch AND OLS AND NOT WS1;                      (* FMech  *)
Valve := SysOn AND     Disch AND Mlted AND FS2;                        (* Valve  *)
(* Impact_On_Timers *)
FTmr.In:= FMech  AND NOT WS1;              (* FTmr.In *)
HTmr.In:= Heater AND NOT WTS;              (* HTmr.In *)
CTmr.In:= Convr  AND SysOn AND NOT FS2;    (* CTmr.In *)
MTmr.In:= Mltng  AND WTS AND WS0 AND CLS;  (* MTmr.In *)
(* Pseudo operator section: saving previous values of variables *)
_SysOn:=SysOn; _Compl:=Compl; _Disch:=Disch; _Mlted:=Mlted; _Heater:=Heater;
_LwSpd:=LwSpd; _FErr:=FErr; _CErr:=CErr; _HErr:=HErr; _WTS:=WTS;
```