Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра Системного Программирования

Копытов Дмитрий Сергеевич

Верификация диаграмм по методу Model Checking в QReal

Бакалаврская работа

Допущен к защите.

Зав. кафедрой:

д. ф.м. н., профессор Терехов А.Н.

Научный руководитель:

ст. преподаватель Кириленко Я.А.

Рецензент:

ст. преподаватель Брыксин Т.А.

Санкт-Петербург

2015

SAINT-PETERSBURG STATE UNIVERSITY

Department of Software Engineering

Dmitry Kopytov

Model checking in QReal DSM Platform

Bachelor's Thesis

Admitted for defence.
Head of the chair:
Professor Terekhov A.N.

Scientific supervisor:
Senior Lecturer Kirilenko I.A.

Reviewer:
Senior Lecturer Bryksin T.A.

Saint-Petersburg
2015

Оглавление

Введение

QReal[1] - metaCASE система, разрабатываемая на кафедре системного программирования математико-механического факультета СПбГУ. На базе неё создана система визуального программирования роботов TRIK Studio[2]. Среда позволяет создавать и исполнять графические программы для различных роботических платформ. К ним относятся: Lego Mindstorms[3], TRIK[4]. Среда предоставляет графический язык программирования, благодаря чему используется для обучения школьников и студентов основам кибернетики и программирования.

Несмотря на простоту использования среды, её возможности позволяют программировать довольно сложные поведенческие алгоритмы роботов и групп роботов. В таких случаях программы подвержены большому количеству нетривиальных ошибок, особенно когда речь идёт о протоколах взаимодействия между группами роботов, либо содержащих большое количество взаимодействующих потоков. Хорошим примером здесь может служить протокол передачи данных Линча [14], корректный на первый взгляд, но содержащий ряд нетривиальных ошибок.

Ситуация усложняется ещё и тем что отладка на реальном устройстве зачастую затруднена или даже дорогостоящая, поэтому полезен был бы инструмент, позволяющий отыскать нетривиальные ошибки в протоколах взаимодействия роботов или потоков. К счастью, в индустрии пару десятилетий уже существует подход к решению данной проблемы ещё на этапе разработки. Речь идёт о методе автоматизированной верификации под названием Model Checking [7][10]. Метод основан на построении конечной модели системы и проверки выполнения на данной модели требуемого свойство.

[1] http://qreal.ru/
[2] http://robots.qreal.ru/
[3] http://mindstorms.lego.com/
[4] http://www.trikset.com/

Постановка задачи

В настоящее время в разработке находится новый визуальный язык, позволяющий описывать асинхронное взаимодействие как процессов внутри робота, так и между роботами. При таком взаимодействии устройств важное значение имеет отслеживание корректности протоколов взаимодействия вычислительных устройств между собой - отлаживать такого рода ошибки чрезвычайно непросто. Целью данной работы является исследование применимости подходов к автоматизированной верификации моделей, создаваемых в TRIK Studio, методами Model Checking, и внедрение одного из них в среду TRIK Studio.

Для достижения это цели были поставлены следующие задачи:

- исследовать существующие системы верификации, использующие данный метод;
- на основе полученных данных выбрать наиболее подходящую;
- внедрить выбранную систему верификации в среду;
- апробировать.

Глава 1. Обзор существующих решений

Существует большое количество систем верификации, использующих метод Model Checking. Среди них имеют как коммерческие, так и свободно распространяемые с открытым исходным кодом. Выбор осуществлялся из последних, поскольку TRIK Studio свободно распространяемый инструмент. В рамках данном работы были рассмотрены следующие наиболее зрелые системы верификации:

- UPPAAL [13]
- SPIN [11]
- NuSMV [6]
- NuXMV [5]

Прежде чем приступить к рассмотрению систем верификации, перечислим требования, которым должен удовлетворять выбранный верификатор. Выбранный верификатор должен уметь предоставлять траекторию исполнения кода, нарушающую проверяемое свойство (контрпример). Перед тем как сформулировать следующее требование, рассмотрим саму среду TRIK Studio.

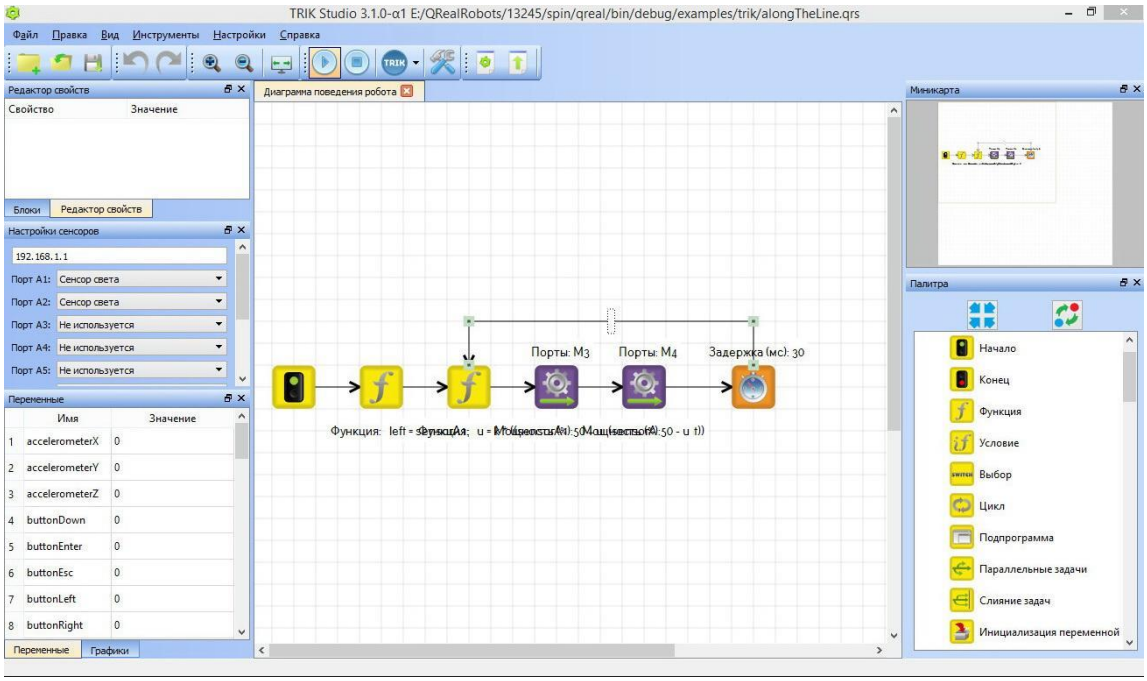


Рис. 1. Интерфейс среды TRIK Studio и небольшой пример программы.

Программа в TRIK Studio представляется в виде последовательности графических блоков, соединённых стрелками потока управления, объединяющихся в диаграммы. Каждой программе соответствует одна или несколько диаграмм. В зависимости от конкретной робототехнической платформы, доступен определённый набор блоков.

Программа может состоять не только из одного потока исполнения. Имеется специальный блок для разделения выполнения программы на несколько потоков. Каждому потоку дается свой идентификатор, он используется при обмене сообщениями между потоками. Для этой цели существует два блока: один для получения сообщения от другого потока, другой для передачи сообщения другому потоку. Это все происходит синхронно, т.е поток-отправитель блокируется до тех пор, пока его сообщение не будет доставлено, аналогично для потока-адресата. Данный механизм носит название рандеву [1].

В связи этим, ещё одним требованием к выбранному верификатору является наличие средств для моделирования синхронного взаимодействия. Так же не стоит забывать про разрабатываемый язык. Поэтому следующим требованием является наличие у верификатора средств для описания асинхронного взаимодействия.

UPPAAL

UPPAAL - среда для моделирования и верификации систем реального времени. Подходит для верификации синхронных систем с общими часами. Состоит из двух главных частей: пользовательского интерфейса и верификатора. В качестве модели UPPAAL использует сети *временных автоматов*. Каждый автомат описывает свой процесс. Такие автоматы позволяют использовать часы. Показания всех часов изменяются на одинаковые величины со временем. Синхронизация процессов происходит с

помощью рандеву каналов, либо посредством разделяемых переменных. Модель описывается с помощью графического редактора (см. Рис. 2).

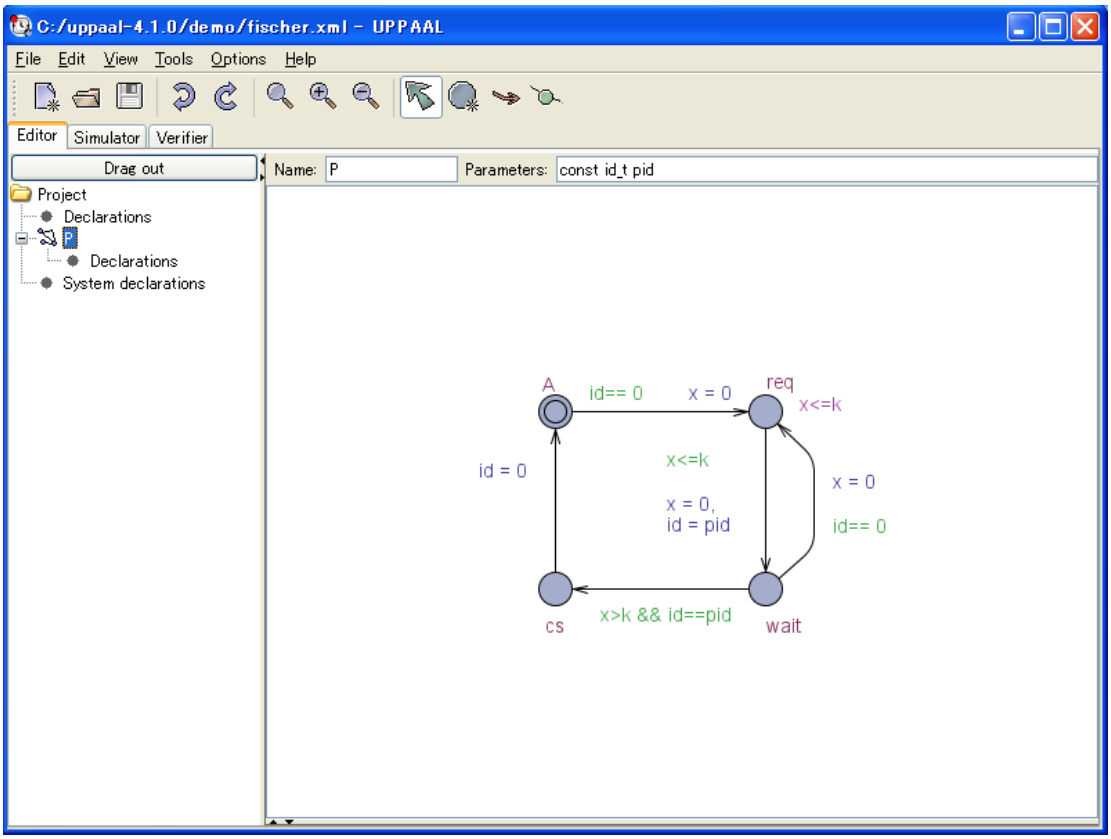


Рис. 2. Графический редактор UPPAAL

UPPAAL позволяет пользователю описывать и использовать при описании модели следующие объекты:

- целочисленные переменные (с ограниченным набором значений);
- часы;
- каналы;
- массивы (статические);
- записи (конструкция struct, как в языке C);
- пользовательские типы из базовых типов, в том числе записей (C-подобный typedef);
- пользовательские функции.

Свойства, проверяемые у модели, могут быть выражены только с помощью одноуровневых темпоральных формул логики CTL[5]. Несмотря на

[5] Computation Tree Logic [8]

это ограничение, можно выразить такие свойства систем, как безопасность, ограниченная живучесть и достижимость. Имеется два варианта запуска верификатора: из графической среды, либо автономно из командной строки. В случае нарушения проверяемого свойства верификатор выдаёт контрпример. Так же имеется встроенный симулятор (см. Рис. 3), на котором можно воспроизвести контрпример, ведущий к состоянию, не удовлетворяющему проверяемому свойству.

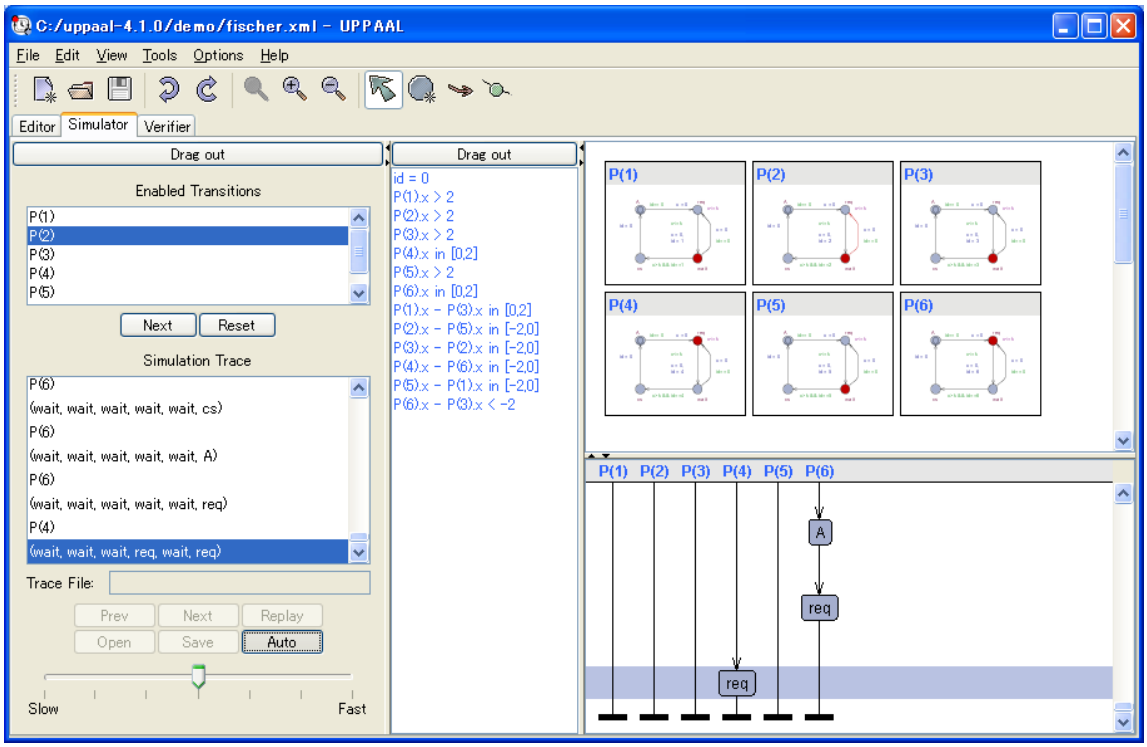


Рис. 3. Симулятор UPPAAL.

SPIN

Spin - инструмент для моделирования и автоматизированной проверки параллельных и распределённых систем, сфокусирован на проверке корректности межпроцессного взаимодействия. Для описания модели используется недетерминированный язык Promela[6] (т.е. язык позволяет задать в определённых точках программы различные альтернативы для программного потока), синтаксис которого схож с синтаксисом языка C. Promela содержит в себе необходимые примитивы для создания процессов и описания межпроцессного взаимодействия. Поддерживается динамическое создание процессов. Межпроцессное взаимодействие может быть описано посредством рандеву каналов, асинхронных каналов с буфером, разделяемых переменных или комбинации выше перечисленных.

При создании модели язык Promela позволяет описывать и использовать следующие сущности:

- целочисленные типы (int, short, byte);
- логический тип (bool или bit);
- структуры (struct);
- одномерные массивы (многомерные с помощью структур);
- inline определения;
- каналы (рандеву, асинхронные с буфером);
- символьные имена для константных значений (mtype, либо #define).

[6] Process Meta Language, URL: http://spinroot.com/spin/Man/Manual.html

```
bool want[2];
bool turn;
byte cnt;

proctype P(bool i)
{
  want[i] = 1;
  do
  :: (turn != i) ->
          (!want[1-i]);
          turn = i
  :: (turn == i) ->
          break
  od;
  cnt = cnt+1;
  skip;   /* critical section */
  cnt = cnt-1;
  want[i] = 0
}

proctype monitor()
{
  assert(cnt == 0 || cnt == 1)
}

init {
  run P(0); run P(1); run monitor()
}
```

Листинг 1. Пример кода на языке Promela

Однако Promela не позволяет описывать функции и нет понятия времени или часов. Поэтому нет встроенных средств для описания систем с общими часами.

Для описания проверяемых свойств используется формулы темпоральной логики LTL[7], либо можно использовать assert для задания простых свойств безопасности. В случае нарушения проверяемого свойства Spin выдаёт контрпример в виде трассы выполнения.

[7] Linear Temporal Logic [16]

NuSMV

NuSMV - инструмент для верификации синхронных и асинхронных систем. Начиная с версии 2.5.0 прекратилась поддержка верификации асинхронных систем. Поскольку ядро NuSMV заточено под верификацию синхронных систем, разработчики решили отказаться от поддержки асинхронных процессов. NuSMV это переработка и расширение старого верификатора SMV [15] - первого Model Checking верификатора, базирующегося на BDD[8]. Для описания модели используется тот же встроенный язык, что и у SMV. Нет понятия рандеву синхронизации. Синхронизация процессов происходит посредством разделяемых переменных.

Так же как и SMV, NuSMV поддерживает следующие типы данных:

- логический тип (bool);
- битовые вектора;
- константы;
- статические массивы базовых типов.

Проверяемые свойства описываются с помощью темпоральной логики CTL и LTL (сведением к CTL согласно алгоритму [9]).

NuXMV

NuXMV - расширение NuSMV. Добавлены более продвинутые алгоритмы для верификации систем с конечным и бесконечным числом состояний. Поддерживает верификацию только синхронных систем. Данный верификатор не был более детально рассмотрен, поскольку использует туже парадигму, что и NuSMV, и отличается от него лишь использованием более продвинутых алгоритмов.

[8] Binary Decision Diagram [3]

Выбор системы верификации

Среди рассмотренных систем верификации наиболее подходящими по своей функциональности оказались Spin и UPPAAL, поскольку оба поддерживают синхронизацию процессов посредством рандеву-каналов, что удобнее для моделирования, в отличие от разделяемых переменных, как это сделано в NuXMV и NuSMV.

Окончательный выбор пал на Spin, поскольку он имеет средства для моделирование асинхронного взаимодействия - асинхронные каналы с буфером. Для текущего языка TRIK Studio это не так важно, но понадобится для разрабатываемого языка.

Глава 2. Реализация

Для поддержки верификации в TRIK Studio необходимо реализовать следующие модули:

- генератор для трансляции диаграмм в код на языке Promela;
- связь блоков диаграммы с кодом.

Генератор

Модель, подающаяся на вход верификатора SPIN, описывается с помощью языка Promela. Для трансляции диаграмм в код на языке Promela был реализован генератор кода. Он был сделан на основе уже имеющегося в TRIK Studio инструментария для быстрого создания генераторов кода из диаграмм.

Процесс генерации в фреймворке разделён на два этапа. На первом этапе генератор потока управления преобразует диаграмму в семантическое дерево. Данный этап одинаков для всех генераторов кода. Следующий этап - печать получившегося семантического дерева код. Эта часть реализуется для каждого генератора по разному. Каждый генератор целевого языка с помощью генераторов кода для конкретных узлов дерева и шаблонов кода транслирует дерево, полученное на предыдущем этапе, в код.

Ниже перечислены некоторые особенности генератора, про которые далее будет более подробно рассказано:

- потоки и подпрограммы:
- типы данных;
- сенсоры;
- таймеры;
- приём и передача сообщений потоками.

Потоки и подпрограммы

Программа в TRIK Studio может состоять не только из одного потока, имеется блок “Параллельные задачи” (см. Рис. 4), позволяющий разделять

поток на несколько. Каждому потоку даётся имя, которое в дальнейшем можно использовать при общении между потоками. Кроме этого, имеется возможность создавать подпрограммы без входных параметров.

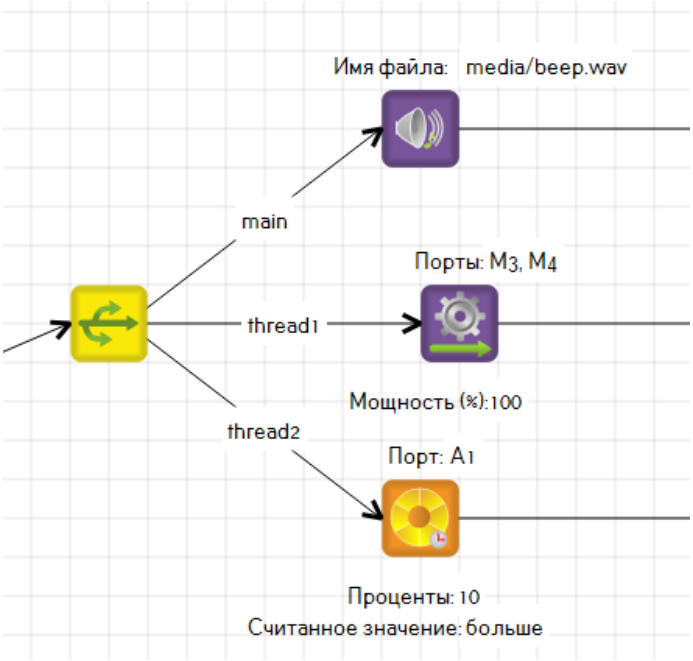


Рис. 4. Блок “Параллельные задачи”. На исходящих стрелках указаны имена потоков.

Для реализации потоков используется конструкция proctype языка Promela для описания процессов. Каждый процесс может создавать другие процессы посредством унарного оператора run. Подпрограммы же транслируются в inline-определени языка Promela (см. Листинг 2).

```
inline subprogram()
{
    ...
}

proctype thread1()
{
    ...
    subprogram();
    ...
}
```

```
proctype thread2()
{
    ...
}

proctype main()
{
    run thread1();
    run thread2();
    ...
}

init
{
    run main();
}
```

Листинг 2. Представление потоков и подпрограммы на языке Promela.

Типы данных

Программы в TRIK Studio поддерживают следующие типы данных:

- логический тип;
- целые числа;
- вещественные числа;
- строки;
- массивы.

Все вышеперечисленные типы данных кроме вещественных чисел имеются и в языке Promela или могут быть представлены с помощью него. При трансляции в язык Promela конструкции, использующие вещественный тип данных либо игнорируются (т.е. заменяются на оператор skip, который ничего не делает), либо становятся недетерминированными, а именно транслируются в недетерминированный оператор if. Оператор if языка Promela похож на оператор множественного выбора switch, с тем отличием, что вместо возможных значений выражения - стражи. Страж это выражение, при ложном значении которого, процесс блокируется, до тех пор пока оно не примет истинное значение. В случае с оператором if ложность одного из стражей не блокирует процесс, для блокировки процесса нужна ложность все стражей. При истинности только одного стража, оператор if ведёт себя как оператор switch. Если больше одного стража истинны, происходит недетерминированный выбор одной из веток.

К конструкциям, которые могут игнорироваться, относятся:

- присваивания (блоки “Инициализация переменной” и “Функция”);
- приём/передача сообщений (блоки “Отправить сообщение в задачу” и “Получить сообщение из другой задачи”).

Конструкции, которые могут стать недетерминированными:

- оператор if (блок “Условие”, см. Рис. 5);
- оператор switch (блок “Выбор”);
- цикл for (блок “Цикл”);
- цикл с предусловием while.

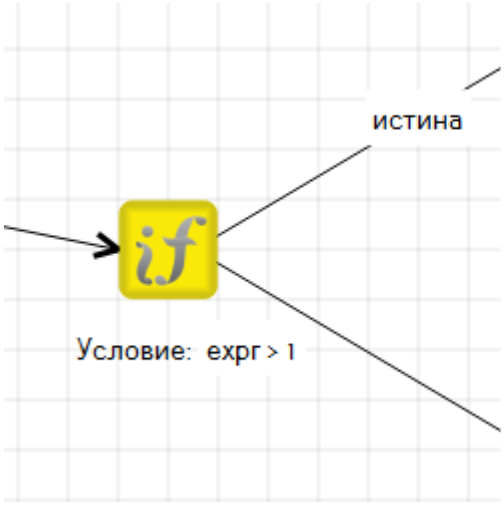


Рис. 5. Пример недетерминированного условия, где expr - выражение вещественного типа.

Строки в языке Promela не поддерживаются, но их можно реализовать с помощью целочисленных массивов. Поскольку Promela не поддерживает динамические типы данных, то размер массивов постоянен. Для константных строк выделяются отдельные переменные вида CONST_STRINGX, где X - число (см. Листинг 3).

```
typedef string { int size; int a[n] };
...
string CONST_STRING0;
...
inline initialization()
{
    d_step {
        /*dsd*/
        CONST_STRING0.size = 3;
        CONST_STRING0.a[0] = 100;
        CONST_STRING0.a[1] = 115;
        CONST_STRING0.a[2] = 100;
    }
}
...
init
{
    initialization();
    ...
}
```

Листинг 3. Представление константных строк в языке Promela.

Из-за того, что язык Promela не поддерживает функции, такие операции, как сравнение, конкатенация строк реализуются посредством inline определений (см. Листинг 4). Результат операции сравнения сохраняется в глобальную переменную. Чтобы предотвратить одновременное использование операции сравнения в нескольких потоках (иначе результат какого-то сравнения может затереться результатом другого), последовательность операторов помечается как атомарная (d_step, либо atomic). Операторы, перечисленные в атомарной последовательности, выполняются за один шаг, т.е. ни один поток не может прервать исполнение этой последовательности.

```
bool ifexpr1;

d_step {
    compareStr(str1, str2);
    ifexpr1 = compare_res;
}

if
:: (ifexpr1 == true) ->
    ...
:: else ->
    ...
fi;
```

Листинг 4. Использования сравнения строк в условном операторе (compare_res - глобальная переменная).

Сенсоры

Диаграммы в TRIK Studio могут содержать блоки ожидания показаний с сенсоров, а так же использовать показания с сенсоров. Диапазон значений большинства сенсоров слишком велик, что представляет сложность при моделировании на языке Promela. Исключения составляют сенсоры с бинарным набором значений, например, сенсор касания. В связи с этим блоки ожидания показаний с сенсоров транслируются в код на языке Promela в виде недетерминированного цикла, т.е. решение выйти из цикла происходит недетерминированно. Для показаний с сенсоров применяется тот

же подход, что и для вещественных выражений. Что касается показаний сенсоров с бинарным набором значений (к ним отнесём ещё кнопки на самом роботе, на пульте дистанционного управления и т.п.), то для них можно завести процесс-демон, который будет недетерминированно менять значения этих сенсоров. Блоки ожидания для таких сенсоров представляются в виде стражей.

```
//недетерминированный цикл
do
:: skip;
:: break;
od;

//страж
(keyDown == pressed);
```

```
mtype = {
         pressed, not_pressed,
         ...
        }
mtype keyDown = not_pressed;

...

//процесс-демон
proctype daemon()
{
    do
    :: timeout ->
        d_step {
            if
            ::keyDown = pressed;
            ::keyDown = not_pressed;
            fi;
            ...
        }
    od;
}
```

Листинг 5. Слева вариант кода для блоков ожидания показаний с сенсоров. Справ процесс-демон для бинарных сенсоров.

Таймеры

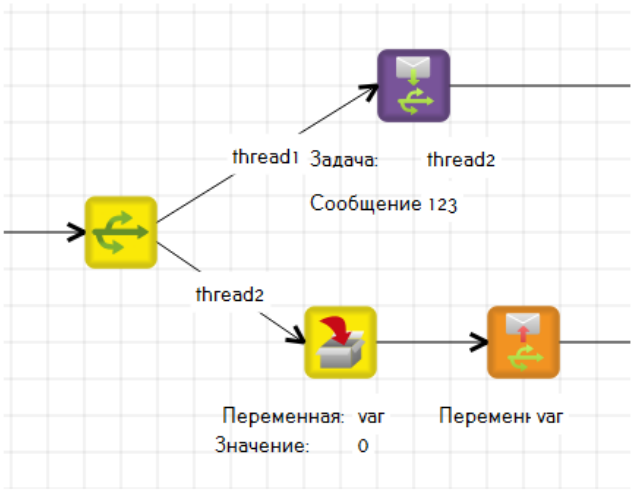
В языке Promela нет средств для описания временны*х* ограничений. Были попытки расширения языка (Discrete Time Promela [4], Real Time Promela [17], Translation of Timed Promela to Timed Automata with Discrete Data [12]), но на данный момент поддержки описания временных ограничений в языке нет.

В связи с этим фактом при трансляции в язык Promela, блоки “Таймер” игнорируются, т.е заменяются на оператор skip.

Приём и передача сообщений потоками

Общение между потоками происходит посредством передачи сообщений, причём это происходит синхронно. Для моделирования этого взаимодействия используются рандеву-каналы языка Promela (см. Рис. 6).

Каждый поток имеет один канал и идентификатор (именованная целочисленная константа). Посылаемые сообщения состоят из двух частей: имя отправителя и данные. Имя отправителя - идентификатор потока, данные - целочисленный массив. Поскольку Promela не позволяет в качестве передаваемых данных явно указывать массив, то массив оборачивается в структуру.




```
//идентификаторы потоков
#define thread1proc 0
#define thread2proc 1

typedef message { int size; int a[n] };

//рандеву-каналы потоков
chan thread1chan = [0] of {int, message};
chan thread2chan = [0] of {int, message};

int var;

proctype thread1()
{
    message temp;

    d_step {
        temp.size = 1;
        temp.a[0] = 123;
    }
    //передача сообщения потоку thread2
    thread2 ! thread1proc(temp);
    ...
}

proctype thread2()
{
    message temp;
    var = 0;
    //приём сообщения от потока thread1
    thread2 ? thread1proc(temp);
    var = temp.a[0];
    ...
}
...
```

Рис. 6. Передача и приём сообщений потоками на языке Promela. Слева, часть диаграммы. Справа, код в который она транслируется.

Для того, чтобы передавать по каналам строки, каждому потоку добавляется по одному буферу строк. Этот буфер используется следующим образом. Поток-отправитель кладёт в свой буфер строки, которые он собирается передать потоку-адресату, и в качестве данных передает количество строк. Поток-адресат, приняв сообщения, считывает указанное количество строк в сообщении из буфера потока-отправителя (см. Листинг 6).

В некоторых ситуациях поток-отправитель может затереть буфер с только что переданными строками, например, если сразу же после отправки отправляет строки ещё кому-нибудь. Для избежания такой ситуации последовательность операторов приёма и считывания данных из буфера помечается как атомарная. В данном случае с помощью atomic, поскольку, в отличие от d_step, он позволяет использование операторов отправки и приёма сообщений.

```
//буфер строк потока thread1
string thread1_buffer[m];
chan thread1chan = [0] of {int, message};
chan thread2chan = [0] of {int, message};

string var;

proctype thread1()
{
    message temp;

    d_step {
        temp.size = 1;
        copyStr(thread1_buffer[0], CONST_STRING1);
    }
    //передача сообщения потоку thread2
    thread2 ! thread1proc(temp);
    ...
}

proctype thread2()
{
    message temp;
    //приём сообщения от потока thread1
    atomic {
        thread2 ? thread1proc(temp);
        copyStr(var, thread1_buffer[0]);
    }
    ...
}
...
```

Листинг 6. Передача и приём строки потоками на языке Promela.

Связь блоков диаграммы с кодом

В случае нарушения проверяемых свойств верификатор Spin даёт на выходе trail-файл с траекторией исполнения программы, которая ведёт к нарушению свойства. Полученный trail-файл с помощью средств Spin можно представить в более понятном виде, а именно в виде последовательности номеров исполняемых строк кода на языке Promela. Рядом с номером строки указывается имя процесса, который в данной точке выполняется. Также в конце выведен список всех переменных со значениями в конце траектории.

Для того, чтобы подсветить на диаграмме траекторию, ведущую к нарушению проверяемого свойства, был реализован модуль,

осуществляющий двунаправленную связь блоков диаграммы с кодом (т.е по данному блоку можно получить интервал номеров строк кода, в которые блок был транслирован, и наоборот, по номеру строки кода можно получить блок). Кроме кода на языке Promela генератор создаёт ещё файл с дополнительной информацией о связи различных частей кода с блоками диаграммы. Информация хранится в следующем формате:

<идентификатор блока>@<начало блока кода>@<конец блока кода>

Кроме того данный модуль это отдельный результат, с помощью него можно реализовать совместную интерпретацию кода и модели, что не является частью данной работы, но полезно для платформы QReal в целом.

Глава 3. Апробация

Для апробации всей системы был выбран протокол передачи данных Линча [14]. Цель протокола - полнодуплексная передача последовательности символов (т.е. одновременная передача в обоих направлениях) через ненадёжную среду между двумя устройствами.

Передаваемое сообщение состоит из заголовка и данных. Заголовок принимает одно из трёх значений:

- ack (положительное подтверждение);
- nack (отрицательное подтверждение);
- err (ошибка).

Канал может искажать сообщения, но не может терять, дублировать или создавать новые сообщения. Все ошибки обнаруживаются на нижнем уровне протокола, верхний в этом случае получает err в заголовке. Правила функционирования протокола следующие:

1. если полученное сообщение не содержало ошибок, то в ответном сообщении посылается ack, иначе nack;

2. если полученное сообщение содержало nack или err в заголовке, то в ответ посылаются старые данные, иначе посылаются следующие данные.

Формальное представление этих правил в виде диаграммы SDL дано на рис. 7.

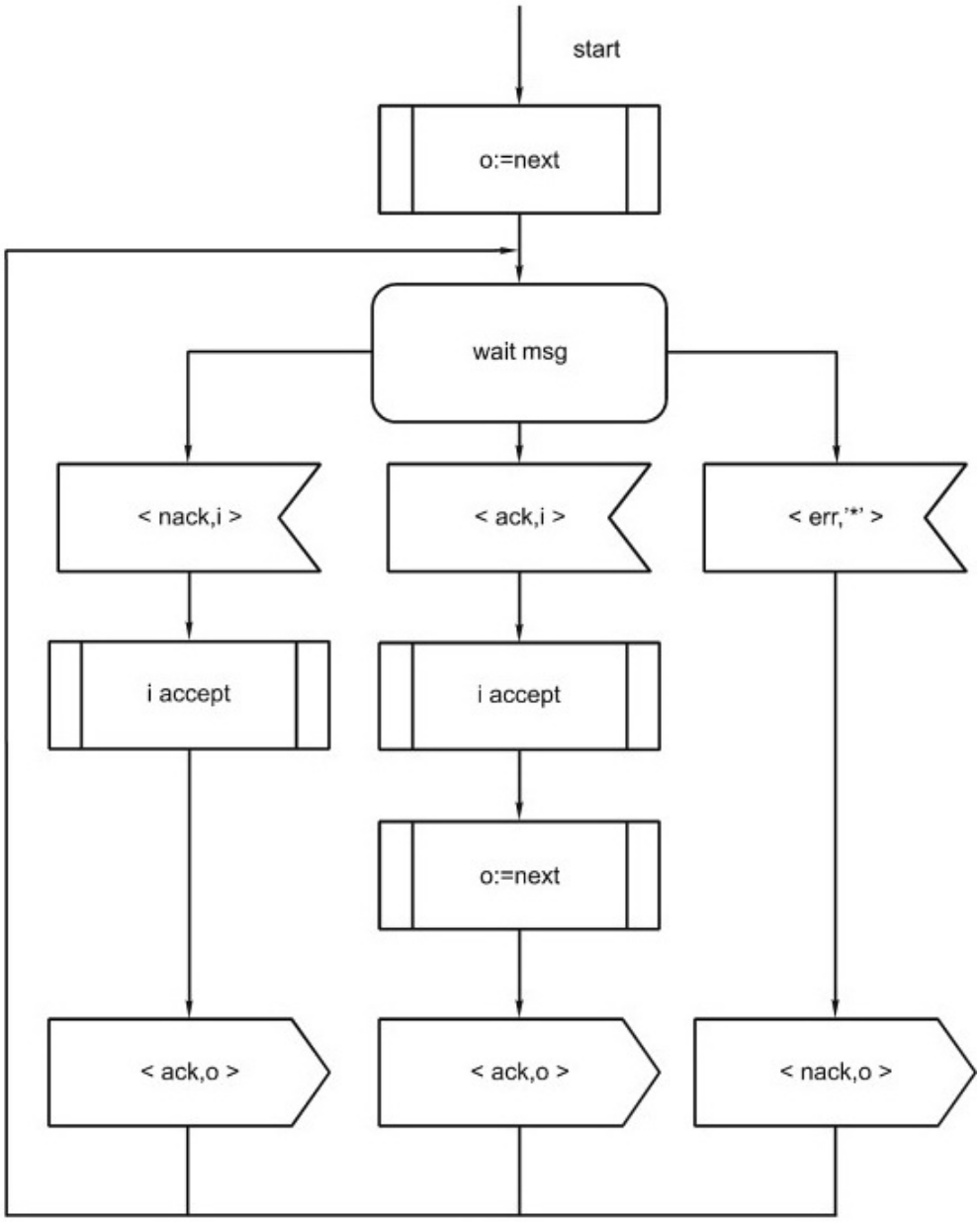


Рис. 7. Правила функционирования протокола [2, с. 22].

При некоторых редко встречающихся сценариях протокол Линча может принять дубликат предыдущих данных как очередную информацию. Один из таких сценариев приведён на рис. 8. Двойная повторяющаяся ошибка,

искажающая как положительное подтверждение от A к B, так и отрицательное подтверждение от B к A, приводит к дублированию принятия символа ‘m’ устройством A.

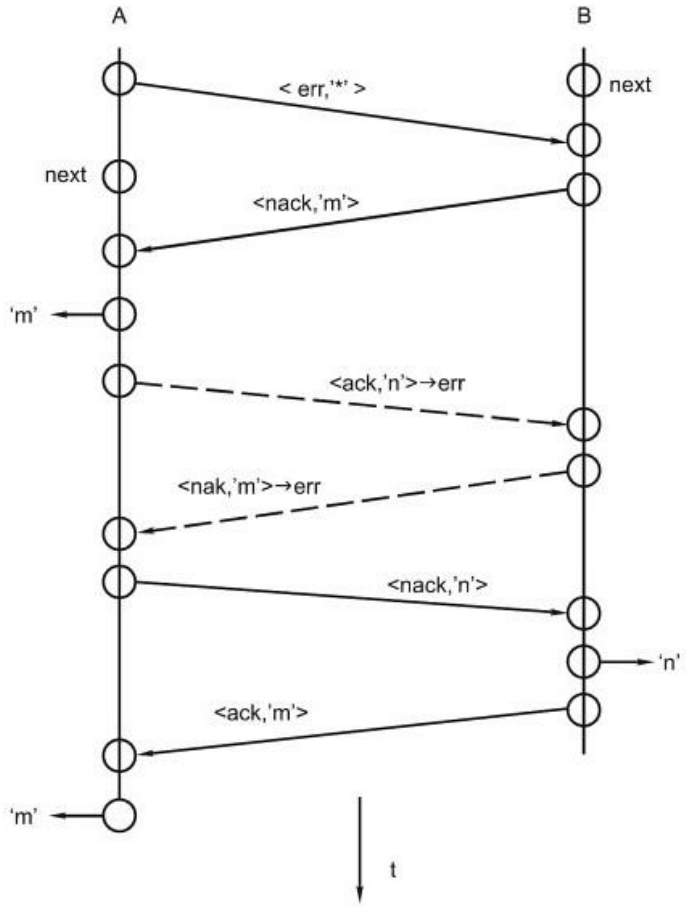


Рис. 8. Дублирование данных протоколом Линча [2, с. 23].

Реализация в TRIK Studio

Протокол Линча был реализован на TRIK Studio. Главная диаграмма изображена на рис. 9. Имеется: два потока, реализующие устройства, которые обмениваются сообщениями (transfer1 и transfer2); два потока, от которых устройства получают очередные данные (send1 и send2); два потока, реализующие ненадёжные каналы (buggy_chan1 и buggy_chan2); один поток, которому отсылаются принятые данные первым устройством (receive1). Потоки transfer1 и transfer2 обмениваются сообщениями между собой через потоки buggy_chan2 и buggy_chan1 соответственно.

На рис. 10 изображена диаграмма подпрограммы transfer1, реализующей первое устройство. Для второго устройства диаграмма выглядит аналогично за исключением потоков, через которые получаются и отправляются данные. На листинге 7 представлен код на языке Promela, в который транслируется подпрограмма transfer1.

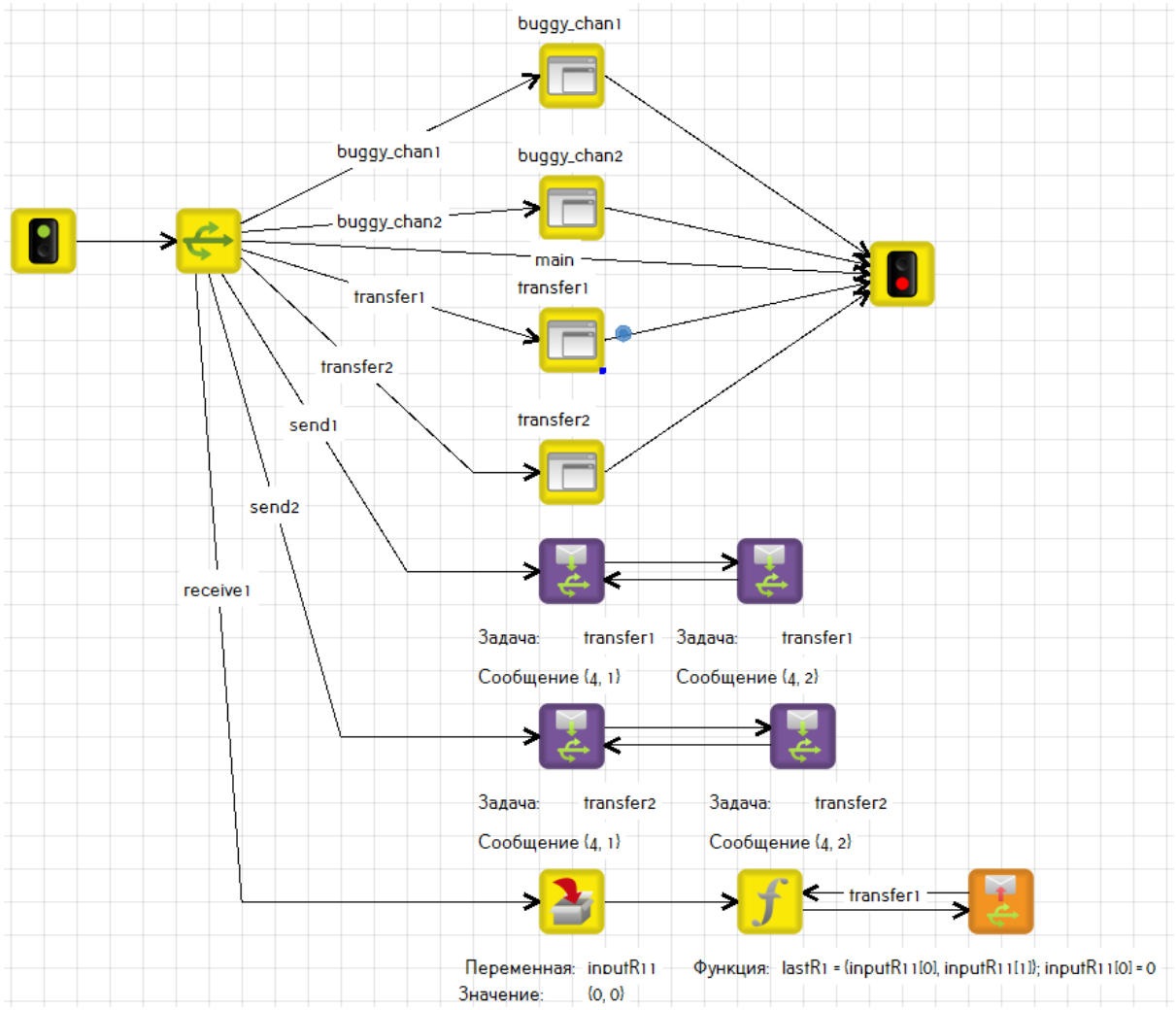


Рис. 9. Главная диаграмма реализации протокола Линча на TRIK Studio

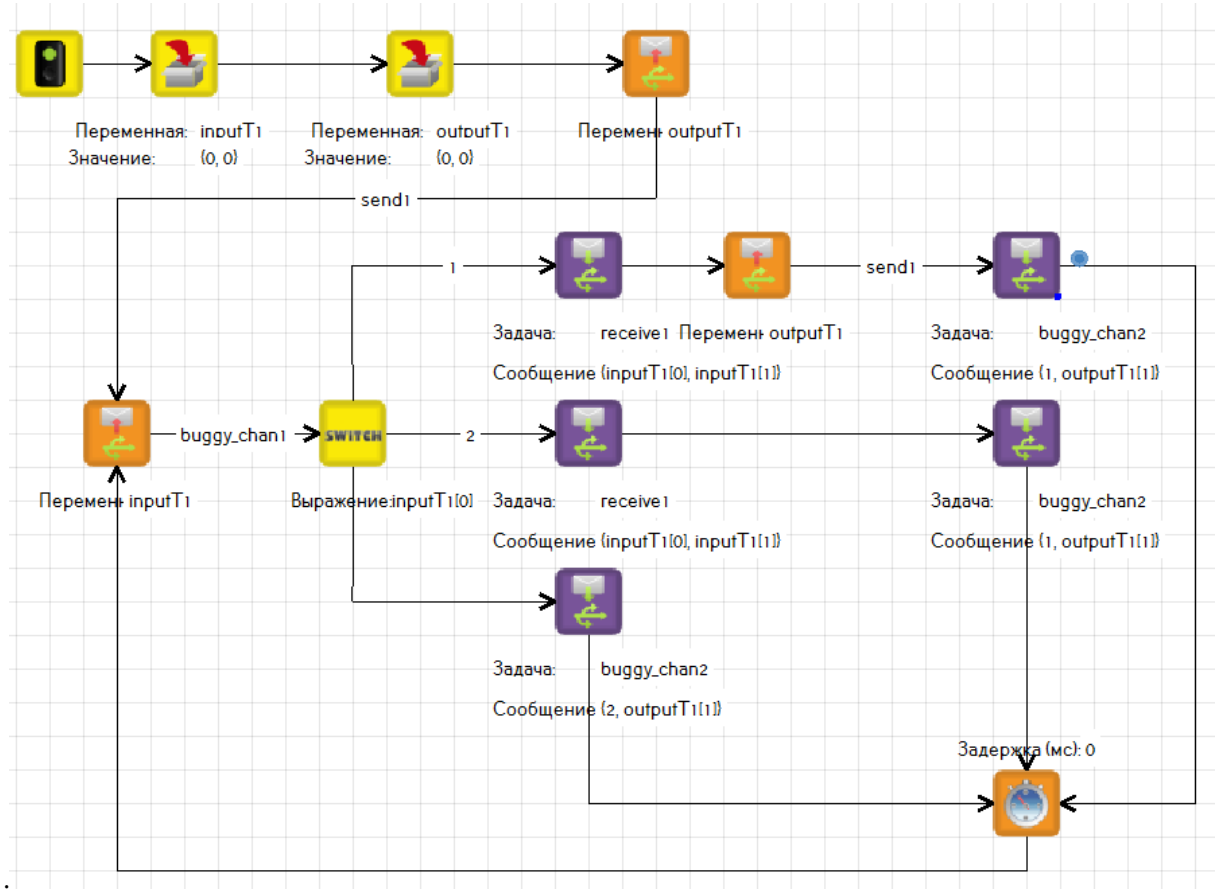


Рис. 10. Подпрограмма transfer1

```
inline transfer1()
{
    d_step {
        inputT1.size = 2;
        inputT1.a[0] = 0;
        inputT1.a[1] = 0;
    };
    d_step {
        outputT1.size = 2;
        outputT1.a[0] = 0;
        outputT1.a[1] = 0;
    };
    transfer1chan?send1proc(temp);
    d_step {
        int receive_i;
        for (receive_i : 0 .. temp.size - 1) {
            outputT1.a[receive_i] = temp.a[receive_i];
        }
        outputT1.size = temp.size;
    };
    do /*infinite cycle*/
    ::
        transfer1chan?buggy_chan1proc(temp);
        d_step {
            int receive_i;
            for (receive_i : 0 .. temp.size - 1) {
                inputT1.a[receive_i] = temp.a[receive_i];
            }
            inputT1.size = temp.size;
        };
        if
        :: (inputT1.a[0] == 2) ->
            d_step {
                temp.size = 2;
                temp.a[0] = inputT1.a[0];
                temp.a[1] = inputT1.a[1];
            };
            receive1chan!transfer1proc(temp);
            d_step {
                temp.size = 2;
                temp.a[0] = 1;
                temp.a[1] = outputT1.a[1];
            };
            buggy_chan2chan!transfer1proc(temp);
        :: (inputT1.a[0] == 1) ->
            d_step {
                temp.size = 2;
                temp.a[0] = inputT1.a[0];
                temp.a[1] = inputT1.a[1];
            };
            receive1chan!transfer1proc(temp);
            transfer1chan?send1proc(temp);
            d_step {
                int receive_i;
                for (receive_i : 0 .. temp.size - 1) {
                    outputT1.a[receive_i] = temp.a[receive_i];
                }
                outputT1.size = temp.size;
            };
            d_step {
                temp.size = 2;
                temp.a[0] = 1;
                temp.a[1] = outputT1.a[1];
            };
            buggy_chan2chan!transfer1proc(temp);
        ::else ->
            d_step {
                temp.size = 2;
                temp.a[0] = 2;
                temp.a[1] = outputT1.a[1];
            };
            buggy_chan2chan!transfer1proc(temp);
        fi;
        skip;/*wait 0*/
    od;
}
```

Листинг 7. Подпрограмма transfer1 на языке Promela

На рис. 11 изображена диаграмма подпрограммы buggy_chan1, реализующей ненадёжный канал, через который второе устройство передаёт

сообщения первому устройству. Диаграмма второго канала (buggy_chan2) выглядит аналогично за исключением потоков, через которые получаются и отправляются данные. Для моделирование ошибки в канале, было использовано свойство генератора, касающееся конструкций, использующих показания с сенсоров, а именно трансляция их в недетерминированный выбор. В данном случае используется показание интенсивности красного цвета с сенсора цвета (для этого используется выделенная переменная colorSensorR). При трансляции в код на языке Promela блок “Условие” транслируется в недетерминированный оператор if.

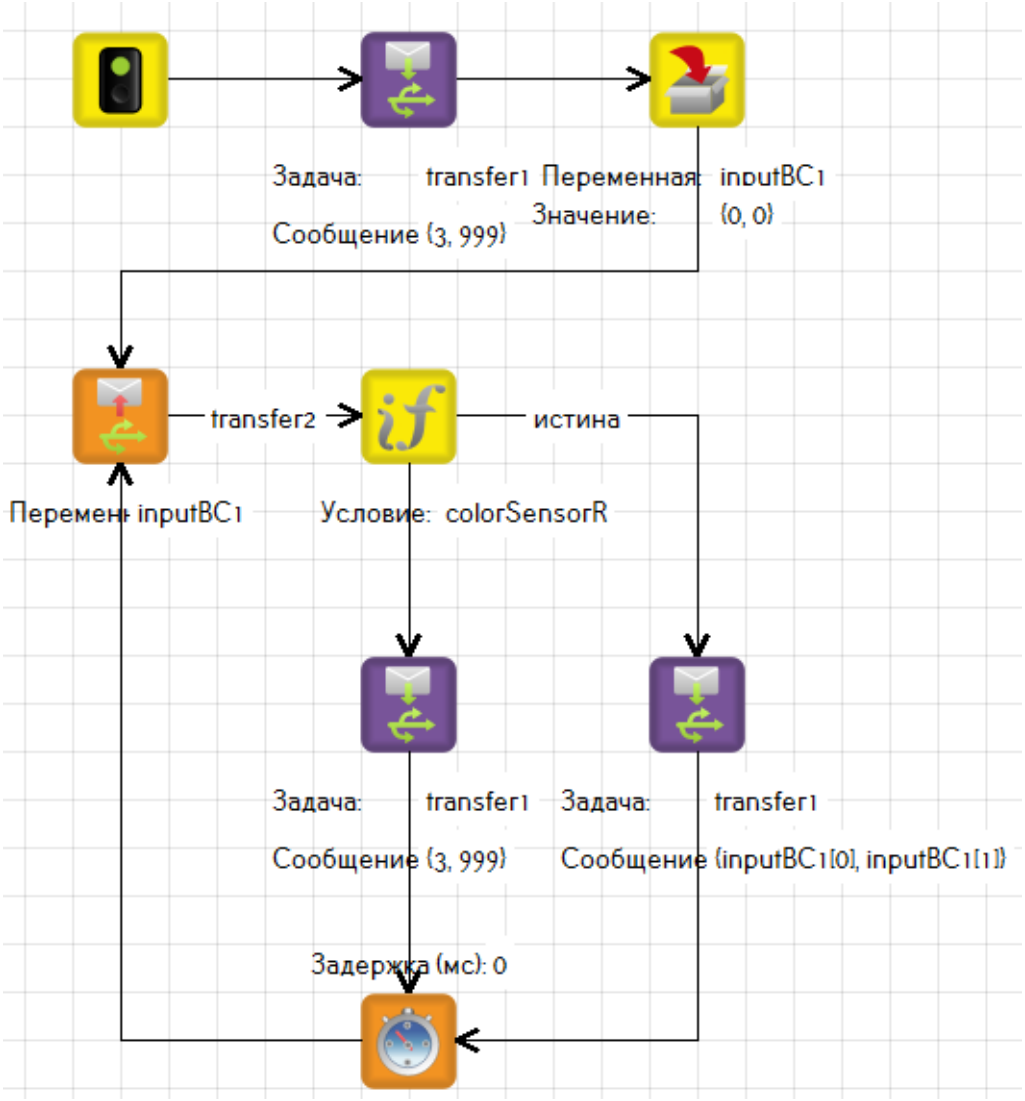


Рис. 11. Подпрограмма buggy_chan1

Проверяемое свойство

Реализованный протокол Линча был верифицирован на удовлетворение следующему свойству: “Первое устройство никогда не получит дубликат предыдущих данных как очередную информацию”. Для того, чтобы это проверить был введён дополнительный поток (receive1), который получает все приходящие первому устройству (transfer1) сообщения, не содержащие ошибки. LTL формула данного свойства выглядит следующим образом:

[]!(lastR1[0] != 0 && inputR11[0] == 1 && inputR11[1] == lastR1[1]),

где

- lastR1 - предыдущее сообщение (массив, lastR1[0] - заголовок, lastR1[1] - данные) ,
- inputR11 - текущее сообщение (массив, inputR11[0] - заголовок, inputR11[1] - данные),
- lastR1[0] != 0 - заголовок предыдущего сообщения либо 1 (ack), либо 2 (nack),
- inputR11[0] == 1 - заголовок текущего сообщения 1 (ack),
- inputR11[1] == lastR1[1] - полученные данные совпадают с полученными в прошлый раз.

Результат верификации

В результате верификации был найден контрпример, ведущий к нарушению проверяемого свойства. Имеется два режима просмотра контрпримера - пошаговый или в режиме анимации, когда блоки подсвечиваются последовательно друг за другом через заданный промежуток времени. На рис. 12 представлен пошаговый режим. По нажатию кнопки “Next Step” подсвечивается очередной блок из контрпримера, в информационном окне отображается имя потока, исполняемого в данным момент.

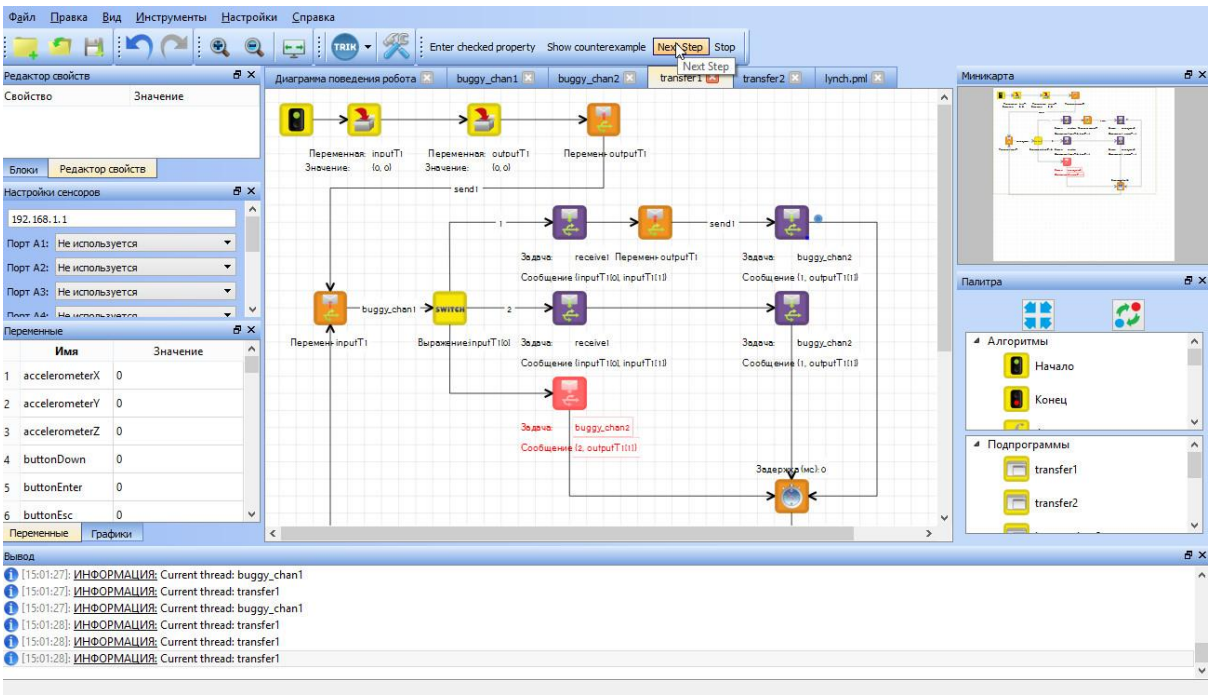


Рис. 12. Отображение контрпримера

Заключение

В рамках данной работы были достигнуты следующие результаты:

- были исследованы существующие системы верификации использующие данный метод;
- на основе полученных данных была выбрана наиболее подходящая система верификации;
- были реализованы необходимые компоненты для внедрения и выбранная система верификации была внедрена в среду TRIK Studio;
- была проведена апробация на протоколе передачи данных Линча.

Список литературы

[1] Гавва А. Е. Адское программирование. Ada-95. Компилятор GNAT [Электронный ресурс] / А. Е. Гавва // Ada_Ru. 2004. Гл. 15. URL: http://www.ada-ru.org/V-0.4w/index.html

[2] Карпов Ю. Г. Model Checking. Верификация параллельных и распределенных программных систем // СПб.: БХВ-Петербург, 2010.

[3] Akers S.B. Binary Decision Diagrams // IEEE Transactions on Computers. 1978. Vol. C-27(6). P. 509-516

[4] Bosnacki D., Dams D. Discrete-time Promela and Spin // FTRTFT ’98: Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems. 1998. P. 307–310

[5] Cavada R., Cimatti A., Dorigatti M., Griggio A., Mariotti A, Micheli A., Mover S., Roveri M., Tonetta S. The nuXmv symbolic model checker // Proceedings of the 16th

[6] Cimatti A., Clarke E., Giunchiglia E., Giunchiglia F., Pistore M., Roveri M., Sebastiani R., Tacchella A. Nusmv 2: An opensource tool for symbolic model checking // CAV '02 Proceedings of the 14th International Conference on Computer Aided Verification. 2002. P. 359-364

[7] Clarke E. M., Emerson E. A. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic // Proceedings of the Workshop on Logics of Programs. 1982. P. 52-71

[8] Clarke E.M., Emerson E.A., Sistla A.P. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications // ACM Transactions on Programming Languages and Systems. 1986. Vol. 8(2). P. 244-263

[9] Clarke E., Grumberg O., Hamaguchi K. Another Look at LTL Model Checking // Formal Methods in System Design. 1997. Vol. 10(1). P. 57-71

[10] Emerson E. A. The Beginning of Model Checking: A Personal Perspective // 25 Years of Model Checking. 2008. P. 27 - 45

[11] Holzmann G. J. The Model Checker Spin // IEEE Transactions on Software Engineering. 1997. Vol. 23(5). P. 279-295
[12] Janowska A., Janowski P., Wróblewski D. Translation of Timed Promela to Timed Automata with Discrete Data // Fundamenta Informaticae-Concurrency Specification and Programming (CS&P). 2008. Vol. 85(1-4). P. 409-424
[13] Larsen K. G., Pettersson P., Yi W. Model-checking for real-time systems // FCT '95 Proceedings of the 10th International Symposium on Fundamentals of Computation Theory. 1995. P. 62–88
[14] Lynch W.C. Computer Systems: Reliable full-duplex file transmission over half-duplex telephone line // Communications of the ACM. June 1968. Vol. 11(6). P. 407-410
International Conference on Computer Aided Verification. 2014. Vol. 8559. P. 334–342
[15] McMillan K. L. Symbolic model checking: an approach to the state explosion problem // Doctoral Dissertation. 1992
[16] Pnueli A. The temporal logic of programs // SFCS '77 Proceedings of the 18th Annual Symposium on Foundations of Computer Science. 1977. P. 46-57
[17] Tripakis S., Courcoubetis C. Extending Promela and SPIN for real time // TACAs '96 Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems. 1996. P. 329–348