

УДК 517.9

## Построение ПЛ-программ ПЛК по LTL-спецификации

Рябухин Д. А., Кузьмин Е. В.<sup>1</sup>, Соколов В. А.

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

*e-mail: dmitriy\_ryabukhin@mail.ru, {kuzmin,sokolov}@uniyar.ac.ru*

*получена 3 февраля 2014*

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

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

Целью статьи является описание подхода к программированию ПЛК, который бы обеспечивал возможность анализа корректности ПЛ-программ ПЛК 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 ПЛ-программы ПЛК, которая затем проверяется на корректность (относительно дополнительных общепрограммных LTL-свойств) методом проверки модели с помощью средства верификации Cadence SMV.

---

<sup>1</sup>Работа проводилась при финансовой поддержке РФФИ, грант №12-01-00281-а.

## Введение

Применение программируемых логических контроллеров (ПЛК) в системах управления сложными производственными процессами предъявляет строгие требования корректности к программам ПЛК. Любая программная ошибка считается недопустимой. Вместе с тем программирование ПЛК представляет собой прикладную область, в которой существующие наработки в области формальных методов моделирования и анализа программных систем могут иметь успешное применение.

Ранее в работах [1, 2, 3, 4] был предложен подход к построению и верификации «дискретных» ПЛК-программ, обеспечивающий возможность анализа их корректности с помощью метода проверки модели (model checking) [8], — программирование и верификация по LTL-спецификации. При этом подходе в качестве языка спецификации программного поведения используется язык темпоральной логики LTL. Анализ корректности LTL-спецификации производится с помощью программного средства символьной проверки модели Cadence SMV [11].

В этой статье описывается технология построения и верификации ПЛ-программ ПЛК на основе подхода к программированию и верификации по LTL-спецификации. Технология демонстрируется на примере ПЛ-программы ПЛК управления библиотечным подъемником.

ПЛК — «реагирующая» система, имеющая множество входов, подключаемых посредством датчиков к объекту управления, и множество выходов, подключаемых к исполнительным устройствам [10, 7]. Программа ПЛК выполняется в рабочем цикле: считывание входов, выполнение программы и выставление выходов.

Язык ПЛ (Instruction List) — это типичный ассемблер с аккумулятором и переходами по меткам. Набор инструкций стандартизован (МЭК 61131-3) и не зависит от конкретной целевой платформы. Язык ПЛ позволяет работать с любыми типами данных, вызывать функции и функциональные блоки, реализованные на любом языке стандарта МЭК 61131-3. В составе МЭК-языков ПЛ применяется при создании компактных компонентов, требующих тщательной проработки.

Текст на ПЛ — это текстовый список последовательных инструкций. Каждая инструкция записывается на отдельной строке. Инструкция может включать 4 поля, разделенные пробелами или знаками табуляции:

Метка: Оператор    Операнд                    Комментарий

Метка и комментарий — необязательные поля.

Инструкции языка ПЛ выполняют операции с содержимым аккумулятора. Аккумулятор ПЛ — универсальный контейнер, способный сохранять значения переменных любого типа. Программа на ПЛ выполняется подряд, сверху вниз. В каждом рабочем цикле однократно выполняются все операторы программы. Для изменения порядка выполнения операций применяется переход на метку. Переход выполняется как вверх, так и вниз. Метки являются локальными.

К некоторым операторам языка ПЛ может быть добавлен модификатор. Символ 'N' (negation) вызывает инверсию значения операнда до выполнения инструкции. Операнд должен быть логического типа. Символ 'C' (condition) добавляет проверку условий к командам перехода, вызова и возврата. Команды JMP, CALC, RETC будут выполняться только при значении аккумулятора TRUE. Добавление символа 'N' приводит к сравнению условия с инверсным значением аккумулятора.

Стандартные операторы IL:

- LD — загрузить значение операнда в аккумулятор;
- ST — присвоить значение аккумулятора операнду;
- S — если аккумулятор TRUE, установить логический операнд (TRUE);
- R — если аккумулятор TRUE, сбросить логический операнд (FALSE);
- AND, OR, XOR, NOT — поразрядные логические операторы;
- ADD, SUB, MUL, DIV, MOD — арифметические операторы;
- GT, GE, EQ, NE, LE, LT — операторы сравнения;
- JMP, CAL, RET — операторы перехода к метке, вызова функционального блока и выхода из POU соответственно.

## 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  $\mathcal{S} = (S, s_0, \rightarrow, L)$ , где  $S$  — конечное множество состояний (модели программы),  $s_0 \in S$  — начальное состояние,  $\rightarrow \subseteq S \times S$  — отношение переходов,  $L : S \rightarrow 2^P$  — 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  $\forall i \geq 0$  выполняется  $s_i \rightarrow s_{i+1}$ .

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  $p_i \in P$ :

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

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  $\varphi$  должна выполняться в следующем состоянии,  $F\varphi$  —  $\varphi$  должна выполняться в некотором будущем состоянии пути,  $G\varphi$  —  $\varphi$  должна выполняться в текущем состоянии и во всех будущих состояниях пути,  $\psi U \varphi$  —  $\varphi$  должна выполняться в текущем или будущем состоянии при том, что во всех состояниях (начиная с текущего) до этого

момента должна выполняться формула  $\psi$ . Операторы  $F$  и  $G$  являются производными и вводятся для удобства спецификации свойств:  $F\varphi = true U\varphi$ ,  $G\varphi = \neg F\neg\varphi$ . Кроме того, далее будут использоваться классические логические связки  $\vee$  и  $\Rightarrow$ .

Структура Крипке удовлетворяет формуле (свойству)  $\varphi$  логики LTL, если  $\varphi$  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. [4]). 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 и верификации

В соответствии с концепцией построения и верификации программ ПЛК по LTL-спецификации [2, 3, 4], смысл которой состоит в обеспечении возможности анализа корректности ПЛК-программ методом проверки модели, значение каждой переменной должно изменяться только в одном месте программы и не более одного раза за одно полное выполнение программы при прохождении рабочего цикла ПЛК. 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.

Для описания ситуаций, которые приводят к увеличению или уменьшению значения целочисленной переменной  $V$  предлагается использовать LTL-формулы вида

$$\mathbf{G X}(V >_{\_} V \Rightarrow OldValCond \wedge FiringCond \wedge V = NewValExpr), \quad (1)$$

$$\mathbf{G X}(V <_{\_} V \Rightarrow OldValCond' \wedge FiringCond' \wedge V = NewValExpr'). \quad (1')$$

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  $\mathbf{X}$ .

Условия  $FiringCond$  и  $OldValCond$  являются логическими выражениями над программными переменными и константами, которые строятся с применением операторов сравнения, логических и арифметических операторов и псевдооператора « $\_$ » (который по определению может быть применен 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 формулах (1) и (1') после оператора  $\Rightarrow$  может потребоваться несколько наборов рассмотренных конъюнктивных членов  $OldValCond_i \wedge FiringCond_i \wedge V = NewValExpr_i$  объединенных в дизъюнкцию.

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:

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

которая означает, что всякий раз, когда новое значение переменной  $V$  оказывается больше ее предыдущего значения, записанного в переменной  $\_V$ , из этого следует, что было выполнено условие соответствующего внешнего воздействия  $FiringCond$ .

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

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

Рассмотрим особый случай спецификаций вида (2) и (2'), при котором для  $V$  имеем  $FiringCond = \neg FiringCond'$ . Такая спецификация может быть заменена на одну LTL-формулу вида

$$\mathbf{G X}(V = FiringCond). \quad (3)$$

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.

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, по необходимости сопровождается указанием начального значения переменной, которое она получает при инициализации. Для этого задействуется

ключевое слово `Init`. Например,  $\text{Init}(V) = 1$  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.

Рассмотрим способ построения IL-кода по конструктивной LTL-спецификации поведения программных переменных. Схема трансляции LTL-формулы в IL-код следующая. Двум темпоральным формулам (1) и (1') для целочисленной переменной  $V$  ставятся в соответствие следующие операторы

```

вычисление OldValCond AND FiringCond
JMPCN VL1
вычисление NewValExpr
ST      V
JMP     VLEND

```

VL1:

```

вычисление OldValCond' AND FiringCond'
JMPCN VL2
вычисление NewValExpr'
ST      V
JMP     VLEND

```

VL2:

VLEND:

Если LTL-формулы спецификации поведения переменной  $V$  будет больше двух, то возрастает число меток (по одной метке для каждого нового блока).

Для упрощенных формул (2) и (2') (выставление и снятие сигнала) ставится в соответствие IL-блок

```

вычисление NOT _V AND FiringCond
JMPCN VL1
S      V
JMP     VLEND

```

VL1:

```

вычисление _V AND FiringCond'
JMPCN VL2
R      V
JMP     VLEND

```

VL2:

VLEND:

Для вычисления выражений  $\text{OldValCond}$ ,  $\text{FiringCond}$  и  $\text{NewValExpr}$  осуществляется последовательно. Порядок выполнения команд IL можно изменять при помощи скобок. Инструкции, заключенные в скобки, выполняются в первую очередь. Результат вычисления инструкций в скобках помещается в дополнительный аккумулятор, после чего выполняется команда, содержащая открывающую скобку. Скобки могут быть вложенными. Внутри скобок применять команды `JMP`, `RET`, `CAL` и `LD` нельзя.

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` [9] 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`, записываются команды, реализующие присваивание.

```
LD      V
ST      _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 значение.

В качестве программного средства анализа корректности методом проверки модели мы рассматриваем верификатор Cadence SMV [11]. 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. Когда все программные свойства проверены с положительным результатом, по спецификации строится П-программа ПЛК. Построение SMV-модели описано в статьях [1, 2, 3, 4].

### 3. Библиотечный подъемник

Библиотечный подъемник, схема которого представлена на рис. 1, предназначен для подъема книг по требованию (заполненному формуляру требования) из хранилища подвального этажа на первый и второй этажи библиотеки и возврата книг на подвальный этаж. Вызов кабины подъемника на верхние этажи и отправка кабины с верхних этажей на подвальный этаж осуществляется нажатием кнопок «Вызов 2», «Вызов 1», «Вниз 2» и «Вниз 1» соответственно. Отправка кабины на верхние этажи из подвального этажа производится нажатием кнопок «Вверх 2» и «Вверх 1». Если соответствующая нажатой кнопке команда была принята, зажигается лампа этой кнопки, которая гасится после выполнения команды.

При заходе кабины на заданный этаж включается лампа этого этажа «Этаж 2», «Этаж 1» или «Этаж 0». Кабина подъемника своей двери не имеет, а есть только двери шахты, которые открываются и закрываются вручную. Датчики «Д2», «Д1» и «Д0» служат для определения положения дверей. Датчик двери выдает сигнал, если дверь закрыта. Когда дверь открыта, сигнал снимается. Визуально сигнал от датчика двери определяется с помощью лампы этого датчика.

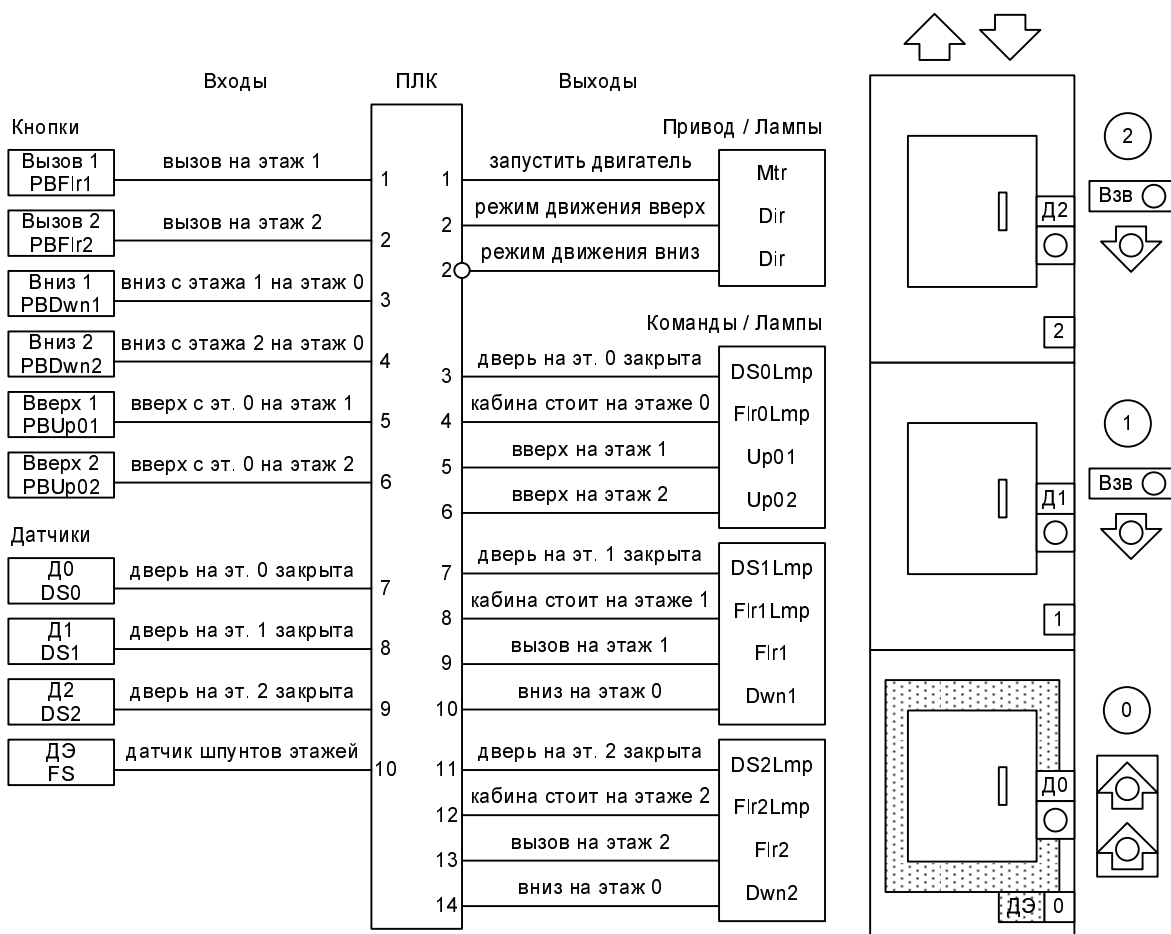


Рис. 1. Интерфейс ПЛК управления и схема библиотечного подъемника

Датчик этажей «ДЭ», который располагается на кабине подъемника, используется для определения положения кабины в шахте. Датчик этажа выставляет сигнал только тогда, когда кабина полностью находится на том или ином этаже, реагируя на специальную металлическую пластину. В противном случае сигнал снимается.

Управление библиотечным подъемником осуществляется с помощью ПЛК, получающего входные сигналы от датчиков и кнопок и подающего выходные сигналы на двигатель подъемника и лампы.

Задача состоит в написании программы для ПЛК с 10 входами и 14 выходами, предназначенного для управления подъемником. Интерфейс ПЛК управления библиотечным подъемником представлен на рис. 1.

Более подробные требования к программе библиотечного подъемника приведены в статье [1]. По этим требованиям в соответствии с концепцией программирования и верификации по LTL-спецификации была построена конструктивная спецификация программы управления библиотечным подъемником.

```

Ctr0+: GX(~_Ctr0 & Ctr0 -> FS & ~_FS & _Ctr1 & ~_Dir);
Ctr0-: GX( _Ctr0 & ~Ctr0 -> FS & ~_FS );
Ctr1+: GX(~_Ctr1 & Ctr1 -> FS & ~_FS & (_Ctr2 & ~_Dir | _Ctr0 & _Dir));
Ctr1-: GX( _Ctr1 & ~Ctr1 -> FS & ~_FS ); init(Ctr1)=1;
    
```



```

Ctr2+: GX(~_Ctr2 & Ctr2 -> FS & ~_FS & _Ctr1 & _Dir);
Ctr2-: GX( _Ctr2 & ~Ctr2 -> FS & ~_FS );
Up01+: GX(~_Up01 & Up01 -> FS & Ctr0 & PBU01);
Up01-: GX( _Up01 & ~Up01 -> FS & Ctr1 & ~_Mtr );
Up02+: GX(~_Up02 & Up02 -> FS & Ctr0 & PBU02);
Up02-: GX( _Up02 & ~Up02 -> FS & Ctr2 & ~_Mtr );
Dwn2+: GX(~_Dwn2 & Dwn2 -> FS & Ctr2 & ~_Mtr & (PBDwn2 | _Tmr.Q));
Dwn2-: GX( _Dwn2 & ~Dwn2 -> FS & Ctr0 & ~_Mtr);
Dwn1+: GX(~_Dwn1 & Dwn1 -> FS & Ctr1 & ~_Mtr & (PBDwn1 | _Tmr.Q));
Dwn1-: GX( _Dwn1 & ~Dwn1 -> FS & Ctr0 & ~_Mtr ); Init(Dwn1)=1;
Flr2+: GX(~_Flr2 & Flr2 -> PBFlr2); Flr2-: GX( _Flr2 & ~Flr2 -> FS & Ctr2 & ~_Mtr);
Flr1+: GX(~_Flr1 & Flr1 -> PBFlr1); Flr1-: GX( _Flr1 & ~Flr1 -> FS & Ctr1 & ~_Mtr);
Dir+: GX(~_Dir & Dir -> FS & Ctr0 & ~_Mtr);
Dir-: GX( _Dir & ~Dir -> FS & (Ctr2 | Ctr1 & Dwn1 & ~Up02 & (~Flr2 | Dwn2)) & ~_Mtr);
DS: GX(DS = DS0 & DS1 & DS2);
Mtr+: GX(~_Mtr & Mtr -> DS & (~FS & (Dwn1 | Dwn2 | Up01 | Up02 | Flr1 | Flr2) |
FS & Ctr0 & (Up01 | Up02 | _Tmr.Q & (Flr1 | Flr2)) |
FS & Ctr1 & Dwn1 | FS & Ctr2 & Dwn2) );
Mtr-: GX( _Mtr & ~Mtr -> (~DS | FS & Ctr0 & ~_Dir | FS & Ctr2 & _Dir |
FS & Ctr1 & (Flr1 & ~Dwn1 | Up01)) );
Tmr.In: GX(Tmr.In = ~Mtr & FS & (Ctr0 & DS0 | Ctr1 & DS1 | Ctr2 & DS2));
Flr0Lmp: GX(Flr0Lmp = ~Mtr & FS & Ctr0); DS0Lmp: GX(DS0Lmp = DS0);
Flr1Lmp: GX(Flr1Lmp = ~Mtr & FS & Ctr1); DS1Lmp: GX(DS1Lmp = DS1);
Flr2Lmp: GX(Flr2Lmp = ~Mtr & FS & Ctr2); DS2Lmp: GX(DS2Lmp = DS2).

```

Эта спецификация позволяет проверять следующие общепрограммные свойства программы управления библиотечным подъемником.

1. Свойство  $\mathbf{G}(Ctr0 + Ctr1 + Ctr2 = 1)$  означает, что в любой момент времени кабина лифта обязательно находится только на одном этаже.

2. «Выход за пределы шахты». Не существует ситуаций, при которых кабина лифта находится точно на подвальном этаже, а двигатель подъемника включен в режиме спуска:  $\mathbf{G}\neg(FS \wedge Ctr0 \wedge Dir = 0 \wedge Mtr)$ . Не существует ситуаций, при которых кабина лифта находится точно на втором этаже, а двигатель подъемника включен в режиме подъема:  $\mathbf{G}\neg(FS \wedge Ctr2 \wedge Dir = 1 \wedge Mtr)$ .

3. «Движение». Всегда верно, что если двигатель подъемника включен (работает), то двери шахты закрыты:  $\mathbf{G}(Mtr \Rightarrow DS)$ . И если двери шахты закрыты и кабина не находится точно на одном из этажей, то двигатель подъемника включен (работает):  $\mathbf{G}(DS \wedge \neg FS \Rightarrow Mtr)$ . Не существует ситуаций, при которых двигатель подъемника выключен, двери шахты закрыты, а кабина не находится точно на одном из этажей:  $\mathbf{G}\neg(\neg Mtr \wedge DS \wedge \neg FS)$ .

4. «Выключение двигателя». Всякий раз, когда двигатель подъемника запускается, то обязательно рано или поздно он будет выключен, т.е. двигатель не может работать бесконечно долго:  $\mathbf{G}(Mtr \Rightarrow \mathbf{F}(\neg Mtr))$ .

5. «Выполнение команды». Если рассмотреть только те программные исполнения, при которых после поступления некоторой команды вызова/отправки на этаж  $Cmd$  до ее выполнения двери шахты будут открываться/закрываются только конечное число раз, то всегда при поступлении команды  $Cmd$  она рано или поздно будет выполнена, где  $Cmd$  — это  $Flr1$ ,  $Flr2$ ,  $Up01$ ,  $Up02$ ,  $Dwn1$  или  $Dwn2$ :  $\mathbf{G}(Cmd \Rightarrow \neg \mathbf{G}(Cmd \mathbf{U} \neg DS)) \Rightarrow \mathbf{G}(Cmd \Rightarrow \mathbf{F}(\neg Cmd))$ .

После проверки общепрограммных свойств происходит построение (по конструктивной LTL-спецификации) IL-программы ПЛК.

```

LDN   _Ctr0      (* Ctr0+ *)
AND   FS
ANDN  _FS
AND   _Ctr1
ANDN  _Dir
JMPCN Ctr0L1
S     Ctr0
JMP  Ctr0LEND

Ctr0L1:
LD     _Ctr0      (* Ctr0- *)
AND   FS
ANDN  _FS
JMPCN Ctr0L2
R     Ctr0
JMP  Ctr0LEND

Ctr0L2:
Ctr0LEND:
LDN   _Ctr1      (* Ctr1+ *)
AND   FS
ANDN  _FS
AND   (_Ctr2
ANDN  _Dir
OR    (_Ctr0
AND   _Dir
)
)
JMPCN Ctr1L1
S     Ctr1
JMP  Ctr1LEND

Ctr1L1:
LD     _Ctr1      (* Ctr1- *)
AND   FS
ANDN  _FS
JMPCN Ctr1L2
R     Ctr1
JMP  Ctr1LEND

Ctr1L2:
Ctr1LEND:
LDN   _Ctr2      (* Ctr2+ *)
AND   FS
ANDN  _FS
AND   _Ctr1
ANDN  _Dir
JMPCN Ctr2L1
S     Ctr2
JMP  Ctr2LEND

Ctr2L1:
LD     _Ctr2      (* Ctr2- *)
AND   FS
ANDN  _FS
JMPCN Ctr2L2
R     Ctr2
JMP  Ctr2LEND

Ctr2L2:
Ctr2LEND:
LDN   _Up01      (* Up01+ *)
AND   FS
AND   Ctr0
AND   PBU01
JMPCN Up01L1
S     Up01
JMP  Up01LEND

Up01L1:
LD     _Up01      (* Up01- *)
AND   FS
AND   Ctr1
ANDN  _Mtr
JMPCN Up01L2
R     Up01
JMP  Up01LEND

Up01L2:
Up01LEND:
LDN   _Up02      (* Up02+ *)
AND   FS
AND   Ctr0
AND   PBU02
JMPCN Up02L1
S     Up02
JMP  Up02LEND

Up02L1:
LD     _Up02      (* Up02- *)
AND   FS
AND   Ctr2
ANDN  _Mtr
JMPCN Up02L2
R     Up02
JMP  Up02LEND

Up02L2:
Up02LEND:
LDN   _Dwn2      (* Dwn2+ *)
AND   FS
AND   Ctr2
ANDN  _Mtr
AND   (PBDwn2
OR    _Tmr.Q
)
)
JMPCN Dwn2L1
S     Dwn2
JMP  Dwn2LEND

Dwn2L1:
LD     _Dwn2      (* Dwn2- *)
AND   FS
AND   Ctr0
ANDN  _Mtr
JMPCN Dwn2L2
R     Dwn2
JMP  Dwn2LEND

Dwn2L2:
Dwn2LEND:
LDN   _Dwn1      (* Dwn1+ *)
AND   FS
AND   Ctr1
ANDN  _Mtr
AND   (PBDwn1
OR    _Tmr.Q
)
)
JMPCN Dwn1L1
S     Dwn1
JMP  Dwn1LEND

Dwn1L1:
LD     _Dwn1      (* Dwn1- *)
AND   FS
AND   Ctr0
ANDN  _Mtr
JMPCN Dwn1L2
R     Dwn1
JMP  Dwn1LEND

Dwn1L2:
Dwn1LEND:
LDN   _Flr2      (* Flr2+ *)
AND   FS
AND   PBF2
JMPCN Flr2L1
S     Flr2
JMP  Flr2LEND

Flr2L1:
LD     _Flr2      (* Flr2- *)
AND   FS
AND   Ctr2
ANDN  _Mtr

```

```

JMPCN Flr2L2
R Flr2
JMP Flr2LEND
Flr2L2:
Flr2LEND:
LDN _Flr1 (* Flr1+ *)
AND FS
AND PBFlr1
JMPCN Flr1L1
S Flr1
JMP Flr1LEND
Flr1L1:
LD _Flr1 (* Flr1- *)
AND FS
AND Ctr1
ANDN _Mtr
JMPCN Flr1L2
R Flr1
JMP Flr1LEND
Flr1L2:
Flr1LEND:
LDN _Dir (* Dir+ *)
AND FS
AND Ctr0
ANDN _Mtr
JMPCN DirL1
S Dir
JMP DirLEND
DirL1:
LD _Dir (* Dir- *)
AND FS
AND (Ctr2)
OR Ctr1
AND Dwn1
ANDN Up02
AND (Dwn2)
ORN Flr2
)
ANDN _Mtr
)
JMPCN DirL2
R Dir
JMP DirLEND
DirL2:
DirLEND:
LD DS0 (* DS *)
AND DS1
AND DS2
ST DS
LDN _Mtr (* Mtr+ *)
AND DS
AND (FS
NOT
AND (Dwn1)
OR Dwn2
OR Up01
OR Up02
OR Flr1
OR Flr2
)
OR (FS
AND Ctr0
AND (Up01)
OR Up02
OR _Tmr.Q
AND (Flr1)
OR Flr2
)
)
)
)
OR (FS
AND Ctr1
AND Dwn1
)
)
)
OR (FS
AND Ctr2
AND _Dir
)
)
OR (FS
AND Ctr1
AND (Flr1)
ANDN Dwn1
OR Up01
)
)
)
JMPCN MtrL2
R Mtr
JMP MtrLEND
MtrL2:
MtrLEND:
LDN Mtr (* Tmr.In *)
AND FS
AND (Ctr0)
AND DS0
OR (Ctr1)
AND DS1
)
OR (Ctr2)
AND DS2
)
)
ST Tmr.In
LDN Mtr (* Flr0Lmp *)
AND FS
AND Ctr0
ST Flr0Lmp
LDN Mtr (* Flr1Lmp *)
AND FS
AND Ctr1
ST Flr1Lmp
LDN Mtr (* Flr2Lmp *)
AND FS
AND Ctr2
ST Flr2Lmp
LD DS0 (* DS0Lmp *)
ST DS0Lmp
LD DS1 (* DS1Lmp *)
ST DS1Lmp
LD DS2 (* DS2Lmp *)
ST DS2Lmp

```

## Список литературы

1. Кузьмин Е. В., Соколов В. А., Рябухин Д. А. Построение и верификация LD-программ ПЛК по LTL-спецификации // Моделирование и анализ информационных систем. Ярославль, 2013. Т. 20, №6 (2013). С. 78–94. [Kuzmin E. V., Sokolov V. A., Ryabukhin D. A. Construction and Verification of PLC LD-programs by LTL-specification // Modeling and analysis of information systems. 2013. V. 20, №6. P. 78–94 (in Russian)].
2. Кузьмин Е. В., Соколов В. А., Рябухин Д. А. Построение и верификация ПЛК-программ по LTL-спецификации // Моделирование и анализ информационных систем. 2013. Т. 20, №4. С. 5–22. [Kuzmin E. V., Sokolov V. A., Ryabukhin D. A. Construction and Verification of PLC-programs by LTL-specification // Modeling and analysis of information systems. 2013. V. 20, №4. P. 5–22 (in Russian)].
3. Кузьмин Е. В., Рябухин Д. А., Шипов А. А. Построение и верификация ПЛК-программ по LTL-спецификации // Международная научно-практическая конференция «Инструменты и методы анализа программ». Кострома, КГТУ, 2013. С. 17–34. [Kuzmin E. V., Ryabukhin D. A., Shipov A. A. Construction and Verification of PLC-programs by LTL-specification // Proc. of Int. Conf. «Tools and Methods of Program Analysis (ТМРА-2013)». Kostroma, KSTU, 2013. P. 17–34 (in Russian)].
4. 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)].
5. 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)].
6. 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)].
7. Петров И. В. Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования. М.: СОЛОН-Пресс, 2004. 256 с. [Petrov I. V. Programmierbare kontrollery. Standartnyye jazyki i priemy prikladnogo projektirovaniya. M.: SOLON-Press, 2004. 256 p. (in Russian)].
8. Clark E. M., Grumberg O., Peled D. A. Model Checking. The MIT Press, 2001.
9. CoDeSys. Controller Development System. <http://www.3s-software.com/>
10. Parr E. A. Programmable Controllers. An engineer's guide. Newnes, 2003. 442 p.
11. SMV. The Cadence SMV Model Checker. <http://www.kenmcmil.com/smv.html>

## Construction of PLC IL-Programs by LTL-Specification

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

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

**Keywords:** programmable logic controllers, software engineering, specification and verification of PLC-programs, Instruction List

An approach to the construction and verification of PLC IL-programs for discrete problems is proposed. For the specification of the program behavior, we use the linear-time temporal logic LTL. Programming is carried out in the IL-language (Instruction List) 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 IL-programs is shown by an example. For a discrete problem, we give an IL-program and its LTL-specification.

The purpose of the article is to describe an approach to programming PLC, which would provide a possibility of IL-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 which 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 (IL-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 IL-program is constructed according to LTL-specification. Then, the SMV-model is analysed by the symbolic model checking tool Cadence SMV.

### Сведения об авторах:

**Рябухин Дмитрий Александрович,**

Ярославский государственный университет им. П.Г. Демидова,  
аспирант;

**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.