Модел. и анализ информ. систем. Т. **20**, № **2** (2013) 104–120

УДК 519.7

Моделирование, спецификация и построение программ логических контроллеров

Кузьмин Е. В.[1], Соколов В. А.[2]

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

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

получена 10 января 2013

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

Предлагается новый подход к построению надежных «дискретных» ПЛК-программ с таймерами — программирование исходя из задач спецификации и верификации. Для спецификации программного поведения используется язык темпоральной логики LTL. Программирование осуществляется на языке ST по LTL-спецификации. Проводится дискретное моделирование таймера. Новый подход к программированию ПЛК демонстрируется на примере.

Предлагаемый подход к программированию ПЛК обеспечивает возможность анализа корректности ПЛК-программ с помощью метода проверки модели. При программировании требуется соблюдение следующих двух условий: 1) значение каждой переменной должно изменяться не более одного раза за одно полное выполнение программы при прохождении рабочего цикла ПЛК; 2) значение каждой переменной должно изменяться только в одном месте программы ПЛК.

В рамках предлагаемого подхода 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.

[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 (не гарантирующее полное отсутствие ошибок) посредством визуализации объектов управления ПЛК. Вместе с тем в настоящее время накоплены определенные теоретические знания и опыт использования существующих разработок в области формальных методов моделирования и анализа программных систем. Программирование логических контроллеров представляет собой прикладную область, в которой существующие наработки могли бы иметь успешное применение. Под успешным применением понимается внедрение формальных методов в процесс создания программ на уровне отлаженной технологии, понятной всем специалистам, задействованным в этом процессе — инженерам, программистам и тестировщикам. Обычно имея небольшой размер и конечное пространство состояний, программы ПЛК представляют собой исключительно удобный объект для формального (и в том числе автоматического) анализа корректности.

Программируемый логический контроллер (ПЛК) — классическая «реагирующая» система, представляющая собой программно управляемый дискретный автомат, имеющий некоторое множество входов, подключенных посредством датчиков к объекту управления, и множество выходов, подключенных к исполнительным устройствам [5, 6]. ПЛК контролирует состояния входов и вырабатывает определенные последовательности программно заданных действий, отражающихся в изменении выходов. ПЛК предназначен для работы в режиме реального времени в условиях промышленной среды.

ПЛК имеют широкое распространение. Они задействованы начиная с бытовых приборов и заканчивая сложными системами (процессами) управления промышленными объектами. Любая машина, способная автоматически выполнять некоторые операции, имеет в своем составе управляющий контроллер — модуль, обеспечивающий логику работы устройства. Задачей прикладного программирования ПЛК является реализация алгоритма управления конкретной машиной. К управляющим программам ПЛК, реализующим логику процессов, предъявляются строгие требования корректной работы, поскольку программная ошибка может привести к серьезным негативным последствиям (причинение вреда здоровью персонала, экономический ущерб и т. д.).

Программа ПЛК выполняется в рабочем цикле (скорость прохождения которого зависит от мощности микрокомпьютерного ядра). За каждый проход рабочего цикла происходит считывание входов, выполнение программы и выставление выходов. Предполагается, что программы ПЛК должны быть доступны для понимания широкому кругу специалистов.

Языки программирования ПЛК определяются стандартом МЭК 61131-3. Этот стандарт включает в себя описание пяти языков: SFC, IL, ST, LD и FBD. Язык IL (Instruction list) — ассемблер с аккумулятором и переходами по меткам. Язык ST

(Structured Text) — высокоуровневый язык, синтаксически представляющий собой несколько адаптированный язык Паскаль. Язык релейных диаграмм LD (Ladder Diagram) — графический язык, реализующий структуры электрических цепей. FBD (Function Block Diagram) — графический язык диаграмм принципиальных схем электронных устройств на микросхемах. SFC (Sequential Function Chart) — последовательные функциональные схемы. Диаграммы SFC являются высокоуровневым графическим инструментом, они состоят из шагов и переходов между ними, которые разделяют задачи на простые этапы с формально определенной логикой работы системы. Разрешение перехода определяется условием. С шагом связаны определенные действия, которые описываются на любом из языков МЭК 61131-3.

Указанные языки представляют собой простой, но достаточно мощный инструмент для реализации задач ПЛК. «Простота» языков обеспечивает возможность применения всех существующих методов анализа корректности программ — тестирования, дедуктивного анализа (theorem proving) [1] и автоматического метода проверки модели (model checking) [2] — для верификации программ ПЛК. Дедуктивный анализ в большей степени применим к «непрерывным» задачам обеспечения устойчивости и качества регулирования инженерной теории управления, реализация которых на ПЛК сопряжена с программированием 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. Наиболее удобными для программирования, спецификации и верификации программ ПЛК являются языки ST, LD и SFC, поскольку они не вызывают трудностей ни у разработчиков, ни у инженеров, и легко могут быть транслированы в языки программных средств автоматической верификации.

Ранее в статье [3] проводился обзор методов и подходов к программированию «дискретных» задач ПЛК на примере задачи построения программы управления кодовым замком на языках LD, SFC и ST. Для этих подходов оценивалось удобство анализа программной корректности методом проверки модели относительно средства автоматической верификации Cadence SMV [8]. Были выявлены возможные уязвимости ПЛК-программ и трудности анализа программной корректности, возникающие при традиционных подходах к программированию.

В этой статье предлагается новый подход к построению надежных «дискретных» ПЛК-программ с таймером (как неотъемлемым элементом большинства программ логических контроллеров) — программирование исходя из задач спецификации и верификации. Предлагаемый подход к программированию ПЛК обеспечивает возможность анализа корректности ПЛК-программ с помощью метода проверки модели. В качестве языка спецификации программного поведения предлагается использовать язык темпоральной логики линейного времени LTL. Программирование осуществляется на языке ST по LTL-спецификации. Проводится дискретное моделирование таймера. Новый подход к программированию ПЛК демонстрируется на примере.

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

1. Метод проверки модели

Задача проверки модели (Model Checking) состоит в определении выполнимости для конечной модели программы, которая задается системой переходов (структурой 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 $\pi = s_0 s_1 s_2 \dots$ такая, что для всех $i \geq 0$ выполняется $s_i \to s_{i+1}$. Для пути $\pi = s_0 s_1 s_2 \dots s_i s_{i+1} s_{i+2} \dots$ имеем $\pi^i = s_i s_{i+1} s_{i+2} \dots$ и $\pi(i) = s_i$.

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 $p_i \in P$ $(0 \leqslant i \leqslant n)$:

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

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 $\Rightarrow$: $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \Rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$.

Формально отношение выполнимости $\models$ формулы φ логики LTL для некоторого пути π структуры Крипке $\mathcal{S}$ над P индуктивно определяется следующим образом:

$\pi \models \mathit{true}$; $\quad \pi \not\models \mathit{false}$;

$\pi \models p$ для $p \in P \iff p \in L(\pi(0))$;

$\pi \models \neg\,\varphi \iff \pi \not\models \varphi$;

$\pi \models \varphi \wedge \psi \iff \pi \models \varphi$ и $\pi \models \psi$;

$\pi \models X\,\varphi \iff \pi^1 \models \varphi$;

$\pi \models \psi\, U \varphi \iff \exists j \geqslant 0$, что $\pi^j \models \varphi$ и при этом для всех i, $0 \leqslant i < j$, $\pi^i \models \psi$.

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

2. Модель Крипке программы ПЛК

Модель Крипке для программы ПЛК может быть построена вполне естественным образом. Исполнение любой ПЛК-программы осуществляется последовательно слева направо и сверху вниз. Значения выходов и внутренних переменных на текущем рабочем цикле ПЛК формируются на основе 1) значений переменных, полученных после предыдущего прохода рабочего цикла, 2) значений входов, 3) значений выходов и внутренних переменных, находящихся выше по программе. Поэтому в качестве состояния модели естественно рассмотреть вектор значений всех программных переменных после одного полного прохода рабочего цикла. Этот вектор значений переменных можно условно разделить на две части. Первая часть — вектор значений входов на момент начала текущего рабочего цикла ПЛК. Вторая часть представляет собой вектор значений выходов и значений внутренних переменных, полученных после прохождения полного рабочего цикла (на входах из первой части). Другими словами, состояние модели — это состояние программы ПЛК после одного полного прохода рабочего цикла. Начальное состояние модели — состояние программы после инициализации значений переменных. Отсюда естественным образом вытекает, что в качестве перехода из одного состояния модели в другое (или то же самое) может быть рассмотрено полное исполнение программы за один проход рабочего цикла ПЛК.

Отметим, что модель от исходной ПЛК-программы отличается лишь дискретным представлением работы таймеров (см. следующий раздел), абстрагированным от понятия реального времени. 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.

3. Дискретное моделирование таймера

Рассмотрим пример дискретного моделирования таймера типа TON. Таймер с задержкой включения TON(IN,PT,Q,ET) представляет собой функциональный блок, имеющий входы IN и PT типов BOOL и TIME соответственно и выходы Q и ET аналогично типов BOOL и TIME, где BOOL — это логический тип данных, а TIME — тип длительности интервалов времени в миллисекундах. При каждом вызове таймера пока IN равен 0, выходы Q = 0 и ET = 0. Как только происходит вызов таймера при IN = 1, начинается отсчет времени (в миллисекундах) на выходе ET до значения, равного PT. Далее счетчик ET не увеличивается. Выход Q равен 1, когда IN = 1 и ET = PT, иначе Q выставляется в 0. Таким образом, выход Q устанавливается с задержкой PT от фронта входа IN. Интерфейс таймера TON представлен на рис. 1 слева.

Не имея информации о скорости работы ПЛК (о времени прохода одного рабочего цикла ПЛК), на котором будет выполняться программа с таймером, естественным шагом является уход от конкретных временных значений и действительного понимания времени. Абстрагируясь от реального времени, можно утверждать, что таймер TON имеет три состояния: 1) таймер неактивен, т. е. IN = 0 и Q = 0,

или просто ¬IN/¬Q; 2) таймер запущен, но еще не сработал, т. е. IN = 1 и Q = 0, или просто IN/¬Q; 3) таймер сработал, т. е. IN = 1 и Q = 1, или просто IN/Q.

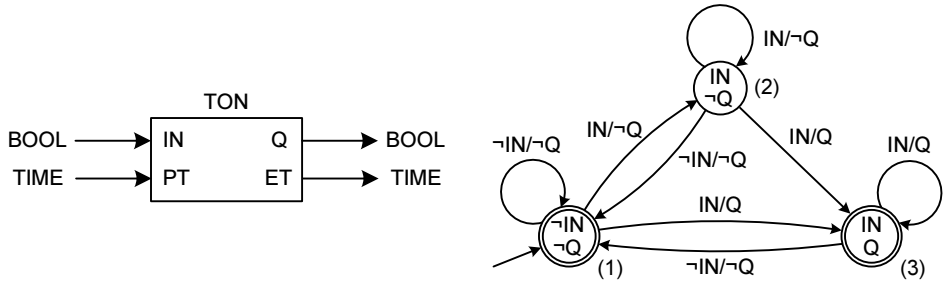

Рис. 1. Интерфейс и модель таймера TON

Предлагаемая модель таймера TON в виде конечного автомата с тремя состояниями представлена на рис. 1 справа. На этом рисунке дуги переходов по состояниям соответствуют вызовам функционального блока таймера. Например, если таймер находится в состоянии 2, то при следующем вызове функционального блока таймера со значением IN = 1 он может либо остаться в состоянии 2 (таймер запущен, но ещё не сработал), либо перейти в состояние 3 (таймер сработал). Если функциональный блок таймера был вызван со значением IN = 0, то таймер перейдет в состояние 1 (таймер неактивен).

Следует отметить, что при бесконечно частом вызове функционального блока таймер не может оставаться в состоянии 2 бесконечно долго, так как в противном случае это означало бы, что активный таймер так никогда и не сработает, что не соответствует реальному поведению моделируемого таймера TON. В связи с этим необходимо наложить следующие дополнительные ограничения на поведение модели таймера.

В предлагаемой модели таймера бесконечную последовательность переходов по дугам конечного автомата из одного состояния в другое (или то же самое состояние) будем считать «честной», или «справедливой», если эта последовательность переходов бесконечно часто проходит через состояние 1 или состояние 3 (не обязательно зацикливаясь в них). Эти два «допускающих» состояния на рис. 1 выделены двойным кружком. Последовательность переходов, которая навсегда зацикливается по петле в состоянии 2, является «нечестной», или «несправедливой», и считается недопустимой. Недопустимые последовательности переходов исключаются из поведения модели таймера TON.

Например, на языке SMV программного средства верификации методом символьной проверки модели Cadence SMV модель таймера может быть реализована следующим образом.

```
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 */
}
```

Здесь next(I) представляет собой значение входа I при каждом новом вызове функционального блока таймера, а next(Q) — формируемое при этом вызове новое значение выхода Q. Старое значение выхода таймера, полученное на предыдущем вызове, хранится просто в переменной Q. Недетерминированный выбор {0, 1} моделирует возможные альтернативы — срабатывание или несрабатывание таймера при наличии сигнала next(I) и при условии, что на предыдущем вызове таймер еще не сработал. Допустимые «честные» последовательности переходов определяются с помощью ключевого слова FAIRNESS и последующей импликации I -> Q, которая должна выполняться бесконечно часто (но не обязательно на каждом вызове).

4. Концепция программирования

Целью данной статьи является описание такого подхода к программированию ПЛК, который бы учитывал крайнюю необходимость и обеспечивал возможность анализа корректности ПЛК-программ с помощью метода проверки модели. В связи с этим мы при описании подхода к созданию надежных программ для ПЛК будем исходить прежде всего из соображений удобства и простоты применения метода проверки модели. Поэтому, чтобы иметь прозрачное и наглядное представление о том, каким образом происходит изменение значения той или иной программной переменной за один переход из состояния в состояние модели ПЛК-программы, потребуем при программировании 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.

Эти условия для каждой переменной позволяют установить четкую зависимость нового значения переменной от предыдущих значений программных переменных, полученных при выполнении программы на предыдущем проходе рабочего цикла, и новых значений переменных, уже вычисленных при выполнении программы на текущем проходе рабочего цикла ПЛК. Условие изменения значения переменной только в одном месте программы облегчает отладку, дает возможность простой оценки степени готовности и объема текста программы.

Очевидно, что за один проход рабочего цикла значение любой переменной возрастает, убывает или остается без изменения по отношению к ее значению, полученному на предыдущем проходе рабочего цикла. При этом если вообще не обращаться к переменной с целью присваивания какого-либо значения, то переменная сохранит свое старое значение. Учитывая это, 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 определенные 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

$$(1)\qquad \mathbf{G}\,\mathbf{X}(\,V{>}\_V \Rightarrow (\mathit{OldValCond}_1 \wedge \mathit{FiringCond}_1 \wedge V = \mathit{NewValExpr}_1 \vee \cdots \vee \mathit{OldValCond}_n \wedge \mathit{FiringCond}_n \wedge V = \mathit{NewValExpr}_n)\,),$$

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 $\mathit{OldValCond}_i$, было выполнено условие соответствующего внешнего воздействия $\mathit{FiringCond}_i$, а новое значение переменной V является значением выражения $\mathit{NewValExpr}_i$ ($i \in \{1, \ldots, n\}$).

Символ лидирующего подчеркивания «_» в обозначении переменной $\_V$ удобно воспринимать как псевдооператор, позволяющий обратиться к значению переменной V, которое она имела в предыдущем состоянии. Однако при этом необходимо наложить обязательное условие, что этот псевдооператор может использоваться только под действием темпорального оператора $\mathbf{X}$. Смысл псевдооператора «_» можно продемонстрировать, например, с помощью того факта, что темпоральная формула $\mathbf{X}(\_V = \mathit{Const})$ является эквивалентной простой формуле $V = \mathit{Const}$, где Const — некоторая константа.

Условия $\mathit{FiringCond}_i$ и $\mathit{OldValCond}_i$ 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 $\mathit{FiringCond}_i$ описывает ситуации, при которых возникает необходимость изменения значения переменной V, если это, конечно, допускается условием $\mathit{OldValCond}_i$.

Выражение $\mathit{NewValExpr}_i$ строится с помощью переменных и констант, операторов сравнения, логических, арифметических операторов и псевдооператора «_».

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

$$(1')\qquad \mathbf{G}\,\mathbf{X}(\,V{<}\_V \Rightarrow (\mathit{OldValCond}'_1 \wedge \mathit{FiringCond}'_1 \wedge V = \mathit{NewValExpr}'_1 \vee \cdots \vee \mathit{OldValCond}'_n \wedge \mathit{FiringCond}'_n \wedge V = \mathit{NewValExpr}'_n)\,).$$

Разберем спецификацию поведения переменной *Ctr* из программы «Электромагнитный кодовый замок», которая будет рассмотрена в качестве примера далее. Переменная *Ctr* — это счетчик, принимающий целые значения от 0 до 3. При поступлении сигналов $PB = 1$ и $K1 = 1$, означающих корректное нажатие кнопки K1, значение счетчика должно установиться в 1. Когда поступают сигналы $PB = 1$ и $K1 = 0$ корректного нажатия некоторой кнопки отличной от K1, значение счетчика *Ctr* должно увеличиться на 1, если старое значение счетчика было больше 0 и меньше 3, и сброситься в 0, если старое значение счетчика равнялось 3. Если электромагнит отключен, т. е. выставлен сигнал $\_EM = 0$, значение счетчика сбрасывается в 0. (Заранее известно, что если $\_EM = 0$, то $PB = 0$.)

$$\mathbf{G}\,\mathbf{X}(\,Ctr > \_Ctr \Rightarrow (\_Ctr = 0 \quad \wedge PB \wedge \quad K1 \wedge Ctr = 1 \vee \_Ctr \geqslant 1 \wedge \_Ctr \leqslant 2 \wedge PB \wedge \neg K1 \wedge Ctr = \_Ctr + 1));$$

$$\mathbf{G}\,\mathbf{X}(\,Ctr < \_Ctr \Rightarrow (\_Ctr = 3 \quad \wedge PB \wedge \neg K1 \wedge Ctr = 0 \vee \_Ctr \geqslant 2 \wedge \_Ctr \leqslant 3 \wedge PB \wedge \quad K1 \wedge Ctr = 1 \vee \_Ctr \geqslant 1 \wedge \_Ctr \leqslant 3 \wedge \neg \_EM \quad \wedge Ctr = 0 \quad)).$$

Здесь $OldValCond_1 = (\_Ctr = 0)$, $FiringCond_1 = (PB \wedge K1)$, $NewValExpr_1 = 1$, $OldValCond_2 = (\_Ctr \geqslant 1 \wedge \_Ctr \leqslant 2)$, $FiringCond_2 = (PB \wedge \neg K1)$, $NewValExpr_2 = \_Ctr + 1$, и $OldValCond'_1 = (\_Ctr = 3)$, $FiringCond'_1 = (PB \wedge \neg K1)$, $NewValExpr'_1 = 0$, $OldValCond'_2 = (\_Ctr \geqslant 2 \wedge \_Ctr \leqslant 3)$, $FiringCond'_2 = (PB \wedge K1)$, $NewValExpr'_2 = 1$, $OldValCond'_3 = (\_Ctr \geqslant 1 \wedge \_Ctr \leqslant 3)$, $FiringCond'_3 = (\neg \_EM)$, $NewValExpr'_3 = 0$.

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:

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

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

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

Обращаясь к примеру «Электромагнитный кодовый замок», приведем спецификацию логической переменной EM. С помощью переменной EM происходит включение или отключение электромагнита, удерживающего дверь в закрытом положении. Электромагнит отключается, т. е. выставляется сигнал $EM = 0$, если правильно набрана открывающая кодовая последовательность $Z1 = 1$, $Z3 = 1$ и $Z5 = 1$ или нажата кнопка «Открыть» $Opn = 1$. Электромагнит снова включается $EM = 1$ по истечении некоторого времени при срабатывании таймера, вызов которого осуществлялся на предыдущем проходе рабочего цикла, т. е. при $\_Tmr.Q = 1$.

$$\mathbf{G}\,\mathbf{X}(\,\neg \_EM \wedge \quad EM \Rightarrow \_Tmr.Q\,);$$

$$\mathbf{G}\,\mathbf{X}(\quad \_EM \wedge \neg EM \Rightarrow (Z1 \wedge Z3 \wedge Z5 \vee Opn)\,).$$

Здесь $FiringCond = \_Tmr.Q$ и $FiringCond' = (Z1 \wedge Z3 \wedge Z5 \vee Opn)$.

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

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

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

Такая спецификация может быть заменена на всего лишь одну LTL-формулу вида

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

Переменную V, для которой строится спецификация 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 $NewValExpr$ не содержит псевдооператора лидирующего подчеркивания «_», переменную V будем называть *переменной-подстановкой*.

Примером переменной-функции является логическая переменная PB, единичное значение которой в программе «Электромагнитный кодовый замок» соответствует корректному $Skp = 0$ нажатию $Alm = 1 \wedge \_Alm = 0$ одной из кнопок кодовой панели при включенном электромагните $\_EM = 1$ и ненажатой кнопке «Открыть» $Opn = 0$.

$\mathbf{G}\,\mathbf{X}(PB = Alm \wedge \neg \_Alm \wedge \neg Skp \wedge \neg Opn \wedge \_EM\ )$.

Примеры переменных-подстановок — переменные «пропуск» Skp (отслеживает одновременное нажатие нескольких кодовых кнопок) и «звонок» Alm (соответствует звуковому сигналу при нажатии одной из кодовых кнопок):

$\mathbf{G}\,\mathbf{X}(Skp = (K1 + K2 + K3 + K4 + K5 + K6 + K7 + K8 + K9 > 1))$;

$\mathbf{G}\,\mathbf{X}(Alm = (K1 \vee K2 \vee K3 \vee K4 \vee K5 \vee K6 \vee K7 \vee K8 \vee K9))$.

Важно отметить, что каждый из рассмотренных шаблонов LTL-формул спецификации поведения переменной является конструктивным, т. е. по LTL-формулам 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. При этом сам процесс (стадия) написания кода программы заканчивается, как только для каждой такой переменной создана спецификация. Отметим, что количество и смысл выходных переменных определяются интерфейсом ПЛК исходя из постановки задачи. (Постановка задачи также определяет наличие таймеров в программе.) Внутренние вспомогательные переменные вводятся по мере необходимости при построении ПЛК-программы, а также, что очень важно, диктуются необходимостью выражения общепрограммных поведенческих свойств на языке логики LTL.

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 программной спецификации оказывает влияние на количество и смысл внутренних вспомогательных переменных ПЛК-программы. (Пример построения спецификации и написания по ней программного кода будет рассмотрен далее.)

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, $\mathrm{Init}(EM) = 1$ означает, что при инициализации переменная EM получает значение 1, т. е. изначально считается, что электромагнит включен, а с помощью выражения $\mathrm{Init}(Tmr) = 5s$ устанавливается пороговое значение для таймера Tmr. Если в спецификации явно не указывается начальное значение для некоторой переменной, то считается, что это значение равняется нулю (независимо от типа данных переменной).

5. Программирование по 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) и $V-$ (убывание значения),

$$V+:\ \mathbf{G}\,\mathbf{X}(\,V{>}\_V \Rightarrow (\mathit{OldValCond}_1 \wedge \mathit{FiringCond}_1 \wedge V = \mathit{NewValExpr}_1 \vee \ldots \vee \mathit{OldValCond}_n \wedge \mathit{FiringCond}_n \wedge V = \mathit{NewValExpr}_n\ ))\,;$$

$$V-:\ \mathbf{G}\,\mathbf{X}(\,V{<}\_V \Rightarrow (\mathit{OldValCond}'_1 \wedge \mathit{FiringCond}'_1 \wedge V = \mathit{NewValExpr}'_1 \vee \ldots \vee \mathit{OldValCond}'_n \wedge \mathit{FiringCond}'_n \wedge V = \mathit{NewValExpr}'_n\ ))\,;$$

ставится в соответствие текстовый блок IF-ELSIF на языке ST

```
IF      OldValCond_1 AND FiringCond_1 THEN V := NewValExpr_1 ; (* V+ *)
...
ELSIF OldValCond_n AND FiringCond_n THEN V := NewValExpr_n;
ELSIF OldValCond'_1 AND FiringCond'_1 THEN V := NewValExpr'_1; (* V- *)
...
ELSIF OldValCond'_n AND FiringCond'_n THEN V := NewValExpr'_n;
END_IF.
```

Например, для спецификации поведения рассмотренной ранее переменной Ctr

$$\begin{aligned}\mathit{Ctr}+:\ \mathbf{G}\,\mathbf{X}(\,\mathit{Ctr} > \_\mathit{Ctr} \Rightarrow (&\_\mathit{Ctr} = 0 \wedge \mathit{PB} \wedge \mathit{K1} \wedge \mathit{Ctr} = 1 \vee\\ &\_\mathit{Ctr} \geqslant 1 \wedge \_\mathit{Ctr} \leqslant 2 \wedge \mathit{PB} \wedge \neg\mathit{K1} \wedge \mathit{Ctr} = \_\mathit{Ctr} + 1)),\end{aligned}$$

$$\begin{aligned}\mathit{Ctr}-:\ \mathbf{G}\,\mathbf{X}(\,\mathit{Ctr} < \_\mathit{Ctr} \Rightarrow (&\_\mathit{Ctr} = 3 \wedge \mathit{PB} \wedge \neg\mathit{K1} \wedge \mathit{Ctr} = 0 \vee\\ &\_\mathit{Ctr} \geqslant 2 \wedge \_\mathit{Ctr} \leqslant 3 \wedge \mathit{PB} \wedge \mathit{K1} \wedge \mathit{Ctr} = 1 \vee\\ &\_\mathit{Ctr} \geqslant 1 \wedge \_\mathit{Ctr} \leqslant 3 \wedge \neg\_\mathit{EM} \wedge \mathit{Ctr} = 0\ ))\end{aligned}$$

имеем следующий программный IF-ELSIF-блок на языке ST

```
IF      _Ctr = 0                    AND PB AND       K1 THEN Ctr := 1;  (* Ctr+ *)
ELSIF _Ctr >= 1 AND _Ctr <= 2 AND PB AND NOT K1 THEN Ctr := _Ctr + 1 ;
ELSIF _Ctr = 3                    AND PB AND NOT K1 THEN Ctr := 0;  (* Ctr- *)
ELSIF _Ctr >= 2 AND _Ctr <= 3 AND PB AND       K1 THEN Ctr := 1;
ELSIF _Ctr >= 1 AND _Ctr <= 3 AND NOT _EM          THEN Ctr := 0;
END_IF.
```

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

При трансляции упрощенных LTL-формул спецификации поведения логической переменной V

$$V+:\ \mathbf{G}\,\mathbf{X}(\,\neg\_V \wedge\ V \Rightarrow \mathit{FiringCond}\,),$$
$$V-:\ \mathbf{G}\,\mathbf{X}(\ \ \_V \wedge \neg V \Rightarrow \mathit{FiringCond}'\,)$$

получаем такой IF-ELSIF-блок

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

Для логической переменной EM LTL-спецификация

$$\mathit{EM}+:\ \mathbf{G}\,\mathbf{X}(\,\neg\_\mathit{EM} \wedge\ \mathit{EM} \Rightarrow \_\mathit{Tmr}.Q\,);$$

$EM-$: $\mathbf{G}\,\mathbf{X}(\quad \_EM \wedge \neg EM \Rightarrow (Z1 \wedge Z3 \wedge Z5 \vee Opn)\,)$.

превращается в ST-код

IF NOT $\_EM$ AND $\_Tmr.Q$ THEN $EM := 1$; (* $EM+$ *)
ELSIF $\_EM$ AND ($Z1$ AND $Z3$ AND $Z5$ OR Opn) THEN $EM := 0$; (* $EM-$ *)
END_IF.

В случае со спецификацией поведения переменной-функции V

V: $\mathbf{G}\,\mathbf{X}(\,V = NewValExpr\,)$

имеем простое присваивание вида

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

На примере рассмотренной ранее переменной Alm

Alm: $\mathbf{G}\,\mathbf{X}(Alm = (K1 \vee K2 \vee K3 \vee K4 \vee K5 \vee K6 \vee K7 \vee K8 \vee K9))$

это выглядит так

$Alm := K1$ OR $K2$ OR $K3$ OR $K4$ OR $K5$ OR $K6$ OR $K7$ OR $K8$ OR $K9$. (* Alm *)

Важно отметить, что после обработки спецификации поведения переменной, которая является входом таймера, необходимо прописать вызов самого функционального блока таймера, чтобы таймер смог сработать на установленном входе и выставить свои выходы.

В примере программы «Электромагнитный кодовый замок» работа с таймером осуществляется следующим образом. Спецификация поведения входа $Tmr.In$ таймера Tmr имеет вид

$Tmr.In$: $\mathbf{G}\,\mathbf{X}(Tmr.In = \neg EM)$.

Программный код на языке ST для таймера Tmr строится с добавлением вызова функционального блока таймера $Tmr()$

$Tmr.In := \mathrm{NOT}EM$; (* $Tmr.In$ *)
$Tmr()$. (* вызов таймера *)

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 весьма 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 вполне 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.

6. Электромагнитный кодовый замок

Рассмотрим дверь с электромагнитным кодовым замком, схема которой изображена на рис. 2. В исходном положении дверь закрыта и удерживается посредством включенного электромагнита, о чем сигнализирует соответствующая лампа. Для открытия двери необходимо нажать (и затем отпустить) последовательно кнопки «1», «3» и «5». Нажатие кнопки не засчитывается, если она была нажата одновременно с другой кнопкой. Кроме того, панель кнопок является неактивной, если электромагнит отключен или нажата внутренняя кнопка «Открыть».

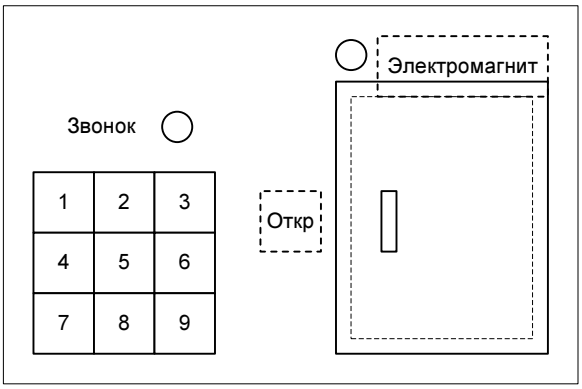

Рис. 2. Схема двери с электромагнитным кодовым замком

Для звуковой защиты от попыток подбора входного кода нажатие каждой кнопки сопровождается звуковым сигналом «Звонок» и включением соответствующей лампы. При верной последовательности нажатий кнопок электромагнит, удерживающий дверь в закрытом положении, отключается на 5 секунд и дверь может быть приведена в открытое положение. Электромагнит может быть отключен (также на 5 секунд) нажатием внутренней кнопки «Открыть». Отключенный электромагнит сбрасывает последовательность нажатий кнопок в начальное состояние.

Задача состоит в реализации электромагнитного кодового замка при помощи ПЛК с 10 входами и 2 выходами. Интерфейс ПЛК, реализующего электромагнитный кодовый замок, представлен на рис. 3.

Рис. 3. Интерфейс ПЛК управления электромагнитным кодовым замком

Глобальные переменные ПЛК-программы определяются интерфейсом ПЛК управления. Кроме необходимости реализации алгоритма решения задачи для ПЛК, введение вспомогательных внутренних переменных во многом диктуется необходимостью выражать на языке темпоральной логики LTL общепрограммные свойства, вытекающие из постановки задачи. Отметим, что описание некоторых внутренних переменных, а также таймеров, следует непосредственно из постановки задачи.

Ниже представлена LTL-спецификация задачи «Электромагнитный кодовый замок». Построение спецификации осуществлялось снизу вверх. Опорными являются общепрограммные (и в то же время конструктивные) LTL-свойства для переменной EM, описывающие ситуации, приводящие соответственно к включению и отключению электромагнита. Свойство $EM+$ требует, чтобы электромагнит переводился из выключенного состояния во включенное только по срабатыванию таймера. Отключение электромагнита, свойство $EM-$, производится только тогда, когда нажата кнопка «Открыть» или подряд были нажаты три кодовые кнопки так, что первой была нажата кнопка K1, переменная $Z1 = 1$, второй — кнопка K3, $Z3 = 1$, а третьей — кнопка K5, $Z5 = 1$.

Ниже символы «&», «|», «$\sim$» и «->» означают соответственно логические «и», «или», «не» и импликацию.

```
Alm:  GX(Alm = K1 | K2 | K3 | K4 | K5 | K6 | K7 | K8 | K9);
Skp:  GX(Skp = (K1+K2+K3+K4+K5+K6+K7+K8+K9>1));
PB:   GX(PB = Alm & ~_Alm & ~Skp & ~Opn & _EM);
Ctr+: GX(Ctr > _Ctr -> (_Ctr =0           & PB &  K1 & Ctr=1    |
                        _Ctr>=1 & _Ctr<=2 & PB & ~K1 & Ctr=_Ctr+1));
Ctr-: GX(Ctr < _Ctr -> (_Ctr =3           & PB & ~K1 & Ctr=0 |
                        _Ctr>=2 & _Ctr<=3 & PB &  K1 & Ctr=1 |
                        _Ctr>=1 & _Ctr<=3 & ~_EM     & Ctr=0  ));
Z1+:  GX(~_Z1 &  Z1 ->  Ctr=1 & PB & K1);
Z1-:  GX( _Z1 & ~Z1 ->  ~_EM);
Z3+:  GX(~_Z3 &  Z3 ->  Ctr=2 & PB & K3);
Z3-:  GX( _Z3 & ~Z3 -> (Ctr=1 | ~_EM) );
Z5+:  GX(~_Z5 &  Z5 ->  Ctr=3 & PB & K5);
Z5-:  GX( _Z5 & ~Z5 -> (Ctr=1 | ~_EM) );
Init(EM)=1; (* по умолчанию переменные инициализируются нулем *)
EM+:  GX(~_EM &  EM ->  _Tmr.Q);
EM-:  GX( _EM & ~EM -> (Z1 & Z3 & Z5 | Opn) );
(* ------- работа с таймером ------- *)
init(Tmr)=5s;
Tmr.In: GX(Tmr.In = ~EM);
```

По этой LTL-спецификации строится ST-программа ПЛК управления электромагнитным кодовым замком.

```
VAR_GLOBAL
  (* Входы *)
  K1, K2, K3, K4, K5, K6, K7, K8, K9, Opn : BOOL;
  (* Выходы *)
  Alm: BOOL;
  EM : BOOL:=1;                                          (* Init(EM)=1 *)
END_VAR
PROGRAM PLC_PRG
VAR
  Tmr: TON:= (PT:= T#5s); (* таймер *)                 (* init(Tmr)=5s *)
  Skp, PB: BOOL;
  Ctr, _Ctr: BYTE;
  Z1,  Z3,  Z5, _Z1, _Z3, _Z5: BOOL;
  _Alm, _EM, _TmrQ: BOOL;
END_VAR
Alm:= K1 OR K2 OR K3 OR K4 OR K5 OR K6 OR K7 OR K8 OR K9;       (* Alm  *)
Skp:=(BOOL_TO_BYTE(K1)+BOOL_TO_BYTE(K2)+BOOL_TO_BYTE(K3)+
      BOOL_TO_BYTE(K4)+BOOL_TO_BYTE(K5)+BOOL_TO_BYTE(K6)+
      BOOL_TO_BYTE(K7)+BOOL_TO_BYTE(K8)+BOOL_TO_BYTE(K9) > 1);  (* Skp  *)
PB:= Alm AND NOT _Alm AND NOT Skp AND NOT Opn AND _EM;          (* PB   *)
IF     _Ctr=0               AND PB AND     K1 THEN Ctr:=1;      (* Ctr+ *)
ELSIF (_Ctr>=1 AND _Ctr<=2) AND PB AND NOT K1 THEN Ctr:=_Ctr+1;
ELSIF  _Ctr=3               AND PB AND NOT K1 THEN Ctr:=0;      (* Ctr- *)
ELSIF (_Ctr>=2 AND _Ctr<=3) AND PB AND     K1 THEN Ctr:=1;
ELSIF (_Ctr>=1 AND _Ctr<=3) AND NOT _EM       THEN Ctr:=0; END_IF;
IF NOT _Z1 AND Ctr=1 AND PB AND K1 THEN Z1:=1;                  (* Z1+  *)
```

```
ELSIF  _Z1 AND NOT _EM             THEN Z1:=0; END_IF;          (* Z1-  *)
IF NOT _Z3 AND Ctr=2 AND PB AND K3 THEN Z3:=1;                  (* Z3+  *)
ELSIF  _Z3 AND (Ctr=1 OR NOT _EM)  THEN Z3:=0; END_IF;          (* Z3-  *)
IF NOT _Z5 AND Ctr=3 AND PB AND K5 THEN Z5:=1;                  (* Z5+  *)
ELSIF  _Z5 AND (Ctr=1 OR NOT _EM)  THEN Z5:=0; END_IF;          (* Z5-  *)
IF NOT _EM AND _TmrQ                    THEN EM:=1;             (* EM+  *)
ELSIF  _EM AND (Z1 AND Z3 AND Z5 OR Opn) THEN EM:=0; END_IF;    (* EM-  *)
(* ------- работа с таймером ------- *)
Tmr.In:= NOT EM;                                              (* Tmr.In *)
Tmr(); (* вызов таймера *)
(* ------- псевдооператорный раздел ------- *)
_EM:=EM; _Z1:=Z1; _Z3:=Z3; _Z5:=Z5;
_Alm:=Alm; _Ctr:=Ctr; _TmrQ:=Tmr.Q;
```

Список литературы

1. *Грис Д.* Наука программирования: Пер. с англ. М.: Мир, 1984. 416 с. (*Gries D.* The Science of Programming. Springer-Verlag, 1981).

2. *Кларк Э. М.*, *Грамберг О.*, *Пелед Д.* Верификация моделей программ: Model Checking: Пер. с англ. М.: МЦНМО, 2002. 416 с. (*Clark E. M.*, *Grumberg O.*, *Peled D. A.* Model Checking. The MIT Press, 2001).

3. *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]).

4. *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]).

5. *Парр Э.* Программируемые контроллеры: руководство для инженера. М.: БИНОМ. Лаборатория знаний, 2007. 516 с. (*Parr E. A.* Programmable Controllers. An engineer's guide. Newnes, 2003. 442 p.).

6. *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]).

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

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

Modeling, Specification and Construction of PLC-programs

Kuzmin E. V., Sokolov V. A.

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

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

A new approach to construction of reliable discrete PLC-programs with timers — programming based on specification and verification — is proposed. Timers are modelled in a discrete way. 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 a LTL-specification. A new approach to programming of PLC is shown by an example.

The proposed programming approach provides an ability of a correctness analysis of PLC-programs using the model checking method. The programming requires fulfillment of the following two conditions: 1) a value of each variable should be changed not more than once per one full PLC-program implementation (per one full working cycle of PLC); 2) a value of each variable should only be changed in one place of a PLC-program.

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.

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.