Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

Б.М. Ярцев, А.А. Шалыто

Разработка программного обеспечения роботов *Lego Mindstorms* на основе автоматного подхода (Проект *Isenguard*)

Проектная документация

Проект создан в рамках

“Движения за открытую проектную документацию”

http://is.ifmo.ru

Санкт-Петербург

2005

Оглавление

Введение

Начнем с цитаты из книги Коупленда Д. Рабы Майкрософта. М.: Изд-во АСТ. 2004:

“Вы когда-нибудь замечали, что Lego играет намного большую роль в жизни компьютерщиков, чем в жизни всего остального населения? Сначала все специалисты по компьютерам проводят огромную часть своего детства, погрузившись в Lego по уши, да и, кроме того, это чрезвычайно сосредоточенная и приучающая к одиночеству культура. Игра в Lego – это их общий знаменатель.

Еще можно с уверенностью сказать, что Lego является патентованным, трехмерно моделирующим инструментом и языком сам по себе. А продолжительное воздействия любого языка, визуального или вербального, несомненно, видоизменяет восприятие ребенком окружающего мира. Lego онтологически сродни компьютерам. Чтобы воспользоваться электронной таблицей или создать гоночную машину, вот для чего нам нужны и компьютер, и Lego.

Lego представляет собой бинарную “да-нет” структуру. Маленькие выпуклости на поверхности любого кирпичика либо подходят и присоединяются к другой детали, либо нет. Аналоговых отношений не существует.

Lego предвосхищает будущее самых бредовых идей. Оно цифровое. Весь шарм и веселье от Lego заключается в переводе органического в модульное.

Кстати: вы знали, что Lego выпускает пластиковые пылесосы в форме морковки для сбора закатившихся деталек?”

В 1998 году компанией *Lego* был создан конструктор *Lego Mindstorms,* являющийся одним из расширений базового комплекта *Lego Technics*. Этот конструктор отличается от предыдущих конструкторов этой фирмы наличием различных датчиков (сенсоров) и микроконтроллеров, программируемых пользователем. Каждый комплект этого конструктора содержит определенную комплектацию датчиков и один микроконтроллер.

Сначала конструктор предназначался для детей. Его программирование выполнялось в примитивной визуальной среде, которая позволяла задавать поведение роботов при помощи весьма специфических схем.

Позднее появились энтузиасты (сотрудники различных университетов и IT-фирм), которые создали язык программирования *NQC* (*not quite C*), очень похожий на *C*. При помощи этого языка стало возможным программировать роботов *Lego* традиционным способом. Первоначально программа на этом языке пишется на персональном компьютере, компилируется и загружается в память микроконтроллера через инфракрасный порт. Эта память составляет всего 32 Кб, что ограничивает круг решаемых задач, и, в частности, осложняет решение задачи, рассматриваемой в настоящем проекте.

При этом необходимо отметить, что специализированная операционная система (ОС), которая была разработана для установки на роботе для исполнения программ на языке *NQC* [1], занимала 26 Кб – для прикладных программ памяти практически не оставалось.

После этого на сайте www.sourceforge.net появился проект *legos* [2]. Разработанная там ОС. позволяет писать программы для этого робота на языке *C*. Она занимает 18 Кб. Сейчас этот проект переименован в *BrickOS*, и работа над ним продолжается.

В настоящем проекте в качестве основной ОС была выбрана *leJOS* [3]. Она написана на языке программирования *Java* и занимает 16 Кб. В ней присутствует урезанный стандартный *Java API*, а также классы и интерфейсы для интерпретации команд программы и получения информации от датчиков.

Теперь о проекте *Isenguard*. Это название навеяно одноименной башней из романа Дж. Толкиена, так как робот-поставщик сконструирован из двух частей, одна из которых башня. В проекте предложена технология программирования рассматриваемых роботов на основе автоматного подхода.

Завершая введение, авторы выражают благодарность Аркадию Григорьевичу Хотину (генеральному директору ЗАО “Аркадия”) за инициативу создания этого проекта и предоставление набора *Lego Mindstorms*.

По материалам настоящей работы сделан доклад на тему *Automata-Based Programming of the Reactive Multi-Agent Control Systems* на конференции *International Conference Integration of Knowledge Intensive Multi-Agent Systems (KIMAS 2005): Modeling, Evolution and Engineering*. 8 – 21 April 2005, Waltham, Massachusetts (http://www.ieeeboston.org/kimas05/kimas05_program.htm), который опубликован в трудах этой конференции (http://is.ifmo.ru/articles_en/_kimas05-2.pdf).

1. Постановка задачи

Создать систему, обеспечивающую доставку заданного количество предметов определенного размера из одного места в другое.

Система состоит из двух роботов – транспортного робота и робота-поставщика предметов.

Доставка предметов ограничивается размерами небольшой комнаты или стола.

Транспортный робот должен ездить по определенному пути, который ведет от места доставки до места выдачи предметов и обратно. Под путем понимается черная линия на белом поле (бумаге). Характеристики пути описаны в разд. 2.

В начале транспортный робот находится в некотором месте – начале пути. Туда он и должен доставлять предметы.

Место выдачи и место доставки предметов отмечены соответствующими кусками фольги.

Препятствий на пути нет.

Цвета черной линии и поля предполагаются равномерными. Характеристики “равномерности” изложены в разд. 2.

При считывании показаний светового датчика транспортный робот должен уметь различать путь, поле и фольгу.

Обмен информацией между транспортным роботом, роботом-поставщиком и пультом управления происходит посредством сообщений, передаваемых через инфракрасные порты.

Робот-поставщик стоит на месте выдачи предметов. Он может получать сообщения, передаваемые с помощью инфракрасного порта транспортного робота о выдаче необходимого количества предметов (одного, двух или трех).

Информация о количестве выдаваемых предметов передается транспортному роботу при помощи пульта с тремя функциональными кнопками, помеченными символами “1”, “2” и “3”. Каждая из кнопок соответствует определенному типу сообщения. Пульт нельзя перепрограммировать.

Проект должен быть выполнен при помощи конструктора *Lego Mindstorms*.

Конструирование механической части роботов должно быть выполнено с помощью свободно распространяемой программы *MLCAD*, позволяющей строить конфигурации роботов виртуально. Эта программа размещена по адресу http://www.lmsoftware.com/mlcad/.

2. Функциональные возможности комплекта *Lego Mindstorms*

Для реализации проекта использовались:

- два микроконтроллера;
- два световых датчика;
- два датчика касания;
- четыре двигателя;
- пульт управления;
- различные детали из стандартного набора *Lego Technics*.

Основной частью комплекта *Lego Mindstorms* является микроконтроллер, в который загружаются ОС и управляющая программа. Корпус микроконтроллера имеет три гнезда для подключения датчиков и три гнезда для подключения двигателей. Сверху на корпусе расположен небольшой дисплей. На него можно вывести до пяти цифр одновременно. Микроконтроллер основан на микропроцессоре *Renesas/H8300 Series* корпорации *Hitachi*.

Движение робота осуществляется с помощью двигателей, каждый из которых имеет четыре режима работы:

- вращение вперед;
- вращение назад;
- торможение (двигатель противодействует прокрутке);
- движение по инерции (двигатель свободно прокручивается).

В режимах вращения вперед и назад можно задать одну из семи мощностей работы.

Датчики могут находиться в одном из двух режимов:

- пассивном;
- активном.

При пассивном режиме, в отличие от активного, на датчики не подается напряжение.

Световой датчик может работать в обоих режимах. В пассивном режиме датчик получает информацию об интенсивности освещения, а в активном – излучает ярко-красный пучок света и измеряет интенсивность отраженного пучка.

Поле содержит элементы трех цветов – черного, белого и серебристого (фольга). Показания светового датчика лежат в диапазоне от 0 до 100. Интенсивность света, отраженного от черного цвета, составляет около 30-40 пунктов, от белого – 44-55 пунктов, а от фольги – 63-70 пунктов.

Датчик касания всегда работает в пассивном режиме.

На микроконтроллере находится инфракрасный порт (ИК-порт). С помощью этого порта производится загрузка управляющей программы в память микроконтроллера. Также с его помощью роботы могут обмениваться сообщениями друг с другом и принимать сообщения от дистанционного пульта управления.

Дистанционный пульт управления может посылать три типа сообщений, соответствующих кнопкам, помеченным символами “1”, “2” и “3”. При нажатой кнопке он постоянно посылает сообщения. Это следует учитывать при написании обработчика событий от ИК-порта.

3. Процесс разработки управляющей системы

В данном разделе в общих чертах будет описано, как автор-студент пришел к необходимости использования автоматного подхода для разработки управляющей системы.

В начале работы над проектом было собрано несколько простых роботов, и для каждого из них были написаны управляющие программы, используя рекомендации из книг [4, 5].

При этом программы писались на языке *Java* под ОС *leJOS*. В результате студент убедился, что систему с достаточно сложным поведением с помощью указанных рекомендаций запрограммировать крайне трудно. При рекомендованном, по сути традиционном, программировании даже небольшие изменения по мере роста программы становилось вносить все сложнее и сложнее. В итоге, после нескольких дней работы по написанию кода, была создана малопонятная программа, работавшая нестабильно. При этом отладка была чрезвычайно трудна, так как на дисплей используемого робота можно вывести только несколько цифр. Кроме того, загрузка программы в микроконтроллер занимала довольно много времени – три-четыре минуты. Поэтому каждое изменение программы было слишком длительным. Стало понятно, что дальше работать, используя традиционный подход, бесперспективно.

После этого использовали рекомендацию, предложенную разработчиками ОС *leJOS*. При применении операционной системы *leJOS* ее авторы предложили применять концепцию, основанную на представлении действий робота в виде нескольких объектов типа `Behavior` (поведение). Эти объекты не связаны друг с другом. Для того чтобы с ними

работать, необходимо сначала загрузить их в объект типа `Arbitrator` (планировщик), а затем запустить `Arbitrator`. Он, в зависимости от приоритета объекта `Behavior`, а также определенного метода, возвращающего значение булевского типа и реализуемого для каждого такого объекта, выбирает тот объект `Behavior`, который должен работать в данный момент. Это похоже на то, как в современных операционных системах реализуется мультизадачность. Недостатком этого подхода является невозможность обмена информацией между объектами. Реализация рассматриваемой программы в рамках этого подхода прекратилась, когда понадобилось отличить кусок фольги, расположенный в одном конце пути, от куска фольги, расположенного в другом его конце. Это оказалось непосильной задачей для такого подхода. После этого было решено реализовать проект на основе автоматного подхода.

Сначала конечные автоматы использовались “приблизительно” – не так, как это сделано в окончательном варианте. Была создано некое описание алгоритма, с использованием графов переходов конечных автоматов. При этом, однако, не были четко прописаны условия перехода из одного состояния в другое, а выходные воздействия описывались словами, как это любят делать многие. При реализации программы не выполнялся изоморфный переход от графов переходов к исходному тексту программы. Конечно, такой подход к разработке несколько ускорил само написание программы, но полностью от “скользких” мест избавиться не удалось. Программу по-прежнему было трудно отлаживать, количество ошибок уменьшилось, но не до конца. Эту реализацию все-таки удалось довести до “рабочего” состояния. Однако робот вел себя достаточно непредсказуемо при попадании в необычную ситуацию. Например, если транспортный робот резко сдвигался из-за неровности пути влево или вправо, то он благополучно “зависал”, причем сказать что-либо об его поведении в этот момент было сложно.

Ввиду проблем с предыдущими тремя подходами, было решено использовать автоматный подход, основанный на SWITCH-технологии [6 – 8]. Можно привести много причин для обоснования достоинств этого подхода, о чем и будет сказано в следующих разделах работы. Единственное, что можно сказать о подходе в этом разделе, так это то, что после весьма длительного и тщательного проектирования программ управления роботами, эти программы были отлажены за *один* день. При этом ни разу не возникало вопросов о том, что робот сделает в данный момент времени.

Существует большое количество книг [4 – 5] и проектов [9 – 11], посвященных программированию *Lego Mindstorms*. К сожалению, и у тех и у других есть недостатки: в книгах не приводится никакой технологии проектирования, а в проектах отсутствует проектная документация.

Стоит отметить интересную разработку скандинавских исследователей [12], в которой они использовали концепцию конечных автоматов для создания виртуальных моделей различных роботов, собранных из *Lego Mindstorms*. При помощи этих моделей можно было тестировать управляющие программы, не загружая их в микроконтроллер. Однако в этой работе концепция конечных автоматов не рассматривалась в качестве логической модели для самой управляющей системы. В работах [13 – 16] конечные автоматы также используются для построения разного рода виртуальных моделей различных механизмов, однако проектная документация на них отсутствует.

4. Технология разработки программного обеспечения управляющей системы на основе автоматного подхода

Основные положения этого подхода были разработаны при создании управляющей программы для виртуального автономного реактивного агента, применяемого в известной игре “Robocode” [17 – 19].

Для систем реактивных агентов, работающих в реальных условиях, этот подход может быть сформулирован следующим образом [20].

1. Строится схема взаимодействия агентов.

2. Для каждого агента разрабатывается диаграмма классов.

3. Для каждого класса создается его структурная схема.

4. Если поведение класса описывается системой автоматов, то строится схема их взаимодействия. Автоматы могут взаимодействовать тремя способами:

- по вложенности;
- по вызываемости;
- за счет обмена номерами состояний.

5. Для каждого автомата создается четыре документа:

- словесное (вербальное) описание поведения автомата;
- схема связей автомата, задающая его интерфейс, в которой указываются символьные обозначения входных переменных и выходных воздействий и их полные названия, а также названия источников и приемников информации;
- граф переходов, в котором состояния имеют названия, а все остальные составляющие помечены символами. Это позволяет компактно и формально описывать даже весьма сложное поведение агентов;
- по графу переходов формально и изоморфно строится текст программы.

Отметим, что входные переменные могут быть функциями, в том числе весьма сложными.

6. Для каждого класса пишется программа, изоморфная его структурной схеме [18].

7. Отлаженные на модели управляющие программы загружаются в соответствующие физические агенты (роботы *Lego Mindstorms*).

8. Выполняется отладка системы, которая упрощается за счет возможности наблюдения за состояниями каждого автомата [20].

9. Осуществляется разработка и выпуск открытой проектной документации [21].

5. Построение программы по схемам переходов автоматов

В этом разделе описывается алгоритм построения исходного кода программы по графу переходов автоматов. Это будет сделано для языка *Java*. По графу переходов, который приводится ниже, будет построен исходный код.

Отметим, что в **вершинах** графов переходов могут использоваться выходные воздействия, помеченные словами **`in`** и **`out`**. Выходные воздействия, помеченные **`in,`** выполняются сразу после **входа** в состояние, а воздействия с пометкой **`out`** – после **выхода** из состояния, но до того, как будут выполнены воздействия на переходе из этого состояния. В дальнейшем эти воздействия будут называться in-воздействиями и out-воздействиями. В языке *UML* [22] такие действия называются “действиями при входе в состояние” и “действиями при выходе из состояния” соответственно. Если при переходе состояние не изменилось, то ни `in`-воздействия, ни `out`-воздействия не выполняются. Таким образом, **переход** из одного состояния в другое сопровождается **сначала out-воздействиями** в **первом** состоянии, затем **воздействиями на переходе**, и уже **потом in-воздействиями** во **втором** состоянии.

Иногда переходы из состояния помечаются **приоритетами**. В этом случае **вначале** проверяются условия, соответствующие переходам с **меньшим номером** (большим приоритетом).

Блокировкам, выполненным за счет обмена номерами состояний, (например, `y==4`) всегда присваивается **наивысший** приоритет.

Иногда для устранения повторяющихся фрагментов кода используются ветвящиеся переходы. Они обозначаются при помощи кружка. Переход выполняется при условии, что выполняется как условие на стрелке, входящей в кружок, так и условие на стрелке, выходящей из кружка.

Помимо ветвящихся переходов также могут существовать **мультипереходы**. Они используются, если необходимо при **переходе** из одного состояние в другое совершать **различные выходные воздействия** при **различных** значениях **входных переменных**. Такие переходы обозначаются при помощи нескольких пометок на переходе между двумя состояниями.

Вложенные автоматы всегда реализуются до функций выходных воздействий.

На рис. 1, рис. 2 изображены примеры графов переходов.

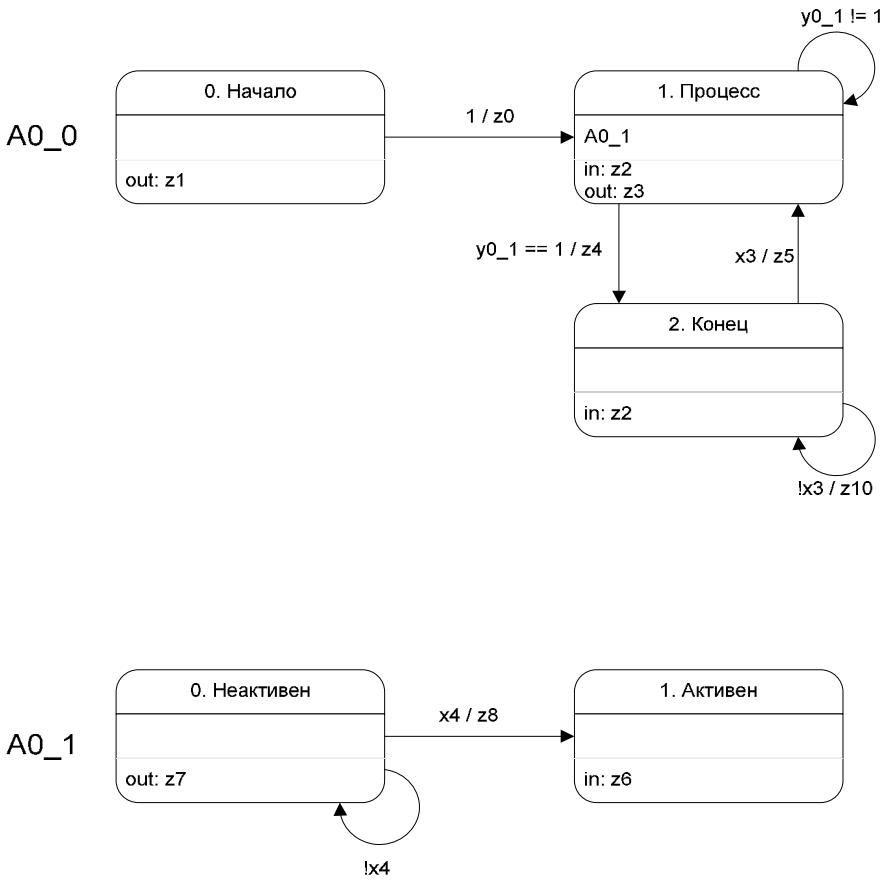

Рис. 1. Граф переходов с вложенным автоматом

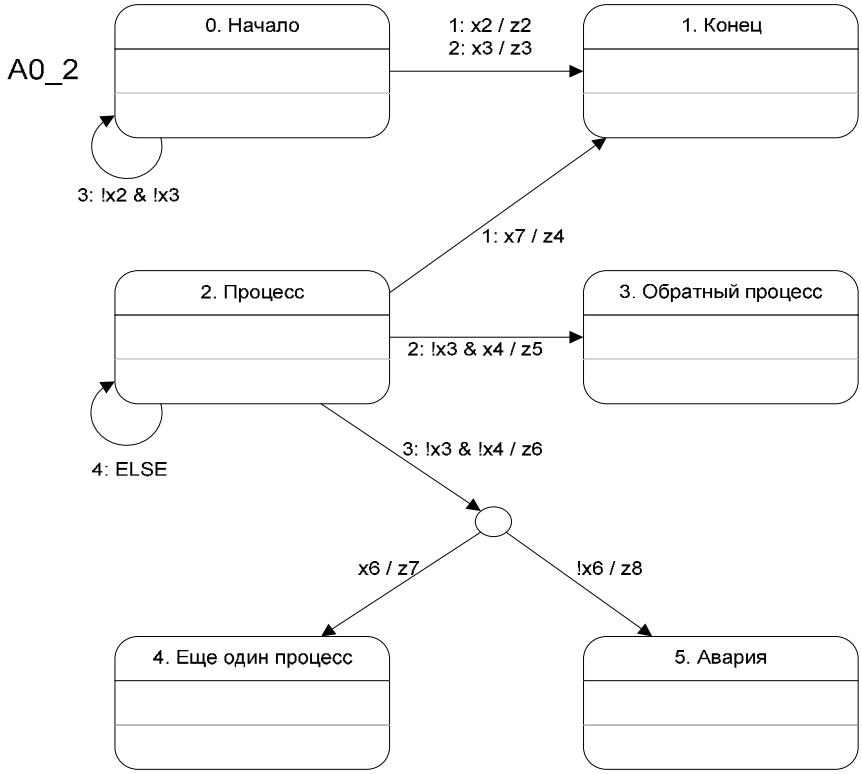

Рис. 2. Граф переходов с приоритетами, мультипереходами и ветвящимся переходом

5.1. Реализация автоматов на языке *Java*

Опишем процесс создания программы на языке *Java*. Одним из недостатков ОС *leJOS* является невозможность применения оператора `switch`. Поэтому вместо этого оператора будем использовать несколько идущих друг за другом конструкций `if…else`. Для экономии памяти все входные и выходные воздействия, а также автоматные функции реализованы в виде статических функций одного класса.

```
//Переменные и константы реализуются следующим образом
public static final int ES_POWER = 1;
public static final int VOLTAGE_TIME = 2000;
```

Функция булевского типа, соответствующая входной переменной с номером `n`, обозначаются как `xn`. Для случая `n==2` имеем следующий фрагмент кода:

```
public static boolean x2() {
     //Здесь может находиться произвольный код
}
```

Функция, соответствующая выходному воздействию с номером `n`, обозначается как `zn`. Для случая `n=3` имеем следующий фрагмент кода:

```
private static void z3() {
     //Здесь может находиться произвольный код
}
```

Переменные, содержащие номер состояния автомата, являются статическими и обозначаются как `yn`. Здесь `n` – номер автомата. Для случая `n=0_0` имеем следующий фрагмент кода:

```
public static int y0_0 = 0;
```

При этом первый ноль – номер робота, а второй – номер автомата в этом роботе.

Помимо переменных, содержащих номер состояния, заданы переменные булевского типа. Они обозначаются следующим образом: имя переменной, содержащей номер состояния автомата, а затем суффикс `changed`. Значение этой переменной равное `true` показывает, что состояние автомата изменилось, и следует выполнить `in`-воздействия в новом состоянии. Инициализация этих переменных производится значением `false`:

```
public static boolean y0_0changed = false;
```

Для удобства in- и out-воздействия автоматов вынесены в отдельные функции. Их названия образуются следующим образом – сначала префикс `A`, потом номер робота, потом символ подчеркивания, потом номер автомата в роботе, затем символ `S` (от слова “state”), номер состояния и, в зависимости от типа воздействия, суффикс `IN` или `OUT`.

```
//out-воздействия состояния 0 автомата A0_0 (рис. 1)
public static void A0_0S0OUT() {
     z1();
}
```

```
//in-воздействия состояния 1 автомата A0_0 (рис. 1)
public static void A0_0S1IN() {
     z2();
}
```

Теперь рассмотрим схему реализации автоматных функций на примере автомата `A0_0` (рис. 1). Эта схема изображена на рис. 3.

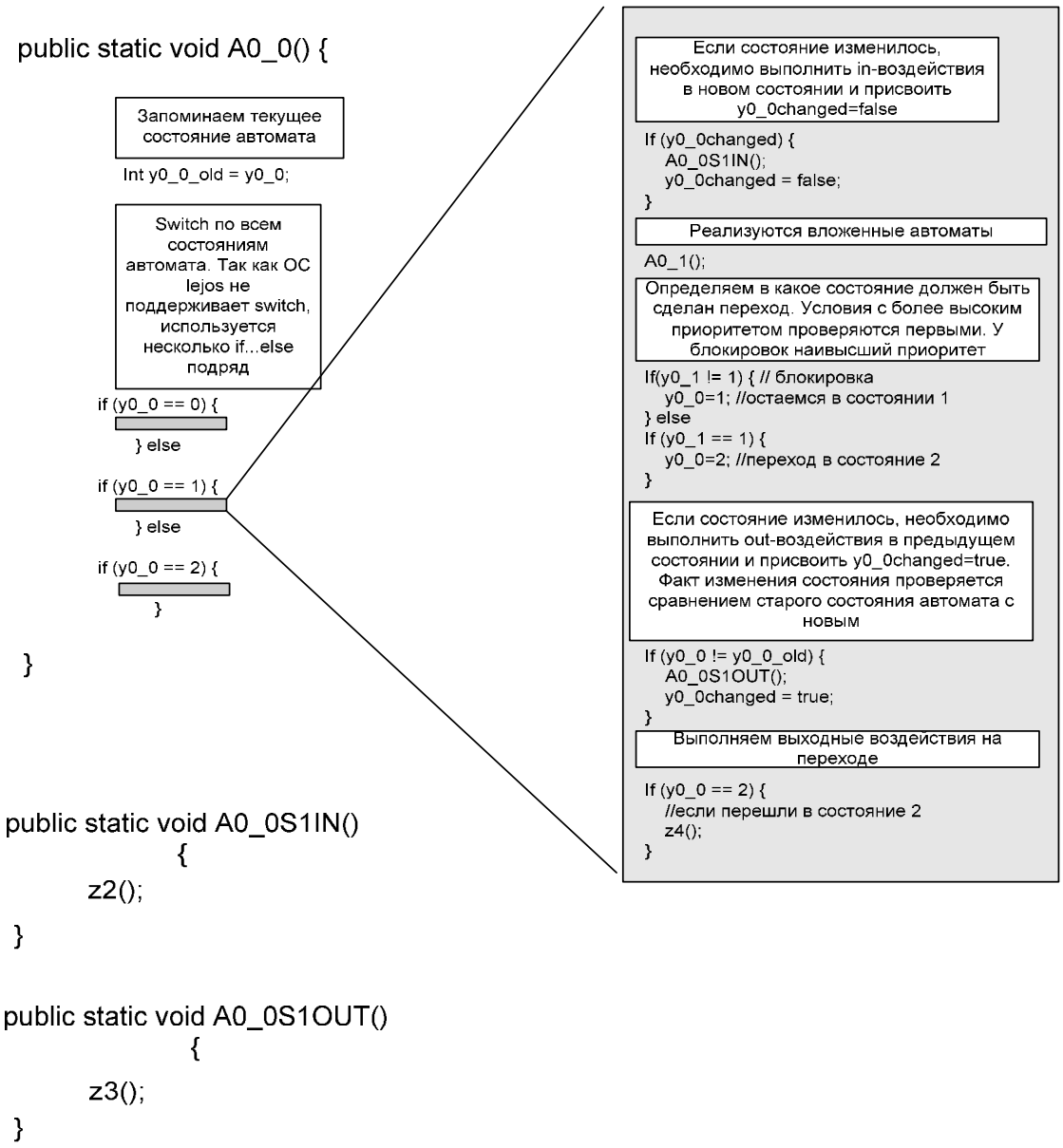

Рис. 3. Метод построения изоморфного Java-кода по графу переходов. Автомат A0_0

На рис. 4, рис. 5 показаны схемы реализации мультипереходов и ветвящихся переходов.

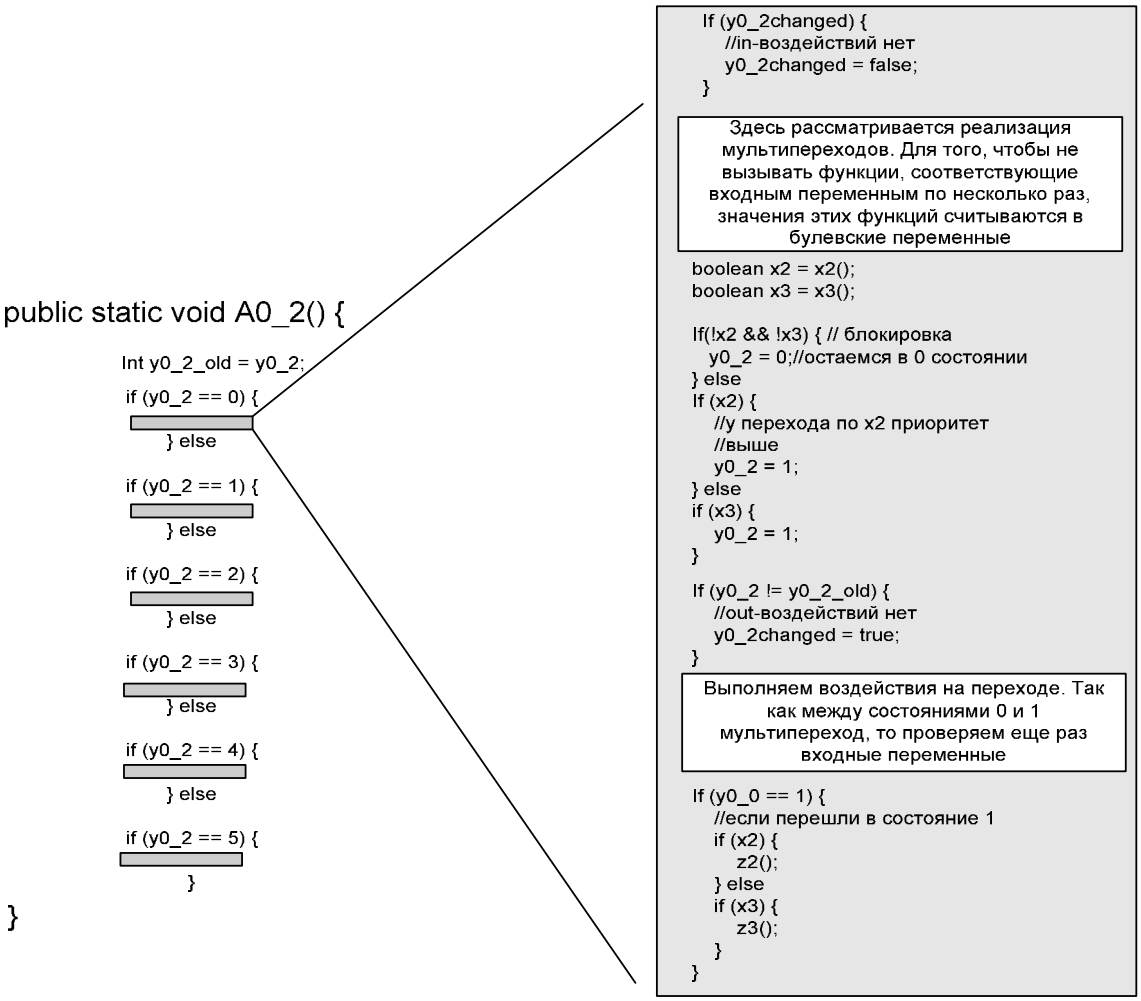

Рис. 4. Реализация мультипереходов. Автомат A0_2

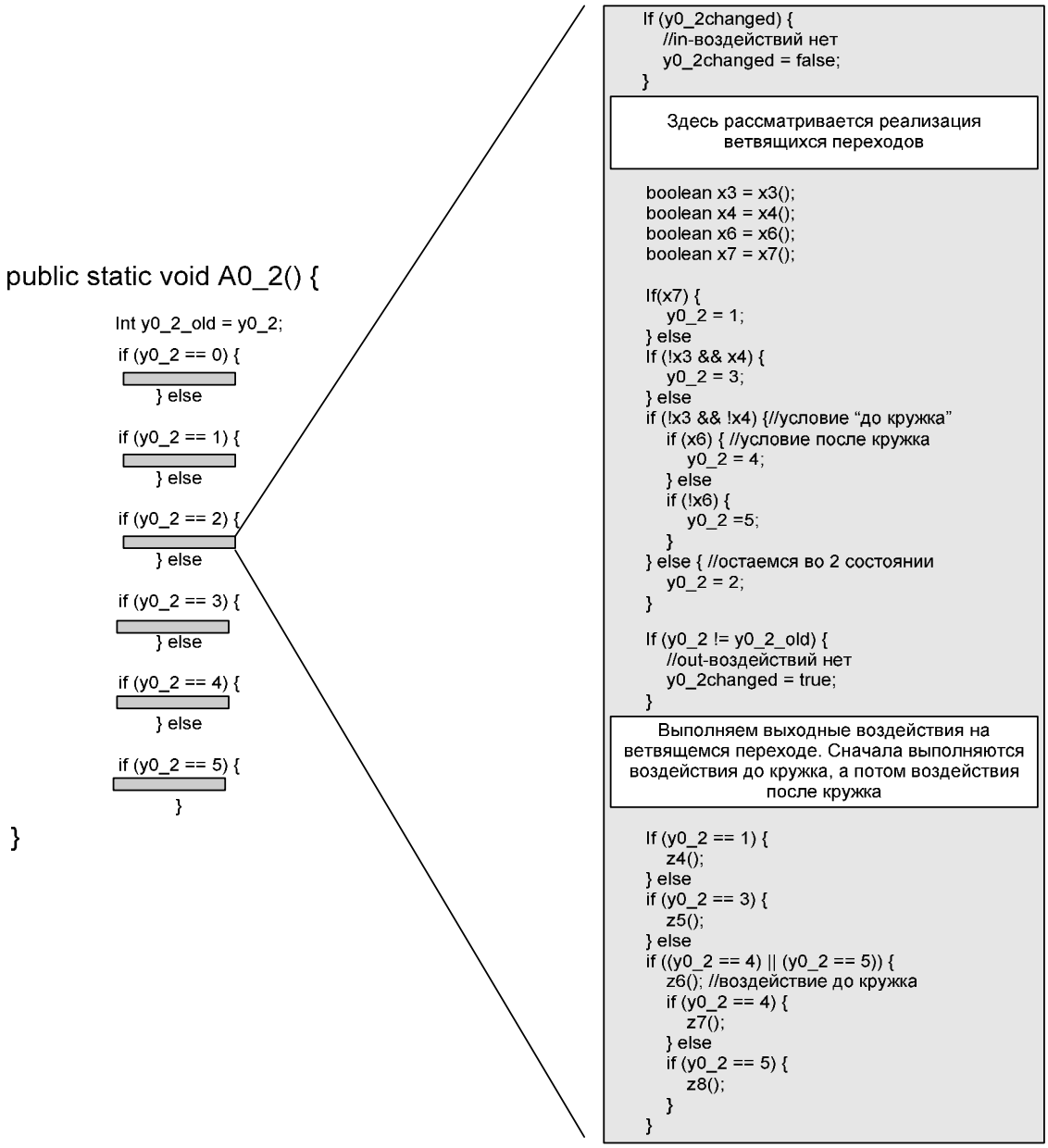

Рис. 5. Реализация ветвящихся переходов. Автомат `A0_2`

При программировании на языке *Java* основная функция `main` выглядит следующим образом:

```
public static void main(String args[]) {
     while (true) {
          A0_0();
     }
}
```

Функция автомата `A0_0`, как внешнего (головного) автомата, вызывается в бесконечном цикле.

5.2. Взаимодействие между автоматами

Система управления транспортным роботом состоит из нескольких вложенных автоматов, каждый из которых выполняет определенную задачу. Для того, чтобы внешний автомат мог дать команду вложенным в него автоматам, ему необходимо выполнить соответствующее выходное воздействие. Вложенный автомат узнает о том, что ему была дана команда что-то сделать, когда соответствующая входная переменная примет значение `true`. После этого он должен сбросить эту переменную. Это выполняется при помощи соответствующего выходного воздействия. Если вложенному автомату для работы нет необходимости получать команды от внешнего автомата, он может запретить получение команд от внешнего автомата. Разрешение и запрещение получения команд выполняется при помощи выходных воздействий вложенного автомата. Внешний автомат также может получать команды от вложенного автомата и разрешать/запрещать получение команд. **Таким образом, один автомат может выставить входную переменную другому автомату посредством своего выходного воздействия. Другой автомат может сбросить значение этой переменной также при помощи своего выходного воздействия.**

Рассмотрим фрагмент *Java*-кода из реализации управляющей программы для транспортного робота. Эта программа будет подробно рассмотрена далее. Здесь этот фрагмент кода приводится только для иллюстрации. Для удобства все булевские переменные и передаваемые параметры, относящиеся к одному автомату, имеют префикс `An_`, где `n` – номер автомата. Переменная `An_AkListener` принимает значение `true`, если команды от автомата `Ak` разрешены автомату `An`, и `false` в противном случае.

```
//Выходное воздействие автомата A2, передающее команду
//автомату A3
public static void z2_6() {
     if (A3_A2Listener) {
          A3_findPath = true;
     }
}

//Входная переменная автомата A3, принимает значение true,
//если пришла команда от автомата A2
public static boolean x3_1() {
     return A3_findPath;
}
//Следующие два выходных воздействия автомата A3
//разрешают получение команд от автомата A2
public static void z3_2_0() {
     A3_A2Listener = false;
}
public static void z3_2_1() {
     A3_A2Listener = true;
}
//Это выходное воздействие автомата A3 сбрасывает
//переменную A3_findPath и соответственно x3_1
public static void z3_2_2() {
     A3_findPath = false;
}
```

5.3. Обмен сообщениями по инфракрасному порту в операционной системе *leJOS*

Операционная система *leJOS* предоставляет интерфейс *Serial* для реализации обмена сообщениями по ИК-порту. Этот интерфейс дает возможность посылать однобайтные сообщения от одного робота к другому, а также обрабатывать нажатия кнопок на пульте дистанционного управления. Разработчики ОС *leJOS* пытались реализовать стандартные *Java* интерфейсы для обмена информацией, но эта попытка оказалась неудачной. В качестве протокола нижнего уровня разработчики ОС *leJOS* использовали протокол, разработанный в Стэнфордском университете [23]. Принцип его работы можно охарактеризовать следующим образом. Для того, чтобы послать какую-либо информацию, необходимо передать в систему специальный префикс, который определяет параметры передачи, а затем уже собственно информацию. Число `f7`, к примеру, является указанием системе о том, что следующий байт необходимо послать по ИК-порту.

Посылка сообщений реализуется следующим образом. Достаточно записать в массив из двух байтов следующую информацию: в первый байт – шестнадцатеричное число `f7`, а второй байт – то, что необходимо передать. После этого вызвать функцию `Serial.sendPacket(byte[] packet/*массив*/, 0, 2 /*число байт в массиве*/)`. В результате по ИК-порту будет передано однобайтовое сообщение другому роботу.

Прием сообщений немного сложнее – сообщения от пульта и от другого робота отличаются префиксами. Значение **первого байта** сообщения от **пульта** всегда равно **210**, второго – 0. Если в третьем байте единица стоит на позиции, соответствующей нулевой степени двух, то, следовательно, была нажата кнопка “1”. Если единица стоит на позиции для единичной степени двух, то была нажата кнопка “2”. Если она расположена на позиции для второй степени двух, то была нажата кнопка “3”. Значение **первого байта** сообщения от **другого робота не равно 210**. Следующий байт содержит посланную информацию. Метод `Serial.isPacketAvailable()` возвращает `true`, если был получено сообщение. Метод `Serial.readPacket(byte[])` читает полученное сообщение в массив байтов.

Функция `updateSignals()`, реализованная как для транспортного робота, так и для робота поставщика, просматривает принятые сообщения и при необходимости обновляет значения булевских переменных, соответствующих факту принятия сообщения от пульта и/или другого робота, а также обновляет значение целочисленной переменной `remoteValue`, которая показывает, какая кнопка была нажата на пульте. Эта функция должна вызываться всегда в начале выполнения функции, соответствующей входной переменной или выходному воздействию, ответственному за прием и передачу сообщений по ИК-порту. Одной из особенностей пульта является то, что он постоянно посылает сообщения при нажатой кнопке – он, как многие другие пульты, не прекращает передачу после отправки одного сообщения. Так как транспортному роботу необходимо только первое сообщение из этой “очереди”, остальные сообщения удаляются без обработки.

В данном разделе рассматривается реализация обмена сообщениями по ИК-порту для транспортного робота. Реализация для робота-поставщика практически идентична.

Чтобы проверить, есть ли сообщения от другого робота или от пульта, необходимо вызвать функцию `updateSignals()`, а затем проверить значения соответствующих булевских переменных. В случае транспортного робота следует проверить значения переменных `dispencerSignal` и `remoteSignal`. В случае если одно из этих значений

true, значит пришло сообщение от робота-поставщика или дистанционного пульта соответственно. Также можно запретить сообщения от робота-поставщика и от пульта, присвоив переменным dispencerActive и remoteActive значение false. Разрешить сообщения, можно присвоив переменным dispencerActive и remoteActive значение true.

Значения переменных dispencerSignal и remoteSignal используются при формировании значений, соответствующих входных переменных. Значения переменных dispencerActive и remoteActive также формируют значения соответствующих входных переменных. Они также могут быть изменены при помощи соответствующих выходных воздействий.

```
//Функция updateSignals()
public static void updateSignals() {
     while (Serial.isPacketAvailable()) {
          Serial.readPacket(packet);
     //первый байт не равен 210, пришло сообщение от
     //робота-поставщика
          if ((packet[0] & 255) != 210) {
               if (dispencerActive) {
                    //Пришло сообщение от робота-постащика
                    dispencerSignal = true;
               }
          } else {
     //Пришло сообщение от пульта
               if (!remoteSignal && remoteActive) {
                    int c1=packet[1] & 255;
                    int c2=packet[2] & 255;
                    if (c1 == 0) {
                         //Нажата кнопка 1
                         if (c2==0x01) remoteValue = 1;
                         //Нажата кнопка 2
                         else if (c2==0x02) remoteValue = 2;
                         //Нажата кнопка 3
                         else if (c2==0x04) remoteValue = 3;
                    }
                    //Пришло сообщение от пульта. Обновляем
                    //значение переменной remoteSignal
                    remoteSignal = true;
               }
          }
     }
}

//Пример входной переменной, возвращающей значение true, если
//получено сообщение по ИК-порту от пульта и значение false – в
//противном случае

public static boolean x1_0_0() {
     //Сначала вызывается функция updateSignals()
     updateSignals();
     //Затем возвращается значение булевской переменной
     //remoteSignal
```

```
     return remoteSignal;
}

//Пример выходного воздействия, запрещающего сообщения от пульта
public static void z1_3_0() {
          updateSignals();
          remoteActive = false;
}

//Пример выходного воздействия, разрешающего сообщения от пульта
public static void z1_3_1() {
          updateSignals();
          remoteActive = true;
}
//Пример выходного воздействия, обрабатывающего сообщение от
//пульта.
//Количество предметов считывается в переменную candyCount
     public static void z1_3_2() {
          updateSignals();
          if (remoteSignal) {
               remoteSignal = false;
               candyCount = remoteValue;
          }
     }
```

6. Взаимодействие роботов

Взаимодействие роботов описывается при помощи соответствующей схемы. В данном проекте она оказалось довольно простой – пульт посылает три типа сообщений, робот-поставщик посылает сообщение транспортному роботу о конце погрузки и транспортный робот посылает сообщение роботу-поставщику. В сообщении указано о количестве предметов для доставки. Схема взаимодействия роботов приведена на рис. 6.

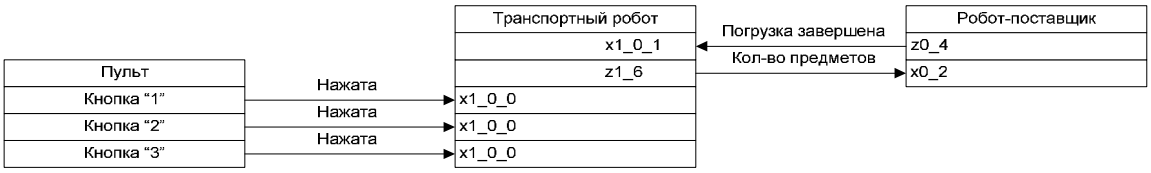

Рис. 6. Схема взаимодействия роботов

7. Транспортный робот

7.1. Модель транспортного робота

На рис. 7 приведен виртуальный транспортный робот, сконструированный с помощью программы *MLCAD*. Файл с описанием транспортного робота для программы *MLCAD* размещен по адресу http://is.ifmo.ru/projects/lego.

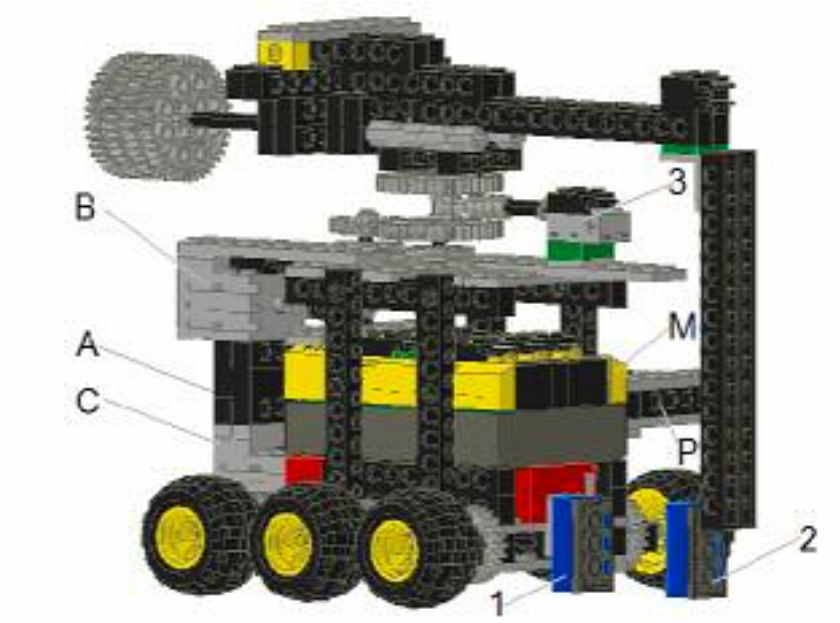

Рис. 7. Виртуальный транспортный робот

На этом рисунке отмечены следующие части робота:

- световой датчик 1 подключен к гнезду 1 корпуса микроконтроллера (номер гнезда указан на корпусе микроконтроллера);
- световой датчик поиска пути 2, размещенный на вращающейся штанге и подключенный к гнезду 2;
- датчик касания 3, необходимый для центрирования штанги, подключен к гнезду 3;
- двигатель *A*, вращающий левые колеса, подключен к гнезду A;
- двигатель *C*, вращающий правые колеса, подключен к гнезду C;
- двигатель *B*, отвечающий за вращение штанги, подключен к гнезду B;
- поддон для предметов *P*;
- микроконтроллер с инфракрасным портом *M*.

Датчик 1 закреплен неподвижно. Датчик 2 размещен на вращающейся штанге, прикрепленной к двигателю B. Она способна поворачиваться на 180º влево и вправо попеременно.

На основе схемы виртуального робота на рис. 7 собран транспортный робот (рис. 8).

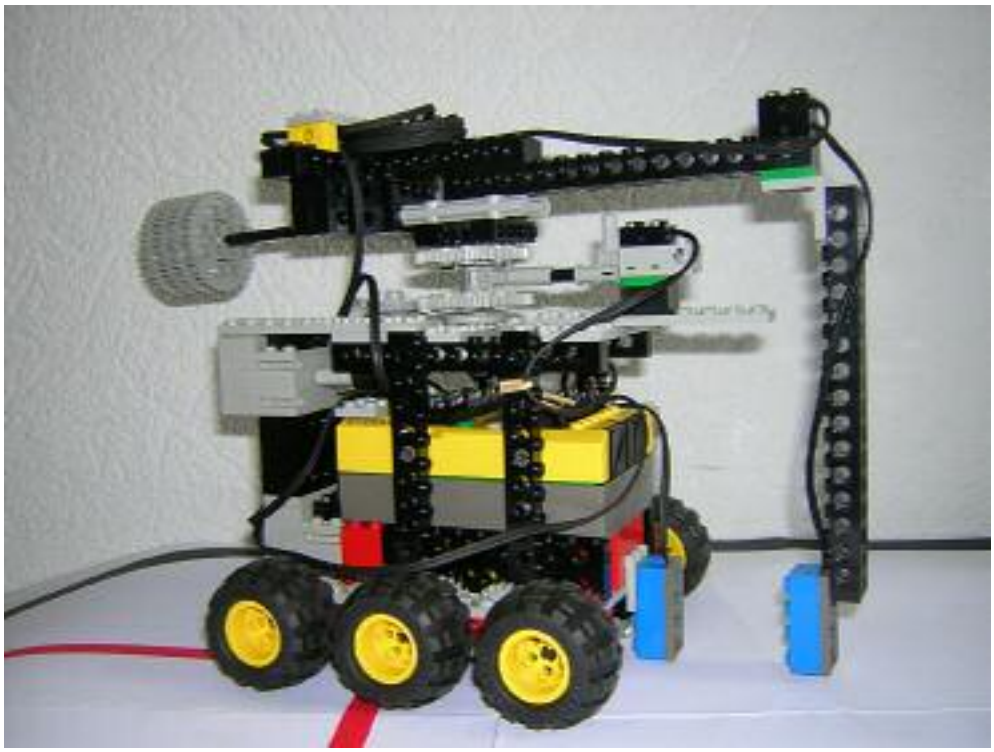

Рис. 8. Транспортный робот

На рис. 9 показано прохождение поворота транспортным роботом.

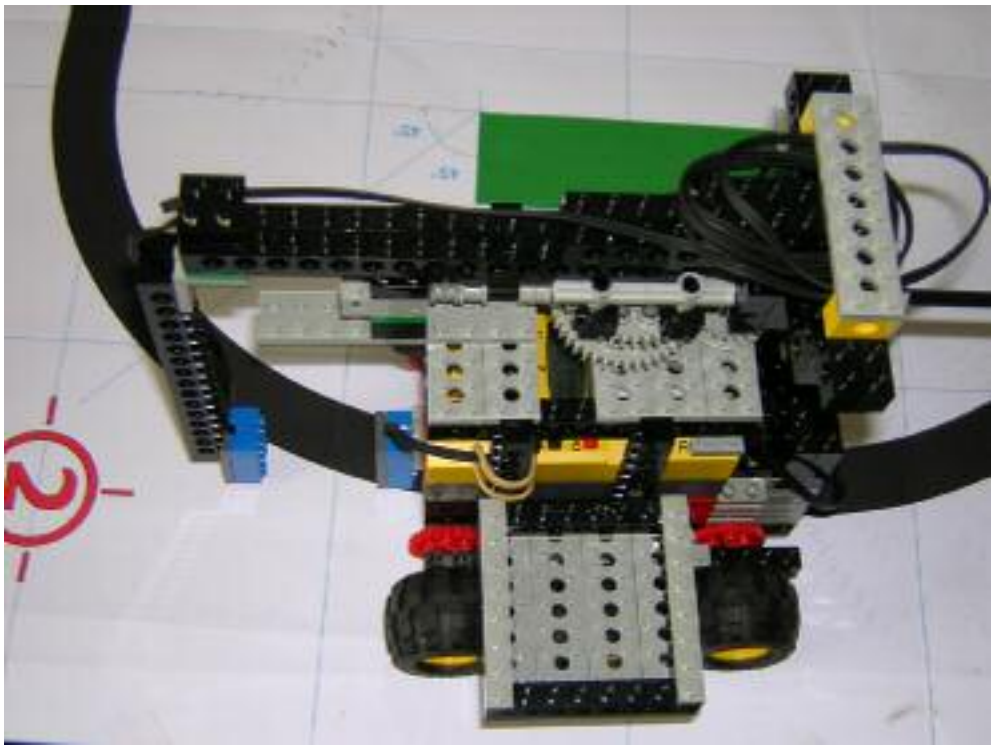

Рис. 9. Прохождение поворота транспортным роботом

7.2. Особенности конструкции транспортного робота

Благодаря использованию червячной передачи для связи двигателей с колесами, робот имеет возможность практически мгновенно останавливаться. Тормозной путь достаточно мал. Червячная передача не позволяет роботу двигаться самопроизвольно.

Робот является колесным и способен с высокой точностью вращаться вокруг своего центра. Это объясняется тем, что колеса имеют глубокий протектор, повышающий сцепление с “дорогой” и предотвращающий проскальзывание на поворотах. Сначала был построен гусеничный робот, однако авторы были вынуждены отказаться от него, так как его сильно сносило с пути на поворотах.

Для соединения мотора со штангой, к которой прикреплен световой датчик поиска пути 2, также используется червячная передача. Это позволяет уменьшить инерцию штанги при вращении. Штанга способна вращаться на 180º влево и вправо попеременно. При повороте более чем на 180º провод, соединяющий штангу с микроконтроллером, намотается на шестеренки, что приведет к аварии. Управляющая программа противодействует таким поворотам.

На другом конце штанги прикреплен противовес, который обеспечивает плавность хода. Благодаря уравновешенности конструкции штанги и червячной передачи, штангу можно центрировать относительно светового датчика 1 с достаточно высокой точностью (погрешность не более 5º).

Для центрирования применяется датчик касания 3, установленный на корпусе робота. В тот момент, когда световые датчики и штанга находятся на одной прямой, датчик срабатывает и передает соответствующий сигнал своему микроконтроллеру для остановки двигателя *B*.

Световой датчик поиска пути 2, закрепленный на конце штанги, соединяется с микроконтроллером проводом, который укреплен вдоль всей штанги. Световой датчик 1 закреплен на корпусе робота спереди, его положение фиксировано.

ИК-порт встроен в микроконтроллер и расположен в передней части робота. Конструкция робота выбрана таким образом, чтобы не препятствовать распространению сигнала от порта.

Поддон находится слева от микроконтроллера. В его конструкции предусмотрены бортики, чтобы предметы не вываливались.

В целом конструкция робота, если не считать штанги, крепкая и устойчивая. Возможна иная конструкция робота, в которой штанга крепится не над микроконтроллером, а под ним.

8. Управление транспортным роботом

8.1. Спецификация на управляющую систему

Транспортный робот должен, обработав сообщение от пульта, развернуться, проехать по пути, используя штангу со световым датчиком. Доехав до участка пути, помеченного фольгой, он должен подъехать к роботу-поставщику, послать ему сообщение о количестве

предметов к выдаче, получить предметы, отъехать, развернуться, проехать по пути до участка, помеченного фольгой, и ожидать следующего сообщения от пульта.

8.2. Алгоритм для движения по пути

Для корректного движения по пути был разработан соответствующий алгоритм. Он довольно прост, но без знания его весьма проблематично понять логику работы автоматов `A2`, `A3` и `A5`, входящих в состав управляющей системы транспортного робота. Предположим, что первоначально световой датчик 1 находится на пути. Транспортный робот также ориентирован по пути. Пока световой датчик 1 находится на пути, робот движется вперед. Как только световой датчик 1 покажет отличное от пути значение, следует этап корректировки этого значения – в течение определенного промежутка времени робот продолжит двигаться вперед, потом показания фиксированного светового датчика 1 снимаются повторно. Это сделано для того, чтобы была возможность отличить фольгу от белого поля. Как уже говорилось ранее, показания светового датчика лежат в промежутке от 0 до 100. Когда световой датчик снимает показания на границе между фольгой и путем, он зачастую выдает число, соответствующее белому полю, что ведет к неправильной работе. С этой целью был введен этап корректировки показаний. Если после этапа корректировки световой датчик 1 возвращает цвет пути, то робот продолжает двигаться вперед. Если цвет белый, то робот пытается найти путь при помощи штанги. Если цвет фольги, то робот доехал до конца пути и останавливается. Если цвет определен некорректно, то робот также останавливается, а алгоритм завершает работу с сообщением об ошибке.

Теперь опишем то, как робот ищет путь. Сначала штанга движется влево, если путь после определенного промежутка времени не найден, то она движется вправо. Если и справа путь не найден – то выводится сообщение об ошибке. Если путь найден справа или слева, то робот поворачивает в соответствующем направлении, штанга возвращается в центрированное положение. После того, как световой датчик 1 обнаружит путь, то робот совершает еще небольшой поворот в том же направлении, чтобы световой датчик 1 оказался на центре пути. Это делается для более эффективного движения по пути. Далее транспортный робот продолжает движение вперед.

8.3. Общее описание управляющей системы

Транспортный робот управляется четырьмя автоматами – `A1`, `A2`, `A3` и `A5`. Автоматы взаимодействуют друг с другом посредством вложенности и командами (разд. 5.2). Основным автоматом является автомат `A1`. Он отвечает за обработку сообщений от пульта управления, обмен сообщениями с роботом-поставщиком, а также за поворот после получения сообщений от пульта и робота-поставщика. Автомат `A2` отвечает за движение по пути. Автомат `A3` управляет штангой, на которой закреплен световой датчик 2. Автомат `A5` используется для возвращения штанги в центральное положение – центрирование штанги. Идентификаторы входных переменных и выходных воздействий, общих для автоматов транспортного робота, записываются в форме `z_0..` или `x_0..`, где вместо точек стоит уникальное выражение, состоящее из цифр и символов подчеркивания. Входные переменные и выходные воздействия, специфичные для одного автомата, записываются в форме `z_n..` или `x_n..`, где вместо символа `n` стоит номер этого автомата.

Перечень констант

Здесь перечислены все константы, определенные для транспортного робота. Значения констант, определенных опытным путем, приведены в приложении 3.

VOLTAGE_TIME – время, в течение которого выведено на дисплей напряжение на батареях. Измеряется в миллисекундах.

TURN_TIME – время, необходимое для поворота транспортного робота на 180º. Определяется экспериментально.

BACK_TIME – время, необходимое для транспортному роботу, чтобы отъехать от робота-поставщика. Определяется экспериментально.

ROT_TIME – максимальное время поворота транспортного робота, если он исправен.

ROT_EX_TIME – время, необходимое для поворота транспортного робота влево или вправо, чтобы световой датчик 1 сместился с границы пути к его центру.

CORRECT_TIME – время, необходимое для движения вперед с целью подтверждения цвета светового датчика 1.

ES_POWER – значение мощности работы двигателей колес для движения вперед и назад.

ER_POWER – значение мощности работы двигателей колес для поворота.

RR_TIME – константа, соответствующая времени, за которое штанга повернется на 180º.

RB_TIME – константа, соответствующая максимальному возможному времени возвращения штанги назад, если она исправна.

ERR_POWER – значение мощности работы двигателя штанги.

BLACK_UP – верхняя граница значения на световых датчиках, соответствующая черному цвету (цвету пути).

BLACK_DOWN – нижняя граница, соответствующая черному цвету (цвету пути).

WHITE_UP – верхняя граница значения на световых датчиках, соответствующая белому цвету (цвету поля).

WHITE_DOWN – нижняя граница, соответствующая черному цвету (цвету пути).

FOIL_UP – верхняя граница значения на световых датчиках, соответствующая фольге (конец пути).

FOIL_DOWN – нижняя граница, соответствующая фольге (конец пути).

OTHER_UP – верхняя граница значения на световых датчиках, соответствующая некорректному цвету.

OTHER_DOWN – нижняя граница, соответствующая некорректному цвету.

Перечень входных переменных, общих для автоматов A2 и A3

x0_4_0 – значение светового датчика 1 соответствует полю.

x0_4_1 – значение светового датчика 1 соответствует пути.

x0_4_2 – значение светового датчика 1 соответствует фольге.

x0_4_3 – значение светового датчика 1 ничему не соответствует.

x0_5_0 – значение светового датчика 2 соответствует полю.

x0_5_1 – значение светового датчика 2 соответствует пути.

x0_5_2 – значение светового датчика 2 соответствует фольге.

x0_5_3 – значение светового датчика 2 ничему не соответствует.

Перечень выходных воздействий, общих для автоматов A1, A2, A3 и A5

z0_5_1 – включить таймер 1.

z0_6_1 – остановить и обнулить таймер 1.

z0_5_2 – включить таймер 2.

z0_6_2 – остановить и обнулить таймер 2.

z0_5_3 – включить таймер 3.

z0_6_3 – остановить и обнулить таймер 3.

z0_5_4 – включить таймер 4.

z0_6_4 – остановить и обнулить таймер 4.

z0_7 – запустить левый и правый двигатели вперед со значением мощности ES_POWER.

z0_8 – запустить левый и правый двигатели назад со значением мощности ES_POWER.

z0_9 – остановить левый и правый двигатели.

z0_10 – запустить левый двигатель назад, а правый вперед со значением мощности ER_POWER. Для поворота налево.

z0_11 – запустить левый двигатель вперед, а правый назад со значением мощности ER_POWER. Для поворота направо.

8.4. Автомат A1. Управление транспортным роботом

Краткое описание

Автомат A1 получает сообщение от пульта, далее он (посредством автомата A2) проводит транспортный робот по пути, получает предметы от робота-поставщика и возвращается к месту доставки (также посредством автомата A2).

Перечень констант

VOLTAGE_TIME – время, в течение которого выведено на дисплей напряжение на батареях. Измеряется в миллисекундах.

TURN_TIME – время, необходимое для разворота робота. Определяется экспериментально.

BACK_TIME – время, необходимое для подъезда и отъезда от робота-поставщика. Определяется экспериментально.

Перечень переменных

candyCount – количество предметов для доставки. Начальное значение – ноль.

Перечень входных переменных

x1_0_0 – сообщение от пульта по ИК-порту получено. Не может быть выставлена одновременно с переменной x1_0_1.

x1_0_1 – сообщение от робота-поставщика по ИК-порту получено. Не может быть выставлена одновременно с переменной x1_0_0.

x1_2 – таймер 1 превысил значение TURN_TIME.

x1_3 – таймер 1 превысил значение BACK_TIME.

x1_4 – таймер 1 превысил значение VOLTAGE_TIME.

x1_6 – получена команда от автомата A2 посредством выходного воздействия z2_3 о том, что конец пути достигнут. На схеме связей и далее по тексту последнее обозначается как A2(z2_3).

x1_8 – транспорт пустой.

Блокировки

y2 == 3 – транспортный робот не смог проехать по заданному пути (авария).

Перечень состояний y1

0 – Начало.

1 – Ожидание сообщения.

2 – Позворот.

3 – Движение по линии.

4 – Реверс.

5 – Авария.

6 – Въезд.

7 – Вывод напряжения на дисплей.

Перечень выходных воздействий

`z0_5_1` – включить таймер 1.

`z0_6_1` – остановить и обнулить таймер 1.

`z0_7` – запустить левый и правый двигатели вперед со значением мощности `ES_POWER`.

`z0_8` – запустить левый и правый двигатели назад со значением мощности `ES_POWER`.

`z0_9` – остановить левый и правый двигатели.

`z0_11` – запустить левый двигатель вперед, правый назад со значением мощности `ER_POWER`. Для поворота направо.

`z1_0` – вывести на дисплей напряжение на батареях.

`z1_1` – перевести световой датчик 1 в активный режим.

`z1_2` – перевести световой датчик 2 в активный режим.

`z1_3_0` – запретить сообщения от пульта.

`z1_3_1` – разрешить сообщения от пульта.

`z1_3_2` – обработать сообщение от пульта.

`z1_4_0` – запретить сообщения от робота-поставщика.

`z1_4_1` – разрешить сообщения от робота-поставщика.

`z1_4_2` – обработать сообщения от робота-поставщика.

`z1_5` – вывести на дисплей сообщение об аварии – число 666.

`z1_6` – послать сообщение роботу-поставщику о количестве предметов для погрузки.

`z1_7_0` – запретить получение команд от автомата `A2 (A1_A2_Listener = false)`.

`z1_7_1` – разрешить получение команд от автомата `A2 (A1_A2_Listener = true)`.

`z1_7_2` – сбросить переменную `x1_6`.

z1_8 – выдать команду автомату A2 о начале работы посредством входной переменной x2_1. На схеме связей и далее по тексту последнее обозначается как A2(x2_1).

На рис. 10 приведена схема связей автомата A1, а на рис. 11 – его граф переходов.

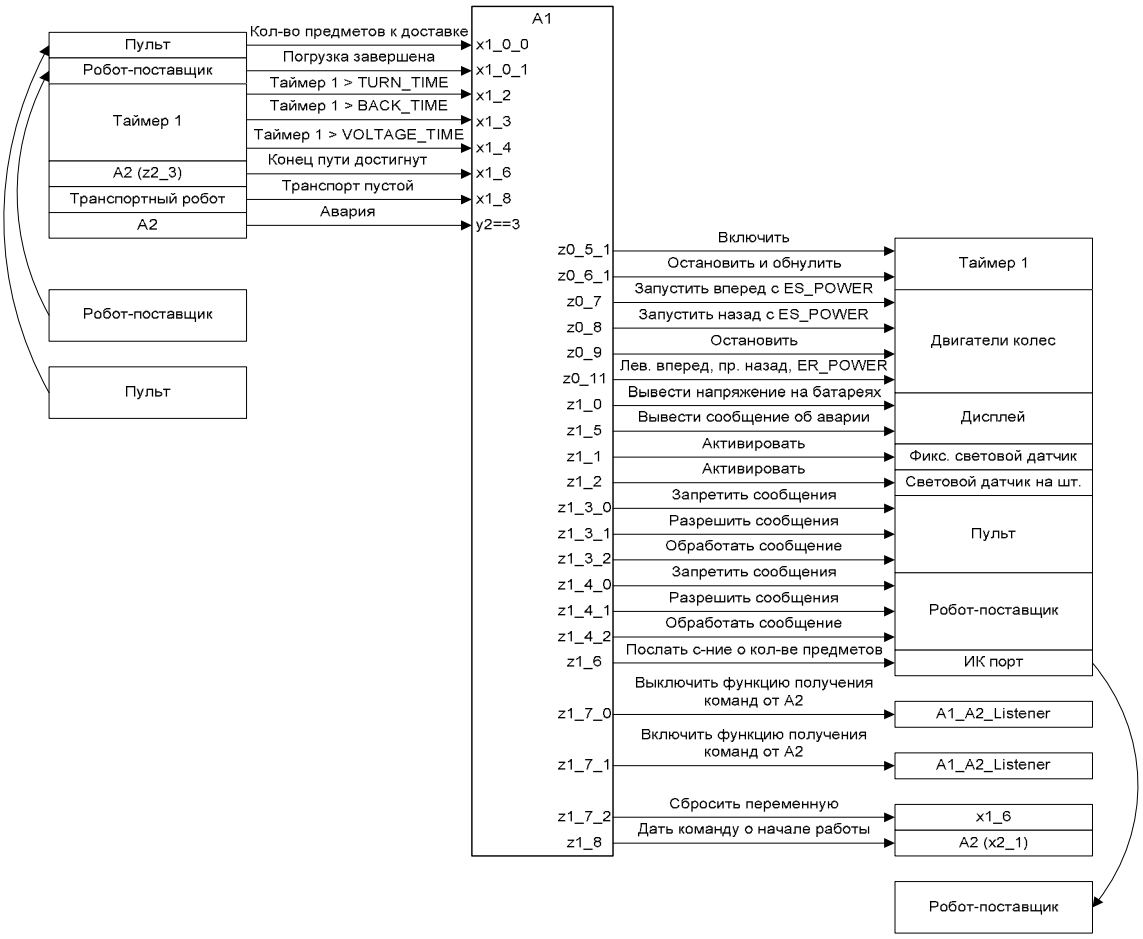

Рис. 10. Схема связей автомата A1. Управление транспортным роботом

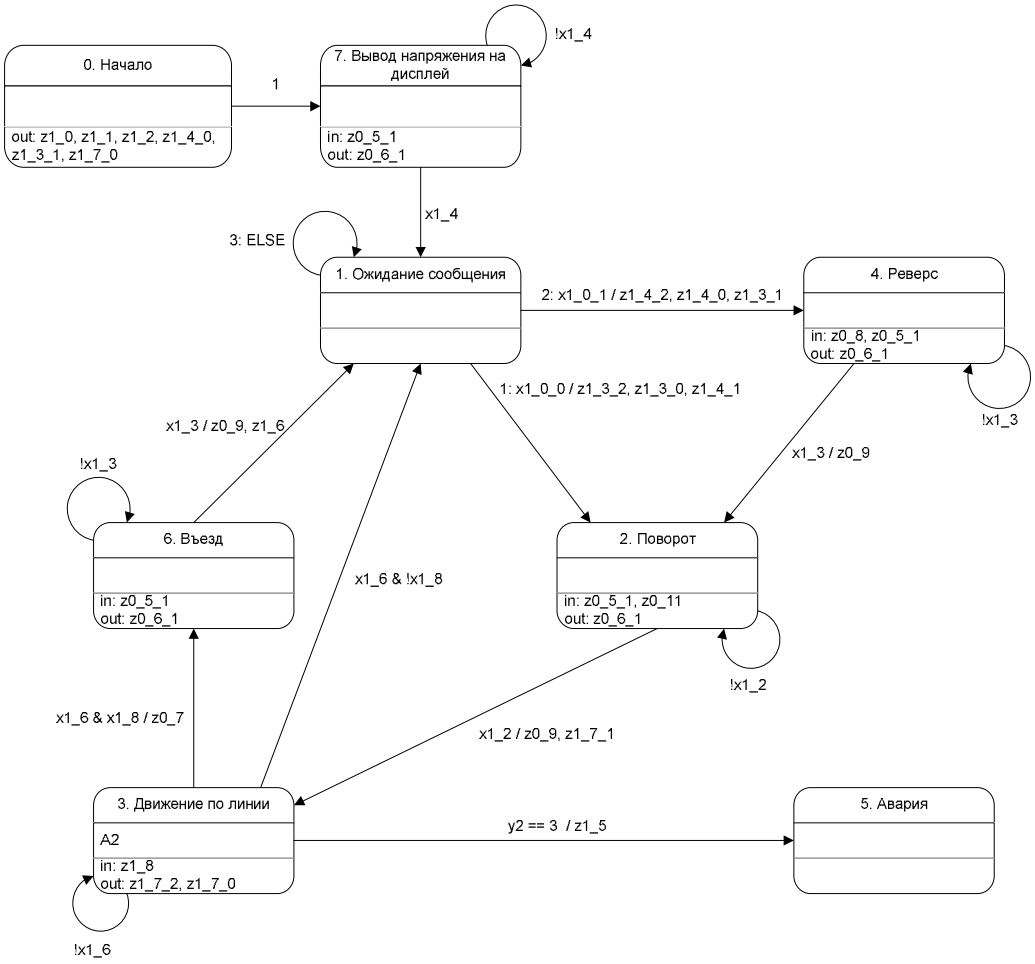

Рис. 11. Граф переходов автомата A1. Управление транспортным роботом

8.5. Автомат A2. Движение по пути

Краткое описание

Логика работы этого автомата реализует алгоритм, описанный в разд. 8.2.

Автомат `A2` управляет транспортным роботом при его движении по пути. Робот едет равномерно по пути, либо пока не доедет до конца пути (переменная `x0_4_2 == true`), либо не съедет с пути на поле (переменная `x0_4_0 == true`). В случае, если робот съезжает с пути, то автомат `A2` при помощи штанги ищет потерянный путь и возвращает робот обратно на этот путь.

Перечень констант

`ROT_EX_TIME` – время, необходимое для поворота робота влево или вправо для того, чтобы световой датчик 1 сместился от границы пути к его центру.

`CORRECT_TIME` – время, необходимое для движения вперед с целью подтверждения цвета светового датчика 1.

`ES_POWER` – значение мощности работы двигателей колес для движения вперед и назад.

`ER_POWER` – значение мощности работы двигателей колес для поворота.

ROT_TIME – максимальное время поворота транспортного робота, если он исправен.

Блокировки

y3 == 5 – авария. Путь не был найден.

Перечень входных переменных

x0_4_0 – значение светового датчика 1 соответствует полю.

x0_4_1 – значение светового датчика 1 соответствует пути.

x0_4_2 – значение светового датчика 1 соответствует фольге.

x0_4_3 – значение светового датчика 1 ничему не соответствует.

x0_5_0 – значение светового датчика 2 соответствует полю.

x0_5_1 – значение светового датчика 2 соответствует пути.

x0_5_2 – значение светового датчика 2 соответствует фольге.

x0_5_3 – значение светового датчика 2 ничему не соответствует.

x2_1 – получена команда от автомата A1 посредством выходного воздействия z1_8, о том, что следует начинать работу. На схеме связей и далее по тексту последнее обозначается как A1(z1_8).

x2_3 – получена команда от автомата A3 посредством выходного воздействия z3_0, о том, что путь найден слева. На схеме связей и далее по тексту последнее обозначается как A3(z3_0).

x2_4 – получена команда от автомата A3 посредством выходного воздействия z3_1, о том, что путь найден справа. На схеме связей и далее по тексту последнее обозначается как A3(z3_1).

x2_7 – таймер 2 превысил значение CORRECT_TIME.

x2_8 – таймер 2 превысил значение ROT_EX_TIME.

x2_9 – таймер 2 превысил значение ROT_TIME.

Перечень состояний y2

0 – Начало пути.

1 – Движение вперед.

2 – Подтверждение цвета.

3 – Авария.

4 – Корректировка.

5 – Поиск пути.

6 – Поворот.

7 – Центрирование по пути.

Перечень выходных воздействий

z0_5_2 – включить таймер 2.

z0_6_2 – остановить и обнулить таймер 2.

z0_7 – запустить левый и правый двигатели колес вперед со значением мощности ES_POWER.

z0_9 – остановить левый и правый двигатели колес.

z0_10 – для поворота налево запустить левый двигатель назад, а правый вперед со значением мощности ER_POWER.

z0_11 – для поворота направо запустить левый двигатель вперед, а правый назад со значением мощности ER_POWER.

z2_3 – выдать команду автомату A1 посредством входной переменной x1_6 о том, что достигнут конец пути. На схеме связей и далее по тексту последнее обозначается как A1(x1_6).

z2_4_0 – запретить получение команд от автомата A1(A2_A1_Listener = false).

z2_4_1 – разрешить получение команд от автомата A1(A2_A1_Listener = true).

z2_4_2 – сбросить переменную (x2_1).

z2_5_0 – запретить получение команд от автомата A3(A2_A3_Listener = false).

z2_5_1 – разрешить получение команд от автомата A3(A2_A3_Listener = true).

z2_5_2 – сбросить переменные x2_3 и x2_4.

z2_6 – дать команду автомату A3 посредством переменной x3_1 о начале работы. На схеме связей и далее по тексту последнее обозначается как A3(x3_1).

На рис. 12 приведена схема связей автомата A2, а на рис. 13 – его граф переходов.

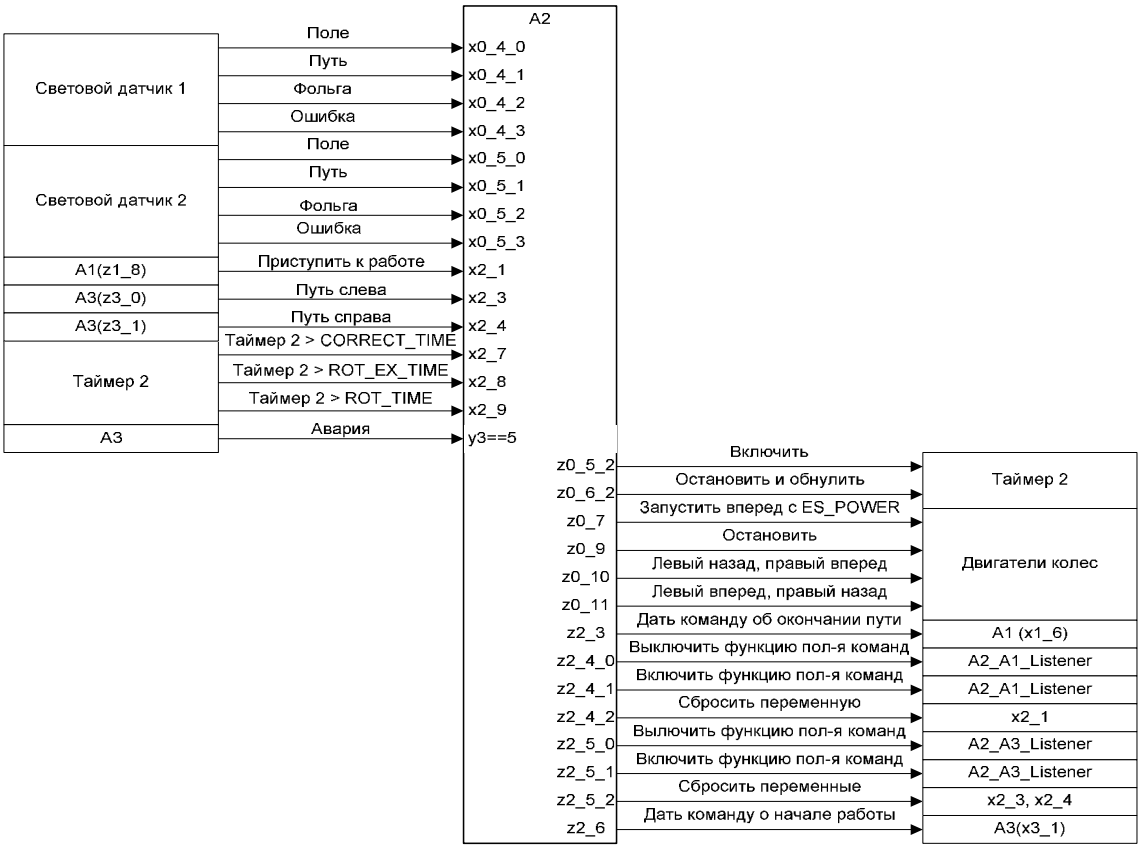

Рис. 12. Схема связей автомата A2. Транспортный робот. Движение по пути

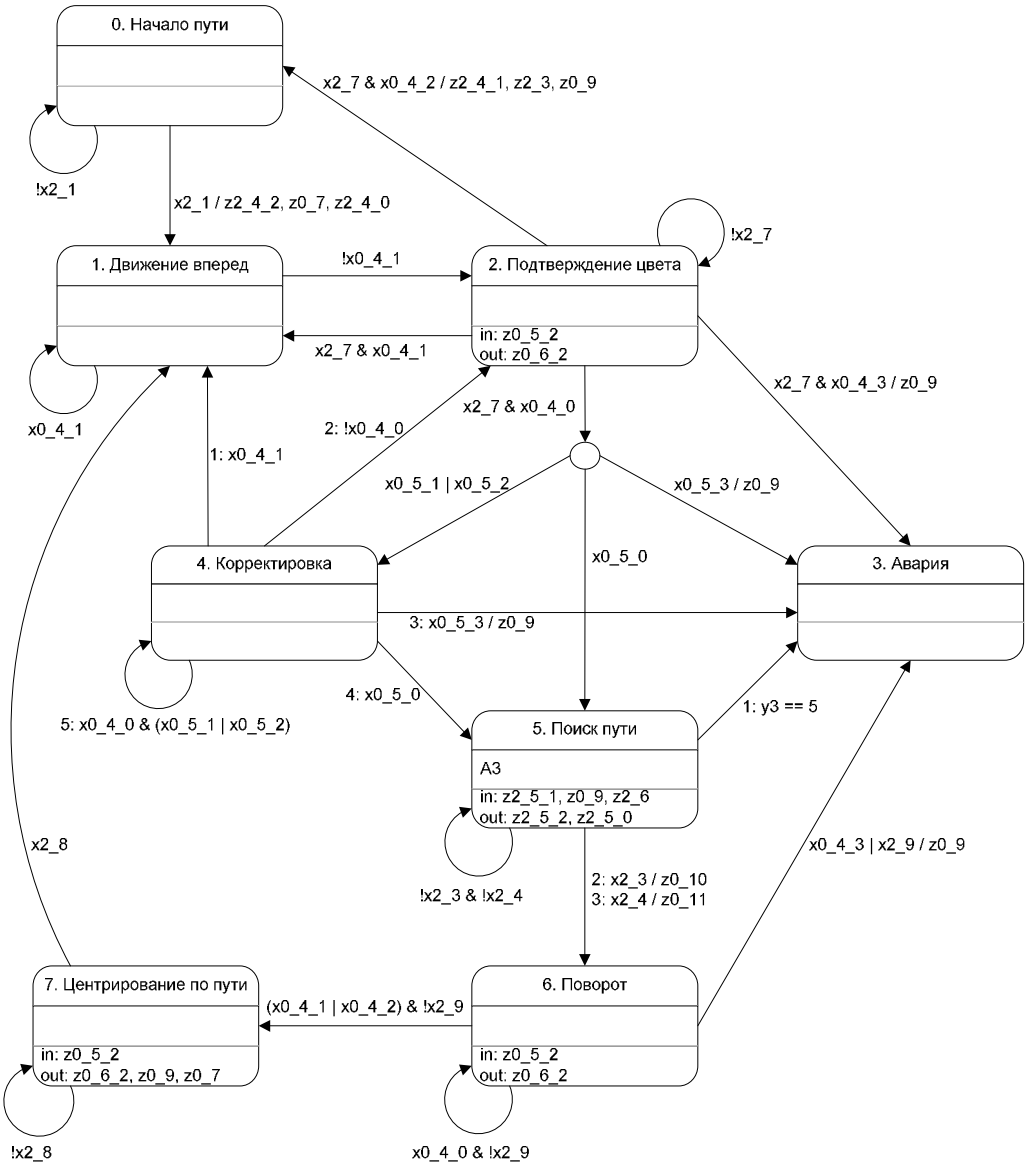

Рис. 13. Граф переходов автомата A2. Транспортный роб от. Движение по пути

8.6. Автомат A3. Управление штангой

Краткое описание

Автомат `A3` ищет путь при помощи штанги. Вначале автомат запускает штангу для движения налево. Если путь слева не найден, то он ищет путь справа. Если путь был найден либо слева, либо справа, то автомату `A2` дается команда о том, что путь был найден соответственно слева или справа. В противном случае автомат `A3` переходит в состояние “Авария”.

Перечень констант

`RR_TIME` – константа, соответствующая времени, за которое штанга повернется на 180º.

`ERR_POWER` – значение мощности работы двигателя штанги.

Перечень входных переменных

x0_5_0 – значение светового датчика 2 соответствует полю.

x0_5_1 – значение светового датчика 2 соответствует пути.

x0_5_2 – значение светового датчика 2 соответствует фольге.

x0_5_3 – значение светового датчика 2 ничему не соответствует.

x3_1 – получена команда от автомата A2 посредством выходного воздействия z2_6 о том, что требуется начать работу. На схеме связей и далее по тексту последнее обозначается как A2(z2_6).

x3_3 – таймер 3 превысил значение RR_TIME.

x3_5 – получена команда от автомата A5 посредством выходного воздействия z5_4 о том, что штанга центрирована. На схеме связей и далее по тексту последнее обозначается как A5(z5_4).

Перечень состояний

0 – Штанга по центру.

1 – Поиск пути слева.

2 –Центрирование слева.

3 – Поиск пути справа.

4 – Центрирование. Путь не найден.

5 – Авария.

6 – Центрирование справа.

Перечень выходных воздействий

z0_5_3 – включить таймер 3.

z0_6_3 – остановить и обнулить таймер 3.

z3_0 – дать команду автомату A2 посредством входной переменной x2_3, о том, что путь найден слева. На схеме связей и далее по тексту последнее обозначается как A2(x2_3).

z3_1 – дать команду автомату A2 посредством входной переменной x2_4, о том, что путь найден справа. На схеме связей и далее по тексту последнее обозначается как A2(x2_4).

z3_2_0 – запретить получение команд от автомата A2 (A3_A2_Listener = false).

z3_2_1 – разрешить получение команд от автомата A2 (A3_A2_Listener = true).

z3_2_2 – сбросить переменную x3_1.

z3_4_0 – запретить получение команд от автомата A5 (A3_A5_Listener = false).

z3_4_1 – разрешить получение команд от автомата A5 (A3_A5_Listener = true).

z3_4_2 – сбросить переменную x3_5

z3_5 – запустить двигатель штанги влево с мощностью ERR_POWER.

z3_6 – запустить двигатель штанги вправо с мощностью ERR_POWER.

z3_7 – остановить двигатель штанги.

z3_8 – выдать команду A5 посредством входной переменной x5_1 о том, что требуется центрировать штангу. На схеме связей и далее по тексту последнее обозначается как A5(x5_1).

На рис. 14 приведена схема связей автомата A3, а на рис. 15 – его граф переходов.

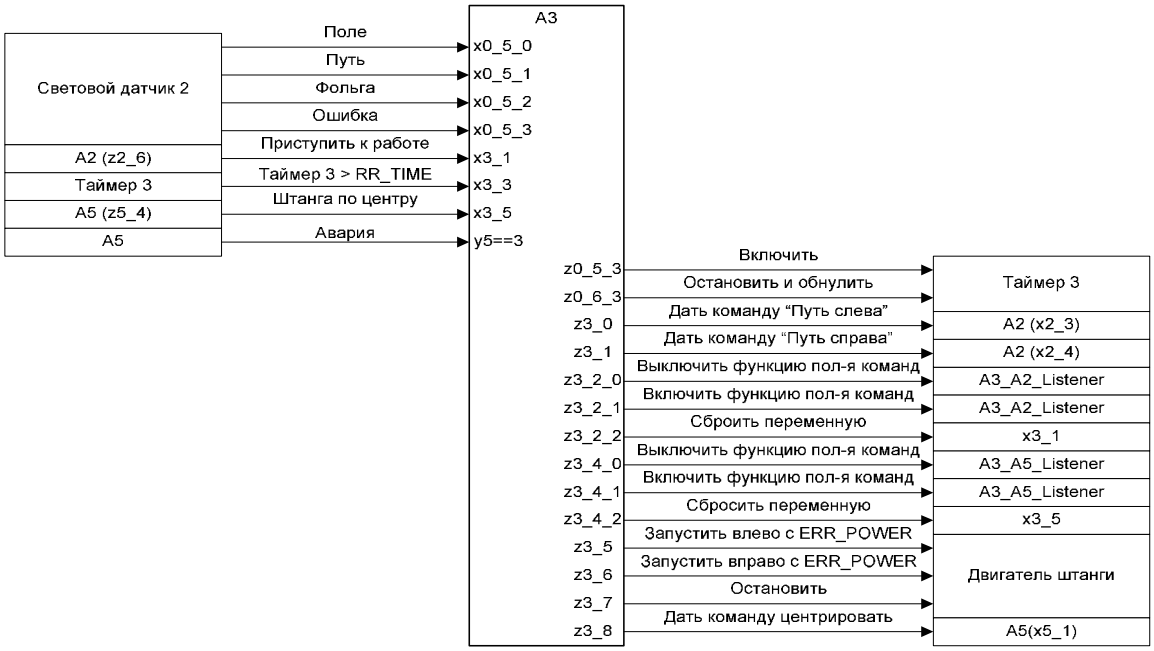

Рис. 14. Схема связей автомата A3. Транспортный робот. Управление штангой

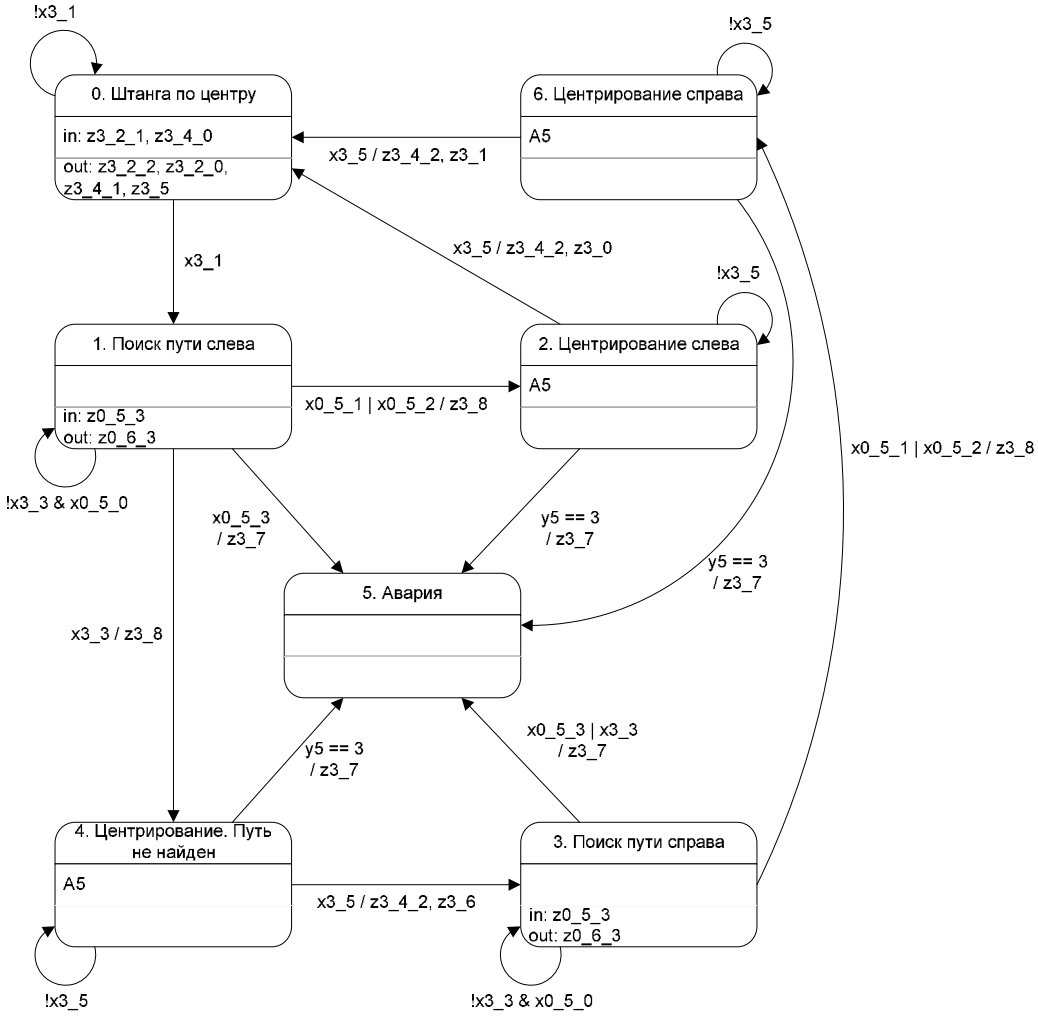

Рис. 15. Граф переходов автомата A3. Транспортный робот. Управление штангой

8.7. Автомат A5. Центрирование штанги

Краткое описание

Автомат A5 выполняет центрирование штанги путем изменения направления движения штанги и отслеживания показаний датчика касаний 3.

Перечень констант

`RB_TIME` – константа, соответствующая максимальному времени возвращения штанги, если она исправна.

Перечень входных переменных

`x5_0` – датчик касания 3 штанги нажат.

`x5_1` – пришла команда от автомата `A3` посредством выходного воздействия `z3_8` о том, что надо начать работу. На схеме связей и далее по тексту последнее обозначается как `A3(z3_8)`.

`x5_2` – таймер 4 превысил значение `RB_TIME`.

Перечень состояний

0 – Ожидание сообщения.

1 – Поиск датчика.

2 – Движение обратно.

3 – Авария.

Перечень выходных воздействий

`z0_5_4` – включить таймер 4.

`z0_6_4` – остановить и обнулить таймер 4.

`z5_1_0` – запретить получение команд от автомата `A3 (A5_A3_Listener = false)`.

`z5_1_1` – разрешить получение команд от автомата `A3 (A5_A3_Listener = true)`.

`z5_1_2` – сбросить переменную `x5_1`.

`z5_2` – изменить направление движения штанги.

`z5_3` – остановить штангу.

`z5_4` – дать команду автомату `A3` посредством входной переменной `x3_5` о том, что штанга по центру. На схеме связей и далее по тексту последнее обозначается как `A5(x5_1)`.

На рис. 16 приведена схема связей автомата `A5`, а на рис. 17 – его граф переходов.

Рис. 16. Схема связей автомата A5. Транспортный робот. Центрирование штанги

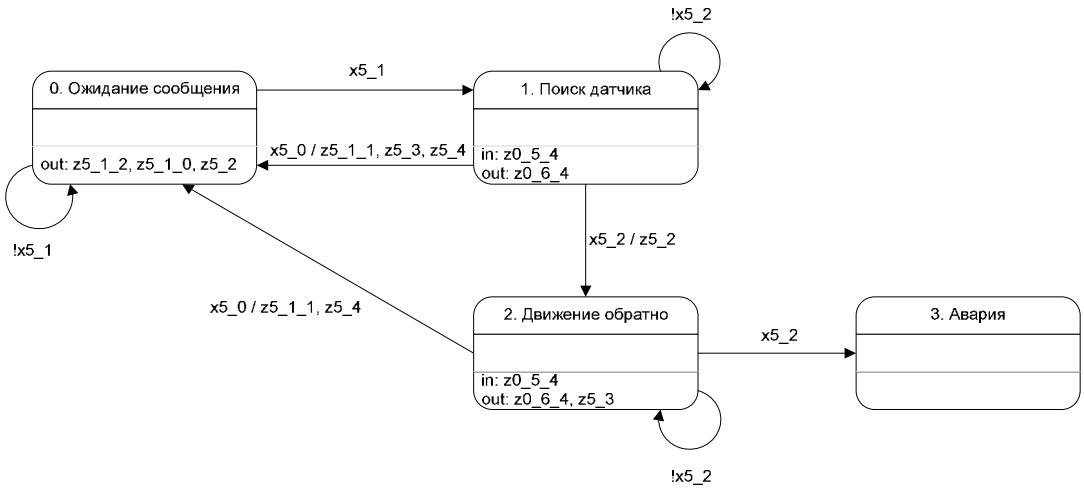

Рис. 17. Граф переходов автомата A5. Транспортный робот. Центрирование штанги

9. Робот-поставщик

На рис. 18 приведен виртуальный робот-поставщик, сконструированный с помощью программы *MLCAD*. Файл с описанием робота-поставщика размещен по адресу http://is.ifmo.ru/projects/lego

Рис. 18. Виртуальный робот-поставщик

Робот-поставщик, как следует из приведенного рисунка, состоит из двух основных частей: устройство для выдачи предметов и башня.

На роботе-поставщике установлены двигатель и сенсор касания, при помощи которого можно подсчитывать количество оборотов сбрасывающего рычажка. При каждом обороте рычажка, выдается один предмет с наклонной плоскости. Таким образом, обеспечивается выдача необходимого количества предметов. Поддон транспортного робота находится сбоку, устройство для выдачи предметов также находится также сбоку от микроконтроллера, обеспечивающего функционирование робота-поставщика.

На рис. 19 приведен робот-поставщик.

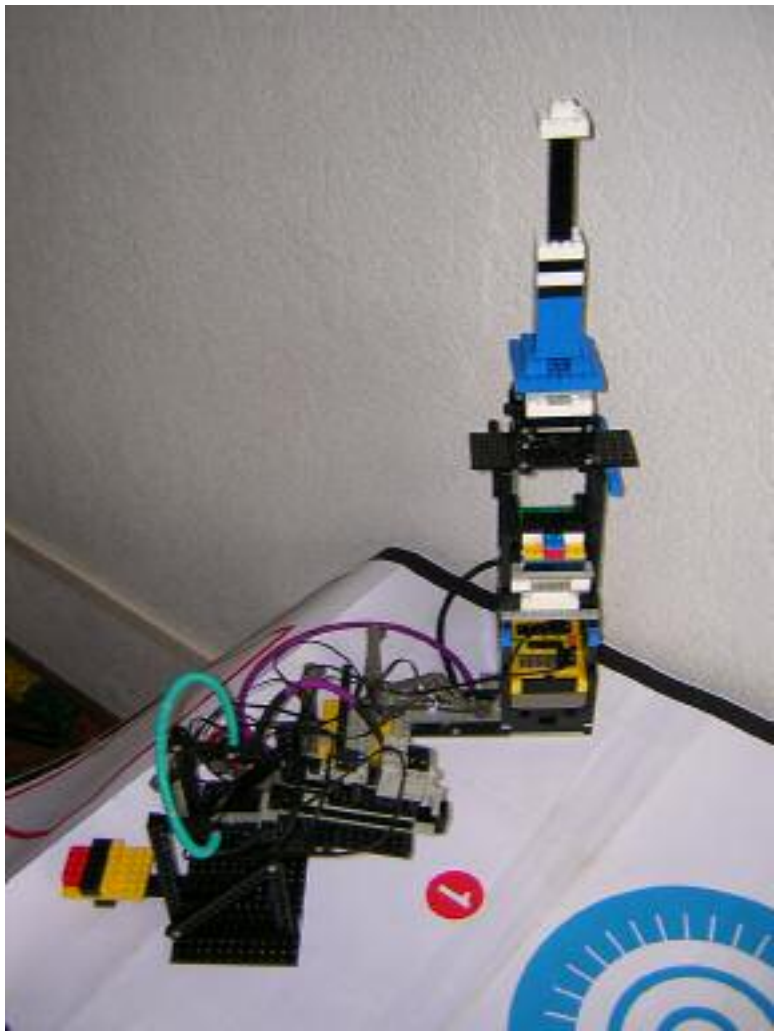

Рис. 19. Робот-поставщик

На рис. 20 показан микроконтроллер робота-поставщика, расположенный в основании башни.

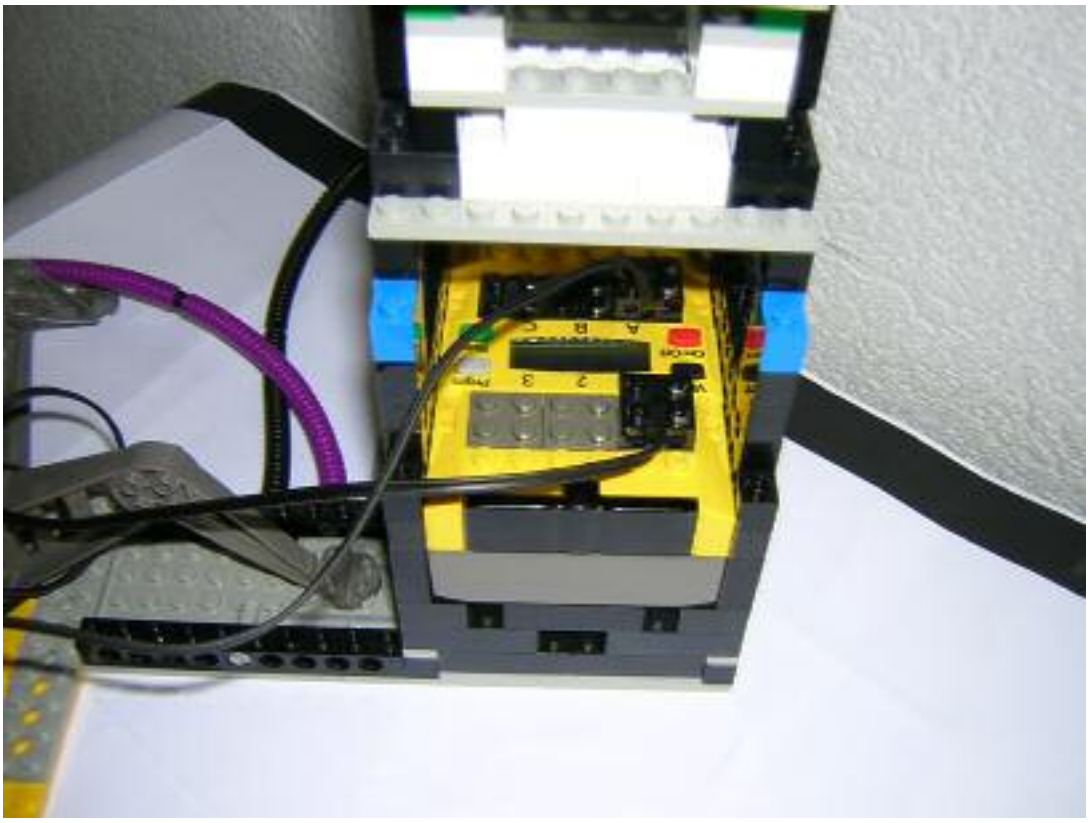

Рис. 20. Микроконтроллер робота-поставщика

На рис. 21 показано взаимодействие роботов при выдаче предметов.

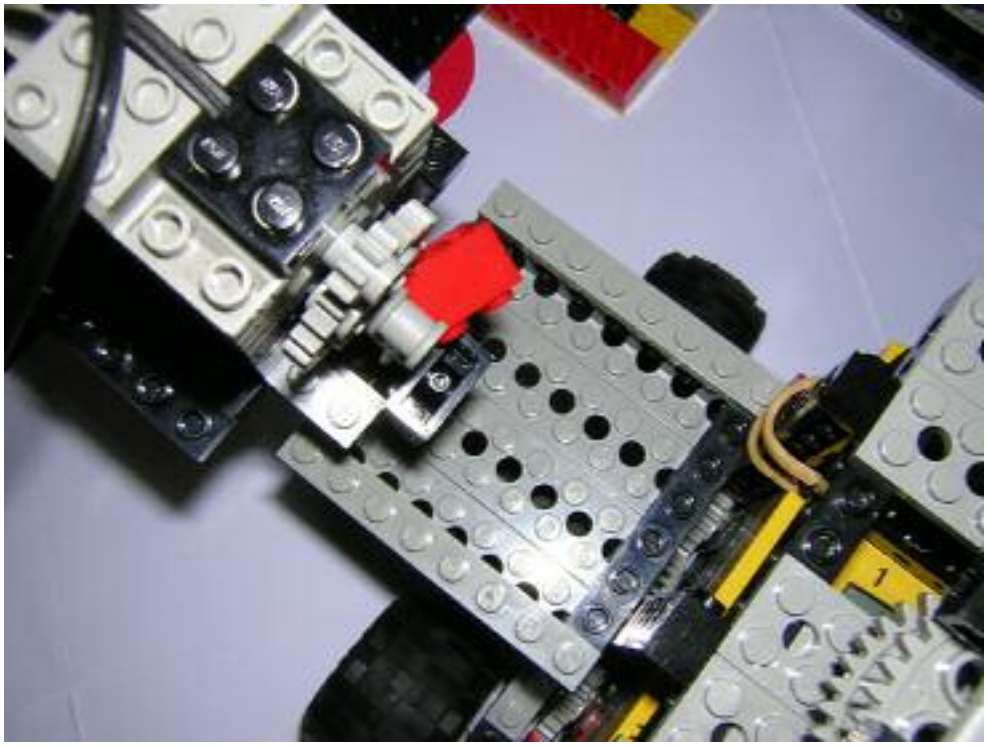

Рис. 21. Выдача предмета

На рис. 22 показано взаимодействие роботов после выдачи предмета.

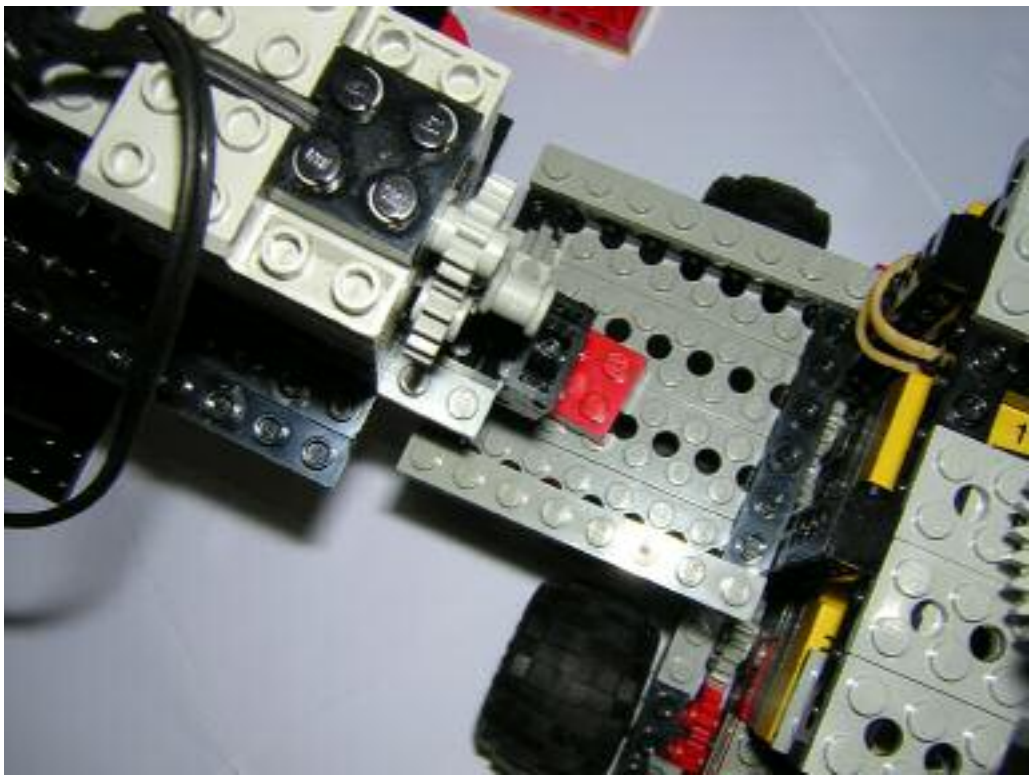

Рис. 22. После выдачи предмета

10. Управление роботом-поставщиком

10.1. Спецификация на управляющую систему

Управляющая система робота-поставщика при запуске должна сначала вывести информацию о напряжении на батареях, затем ожидать сообщения о начале выдачи от транспортного робота. После получения этого сообщения, система должна выдать указанное количество предметов. После их выдачи, система вновь должна ожидать дальнейшие сообщения от транспортного робота.

10.2. Автомат A0

Краткое описание

Автомат `A0` ожидает сообщения от транспортного робота о количестве предметов к доставке. После получения этого сообщения автомат включает двигатель *B* (выходное воздействие `z0_1`) и не выключает, пока необходимое количество предметов не будет погружено. После этого автомат посылает сообщение транспортному роботу об окончании погрузки (выходное воздействие `z0_4`).

Перечень констант

Здесь перечислены все константы, определенные для робота-поставщика. Значения констант, определенных опытным путем, приведены в приложении 4.

VOLTAGE_TIME – время, в течение которого на дисплей выводится напряжение на батареях. Измеряется в миллисекундах.

ES_POWER – значение мощности работы двигателя для выдачи предметов.

Перечень переменных

candyCount – количество предметов для выдачи. Начальное значение – ноль.

Перечень входных переменных

x0_0 – датчик касания нажат.

x0_1 – перменная candyCount больше нуля.

x0_2 – получено сообщение от транспортного робота о количестве предметов для выдачи.

x0_3 – таймер превысил значение VOLTAGE_TIME.

Перечень состояний y0

0 – Начало.

1 – Ожидание сообщения.

2 – Выдача предмета.

3 – Подготовка к выдаче.

4 – Вывод напряжения на дисплей.

Перечень выходных воздействий

z0_0 – вывести на дисплей напряжение на батареях.

z0_1 – запустить двигатель со значением мощности ES_POWER.

z0_2 – остановить двигатель.

z0_3 – уменьшить на единицу количество предметов, которые необходимо выдать.

z0_4 – послать сообщение транспортному роботу об окончании погрузки.

z0_5_0 – запретить сообщения от робота-поставщика.

z0_5_1 – разрешить сообщения от робота-поставщика.

z0_5_2 – обработать сообщение от робота-поставщика.

z0_6 – включить таймер.

z0_7 – остановить и обнулить таймер.

На рис. 23 приведена схема связей автомата A0, а на рис. 24 – его граф переходов.

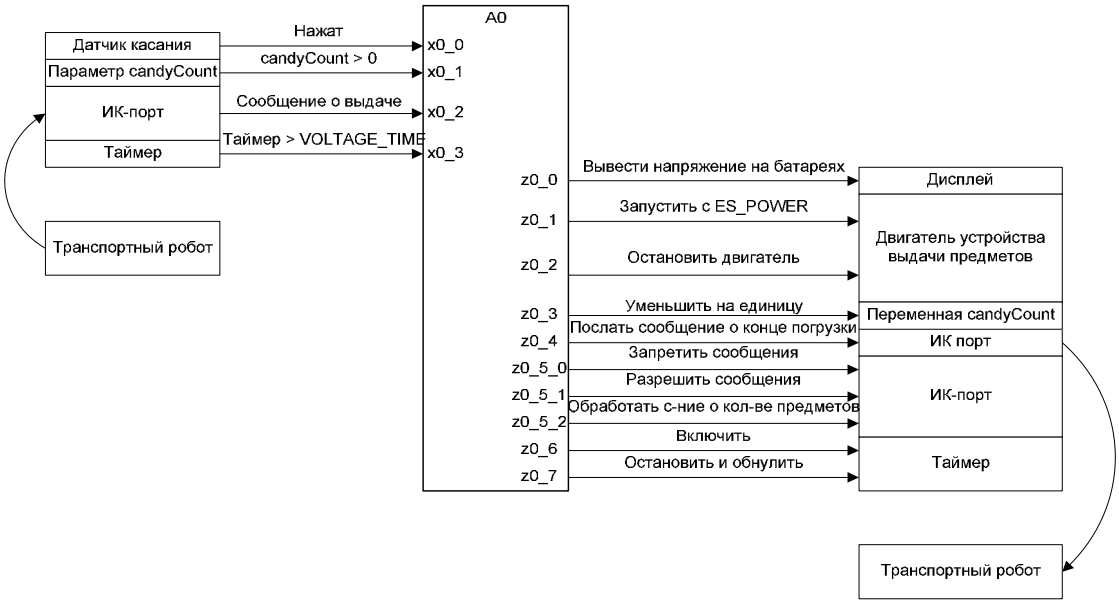

Рис. 23. Схема связей автомата A0. Управление роботом-поставщиком

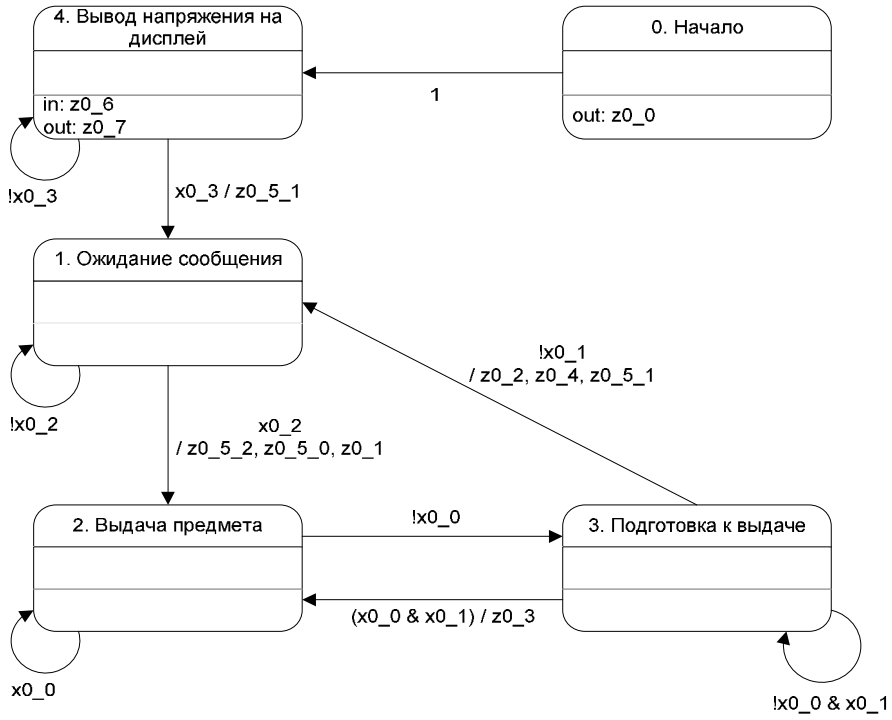

Рис. 24. Граф переходов автомата A0. Управление роботом-поставщиком

Заключение

Спроектированная система управления транспортным роботом и роботом-поставщиком решает поставленную задачу и позволяет четко проследить последовательность действий, выполняемых роботами, в различных ситуациях. Вся логика работы вынесена в графы переходов. Отладка управляющих программ даже в таких “аскетических” условиях, при которых на дисплей можно вывести только четыре цифры, сильно упрощается. Для того чтобы быть уверенным, что написанная программа соответствует схемам связи и графам переходов, достаточно проверить корректность выполнения каждого перехода.

Проект *Isenguard* демонстрирует возможность успешного и эффективного применения автоматного подхода для проектирования систем управления роботов *Lego Mindstorms* и аналогичных роботов.

Список литературы

1. NQC – Not Quite C. http://bricxcc.sourceforge.net/nqc/

2. OS legos. http://sourceforge.net/projects/legos/

3. OS *leJOS*. http://lejos.sourceforge.net/

4. *Baum D.* Definitive Guide to Lego Mindstroms, NY: Appress, 2000.

5. *Baum D., Gasperi M., Hempel R., Villa L.* Extreme Mindstroms. An advanced guide to Lego Mindstorms. NY: Appress, 2000.

6. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

7. *Шалыто А.А., Туккель Н.И.* SWITCH-технология — автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. 2001. № 5. http://is.ifmo.ru/works/switch/1/

8. *Шалыто А.А., Туккель Н.И.* Система управления дизель-генератором. Проектная документация. http://is.ifmo.ru/projects/dg/

9. *Киви Б.* Сделай сам // Компьютерра. 2002. № 5.

10. *Brown JP*. Serious Lego. http://jpbrown.i8.com/cubesolver.html

11. *Crosby M.* Lego projects. http://www.mastincrosbie.com/mark/lego/lego.html

12. *Iversen T., Kristoffersen K., Larsen K., et al.* Model-Checking Real-Time Control Programs. Verifying Lego Mindstorms Systems Using UPPAAL // 12 Euromicro Conference on Real-Time Systems 2000.

13. *Alur R., Kannan S., Yannakakis M.* Communicating hierarchical state machines // *Proceedings of the 26th International Colloquium on Automata, Languages, and Programming*, LNCS 1644, 1999. pp. 169–178. http://cis.upenn.edu/~alur/Icalp99chsm.html

14. *Alur R.* Timed Automata. NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems. http://www.cis.upenn.edu/~alur/Nato97.ps.gz

15. *Alur R., Dill D.* Automata-theoretic Verification of Real-Time Systems // *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science,* 1992. pp. 177-186.

16. *Laursen M, Madsen R., and Mortensen S.* Verifying Distributed LEGO RCX Programs Using UPPAAL. http://citeseer.ist.psu.edu/laursen99verifying.html

17. *Шалыто А.А., Туккель Н. И.* Танки и автоматы // Byte/Россия. 2003. № 2. http://is.ifmo.ru/download/tanks_new.pdf

18. *Туккель Н.И., Шалыто А.А.* Система управления танком для игры “Robocode”. Версия 1. Проектная документация. http://is.ifmo.ru/projects/tanks/

19. *Кузнецов Д.В., Шалыто А.А.* Система управления танком для игры “Robocode”. Версия 2. http://is.ifmo.ru/projects/robocode2/

20. *Шалыто А.А.* Алгоритмизация и программирование для систем логического управления и реактивных систем // Автоматика и телемеханика. 2001. № 1. http://is.ifmo.ru/works/app-aplu/1/

21. *Шалыто А.* Новая инициатива в программировании. Движение за открытую проектную документацию // Мир ПК. 2003. № 9. http://is.ifmo.ru/works/open_doc/

22. *Буч Г., Рамбо Д., Якобсон И.* UML. Руководство пользователя, М.:ДМК. 2000.

23. *leJOS* infrared serial protocol. http://graphics.stanford.edu/~kekoa/rcx/#Protocol

Приложение 1. Java-код программы транспортного робота

```
import josx.platform.rcx.*;

public class Transport {

    public static final int VOLTAGE_TIME = 2000;
    public static final int TURN_TIME = 15000;
    public static final int BACK_TIME = 3000;

    public static final int ROT_EX_TIME = 500;
    public static final int ROT_TIME = 12000;
    public static final int CORRECT_TIME = 500;
    public static final int ES_POWER = 4;
    public static final int ER_POWER = 4;

    public static final int ERR_POWER = 3;
    public static final int RR_TIME = 4000;
    public static final int RB_TIME = 8000;

    public static final int BLACK_UP = 41;
    public static final int BLACK_DOWN = 20;
    public static final int WHITE_UP = 63;
    public static final int WHITE_DOWN = 42;
    public static final int FOIL_UP = 80;
    public static final int FOIL_DOWN = 64;
    public static final int OTHER_UP = 0;
    public static final int OTHER_DOWN = 0;

    public static final Motor motorLeft = Motor.A;
    public static final Motor motorRight = Motor.C;
    public static final Motor motorRod = Motor.B;
    public static final Sensor lightFixed = Sensor.S1;
    public static final Sensor lightRod = Sensor.S2;
    public static final Sensor pressRod = Sensor.S3;

    public static boolean trEmpty = false;

    public static boolean A1_pathFinish = false;
    public static boolean A1_A2Listener = false;

    public static boolean A2_tracePath = false;
    public static boolean A2_A1Listener = true;
    public static boolean A2_pathFound = false;
    public static boolean A2_pathDir = false;
    public static boolean A2_A3Listener = false;

    public static boolean A3_findPath = false;
    public static boolean A3_A2Listener = true;
    public static boolean A3_pathFound = false;
    public static boolean A3_A5Listener = true;

    public static boolean A5_findPath = false;
    public static boolean A5_A3Listener = true;

    public static byte candyCount = 0;
    public static int timerStart1 = 0;
    public static int timerStart2 = 0;
    public static int timerStart3 = 0;
    public static int timerStart4 = 0;

    public static byte packet[] = new byte[10];
```

```
    public static byte sendPacket[] = new byte[2];

    public static byte remoteValue = -1;
    public static boolean remoteSignal = false;
    public static boolean dispencerSignal = false;
    public static boolean remoteActive = true;
    public static boolean dispencerActive = true;


    public static int y1 = 0;
    public static int y2 = 0;
    public static int y3 = 0;
    public static int y5 = 0;

    public static boolean y1changed = false;
    public static boolean y2changed = false;
    public static boolean y3changed = false;
    public static boolean y5changed = false;

    public static void updateSignals() {
        while (Serial.isPacketAvailable()) {
            Serial.readPacket(packet);

            if ((packet[0] & 255) != 210) {
                if (dispencerActive) {
                    dispencerSignal = true;
                }
            } else {
                if (!remoteSignal && remoteActive) {

                    int c1 = packet[1] & 255;
                    int c2 = packet[2] & 255;

                    if (c1 == 0) {
                        if (c2 == 0x01)
                            remoteValue = 1;
                        else if (c2 == 0x02)
                            remoteValue = 2;
                        else if (c2 == 0x04) remoteValue = 3;
                    }
                    remoteSignal = true;
                }
            }
        }
    }

    //------------------------------------------------------------------------
    //------------------------------------------------------------------------

    public static boolean x0_4_0() {
        return (lightFixed.readValue() >= WHITE_DOWN &&
lightFixed.readValue() <= WHITE_UP);
    }

    public static boolean x0_4_1() {
        return (lightFixed.readValue() >= BLACK_DOWN &&
lightFixed.readValue() <= BLACK_UP);
    }

    public static boolean x0_4_2() {
        return (lightFixed.readValue() >= FOIL_DOWN && lightFixed.readValue()
<= FOIL_UP);
```

```
    }

    public static boolean x0_4_3() {
        return (lightFixed.readValue() >= OTHER_DOWN &&
lightFixed.readValue() <= OTHER_UP);
    }

    public static boolean x0_5_0() {
        return (lightRod.readValue() >= WHITE_DOWN && lightRod.readValue() <=
WHITE_UP);
    }

    public static boolean x0_5_1() {
        return (lightRod.readValue() >= BLACK_DOWN && lightRod.readValue() <=
BLACK_UP);
    }

    public static boolean x0_5_2() {
        return (lightRod.readValue() >= FOIL_DOWN && lightRod.readValue() <=
FOIL_UP);
    }

    public static boolean x0_5_3() {
        return (lightRod.readValue() >= OTHER_DOWN && lightRod.readValue() <=
OTHER_UP);
    }

    //-------------------------------------------------------------------------
    //-------------------------------------------------------------------------

    public static boolean x1_0_0() {
        updateSignals();
        return remoteSignal;
    }

    public static boolean x1_0_1() {
        updateSignals();
        return dispencerSignal;
    }

    public static boolean x1_2() {
        int timePassed = ((int) System.currentTimeMillis()) - timerStart1;
        return timePassed > TURN_TIME;
    }

    public static boolean x1_3() {
        int timePassed = ((int) System.currentTimeMillis()) - timerStart1;
        return timePassed > BACK_TIME;
    }

    public static boolean x1_4() {
        int timePassed = ((int) System.currentTimeMillis()) - timerStart1;
        return timePassed > VOLTAGE_TIME;
    }

    public static boolean x1_6() {
        return A1_pathFinish;
    }

    public static boolean x1_8() {
        return trEmpty;
    }
```

```
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

public static boolean x2_1() {
    return A2_tracePath;
}

public static boolean x2_3() {
    return A2_pathFound && !A2_pathDir;
}

public static boolean x2_4() {
    return A2_pathFound && A2_pathDir;
}

public static boolean x2_7() {
    int timePassed = ((int) System.currentTimeMillis()) - timerStart2;
    return timePassed > CORRECT_TIME;
}

public static boolean x2_8() {
    int timePassed = ((int) System.currentTimeMillis()) - timerStart2;
    return timePassed > ROT_EX_TIME;
}

public static boolean x2_9() {
    int timePassed = ((int) System.currentTimeMillis()) - timerStart2;
    return timePassed > ROT_TIME;
}

//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

public static boolean x3_1() {
    return A3_findPath;
}

public static boolean x3_3() {
    int timePassed = ((int) System.currentTimeMillis()) - timerStart3;
    return timePassed > RR_TIME;
}

public static boolean x3_5() {
    return A3_pathFound;
}

//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

public static boolean x5_0() {
    return pressRod.readBooleanValue();
}

public static boolean x5_1() {
    return A5_findPath;
}

public static boolean x5_2() {
    int timePassed = ((int) System.currentTimeMillis()) - timerStart4;
    return timePassed > RB_TIME;
}
```

```
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
public static void z0_5_1() {
    timerStart1 = (int) System.currentTimeMillis();
}

public static void z0_6_1() {
    timerStart1 = 0;
}

public static void z0_5_2() {
    timerStart2 = (int) System.currentTimeMillis();
}

public static void z0_6_2() {
    timerStart2 = 0;
}

public static void z0_5_3() {
    timerStart3 = (int) System.currentTimeMillis();
}

public static void z0_6_3() {
    timerStart3 = 0;
}

public static void z0_5_4() {
    timerStart4 = (int) System.currentTimeMillis();
}

public static void z0_6_4() {
    timerStart4 = 0;
}

public static void z0_7() {
    motorLeft.setPower(ES_POWER);
    motorRight.setPower(ES_POWER);
    motorLeft.forward();
    motorRight.forward();
}

public static void z0_8() {
    motorLeft.setPower(ES_POWER);
    motorRight.setPower(ES_POWER);
    motorLeft.backward();
    motorRight.backward();
}

public static void z0_9() {
    motorLeft.stop();
    motorRight.stop();
}

public static void z0_10() {
    motorLeft.setPower(ER_POWER);
    motorRight.setPower(ER_POWER);
    motorLeft.backward();
    motorRight.forward();
}

public static void z0_11() {
    motorLeft.setPower(ER_POWER);
    motorRight.setPower(ER_POWER);
```

```
    motorLeft.forward();
    motorRight.backward();
}

//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

public static void z1_0() {
    LCD.showNumber(Battery.getVoltageMilliVolt());
}

public static void z1_1() {
    lightFixed.activate();
}

public static void z1_2() {
    lightRod.activate();
}

public static void z1_3_0() {
    updateSignals();
    remoteActive = false;
}

public static void z1_3_1() {
    updateSignals();
    remoteActive = true;
}

public static void z1_3_2() {
    updateSignals();

    if (remoteSignal) {
        trEmpty = true;
        remoteSignal = false;
        candyCount = remoteValue;
    }
}

public static void z1_4_0() {
    updateSignals();
    dispencerActive = false;
}

public static void z1_4_1() {
    updateSignals();
    dispencerActive = true;
}

public static void z1_4_2() {
    updateSignals();

    if (dispencerSignal) {
        trEmpty = false;
        dispencerSignal = false;
    }
}

public static void z1_5() {
    LCD.showNumber(000);
}

public static void z1_6() {
```

```
    sendPacket[0] = (byte) 0xf7;
    sendPacket[1] = candyCount;
    Serial.sendPacket(sendPacket, 0, 2);
}

public static void z1_7_0() {
    A1_A2Listener = false;
}

public static void z1_7_1() {
    A1_A2Listener = true;
}

public static void z1_7_2() {
    A1_pathFinish = false;
}

public static void z1_8() {
    if (A2_A1Listener) {
        A2_tracePath = true;
    }
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

public static void z2_3() {
    if (A1_A2Listener) {
        A1_pathFinish = true;
    }
}

public static void z2_4_0() {
    A2_A1Listener = false;
}

public static void z2_4_1() {
    A2_A1Listener = true;
}

public static void z2_4_2() {
    A2_tracePath = false;
}

public static void z2_5_0() {
    A2_A3Listener = false;
}

public static void z2_5_1() {
    A2_A3Listener = true;
}

public static void z2_5_2() {
    A2_pathFound = false;
}

public static void z2_6() {
    if (A3_A2Listener) {
        A3_findPath = true;
    }
}

//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------

public static void z3_0() {
    if (A2_A3Listener) {
        A2_pathFound = true;
        A2_pathDir = false;
    }
}

public static void z3_1() {
    if (A2_A3Listener) {
        A2_pathFound = true;
        A2_pathDir = true;
    }
}

public static void z3_2_0() {
    A3_A2Listener = false;
}

public static void z3_2_1() {
    A3_A2Listener = true;
}

public static void z3_2_2() {
    A3_findPath = false;
}

public static void z3_4_0() {
    A3_A5Listener = false;
}

public static void z3_4_1() {
    A3_A5Listener = true;
}

public static void z3_4_2() {
    A3_pathFound = false;
}

public static void z3_5() {
    motorRod.setPower(ERR_POWER);
    motorRod.forward();
}

public static void z3_6() {
    motorRod.setPower(ERR_POWER);
    motorRod.backward();
}

public static void z3_7() {
    motorRod.stop();
}

public static void z3_8() {
    if (A5_A3Listener) {
        A5_findPath = true;
    }
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
```

```
public static void z5_1_0() {
    A5_A3Listener = false;
}

public static void z5_1_1() {
    A5_A3Listener = true;
}

public static void z5_1_2() {
    A5_findPath = false;
}

public static void z5_2() {
    motorRod.reverseDirection();
}

public static void z5_3() {
    motorRod.stop();
}

public static void z5_4() {
    if (A3_A5Listener) {
        A3_pathFound = true;
    }
}


//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

public static void A1S2IN() {
    z0_5_1();
    z0_11();
}

public static void A1S3IN() {
    z1_8();
}

public static void A1S4IN() {
    z0_8();
    z0_5_1();
}

public static void A1S6IN() {
    z0_5_1();
}

public static void A1S7IN() {
    z0_5_1();
}

public static void A1S0OUT() {
    z1_0();
    z1_1();
    z1_2();
    z1_4_0();
    z1_3_1();
    z1_7_0();
}

public static void A1S2OUT() {
    z0_6_1();
```

```
}

public static void A1S3OUT() {
    z1_7_2();
    z1_7_0();
}

public static void A1S4OUT() {
    z0_6_1();
}

public static void A1S6OUT() {
    z0_6_1();
}

public static void A1S7OUT() {
    z0_6_1();
}

//-------------------------------------------------------------------------

public static void A2S2IN() {
    z0_5_2();
}

public static void A2S5IN() {
    z2_5_1();
    z0_9();
    z2_6();
}

public static void A2S6IN() {
    z0_5_2();
}

public static void A2S7IN() {
    z0_5_2();
}

public static void A2S2OUT() {
    z0_6_2();
}

public static void A2S5OUT() {
    z2_5_2();
    z2_5_0();
}

public static void A2S6OUT() {
    z0_6_2();
}

public static void A2S7OUT() {
    z0_6_2();
    z0_9();
    z0_7();
}

//-------------------------------------------------------------------------

public static void A3S0IN() {
  z3_2_1();
        z3_4_0();
```

```
  }

public static void A3S1IN() {
    z0_5_3();
}

public static void A3S3IN() {
    z0_5_3();
}

public static void A3S5IN() {
    z3_7();
}

public static void A3S0OUT() {
    z3_2_2();
        z3_2_0();
        z3_4_1();
    z3_5();
}

public static void A3S1OUT() {
    z0_6_3();
}

public static void A3S3OUT() {
    z0_6_3();
}

//-------------------------------------------------------------------------

public static void A5S1IN() {
    z0_5_4();
}

public static void A5S2IN() {
    z0_5_4();
}

public static void A5S0OUT() {
    z5_1_2();
    z5_1_0();
    z5_2();
}

public static void A5S1OUT() {
    z0_6_4();
}

public static void A5S2OUT() {
    z0_6_4();
    z5_3();
}

//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

public static void A1() {

    int y1_old = y1;

    if (y1 == 0) {
```

```
    if (y1changed) {
        y1changed = false;
    }

    if (true) {
        y1 = 7;
    }

    if (y1 != y1_old) {
        A1S0OUT();
        y1changed = true;
    }

    /*
    if (y1 == 7) {
    }
    */

} else if (y1 == 1) {

    boolean x1_0_1 = x1_0_1();
    boolean x1_0_0 = x1_0_0();

    if (y1changed) {
        y1changed = false;
    }


    if (x1_0_1) {
        y1 = 4;
    } else if (x1_0_0) {
        y1 = 2;
    } else if (true) {
        y1 = 1;
    }

    if (y1 != y1_old) {
        y1changed = true;
    }

    if (y1 == 4) {
        z1_4_2();
        z1_4_0();
        z1_3_1();
    } else if (y1 == 2) {
        z1_3_2();
        z1_3_0();
        z1_4_1();
    } else if (y1 == 1) {

    }

} else if (y1 == 2) {

    boolean x1_2 = x1_2();

    if (y1changed) {
        A1S2IN();
        y1changed = false;
    }

    if (x1_2) {
        y1 = 3;
```

```
    } else if (!x1_2) {
        y1 = 2;
    }

    if (y1 != y1_old) {
        A1S2OUT();
        y1changed = true;
    }

    if (y1 == 3) {
        z0_9();
        z1_7_1();
    } else if (y1 == 2) {

    }

} else if (y1 == 3) {

    boolean x1_6 = x1_6();
    boolean x1_8 = x1_8();

    if (y1changed) {
        A1S3IN();
        y1changed = false;
    }

    A2();

    if (y2 == 3) {
        y1 = 5;
    } else if (x1_6 && x1_8) {
        y1 = 6;
    } else if (x1_6 && !x1_8) {
        y1 = 1;
    } else if (!x1_6) {
        y1 = 3;
    }

    if (y1 != y1_old) {
        A1S3OUT();
        y1changed = true;
    }

    if (y1 == 5) {
        z1_5();
    } else if (y1 == 6) {
        z0_7();
    } else if (y1 == 1) {

    } else if (y1 == 3) {

    }


} else if (y1 == 4) {

    boolean x1_3 = x1_3();

    if (y1changed) {
        A1S4IN();
        y1changed = false;
    }
```

```
    if (x1_3) {
        y1 = 2;
    } else if (!x1_3) {
        y1 = 4;
    }

    if (y1 != y1_old) {
        A1S4OUT();
        y1changed = true;
    }

    if (y1 == 2) {
        z0_9();
    } else if (y1 == 4) {

    }

} else if (y1 == 5) {

    if (y1changed) {
        y1changed = false;
    }

} else if (y1 == 6) {

    boolean x1_3 = x1_3();

    if (y1changed) {
        A1S6IN();
        y1changed = false;
    }

    if (x1_3) {
        y1 = 1;
    } else if (!x1_3) {
        y1 = 6;
    }

    if (y1 != y1_old) {
        A1S6OUT();
        y1changed = true;
    }

    if (y1 == 1) {
        z0_9();
        z1_6();
    } else if (y1 == 6) {

    }

} else if (y1 == 7) {

    boolean x1_4 = x1_4();

    if (y1changed) {
        A1S7IN();
        y1changed = false;
    }

    if (x1_4) {
        y1 = 1;
    } else if (!x1_4) {
        y1 = 7;
```

```
        }

        if (y1 != y1_old) {
            A1S7OUT();
            y1changed = true;
        }

        /*
        if (y1 == 1) {

        } else

        if (y1 == 7) {

        }
        */
    }

    if (y1_old != y1) {
        log(1, y1_old, y1);
    }

}

public static void A2() {

    int y2_old = y2;

    if (y2 == 0) {

        boolean x2_1 = x2_1();

        if (y2changed) {
            y2changed = false;
        }

        if (x2_1) {
            y2 = 1;
        } else if (!x2_1) {
            y2 = 0;
        }

        if (y2 != y2_old) {
            y2changed = true;
        }

        if (y2 == 1) {
            z2_4_2();
            z0_7();
            z2_4_0();
        }

    } else if (y2 == 1) {

        boolean x0_4_1 = x0_4_1();

        if (y2changed) {
            y2changed = false;
        }

        if (!x0_4_1) {
            y2 = 2;
        } else if (x0_4_1) {
```

```
        y2 = 1;
    }

    if (y2 != y2_old) {
        y2changed = true;
    }

} else if (y2 == 2) {

    boolean x2_7 = x2_7();
    boolean x0_4_0 = x0_4_0();
    boolean x0_4_1 = x0_4_1();
    boolean x0_4_2 = x0_4_2();
    boolean x0_4_3 = x0_4_3();
    boolean x0_5_0 = x0_5_0();
    boolean x0_5_1 = x0_5_1();
    boolean x0_5_2 = x0_5_2();
    boolean x0_5_3 = x0_5_3();

    if (y2changed) {
        A2S2IN();
        y2changed = false;
    }

    if (x2_7 && x0_4_3) {
        y2 = 3;
    } else if (x2_7 && x0_4_1) {
        y2 = 1;
    } else if (x2_7 && x0_4_2) {
        y2 = 0;
    } else if (x2_7 && x0_4_0) {
        if (x0_5_0) {
            y2 = 5;
        } else if (x0_5_3) {
            y2 = 3;
        } else if (x0_5_1 || x0_5_2) {
            y2 = 4;
        }
    } else if (!x2_7) {
        y2 = 2;
    }

    if (y2 != y2_old) {
        A2S2OUT();
        y2changed = true;
    }

    if (y2 == 0) {
        z2_4_1();
        z2_3();
        z0_9();
    } else if (y2 == 3) {
        z0_9();
    }

} else if (y2 == 3) {

    if (y2changed) {
        y2changed = false;
    }

} else if (y2 == 4) {
```

```
        boolean x0_4_0 = x0_4_0();
        boolean x0_4_1 = x0_4_1();
        boolean x0_5_0 = x0_5_0();
        boolean x0_5_1 = x0_5_1();
        boolean x0_5_2 = x0_5_2();
        boolean x0_5_3 = x0_5_3();

        if (y2changed) {
            y2changed = false;
        }

        if (x0_4_1) {
            y2 = 1;
        } else if (!x0_4_0) {
            y2 = 2;
        } else if (x0_5_3) {
            y2 = 3;
        } else if (x0_5_0) {
            y2 = 5;
        } else if (x0_4_0 && (x0_5_1 || x0_5_2)) {
            y2 = 4;
        }

        if (y2 != y2_old) {
            y2changed = true;
        }


        if (y2 == 3) {
            z0_9();
        }

    } else if (y2 == 5) {

        boolean x2_3 = x2_3();
        boolean x2_4 = x2_4();

        if (y2changed) {
            A2S5IN();
            y2changed = false;
        }

        A3();

        if (y3 == 5) {
            y2 = 3;
        } else if (x2_3) {
            y2 = 6;
        } else if (x2_4) {
            y2 = 6;
        } else if (!x2_3 && !x2_4) {
            y2 = 5;
        }

        if (y2 != y2_old) {
            A2S5OUT();
            y2changed = true;
        }

        if (y2 == 6) {

            if (x2_3) {
                z0_10();
```

```
            } else if (x2_4) {
                z0_11();
            }
        }

    } else if (y2 == 6) {

        boolean x2_9 = x2_9();
        boolean x0_4_0 = x0_4_0();
        boolean x0_4_1 = x0_4_0();
        boolean x0_4_2 = x0_4_0();
        boolean x0_4_3 = x0_4_0();

        if (y2changed) {
            A2S6IN();
            y2changed = false;
        }

        if ((x0_4_1 | x0_4_2) && !x2_9) {
            y2 = 7;
        } else if (x0_4_3 | x2_9) {
            y2 = 3;
        } else if (x0_4_0 & !x2_9) {
            y2 = 6;
        }

        if (y2 != y2_old) {
            A2S6OUT();
            y2changed = true;
        }
    } else if (y2 == 7) {

        boolean x2_8 = x2_8();

        if (y2changed) {
            A2S7IN();
            y2changed = false;
        }

        if (x2_8) {
            y2 = 1;
        } else if (!x2_8) {
            y2 = 7;
        }

        if (y2 != y2_old) {
            A2S7OUT();
            y2changed = true;
        }
    }


    if (y2_old != y2) {
        log(2, y2_old, y2);
    }

}

public static void A3() {

    int y3_old = y3;

    if (y3 == 0) {
```

```
    boolean x3_1 = x3_1();

    if (y3changed) {
        A3S0IN();
                y3changed = false;
    }

    if (x3_1) {
        y3 = 1;
    } else if (!x3_1) {
        y3 = 0;
    }

    if (y3 != y3_old) {
        A3S0OUT();
        y3changed = true;
    }

} else if (y3 == 1) {

    boolean x3_3 = x3_3();
    boolean x0_5_0 = x0_5_0();
    boolean x0_5_1 = x0_5_1();
    boolean x0_5_2 = x0_5_2();
    boolean x0_5_3 = x0_5_3();

    if (y3changed) {
        A3S1IN();
        y3changed = false;
    }

    if (x3_3) {
        y3 = 4;
    } else if (x0_5_3) {
        y3 = 5;
    } else if (x0_5_1 || x0_5_2) {
        y3 = 2;
    } else if (x0_5_0 && !x3_3) {
        y3 = 1;
    }

    if (y3 != y3_old) {
        A3S1OUT();
        y3changed = true;
    }

    if (y3 == 2) {
        z3_8();
    } else if (y3 == 4) {
        z3_8();
    } else if (y3 == 5) {
          z3_7();
          }

} else if (y3 == 2) {

    boolean x3_5 = x3_5();

    if (y3changed) {
        y3changed = false;
    }
```

```
    A5();

    if (y5 == 3) {
        y3 = 5;
    } else if (x3_5) {
        y3 = 0;
    } else if (!x3_5) {
        y3 = 2;
    }

    if (y3 != y3_old) {
        y3changed = true;
    }

    if (y3 == 0) {
        z3_4_2();
        z3_0();
    } else if (y3 == 5) {
          z3_7();
          }

} else if (y3 == 3) {

    boolean x3_3 = x3_3();
    boolean x0_5_0 = x0_5_0();
    boolean x0_5_1 = x0_5_1();
    boolean x0_5_2 = x0_5_2();
    boolean x0_5_3 = x0_5_3();

    if (y3changed) {
        A3S3IN();
        y3changed = false;
    }

    if (x3_3 | x0_5_3) {
        y3 = 5;
    } else if (x0_5_1 || x0_5_2) {
        y3 = 6;
    } else if (x0_5_0 && !x3_3) {
        y3 = 3;
    }

    if (y3 != y3_old) {
        A3S3OUT();
        y3changed = true;
    }

    if (y3 == 6) {
        z3_8();
    } else if (y3 == 5) {
          z3_7();
          }

} else if (y3 == 4) {

    boolean x3_5 = x3_5();

    if (y3changed) {
        y3changed = false;
    }

    A5();
```

```
        if (y5 == 3) {
            y3 = 5;
        } else if (x3_5) {
            y3 = 3;
        } else if (!x3_5) {
            y3 = 4;
        }

        if (y3 != y3_old) {
            y3changed = true;
        }

        if (y3 == 3) {
            z3_4_2();
            z3_6();
        } else if (y3 == 5) {
              z3_7();
              }

    } else if (y3 == 5) {

        if (y3changed) {
            A3S5IN();
            y3changed = false;
        }

    } else if (y3 == 6) {

        boolean x3_5 = x3_5();

        if (y3changed) {
            y3changed = false;
        }

        A5();

        if (y5 == 3) {
            y3 = 5;
        } else if (x3_5) {
            y3 = 0;
        } else if (!x3_5) {
            y3 = 6;
        }

        if (y3 != y3_old) {
            y3changed = true;
        }

        if (y3 == 0) {
            z3_4_2();
            z3_1();
        }

    }

    if (y3_old != y3) {
        log(3, y3_old, y3);
    }
}

public static void A5() {
    int y5_old = y5;
```

```
if (y5 == 0) {

    boolean x5_1 = x5_1();

    if (y5changed) {
        y5changed = false;
    }

    if (x5_1) {
        y5 = 1;
    } else if (!x5_1) {
        y5 = 0;
    }

    if (y5 != y5_old) {
        A5S0OUT();
        y5changed = true;
    }

} else if (y5 == 1) {

    boolean x5_0 = x5_0();
    boolean x5_2 = x5_2();

    if (y5changed) {
        A5S1IN();
        y5changed = false;
    }

    if (x5_0) {
        y5 = 0;
    } else if (x5_2) {
        y5 = 2;
    } else if (!x5_2) {
        y5 = 1;
    }

    if (y5 != y5_old) {
        A5S1OUT();
        y5changed = true;
    }

    if (y5 == 0) {
        z5_1_1();
        z5_3();
        z5_4();
    } else if (y5 == 2) {
        z5_2();
    }

} else if (y5 == 2) {

    boolean x5_0 = x5_0();
    boolean x5_2 = x5_2();

    if (y5changed) {
        A5S2IN();
        y5changed = false;
    }

    if (x5_0) {
        y5 = 0;
    } else if (x5_2) {
```

```
                y5 = 3;
            } else if (!x5_2) {
                y5 = 2;
            }

            if (y5 != y5_old) {
                A5S2OUT();
                y5changed = true;
            }

            if (y5 == 0) {
                z5_1_1();
                z5_4();
            }

        } else if (y5 == 3) {

            if (y5changed) {
                y5changed = false;
            }

        }

        if (y5_old != y5) {
            log(5, y5_old, y5);
        }
    }

    public static void log(int autoNum, int oldState, int newState) {
        LCD.showNumber(autoNum * 100 + oldState * 10 + newState);
    }


    public static void main(String args[]) {
        while (true) {
            A1();
        }
    }
}
```

Приложение 2. Java-код программы робота-поставщика

```
import josx.platform.rcx.*;

public class Dispencer {
    public static final int ES_POWER = 1;
    public static final int VOLTAGE_TIME = 2000;

    public static final Sensor ts = Sensor.S1;
    public static final Motor motor = Motor.A;

    public static int candyCount = 0;

    public static int timerStart = 0;

    public static byte packet[] = new byte[10];
    public static byte sendPacket[] = new byte[2];

    public static boolean transportSignal = false;
    public static byte transportValue = 0;
```

```
public static int y0 = 0;
public static boolean y0changed = false;


public static void updateSignals() {
    while (Serial.isPacketAvailable()) {
        Serial.readPacket(packet);

        if ((packet[0] & 255) != 210) {
            transportSignal = true;
            transportValue = packet[1];
        }
    }
}


public static boolean x0_0() {
    return ts.readBooleanValue();
}

public static boolean x0_1() {
    return candyCount > 0;
}

public static boolean x0_2() {
    updateSignals();
    return transportSignal;
}

public static boolean x0_3() {
    int timePassed = ((int) System.currentTimeMillis()) - timerStart;
    return timePassed > VOLTAGE_TIME;
}

public static void z0_0() {
    LCD.showNumber(Battery.getVoltageMilliVolt());
}

public static void z0_1() {
    motor.setPower(ES_POWER);
    motor.forward();
}

public static void z0_2() {
    motor.stop();
}

public static void z0_3() {
    candyCount--;
}

public static void z0_4() {
    sendPacket[0] = (byte) 0xf7;
    sendPacket[1] = (byte) 0x00;
    Serial.sendPacket(sendPacket, 0, 2);
}

  public static void z0_5_0() {
  }

  public static void z0_5_1() {
  }
```

```
public static void z0_5_2() {
    if (transportSignal) {
        transportSignal = false;
        candyCount = transportValue;
    }
}

public static void z0_6() {
    timerStart = (int) System.currentTimeMillis();
}

public static void z0_7() {
    timerStart = 0;
}

public static void A0S4IN() {
    z0_6();
}

public static void A0S0OUT() {
    z0_0();
}

public static void A0S4OUT() {
    z0_7();
}


public static void A0() {
    int y0_old = y0;

    if (y0 == 0) {

        //-----------------------------------------------------------------
        if (y0changed) {
            y0changed = false;
        }
        //-----------------------------------------------------------------

        //-----------------------------------------------------------------
        if (true) {
            y0 = 4;
        }
        //-----------------------------------------------------------------
        if (y0 != y0_old) {
            A0S0OUT();
            y0changed = true;
        }
        //-----------------------------------------------------------------
        if (y0 == 4) {

        }
        //-----------------------------------------------------------------

    } else if (y0 == 1) {

        boolean x0_2 = x0_2();

        //-----------------------------------------------------------------
        if (y0changed) {
            y0changed = false;
        }
        //-----------------------------------------------------------------
```

```
        //-----------------------------------------------------------------
        if (x0_2) {
            y0 = 2;
        } else if (!x0_2) {
            y0 = 1;
        }
        //-----------------------------------------------------------------
        if (y0 != y0_old) {
            y0changed = true;
        }
        //-----------------------------------------------------------------

        if (y0 == 2) {
            z0_5_2();
            z0_5_0();
            z0_1();
        }

    } else if (y0 == 2) {

        boolean x0_0 = x0_0();

        //-----------------------------------------------------------------
        if (y0changed) {
            y0changed = false;
        }
        //-----------------------------------------------------------------
        //-----------------------------------------------------------------
        if (!x0_0) {
            y0 = 3;
        } else if (x0_0) {
            y0 = 2;
        }
        //-----------------------------------------------------------------
        if (y0 != y0_old) {
            y0changed = true;
        }
        //-----------------------------------------------------------------

    } else if (y0 == 3) {

        boolean x0_0 = x0_0();
        boolean x0_1 = x0_1();

        //-----------------------------------------------------------------

        if (y0changed) {
            y0changed = false;
        }
        //-----------------------------------------------------------------

        if (x0_0 && x0_1) {
            y0 = 2;
        } else if (!x0_1) {
            y0 = 1;
        } else if (!x0_0 && x0_1) {
            y0 = 3;
        }

        //-----------------------------------------------------------------
        if (y0 != y0_old) {
            y0changed = true;
```

```
            }
            //---------------------------------------------------------------

            if (y0 == 2) {
                z0_3();
            } else if (y0 == 1) {
                z0_2();
                z0_4();
                        z0_5_1();
            }

        } else if (y0 == 4) {

            boolean x0_3 = x0_3();

            //---------------------------------------------------------------

            if (y0changed) {
                A0S4IN();
                y0changed = false;
            }

            //---------------------------------------------------------------

            if (x0_3) {
                y0 = 1;
            } else if (!x0_3) {
                y0 = 4;
            }

            //---------------------------------------------------------------

            if (y0 != y0_old) {
                A0S4OUT();
                y0changed = true;
            }

            //---------------------------------------------------------------

            if (y0 == 1) {

            }
        }

        if (y0_old != y0) {
            log(4, y0_old, y0);
        }
    }

    public static void log(int autoNum, int oldState, int newState) {
        LCD.showNumber(autoNum * 100 + oldState * 10 + newState);
    }


    public static void main(String args[]) {
        while (true) {
            A0();
        }
    }
}
```

Приложение 3. Список констант транспортного робота

```
VOLTAGE_TIME = 2000
TURN_TIME = 15000
BACK_TIME = 3000

ROT_TIME = 12000
ROT_EX_TIME = 500
CORRECT_TIME = 500
ES_POWER = 4
ER_POWER = 4

RR_TIME = 4000
RB_TIME = 8000
ERR_POWER = 3

BLACK_UP = 41
BLACK_DOWN = 20
WHITE_UP = 63
WHITE_DOWN = 42
FOIL_UP = 80
FOIL_DOWN = 64
OTHER_UP = 0
OTHER_DOWN = 0
```

Приложение 4. Список констант робота-поставщика

```
VOLTAGE_TIME =2000
ES_POWER = 1
```