МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра Компьютерных технологий

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Кондратьева Ильи Игоревича

Тема работы:

Разработка транслятора языка poST в язык Promela для целей верификации методом проверки моделей

«К защите допущена»

Заведующий кафедрой КТ,
д.т.н., доцент
Зюбин В.Е. / ……..
(ФИО) / (подпись)
«31» мая 2022 г.

Руководитель ВКР

к.ф-м.н., доцент
кафедры КТ ФИТ НГУ
Гаранина Н.О. / ……...
(ФИО) / (подпись)
«31» мая 2022 г.

Соруководитель ВКР

к.т.н., доцент
кафедры КТ ФИТ НГУ
Розов А.С. / ……..
(ФИ О) / (подпись)
«31» мая 2022 г.

Новосибирск, 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра Компьютерных технологий

Направление подготовки 09.03.01 Информатика и вычислительная техника

Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ
Зав. кафедрой, Зюбин В.Е.
…….
(подпись)
«29» октября 2022 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Кондратьеву Илье Игоревичу, группы 18203

Тема Разработка транслятора языка poST в язык Promela для целей верификации методом проверки моделей

утверждена распоряжением проректора по учебной работе от 29 октября 2021 г. № 0297

Срок сдачи студентом готовой работы 20 мая 2022 г.

Исходные данные (или цель работы): Разработать транслятор процесс-ориентированных программ на языке poST в язык Promela

Структурные части работы: анализ языков poST и Promela и формирование требований к правилам трансляции и транслятору, разработка правил трансляции, реализация транслятора, проверка транслятора на наборе тестовых задач.

Руководитель ВКР

к.ф-м.н, доцент

кафедры КТ НГУ

Гаранина Н.О. / ……..
(ФИ О) / (подпись)

«29» октября 2022 г.

Задание принял к исполнению

Кондратьев И.И. / ……..
(ФИО студента) / (подпись)

«29» октября 2022 г..

Соруководитель ВКР

Старший преподаватель

кафедры КТ ФИТ НГУ

Розов А.С. / ……..
(ФИ О) / (подпись)

«29» октября 2022 г..

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ

СО РАН – сибирское отделение Российской академии наук

КФС – кибер-физическая система

poST – process-oriented Structured Text

Promela – Process Meta Language

SPIN – Simple Promela Interpreter

AST – Abstract Syntax Tree

LTL – Linear Temporal Logic

ВВЕДЕНИЕ

Задача автоматизации промышленных систем актуальна в течение долгого времени, при этом сложность этих систем возрастает. Использование специализированных языков программирования автоматических систем управления позволяет упростить написание кода и улучшить его читаемость, что приводит к сокращению количества ошибок и уменьшению стоимости разработки. Одним из таких языков программирования является процесс-ориентированный язык poST, разработанный в Институте Автоматики и Электрометрии СО РАН специально для написания на нем управляющих программ для киберфизических систем (КФС) – то есть систем, активно взаимодействующих с окружающей средой.

Кроме того, одной из важнейших задач, особенно в данной области, является формальная верификация программного обеспечения, для выполнения которой разработано множество различных методов. Одним из основных подходов является метод проверки моделей (model checking) – при его применении происходит строгое доказательство соответствия проверяемой системы предъявленным к ней требованиям. Это достигается тем, что специфицированные свойства проверяются во всех достижимых состояниях системы. Одним из широко используемых инструментов для автоматической проверки моделей является верификатор SPIN [1], входным языком которого является Promela.

Цель данной дипломной работы: разработка транслятора программ на языке poST в язык Promela для целей последующей верификации методом проверки моделей с помощью инструмента SPIN.

Для достижения этой цели были поставлены следующие задачи:

1. Анализ специфики языков poST и Promela.
2. Определение требований к правилам трансляции и транслятору.
3. Разработка правил трансляции конструкций poST в Promela.
4. Реализация транслятора.
5. Проверка работы транслятора на наборе тестовых задач.

Новизна: в работе предложены правила трансляции конструкций языка poST в конструкции языка Promela, метод проверки свойств автоматических систем управления, основанный на их цикличности. Была подтверждена корректность транслятора на наборе тестовых задач с использованием верификатора SPIN.

Практическая ценность: в результате работы был разработан транслятор, который позволяет упростить верификацию управляющих программ на языке poST с использованием инструмента SPIN.

Структура работы:

Работа изложена в четырех главах. В первой главе происходит анализ языков poST и Promela и определяются требования к правилам трансляции и транслятору. Во второй главе предлагаются правила трансляции из языка poST в язык Promela. Третья глава посвящена созданию транслятора. В четвертой главе описана апробация транслятора на наборе тестовых задач с использованием инструмента SPIN.

1 Анализ poST и Promela

1.1 Синтаксис и семантика poST

Язык poST - процесс-ориентированный, то есть программа на нем состоит из блоков, представляющих собой множества взаимодействующих процессов, каждый из которых в каждый момент времени находится в определенном состоянии. Процессы вызываются циклически друг за другом в порядке объявления.

1.1.1 Структура файла

Файл на языке poST состоит из блоков PROGRAM, которые могут отвечать за разные компоненты автоматизируемой системы. Также в файле может присутствовать конфигурация (CONFIGURATION), в которой, в частности, можно задать временной интервал (INTERVAL) вызова цикла активации процессов.

Пример структуры файла poST:

```
CONFIGURATION testConfig
        RESOURCE testResource ON TestCPU
                TASK PromelaVerificationTask (INTERVAL := T#100ms);
        END_RESOURCE
END_CONFIGURATION

PROGRAM User
  …
END_PROGRAM

PROGRAM Sanitizer
  …
END_PROGRAM
```

Листинг 1 – Структура файла poST

Файлу, приведенному в листинге 1, соответствует поведение системы, при котором каждые 100 миллисекунд запускается итерация цикла активации программ в последовательности: User, Sanitizer.

1.1.2 Структура программы

Программа (PROGRAM) на языке poST представляет собой множество процессов (PROCESS), которые исполняются последовательно во время исполнения этой программы. Кроме того, программа может включать в себя переменные, объявленные в блоках VAR, VAR

CONSTANT, VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT, область видимости которых – данная программа.

Пример структуры программы:

```
PROGRAM Sanitizer
  VAR CONSTANT
    ONN : BOOL := TRUE;
    OFF : BOOL := FALSE;
    HANDS_OFF_TIMEOUT : TIME := T#550ms;
    SPRAY_DURATION : TIME := T#5s;
    DELAY_BETWEEN_SPRAYS : TIME := T#2s;
  END_VAR
  VAR_INPUT
    hands : BOOL;
  END_VAR
  VAR_OUTPUT
    pump : BOOL;
  END_VAR

  PROCESS Controller
    …
  END_PROCESS

  PROCESS DurationControl
    …
  END_PROCESS
END_PROGRAM
```

Листинг 2 – Структура программы poST

При передаче управления данной программе процессы (PROCESS) будут вызваны по одному разу в следующей последовательности: Controller, DurationControl.

1.1.3 Процессы и состояния

У каждого процесса (PROCESS) есть множество состояний (STATE) – последовательностей действий, одна из которых текущая и исполняется один раз при очередном вызове данного процесса. Есть специальные состояния в которых процесс неактивен: STOP и ERROR. Текущее состояние первого процесса в программе (PROGRAM) при первом цикле активации – первое объявленное, для остальных процессов – STOP. Как и в программе, в процессе могут быть объявлены переменные, область видимости которых – данный процесс.

Пример процесса:

```
PROCESS Controller
   STATE Init
      pump := OFF;
      SET NEXT;
   END_STATE

   STATE Idle
      IF hands THEN
         pump := ONN;
         START PROCESS DurationControl;
         SET NEXT;
      END_IF
   END_STATE

   STATE Spray
      IF hands THEN
         RESET TIMER;
      END_IF
      TIMEOUT HANDS_OFF_TIMEOUT THEN
         pump := OFF;
         SET STATE Idle;
      END_TIMEOUT
   END_STATE
END+PROCESS
```

Листинг 3 – Пример процесса poST

При каждой итерации цикла вызова процессов будет вызываться код только одного из состояний данного процесса. Например, при первой итерации будет исполнен код состояния Init, второй инструкцией которого является “SET NEXT” – это значит, что на следующей итерации будет исполнен код следующего состояния, то есть Idle.

Существуют следующие способы изменения состояния процесса:

- START PROCESS p1 – запуск процесса p1 – при следующей итерации его состояние будет первым объявленным.
- STOP PROCESS p1 / ERROR PROCESS p1 – нормальная или “аварийная” остановка процесса p1 – состояние изменено на STOP / ERROR соответственно.
- RESTART – переход текущего процесса в первое объявленное состояние.
- STOP / ERROR – остановка текущего процесса.
- SET STATE s1 – установка состояния текущего процесса в s1.
- SET NEXT – установка состояния текущего процесса в следующее. Запрещено для последнего состояния процесса.

Представленные способы следуют принципу инкапсуляции, то есть текущее состояние некоторого процесса недоступно другим процессам.

Состояния могут содержать блоки таймаутов (TIMEOUT), исполнение которых начинается спустя указанное время после перехода процесса в данной состояние.

1.2 Верификация poST программ

1.2.1 Проверка моделей

Проверка моделей – способ верификации системы, при котором происходит проверка свойств системы во всех ее достижимых состояниях. Свойства системы задаются на языке модальной, в частности, темпоральной логики. Язык, на котором описывается модель системы, зависит от инструмента, используемого для верификации – часто это языки похожие на C, Java, Verilog или VHDL [2].

При обнаружении состояния, в котором проверяемые свойства нарушаются, верификатор моделей предоставляет этот контрпример, представляющий собой путь достижения нарушающего состояния из начального. Если же во время прохода по всем достижимым состояниям верификатор не обнаружил контрпримера, то это служит доказательством того, что программа соответствует спецификации.

Метод проверки моделей используется для проверки критических систем, например, с его помощью были обнаружены 5 неизвестных ранее ошибок в контроллере для Deep Space 1 [2]. К сожалению, для применения метода верификации моделей зачастую требуется больше ресурсов, чем на разработку самой системы, что затрудняет использование данной технологии.

1.2.2 SPIN

SPIN – пакет программ для формальной верификации методом проверки моделей, разработанный в исследовательском центре Bell Labs. Применяется в промышленности и для обучения принципам формальной верификации [3].

SPIN предназначен для верификации моделей многопоточных программных систем, описанных на языке Promela. Находясь в режиме верификации, SPIN проверяет состояния системы, пытаясь найти то, которое не соответствует спецификации. При нахождении такого состояния, пользователь может проследить путь до него в режиме симуляции.

К преимуществам использования SPIN можно отнести следующее:

- SPIN свободно распространяемый.
- SPIN поддерживает многопоточную верификацию.

- SPIN не строит граф всех состояний заранее, что снижает требования к объему памяти [4].
- Promela имеет структуру похожую на poST.
- Promela C-подобный.

Таким образом целесообразно использовать SPIN в качестве инструмента для верификации poST-программ.

1.2.3 Promela

Программа на языке Promela, как и на poST, представляет собой множество взаимодействующих процессов. Кроме того, Promela C-подобный – эти особенности упрощают изучение языка и разработку правил трансляции.

Но в отличие от poST, активные процессы в Promela запускаются недетерминированно, кроме того в Promela нет понятия цикла активации – процесс завершается, когда завершается исполнение его тела. Это учтено при разработке правил трансляции.

Основу программы составляют процессы. Кроме того, как и в языке C, в файле Promela-программы могут присутствовать директивы препроцессора, например, #define. Также доступны объявления глобальных и локальных переменных [5].

Пример Promela-программы:

```
#define d 17
byte b;

init {
   run p1();
}

proctype p1() {
   b = 3;
}

active proctype p2() {
   unsigned v : 3 = 7;
   printf("DEBUG: v == %d\n", v);
}
```

Листинг 4 – Пример Promela-программы

Изначально активные процессы – те, которые помечены как active и процесс init. Здесь сначала запускаются процессы init и p2. В процессе исполнения init запускает p1 (run p1();).

1.2.4 Требования к транслятору

Исходя из специфики использования языка poST, были определены следующие требования к правилам трансляции:

- Функциональные требования:
  - Поддержка программ, процессов, состояний (в т.ч. состояний с таймаутами).
  - Поддержка числовых типов данных.
  - Поддержка массивов числовых типов.
  - Поддержка констант типа TIME.
  - Поддержка операций над переменными, кроме возведения в степень.
  - Поддержка операторов poST: циклы, условный, ветвления, изменения состояния процесса, проверки состояния процесса.
  - Возможность взаимодействия программ (PROGRAM) посредством передачи значений между INPUT, OUTPUT и IN_OUT переменными.
  - Возможность определения INPUT переменных с недетерминированными значениями.
- Нефункциональные требования:
  - Уменьшение сложности верификации программ с таймаутами и константами типа TIME.
  - Читаемость генерируемого кода на Promela (форматирование, имена сущностей, выделение разделов файла модели, и т.п.).

Транслятор в дальнейшем будет являться частью модуля расширения poST IDE, что влияет на выбор стека используемых технологий и формата реализации решения:

- Фреймворк разработки – Xtext, язык – Xtend.

  Они предназначены и удобны для использования в разработке специализированных языков. Кроме этого, poST IDE разрабатывается также с использованием этих технологий – их использование для разработки модулей расширения упростит дальнейшую поддержку системы, так как будет требовать меньше компетенций от разработчиков.

- Возможность формирования файла модели на Promela из файла на poST, путем запуска исполняемого jar-файла с указанием путей до входного и выходного файла.

2 Трансляция poST в Promela

В данной главе представлены разработанные правила трансляции конструкций poST [6] в Promela [7].

2.1 Трансляция имен

В poST существуют следующие пространства имен: глобальное, для каждой программы, для каждого процесса. В Promela существует только глобальное пространство имен и пространства имен для каждого процесса. Таким образом, так как в Promela уровней пространств имен меньше, чем в poST, при тривиальной трансляции имен сущностей могут возникать конфликты. Чтобы их избежать, имена для Promela-программы в большинстве случаев формируются в соответствии со следующим шаблоном:

<type>__<ns[0]>_<ns[1]>_…_<ns[m]>__<name>, где

type – служит для обозначения вида сущности, например “process” для процесса,
ns[i] – префикс имени программы или процесса, в котором объявлена сущность,
name – имя сущности в poST-программе.

Шаблон 1 – Имя для Promela-программы

После получения таких полных имен для всех сущностей, имена транслируются в более короткие в соответствии со следующим шаблоном:

<name>__<type><#> или
<name>, если <type><#> - пустая строка, где

name – имя сущности в poST-программе,
type – служит для обозначения вида сущности,
– счетчик, если переменных с данной комбинацией name и prefix несколько.

Шаблон 2 – Короткое имя для Promela-программы

2.2 Переменные

В poST переменные можно объявлять в программах (PROGRAM) и процессах (PROCESS), помечая их как:

- VAR,
- VAR CONSTANT,
- VAR_INPUT,
- VAR_OUTPUT,
- VAR_IN_OUT.

При трансляции переменные-константы преобразуются в использования директивы #define препроцессора, остальные переменные – в обычные глобальные переменные Promela.

Перевод типов данных осуществляется в соответствии со следующей таблицей (“-” – нет аналога в Promela, если при этом указан тип в скобках - то переменная транслируется в переменную этого типа с предупреждением):

Тип данных		**poST**	**Promela**
Целочисленные знаковые	8 бит	SINT	- (short)
	16 бит	INT	short
	32 бита	DINT	int
	64 бита	LINT	- (int)
Целочисленные беззнаковые	8 бит	USINT	byte
	16 бит	UINT	unsigned : 16
	32 бита	UDINT	- (int)
	64 бита	ULINT	- (int)
Вещественные	32 бита	REAL	- (int)
	64 бита	LREAL	- (int)
Битовые строки	1 бит (логическая)	BOOL	bool
	8 бит	BYTE	byte
	16 бит	WORD	short
	32 бита	DWORD	int
	64 бита	LWORD	- (int)

Время	TIME	unsigned : <n> (*)
Строки	STRING, WSTRING	-
Массивы	ARRAY [x..y] OF t	[] (**)

Таблица 1 – Трансляция типов данных

*Определение числа битов n при трансляции типа TIME освещается при рассмотрении трансляции таймаутов.

**Любые массивы транслируются в соответствии с правилами трансляции примитивных типов, за исключением: TIME – не транслируются, SINT – транслируются в byte[].

Таким образом, большинство типов транслируются тривиально, а присутствия в Promela аналогов переменных большого размера не требуется, так как использование таких переменных сильно увеличивает число состояний системы и затрудняет верификацию.

2.3 Операции

Операции в poST и Promela также во многом совпадают, способ трансляции представлен в таблице:

Операции		**poST**	**Promela**
Логические	Логическое исключающее ИЛИ	XOR	^
	Логическое ИЛИ	OR	||
	Логическое И	AND	&
Сравнения	Проверка на равенство / неравенство	=	==
		<>	!=
	Проверка на меньше / больше	<	<
		>	>
	Проверка на меньше или равно / больше или равно	<=	<=
		>=	>=
Арифметические	Сложение	+	+
	Вычитание	-	-

	Умножение	*	*
	Деление	/	/
	Остаток от деления	MOD	%
	Возведение в степень	**	Нет аналога
Унарные	Логическое отрицание	NOT	!
	Взятие противоположного	-	-

Таблица 2 – Трансляция операций

2.4 Выражения

В этом пункте описывается трансляция в Promela выражений языка poST, таких как объявления переменных, условные операторы, процессы, состояния, таймауты, и т.д.

2.4.1 Объявление переменной

Для трансляции констант используется директива #define, для остальных переменных – объявление глобальной переменной. Пример трансляции приведен ниже:

Вид переменной	**poST**	**Promela**
Константа	VAR CONSTANT v : INT := 17; END_VAR	#define v 17
Обычная переменная	VAR v : INT := 17; END_VAR	int v = 17;

Таблица 3 – Трансляция объявления переменной

2.4.2 Условный оператор

В отличие от poST условный оператор в Promela блокирующийся и имеет следующий синтаксис:

```
if
   :: <cond[0]> -> {...}
```

```
   …
   :: <cond[m]> -> {...}
fi;

cond[i] – логическое выражение или “else”
```

Шаблон 3 – Синтаксис условного оператора в Promela

Promela-процесс будет заблокирован, до тех пор пока не будет выполнено хотя бы одно из условий cond[i]. При истинности условия будут выполнены инструкции из соответствующего блока {}, после чего исполнение процесса продолжится с места после окончания оператора if. При истинности нескольких условий выбор блока происходит недетерминированно – то есть каждое истинное условие порождает новую ветку возможных состояний. Условие “else” не обязательно и оно истинно, если все остальные условия ложны. Оператор “skip” позволяет выйти из ветки if.

В связи с данными особенностями условного оператора в Promela предлагается следующий способ трансляции:

poST	**Promela**
IF без ELSE	
IF <cond> THEN <body> END_IF	if :: <cond> -> { <body> } :: else -> skip; fi;
IF с ELSE	

IF <cond> THEN
  <body1>
ELSE
  <body2>
END_IF
	if
	  :: <cond> -> {
	  <body1>
	  }
	  :: else -> {
	  <body2>
	  }
	fi;

IF с ELSIF	

IF <cond[1]> THEN
  <body[1]>
ELSIF <cond[2]>
  <body[2]>
…
ELSIF <cond[n]>
  <body[n]>
	if
	  :: <cond[1]> -> {
	  <body[1]>
	  }
	  :: else -> if :: <cond[2]> -> {
	  <body[2]>
	  }

ELSE
  <bodyElse>
END_IF
	…
	:: else -> if :: <cond[n]> -> {
	  <body[n]>
	}
	::else -> {
	  <bodyElse>
	}
	fi; … fi;

Таблица 4 – Трансляция условного оператора

Заметим, что в последнем случае нельзя ограничиться использованием одного оператора if с множеством веток, так как в poST проверка условий происходит последовательно.

2.4.3 Оператор ветвления

Оператор ветвления в poST рассчитывает результат выражения (тип – INT) и исполняет ту ветку, которой соответствует список значений, содержащий этот результат. Трансляцию можно осуществить с использованием оператора if (имя переменной упрощено для улучшения читаемости):

poST	**Promela**

CASE <expr> OF
  <list[1]> : <body[1]>
  …
  <list[n]> : <body[n]>
ELSE
  <elseBody>
END_CASE
	int v = <expr>;
	if
	  :: v == list[1][1] || … || v == list[1][m1] -> {
	  <body[1]>
	  }
	  …
	  :: v == list[n][1] || … || v == list[n][mn] -> {
	  <body[n]>
	  }
	  :: else -> {
	  <elseBody>
	  }
	fi;

Таблица 5 – Трансляция оператора ветвления

Здесь в отличие от оператора IF не обязательно использовать вложенные условные операторы, так как указание одного и того же значения для нескольких веток не поддерживается в poST, таким образом возможность возникновения неопределенности в выборе ветки исключена.

2.4.4 Операторы цикла

В poST существуют три оператора цикла:

- WHILE
- REPEAT
- FOR

В Promela представлен единственный оператор цикла “do” и он, как и оператор if, блокрующийся – чтобы выходить из цикла после окончания истинности условия, нужны ветки else с оператором выхода из цикла break.

Трансляция происходит в соответствии со следующей таблицей:

poST	**Promela**

WHILE <cond> DO
  <body>
END_WHILE
	do
	  :: <cond> -> {
	  <body>
	  }
	  :: else -> break;
	od;

REPEAT
  <body>
UNTIL <cond> END_REPEAT
	<body>
	do
	  :: <cond> -> {
	  <body>
	  }
	  :: else -> break;
	od;

FOR <i> := <start> TO <end> DO
  <body>
END_FOR
	<v> = <start>;
	do
	  :: <v> <= <end> -> {
	  <body>
	  <v> = <v> + 1;
	  }
	  :: else -> break;
	od;

FOR <i> := <start> TO <end> BY <step> DO
  <body>
END_FOR
	<v> = <start>;
	do
	  :: <v> <= <end> -> {
	  <body>
	  <v> = <v> + <step>;
	  }
	  :: else -> break;
	od;

Таблица 6 – Трансляция операторов цикла

2.4.5 Цикл активации процессов

Процессы составляют основу программы на poST. Они активируются в цикле друг за другом. Таким образом, нужно осуществить последовательную передачу хода от процесса к процессу – для этих целей будем использовать переменную типа chan из Promela.

Канал (chan) в Promela – тип данных для которого определены блокирующиеся операции чтения и записи. Канал способен содержать в себе различные структуры данных [8]. В данном случае будет использоваться канал емкости 1, вмещающий в себя переменную типа mtype (аналог enum из C), значения которой соответствуют множеству процессов poST.

Все процессы изначально находятся в состоянии блокирующегося чтения своего значения mtype из канала – разблокировки процесса не произойдет, пока соответствующее именно ему значение mtype не попадет в канал. Процесс init записывает в канал значение соответствующее первому процессу – так начинается исполнение программы. В конце своего “хода” процесс помещает в канал значение для следующего процесса – так происходит передача “хода” (последний процесс помещает в канал значение для первого).

Из того что единовременно только один процесс может быть активен следует, что во время “хода” некоторого процесса можно разрешить верификатору не учитывать для остальных процессов возможные изменения переменных, соответственно, можно окружить тело процесса в блоком atomic{}, который служит как раз для этого (то есть, остальные процессы “увидят” изменения переменных по окончании блока atomic{}) – это не повлияет на семантику, но при этом значительно упростит верификацию.

Трансляция процессов происходит следующим образом:

poST	**Promela**

PROCESS P1
  <states1>
END_PROCESS
 	
PROCESS P2
  <states2>
END_PROCESS
 	
PROCESS P3
  <states3>
END_PROCESS
	mtype:P__ = {
	  P1__p,
	  P2__p,
	  P2__p
	}
	chan __currentProcess = [1] of { mtype:P__ };
	
	init {
	  __currentProcess ! P1__p;
	}

	active proctype P1() {
	  do :: __currentProcess ? P1__p ->
	  atomic {
	<states1>
	  __currentProcess ! P2__p;
	  }
	  od;
	}
	
	active proctype P2() {
	  do :: __currentProcess ? P2__p ->
	  atomic {
	<states2>
	  __currentProcess ! P3__p;
	  }
	  od;
	}
	
	active proctype P3() {
	  do :: __currentProcess ? P3__p ->
	  atomic {
	<states3>
	  __currentProcess ! P1__p;
	  }
	  od;
	}

Таблица 7 – Цикл активации процессов

2.4.6 Процессы и состояния

Процесс всегда находятся в одном из своих состояний (STATE) – при активации исполняется код этого состояния. Это значит, что нужно запоминать текущее состояние процесса – для этого будем использовать переменную типа mtype. В poST изначально первые объявленные процессы в каждой из программ находятся в первом из своих состояний, а все остальные процессы – в состоянии STOP – это учитывается при трансляции.

Трансляция процесса происходит представленным образом:

poST	**Promela**

PROCESS P1
  <vars>
 	
  STATE s[1]
  <body[1]>
  END_STATE
	mtype:P1__S = {
	  s[1]__s,
	  …
	  s[n]__s,
	  Stop__s,
	  Error__s
	}

…
 	
STATE s[n]
<body[n]>
END_STATE
END_PROCESS
	mtype:P1__S P1__cs = s[1]__s;
	
	active proctype P1() {
	do :: __currentProcess ? P1__p ->
	atomic {
	if
	:: s[1]__s == P1__cs -> {
	<body[1]>
	}
	…
	:: s[n]__s == P1__cs -> {
	<body[n]>
	}
	:: else -> skip;
	fi;
	__currentProcess ! P2__p;
	}
	od;
	}

Таблица 8 – Трансляция процесса

Операторы проверки статуса процесса (ACTIVE, INACTIVE, STOP, ERROR), операторы изменения состояний процессов (START PROCESS, STOP PROCESS, ERROR PROCESS, STOP, ERROR, SET STATE, SET NEXT) на состояния без таймаутов транслируются тривиально.

Максимальное количество процессов в текущей версии SPIN – 255. Если модель превышает этот лимит, то программа транслируется с предупреждением.

2.4.7 Таймауты

В poST существует возможность объявить состояние с блоком TIMEOUT – инструкции из этого блока будут выполняться по истечении указанного в таймауте времени с момента входа процесса в данное состояние. Для организации такой возможности в Promela-программе понадобится ввести переменные, отсчитывающие время от входа в состояние с таймаутом – по одной для каждого процесса, содержащего такие состояния. Для уменьшения количества состояний системы будут использоваться переменные типа unsigned минимального достаточного размера. Например, если у процесса есть два состояния с таймаутами, которые отсчитывают 5 (101b) и 9 (1001b) единиц времени, то для таймаутов этого процесса понадобится одна переменная имеющая размер 4 бита.

Кроме того, можно уменьшить сами значения таймаутов, разделив их все на их НОД – в ряде случаев это позволит существенно уменьшить количество состояний. Например, если

в программе объявлены таймауты 100ms и 150ms, то, разделив их на НОД, получим соответственно 2 и 3 у.е. В таком случае, если, например, эти таймауты находятся в разных процессах, то для них понадобятся две переменные по два бита, вместо 7 и 8 бит соответственно. Однако такая оптимизация может привести к несоблюдению семантики, так что при использовании транслятора будет предоставляться возможность выбора, применять ее при трансляции данной программы или нет.

Еще одной полезной особенностью языка poST является возможность задания интервала вызова цикла активации процессов (INTERVAL) – если он используется, то все таймауты уменьшаются до ближайшего кратного этому интервалу – это также позволило в части случаев уменьшить значения таймаутов и, следовательно, упростить верификацию.

Трансляция процесса с таймаутами представлена ниже:

poST	**Promela**

…(INTERVAL := T#100ms)…
 	
PROCESS P1
  <vars>
  <states>
 	
  STATE S1
  <body1>
  TIMEOUT T#150ms then
  <tBody1>
  END_TIMEOUT
  END_STATE
 	
  STATE S2
  <body2>
  TIMEOUT T#1s then
  <tBody2>
  END_TIMEOUT
  END_STATE
END_PROCESS
	unsigned P1__t : 4;
	
	<vars>
	
	active proctype P1() {
	  …
	  <states>
	
	  :: S1__s == P1__cs -> {
	  <body1>
	  if :: P1__t > 1 -> {
	  <tBody1>
	  } :: else -> P1__t = P1__t + 1; fi;
	  }
	
	  :: S2__s == P1__cs -> {
	  <body2>
	  if :: P1__t > 10 -> {
	  <tBody2>
	  } :: else -> P1__t = P1__t + 1; fi;
	  }
	  …
	}

Таблица 9 – Трансляция таймаутов

При нахождении в состоянии с таймаутом соответствующую переменную необходимо инкрементировать каждый “ход”, кроме того нужно устанавливать ее в 0 или в 1 при изменении состояния любого процесса на состояние с таймаутом (в зависимости от того, на какой итерации, текущей или следующей, процесс с таймаутом будет запущен в следующий

раз) и в 0 при сбросе таймаута. При запуске процесса нужно так же устанавливать значение переменной равным 0 или 1, так как запуск – изменение состояния процесса на первое объявленное.

При проверке значения переменной таймаута используется знак “>”, так как, по семантике poST, исполнение блока таймаута начинается на следующей итерации после того, как время таймаута прошло. По этой причине размер переменной таймаута определяется из ее максимального значения, которое на 1 больше значения таймаута.

2.5 Взаимодействие программ

Программы взаимодействуют посредством передачи значений между INPUT, OUTPUT и IN_OUT переменными, имеющими одинаковые названия. Поддерживается “подключение” произвольного числа INPUT переменных к одной OUTPUT или IN_OUT переменной – в случае наличия таких связанных переменных в Promela модели создается отдельный процесс, осуществляющий копирование значений между итерациями цикла активации процессов:

poST	**Promela**

PROGRAM P1
  VAR_OUTPUT
  v : INT;
  END_VAR
  …
END_PROGRAM
 	
PROGRAM P2
  VAR_INPUT
  v : INT;
  END_VAR
  …
END_PROGRAM
 	
PROGRAM P3
  VAR_INPUT
  v : INT;
  END_VAR
  …
END_PROGRAM
	active proctype VarsSetter__specialProcess() {
	  do :: __currentProcess ? VarsSetter__sp ->
	  atomic {
	  v__1 = v__0; //P2__v ← P1__v
	  v__2 = v__0; //P3__v ← P1__v
	  __currentProcess ! Helper__sp;
	  }
	  od;
	}

Таблица 10 – Взаимодействие программ через переменные

2.6 Недетерминированные переменные

INPUT переменные программ poST, с которыми не ассоциированы OUTPUT и IN_OUT переменные, то есть для которых не представлено источников значений, считаются “недетерминированными” – то есть их значения определяются недетерминированно перед каждой итерацией цикла активации процессов, что и приводит к образованию множества различных сценариев работы системы.

Если в poST программе существуют такие недетерминированные переменные, то в Promela модель добавляется следующий процесс, реализующий недетерминированное поведение окружения:

poST	**Promela**

PROCESS P1
  VAR_INPUT
  b : BOOL;
  usInt : USINT;
  sInt : SINT;
  END_VAR
  …
END_PROCESS
	active proctype Gremlin__specialProcess() {
	  do :: __currentProcess ? Gremlin__sp ->
	  atomic {
	  if
	  :: b = true;
	  :: b = false;
	  fi;
	  select (usInt : 0..255);
	  select (sInt : -128..127);
	  __currentProcess ! VarsSetter__sp;
	  }
	  od;
	}

Таблица 11 – Недетерминированные переменные

Было решено разрешить только недетерминированные переменные типов BOOL, USINT и SINT, так как переменные большего размера могут слишком сильно усложнить верификацию.

2.7 Упрощение описания свойств

Для обеспечения возможности задания свойств на языке LTL для систем управления, основанных на циклах, в программу добавляется еще один процесс, который вызывается каждую итерацию цикла активации перед вызовом основных процессов, но после вызова рассмотренных специальных процессов Gremlin__specialProcess и VarsSetter__specialProcess:

```
bool cycle__u;
```

```
active proctype Helper__specialProcess() {
	do :: __currentProcess ? Helper__sp ->
		cycle__u = true;
		atomic {
			cycle__u = false;
			__currentProcess ! P1__p;
		}
	od;
}
```

Таблица 12 – Индикатор начала итерации цикла активации

Этот процесс создает импульс значения переменной cycle__u каждый раз перед вызовом основных процессов – таким образом в LTL формулах можно привязаться к началу итерации цикла активации.

Кроме того есть возможность включить добавление в конец файла на Promela представленных ниже макроопределений, позволяющих проверять свойства только между циклами активации. Это может быть важно, так как при применении poST INPUT и OUTPUT переменные синхронизируются с пинами микроконтроллера между итерациями цикла активации, а их изменения внутри итерации не влияют на окружение и не изменяются в зависимости от него.

Описание	**Promela**
Вспомогательные определения для использования в последующих. В итоге макрос apply__ltl преобразуется в n последовательных преобразований аргумента arg функцией f. Например apply__ltl(3, f, arg) соответствует f(f(f(arg))).	#define apply1__ltl(f, arg) f(arg) #define apply2__ltl(f, arg) f(apply1__ltl(f, arg)) #define apply3__ltl(f, arg) f(apply2__ltl(f, arg)) #define apply4__ltl(f, arg) f(apply3__ltl(f, arg)) #define apply5__ltl(f, arg) f(apply4__ltl(f, arg)) #define apply6__ltl(f, arg) f(apply5__ltl(f, arg)) #define apply7__ltl(f, arg) f(apply6__ltl(f, arg)) #define apply8__ltl(f, arg) f(apply7__ltl(f, arg)) #define apply9__ltl(f, arg) f(apply8__ltl(f, arg)) #define apply10__ltl(f, arg) f(apply9__ltl(f, arg)) #define apply__ltl(n, f, arg) apply##n##__ltl(f, arg)
Истинно ↔ expr истинно перед началом следующего цикла активации, то есть в момент следующего импульса значения cycle__u. Если импульс происходит в текущем состоянии модели, то expr все равно будет проверяться для следующего импульса.	#define afterCycle__ltl(expr) (cycle__u U (!cycle__u W (cycle__u && (expr))))
Истинно ↔ Если условие cond	#define afterNCyclesWith__ltl(n, cond, expr)

выполняется n итераций цикла активации подряд, то на итерации следующей за ними выражение expr будет истинно.	(apply__ltl(n, (cond) -> afterCycle__ltl, expr))
Истинно ↔ Если условие cond выполняется n или меньше итераций цикла активации подряд, то на итерации следующей за ними выражение expr будет истинно.	#define afterNCyclesOrSoonerWith__ltl(n, cond, expr) afterNCyclesWith__ltl(n, (cond) && !(expr), expr)
Макроопределения для проверки свойств только между итерациями цикла активации	
Аналоги для выражения и операторов ltl.	#define cltl(expr) (cycle__u -> (expr)) #define G__cltl(expr) [](cycle__u -> (expr)) #define F__cltl(expr) <>(cycle__u && (expr)) #define U__cltl(expr1, expr2) (cycle__u -> (expr1)) U (cycle__u && (expr2)) #define W__cltl(expr1, expr2) (cycle__u -> (expr1)) W (cycle__u && (expr2)) #define V__cltl(expr1, expr2) (cycle__u && (expr1)) V (cycle__u -> (expr2))
Аналог afterCycle__ltl, который проверяется только между итерациями.	#define next__cltl(expr) (cycle__u -> afterCycle__ltl(expr))
Аналог afterNCyclesWith__ltl.	#define afterNWith__cltl(n, cond, expr) (cycle__u -> afterNCyclesWith__ltl(n, cond, expr))
Аналог afterNCyclesOrSoonerWith__ltl.	#define afterNOrSoonerWith__cltl(n, cond, expr) (cycle__u -> afterNCyclesOrSoonerWith__ltl(n, cond, expr))

Таблица 13 – Вспомогательные макроопределения для LTL формул

3 Реализация транслятора

Транслятор предназначается для преобразования программы на языке poST в модель на Promela. За получение AST poST программы отвечает часть, относящаяся к ядру poST IDE. Задача основной части транслятора – получив на вход это AST, сформировать внутри себя представление соответствующей ему Promela-модели и затем предоставить его в виде программы, для проведения ее последующего анализа с помощью SPIN.

В соответствии с требованиями, транслятор разработан с использованием фреймворка xtext и языка xtend. Поставляется в виде jar файла, который запускается следующей командой (порядок ключей любой):

```
java -jar ./poST_to_promela.jar [-rt, -lm] -i <in> -o <out>

in – расположение файла poST программы,
out – расположение для файла Promela модели,
-rt – включает деление значения типа времени на их НОД,
-lm – включает добавление в конец файла вспомогательных
макроопределений для LTL формул.
```

Шаблон 4 – Запуск транслятора

3.1 Структура пакетов

Класс Main содержится в корневом пакете su.nsk.iae.post.generator.promela.

Остальные компоненты транслятора сгруппированы в следующие подпакеты корневого пакета:

- model – содержит классы, соответствующие основным сущностям языка poST и модели Promela (программы, процессы, состояния, специальные процессы, и т.д.).
    - model.expressions – пакет, содержащий классы, предназначенные для трансляции операций над переменными в poST.
    - model.statements – классы, соответствующие выражениям языка poST (операторы цикла, условия, таймаута, и т.п.).
    - model.vars – классы для трансляции переменных.
- context – классы контекстов (информация, доступ к которой предоставляют эти классы, запрашивается и модифицируется многими другими классами)
- exceptions – пакет для исключений.

3.2 Пакет model

Содержит следующие классы:

- IPromelaElement – интерфейс с методом “String toText()”. Из реализаций этого интерфейса состоит Promela-образ AST.
- PromelaModel – корневой класс для модели на Promela.
- PromelaProgram – в экземпляр этого класса транслируется программа (POGRAM) на poST.
- PromelaProcess – в экземпляр этого класса транслируется процесс на poST.
- PromelaState – для состояния процесса.
- PromelaElementList – вспомогательный класс для хранения и отображения списка IPromelaElement с разделителями.
- VarSettingProgram – соответствует двум специальным процессам Promela-модели, первый из которых недетерминированно устанавливает значения “свободных” INPUT-переменных, а второй предназначен для копирования значений между связанными переменными разных программ (PROGRAM).

3.2.1 Пакет model.expressions

Содержит следующие классы:

- PromelaExpression – его наследники соответствуют различным операциям в языке poST, переведенным в Promela:
    - Constant – константы (4, 17, true, и т.п.).
    - TimeConstant – число временных единиц, соответствующее временной константе в poST (T#550ms, T#1h8m5s17ms, и т.п.).
    - Var – переменная.
    - Not – отрицание.
    - Invert – взятие противоположного.
    - Primary – выражение в скобках.
    - Binary – бинарное выражение.
    - ProcessStatus – выражение проверки статуса процесса (ACTIVE, INACTIVE, STOP, ERROR).
    - ArrayVar – выражение доступа к элементу массива по индексу.
- PromelaExpressionsHelper – на основании вида операции на poST возвращает экземпляр соответствующего наследника PromelaExpression.

3.2.2 Пакет model.statements

Содержит следующие классы:

- PromelaStatement – его наследники соответствуют выражениям языка poST:

    - Assign – присваивание переменной значения.
    - If – условный оператор.
    - Case – оператор ветвления.
    - StartProcess – оператор запуска процесса.
    - SetState – оператор установки состояния процесса.
    - StopProcess – оператор нормальной остановк процесса.
    - ErrorProcess – оператор остановки процесса ввиду ошибки.
    - ResetTimer – оператор сброса таймаута.
    - Timeout – оператор таймаута (возникает в конце состояний с таймаутами).
    - While, Repeat, For – операторы цикла
- PromelaStatementsHelper – в зависимости от типа выражения на poST, создает нужного наследника PromelaStatement.

Кроме интерфейса IPromelaElement, PromelaStatement имплементирует интерфейс IPostConstuctible, экземпляры которого регистрируются в PostConstructContext, для того чтобы произвести для них вторую стадию построения (например, вторая стадия конструктора требуется для экземпляров StartProcess, так как запускаемый процесс может быть объявлен в программе на poST позже использования данного оператора, соответственно, во время первой стадии конструктора запускаемому процессу еще не будет соответствовать PromelaProcess).

3.2.3 Пакет model.vars

Содержит следующие классы:

- PromelaVar – соответствует переменной Promela-модели. Имеются следующие наследники:
    - Bool – логическая переменная.
    - Short – знаковая целочисленная переменная, 16 бит.
    - Int – знаковая целочисленная переменная, 32 бит.
    - Byte – беззнаковая целочисленная переменная, 8 бит.
    - Unsigned – беззнаковая целочисленная переменная, 1 - 31 бит.
    - TimeInterval – переменная для хранения значения временного типа. Реализована с использованием типа unsigned с минимальной необходимой разрядностью.
    - Array – массив.
- PromelaVarsHelper – в зависимости от VarInitDeclaration из AST poST создает экземпляр нужного наследника PromelaVar.

3.3 Пакет context

Содержит классы:

- PromelaContext – контекст, содержащий списки некоторых сущностей (временные переменные, временные значения, все процессы, все массивы, и т.п.).
- CurrentContext – в течение работы первой фазы конструкторов Promela-элементов содержит текущую программу, процесс и состояние.
- NamespaceContext – во время первой фазы конструкторов содержит информацию о текущем пространстве имен.
- PostConstructContext – содержит элементы, которым требуется запуск второй фазы конструктора.
- WarningsContext – содержит возникшие в ходе трансляции предупреждения.
- FullIdsToNamesMapper – вспомогательный класс, инкапсулированный в NamespaceContext. Служит для преобразования полных имен сущностей в упрощенные непосредственно перед образованием текста Promela-модели.

3.4 Пакет exceptions

Содержит исключения:

- ConflictingOutputsOrInOutsException – возникает, если несколько OUTPUT или IN_OUT переменных соответствуют одному и тому же множеству INPUT переменных.
- NotSupportedElementException – элемент языка poST не поддерживается.
- UnknownElementException – неизвестный элемент языка poST (который не подразумевается грамматикой poST).
- WrongModelStateException – непредвиденное состояние Promela модели.

4 Апробация транслятора

4.1 Схема тестирования

При решении задачи выделяются следующие этапы:

1. Формулируется условие задачи на естественном языке.
2. Определяются требования к программе на языке LTL с возможностью использования определенных ранее макросов.
3. Разрабатывается программа на poST.
4. С помощью транслятора строится модель на Promela.
5. Модель верифицируется с помощью Spin.
6. При обнаружении контрпримера, он анализируется, вносятся необходимые поправки и происходит возврат к этапу 4.

4.2 Задача “Светильник”

4.2.1 Описание задачи

Система состоит из осветительного прибора (далее “светильник”) и пользователя. Светильник предназначен для освещения пространства при обнаружении движения. Для этого контроллер светильника при обнаружении движения инфракрасным датчиком включает лампу. После того как в течение определенного времени не будет обнаружено движения, лампа будет выключена. Пользователь может двигаться либо не двигаться, причем менять свое состояние он может с любой частотой.

К программе были выдвинуты следующие требования (используются описанные ранее вспомогательные макроопределения):

Естественный язык	**Promela**
1) Лампа не включится впервые до обнаружения движения.	(lamp == OFF) W move
2) Выключенная лампа не включится до обнаружения движения.	[] cltl(lamp == OFF -> W__cltl(lamp == OFF, move))
3) Включенная лампа не выключится, пока движение есть.	[] cltl(lamp == ONN -> W__cltl(lamp == ONN, !move))
4) Лампа включится не позднее, чем через 100ms после обнаружения	[] afterNOrSoonerWith__cltl(N, move, lamp == ONN)

движения.	Значение N выбирается в зависимости от интервала между итерациями цикла активации.

Таблица 14 – Требования к светильнику

4.2.2 Разработка программы

В данном случае явным образом определить процесс необходимо только для контроллера светильника. В качестве непредсказуемо двигающегося или не двигающегося пользователя будет выступать Gremlin__specialProcess – он будет недетерминированно задавать значение переменной move. Надстроек над переменными move и lamp в виде процессов для датчика движения и устройства включения лампы было решено не создавать, так как они по условию не содержат дополнительной логики.

Программа представлена в приложении А.

4.2.2 Верификация программы

Исходя из значения INTERVAL равного 100ms, для 4-го свойства N было взято равным 1. В таком случае свойство 4 означает, что, если появится движение, то одной итерации цикла активации должно хватить, чтобы отреагировать на это включением лампы.

Модель (приложение А) на Promela была получена с помощью транслятора и верифицирована с помощью Spin.

Проверка доказала выполнение свойств 1 – 3 и выявила невыполнение свойства 4:

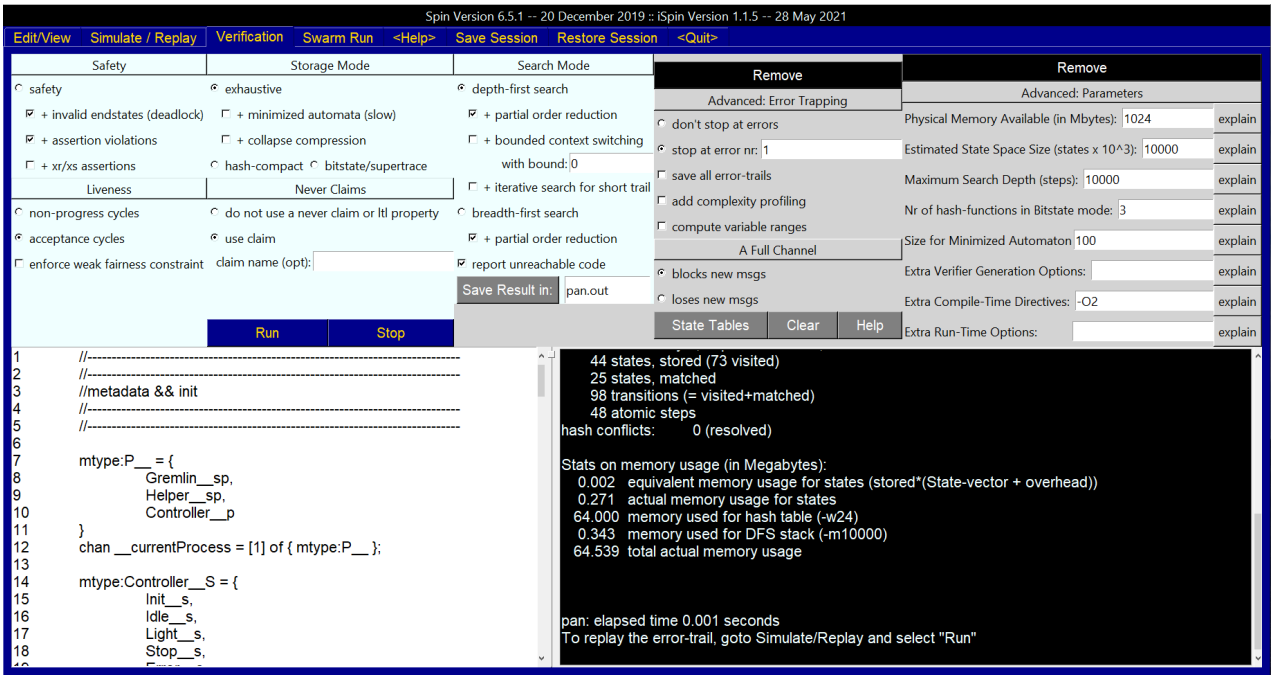

Рисунок 1 – Верификация требования 4 задачи “Светильник”

Рисунок 2 – Контрпример требования 4 задачи “Светильник”

Изучение контрпримера показало, что несоответствие возникает, если движение есть (строка 6) с самого начала работы контроллера (когда его состояние еще Init (строка 18)), потому что лампа в этом состоянии устанавливается выключенной безусловно и включается при сохранении движения только после следующего цикла, в котором контроллер будет в состоянии Idle.

Для исправления поведения программы были опробованы два пути:

- Уменьшить вдвое INTERVAL, что приведет к необходимости взять в свойстве 4 N равным 2, а это более слабое требование.
- Добавить в состояние Init условие для определения начального состояния лампы.

При проверке оба способа показали свою работоспособность, хотя можно было ослабить свойство, не требуя реакции контроллера после первого цикла активации, увеличив на 100ms характеристику времени инициализации устройства.

4.3 Задача “Солнечная электростанция”

4.3.1 Описание задачи

Составляющие системы:

- Солнце - не светит ночью (14 часов), днем (10 часов) может светить или не светить (облака) – это может меняться в течение дня.
- Солнечная панель – вырабатывает одну единицу мощности, если светит солнце, иначе не вырабатывает.

- Батарея – накапливает до 19 единиц энергии, если энергии становиться 0 или 20 единиц, то батарея “ломается”.
- Город – может потреблять или не потреблять одну единицу мощности.
- Нагрузка – может быть включена для рассеивания лишней энергии, если солнце светит, а город не потребляет энергию. Включается мгновенно, время выключения – один час.
- Генератор – может быть включен при недостатке мощности, когда солнце не светит, а город потребляет энергию. Время включения – 1 час, выключается мгновенно.
- Контроллер – включает и выключает генератор и нагрузку для удержания энергии батареи в допустимом диапазоне.

К программе были выдвинуты следующие требования:

Естественный язык	**Promela**
1) Батарея не сломается (из этого следует, что город всегда будет обеспечен энергией)	G__cltl(!batteryBroken)
2) Потребитель и генератор не включаются одновременно (это привело бы к излишнему потреблению топлива)	G__cltl(!(generatorOn & consumerOn))

Таблица 15 – Требования к солнечной электростанции

4.3.2 Разработка программы

Разработанная программа состоит из следующих блоков PROGRAM:

- Environment – содержит процесс Sun с состояниями Day и Night и переменной clouds с недетерминированным значением.
- EnergyProvider – содержит процессы:
    - Init – запускает все процессы данного блока PROGRAM,
    - Controller – содержит состояния Normal, energyLack, energyOverflow,
    - Battery – реализует зарядку и разрядку батареи, “ломается” при его выходе за допустимый диапазон.
- ReserveEnergyProvider – содержит процесс Generator.
- ReserveEnergyConsumer – содержит процесс Consumer.

Город представлен в виде логической INPUT переменной cityConsuming. Процессы из разных блоков PROGRAM взаимодействуют через INPUT и OUTPUT переменные.

Программа представлена в приложении Б.

4.3.3 Верификация программы

Модель (приложение Б) на Promela была получена с помощью транслятора и верифицирована с помощью Spin. Проверка показала невыполнение требований 1 и 2:

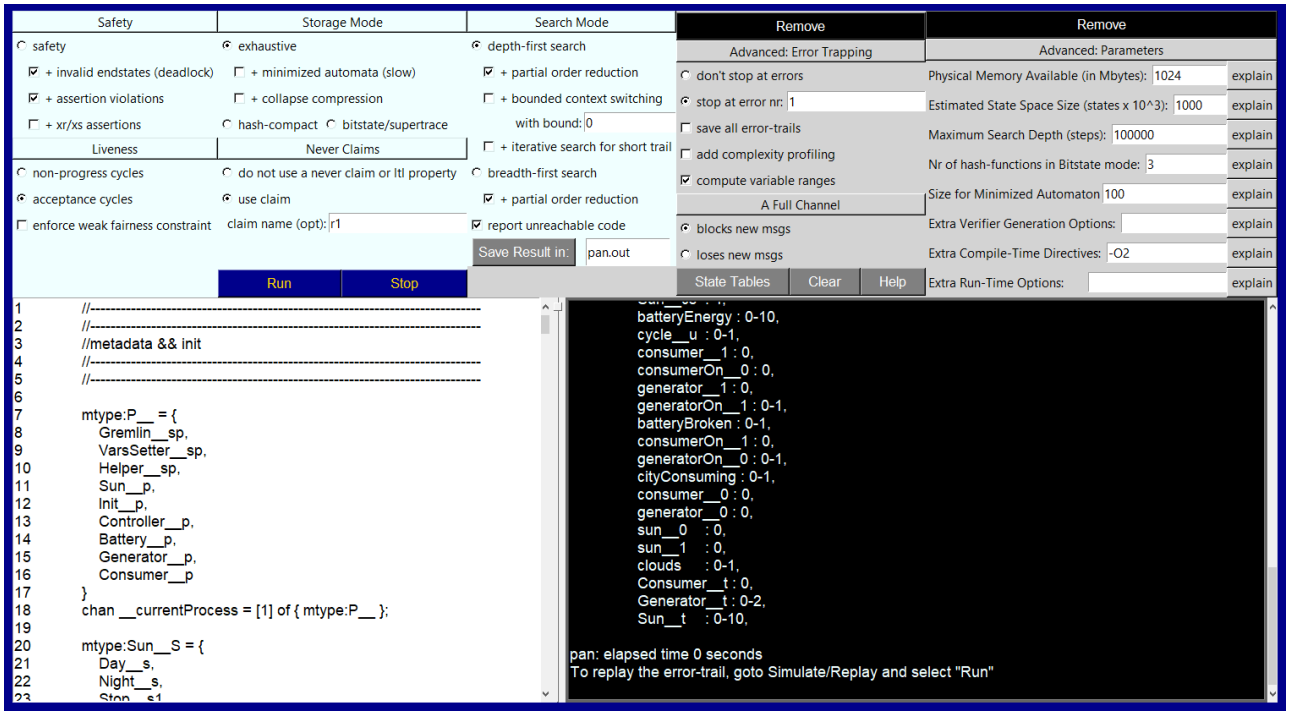

Рисунок 3 – Верификация требования 1 к задаче “Солнечная электростанция”

Изучение контрпримера с учетом диапазона переменной batteryEnergy указало на ошибки в коде, в том числе неверно определенный порог для вхождения в состояние нехватки энергии. Были внесены правки в программу (перечислены в приложении Б). Верификация ее модели показала выполнение поставленных требований.

ЗАКЛЮЧЕНИЕ

В ходе работы проводился анализ специфики языков poST и Promela, на основе которого определялись требования к правилам трансляции и транслятору. Разрабатывались правила трансляции конструкций poST в конструкции Promela, удовлетворяющие требованиям. Разрабатывался метод представления на языке LTL свойств для систем управления, основанных на циклах. Осуществлялась реализация транслятора, использующего разработанные правила трансляции и удовлетворяющий требованиям. Проводилась апробация транслятора на наборе тестовых задач.

Были получены следующие результаты:

- Определены правила трансляции poST в Promela,
- Разработан метод представления на языке LTL свойств для систем управления, основанных на циклах,
- Реализован транслятор poST в Promela,
- Продемонстрирована корректность работы транслятора на наборе тестовых задач.

В дальнейшем планируется реализация модуля для poST IDE на основе разработанного транслятора. Также планируется разработка решения для трассировки изменений Promela модели относительно исходной poST-программы и предложения варианта ее коррекции для соответствия модели.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Кондратьев И.И. _____

ФИО студента *Подпись студента*

« ____ »_____20 __г.

(заполняется от руки)

СПИСОК ЛИЕТРАТУРЫ

1. Gerard Holzmann. The SPIN Model Checker: Primer and Reference Manual / Gerard Holzmann; Boston, Massachusetts. – Addison-Wesley Professional, Boston, MA, 2011 – 608 p.
2. Christel Baier. Principles of Model Checking / Christel Baier, Joost-Pieter Katoen; Cambridge, Massachusetts. – The MIT Press, 55 Hayward Street, Cambridge, MA, 2008 – 975 p.
3. Mordechai Ben-Ari. Principles of the Spin Model Checker / Mordechai Ben-Ari, G.J. Holzman; Weizmann inst. of science. – Springer-Verlag London Ltd, 2008. – 255 p.
4. Шошмина, И.В. Введение в язык Promela и систему комплексной верификации Spin / И.В. Шошмина, Ю.Г. Карпов; С.-Петерб. гос. политехн. ун-т. – СПб: Изд-во Политехн. ун-та, 2010. – 66 с.
5. Spin general description // URL: https://spinroot.com/spin/what.html (дата обращения: 02.05.2022)
6. Concise Promela Reference // URL: http://spinroot.com/spin/Man/Quick.html?__cf_chl_managed_tk__=pmd_9hCm3AX90ynJbUqvDDO9HZebUx_sZyAimw7wbUO.IN0-1633775864-0-gqNtZGzNAuWjcnBszQPl (дата обращения: 02.05.2022)
7. Грамматика языка poST // URL: https://github.com/v-bashev/post_core/blob/main/su.nsk.iae.post.parent/su.nsk.iae.post/src/su/nsk/iae/post/PoST.xtext (дата обращения: 02.05.2022)
8. Грамматика языка Promela // URL: http://spinroot.com/spin/Man/grammar.html?__cf_chl_managed_tk__=pmd_We1eGuNpbXmjSu0Z.1tCBu_VCheq4PBuT_SYroXz6ew-1634641460-0-gqNtZGzNAuWjcnBszQPl#stmnt (дата обращения: 02.05.2022)
9. Promela channels // URL: http://spinroot.com/spin/Man/chan.html (дата обращения: 02.05.2022)
10. Xtext документация // URL: https://www.eclipse.org/Xtext/documentation/index.html (дата обращения: 02.05.2022)
11. Карпов, Ю.Г. Темпоральные логики для спецификации свойств программных и аппаратных систем / Ю.Г. Карпов // Компьютерные инструменты в образовании. – 2009. – 2. – С. 45-56
12. Башев, В.И. Разработка процесс-ориентированного расширения языка Structured Text из состава IEC 61131-3 // Материалы Международной научно-технической

конференции «Автоматизация» (RusAutoCon) (Сочи, 6-12 сент. 2020). - Сочи, 2020. - С. 994-999.

13. Верификация программ методом model checking / А.М. Миронов. URL: http://intsys.msu.ru/staff/mironov/model_checking.pdf (дата обращения: 02.05.2022)
14. Карпов, Ю.Г. Model Checking. Верификация параллельных и распределенных программных систем / Ю.Г. Карпов. - БХВ-Петербург, 2010. - 552 с.
15. ST документация // URL: https://infosys.beckhoff.com/english.php?content=../content/1033/tcplccontrol/html/TcPlcCtrl_Languages%20ST.htm&id=#:~:text=The%20structured%20text%20consists%20of,.DO)%20can%20be%20executed.&text=An%20expression%20is%20a%20construction,composed%20of%20operators%20and%20operands. (дата обращения: 02.05.2022)
16. Structured Text Programming: A Step by Step Guide (With Examples) // URL: https://www.plcacademy.com/structured-text-tutorial/ (дата обращения: 02.05.2022)
17. Транслятор из poST в ST / URL: http://post2st.iae.nsk.su/ (дата обращения: 02.05.2022)

ПРИЛОЖЕНИЕ А – “Светильник”

Строка запуска транслятора:

java -jar poST_to_promela.jar -lm -i program.post -o model.pml

Программа на poST:

```
CONFIGURATION testConfig
        RESOURCE testResource ON TestCPU
                TASK PromelaVerificationTask (INTERVAL := T#100ms);
        END_RESOURCE
END_CONFIGURATION

PROGRAM Light

   VAR CONSTANT
      ONN : BOOL := TRUE;
      OFF : BOOL := NOT ONN;
      TURN_OFF_TIMEOUT : TIME := T#10s;
   END_VAR
   VAR_INPUT
      move : BOOL;
   END_VAR
   VAR_OUTPUT
      lamp : BOOL;
   END_VAR

   PROCESS Controller

      STATE Init
         lamp := OFF;
         SET NEXT;
      END_STATE

      STATE Idle
         IF move THEN
            lamp := ONN;
            SET NEXT;
         END_IF
      END_STATE

      STATE Light
         IF move THEN
            RESET TIMER;
         END_IF
         TIMEOUT TURN_OFF_TIMEOUT THEN
            lamp := OFF;
            SET STATE Idle;
         END_TIMEOUT
      END_STATE
```

```
   END_PROCESS

END_PROGRAM
```

Модель на Promela:

```
//------------------------------------------------------------------------
//------------------------------------------------------------------------
//metadata && init
//------------------------------------------------------------------------
//------------------------------------------------------------------------

mtype:P__ = {
   Gremlin__sp,
   Helper__sp,
   Controller__p
}
chan __currentProcess = [1] of { mtype:P__ };

mtype:Controller__S = {
   Init__s,
   Idle__s,
   Light__s,
   Stop__s,
   Error__s
}
mtype:Controller__S Controller__cs = Init__s;
unsigned Controller__t : 8;

init {
   __currentProcess ! Gremlin__sp;
}



//------------------------------------------------------------------------
//------------------------------------------------------------------------
//program Light
//------------------------------------------------------------------------
//------------------------------------------------------------------------

//constants
#define ONN true
#define OFF !ONN
#define TURN_OFF_TIMEOUT 200 //10s

//input
bool move;

//output
bool lamp;
```

```
//------------------------------------------------------------------------
//Controller
//------------------------------------------------------------------------

active proctype Controller() {
   do :: __currentProcess ? Controller__p ->
      atomic {
         if
            :: Init__s == Controller__cs -> {
               lamp = OFF;
               Controller__cs = Idle__s;
            }
            :: Idle__s == Controller__cs -> {
               if :: move -> {
                  lamp = ONN;
                  Controller__t = 1;
                  Controller__cs = Light__s;
               } :: else -> skip; fi;
            }
            :: Light__s == Controller__cs -> {
               if :: move -> {
                  Controller__t = 0;
               } :: else -> skip; fi;
               if :: Controller__t == TURN_OFF_TIMEOUT -> {
                  lamp = OFF;
                  Controller__cs = Idle__s;
               } :: else -> Controller__t = Controller__t + 1; fi;
            }
            :: else -> skip;
         fi;
         __currentProcess ! Gremlin__sp;
   }
   od;
}


//------------------------------------------------------------------------
//------------------------------------------------------------------------
//special processes
//------------------------------------------------------------------------
//------------------------------------------------------------------------

active proctype Gremlin__specialProcess() {
   do :: __currentProcess ? Gremlin__sp ->
      atomic {
         if
            :: move = true;
            :: move = false;
         fi;
         __currentProcess ! Helper__sp;
      }
```

```
    od;
}


bool cycle__u;
active proctype Helper__specialProcess() {
    do :: __currentProcess ? Helper__sp ->
        cycle__u = true;
        atomic {
            cycle__u = false;
            __currentProcess ! Controller__p;
    }
    od;
}


//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
//ltl
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

#define apply1__ltl(f, arg) f(arg)
#define apply2__ltl(f, arg) f(apply1__ltl(f, arg))
#define apply3__ltl(f, arg) f(apply2__ltl(f, arg))
#define apply4__ltl(f, arg) f(apply3__ltl(f, arg))
#define apply5__ltl(f, arg) f(apply4__ltl(f, arg))
#define apply6__ltl(f, arg) f(apply5__ltl(f, arg))
#define apply7__ltl(f, arg) f(apply6__ltl(f, arg))
#define apply8__ltl(f, arg) f(apply7__ltl(f, arg))
#define apply9__ltl(f, arg) f(apply8__ltl(f, arg))
#define apply10__ltl(f, arg) f(apply9__ltl(f, arg))
#define apply__ltl(n, f, arg) apply##n##__ltl(f, arg)

#define afterCycle__ltl(expr) (cycle__u U (!cycle__u W (cycle__u && (expr))))
#define afterNCyclesWith__ltl(n, cond, expr) (apply__ltl(n, (cond) -> afterCycle__ltl, expr))
#define afterNCyclesOrSoonerWith__ltl(n, cond, expr) afterNCyclesWith__ltl(n, (cond) && !(expr), expr)

//-------------------------------------------------------------------------
//ltl between cycles
//-------------------------------------------------------------------------

#define cltl(expr) (cycle__u -> (expr))
#define G__cltl(expr) [](cycle__u -> (expr))
#define F__cltl(expr) <>(cycle__u && (expr))
#define U__cltl(expr1, expr2) (cycle__u -> (expr1)) U (cycle__u && (expr2))
#define W__cltl(expr1, expr2) (cycle__u -> (expr1)) W (cycle__u && (expr2))
#define V__cltl(expr1, expr2) (cycle__u && (expr1)) V (cycle__u -> (expr2))

#define next__cltl(expr) (cycle__u -> afterCycle__ltl(expr))
```

```
#define afterNWith__cltl(n, cond, expr) (cycle__u -> afterNCyclesWith__ltl(n, cond, expr))
#define afterNOrSoonerWith__cltl(n, cond, expr) (cycle__u -> afterNCyclesOrSoonerWith__ltl(n, cond, expr))


ltl r1 { (lamp == OFF) W move }
ltl r2 {[] cltl(
   lamp == OFF -> W__cltl(lamp == OFF, move)
)}
ltl r3 {[] cltl(
   lamp == ONN -> W__cltl(lamp == ONN, !move)
)}
ltl r4 {[]
   afterNOrSoonerWith__cltl(1, move, lamp == ONN)
}
```

ПРИЛОЖЕНИЕ Б – “Солнечная электростанция”

Строка запуска транслятора:

java -jar poST_to_promela.jar -lm -rt -i program.post -o model.pml

Первая версия программы на poST:

```
PROGRAM Environment
  VAR CONSTANT
    DAY_DURATION : TIME := T#14h;
    NIGHT_DURATION : TIME := T#10h;
  END_VAR
  VAR_INPUT
    clouds : BOOL;//gremlin
  END_VAR
  VAR_OUTPUT
    sun : BOOL;
  END_VAR

  PROCESS Sun
    STATE Day
      sun := NOT clouds;
      TIMEOUT DAY_DURATION THEN
        SET STATE Night;
      END_TIMEOUT
    END_STATE

    STATE Night
      sun := FALSE;
      TIMEOUT NIGHT_DURATION THEN
        SET STATE Day;
      END_TIMEOUT
    END_STATE
  END_PROCESS
END_PROGRAM

PROGRAM EnergyProvider
  VAR CONSTANT
    BATTERY_INIT_ENERGY : USINT := 10;
    LACK_THRESHOLD : USINT := 3;
    OVERFLOW_THRESHOLD : USINT := 17;
  END_VAR
  VAR_INPUT
    sun : BOOL;
    generator : BOOL;
    consumer : BOOL;
    cityConsuming : BOOL;//gremlin
  END_VAR
  VAR_OUTPUT
    generatorOn : BOOL;
    consumerOn : BOOL;
```

```
  batteryBroken : BOOL;
END_VAR
VAR
  batteryEnergy : USINT := BATTERY_INIT_ENERGY;
END_VAR

PROCESS Init
  STATE Init
    START PROCESS Battery;
    START PROCESS Controller;
    STOP;
  END_STATE
END_PROCESS

PROCESS Controller
  STATE Normal
    IF batteryEnergy <= LACK_THRESHOLD THEN
      generatorOn := TRUE;
      SET STATE energyLack;
    END_IF
    IF batteryEnergy >= OVERFLOW_THRESHOLD THEN
      consumerOn := TRUE;
      SET STATE energyOverflow;
    END_IF
  END_STATE
  STATE energyLack
    IF batteryEnergy > LACK_THRESHOLD THEN
      consumerOn := FALSE;
      SET STATE Normal;
    END_IF
  END_STATE
  STATE energyOverflow
    IF batteryEnergy < OVERFLOW_THRESHOLD THEN
      generatorOn := FALSE;
      SET STATE Normal;
    END_IF
  END_STATE
END_PROCESS

PROCESS Battery
  VAR CONSTANT
    MAX_ENERGY : USINT := 19;
    MIN_ENERGY : USINT := 1;
  END_VAR
  VAR
    change : SINT;
  END_VAR
  STATE active
    change := 0;
    IF sun THEN
      change := change + 1;
    END_IF
```

```
            IF generator THEN
               change := change + 1;
            END_IF
            IF consumer THEN
               change := change - 1;
            END_IF
            IF cityConsuming THEN
               change := change - 1;
            END_IF
            batteryEnergy := batteryEnergy + change;
                  IF (batteryEnergy < MIN_ENERGY) OR (batteryEnergy > MAX_ENERGY)
THEN
               batteryBroken := TRUE;
               ERROR;
            END_IF
         END_STATE
      END_PROCESS
   END_PROGRAM

   PROGRAM ReserveEnergyProvider
      VAR_INPUT
         generatorOn : BOOL;
      END_VAR
      VAR_OUTPUT
         generator : BOOL;
      END_VAR

      PROCESS Generator
         VAR CONSTANT
            START_TIME : TIME := T#1h;
         END_VAR

         STATE Wait
            IF generatorOn THEN
               SET NEXT;
            END_IF
         END_STATE
         STATE Starting
            IF NOT generatorOn THEN
               SET STATE Wait;
            END_IF
            TIMEOUT START_TIME THEN
               generator := TRUE;
               SET NEXT;
            END_TIMEOUT
         END_STATE
         STATE Working
            IF NOT generatorOn THEN
               generator := FALSE;
               SET STATE Wait;
            END_IF
         END_STATE
```

```
  END_PROCESS
END_PROGRAM

PROGRAM ReserveEnergyConsumer
  VAR_INPUT
    consumerOn : BOOL;
  END_VAR
  VAR_OUTPUT
    consumer : BOOL;
  END_VAR

  PROCESS Consumer
    VAR CONSTANT
      STOP_TIME : TIME := T#1h;
    END_VAR
    STATE Off
      IF consumerOn THEN
        consumer := TRUE;
        SET STATE On;
      END_IF
    END_STATE
    STATE On
      IF NOT consumerOn THEN
        SET STATE Stopping;
      END_IF
    END_STATE
    STATE Stopping
      TIMEOUT STOP_TIME THEN
        consumer := FALSE;
        SET STATE Off;
      END_TIMEOUT
    END_STATE
  END_PROCESS
END_PROGRAM
```

Модель первой версии программы:

```
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//metadata && init
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------

mtype:P__ = {
  Gremlin__sp,
  VarsSetter__sp,
  Helper__sp,
  Sun__p,
  Init__p,
  Controller__p,
  Battery__p,
  Generator__p,
```

```
    Consumer__p
}
chan __currentProcess = [1] of { mtype:P__ };

mtype:Sun__S = {
    Day__s,
    Night__s,
    Stop__s1,
    Error__s3
}
mtype:Sun__S Sun__cs = Day__s;
unsigned Sun__t : 4;

mtype:Init__S = {
    Init__s,
    Stop__s5,
    Error__s2
}
mtype:Init__S Init__cs = Init__s;

mtype:Controller__S = {
    Normal__s,
    energyLack__s,
    energyOverflow__s,
    Stop__s4,
    Error__s0
}
mtype:Controller__S Controller__cs = Stop__s4;

mtype:Battery__S = {
    active__s,
    Stop__s0,
    Error__s5
}
mtype:Battery__S Battery__cs = Stop__s0;

mtype:Generator__S = {
    Wait__s,
    Starting__s,
    Working__s,
    Stop__s3,
    Error__s4
}
mtype:Generator__S Generator__cs = Wait__s;
unsigned Generator__t : 2;

mtype:Consumer__S = {
    Off__s,
    On__s,
    Stopping__s,
    Stop__s2,
    Error__s1
```

```
}
mtype:Consumer__S Consumer__cs = Off__s;
unsigned Consumer__t : 2;

init {
   __currentProcess ! Gremlin__sp;
}



//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
//program Environment
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

//constants
#define DAY_DURATION 14 //14h
#define NIGHT_DURATION 10 //10h

//input
bool clouds;

//output
bool sun__1;


//-------------------------------------------------------------------------
//Sun
//-------------------------------------------------------------------------

active proctype Sun() {
   do :: __currentProcess ? Sun__p ->
      atomic {
         if
            :: Day__s == Sun__cs -> {
               sun__1 = !clouds;
               if :: Sun__t > DAY_DURATION -> {
                  Sun__t = 1;
                  Sun__cs = Night__s;
               } :: else -> Sun__t = Sun__t + 1; fi;
            }
            :: Night__s == Sun__cs -> {
               sun__1 = false;
               if :: Sun__t > NIGHT_DURATION -> {
                  Sun__t = 1;
                  Sun__cs = Day__s;
               } :: else -> Sun__t = Sun__t + 1; fi;
            }
            :: else -> skip;
         fi;
         __currentProcess ! Init__p;
```

```
        }
    od;
}


//------------------------------------------------------------------------
//------------------------------------------------------------------------
//program EnergyProvider
//------------------------------------------------------------------------
//------------------------------------------------------------------------

//constants
#define BATTERY_INIT_ENERGY 10
#define LACK_THRESHOLD 3
#define OVERFLOW_THRESHOLD 17

//input
bool sun__0;
bool generator__0;
bool consumer__0;
bool cityConsuming;

//output
bool generatorOn__0;
bool consumerOn__1;
bool batteryBroken;

//vars
byte batteryEnergy = BATTERY_INIT_ENERGY;


//------------------------------------------------------------------------
//Init
//------------------------------------------------------------------------

active proctype Init() {
    do :: __currentProcess ? Init__p ->
        atomic {
            if
                :: Init__s == Init__cs -> {
                    Battery__cs = active__s;
                    Controller__cs = Normal__s;
                    Init__cs = Stop__s5;
                }
                :: else -> skip;
            fi;
            __currentProcess ! Controller__p;
        }
    od;
}
```

```
//------------------------------------------------------------------------
//Controller
//------------------------------------------------------------------------

active proctype Controller() {
   do :: __currentProcess ? Controller__p ->
      atomic {
         if
            :: Normal__s == Controller__cs -> {
               if :: batteryEnergy <= LACK_THRESHOLD -> {
                  generatorOn__0 = true;
                  Controller__cs = energyLack__s;
               } :: else -> skip; fi;
               if :: batteryEnergy >= OVERFLOW_THRESHOLD -> {
                  consumerOn__1 = true;
                  Controller__cs = energyOverflow__s;
               } :: else -> skip; fi;
            }
            :: energyLack__s == Controller__cs -> {
               if :: batteryEnergy > LACK_THRESHOLD -> {
                  consumerOn__1 = false;
                  Controller__cs = Normal__s;
               } :: else -> skip; fi;
            }
            :: energyOverflow__s == Controller__cs -> {
               if :: batteryEnergy < OVERFLOW_THRESHOLD -> {
                  generatorOn__0 = false;
                  Controller__cs = Normal__s;
               } :: else -> skip; fi;
            }
            :: else -> skip;
         fi;
         __currentProcess ! Battery__p;
      }
   od;
}


//------------------------------------------------------------------------
//Battery
//------------------------------------------------------------------------

//constants
#define MAX_ENERGY 19
#define MIN_ENERGY 1

//vars
short change;

active proctype Battery() {
   do :: __currentProcess ? Battery__p ->
```

```
        atomic {
           if
              :: active__s == Battery__cs -> {
                 change = 0;
                 if :: sun__0 -> {
                    change = change + 1;
                 } :: else -> skip; fi;
                 if :: generator__0 -> {
                    change = change + 1;
                 } :: else -> skip; fi;
                 if :: consumer__0 -> {
                    change = change - 1;
                 } :: else -> skip; fi;
                 if :: cityConsuming -> {
                    change = change - 1;
                 } :: else -> skip; fi;
                 batteryEnergy = batteryEnergy + change;
                  if :: (batteryEnergy < MIN_ENERGY) | (batteryEnergy > MAX_ENERGY) ->
{
                    batteryBroken = true;
                    Battery__cs = Error__s5;
                 } :: else -> skip; fi;
              }
              :: else -> skip;
           fi;
           __currentProcess ! Generator__p;
        }
     od;
  }



  //-------------------------------------------------------------------------
  //-------------------------------------------------------------------------
  //program ReserveEnergyProvider
  //-------------------------------------------------------------------------
  //-------------------------------------------------------------------------

  //input
  bool generatorOn__1;

  //output
  bool generator__1;


  //-------------------------------------------------------------------------
  //Generator
  //-------------------------------------------------------------------------

  //constants
  #define START_TIME 1 //1h
```

```
active proctype Generator() {
   do :: __currentProcess ? Generator__p ->
      atomic {
         if
            :: Wait__s == Generator__cs -> {
               if :: generatorOn__1 -> {
                  Generator__t = 1;
                  Generator__cs = Starting__s;
               } :: else -> skip; fi;
            }
            :: Starting__s == Generator__cs -> {
               if :: !generatorOn__1 -> {
                  Generator__cs = Wait__s;
               } :: else -> skip; fi;
               if :: Generator__t > START_TIME -> {
                  generator__1 = true;
                  Generator__cs = Working__s;
               } :: else -> Generator__t = Generator__t + 1; fi;
            }
            :: Working__s == Generator__cs -> {
               if :: !generatorOn__1 -> {
                  generator__1 = false;
                  Generator__cs = Wait__s;
               } :: else -> skip; fi;
            }
            :: else -> skip;
         fi;
         __currentProcess ! Consumer__p;
      }
   od;
}



//------------------------------------------------------------------------
//------------------------------------------------------------------------
//program ReserveEnergyConsumer
//------------------------------------------------------------------------
//------------------------------------------------------------------------

//input
bool consumerOn__0;

//output
bool consumer__1;


//------------------------------------------------------------------------
//Consumer
//------------------------------------------------------------------------

//constants
```

```
#define STOP_TIME 1 //1h

active proctype Consumer() {
   do :: __currentProcess ? Consumer__p ->
      atomic {
         if
            :: Off__s == Consumer__cs -> {
               if :: consumerOn__0 -> {
                  consumer__1 = true;
                  Consumer__cs = On__s;
               } :: else -> skip; fi;
            }
            :: On__s == Consumer__cs -> {
               if :: !consumerOn__0 -> {
                  Consumer__t = 1;
                  Consumer__cs = Stopping__s;
               } :: else -> skip; fi;
            }
            :: Stopping__s == Consumer__cs -> {
               if :: Consumer__t > STOP_TIME -> {
                  consumer__1 = false;
                  Consumer__cs = Off__s;
               } :: else -> Consumer__t = Consumer__t + 1; fi;
            }
            :: else -> skip;
         fi;
         __currentProcess ! Gremlin__sp;
      }
   od;
}


//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
//special processes
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

active proctype Gremlin__specialProcess() {
   do :: __currentProcess ? Gremlin__sp ->
      atomic {
         if
            :: cityConsuming = true;
            :: cityConsuming = false;
         fi;
         if
            :: clouds = true;
            :: clouds = false;
         fi;
         __currentProcess ! VarsSetter__sp;
      }
   od;
```

```
}

active proctype VarsSetter__specialProcess() {
  do :: __currentProcess ? VarsSetter__sp ->
    atomic {
                generatorOn__1 = generatorOn__0; //ReserveEnergyP__generatorOn <-
Ene__generatorOn
      generator__0 = generator__1; //Ene__generator <- ReserveEnergyP__generator
      consumer__0 = consumer__1; //Ene__consumer <- ReserveEnergyC__consumer
              consumerOn__0 = consumerOn__1; //ReserveEnergyC__consumerOn <-
Ene__consumerOn
      sun__0 = sun__1; //Ene__sun <- Env__sun
      __currentProcess ! Helper__sp;
    }
  od;
}

bool cycle__u;
active proctype Helper__specialProcess() {
  do :: __currentProcess ? Helper__sp ->
    cycle__u = true;
    atomic {
      cycle__u = false;
      __currentProcess ! Sun__p;
    }
  od;
}


//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//ltl
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

#define apply1__ltl(f, arg) f(arg)
#define apply2__ltl(f, arg) f(apply1__ltl(f, arg))
#define apply3__ltl(f, arg) f(apply2__ltl(f, arg))
#define apply4__ltl(f, arg) f(apply3__ltl(f, arg))
#define apply5__ltl(f, arg) f(apply4__ltl(f, arg))
#define apply6__ltl(f, arg) f(apply5__ltl(f, arg))
#define apply7__ltl(f, arg) f(apply6__ltl(f, arg))
#define apply8__ltl(f, arg) f(apply7__ltl(f, arg))
#define apply9__ltl(f, arg) f(apply8__ltl(f, arg))
#define apply10__ltl(f, arg) f(apply9__ltl(f, arg))
#define apply__ltl(n, f, arg) apply##n##__ltl(f, arg)

#define afterCycle__ltl(expr) (cycle__u U (!cycle__u W (cycle__u && (expr))))
#define afterNCyclesWith__ltl(n, cond, expr) (apply__ltl(n, (cond) -> afterCycle__ltl,
expr))
#define afterNCyclesOrSoonerWith__ltl(n, cond, expr) afterNCyclesWith__ltl(n, (cond)
&& !(expr), expr)
```

```
//-------------------------------------------------------------------------
//ltl between cycles
//-------------------------------------------------------------------------

#define cltl(expr) (cycle__u -> (expr))
#define G__cltl(expr) [](cycle__u -> (expr))
#define F__cltl(expr) <>(cycle__u && (expr))
#define U__cltl(expr1, expr2) (cycle__u -> (expr1)) U (cycle__u && (expr2))
#define W__cltl(expr1, expr2) (cycle__u -> (expr1)) W (cycle__u && (expr2))
#define V__cltl(expr1, expr2) (cycle__u && (expr1)) V (cycle__u -> (expr2))

#define next__cltl(expr) (cycle__u -> afterCycle__ltl(expr))
#define afterNWith__cltl(n, cond, expr) (cycle__u -> afterNCyclesWith__ltl(n, cond, expr))
#define afterNOrSoonerWith__cltl(n, cond, expr) (cycle__u -> afterNCyclesOrSoonerWith__ltl(n, cond, expr))

ltl r1 {G__cltl( !batteryBroken )}
ltl r2 {G__cltl(!( generatorOn__0 & consumerOn__0 ))}
```

Изменения программы на poST:

1. В состоянии energyLack процесса Controller программы EnergyProvider заменить строчку “consumerOn := FALSE;” на “generatorOn := FALSE;”,
2. В состоянии energyOverflow процесса Controller программы EnergyProvider заменить строчку “generatorOn := FALSE;” на “consumerOn := FALSE;”,
3. В программе EnergyProvider установить значение константы LACK_THRESHOLD равным 5.