Модел. и анализ информ. систем. Т. **19**, № **2** (2012) 138–144

УДК 519.7

О верификации LD-программ логических контроллеров

Кузьмин Е. В., Соколов В. А. [1]

Ярославский государственный университет им. П.Г. Демидова

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

получена 5 апреля 2012 года

Ключевые слова: верификация, проверка модели, программы логических контроллеров, LD-диаграммы

Обсуждаются вопросы построения технологии анализа корректности программ логических контроллеров. Рассматривается пример моделирования и верификации «дискретных» LD-программ с таймером с помощью программного средства символьной проверки модели SMV при спецификации свойств на языке темпоральной логики линейного времени LTL.

1. 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) [6], 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.

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

Языки программирования логических контроллеров определяются стандартом МЭК 61131-3. Этот стандарт включает в себя описание пяти языков программирования: SFC, IL, ST, LD и FBD. Язык IL (Instruction list) дословно — список инструкций. Это типичный ассемблер с аккумулятором и переходами по меткам. Язык ST (Structured Text) — язык высокого уровня, синтаксически представляющий собой несколько адаптированный язык Паскаль. Язык релейных диаграмм LD (Ladder Diagram) или релейно-контактных схем (РКС) — графический язык, реализующий структуры электрических цепей. FBD (Function Block Diagram) – графический язык программирования, диаграммы в рамках которого очень напоминают принципиальную схему электронного устройства 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. В связи с этим, одним из естественных направлений исследований в указанной области является построение трансляторов со стандартных языков МЭК 61131-3 ПЛК в интерфейсные языки программных средств верификации методом проверки модели (model checking) таких, как SPIN и SMV [9], а также наработка удобных моделей программ ПЛК и шаблонов свойств программ ПЛК на языках темпоральных логик LTL и CTL, использующихся в качестве языков спецификации свойств [5, 7, 8].

В этой статье рассматривается пример автоматической верификации «дискретных» LD-программ с таймером (как неотъемлемым элементом большинства программ ПЛК) с помощью программного средства символьной проверки модели SMV при спецификации свойств на языке темпоральной логики линейного времени LTL.

Целью работы по данной тематике является создание теоретической основы для построения программного комплекса моделирования, спецификации и верификации программ логических контроллеров.

2. Моделирование и верификация LD-программ

Задача проверки модели (для LTL) состоит в определении выполнимости для процесса, который задается системой переходов (структурой Крипке), свойства, выраженного формулой темпоральной логики LTL.

Структурой Крипке над множеством элементарных высказываний P называется система переходов $\mathcal{S} = (S, s_0, \rightarrow, L)$, где S – конечное множество состояний, $s_0 \in S$ – 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 $\pi = s_0 s_1 s_2 \ldots$ такая, что для всех $i \geq 0$ выполняется $s_i \rightarrow s_{i+1}$.

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

$$\varphi ::= \mathit{true} \mid \ p \mid \neg\varphi \mid \varphi \wedge \varphi \mid X\varphi \mid \varphi\, U\, \varphi \mid F\,\varphi \mid G\,\varphi.$$

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 выполняется в текущем или будущем состоянии при том, что во всех состояниях (начиная с текущего) до этого момента должна выполняться формула ψ. Структура Крипке удовлетворяет формуле (свойству) φ логики LTL, если φ выполняется для всех путей, выходящих из начального состояния s_0.

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 [4, 3]. 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. 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 широким кругом специалистов.

На рис. 1 представлен пример ПЛК-программы управления взаимоисключающим доступом двух устройств к третьему (общему ресурсу), написанной на языке диаграмм LD, которые имеют вид релейно-контактных электрических схем. Программа состоит из цепей, оказывающих влияние на состояния реле, которыми они заканчиваются. Установленное логическое состояние реле может учитываться далее в программе как в текущем рабочем цикле, так и в следующем. Программа выполняется слева направо и сверху вниз. Входные сигналы ПЛК представлены контактами In1 и In2 (запросы устройств на доступ к общему ресурсу), выходные – контактами

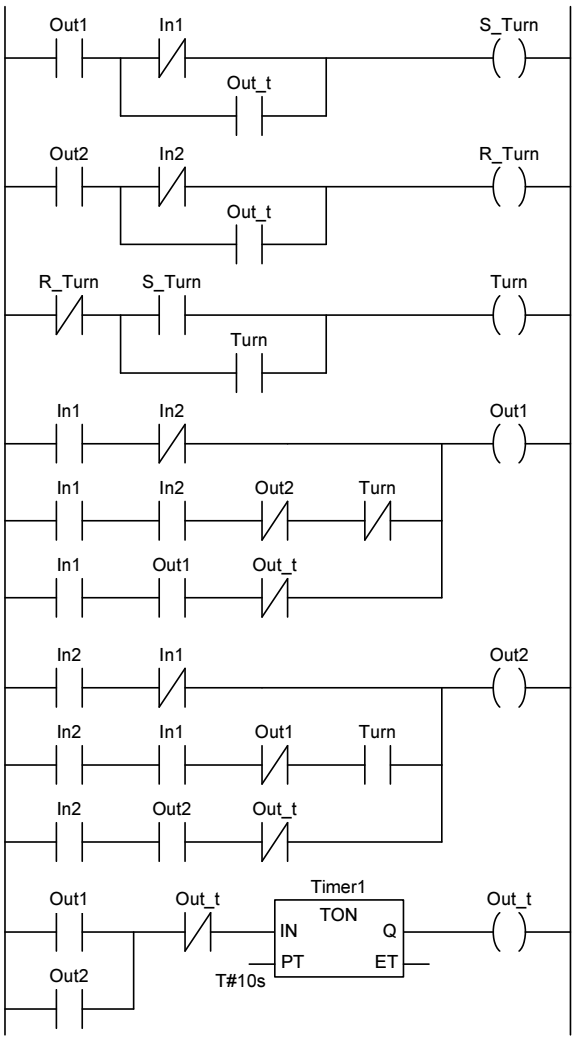

Рис. 1. LD-диаграмма управления взаимоисключающим доступом

реле Out1, Out2 (сигналы, разрешающие/запрещающие доступ устройств к общему ресурсу). Turn, S_Turn, R_Turn и Out_t – внутренние программные переменные-реле. Timer1 – стандартный функциональный блок, реализующий таймер. Таймер запускается, когда сигнал IN меняется с 0 на 1 и остается активным до тех пор, пока IN=1. По истечении 10 *с* таймер выставляет сигнал Q=1 (до этого момента Q=0) и удерживает его, пока остается активным. С помощью переменной ET отслеживается время, прошедшее с момента последнего запуска таймера. Если вход IN устанавливается в 0, таймер сбрасывается в начальное положение, перестает быть активным, выход Q также устанавливается в 0. С помощью таймера Timer1 единовременный сеанс работы устройств с общим ресурсом ограничивается до 10 с.

Построим для LD-программы модель в виде структуры Крипке. За состояние модели возьмем вектор значений всех программных переменных, который условно можно разделить на две части. Первая часть – вектор значений входов на момент начала нового рабочего цикла ПЛК. Вторая часть представляет собой вектор зна-

чений 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 полного прохода рабочего цикла. Таким образом, переход из одного состояния в другое зависит от (прежних) значений выходов и внутренних переменных первого состояния и (новых) значений входов второго состояния. Для каждого состояния степень ветвления отношения переходов определяется числом всех возможных комбинаций входных сигналов ПЛК.

Ниже приведена модель для LD-программы с рис. 1, описанная на языке верификатора SMV [9], которая была построена в соответствии с предложенной структурой Крипке и с учетом направления выполнения (порядка срабатывания элементов) LD-программ. Средства языка SMV позволяют с помощью оператора next задавать значение переменной в следующем состоянии модели, т. е. на следующем шаге (после разового прохождения рабочего цикла ПЛК), относительно новых входных сигналов и последних значений выходов и внутренних переменных. Ветвление отношения переходов обеспечивается «недетерминированным» присваиванием. Например, последовательность команд next(in1) := {0, 1}; next(in2) := {0, 1} означает, что будут порождены (если таких состояний ещё не было) четыре состояния и переходы в них (из текущего состояния) с различным сочетанием значений входных переменных in1 и in2. На языке SMV символы «&», «|», «∼» и «->» означают соответственно логические «и», «или», «не» и импликацию.

```
module main() { /*описание переменных*/
    in1, in2, out1, out2, turn: 0..1;
    r_turn, s_turn, out_t, timer1: 0..1;
    /* инициализация */
    init(in1) := 0; init(in2) := 0; init(out1) := 0; init(out2) := 0;
    init(turn) := 0; init(s_turn) := 0; init(r_turn) := 0;
    init(out_t) := 0; init(timer1) := 0;
    /* описание системы переходов */
    next(in1) := {0, 1}; /* недетерминированное присваивание */
    next(in2) := {0, 1};
    next(s_turn) := out1 & (∼next(in1) | out_t);
    next(r_turn) := out2 & (∼next(in2) | out_t);
    next(turn) := (next(s_turn) | turn) & ∼next(r_turn);
    next(out1) := next(in1) & ∼next(in2) |
                  next(in1) & next(in2) & ∼out2 & ∼next(turn) |
                  next(in1) & out1 & ∼out_t;
    next(out2) := next(in2) & ∼next(in1) |
                  next(in2) & next(in1) & ∼out1 & next(turn) |
                  next(in2) & out2 & ∼out_t;
    next(timer1) := (next(out1) | next(out2)) & ∼out_t;
    next(out_t) := case { next(timer1) & ∼out_t : {0, 1};
                          next(timer1) &    out_t : 1;
                          default : 0; };
    FAIRNESS timer1 -> out_t; /* условие справедливости */
```

```
mutex: assert G(∼( out1 & out2 )); /* описание темпоральных LTL-свойств */
access : assert G((G in1) -> F out1) & G((G in2) -> F out2);
fairplay : assert
        G((X in1) & (X in2) & ∼turn & ∼out1 & ∼out2 -> X out1) &
        G((X in1) & (X in2) &    turn & ∼out1 & ∼out2 -> X out2);
no_alternation: assert
        G(in1 & ∼in2 -> out1) &
        G(in2 & ∼in1 -> out2);                                          }
```

Здесь работа таймера моделируется таким образом, что если таймер выключен (timer1=0), то на выходе он выставит сигнал out_t=0, если он является активным (timer1=1) и уже сработал, т. е. на предыдущем рабочем цикле выход out_t был установлен в 1, то этот сигнал out_t=1 сохраняется, а если таймер является активным и ещё не сработал, т. е. на предыдущем рабочем цикле выход out_t=0, то выход out_t устанавливается либо в 1, что означает срабатывание таймера по истечении заданного времени, либо сохраняется на выходе out_t=0, т. е. таймер продолжает работу. Последнее условие реализуется недетерминированным присваиванием. Но это приводит к тому, что в модели Крипке появляются такие пути, на которых однажды запущенный таймер так никогда и не сработает (при построении переходов в следующие состояния одна из веток всегда будет вести к состоянию, в котором таймер запущен, но ещё не сработал). Такие пути называются «несправедливыми», или «нечестными», и исключаются из модели Крипке с помощью конструкции FAIRNESS timer1->out_t, которая требует, чтобы на каждом пути условие timer1->out_t выполнялось бесконечно часто. Пути, которые не удовлетворяют этой конструкции, не включаются в модель Крипке и не рассматриваются. В целом эта конструкция означает, что если таймер был запущен, то он рано или поздно должен сработать.

Темпоральные LTL-свойства предваряются ключевым словом assert. Свойство mutex выражает то, что в модели Крипке нет состояния, при котором двум устройствам разрешен одновременный доступ к общему ресурсу. Свойство access означает, что если одно из устройств запросит доступ к общему ресурсу, то оно рано или поздно его получит. Свойство fairplay описывает, что при одновременном запросе доступ к общему ресурсу получит то устройство, чей приоритет выше, т. е. чья сейчас очередь. По алгоритму после того, как устройство провело сеанс работы с общим ресурсом, приоритет доступа передается второму устройству. А свойство no_alternation говорит о том, что устройству нет необходимости ждать своей очереди на доступ, т. е. доступ от приоритета не зависит, если второе устройство в доступе не нуждается. Проверка с помощью верификатора SMV показала выполнимость всех этих свойств для построенной модели приведенной LD-диаграммы.

Список литературы

1. *Грис Д.* Наука программирования. Пер. с англ. М.: Мир, 1984. 416 с.

2. *Кларк Э. М.*, *Грамберг О.*, *Пелед Д.* Верификация моделей программ: Model Checking. Пер. с англ. М.: МЦНМО, 2002. 416 с.

3. *Парр Э.* Программируемые контроллеры: руководство для инженера. М.: БИНОМ. Лаборатория знаний, 2007. 516 с.

4. *Петров И. В.* Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования. М.: СОЛОН-Пресс, 2004. 256 с.

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

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

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

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

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

On Verification of PLC-Programs Written in the LD-Language

Kuzmin E. V., Sokolov V. A.

Keywords: verification, model checking, PLC-programs, Ladder Diagrams

We discuss some questions connected with the construction of a technology of analysing correctness of Programmable Logic Controller programs. We consider an example of modeling and automated verification of PLC-programs written in the Ladder Diagram language (including timed function blocks) of the IEC 61131-3 standard. We use the Cadence SMV for symbolic model checking. Program properties are written in the linear-time temporal logic LTL.

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