Модел. и анализ информ. систем. Т. **20**, № **4** (2013) 5–22
©*Кузьмин Е. В., Соколов В. А., Рябухин Д. А.,* 2013

УДК 517.9

Построение и верификация ПЛК-программ по LTL-спецификации

Кузьмин Е. В.[1], Соколов В. А.[2], Рябухин Д. А.

Ярославский государственный университет им. П. Г. Демидова
150000 Россия, г. Ярославль, ул. Советская, 14

e-mail: {kuzmin,sokolov}@uniyar.ac.ru, dmitriy_ryabukhin@mail.ru

получена 20 мая 2013

Ключевые слова: программируемые логические контроллеры, технология программирования, спецификация и верификация программ

Предлагается подход к построению и верификации программ логических контроллеров (ПЛК) для «дискретных» задач. Спецификация программного поведения проводится на языке темпоральной логики линейного времени LTL. Программирование осуществляется на языке ST (Structured Text) по LTL-спецификации. Анализ корректности LTL-спецификации производится с помощью программного средства символьной проверки модели Cadence SMV. Предлагаемый подход к программированию и верификации программ ПЛК демонстрируется на примере. Для дискретной задачи приводятся ST-программа, ее LTL-спецификация и SMV-модель.

Целью статьи является описание подхода к программированию ПЛК, который бы обеспечивал возможность анализа корректности ПЛК-программ с помощью метода проверки модели.

Поэтому изменение значения каждой программной переменной описывается с помощью пары LTL-формул. Первая LTL-формула описывает ситуации, при которых происходит возрастание значения соответствующей переменной, вторая LTL-формула задает условия, приводящие к уменьшению значения переменной. Рассматриваемые для спецификации поведения переменных LTL-формулы являются конструктивными в том смысле, что по ним производится построение ПЛК-программы, которая соответствует темпоральным свойствам, выраженным этими формулами. Таким образом, программирование ПЛК сводится к построению LTL-спецификации поведения каждой программной переменной. Кроме этого, по LTL-спецификации строится SMV-модель, которая затем проверяется на корректность (относительно дополнительных общепрограммных LTL-свойств) методом проверки модели с помощью средства верификации Cadence SMV.

[1]Работа проводилась при финансовой поддержке РФФИ, грант №12-01-00281-а.

[2]Работа проводилась при финансовой поддержке Минобрнауки РФ в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы, соглашение №14.B37.21.0392 от 06.08.2012.

Введение

Применение программируемых логических контроллеров (ПЛК) в системах управления сложными производственными процессами предъявляет строгие требования корректности к программам ПЛК. Любая программная ошибка считается недопустимой. Несмотря на это существующие средства разработки программ для ПЛК, например широко известный комплекс CoDeSys (Controller Development System) [7], предоставляют лишь обычные возможности отладки программ через тестирование посредством визуализации объектов управления ПЛК. Вместе с тем программирование логических контроллеров представляет собой прикладную область, в которой существующие наработки в области формальных методов моделирования и анализа программных систем могли бы иметь успешное применение — внедрение формальных методов в процесс создания программ на уровне отлаженной технологии.

ПЛК — «реагирующая» система, имеющая множество входов, подключенных посредством датчиков к объекту управления, и множество выходов, подключенных к исполнительным устройствам [9, 4]. Программа ПЛК выполняется в рабочем цикле: считывание входов, выполнение программы и выставление выходов.

Языки программирования ПЛК определяются стандартом МЭК 61131, включающим в себя описание пяти языков: SFC, IL, ST, LD и FBD. Эти языки позволяют применять для ПЛК-программ существующие методы анализа корректности — тестирование, дедуктивный анализ (theorem proving) [8] и автоматический метод проверки модели (model checking) [6]. Дедуктивный анализ в большей степени применим к «непрерывным» задачам обеспечения устойчивости и качества регулирования, требующим программирования соответствующей системы формул. Метод проверки модели наиболее подходит для «дискретных» задач логического управления, для реализации которых требуется ПЛК с бинарными входами и выходами, что обеспечивает конечное пространство возможных состояний программы ПЛК.

Ранее в [2] проводился обзор подходов к программированию «дискретных» задач ПЛК на языках LD, SFC и ST. Оценивалось удобство анализа программной корректности методом проверки модели. Были выявлены возможные уязвимости ПЛК-программ и трудности верификации, возникающие при традиционных подходах к программированию. В частности, существующие работы по теме анализа корректности программ ПЛК [5, 10, 11] главным образом посвящены построению трансляторов со стандартных языков МЭК 61131 в интерфейсные языки программных средств верификации методом проверки модели. Демонстрация результатов проводится на тривиальных примерах. Однако опыт работы с практическими задачами логического управления показал, что прямая трансляция ничего не дает для последующего анализа программных свойств, поскольку зачастую не представляется возможным выразить требуемые свойства на языке темпоральной логики.

В этой статье предлагается подход к построению «дискретных» ПЛК-программ, обеспечивающий возможность анализа их корректности с помощью метода проверки модели, — программирование исходя из задач спецификации и верификации. В качестве языка спецификации программного поведения предлагается использовать язык темпоральной логики LTL. Программирование осуществляется на языке ST по LTL-спецификации. Анализ корректности LTL-спецификации производится с помощью программного средства символьной проверки модели Cadence SMV [12].

1. Метод проверки модели для ПЛК-программ

Задача проверки модели (Model Checking) состоит в определении выполнимости для конечной модели программы (в виде структуры Крипке) свойства, выраженного формулой темпоральной логики.

Структурой Крипке над множеством элементарных высказываний P называется система переходов $\mathcal{S} = (S, s_0, \to, L)$, где S — конечное множество состояний (модели программы), $s_0 \in S$ — начальное состояние, $\to \subseteq S \times S$ — отношение переходов, $L : S \to 2^P$ — функция, помечающая каждое состояние множеством элементарных высказываний, истинных в этом состоянии.

Путь в структуре Крипке из состояния s_0 — это бесконечная последовательность состояний $\pi = s_0 s_1 s_2 \ldots$ такая, что $\forall\, i \geq 0$ выполняется $s_i \to s_{i+1}$.

В качестве языка спецификации поведенческих свойств программной модели рассматривается язык темпоральной логики линейного времени LTL (Linear-Time Temporal Logic). Выбор логики LTL связывается с тем, что программа ПЛК является классической реактивной (реагирующей) управляющей системой, которая, будучи однажды запущенной, должна иметь корректное бесконечное поведение. Условия корректности удобно задавать в виде шаблонов свойств, которым должны соответствовать корректные исполнения программы. В темпоральной логике LTL каждая формула по сути представляет собой такой шаблон.

Формулы логики LTL строятся по следующей грамматике при $p_i \in P$:

$$\varphi, \psi ::= \mathit{true} \mid p_0 \mid p_1 \mid \ \ldots \ \mid p_n \mid \neg\, \varphi \mid \psi \wedge \varphi \mid X\varphi \mid \psi\, U\, \varphi \mid F\, \varphi \mid G\, \varphi.$$

Формула логики LTL описывает свойство одного пути структуры Крипке, выходящего из некоторого выделенного текущего состояния. Темпоральные операторы X, F, G и U имеют следующую интерпретацию: $X\varphi$ означает, что формула φ должна выполняться в следующем состоянии, $F\varphi$ — φ должна выполняться в некотором будущем состоянии пути, $G\varphi$ — φ должна выполняться в текущем состоянии и во всех будущих состояниях пути, $\psi\, U\, \varphi$ — φ должна выполняться в текущем или будущем состоянии при том, что во всех состояниях (начиная с текущего) до этого момента должна выполняться формула ψ. Операторы F и G являются производными и вводятся для удобства спецификации свойств: $F\, \varphi = \mathit{true}\, U \varphi$, $G\, \varphi = \neg F \neg \varphi$. Кроме того, далее будут использоваться классические логические связки $\vee$ и $\Rightarrow$.

Структура Крипке удовлетворяет формуле (свойству) φ логики LTL, если φ выполняется для всех путей, выходящих из начального состояния s_0.

Модель Крипке для программы ПЛК строится естественным образом. Состояние модели — это состояние программы (вектор значений всех переменных) ПЛК после одного полного прохода рабочего цикла. Начальное состояние модели — состояние программы после инициализации значений переменных. Таким образом, в качестве перехода из одного состояния модели в другое (или то же самое) рассматривается полное исполнение программы за один проход рабочего цикла ПЛК.

Отметим, что модель от исходной ПЛК-программы отличается лишь дискретным представлением работы таймеров (см. [1]). Если таймеры в программе не применяются, то поведение модели полностью совпадает с поведением программы.

В качестве элементарных высказываний модели будут рассматриваться логические (с применением арифметических операторов и операторов сравнения) выражения над переменными ПЛК-программы.

2. Концепция программирования

Целью статьи является описание подхода к программированию ПЛК, который бы обеспечивал возможность анализа корректности ПЛК-программ методом проверки модели. При описании подхода к программированию будем исходить из соображений удобства и простоты применения этого метода. Необходимость прозрачного и наглядного представления о том, каким образом происходит изменение значения той или иной программной переменной за один переход из состояния в состояние модели ПЛК-программы, требует соблюдения следующих двух условий.

Условие 1. Значение каждой переменной должно изменяться не более одного раза за одно полное выполнение программы при прохождении рабочего цикла ПЛК.

Условие 2. Значение каждой переменной должно изменяться только в одном месте программы в некотором операторном блоке без вложенностей.

Очевидно, что за один проход рабочего цикла значение любой переменной возрастает, убывает или остается без изменения (если, например, не обращаться к ней по присваиванию) по отношению к ее значению, полученному на предыдущем проходе рабочего цикла. Потребуем проводить изменение значения переменной только в том случае, когда это действительно необходимо, т. е. запретим обращение к переменной по присваиванию, если не будут выполнены условия обязательного изменения ее значения. При таком подходе требование того, каким образом должно изменяться значение некоторой переменной V за один проход рабочего цикла ПЛК, удобно записывать в виде формул темпоральной логики LTL следующим образом.

Для описания ситуаций, которые приводят к увеличению значения переменной V, предлагается использовать LTL-формулу вида

$$\mathbf{G}\,\mathbf{X}(\,V{>}_V \Rightarrow \mathit{OldValCond} \wedge \mathit{FiringCond} \wedge V = \mathit{NewValExpr}), \tag{1}$$

которая означает, что всякий раз, когда новое значение переменной V оказывается больше ее предыдущего значения, записанного в переменной $_V$, из этого следует, что старое значение переменой V удовлетворяло условию *OldValCond*, было выполнено условие соответствующего внешнего воздействия *FiringCond*, а новое значение переменной V является значением выражения *NewValExpr*.

Символ лидирующего подчеркивания «_» в обозначении переменной $_V$ удобно воспринимать как псевдооператор, позволяющий обратиться к значению переменной V, которое она имела в предыдущем состоянии. При этом псевдооператор может использоваться только под действием темпорального оператора **X**.

Условия *FiringCond* и *OldValCond* являются логическим выражением над программными переменными и константами, которые строятся с применением операторов сравнения, логических и арифметических операторов и псевдооператора «_» (который по определению может быть применим только к переменным). Выражение *FiringCond* описывает ситуации, при которых возникает необходимость изменения значения переменной V, если это, конечно, допускается условием *OldValCond*. Выражение *NewValExpr* строится с помощью переменных и констант, операторов сравнения, логических, арифметических операторов и псевдооператора «_».

Для описания всех возможных ситуаций, при которых происходит возрастание значения переменной V, в формуле (1) после оператора $\Rightarrow$ может потребоваться несколько наборов рассмотренных конъюнктивных членов $\mathit{OldValCond}_i \wedge \mathit{FiringCond}_i \wedge V = \mathit{NewValExpr}_i$ объединенных в дизъюнкцию.

Аналогичным образом описываются ситуации, приводящие к уменьшению значения переменной V:

$$\mathbf{G}\,\mathbf{X}(\,V{<}_V \Rightarrow \mathit{OldValCond}' \wedge \mathit{FiringCond}' \wedge V = \mathit{NewValExpr}'\,). \qquad (1')$$

Темпоральные формулы вида (1) и (1′) описывают желаемое поведение некоторой целочисленной переменной. В случае с переменной логического (двоичного) типа данных для спецификации ее поведения предлагается использовать более простые LTL-формулы. Для описания ситуаций, при которых значение логической переменной V возрастает, можно использовать следующую формулу:

$$\mathbf{G}\,\mathbf{X}(\,\neg_V \wedge V \Rightarrow \mathit{FiringCond}\,). \qquad (2)$$

Аналогичным образом описываются ситуации, приводящие к уменьшению значения логической переменной V:

$$\mathbf{G}\,\mathbf{X}(\,_V \wedge \neg V \Rightarrow \mathit{FiringCond}'\,). \qquad (2')$$

Рассмотрим особый случай спецификаций вида (1) и (1′), при котором для V имеем $\mathit{FiringCond} = \mathit{FiringCond}' = 1$, $\mathit{NewValExpr} = \mathit{NewValExpr}'$, $\mathit{OldValCond} = (_V < \mathit{NewValExpr})$ и $\mathit{OldValCond}' = (_V > \mathit{NewValExpr})$:

$$\mathbf{G}\,\mathbf{X}(\,V > _V \Rightarrow _V < \mathit{NewValExpr} \wedge V = \mathit{NewValExpr}\,);$$

$$\mathbf{G}\,\mathbf{X}(\,V < _V \Rightarrow _V > \mathit{NewValExpr} \wedge V = \mathit{NewValExpr}\,).$$

Такая спецификация может быть заменена на одну LTL-формулу вида

$$\mathbf{G}\,\mathbf{X}(\,V = \mathit{NewValExpr}\,). \qquad (3)$$

Переменную V из спецификации вида (1) и (1′), а также (2) и (2′), будем называть *переменной-регистром*. Если для переменной V строится спецификация вида (3), назовем ее *переменной-функцией*. В особом случае спецификации (3), при котором выражение $\mathit{NewValExpr}$ не содержит псевдооператора лидирующего подчеркивания «_», переменную V будем называть *переменной-подстановкой*.

Каждый из рассмотренных шаблонов LTL-формул спецификации поведения переменной является конструктивным, т. е. по спецификации можно легко построить программу, которая бы соответствовала темпоральным свойствам, выраженным этими формулами. (Реализация этого будет показана далее.) Таким образом, можно сказать, что по сути все программирование ПЛК сводится к построению спецификации поведения каждой программной переменной, являющейся выходом или вспомогательной внутренней переменной. Условие изменения значения переменной только в одном месте программы облегчает отладку, дает возможность простой оценки степени готовности и объема текста программы. При этом стадия написания кода программы заканчивается, как только для каждой выходной и вспомогательной переменной создана спецификация. Отметим, что количество и смысл выходных переменных определяются интерфейсом ПЛК исходя из постановки задачи.

Такой подход к программированию ПЛК в определенном смысле решает проблему полноты спецификации. В данном случае спецификация программы делится на две части: 1) спецификацию поведения всех программных переменных (кроме входов), 2) спецификацию общепрограммных свойств. При этом вторая составляющая спецификации оказывает влияние на количество и смысл внутренних вспомогательных переменных ПЛК-программы.

При построении спецификации важно учитывать то, в каком порядке располагаются темпоральные формулы, описывающие поведение переменных. Некоторая

переменная без псевдооператора «_» может быть задействована в спецификации поведения другой переменной, только если спецификация ее поведения уже произведена и находится выше по тексту.

В спецификации блок темпоральных формул, описывающий поведение некоторой переменной, будем по необходимости сопровождать указанием начального значения этой переменной, которое она получает при инициализации. Для этого задействуется ключевое слово Init. Например, $\mathrm{Init}(V) = 1$ означает, что при инициализации переменная V получает значение 1. Если в спецификации явно не указывается начальное значение переменной, то считается, что это значение равняется нулю.

3. Программирование по LTL-спецификации

В этом разделе мы рассмотрим способ построения программного ST-кода по конструктивной LTL-спецификации поведения программных переменных. В общем виде схема трансляции LTL-формул в программный код на языке ST следующая. Двум темпоральным формулам переменной V, помеченным $V+$ (возрастание значения, (1)) и $V-$ (убывание значения, (1′)), ставится в соответствие текстовый блок IF-ELSIF на языке ST

IF *OldValCond* AND *FiringCond* THEN $V :=$ *NewValExpr*; (* $V+$ *)
ELSIF *OldValCond*′ AND *FiringCond*′ THEN $V :=$ *NewValExpr*′; (* $V-$ *)
END_IF.

Если в LTL-формулах спецификации поведения переменной V количество конъюнктивных блоков $\mathit{OldValCond}_i \wedge \mathit{FiringCond}_i \wedge V = \mathit{NewValExpr}_i$ будет больше рассмотренных двух, то возрастет число альтернативных веток ELSIF (по одно ветке для каждого нового блока).

Отметим, что по построению поведение полученной программы будет полностью удовлетворять формулам LTL-спецификации.

При трансляции упрощенных LTL-формул спецификации поведения логической переменной V в формах $V+$ (2) и $V-$ (2′) получаем блок

IF NOT _V AND *FiringCond* THEN $V := 1$; (* $V+$ *)
ELSIF _V AND *FiringCond*′ THEN $V := 0$; (* $V-$ *) END_IF.

В случае со спецификацией поведения переменной-функции V (3) имеем простое присваивание вида

$V :=$ *NewValExpr*. (* V *)

Каждая программная переменная должна быть определена в разделе (локальном или глобальном) описания переменных и проинициализирована в соответствии со спецификацией. Отметим, что, например, в среде разработки CoDeSys [7] по умолчанию все переменные инициализируются нулем.

Кроме того, необходимо реализовать идею псевдооператора лидирующего подчеркивания «_». Для этого в самом конце программы выделяется место для псевдооператорного раздела, куда после задания поведения всех переменных спецификации для каждой такой переменной V, к прошлому значению которой обращались как _V, добавляется присваивание _$V := V$. При этом переменную _V также необходимо определить в разделе описания переменных с такой же инициализацией, как и для переменной V.

Отметим, что подход к программированию по спецификации, которая, по сути, описывает причину изменения значения каждой программной переменной, выглядит естественным и оправданным, поскольку выходной сигнал ПЛК является управляющим, а смена значения этого управляющего сигнала обычно несет в себе дополнительную смысловую нагрузку. Например, важно четко представлять себе, почему двигатель должен быть запущен/выключен, а некоторая лампа зажжена/погашена. Поэтому кажется очевидным, что каждая переменная должна сопровождаться двумя свойствами, по одному на каждое направление изменений. При этом предполагается, что если условия изменений не выполнены, то переменная сохраняет свое прежнее состояние.

4. Построение SMV-модели по LTL-спецификации

В качестве программного средства анализа корректности методом проверки модели мы рассматриваем верификатор Cadence SMV [12]. После создания спецификации предлагается осуществлять построение модели Крипке на языке SMV с последующей проверкой выполнимости для этой модели общепрограммных свойств. Если некоторое общепрограммное свойство не выполняется для модели, то верификатор строит пример некорректного пути в модели Крипке, по которому вводятся исправления в спецификацию. И только после того, как все программные свойства были проверены с положительным результатом, по спецификации строится ST-программа ПЛК.

Средства языка SMV позволяют с помощью оператора next задавать значение переменной в следующем состоянии модели Крипке, т. е. на следующем шаге (после разового прохождения рабочего цикла ПЛК), относительно новых входных сигналов и последних значений выходов и внутренних переменных. Оператор next позволяет также обращаться к значению переменной, которое уже было вычислено для нового (следующего) состояния модели. Ветвление отношения переходов обеспечивается «недетерминированным» присваиванием. Например, присваивание вида $\mathrm{next}(V) := \{0, 1\}$ означает, что будут порождены состояния и переходы в них как со значением входа $V = 0$, так и со значением $V = 1$. На языке SMV символы «&», «|», «∼» и «->» означают соответственно логические «и», «или», «не» и импликацию.

Язык SMV ориентирован на построение следующих состояний модели Крипке из текущего состояния. При этом начальным текущим состоянием модели является состояние программы после инициализации. Поэтому спецификацию поведения переменной V (1) и (1′) будет удобнее (нагляднее) переписать в следующем равнозначном виде

$$V+\colon \mathbf{G}(\,\mathbf{X}(V{>}_V) \Rightarrow \mathbf{X}(\mathit{OldValCond}\,) \wedge \mathbf{X}(\mathit{FiringCond}\,) \wedge \mathbf{X}(V = \mathit{NewValExpr}\,)),$$
$$V-\colon \mathbf{G}(\,\mathbf{X}(V{<}_V) \Rightarrow \mathbf{X}(\mathit{OldValCond}') \wedge \mathbf{X}(\mathit{FiringCond}') \wedge \mathbf{X}(V = \mathit{NewValExpr}')).$$

И уже затем, ставя в соответствие темпоральному оператору $\mathbf{X}$ оператор next, естественным образом получаем SMV-модель поведения переменной V:

$$\begin{aligned}\mathrm{case}\{\ & \mathrm{next}(\mathit{OldValCond}\,)\ \&\ \mathrm{next}(\mathit{FiringCond}\,) &&: \mathrm{next}(V) := \mathrm{next}(\mathit{NewValExpr}\,);\\ & \mathrm{next}(\mathit{OldValCond}')\ \&\ \mathrm{next}(\mathit{FiringCond}') &&: \mathrm{next}(V) := \mathrm{next}(\mathit{NewValExpr}');\\ & \mathrm{default} &&: \mathrm{next}(V) := V;\ \}.\end{aligned}$$

Здесь ключевое слово default соответствует тому, что должно происходить, если не выполняются условия срабатывания первых двух веток операторного блока case.

В случае с логической переменной V спецификация (2) и (2′) преобразуется в следующую SMV-модель поведения переменной

$$\begin{aligned}\text{case}\{\ &\sim V\ \&\ \text{next}(\mathit{FiringCond}\ ) &&: \text{next}(V) := 1;\\ &V\ \&\ \text{next}(\mathit{FiringCond}') &&: \text{next}(V) := 0;\\ &\text{default} &&: \text{next}(V) := V;\ \}.\end{aligned}$$

В случае с переменной-функцией V при спецификации (3) SMV-модель ее поведения задается просто как $\text{next}(V) := \text{next}(\mathit{NewValExpr})$.

Рассмотрим теперь спецификацию поведения переменной-подстановки V. Напомним, что в этом случае *NewValExpr* не содержит псевдооператора «_». Это позволяет переписать спецификацию в следующем равнозначном виде:

$V\colon\ \mathbf{X}\,\mathbf{G}(\,V = \mathit{NewValExpr}\,)$.

Фактически эта формула означает, что если не учитывать начальное состояние модели, которое формируется инициализацией (с помощью ключевого слова init), то во всех остальных состояниях модели должно выполняться равенство $V = \mathit{NewValExpr}$. А поскольку из справедливости чуть более общей формулы $\mathbf{G}(\,V = \mathit{NewValExpr}\,)$ следует справедливость формулы $\mathbf{X}\,\mathbf{G}(\,V = \mathit{NewValExpr}\,)$, то в качестве конструктивной спецификации для построения SMV-модели поведения переменной-подстановки V можно использовать первую более общую формулу. По этой спецификации SMV-модель строится просто в виде присваивания

$V := \mathit{NewValExpr}$.

Верификатор Cadence SMV позволяет проверять программные модели, содержащие до 59 двоичных переменных (переменные других типов представляются наборами двоичных переменных). При этом переменные-подстановки в это число не включаются, т. е. считаются только переменные-регистры и переменные-функции.

5. Логическая игра «31»

Рассмотрим логическую игру «31», которая формулируется следующим образом. Выкладываются на стол в шесть рядов рубашкой вниз 24 игральные карты. В первом ряду находятся 4 туза (4 единицы), во втором — 4 двойки, в третьем — 4 тройки, в четвертом — 4 четверки, в пятом — 4 пятерки, в шестом — 4 шестерки. Играют два игрока. Игроки ходят по очереди. Сначала ходит первый игрок, а затем — второй. За один ход игрок переворачивает одну карту. Переворачивать можно только те карты, которые лежат рубашкой вниз. Проигрывает тот игрок, после хода которого суммарный номинал перевернутых карт превысит 31.

Задача состоит в программировании ПЛК с 7 дискретными входами и 18 дискретными выходами для реализации игры таким образом, чтобы, играя за второго игрока, ПЛК гарантированно выигрывал всякий раз, когда первый игрок начинает игру с переворачивания тройки, четверки или шестерки. Когда первый игрок начинает игру с туза (единицы), двойки или пятерки, то ПЛК должен гарантированно выигрывать, если первый игрок не использует стратегий полного выбора единиц, двоек или пятерок соответственно.

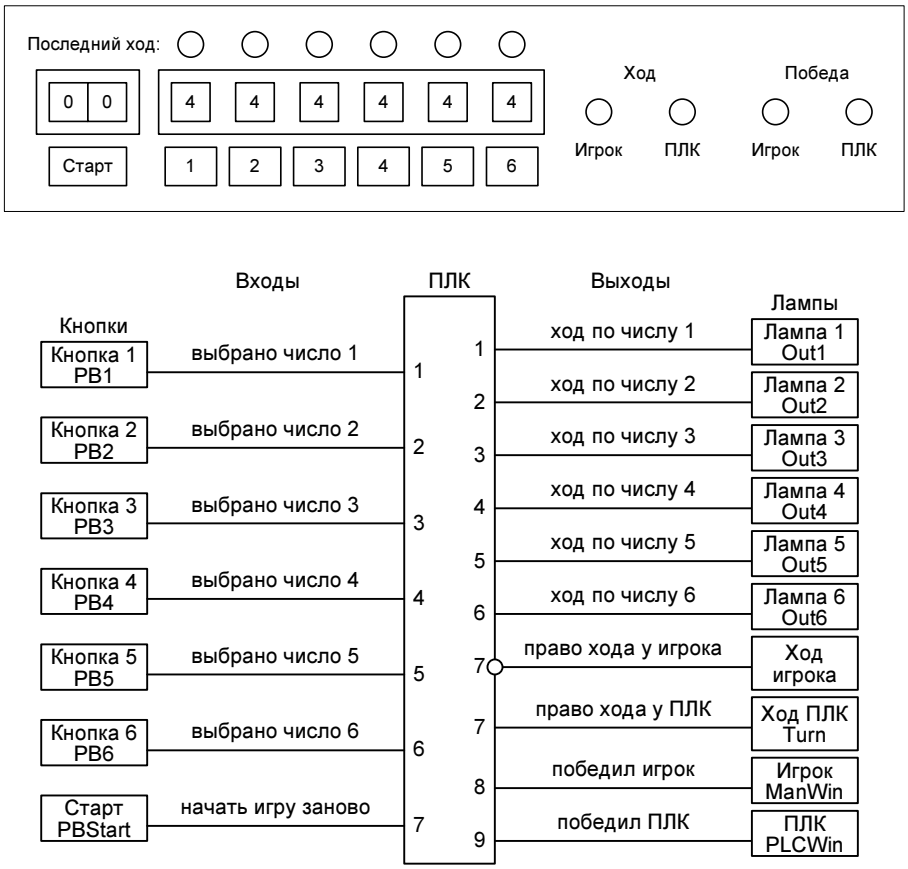

Рис. 1. Панель управления и интерфейс ПЛК управления игрой «31»

На рис. 1 изображена схема панели управления игрой. Кнопки «1», «2», «3», «4», «5» и «6» используются для «переворачивания» соответствующей карты. Нажатие кнопки является некорректным и не засчитывается, если она была нажата одновременно с другой кнопкой. В этом случае она должна быть отжата и затем нажата корректно. После корректного нажатия одной из этих кнопок соответствующее значение на «Дисплее карт» уменьшается на единицу. В исходном состоянии каждый из этих дисплеев отображает цифру 4. Снизу значения, отображаемые дисплеем, ограничены 0. После каждого хода игрока и/или ПЛК сумма значений «перевернутых» карт отображается на «Дисплее суммы». Для удобства визуального восприятия игры после хода игрока должна происходить задержка хода ПЛК на две секунды (в течение которых ПЛК «думает»).

Кнопка «Старт» позволяет начать игру заново. Последний ход показывается с помощью включения одной из шести ламп, расположенных над соответствующей цифрой. Очередность хода обозначается включением соответствующей лампы «Ход игрока» или «Ход ПЛК». Победитель в игре выделяется включением соответственно лампы «Игрок» или лампы «ПЛК».

Интерфейс ПЛК управления игрой также представлен на рис. 1. В целях экономии места на интерфейсе не изображены выходы, идущие на «Дисплей суммы» и «Дисплей карт». Из тех же соображений далее программный код, отвечающий за вывод информации на дисплеи, будет опускаться.

5.1. Программные свойства и вспомогательные переменные

Глобальные переменные программы определяются интерфейсом ПЛК. Кроме необходимости реализации алгоритма решения задачи, введение вспомогательных переменных во многом диктуется необходимостью выражать на языке темпоральной логики LTL общепрограммные свойства, вытекающие из постановки задачи. В данном разделе в качестве примера рассматривается ряд общепрограммных свойств для задачи «Логическая игра 31», которые затем задаются в виде LTL-формул с опорой на вводимые по необходимости вспомогательные переменные.

Важно отметить, что описание некоторых внутренних переменных, а также таймеров, следует непосредственно из постановки задачи. Например, для хранения количества «неперевернутых» карт разного достоинства введем соответственно переменные $V1$, $V2$, $V3$, $V4$, $V5$ и $V6$, а для хранения суммарного значения всех «перевернутых» карт — переменную Sum. Другие внутренние переменные могут появляться исходя из удобства программирования и читаемости программы.

Явными общепрограммными свойствами в случае с логической игрой являются те свойства, которые выражают соответствие поведения программы выигрышным стратегиям, предусмотренным постановкой задачи. Описание стратегии игры предполагает оперирование понятием хода (игрока или ПЛК) как такового, т. е. понятием хода вообще, означающего переворачивание карты, а также понятием конкретного хода, т. е. хода в частности, соответствующего переворачиванию карты определенного достоинства. Для этих целей введем переменные $Mv1$, $Mv2$, $Mv3$, $Mv4$, $Mv5$ и $Mv6$, единичное значение которых будет сигнализировать о том, что был совершен ход, состоящий в переворачивании карты достоинством 1, 2, 3, 4, 5 и 6 соответственно. А единичное значение переменной Mv будет просто указывать на совершение хода игры на данном проходе рабочего цикла ПЛК. Поскольку стратегия игры ПЛК может меняться в зависимости от наличия ресурсов, наличия «неперевернутых» карт, невозможность для ПЛК совершить ход согласно прямой стратегии из-за отсутствия «неперевернутых» карт необходимого достоинства будем определять с помощью переменной Lck.

Для указания необходимости сброса игры в начальное состояние введем переменную Rst. Некорректное нажатие игровых кнопок будем выявлять с помощью переменной Skp.

Итак, рассмотрим несколько примеров общепрограммных свойств.

1. Свойство $\mathbf{G}(\neg PLCWin \vee \neg ManWin)$ означает, что не существует ситуаций, при которых выигравшими считались бы сразу оба игрока.

2. Значение переменной Sum всегда остается в пределах 37: $\mathbf{G}(Sum \leqslant 37)$.

3. Формула $\mathbf{G}(Mv1 + Mv2 + Mv3 + Mv4 + Mv5 + Mv6 \leqslant 1)$ запрещает совершать более одного игрового хода за один проход рабочего цикла ПЛК.

4. Нажатие $PBStart$ должно приводить к сбросу игры в начальное состояние: $\mathbf{G}(PBStart \Rightarrow V1 = 4 \wedge V2 = 4 \wedge V3 = 4 \wedge V4 = 4 \wedge V5 = 4 \wedge V6 = 4 \wedge \neg Mv \wedge \neg Turn \wedge \wedge \neg(PLCWin \vee ManWin) \wedge Sum = 0)$.

5. Если ограничиться рассмотрением только такого поведения программы, при котором игровой ход совершается бесконечно часто, то из любого своего достижимого состояния программа рано или поздно попадает в состояние выигрыша ПЛК, состояние выигрыша Игрока или же в состояние сброса в начальное состояние:

$\mathbf{G}(\mathbf{F}(Mv)) \Rightarrow \mathbf{G}(\mathbf{F}(PLCWin \vee ManWin \vee PBStart))$.

6. Если рассмотреть только такое поведение программы, при котором время от времени совершаются игровые ходы, а кнопка сброса в начальное состояние *PBStart* нажимается только тогда, когда игра закончилась выигрышем одного из игроков, то каждая вновь начатая игра всегда будет приводить к победе одного из игроков (учитывая справедливость предыдущих свойств):

$\mathbf{G}(\mathbf{F}(Mv)) \wedge \mathbf{G}(\neg PBStart \wedge \mathbf{X}(PBStart) \Rightarrow (PLCWin \vee ManWin)) \Rightarrow$

$\mathbf{G}(PBStart \wedge \mathbf{X}(\neg PBStart) \Rightarrow \mathbf{X}(\neg PBStart\, \mathbf{U}\, (PLCWin \vee ManWin))).$

7. Следующие три свойства соответствуют выигрышным стратегиям ПЛК, если Игрок начинает игру с переворачивания карты «3», «4» или «6» соответственно:

$\mathbf{G}(\mathbf{F}(Mv)) \wedge \mathbf{G}(\neg PBStart \wedge \mathbf{X}(PBStart) \Rightarrow (PLCWin \vee ManWin)) \Rightarrow$
$\mathbf{G}(Mv3 \wedge Sum = 3 \Rightarrow ((\neg ManWin \wedge \neg PLCWin)\, \mathbf{U}\, PLCWin));$

$\mathbf{G}(\mathbf{F}(Mv)) \wedge \mathbf{G}(\neg PBStart \wedge \mathbf{X}(PBStart) \Rightarrow (PLCWin \vee ManWin)) \Rightarrow$
$\mathbf{G}(Mv4 \wedge Sum = 4 \Rightarrow ((\neg ManWin \wedge \neg PLCWin)\, \mathbf{U}\, PLCWin));$

$\mathbf{G}(\mathbf{F}(Mv)) \wedge \mathbf{G}(\neg PBStart \wedge \mathbf{X}(PBStart) \Rightarrow (PLCWin \vee ManWin)) \Rightarrow$
$\mathbf{G}(Mv6 \wedge Sum = 6 \Rightarrow ((\neg ManWin \wedge \neg PLCWin)\, \mathbf{U}\, PLCWin)).$

8. Формула $\mathbf{G}(\mathbf{X}(Mv1 \wedge Sum = 1 \vee Mv2 \wedge Sum = 2 \vee Mv3 \wedge Sum = 3 \vee Mv4 \wedge Sum = 4 \vee Mv5 \wedge Sum = 5 \vee Mv6 \wedge Sum = 6) \Rightarrow \neg Turn)$ выражает свойство, что первый ход всегда принадлежит Игроку, а не ПЛК.

В следующем разделе приведена полная LTL-спецификация поведения этих переменных, за исключением переменных-входов, так как спецификация поведения входов не производится. Далее идут разделы, в которых по спецификации строятся ST-программа ПЛК и SMV-модель этой программы.

5.2. Спецификация ПЛК-программы логической игры «31»

```
Rst: GX(Rst = PBStart);
Skp: GX(Skp = (PB1+PB2+PB3+PB4+PB5+PB6>1));
Mv1: GX(Mv1 = ~Rst & ~(_PLCWin | _ManWin) & (_V1>=1 & _V1<=4) &
    (~_Turn & ~_PB1 & PB1 & ~Skp | _Turn & _Tmr.Q & ((31 - _Sum) MOD 7 = 1 | _Lck)));
V1+: GX(V1>_V1 -> _V1>=0 & _V1<=3 & Rst & V1=4);
V1-: GX(V1<_V1 -> _V1>=1 & _V1<=4 & Mv1 & V1=_V1-1); Init(V1)=4;
Mv2: GX(Mv2 = ~Rst & ~(_PLCWin | _ManWin) & (_V2>=1 & _V2<=4) &
              (~_Turn & ~_PB2 & PB2 & ~Skp |
                _Turn & _Tmr.Q & ((31 - _Sum) MOD 7 = 2 | _Lck & ~Mv1)));
V2+: GX(V2>_V2 -> _V2>=0 & _V2<=3 & Rst & V2=4);
V2-: GX(V2<_V2 -> _V2>=1 & _V2<=4 & Mv2 & V2=_V2-1); Init(V2)=4;
Mv3: GX(Mv3 = ~Rst & ~(_PLCWin | _ManWin) & (_V3>=1 & _V3<=4) &
              (~_Turn & ~_PB3 & PB3 & ~Skp |
                _Turn & _Tmr.Q & ((31 - _Sum) MOD 7 = 3 | _Lck & ~(Mv1 | Mv2))));
V3+: GX(V3>_V3 -> _V3>=0 & _V3<=3 & Rst & V3=4);
V3-: GX(V3<_V3 -> _V3>=1 & _V3<=4 & Mv3 & V3=_V3-1); Init(V3)=4;
Mv4: GX(Mv4 = ~Rst & ~(_PLCWin | _ManWin) & (_V4>=1 & _V4<=4) &
             (~_Turn & ~_PB4 & PB4 & ~Skp | _Turn & _Tmr.Q & ((31 - _Sum) MOD 7 = 4 |
              (31 - _Sum) MOD 7 = 0 |  _Lck & ~(Mv1 | Mv2 | Mv3))));
V4+: GX(V4>_V4 -> _V4>=0 & _V4<=3 & Rst & V4=4));
V4-: GX(V4<_V4 -> _V4>=1 & _V4<=4 & Mv4 & V4=_V4-1)); Init(V4)=4;
Mv5: GX(Mv5 = ~Rst & ~(_PLCWin | _ManWin) & (_V5>=1 & _V5<=4) &
             (~_Turn & ~_PB5 & PB5 & ~Skp | _Turn & _Tmr.Q & ((31 - _Sum) MOD 7 = 5 |
              _Lck & ~(Mv1 | Mv2 | Mv3 | Mv4))));
V5+: GX(V5>_V5 -> _V5>=0 & _V5<=3 & Rst & V5=4);
```

```
V5-: GX(V5<_V5 -> _V5>=1 & _V5<=4 & Mv5 & V5=_V5-1); Init(V5)=4;
Mv6: GX(Mv6 = ~Rst & ~(_PLCWin | _ManWin) & (_V6>=1 & _V6<=4) &
             (~_Turn & ~_PB6 & PB6 & ~Skp | _Turn & _Tmr.Q & ((31 - _Sum) MOD 7 = 6 |
              _Lck & ~(Mv1 | Mv2 | Mv3 | Mv4 | Mv5))));
V6+: GX(V6>_V6 -> _V6>=0 & _V6<=3 & Rst & V6=4);
V6-: GX(V6<_V6 -> _V6>=1 & _V6<=4 & Mv6 & V6=_V6-1); Init(V6)=4;
Sum: GX(Sum = 84-V1-2*V2-3*V3-4*V4-5*V5-6*V6);
Lck: GX(Lck = (V1=0 & (31-Sum) mod 7 = 1) | (V2=0 & (31-Sum) mod 7 = 2) |
              (V3=0 & (31-Sum) mod 7 = 3) | (V4=0 & (31-Sum) mod 7 = 4) |
              (V5=0 & (31-Sum) mod 7 = 5) | (V6=0 & (31-Sum) mod 7 = 6) |
              (V4=0 & (31-Sum) mod 7 = 0));
Mv : GX(Mv = (Mv1 | Mv2 | Mv3 | Mv4 | Mv5 | Mv6));
Turn+: GX(~_Turn &  Turn -> Mv);
Turn-: GX( _Turn & ~Turn -> (Mv | Rst));
ManWin+: GX(~_ManWin &  ManWin -> _Turn & Sum>31 & Mv);
ManWin-: GX( _ManWin & ~ManWin -> Rst);
PLCWin+: GX(~_PLCWin &  PLCWin -> ~_Turn & Sum>31 & Mv);
PLCWin-: GX( _PLCWin & ~PLCWin -> Rst);
Out1+: GX(~_Out1 &  Out1 -> (Mv1 | Rst)); Init(Out1)=1;
Out1-: GX( _Out1 & ~Out1 -> (_Rst & ~Rst & ~Mv | Mv & ~Mv1));
Out2+: GX(~_Out2 &  Out2 -> (Mv2 | Rst)); Init(Out2)=1;
Out2-: GX( _Out2 & ~Out2 -> (_Rst & ~Rst & ~Mv | Mv & ~Mv2));
Out3+: GX(~_Out3 &  Out3 -> (Mv3 | Rst)); Init(Out3)=1;
Out3-: GX( _Out3 & ~Out3 -> (_Rst & ~Rst & ~Mv | Mv & ~Mv3));
Out4+: GX(~_Out4 &  Out4 -> (Mv4 | Rst)); Init(Out4)=1;
Out4-: GX( _Out4 & ~Out4 -> (_Rst & ~Rst & ~Mv | Mv & ~Mv4));
Out5+: GX(~_Out5 &  Out5 -> (Mv5 | Rst)); Init(Out5)=1;
Out5-: GX( _Out5 & ~Out5 -> (_Rst & ~Rst & ~Mv | Mv & ~Mv5));
Out6+: GX(~_Out6 &  Out6 -> (Mv6 | Rst)); Init(Out6)=1;
Out6-: GX( _Out6 & ~Out6 -> (_Rst & ~Rst & ~Mv | Mv & ~Mv6));
Tmr.In+: GX(~_Tmr.In &  Tmr.In -> ~_Turn &  Turn);
Tmr.In-: GX( _Tmr.In & ~Tmr.In ->  _Turn & ~Turn); Init(Tmr.PT)=2s;
```

5.3. ST-программа логической игры «31»

```
VAR_GLOBAL
  PB1, PB2, PB3, PB4, PB5, PB6, PBStart: BOOL;  (* Входы *)
  Out1, Out2, Out3, Out4, Out5, Out6: BOOL:=1;  (* Выходы *)
  PLCWin, ManWin, Turn: BOOL;
END_VAR
PROGRAM PLC_PRG
VAR
  Tmr: TON:= (PT:= T#2s);
  V1, V2, V3, V4, V5, V6, _V1, _V2, _V3, _V4, _V5, _V6: BYTE:=4;
  Mv, Mv1, Mv2, Mv3, Mv4, Mv5, Mv6: BOOL;    Sum, _Sum: BYTE;
  Skp, Lck, _Lck, Rst, _Rst, _Turn, _PLCWin, _ManWin, _TmrQ, _TmrIn: BOOL;
  _Out1, _Out2, _Out3, _Out4, _Out5, _Out6: BOOL:=1;
  _PB1, _PB2, _PB3, _PB4, _PB5, _PB6: BOOL;
END_VAR
Rst := PBStart;                                                  (* Rst *)
Skp := BOOL_TO_BYTE(PB1)+BOOL_TO_BYTE(PB2)+BOOL_TO_BYTE(PB3)+
       BOOL_TO_BYTE(PB4)+BOOL_TO_BYTE(PB5)+BOOL_TO_BYTE(PB6)>1;   (* Skp *)
Mv1 := NOT Rst AND NOT(_PLCWin OR _ManWin) AND (_V1>=1 AND _V1<=4) AND
       (NOT _Turn AND NOT _PB1 AND PB1 AND NOT Skp OR
            _Turn AND _TmrQ AND ((31 - _Sum) MOD 7 = 1 OR _Lck)); (* Mv1 *)
```

```
IF    (_V1>=0 AND _V1<=3) AND Rst THEN V1:=4;                    (* V1+ *)
ELSIF (_V1>=1 AND _V1<=4) AND Mv1 THEN V1:=_V1-1; END_IF;         (* V1- *)
Mv2 := NOT Rst AND NOT(_PLCWin OR _ManWin) AND (_V2>=1 AND _V2<=4) AND
       (NOT _Turn AND NOT _PB2 AND PB2 AND NOT Skp OR
            _Turn AND _TmrQ AND ((31 - _Sum) MOD 7 = 2 OR
                                 _Lck AND NOT Mv1));             (* Mv2 *)
IF    (_V2>=0 AND _V2<=3) AND Rst THEN V2:=4;                    (* V2+ *)
ELSIF (_V2>=1 AND _V2<=4) AND Mv2 THEN V2:=_V2-1; END_IF;         (* V2- *)
Mv3 := NOT Rst AND NOT(_PLCWin OR _ManWin) AND (_V3>=1 AND _V3<=4) AND
       (NOT _Turn AND NOT _PB3 AND PB3 AND NOT Skp OR
            _Turn AND _TmrQ AND ((31 - _Sum) MOD 7 = 3 OR
                                 _Lck AND NOT(Mv1 OR Mv2)));     (* Mv3 *)
IF    (_V3>=0 AND _V3<=3) AND Rst THEN V3:=4;                    (* V3+ *)
ELSIF (_V3>=1 AND _V3<=4) AND Mv3 THEN V3:=_V3-1; END_IF;         (* V3- *)
Mv4 := NOT Rst AND NOT(_PLCWin OR _ManWin) AND (_V4>=1 AND _V4<=4) AND
       (NOT _Turn AND NOT _PB4 AND PB4 AND NOT Skp OR
            _Turn AND _TmrQ AND ((31 - _Sum) MOD 7 = 4 OR
                                 (31 - _Sum) MOD 7 = 0 OR       (* Mv4 *)
                                 _Lck AND NOT(Mv1 OR Mv2 OR Mv3)));
IF    (_V4>=0 AND _V4<=3) AND Rst THEN V4:=4;                    (* V4+ *)
ELSIF (_V4>=1 AND _V4<=4) AND Mv4 THEN V4:=_V4-1; END_IF;         (* V4- *)
Mv5 := NOT Rst AND NOT(_PLCWin OR _ManWin) AND (_V5>=1 AND _V5<=4) AND
       (NOT _Turn AND NOT _PB5 AND PB5 AND NOT Skp OR
            _Turn AND _TmrQ AND ((31 - _Sum) MOD 7 = 5 OR       (* Mv5 *)
                                 _Lck AND NOT(Mv1 OR Mv2 OR Mv3 OR Mv4)));
IF    (_V5>=0 AND _V5<=3) AND Rst THEN V5:=4;                    (* V5+ *)
ELSIF (_V5>=1 AND _V5<=4) AND Mv5 THEN V5:=_V5-1; END_IF;         (* V5- *)
Mv6 := NOT Rst AND NOT(_PLCWin OR _ManWin) AND (_V6>=1 AND _V6<=4) AND
       (NOT _Turn AND NOT _PB6 AND PB6 AND NOT Skp OR
            _Turn AND _TmrQ AND ((31 - _Sum) MOD 7 = 6 OR       (* Mv6 *)
                        _Lck AND NOT(Mv1 OR Mv2 OR Mv3 OR Mv4 OR Mv5)));
IF    (_V6>=0 AND _V6<=3) AND Rst THEN V6:=4;                    (* V6+ *)
ELSIF (_V6>=1 AND _V6<=4) AND Mv6 THEN V6:=_V6-1; END_IF;         (* V6- *)
Sum := 84-V1-2*V2-3*V3-4*V4-5*V5-6*V6; (* Sum *)
Lck := (V1=0 AND (31-Sum) MOD 7 = 1) OR (V2=0 AND (31-Sum) MOD 7 = 2) OR
       (V3=0 AND (31-Sum) MOD 7 = 3) OR (V4=0 AND (31-Sum) MOD 7 = 4) OR
       (V5=0 AND (31-Sum) MOD 7 = 5) OR (V6=0 AND (31-Sum) MOD 7 = 6) OR
       (V4=0 AND (31-Sum) MOD 7 = 0);                           (* Lck *)
Mv := Mv1 OR Mv2 OR Mv3 OR Mv4 OR Mv5 OR Mv6;                    (* Mv  *)
IF NOT _Turn AND  Mv         THEN Turn:=TRUE;                   (* Turn+ *)
ELSIF  _Turn AND (Mv OR Rst) THEN Turn:=FALSE; END_IF;          (* Turn- *)
IF NOT _ManWin AND     _Turn AND Sum>31 AND Mv THEN ManWin:=TRUE;
ELSIF  _ManWin AND Rst                         THEN ManWin:=FALSE;
END_IF; (* ManWin+, ManWin- *)
IF NOT _PLCWin AND NOT _Turn AND Sum>31 AND Mv THEN PLCWin:=TRUE;
ELSIF  _PLCWin AND Rst                         THEN PLCWin:=FALSE;
END_IF; (* PLCWin+, PLCWin- *)
IF NOT _Out1 AND (Mv1 OR Rst)                THEN Out1:=TRUE;   (* Out1+ *)
ELSIF  _Out1 AND (_Rst AND NOT Rst AND NOT Mv OR                (* Out1- *)
                  Mv AND NOT Mv1)            THEN Out1:=FALSE; END_IF;
IF NOT _Out2 AND (Mv2 OR Rst)                THEN Out2:=TRUE;   (* Out2+ *)
ELSIF  _Out2 AND (_Rst AND NOT Rst AND NOT Mv OR                (* Out2- *)
                  Mv AND NOT Mv2)            THEN Out2:=FALSE; END_IF;
IF NOT _Out3 AND (Mv3 OR Rst)                THEN Out3:=TRUE;   (* Out3+ *)
ELSIF  _Out3 AND (_Rst AND NOT Rst AND NOT Mv OR                (* Out3- *)
                  Mv AND NOT Mv3)            THEN Out3:=FALSE; END_IF;
```

```
IF NOT _Out4 AND (Mv4 OR Rst)                THEN Out4:=TRUE;     (* Out4+ *)
ELSIF  _Out4 AND (_Rst AND NOT Rst AND NOT Mv OR                  (* Out4- *)
                  Mv AND NOT Mv4)            THEN Out4:=FALSE; END_IF;
IF NOT _Out5 AND (Mv5 OR Rst)                THEN Out5:=TRUE;     (* Out5+ *)
ELSIF  _Out5 AND (_Rst AND NOT Rst AND NOT Mv OR                  (* Out5- *)
                  Mv AND NOT Mv5)            THEN Out5:=FALSE; END_IF;
IF NOT _Out6 AND (Mv6 OR Rst)                THEN Out6:=TRUE;     (* Out6+ *)
ELSIF  _Out6 AND (_Rst AND NOT Rst AND NOT Mv OR                  (* Out6- *)
                  Mv AND NOT Mv6)            THEN Out6:=FALSE; END_IF;
(* ------- таймер ------- *)
IF NOT _TmrIn AND NOT _Turn AND     Turn THEN Tmr.In:=TRUE;     (* Tmr.In+ *)
ELSIF  _TmrIn AND     _Turn AND NOT Turn THEN Tmr.In:=FALSE;    (* Tmr.In- *)
END_IF;  Tmr(); (* вызов таймера *)
(* ------ псевдооператорный раздел ------- *)
_Lck:=Lck;_Rst:=Rst;_PLCWin:=PLCWin;_ManWin:=ManWin; TmrQ:=Tmr.Q; _TmrIn:=Tmr.In;
_Out1:=Out1;_Out2:=Out2;_Out3:=Out3;_Out4:=Out4;_Out5:=Out5;_Out6:=Out6;
_PB1:=PB1;_PB2:=PB2;_PB3:=PB3;_PB4:=PB4;_PB5:=PB5;_PB6:=PB6;
_V1:=V1;_V2:=V2;_V3:=V3;_V4:=V4;_V5:=V5;_V6:=V6;_Turn:=Turn;_Sum:=Sum;
```

5.4. SMV-модель программы логической игры «31»

```
#define Rst  PBStart
#define Skp  (PB1+PB2+PB3+PB4+PB5+PB6>1)
#define Sum  (84-V1-2*V2-3*V3-4*V4-5*V5-6*V6)
#define Lck  ((V1=0 & (31-Sum) mod 7 = 1) | (V2=0 & (31-Sum) mod 7 = 2) |
              (V3=0 & (31-Sum) mod 7 = 3) | (V4=0 & (31-Sum) mod 7 = 4) |
              (V5=0 & (31-Sum) mod 7 = 5) | (V6=0 & (31-Sum) mod 7 = 6) |
              (V4=0 & (31-Sum) MOD 7 = 0))
#define Mv   (Mv1 | Mv2 | Mv3 | Mv4 | Mv5 | Mv6)
module timer(){
  I : 0..1;   /* вход  */
  Q : 0..1;   /* выход */
  init(I):=0; init(Q):=0; /* инициализация */
  next(Q):= next(I) & (Q | {0, 1}); /* новое значение выхода */
  FAIRNESS I -> Q;  /* условие честного срабатывания таймера */
}
module main(){   /* Раздел описания переменных */
  PB1, PB2, PB3, PB4, PB5, PB6, PBStart: 0..1;               /* входы  */
  ManWin, PLCWin, Turn, Out1,Out2,Out3,Out4,Out5,Out6: 0..1; /* выходы */
  V1, V2, V3, V4, V5, V6 : 0..4; /* счетчики */ /* внутренние */
  Mv1, Mv2, Mv3, Mv4, Mv5, Mv6: 0..1;
  Tmr : timer;  /* таймер */
  /* Раздел инициализации */
 init(PBStart):=0; init(Turn):=0; init(ManWin):=0; init(PLCWin):=0;
 init(PB1):=0; init(PB2):=0; init(PB3):=0; init(PB4):=0; init(PB5):=0;
 init(PB6):=0; init(Out1):=1; init(Out2):=1; init(Out3):=1;init(Out4):=1;
 init(Out5):=1; init(Out6):=1; init(V1):=4; init(V2):=4; init(V3):=4;
 init(V4):=4; init(V5):=4; init(V6):=4; init(Mv1):=0; init(Mv2):=0;
 init(Mv3):=0; init(Mv4):=0; init(Mv5):=0; init(Mv6):=0;
 /* Система переходов */
 next(PB1):={0,1}; next(PB2):={0,1}; next(PB3):={0,1}; next(PB4):={0,1};
 next(PB5):={0,1}; next(PB6):={0,1}; next(PBStart):={0,1};
 next(Mv1) := ~next(Rst) & ~(PLCWin | ManWin) & (V1>=1 & V1<=4) &
              (~Turn & ~PB1 & next(PB1) & ~next(Skp) |
                Turn & Tmr.Q & ((31 - Sum) mod 7 = 1 | Lck));    /* Mv1 */
```

```
case{ V1>=0 & V1<=3 & next(Rst) : next(V1):=4;                  /* V1+ */
      V1>=1 & V1<=4 & next(Mv1) : next(V1):=V1-1;               /* V1- */
      default                   : next(V1):=V1;  };
next(Mv2) := ~next(Rst) & ~(PLCWin | ManWin) & (V2>=1 & V2<=4) &
             (~Turn & ~PB2 & next(PB2) & ~next(Skp) |
               Turn & Tmr.Q & ((31 - Sum) mod 7 = 2 |
                               Lck & ~next(Mv1)));              /* Mv2 */
case{ V2>=0 & V2<=3 & next(Rst) : next(V2):=4;                  /* V2+ */
      V2>=1 & V2<=4 & next(Mv2) : next(V2):=V2-1;               /* V2- */
      default                   : next(V2):=V2;  };
next(Mv3) := ~next(Rst) & ~(PLCWin | ManWin) & (V3>=1 & V3<=4) &
             (~Turn & ~PB3 & next(PB3) & ~next(Skp) |
               Turn & Tmr.Q & ((31 - Sum) mod 7 = 3 |
                               Lck & ~next(Mv1 | Mv2)));        /* Mv3 */
case{ V3>=0 & V3<=3 & next(Rst) : next(V3):=4;                  /* V3+ */
      V3>=1 & V3<=4 & next(Mv3) : next(V3):=V3-1;               /* V3- */
      default                   : next(V3):=V3;  };
next(Mv4) := ~next(Rst) & ~(PLCWin | ManWin) & (V4>=1 & V4<=4) &
            (~Turn & ~PB4 & next(PB4) & ~next(Skp) |
              Turn & Tmr.Q & ((31 - Sum) mod 7 = 4 |
                              (31 - Sum) mod 7 = 0 |
                              Lck & ~next(Mv1 | Mv2 | Mv3)));   /* Mv4 */
case{ V4>=0 & V4<=3 & next(Rst) : next(V4):=4;                  /* V4+ */
      V4>=1 & V4<=4 & next(Mv4) : next(V4):=V4-1;               /* V4- */
      default                   : next(V4):=V4;  };
next(Mv5) := ~next(Rst) & ~(PLCWin | ManWin) & (V5>=1 & V5<=4) &
            (~Turn & ~PB5 & next(PB5) & ~next(Skp) |
              Turn & Tmr.Q & ((31 - Sum) mod 7 = 5 |            /* Mv5 */
                              Lck & ~next(Mv1 | Mv2 | Mv3 | Mv4)));
case{ V5>=0 & V5<=3 & next(Rst) : next(V5):=4;                  /* V5+ */
      V5>=1 & V5<=4 & next(Mv5) : next(V5):=V5-1;               /* V5- */
      default                   : next(V5):=V5;  };
next(Mv6) := ~next(Rst) & ~(PLCWin | ManWin)&(V6>=1 & V6<=4) &  /* Mv6 */
             (~Turn & ~PB6 & next(PB6) & ~next(Skp) |
               Turn & Tmr.Q & ((31 - Sum) mod 7 = 6 |
                               Lck & ~next(Mv1 | Mv2 | Mv3 | Mv4 | Mv5)));
case{V6>=0 & V6<=3 & next(Rst) : next(V6):=4;                   /* V6+ */
     V6>=1 & V6<=4 & next(Mv6) : next(V6):=V6-1;                /* V6- */
     default                   : next(V6):=V6;};
case{~Turn & next(Mv)       : next(Turn):=1;                  /* Turn+ */
      Turn & next(Mv | Rst) : next(Turn):=0;                  /* Turn- */
     default                : next(Turn):=Turn;};
case{~ManWin &  Turn & next(Sum>31) & next(Mv): next(ManWin):=1;
      ManWin & next(Rst)                      : next(ManWin):=0;
     default                                  : next(ManWin):=ManWin;};
case{~PLCWin & ~Turn & next(Sum>31) & next(Mv): next(PLCWin):=1;
      PLCWin & next(Rst)                      : next(PLCWin):=0;
     default                                  : next(PLCWin):=PLCWin;};
case{~Out1 & next(Mv1 | Rst)                  : next(Out1):=1; /* Out1+*/
      Out1 & (Rst & next(~Rst) & next(~Mv) |
              next(Mv & ~Mv1))                : next(Out1):=0; /* Out1- */
      default                                 : next(Out1):=Out1;};
case{~Out2 & next(Mv2 | Rst)                  : next(Out2):=1; /* Out2+ */
      Out2 & (Rst & next(~Rst) & next(~Mv) |
              next(Mv & ~Mv2))                : next(Out2):=0; /* Out2- */
     default                                  : next(Out2):=Out2;};
```

```
 case{~Out3 & next(Mv3 | Rst)                 : next(Out3):=1; /* Out3+ */
       Out3 & (Rst & next(~Rst) & next(~Mv) |
               next(Mv & ~Mv3))               : next(Out3):=0; /* Out3- */
      default                                 : next(Out3):=Out3;};
 case{~Out4 & next(Mv4 | Rst)                 : next(Out4):=1; /* Out4+ */
       Out4 & (Rst & next(~Rst) & next(~Mv) |
               next(Mv & ~Mv4))               : next(Out4):=0; /* Out4- */
      default                                 : next(Out4):=Out4;};
 case{~Out5 & next(Mv5 | Rst)                 : next(Out5):=1; /* Out5+ */
       Out5 & (Rst & next(~Rst) & next(~Mv) |
               next(Mv & ~Mv5))               : next(Out5):=0; /* Out5- */
      default                                 : next(Out5):=Out5;};
 case{~Out6 & next(Mv6 | Rst)                 : next(Out6):=1; /* Out6+ */
       Out6 & (Rst & next(~Rst) & next(~Mv) |
               next(Mv & ~Mv6))               : next(Out6):=0; /* Out6- */
      default                                 : next(Out6):=Out6;};
 case{~Tmr.I & ~Turn &  next(Turn) : next(Tmr.I):=1;         /* Tmr.In+ */
       Tmr.I &  Turn & ~next(Turn) : next(Tmr.I):=0;         /* Tmr.In- */
       default                     : next(Tmr.I):=Tmr.I;};
 Prp_notWinBoth:  assert G(~PLCWin | ~ManWin);     /* Раздел свойств */
 Prp_Sum:     assert G(Sum <= 37);
 Prp_Mv:      assert G(Mv1+Mv2+Mv3+Mv4+Mv5+Mv6 <= 1);
 Prp_PBStart: assert G(PBStart -> V1=4 & V2=4 & V3=4 & V4=4 & V5=4 & V6=4 &
                                  ~Mv & ~Turn & ~(PLCWin | ManWin) & Sum=0);
 Prp_Win_PBStart: assert G(F(Mv)) -> G(F(PLCWin | ManWin | PBStart));
 Prp_Win: assert G(F(Mv)) & G(~PBStart & X(PBStart) -> (PLCWin | ManWin))
          -> G(PBStart & X(~PBStart) -> X((~PBStart)U(PLCWin | ManWin)));
 Prp_PLCWin3: assert
              G(F(Mv)) & G(~PBStart & X(PBStart) -> (PLCWin | ManWin))
              -> G(Mv3 & Sum=3 -> ((~ManWin & ~PLCWin)U(PLCWin)));
 Prp_PLCWin4: assert
              G(F(Mv)) & G(~PBStart & X(PBStart) -> (PLCWin | ManWin))
              -> G(Mv4 & Sum=4 -> ((~ManWin & ~PLCWin)U(PLCWin)));
 Prp_PLCWin6: assert
              G(F(Mv)) & G(~PBStart & X(PBStart) -> (PLCWin | ManWin))
              -> G(Mv6 & Sum=6 -> ((~ManWin & ~PLCWin)U(PLCWin)));
 Prp_Turn: assert G(X(Mv1 & Sum=1 | Mv2 & Sum=2 | Mv3 & Sum=3 |
                      Mv4 & Sum=4 | Mv5 & Sum=5 | Mv6 & Sum=6) -> ~Turn);
}
```

Заключение

Предложенный подход был успешно опробован на ряде «дискретных» задач логического управления разного типа со средним количеством (бинарных) входов и выходов ПЛК около 30 и общим количеством (бинарных) программных переменных до 59. Для таких задач, как управление установкой для приготовления смесей, управление системой гидравлических насосов, управление библиотечным подъемником и реализация на ПЛК логической игры, были проверены программные свойства, касающиеся соблюдения технологического процесса подготовки смеси (с целью исключения возможности выхода некондиционного продукта), бесперебойной работы гидравлической системы (своевременности подключения резервных насосов), обязательного выполнения поступивших команд вызова кабины подъемника на этажи

и правильности реализации стратегии логической игры соответственно. Работа проводилась на персональном компьютере с процессором Intel Core i7-2600K 3.40 GHz. Время, затраченное верификатором Cadence SMV на проверку выполнимости указанных свойств, ограничивается несколькими секундами.

По результатам дальнейших работ предполагается создание программного комплекса спецификации, построения, моделирования и верификации программ ПЛК.

Список литературы

1. *Кузьмин Е. В.*, *Соколов В. А.* Моделирование, спецификация и построение программ логических контроллеров // Моделирование и анализ информационных систем. 2013. Т. 20, №2. С. 104–120. (*Kuzmin E. V.*, *Sokolov V. A.* Modeling, Specification and Construction of PLC-programs // Modeling and analysis of information systems. 2013. V. 20, №2. P. 104–120 [in Russian]).

2. *Кузьмин Е. В.*, *Соколов В. А.* О построении и верификации программ логических контроллеров // Моделирование и анализ информационных систем. 2012. Т. 19, №4. С. 25–36. (*Kuzmin E. V.*, *Sokolov V. A.* On Construction and Verification of PLC-Programs // Modeling and analysis of information systems. 2012. V. 19, №4. P. 25–36 [in Russian]).

3. *Кузьмин Е. В.*, *Соколов В. А.* О верификации LD-программ логических контроллеров // Моделирование и анализ информационных систем. 2012. Т. 19, №2. С. 138–144. (*Kuzmin E. V.*, *Sokolov V. A.* On Verification of PLC-Programs Written in the LD-Language // Modeling and analysis of information systems. 2012. V. 19, №2. P. 138–144 [in Russian]).

4. *Петров И. В.* Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования. М.: СОЛОН-Пресс, 2004. 256 с. (*Petrov I. V.* Programmiruemye kontrollery. Standartnye jazyki i priemy prikladnogo proektirovanija. M.: SOLON-Press, 2004. 256 p. [in Russian]).

5. *Canet G.*, *Couffin S.*, *Lesage J.-J.*, *Petit A.*, *Schnoebelen Ph.* Towards the Automatic Verification of PLC Programs Written in Instruction List // Proc. of the IEEE International Conference on Systems, Man and Cybernetics. Argos Press, 2000. P. 2449–2454.

6. *Clark E. M.*, *Grumberg O.*, *Peled D. A.* Model Checking. The MIT Press, 2001.

7. CoDeSys. Controller Development System. http://www.3s-software.com/

8. *Gries D.* The Science of Programming. Springer-Verlag, 1981.

9. *Parr E. A.* Programmable Controllers. An engineer's guide. Newnes, 2003. 442 p.

10. *Pavlovic O.*, *Pinger R.*, *Kollmann M.* Automation of Formal Verification of PLC Programs Written in IL // Proceedings of 4th International Verification Workshop (VERIFY'07). Bremen, Germany, 2007. P. 152–163.

11. *Rossi O.*, *Schnoebelen Ph.* Formal Modeling of Timed Function Blocks for the Automatic Verification of Ladder Diagram Programs // Proc. of the 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems, Shaker Verlag, 2000. P. 177-182.

12. SMV. The Cadence SMV Model Checker. http://www.kenmcmil.com/smv.html

Construction and Verification of PLC-programs by LTL-specification

Kuzmin E. V., Sokolov V. A., Ryabukhin D. A.

P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia

Keywords: programmable pogic controllers, software engineering, specification and verification of PLC-programs

An approach to construction and verification of PLC-programs for discrete tasks is proposed. For the specification of a program behavior we use the linear-time temporal logic LTL. Programming is carried out in the ST-language according to an LTL-specification. The correctness analysis of an LTL-specification is carried out by the symbolic model checking tool Cadence SMV. A new approach to programming and verification of PLC-programs is shown by an example. For a discrete problem we give a ST-program, its LTL-specification and an SMV-model.

A purpose of the article is to describe an approach to programming PLC, which would provide a possibility of PLC-program correctness analysis by the model checking method.

Under the proposed approach the change of the value of each program variable is described by a pair of LTL-formulas. The first LTL-formula describes situations that increase the value of the corresponding variable, the second LTL-formula specifies conditions leading to a decrease of the variable value. The LTL-formulas (used for specification of the corresponding variable behavior) are constructive in the sense that they construct the PLC-program, which satisfies temporal properties expressed by these formulas. Thus, the programming of PLC is reduced to the construction of LTL-specification of the behavior of each program variable. In addition, an SMV-model of a PLC-program is constructed according to LTL-specification. Then, the SMV-model is analysed by the symbolic model checking tool Cadence SMV.

Сведения об авторах:
Кузьмин Егор Владимирович,
Ярославский государственный университет им. П.Г. Демидова,
д-р физ.-мат. наук, профессор;
Соколов Валерий Анатольевич,
Ярославский государственный университет им. П.Г. Демидова,
д-р физ.-мат. наук, профессор;
Рябухин Дмитрий Александрович,
Ярославский государственный университет им. П.Г. Демидова,
аспирант.