Моделирование и анализ информационных систем. Т. 22, № 4 (2015), с. 507–520
Modeling and Analysis of Information Systems. Vol. 22, No 4 (2015), pp. 507–520

DOI: 10.18255/1818-1015-2015-4-507-520

УДК 517.9

О выразительности подхода к построению ПЛК-программ по LTL-спецификации

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

получена 3 августа 2015

Статья посвящена подходу к построению и верификации «дискретных» программ логических контроллеров (ПЛК) по LTL-спецификации. Этот подход обеспечивает возможность анализа корректности программ логических контроллеров с помощью метода проверки модели (Model Checking). В рамках подхода в качестве языка спецификации программного поведения используется язык темпоральной логики LTL. Анализ корректности LTL-спецификации относительно программных свойств производится автоматически с помощью программного средства символьной проверки модели Cadence SMV.

В статье демонстрируется состоятельность подхода к построению и верификации ПЛК-программ по LTL-спецификации с точки зрения тьюринговой мощности. Доказывается, что в соответствии с этим подходом для произвольной счётчиковой машины Минского может быть построена LTL-спецификация, по которой осуществляется её программная реализация на любом из языков программирования ПЛК стандарта МЭК 61131-3. Поскольку счётчиковые машины Минского равномощны машинам Тьюринга, то и рассматриваемый подход к программированию ПЛК будет обладать тьюринговой мощностью.

В доказательстве основное внимание уделяется заданию поведения счётчиковой машины в виде набора LTL-формул и сопоставлению этим формулам конструкций языков ST и SFC. SFC представляет интерес с точки зрения специфики графического языка, а язык ST рассматривается в качестве базового в том смысле, что реализация счётчиковой машины на языках IL, FBD/CFC и LD сводится к переписыванию на них конструкций ST-программы.

Идея доказательства демонстрируется на примере трехсчетчиковой машины Минского, реализующей функцию возведения числа в квадрат.

Ключевые слова: программируемые логические контроллеры (ПЛК), построение и верификация ПЛК-программ, LTL-спецификация, счётчиковые машины Минского

Для цитирования: Кузьмин Е. В., Рябухин Д. А., Соколов В. А., "О выразительности подхода к построению ПЛК-программ по LTL-спецификации", *Моделирование и анализ информационных систем*, **22**:4 (2015), 507–520.

Об авторах:
Кузьмин Егор Владимирович, orcid.org/0000-0003-0500-306X, доктор физ.-мат. наук, доцент, профессор кафедры теоретической информатики, Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: kuzmin@uniyar.ac.ru

Рябухин Дмитрий Александрович, orcid.org/0000-0002-0799-8868, аспирант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: dmitriy_ryabukhin@mail.ru

Соколов Валерий Анатольевич, orcid.org/0000-0003-1427-4937, доктор физ.-мат. наук, профессор, зав. кафедрой теоретической информатики, Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: sokolov@uniyar.ac.ru

Благодарности:

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

Введение

Статья посвящена предложенному ранее [1–5, 11–13] подходу к построению и верификации «дискретных» программ логических контроллеров по LTL-спецификации. Этот подход обеспечивает возможность анализа корректности программ логических контроллеров с помощью метода проверки модели (Model Checking) [8, 9]. В рамках подхода в качестве языка спецификации программного поведения используется язык темпоральной логики LTL. Анализ корректности LTL-спецификации относительно программных свойств производится автоматически с помощью программного средства символьной проверки модели Cadence SMV [16].

Программируемый логический контроллер (ПЛК) — «реагирующая» система, имеющая множество входов, подключаемых посредством датчиков к объекту управления, и множество выходов, подключаемых к исполнительным устройствам [7, 14]. Программа ПЛК выполняется в рабочем цикле: считывание входов, выполнение программы и выставление выходов. Применение ПЛК в системах управления сложными производственными процессами предъявляет строгие требования корректности к программам ПЛК.

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. Разновидностью FBD является язык непрерывных функциональных схем CFC (Continuous Function Chart), позволяющий свободно размещать компоненты и соединения. 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.

В данной статье демонстрируется состоятельность подхода к построению и верификации ПЛК-программ по LTL-спецификации с точки зрения тьюринговой мощности. Доказывается, что в соответствии с этим подходом для произвольной счётчиковой машины Минского может быть построена LTL-спецификация, по которой осуществляется её программная реализация на любом из языков программирования ПЛК стандарта МЭК 61131-3. Поскольку счётчиковые машины Минского равномощны машинам Тьюринга [6, 15], то и рассматриваемый подход к программированию ПЛК будет обладать тьюринговой мощностью. В доказательстве основное внимание уделяется заданию поведения счётчиковой машины в виде набора LTL-формул и сопоставлению этим формулам конструкций языков ST и SFC. SFC представляет интерес с точки зрения специфики графического языка, а язык ST рассматривается в качестве базового в том смысле, что реализация счётчиковой машины на языках IL, FBD/CFC и LD сводится к переписыванию на них конструкций ST-программы. Идея доказательства демонстрируется на примере.

1. Основные понятия и определения

1.1. Построение программ ПЛК по LTL-спецификации

Смысл концепции программирования ПЛК по LTL-спецификации [1–5,11–13] состоит в обеспечении возможности анализа корректности ПЛК-программ методом проверки модели [8,9]. В соответствии с этой концепцией 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 задаётся с помощью пары явных LTL-формул и одной неявной. Первая LTL-формула описывает ситуации, при которых происходит возрастание значения соответствующей переменной, вторая LTL-формула определяет условия, приводящие к уменьшению значения переменной. Третья LTL-формула в явном виде не прописывается, поскольку всегда имеет жёсткую конструкцию, составленную из элементов первых двух формул. Она описывает условия, при которых переменная не меняет своего значения за один проход рабочего цикла ПЛК. 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.

Для описания ситуаций, которые приводят к увеличению и уменьшению значения целочисленной переменной V, используются LTL-формулы вида

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

$$\mathbf{G}\,\mathbf{X}(\ V{<}\_V \Rightarrow \mathit{OldValCond}' \wedge \mathit{FiringCond}' \wedge V = \mathit{NewValExpr}'\ ). \tag{2}$$

Символ лидирующего подчеркивания «_» в обозначении переменной $\_V$ воспринимается как псевдооператор, позволяющий обратиться к значению переменной V, которое она имела в предыдущем состоянии (после последнего полного прохода рабочего цикла ПЛК). При этом псевдооператор может использоваться только под действием темпорального оператора **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 применим только к переменным). Выражение *FiringCond* описывает ситуации, при которых возникает необходимость изменения значения переменной V, если это, конечно, допускается условием *OldValCond*. Выражение *NewValExpr*, определяющее новое значение переменной V, 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 символа оператора импликации $\Rightarrow$ может потребоваться несколько наборов рассмотренных конъюнктивных членов $\mathit{OldValCond}_i \wedge \mathit{FiringCond}_i \wedge V = \mathit{NewValExpr}_i$ объединенных в дизъюнкцию.

Аналогичный смысл имеют выражения $\mathit{FiringCond}'$, $\mathit{OldValCond}'$ и $\mathit{NewValExpr}'$.

В случае с переменной логического типа данных для спецификации ее поведения используются более простые LTL-формулы:

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

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

которые означают, что всякий раз, когда новое значение переменной V оказывается больше или меньше ее предыдущего значения, записанного в переменной $\_V$, из этого следует, что было выполнено условие соответствующего внешнего воздействия $\mathit{FiringCond}$ или $\mathit{FiringCond}'$.

Неявная LTL-формула сохранения прежнего значения переменной V имеет вид

$$\mathbf{G}\,\mathbf{X}(\ V{=}\_V \Rightarrow \neg(\mathit{OldValCond} \wedge \mathit{FiringCond}) \wedge \neg(\mathit{OldValCond}' \wedge \mathit{FiringCond}')\ ). \qquad (3)$$

Для логической переменной V эта неявная LTL-формула выглядит как

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

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.

1.2. Счётчиковая машина Минского

Счетчиковая машина Минского M представляет собой набор (q_0, q_n, Q, X, Δ), где $Q = \{q_0, \ldots, q_n\}$ — конечное непустое множество состояний машины; $q_0 \in Q$ — начальное состояние; $q_n \in Q$ — финальное состояние; $X = \{x_1, \ldots, x_m\}$ — конечное непустое множество счетчиков, которые могут принимать значения из $\mathbb{N} \cup \{0\}$; $\Delta = \{\delta_0, \ldots, \delta_{n-1}\}$ — набор правил переходов по состояниям машины; δ_i — правило переходов для состояния q_i. Состояния q_i, $0 \leqslant i \leqslant n-1$, подразделяются на два типа. Состояния первого типа имеют правила переходов вида:

$$(\delta_i)\ \ q_i\colon\ x_j := x_j + 1;\ \mathsf{goto}\ q_k,$$

где $1 \leqslant j \leqslant m, 0 \leqslant k \leqslant n$. Для состояний второго типа имеем, $1 \leqslant j \leqslant m, 0 \leqslant k, l \leqslant n$:

$$(\delta_i)\ \ q_i\colon\ \mathsf{if}\ x_j > 0\ \mathsf{then}\ (x_j := x_j - 1;\ \mathsf{goto}\ q_k)\ \mathsf{else}\ \mathsf{goto}\ q_l.$$

Для финального состояния q_n правило перехода не предусмотрено. Это означает, что при попадании в состояние q_n машина Минского M завершает свою работу.

Конфигурация машины Минского — это набор $(q_i, c_1, \ldots, c_m)$, где q_i — состояние машины, $c_1, \ldots, c_m$ — натуральные числа (включая ноль), являющиеся значениями соответствующих счетчиков.

Исполнением машины Минского называется последовательность конфигураций $s_0\, s_1\, s_2\, s_3\, s_4\, \ldots$, индуктивно определяемая в соответствии с правилами переходов. Счетчиковая машина имеет одно исполнение из начальной конфигурации s_0, так как для каждого состояния предусмотрено не более одного правила переходов. Машина, получив на вход некоторый набор значений счетчиков, стартует из состояния q_0 и либо останавливается в состоянии q_n с выходным набором значений счетчиков, либо зацикливается, реализуя тем самым частичную числовую функцию.

2. Реализация счётчиковой машины Минского

Теорема 1. *Любая счётчиковая машина Минского может быть реализована на языках программирования логических контроллеров IL, ST, FBD/CFC, LD и SFC в соответствии с подходом к построению и верификации программ логических контроллеров по LTL-спецификации.*

Доказательство. Рассмотрим произвольную счётчиковую машину Минского M. Сопоставим каждому счётчику x_j и каждому состоянию q_i машины M программную переменную `xj` типа данных «беззнаковое целое» и логическую программную переменную `qi` соответственно. При этом все переменные-состояния, кроме `q0`, инициализируются нулём, а переменной-состоянию `q0` изначально присваивается логическое значение 1 (логическая истина). Переменные-счётчики при инициализации получают начальные значения соответствующих счётчиков машины M. Реализация машины Минского M осуществляется в виде программы на языке программирования логических контроллеров. Переход из одной конфигурации машины в другую будет соответствовать исполнению программы в рамках одного прохода рабочего цикла ПЛК.

В соответствии с подходом к построению и верификации программ по LTL-спецификации доказательство теоремы проводится в два этапа. Первый этап предполагает создание LTL-спецификации счётчиковой машины Минского M. Второй этап – построение по этой LTL-спецификации ПЛК-программ (на разных языках программирования), реализующих поведение машины M.

Первый этап. Запишем требуемое поведение каждой переменной программной реализации счётчиковой машины M с помощью пары LTL-формул.

Начнём с описания поведения переменных-счётчиков. LTL-формула, учитывающая ситуации, которые приводят к увеличению значения переменной-счётчика `xj`, имеет следующий вид:

$$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} > \texttt{\_xj} \ \Rightarrow\ (\texttt{\_qi} \vee \texttt{\_qr} \vee \cdots \vee \texttt{\_qs}) \wedge \texttt{xj=\_xj+1}\ ),$$

где переменные `_qi`, `_qr`, …, `_qs` соответствуют таким машинным состояниям первого типа q_i, q_r, …, q_s, в правилах перехода которых участвует счётчик x_j. Например, для состояния первого типа q_i в описании счётчиковой машины M должно быть правило перехода (δ_i) $q_i\colon\ x_j := x_j + 1;$ goto q_k, которое порождает условие LTL-спецификации вида `_qi` $\wedge$ `xj=_xj+1`.

Если в описании машины M для счётчика x_j правил перехода первого типа нет, то LTL-формула «возрастания» для программной переменной `xj` следующая:

$$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} > \texttt{\_xj} \ \Rightarrow\ \text{false}\ ).$$

LTL-формула, учитывающая ситуации, которые приводят к уменьшению значения переменной-счётчика `xj`, имеет вид

$$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} < \texttt{\_xj} \ \Rightarrow\ (\texttt{\_qi} \vee \texttt{\_qr} \vee \cdots \vee \texttt{\_qs}) \wedge \texttt{\_xj}>0 \wedge \texttt{xj=\_xj}-1\ ),$$

где переменные `_qi`, `_qr`, …, `_qs` соответствуют машинным состояниям второго типа q_i, q_r, …, q_s, в правилах перехода которых участвует счётчик x_j, т. е. для состояния второго типа q_i в описании машины M должно быть правило перехода (δ_i) $q_i\colon$ if $x_j > 0$ then $(x_j := x_j - 1;$ goto $q_k)$ else goto q_l, которое порождает условие LTL-спецификации вида `_qi` $\wedge$ `_xj`$>0 \wedge$ `xj=_xj`-1.

Если в описании машины M для счётчика x_j правил перехода второго типа нет, то LTL-формула «убывания» для программной переменной `xj` строится просто как

$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} < \texttt{\_xj} \ \Rightarrow\ \text{false}\ )$.

Теперь опишем построение LTL-спецификации поведения программных переменных-состояний. LTL-формула, учитывающая условия, при которых счётчиковая машина M переходит из некоторого текущего состояния в новое состояние q_k, т. е. логическая программная переменная `qk` получает значение 1, имеет следующий вид:

$\mathbf{G}\,\mathbf{X}(\ \neg\texttt{\_qk} \wedge \texttt{qk} \ \Rightarrow\ \texttt{\_qi} \wedge \texttt{\_xj}{>}0 \vee \cdots \vee \texttt{\_qr} \wedge \neg(\texttt{\_xt}{>}0) \vee \cdots \vee \texttt{\_qs}\ )$,

где условия вида $\texttt{\_qi} \wedge \texttt{\_xj}{>}0$ соответствуют правилам перехода второго типа

$(\delta_i)\ \ q_i\colon\ \mathsf{if}\ x_j > 0\ \mathsf{then}\ (x_j := x_j - 1;\ \mathsf{goto}\ \mathbf{q_k})\ \mathsf{else\ goto}\ q_l$,

а условия вида $\texttt{\_qr} \wedge \neg(\texttt{\_xt}{>}0)$ – правилам

$(\delta_r)\ \ q_r\colon\ \mathsf{if}\ x_t > 0\ \mathsf{then}\ (x_t := x_t - 1;\ \mathsf{goto}\ q_l)\ \mathsf{else\ goto}\ \mathbf{q_k}$.

Условия с одной программной переменной вида `_qs` соответствуют правилам перехода первого типа $(\delta_s)\ \ q_s\colon\ x_g := x_g + 1;\ \mathsf{goto}\ \mathbf{q_k}$.

Важно отметить, что все программные переменные приведённой LTL-формулы, стоящие в условиях после оператора импликации, должны быть отличными от переменной-состояния `qk`, т. е. переход в то же самое состояние не специфицируется.

Если в состояние q_k машины M нет ни одного перехода, то для переменной `qk` LTL-формула «активации» строится как

$\mathbf{G}\,\mathbf{X}(\ \neg\texttt{\_qk} \wedge \texttt{qk} \ \Rightarrow\ \text{false}\ )$.

LTL-формула деактивации (выхода из) состояния q_k, которому соответствует правило перехода без петель (только с переходами в другие состояния), для программной переменной `qk` имеет простой вид:

$\mathbf{G}\,\mathbf{X}(\ \texttt{\_qk} \wedge \neg\texttt{qk} \ \Rightarrow\ \text{true}\ )$,

где логическая константа true означает, что переменная `qk` должна быть сброшена в ноль уже на следующем проходе рабочего ПЛК после её активации, т. е. после присваивания значения 1. Несмотря на то, что импликация внутри LTL-формулы является тавтологией, построение этой формулы имеет смысл, поскольку условие срабатывания в виде константы true участвует в «неявной» LTL-формуле, описывающей ситуации, при которых переменная сохраняет своё значение после одного исполнения программы. Эта неявная LTL-формула представлена ниже и по сути запрещает переменной `qk` иметь значение 1 дольше одного рабочего цикла программы:

$\mathbf{G}\,\mathbf{X}(\ \texttt{\_qk} = \texttt{qk} \ \Rightarrow\ \neg(\neg\texttt{\_qk} \wedge \mathit{FiringCond}) \wedge \neg(\texttt{\_qk} \wedge \text{true})\ )$.

Если правило перехода для q_k имеет петлю $(\delta_k)\ \ \mathbf{q_k}\colon\ x_h := x_h + 1;\ \mathsf{goto}\ \mathbf{q_k}$ или две петли $(\delta_k)\ \ \mathbf{q_k}\colon\ \mathsf{if}\ x_h > 0\ \mathsf{then}\ (x_h := x_h - 1;\ \mathsf{goto}\ \mathbf{q_k})\ \mathsf{else\ goto}\ \mathbf{q_k}$, или же состояние q_k является финальным состоянием, то программная переменная `qk` будет иметь следующую LTL-формулу «деактивации»:

$\mathbf{G}\,\mathbf{X}(\ \texttt{\_qk} \wedge \neg\texttt{qk} \ \Rightarrow\ \text{false}\ )$.

Если состояние второго типа q_k имеет в правиле перехода с переменной x_h только одну петлю, то для программной переменной `qk` выбирается соответствующий вариант LTL-формулы

$\mathbf{G}\,\mathbf{X}(\ \texttt{\_qk} \wedge \neg\texttt{qk} \ \Rightarrow\ \texttt{\_xh}{>}0\ )$ или $\mathbf{G}\,\mathbf{X}(\ \texttt{\_qk} \wedge \neg\texttt{qk} \ \Rightarrow\ \neg(\texttt{\_xh}{>}0)\ )$.

Второй этап. Имея спецификацию поведения каждой переменной-счётчика и каждой переменной-состояния в виде пары LTL-формул, опишем сначала способ построения программной реализации счётчиковой машины M на языке ST.

Если счётчик x_j машины Минского M участвует в правилах перехода обоих типов, LTL-спецификация поведения программной переменной-счётчика `xj` будет иметь следующий вид:

$\mathbf{G}\,\mathbf{X}($ `xj>_xj` $\Rightarrow$ (`_qi` $\vee$ `_qk` $\vee \cdots \vee$ `_ql`) $\wedge$ `xj=_xj+1` $)$;
$\mathbf{G}\,\mathbf{X}($ `xj<_xj` $\Rightarrow$ (`_qr` $\vee$ `_qs` $\vee \cdots \vee$ `_qt`) $\wedge$ `_xj>0` $\wedge$ `xj=_xj-1` $)$.

По этой паре LTL-формул с учётом третьей неявной LTL-формулы спецификации может быть построен следующий программный код на языке ST:

```
IF    (_qi OR _qk OR ... OR _ql)           THEN xj:=_xj+1;
ELSIF (_qr OR _qs OR ... OR _qt) AND _xj>0 THEN xj:=_xj-1; END_IF;.
```

Если одна из формул LTL-спецификации имеет вид $\mathbf{G}\,\mathbf{X}($ `xj>_xj` $\Rightarrow$ false $)$ или $\mathbf{G}\,\mathbf{X}($ `xj<_xj` $\Rightarrow$ false $)$, то соответствующая ветка блока IF-ELSIF не строится. Если обе формулы такого вида, то им блок IF-ELSIF не сопоставляется.

Если состояние q_k счётчиковой машины M имеет входы и выходы без петель, то LTL-спецификация поведения программной переменной-состояния `qk` выглядит следующим образом:

$\mathbf{G}\,\mathbf{X}($ $\neg$`_qk` $\wedge$ `qk` $\Rightarrow$ `_qi` $\wedge$ `_xj>0` $\vee \cdots \vee$ `_qr` $\wedge \neg($`_xt>0`$) \vee \cdots \vee$ `_qs` $)$;
$\mathbf{G}\,\mathbf{X}($ `_qk` $\wedge \neg$`qk` $\Rightarrow$ true $)$.

По этим двум формулам также с учётом третьей неявной LTL-формулы спецификации строится следующий программный код на языке ST:

```
IF NOT _qk AND (_qi AND     _xj>0  OR ... OR
                _qr AND NOT(_xt>0) OR ... OR _qs) THEN qk:=1;
ELSIF  _qk                                        THEN qk:=0; END_IF;.
```

Для формул вида $\mathbf{G}\,\mathbf{X}($ $\neg$`_qk` $\wedge$ `qk` $\Rightarrow$ false $)$ или $\mathbf{G}\,\mathbf{X}($ `_qk` $\wedge \neg$`qk` $\Rightarrow$ false $)$ соответствующие ветки IF-ELSIF блока не строятся. Если из состояния q_k нельзя перейти в отличное от него состояние и не существует перехода в него из другого состояния, то переменной-состоянию `qk` ST-код не сопоставляется. В том случае, если вторая LTL-формула («деактивации») имеет вид $\mathbf{G}\,\mathbf{X}($ `_qk` $\wedge \neg$`qk` $\Rightarrow$ `_xh>0` $)$ или $\mathbf{G}\,\mathbf{X}($ `_qk` $\wedge \neg$`qk` $\Rightarrow \neg($`_xh>0`$)$ $)$, то во вторую ветку блока IF-ELSIF добавляется через оператор `AND` условие `_xh>0` или `NOT(_xh>0)` соответственно.

Отметим, что порядок IF-ELSIF блоков в ST-программе, соответствующей счётчиковой машине M, не имеет значения, поскольку новое значение каждой переменной зависит от предыдущих значений переменных, полученных на последнем проходе рабочего цикла ПЛК. После построения всех IF-ELSIF блоков (по одному на программную переменную) необходимо в конец ST-программы добавить псевдооператорный раздел, реализующий оператор лидирующего подчёркивания «_»:

```
_x1:=x1; _x2:=x2; ... _xm:=xm;
_q0:=q0; _q1:=q1; ... _qn:=qn;.
```

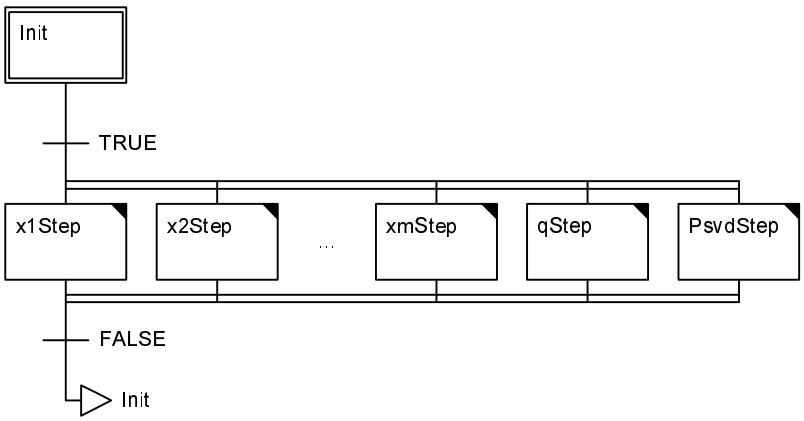
Таким образом, ST-программа машины M будет содержать переменные-счётчики `x1`, `x2`, ..., `xm`, переменные-состояния `q0`, `q1`, ..., `qn` и псевдооператорные переменные `_x1`, `_x2`, ..., `_xm`, `_q0`, `_q1`, ..., `_qn`. При этом переменная-состояние `q0` инициализируется единицей, а переменные-счётчики при инициализации получают

начальные значения соответствующих счётчиков машины M. Остальные переменные, включая псевдооператорные переменные, инициализируются нулём.

Для языков IL, FBD/CFC и LD ограничимся лишь замечанием, что реализация машины M на этих языках главным образом (с небольшими поправками на специфику) сводится к переписыванию на них IF-ELSIF блоков и псевдооператорного раздела ST-программы.

Рассмотрим реализацию машины M на графическом языке последовательных функциональных схем SFC. В качестве языка SFC будем рассматривать упрощенный SFC, имеющийся, например, в инструментальном средстве программирования CoDeSys [10]. В упрощенном SFC каждому шагу могут быть сопоставлены действия трех типов – текущее, входное и выходное. Шаги, содержащие действие, на схеме отличаются тем, что верхний правый угол прямоугольника шага закрашен. Пока шаг активен, текущее действие будет выполняться один раз в каждом рабочем цикле. Выход из шага, т. е. его деактивация, осуществляется при выполнении условий на переходе из шага. Входное действие обозначается сегментом «E» (Entry) в нижнем левом углу прямоугольника шага и выполняется однократно при активации шага. Выходное действие обозначается сегментом «X» (eXit) в нижнем правом углу прямоугольника шага и выполняется однократно при завершении работы шага (при выходе из шага, но на следующем проходе рабочего цикла до активации следующего шага). Каждое из действий, а также громоздкие условия на переходах, могут быть реализованы на любом из языков стандарта МЭК 61131-3.

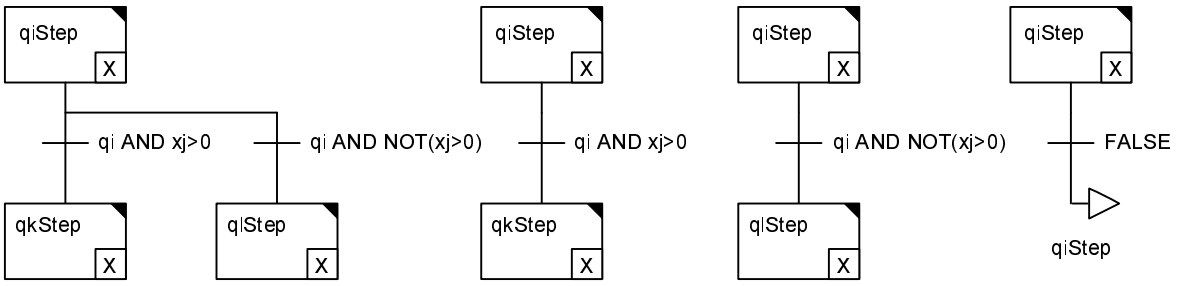
Реализация машины M на упрощенном SFC состоит из двух схем – основной и вложенной. Основная SFC-схема представлена ниже.



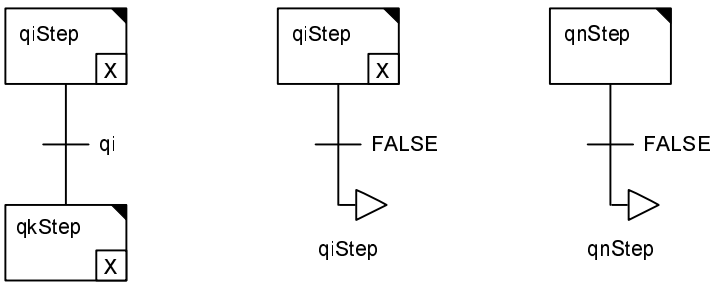
Действию шага x1Step соответствует программная реализация для пары LTL-формул спецификации поведения переменной-счетчика `x1`. Например, действием этого шага может быть уже рассмотренная IF-ELSIF конструкция на языке ST. Аналогичным образом действиям шагов x2Step, ..., xmStep сопоставляются программные реализации для пар LTL-формул спецификации поведения переменных-счетчиков `x2`, ..., `xm` соответственно. Действием шага PsvdStep является весь псевдооператорный раздел, поскольку этот шаг в рамках одного прохода рабочего цикла в программе будет выполняться последним.

Действию шага qStep соответствует вложенная SFC-схема, которая строится из следующих фрагментов с привязкой к переменным-состояниям. Для состояния

второго типа q_i машины M в зависимости от наличия петель в правилах перехода имеем фрагменты (для правил без петель, с одной петлёй и с двумя петлями соответственно):



Для состояния первого типа q_i машины M в зависимости от наличия петли в правиле перехода, а также для финального состояния q_n, имеем фрагменты (для правила без петли, с петлёй и для финального состояния соответственно):



Начальным шагом вложенной SFC-схемы будет являться шаг q0Step. Текущему и выходному действиям шага qiStep соответствует одна и та же программная реализация (на любом из языков IL, ST, FBD/CFC и LD) пары LTL-формул спецификации поведения переменной-состояния `qi`. Выходное действие «X» (eXit) необходимо для сбрасывания значения переменой `qi` в 0 после срабатывания условия выхода из шага qiStep. Вложенная SFC-схема в точности будет соответствовать графу переходов счётчиковой машины M.

Таким образом, мы показали, что для произвольной n-счётчиковой машины Минского M может быть задана LTL-спецификация её поведения, по которой осуществляется построение программной реализации машины M на любом из стандартных языков программирования ПЛК. □

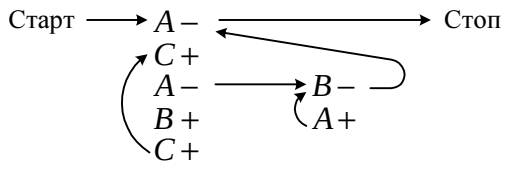
3. Счетчиковая машина возведения в квадрат

Рассмотрим в качестве примера трехсчетчиковую машину Минского $3cM$, реализующую функцию возведения числа N в квадрат. Эта счетчиковая машина имеет восемь состояний q_0, q_1, …, q_7, где q_0 является начальным состоянием, а q_7 – это финальное состояние. В начальной конфигурации счетчик A получает значение N, а начальные значения двух других счетчиков B и C равны нулю. В финальной конфигурации результат вычисления будет содержаться в счетчике C, а значения остальных счетчиков A и B будут равны нулю. Правила перехода по состояниям машины $3cM$ представлены ниже:

(δ_0) q_0: if $A > 0$ then $(A := A - 1;$ goto $q_1)$ else goto q_7;
(δ_1) q_1: $C := C + 1;$ goto q_2;
(δ_2) q_2: if $A > 0$ then $(A := A - 1;$ goto $q_3)$ else goto q_5;
(δ_3) q_3: $B := B + 1;$ goto q_4;
(δ_4) q_4: $C := C + 1;$ goto q_1;
(δ_5) q_5: if $B > 0$ then $(B := B - 1;$ goto $q_6)$ else goto q_0;
(δ_6) q_6: $A := A + 1;$ goto q_5.

В качестве пояснения отметим, что при построении этой счетчиковой машины использовался тот факт, что $N^2 = (2N - 1) + (2N - 3) + \cdots + 3 + 1$.

Правила перехода машины $3cM$ можно наглядно представить в следующем графическом виде, где для некоторой переменной V обозначение «$V+$» соответствует безусловному увеличению счетчика на единицу, а «$V-$» используется для обозначения условного вычитания единицы с переходом в другое состояние по правосторонней стрелке в случае нулевого значения счетчика V.



Построим LTL-спецификацию трехсчетчиковой машины Минского $3cM$ возведения числа N в квадрат. В этой спецификации переменная-счетчик A при инициализации получает значение N. Переменная-состояние q0 инициализируется единицей, т. е. изначально q0 = 1. Остальные переменные (в том числе и псевдооператорные) при инициализации обнуляются.

Init(A) = N;
A+ : **GX**(A > _A ⇒ _q6 ∧ A = _A + 1);
A− : **GX**(A < _A ⇒ (_q0 ∨ _q2) ∧ _A > 0 ∧ A = _A − 1);
B+ : **GX**(B > _B ⇒ _q3 ∧ B = _B + 1);
B− : **GX**(B < _B ⇒ _q5 ∧ _B > 0 ∧ B = _B − 1);
C+ : **GX**(C > _C ⇒ (_q1 ∨ _q4) ∧ C = _C + 1);
C− : **GX**(C < _C ⇒ false);
Init(q0) = 1;
q0+ : **GX**(¬_q0 ∧ q0 ⇒ _q5 ∧ ¬(_B > 0));
q0− : **GX**(_q0 ∧ ¬q0 ⇒ true);
q1+ : **GX**(¬_q1 ∧ q1 ⇒ _q0 ∧ _A > 0 ∨ _q4);
q1− : **GX**(_q1 ∧ ¬q1 ⇒ true);
q2+ : **GX**(¬_q2 ∧ q2 ⇒ _q1);
q2− : **GX**(_q2 ∧ ¬q2 ⇒ true);
q3+ : **GX**(¬_q3 ∧ q3 ⇒ _q2 ∧ _A > 0);
q3− : **GX**(_q3 ∧ ¬q3 ⇒ true);
q4+ : **GX**(¬_q4 ∧ q4 ⇒ _q3);
q4− : **GX**(_q4 ∧ ¬q4 ⇒ true);
q5+ : **GX**(¬_q5 ∧ q5 ⇒ _q2 ∧ ¬(_A > 0) ∨ _q6);
q5− : **GX**(_q5 ∧ ¬q5 ⇒ true);

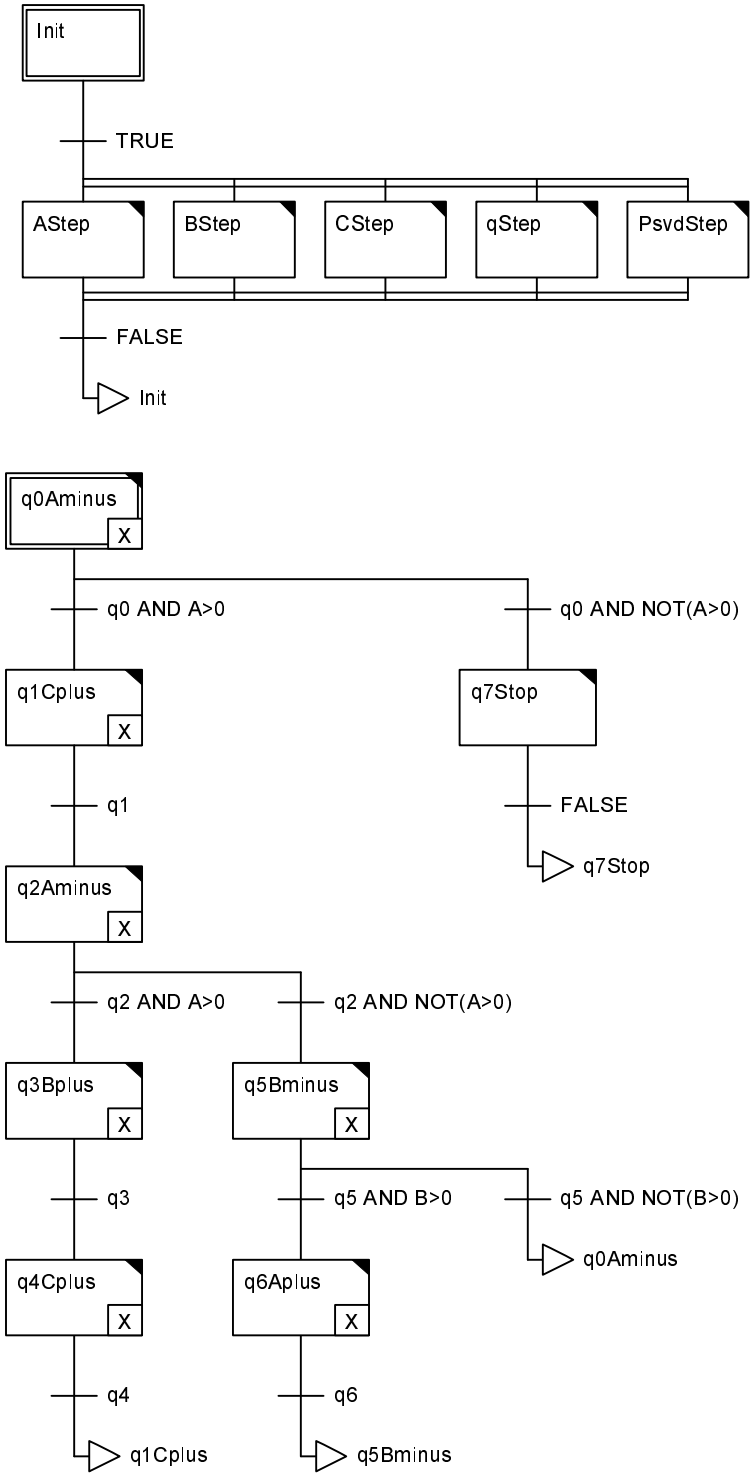
$\texttt{q6+}\ :\ \mathbf{GX}(\ \neg\texttt{\_q6} \wedge\ \ \texttt{q6}\ \Rightarrow\ \texttt{\_q5} \wedge \texttt{\_B} > 0\ );$
$\texttt{q6}-\ :\ \mathbf{GX}(\ \ \texttt{\_q6} \wedge \neg\texttt{q6}\ \Rightarrow\ \texttt{true}\ );$
$\texttt{q7+}\ :\ \mathbf{GX}(\ \neg\texttt{\_q7} \wedge\ \ \texttt{q7}\ \Rightarrow\ \texttt{\_q0} \wedge \neg(\texttt{\_A} > 0)\ );$
$\texttt{q7}-\ :\ \mathbf{GX}(\ \ \texttt{\_q7} \wedge \neg\texttt{q7}\ \Rightarrow\ \texttt{false}\ ).$

Реализация счётчиковой машины $3cM$ на языке программирования ST по приведенной выше LTL-спецификации имеет следующий вид.

```
VAR (* раздел описания переменных *)
   A  : INT :=N; (* N - входное значение, возводимое в квадрат *)
   q0 : BOOL:=1; (* начальное состояние *)
   q1, q2, q3, q4, q5, q6 : BOOL; (* состояния *)
   q7 : BOOL; (* финальное состояние *)
   _A, _B, _C : INT; (* псевдооператорные переменные *)
   _q0, _q1, _q2, _q3, _q4, _q5, _q6, _q7 : BOOL;
END_VAR
   (* Переменные *)
   IF     _q6                   THEN A:=_A+1;          (* A+ *)
   ELSIF (_q0 OR _q2) AND _A>0 THEN A:=_A-1; END_IF; (* A- *)
   IF    _q3           THEN B:=_B+1;                   (* B+ *)
   ELSIF _q5 AND _B>0 THEN B:=_B-1; END_IF;           (* B- *)
   IF    (_q1 OR _q4) THEN C:=_C+1; END_IF;           (* C+ *)
   (* Состояния *)
   IF NOT _q0 AND _q5 AND NOT(_B>0) THEN q0:=1;                   (* q0+ *)
   ELSIF  _q0                       THEN q0:=0; END_IF;           (* q0- *)
   IF NOT _q1 AND (_q0 AND _A>0 OR _q4) THEN q1:=1;               (* q1+ *)
   ELSIF  _q1                           THEN q1:=0; END_IF;       (* q1- *)
   IF NOT _q2 AND _q1 THEN q2:=1;                                 (* q2+ *)
   ELSIF  _q2         THEN q2:=0; END_IF;                         (* q2- *)
   IF NOT _q3 AND _q2 AND _A>0 THEN q3:=1;                        (* q3+ *)
   ELSIF  _q3                  THEN q3:=0; END_IF;                (* q3- *)
   IF NOT _q4 AND _q3 THEN q4:=1;                                 (* q4+ *)
   ELSIF  _q4         THEN q4:=0; END_IF;                         (* q4- *)
   IF NOT _q5 AND (_q2 AND NOT(_A>0) OR _q6) THEN q5:=1;          (* q5+ *)
   ELSIF  _q5                                THEN q5:=0; END_IF; (* q5- *)
   IF NOT _q6 AND _q5 AND _B>0 THEN q6:=1;                        (* q6+ *)
   ELSIF  _q6                  THEN q6:=0; END_IF;                (* q6- *)
   IF NOT _q7 AND _q0 AND NOT(_A>0) THEN q7:=1; END_IF;           (* q7+ *)
   (* Псевдооператорный раздел *)
   _A:=A; _B:=B; _C:=C;
   _q0:=q0; _q1:=q1; _q2:=q2; _q3:=q3; _q4:=q4; _q5:=q5; _q6:=q6; _q7:=q7.
```

Рассмотрим теперь реализацию счетчиковой машины $3cM$ на упрощённом SFC также по описанной ранее LTL-спецификации. Основная и вложенные SFC-схемы представлены ниже (по порядку). Действию шага AStep соответствует программная реализация пары LTL-формул спецификации поведения переменной-счетчика A. Например, действием этого шага может быть взят IF-ELSIF блок на языке ST, помеченный парой A+ и A−. Аналогичным образом действиям шагов BStep и CStep сопоставляются программные реализации пар LTL-формул спецификации поведения переменных-счетчиков B и C соответственно. Действием шага PsvdStp является

псевдооператорный раздел. Действию шага qStep соответствует вложенная SFC-схема. Во вложенной SFC-схеме текущим и выходным действиями шага, имя которого начинается с префикса `qi`, является программная реализация (на IL, ST, FBD/CFC или LD) LTL-формул с метками `qi+` и `qi−`. Вложенная SFC-схема повторяет графическое представление правил перехода машины $3cM$.



Список литературы / References

[1] Кузьмин Е. В., Рябухин Д. А., Соколов В. А., "Моделирование согласованного поведения ПЛК-датчиков", *Моделирование и анализ информационных систем*, **21**:4 (2014), 75–90; [Kuzmin E. V., Ryabukhin D. A., Sokolov V. A., "Modeling a Consistent Behavior of PLC-Sensors", *Modeling and Analysis of Information Systems*, **21**:4 (2014), 75–90, (in Russian).]

[2] Рябухин Д. А., Кузьмин Е. В., Соколов В. А., "Построение IL-программ ПЛК по LTL-спецификации", *Моделирование и анализ информационных систем*, **21**:2 (2014), 26–38; [Ryabukhin D. A., Kuzmin E. V., Sokolov V. A., "Construction of PLC IL-programs by LTL-specification", *Modeling and Analysis of Information Systems*, **21**:2 (2014), 26–38, (in Russian).]

[3] Кузьмин Е. В., Соколов В. А., Рябухин Д. А., "Построение и верификация LD-программ ПЛК по LTL-спецификации", *Моделирование и анализ информационных систем*, **20**:6 (2013), 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*, **20**:6 (2013), 78–94, (in Russian).]

[4] Кузьмин Е. В., Соколов В. А., Рябухин Д. А., "Построение и верификация ПЛК-программ по LTL-спецификации", *Моделирование и анализ информационных систем*, **20**:4 (2013), 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*, **20**:4 (2013), 5–22, (in Russian).]

[5] Кузьмин Е. В., Соколов В. А., "Моделирование, спецификация и построение программ логических контроллеров", *Моделирование и анализ информационных систем*, **20**:2 (2013), 104–120; [Kuzmin E. V., Sokolov V. A., "Modeling, Specification and Construction of PLC-programs", *Modeling and Analysis of Information Systems*, **20**:2 (2013), 104–120, (in Russian).]

[6] Минский М., *Вычисления и автоматы*, М.: Мир, 1971; [Minsky M., *Computation: Finite and Infinite Machines*, Prentice-Hall, Inc., 1967.]

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

[8] Baier C., Katoen J.-P., *Principles of Model Checking*, The MIT Press, 2008.

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

[10] CoDeSys, *Controller Development System*, http://www.3s-software.com/.

[11] Kuzmin E. V., Sokolov V. A., Ryabukhin D. A., "Construction and Verification of PLC LD Programs by the LTL Specification", *Automatic Control and Computer Sciences*, **48**:7 (2014), 424–436.

[12] Kuzmin E. V., Sokolov V. A., "Modeling, Specification and Construction of PLC-programs", *Automatic Control and Computer Sciences*, **48**:7 (2014), 554–563.

[13] Kuzmin E. V., Ryabukhin D. A., Sokolov V. A., "Modeling a Consistent Behavior of PLC-Sensors", *Automatic Control and Computer Sciences*, **48**:7 (2014), 602–614.

[14] Parr E. A., *Programmable Controllers. An engineer's guide*, Newnes, 2003.

[15] Schroeppel R., *A Two Counter Machine Cannot Calculate* 2^N, Memo 257. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1972.

[16] SMV, *The Cadence SMV Model Checker*, http://www.kenmcmil.com/smv.html.

DOI: 10.18255/1818-1015-2015-4-507-520

On the Expressiveness of the Approach to Constructing PLC-programs by LTL-Specification

Kuzmin E. V.[1], Ryabukhin D. A., Sokolov V. A.

Received August 3, 2015

The article is devoted to the approach to constructing and verification of discrete PLC-programs by LTL-specification. This approach provides an ability of correctness analysis of PLC-programs by the model checking method. The linear temporal logic LTL is used as a language of specification of the program behavior. The correctness analysis of LTL-specification is automatically performed by the symbolic model checking tool Cadence SMV.

The article demonstrates the consistency of the approach to constructing and verification of PLC programs by LTL-specification from the point of view of Turing power. It is proved, that in accordance with this approach for any Minsky counter machine can be built an LTL-specification, which is used for machine implementation in any PLC programming language of standard IEC 61131-3. Minsky machines equipollent Turing machines, and the considered approach also has Turing power.

The proof focuses on representation of a counter machine behavior in the form of a set of LTL-formulas and matching these formulas to constructions of ST and SFC languages. SFC is interesting as a specific graphical language. ST is considered as a basic language because an implementation of a counter machine in IL, FBD/CFC and LD languages is reduced to rewriting blocks of ST-program.

The idea of the proof is demonstrated by an example of a Minsky 3-counter machine, which implements a function of squaring.

Keywords: programmable logic controllers (PLC), construction and verification of PLC-programs, LTL-specification, Minsky counter machines

For citation: Kuzmin E. V., Ryabukhin D. A., Sokolov V. A., "On the Expressiveness of the Approach to Constructing PLC-programs by LTL-Specification", *Modeling and Analysis of Information Systems*, **22**:4 (2015), 507–520.

On the authors:
Kuzmin Egor Vladimirovich, orcid.org/0000-0003-0500-306X, doctor of science, associate professor,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: kuzmin@uniyar.ac.ru

Ryabukhin Dmitriy Aleksandrovich, orcid.org/0000-0002-0799-8868, graduate student,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: dmitriy_ryabukhin@mail.ru

Sokolov Valery Anatolievich, orcid.org/0000-0003-1427-4937, doctor of science, professor,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: sokolov@uniyar.ac.ru

Acknowledgments:

[1]This work was supported by the Russian Foundation for Basic Research (RFBR), №12-01-00281-a.