ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТАРНСПОРТА

Федерально государственное бюджетное образовательное учреждение
высшего образования
«Иркутский государственный университет путей и сообщения»
(ФГБОУ ВО ИрГУПС)

Факультет «Управление на транспорте и информационнные технологии»
Кафедра «Информационные системы и защита информации»

К ЗАЩИТЕ ДОПУСКАЮ

зав. кафедрой «ИСиЗИ»
д.т.н., доцент Аршинский Л.В.

ЛОГИЧЕСКОЕ УПРАВЛЕНИЕ ИМИТАЦИОННЫМ ПРОЦЕССОМ НА ОСНОВЕ СИСТЕМЫ АВТОМАТИЧЕСКОГО ДОКАЗАТЕЛЬСТВА ТЕОРЕМ

Магистерская диссертация

МД.430200.09.04.04.001-2018.ПЗ

КОНСУЛЬТАНТ

по нормоконтролю
к.т.н. доцент Матиенко Л.В.

РУКОВОДИТЕЛЬ РАБОТЫ

Звонков И.В.

ИСПОЛНИТЕЛЬ

студент группы ПИм.1-16-1
Арляпов С.В.

Иркутск 2018

Содержание

Введение

На сегодняшний день неотъемлемой частью технического оснащения практически любого здания является подъемное устройство. Это может быть эскалатор, подъемная платформа, но самым распространенным оборудованием является лифт.

Лифт – это грузоподъемное устройство, предназначенное для вертикального перемещения грузов и людей в кабине, передвигающейся по жестким направляющим. По конструктивным особенностям лифты бывают различными, но несмотря на это действуют они по одному принципу.

В устройстве любого лифта обязательно присутствуют определенные компоненты, такие как кабина (или платформа), она закрепляется на стальных тросах, перекинутых через шкив приводного механизма, передающего силу с одного места на другое. В машинном отделении в верхней части шахты лифта расположен сам приводной механизм вместе с аппаратурой управления лифтом, куда и передаются сигналы из кабины лифта.

Лифты подразделяются на категории по видам транспортируемых грузов.

Существуют лифты пассажирские, используемые в жилых, общественных зданиях. В пассажирском лифте допускается перевозка легких грузов и предметов домашнего обихода при условии, что их общая масса вместе с пассажиром не превышает грузоподъемности лифта. Такие лифты могут быть также с увеличенной платформой - для перевозки больных и инвалидов, поскольку они обязательно должны вмещать на платформу инвалидную коляску или кушетку.

Еще одной категорией являются грузовые лифты, которые также могут подразделяться по видам прилагаемой подъемной силы (к верхней части кабины, либо к нижней).

Лифты могут также классифицироваться по способу обслуживания. В этом случае различают лифты самостоятельного пользования, которыми управляет сам пассажир, и лифты, которыми управляет проводник и которые всегда сопровождают груз.

Разные лифты движутся с различной скоростью. По скорости движения каби-

ны они подразделяются на следующие типы:

– тихо-ходные (до 1,0 м/с);

– быстроходные (от 1,0 до 2,0 м/с);

– скоростные (от 2,0 до 4,0 м/с);

– высокоскоростные (свыше 4,0 м/с).

Относительно типа привода подъемного механизма лифты подразделяются на электрические (с приводом от электродвигателя пе-ременного или постоянного тока) и гидравлические (с приводом в виде подъемного гидроцилиндра или лебедки с гидродвигателем вращательного типа).

В устройстве лифта важна не только механика и принцип движения кабины в пространстве, но также и способы реагирования лифта на поступающие от пользователей запросы. По способу управления лифты различают:

– простое раздельное управление. В данном случае регистрируется и выполняется только одна команда (приказ или вызов);

– собирательное управление, при котором регистрируются все команды, а их выполнение осуществляется в соответствии с программой работы лифта. При этом могут совершаться попутные остановки по вызовам или приказам. Для лифтов жилых зданий попутные остановки по вызовам выполняются только при движении кабины вниз, а в общественных зданиях - в обоих направлениях. По приказам попутные остановки предусмотрены во всех лифтах в обоих направлениях;

– одиночное управление (управление одним лифтом);

– групповое - управление группой лифтов, расположенных в одной шахте, обслуживающих одни и те же этажи и имеющих одинаковую скорость. Разновидностью группового управления является парное управление лифтами, применяемое в жилых зданиях повышенной этажности.

В данной работе будут рассмотрены проблемы группового управления лифтами.

Целью выпускной квалификационной работы является разработка программного решения, подтверждающего принципиальную возможность и целесообразность

построения и использования системы логического вывода в интеллектуальном управлении группой лифтов.

В рамках данной цели необходимо выполнить следующие задачи:

– изучить принципы автоматического логического вывода;

– разработать математическую модель интеллектуального управления группой лифтов;

– разработать варианты программной реализации модели интеллектуального управления группой лифтов;

– провести сравнительный анализ полученный программных решений.

1 Анализ предметной области

1.1 Конструкция лифтов

1.1.1 Общая схема лифта

Конструктивные исполнения лифтовых систем могут быть различны. В устройстве любого лифта обязательно присутствуют определенные компоненты, такие как кабина (или платформа), она закрепляется на стальных тросах, перекинутых через шкив приводного механизма, передающего силу с одного места на другое. В машинном отделении в верхней части шахты лифта расположен сам приводной механизм вместе с аппаратурой управления лифтом, куда и передаются сигналы из кабины лифта.

Внутри кабины, а также на этажах, обслуживаемых лифтом, в местах остановки лифта пользователь может выбрать необходимое ему направление движения - выбрать нужный этаж внутри кабины, либо направление вверх или вниз при вызове кабины на этаже. Сигнал, последовавший после нажатия какой-либо кнопки (вызова или приказа), передается на устройство управления лифтом. Устройством отправляется ответное сообщение пользователю, подтверждающее прием команды, после чего устройство направляет кабину в нужном направлении, соответствующем запросу. Помимо приказного поста внутри кабины лифта находится кнопка аварийного вызова, с помощью которой пользователь может в аварийном случае установить связь с управляющим пунктом. Более подробно схематическое строение лифта представлено на рисунке 1.1.

На схеме отображены основные компоненты лифтового оборудования, такие как:

– кабина;
– передвижные тросы;
– противовес;
– генераторная установка;
– подъемный трос;
– направляющие рельсы.

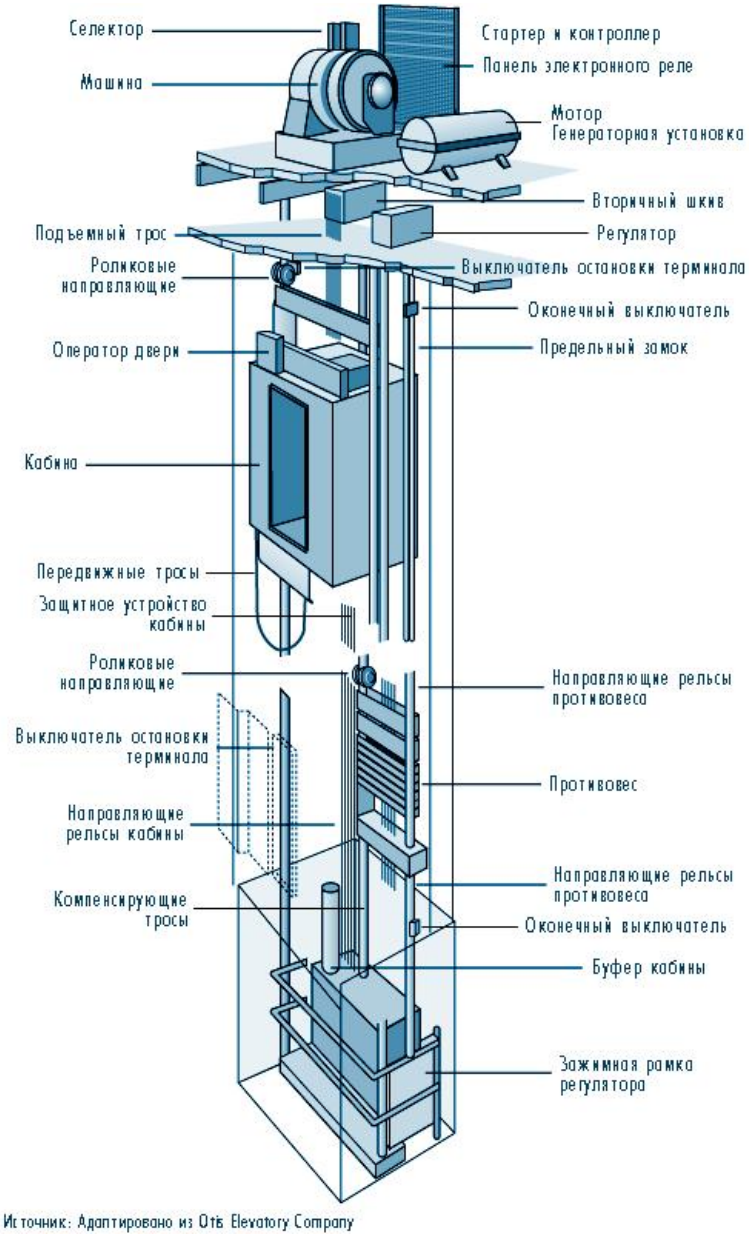

Рисунок 1.1 – Схема устройства лифтового оборудования

Кабина состоит из пола, стен, потолка, панелей управления. Она может быть панорамной, самонесущей, каркасной. В верхней части кабины установлены устройства безопасности – уловители плавного торможения устройства подвески кабины, привод дверей. Внутри кабины лифта предусмотрен приказной пост (кнопочное управление для направления лифта на необходимый этаж), электронное табло с указателями, датчики положения дверей, элементы освещения и вентиляторы. На крыше кабины находится электродвигатель, производящий открытие/закрытие дверей кабины. На боковой стене кабины находятся бесконтактные датчики – один для точной остановки и два датчика начала торможения (при движении вверх и вниз). Каж-

дый из них срабатывает при нахождении напротив металлического элемента, расположенного в шахте лифта между этажами. Количество таких элементов для каждого датчика равно количеству этажей. Вместо данных элементов могут использоваться определенные магнитные ленты, но в таком случае необходимо использовать и магнитные датчики.

Все компоненты лифтового оборудования располагаются в шахте, машинном отделении и на этажах задания. В шахте лифта присутствуют блокировочные выключатели дверей шахты с замками и осветительные устройства. На этажах находятся вызывные посты и электронное табло с указателями. Также лифт оборудован механической блокировкой на каждом этаже, за счёт которой внешняя дверь на этажах, где не кабина не останавливается, открываться не будет.

1.1.2 Особенности конструкции лифтов в высотных зданиях

При проектировании лифта в высотном здании некоторые аспекты, которые не играли особой роли в обычном многоэтажном доме, становятся более существенными. Например, в высотных зданиях применяется такой элемент, как компенсационный трос, который не требуется в зданиях небольшой этажности, поскольку масса троса не оказывает существенного влияния на кинематику системы. В зданиях высокой этажности длина троса может составлять несколько сот метров и его массой пренебречь уже не представляется возможным. При этом во время движения кабины лифта масса кабины (кабины с тросом) и масса противовеса изменяется, поскольку сокращаются или увеличиваются соответствующие участки троса. Система становится разбалансированной. Решением данной проблемы является применение компенсационных тросов, которые соединяют противовес и кабину в нижней части.

При движении кабины вверх происходит уменьшение длины участка троса, на котором подвешена кабина, когда как длина участка компенсационного троса внизу кабины увеличивается. При движении вниз происходит наоборот. В таком случае разбалансировки системы не происходит.

Еще одной особенностью лифтов в высотных зданиях является тот факт, что при скоростях движения кабин лифта, превышающих 6 м/с, роль аэродинамики ка-

бины становится более существенной. При этом выделяются два вида конструктивного исполнения лифтов с точки зрения влияния на аэродинамику. Первый вид подразумевает расположение кабин в одной шахте, и они разделены между собой балками. Второй – каждая лифтовая кабина помещается в отдельную шахту. При реализации второго вида стоит учитывать тот факт, что при движении воздуха в узких промежутках между стенками шахты и кабины возникает шум. По этой причине в высотных зданиях при скоростях движения кабин более 4 м/с рекомендовано использование нескольких кабин, разделенных балками в общей шахте. Если же конструктивные особенности здания этого сделать не позволяют, то в лифтовой шахте реализуются отводы воздуха с перетоками, которые позволяют если не исключить, то хотя бы минимизировать вредные аэродинамические воздействия.

При скоростях лифта выше 6 м/с используются специальные лифтовые кабины, которые имеют аэродинамическую форму, которая подразумевает наличие в верхней и нижней частях кабин обтекателей, позволяющих минимизировать влияние сопротивления воздуха.

1.2 Требования к системе управления лифтами

Основными требованиями, предъявляемым к системе управления лифтовым оборудованием является безопасность работы, надёжность, плавность разгона, движения и торможения, точность остановки кабины, минимизация шума во время работы и недопущение помех радиоприему и телевидению. Эти требование необходимо учитывать, как при проектировании системы управления, так и в процессе монтажа и эксплуатации.

Ключевое требование безопасность работы лифта. При его работе есть вероятность возникновения некоторых аварийных ситуаций, таких как превышение скорости перемещения кабины выше допустимой, произвольный пуск лифта, перегрузка кабины, пуск лифта при открытых дверях кабины и/или шахты, обрыв каната либо элементов подвеса. Таким образом, алгоритм, по которому работает система управления, должен предусматривать возникновение подобных ситуаций.

Наилучшим алгоритмом работы является тот, который обеспечивает общедо-

ступность пользования и комфортабельность пассажиров. Общедоступность пользования лифтом предполагает наличие достаточно простой и понятной системы управления движением из кабины и этажных площадок, не требующей специальной подготовки пассажиров всех возрастных групп. Комфортабельность условий перевозки определяется минимальным значением времени ожидания кабины пассажирами на посадочном этаже и перемещения их между этажами.

При этом алгоритм работы системы должен минимизировать расход электроэнергии и при этом не требовать значительных финансовых затрат.

Наиболее распространенный алгоритм должен предполагать:

– исключение направления на этаж вызова более одной кабины;

– выполнение вызова на определенный этаж назначается идущей в нужном направлении не полностью загруженной кабине. Если таковой нет, ближайшей свободной кабины;

– автоматическое направление первой из освободившихся кабин на этаж наибольшего спроса (обычно - первый), а остальные кабины после освобождения остаются на этажах, на которых они пришли по приказам.

Также необходимо учитывать, что режим работы главного электропривода лифта характеризуется частым включением и отключением. При этом выделяются такие этапы движения:

– разгон электродвигателя до установившейся скорости;

– движение с установившейся скоростью;

– уменьшение скорости при подходе к этажу назначения;

– торможение и остановка кабины лифта на указанном этаже с требуемой точностью.

Наряду с этим имеет место тот факт, что этап движения с установившейся скоростью может отсутствовать, если сумма путей разгона до установившейся скорости и торможения меньше расстояния между этажами отправления и назначения.

Исходя из этого можно заключить, что в зависимости от условий работы целесообразно проектировать систему, в которой реализованы различные скорости дви-

жения.

Так, например, в зависимости от назначения рекомендуется применять пассажирские лифты со следующими номинальными скоростями:

– в административных зданиях и гостиницах: до 9 этажей - от 0,7 м/с до 1 м/с; от 9 до 16 этажей - от 1 до 1,4 м/с;

– в административных зданиях от 16 этажей –1,6 – 4 м/с.

Так как система, разрабатываемая в рамках данной работы, предназначена для двадцатиэтажного здания, то примем скорость лифта равно 1,6 м/с. Исходя из таблиц основных параметров и применяемости лифтов в зависимости от вида здания, которые приведены в ГОСТ 5746-2015, примем грузоподъёмность равной 1000 кг при вместимости кабины 12 человек, подразумевая, что лифтовое оборудование будет применяться в общественных, административных зданиях или в зданиях промышленных предприятий.

Также к системе управления лифтовым оборудованием предъявляются некоторые требования по пожарной безопасности. В случае возникновения пожара все лифты должны опуститься на главный посадочный этаж и открыть двери (в качестве главного посадочного этажа определён первый). В таком состоянии лифты блокируются до тех пор, пока не будет отключена система пожарной безопасности.

1.3 Системы управления лифтами

Лифтовые механизмы подъема различаются по типам электроприводов.

Одним из типов является нерегулируемый привод, в котором используется одно- и двухскоростные двигатели переменного тока. Такой привод используется в тихоходных лифтах с достаточно низкими требованиями к точности остановки кабины.

Другим типов привода является двухскоростной асинхронный привод. В нем используют двигатель с короткозамкнутым ротором и двумя статорными обмотками большой и малой скорости.

Также существует регулируемый привод постоянного тока, обеспечивающий аналогичные условия и использующийся при формирования схемы движения кабины лифта, близкой к оптимальной, и имеющий высокую точность остановки каби-

ны.

В современных лифтовых устройствах используются два принципа управления: замкнутый и разомкнутый. При использовании разомкнутого принципа управление приводом лебедки происходит с помощью сигналов, формируемых в логической управляющей системе. В данном случае не учитываются возможные изменения параметров лифта (лебедки и кабины) в процессе работы.

При замкнутом принципе система позволяет учитывать все возможные изменения параметров и управлять приводом согласно сигналам, направленных от логической управляющей системы, более того, она позволяет учитывать результаты функционирования привода. Такая система управления дает возможность увеличить точность остановки, повысить плавность движения кабины.

Замкнутый контур контроля скорости гарантирует точное и комфортное поведение привода в каждый момент работы. Измеренная скорость электродвигателя вводится в регулятор скорости типа ПИ-регулятора. Динамическая точность регулирования скорости (время устранения системой регулирования ошибки по скорости) высока.

По видам функционального управления движением кабины лифты подразделяются на следующие типы:

– внутренняя кнопочная система, осуществляющая вызов пустой кабины на любой этаж;

– внутренняя кнопочная система, осуществляющая вызов пустой кабины на любой этаж, при этом совершая попутные остановки по вызовам при движении кабины вниз;

– внутренняя кнопочная система, осуществляющая вызов пустой кабины на любой этаж, с попутными остановками по вызову при движении кабины вниз, с использованием группового управления;

– собирательная внутренняя кнопочная система, осуществляющая вызов пустой кабины на любой этаж, с попутными остановками по вызову при движении кабины в любом направлении;

– собирательная внутренняя кнопочная система, осуществляющая вызов пустой кабины на любой этаж, с попутными остановками по вызову при движении кабины в любом направлении, а также с использованием группового управления.

Для пользовательского управления движением кабины лифта ипользуются кнопочные посты управления - вызывной и приказной пост. Внешний вид постов управления представлен на рисунке 1.2.

Рисунок 1.2 – Вызывной пост (слева) и приказной пост (справа)

Самый простой вид управления, осуществляющий вызов пустой кабины на любой этаж, применяется для пассажирских лифтов грузоподъемностью до 320 кг, которые устанавливаются в жилых домах средней этажности (9-12 этажей). Как правило такие лифты монтируются по одному в подъезде, реже - по два в одной или

рядом расположенных шахтах.

Все остальные виды управления используются для пассажирских лифтов грузоподъемностью 320-500 кг, устанавливаемых в домах повышенной и большой этажности (16 этажей и более), а также в административных зданиях. Такие лифты обычно монтируются по два и более в одной или рядом расположенных шахтах и работают в системе парного или группового управления.

1.4 Системы управления группой лифтов

На сегодняшний день существует большое количество систем управления лифтами, в числе которых есть как системы релейного типа, характерные для старых построек, так и системы, на основе микропроцессорной техники. Все системы управления можно разделить на два типа в зависимости от их структуры: централизованные и распределенные.

1.4.1 Централизованные системы группового управления лифтами

Система данного типа подразумевает наличие одного главного (центрального) контроллера, который размещается в станции управления и обеспечивает функционирование системы управления в целом. В некоторых случаях возможно применение небольших локальных плат или контроллеров, обеспечивающих какие-либо примитивные функции управления, но, в основном, их функции сводятся к формированию сигналов для отправки на центральный пост управления и локальному распределению управляющих сигналов.

Примером такой системы может являться станция управления лифтами, разработанная компанией STEP Electric Corporation. Данная станция разработана на основе технологии централизованного группового управления, при этом групповой контроллер производит определение места расположения лифтовой площадки, регистрацию и обработку команд, которые могут быть получены как от индивидуальных контроллеров лифтов, так и напрямую от устройств, расположенных в кабинах или на этажах.

Достоинствами централизованной системой управления является высокий уро-

вень планирования и контроля функционирования системы. Однако при сложной системе могут проявиться некоторые недостатки:

– возникновение, так называемого, «эффекта бутылочного горла», когда обработка информации и выдача управляющего воздействия могут быть продолжительными по времени по причине перегрузки центрального пункта управления.

– увеличение длительности «цикла управления» из-за отдаленности места принятия решений от места их исполнения.

– большое количество проводов, поскольку все сигналы необходимо направить на центральный пост управления.

По состоянию на сегодняшний день в мировой лифтовой индустрии наблюдается отказ от централизованных систем управления и перехода к децентрализованным (распределенным).

1.4.2 Распределенные системы управления лифтами

Значительное снижение цен на микропроцессоры и полупроводниковые микросхемы и приборы, и столь же существенное увеличение производительности микропроцессоров, наблюдаемые на протяжении нескольких лет, сделали актуальными и рентабельными распределенные системы и системы реального времени на базе микроконтроллеров.

Недостатки централизованного управления могут быть скомпенсированы применением распределенной системы, которая характеризуется наличием ряда иерархически, функционально, структурно связанных контроллеров.

В такой системе управление децентрализовано и добавление нового контроллера в сеть системы не ограничено при условии ее грамотной организации. При этом общая задача управления разбивается на несколько подзадач, выполнение каждой из которых производится отдельный контроллером. Решение этих задач выполняется одновременно.

При этом интеграция новых устройств как собственного производства, так и других производителей становится более реализуемой, а аппаратные решения более эффективными – добавление дополнительного функционала решается использова-

нием дополнительных пунктов контроля и управления (контроллеров).

В качестве примера данной структуры может служить система управления грузопассажирским лифтом, которая была разработана компанией Элеси.

Система выполнена как распределенная на современной элементной базе, как следствие экономия кабельной продукции и повышение надежности функционирования.

Ключевыми достоинствами систем данного типа является сокращение числа проводов, возможность оперативно адаптации системы к частным условиям и требованиям за счет соответствующего распределения программного обеспечения, повышенная помехоустойчивость системы программным способом без применения специальных средств.

Недостатками этой системы может являться то, что групповое управление лифтами может сводится лишь к оповещению контроллеров других лифтов о выполнении какой-либо операции, что объясняется отсутствием единого центра диспетчеризации. Другими словами, организация эффективного алгоритма управления в ней ставится под вопрос.

Еще одним примером распределенной системы управления лифтами могут быть системы управления ЛиРа. Данная система имеет в своем составе центральный шкаф управления для выполнения диспетчеризации и множество плат, используемых под каждую конкретную функцию, среди которых есть плата телефонной связи, плата тормоза, плата ключей и так далее.

Достоинства и недостатки данной системы вытекают из ее строения. С одной стороны, широкий выбор специализированных плат даёт возможность собрать систему любой необходимой конфигурации. С другой, большое количество плат может сделать систему сложной и неудобной для монтажа и последующего обслуживания.

1.5 Методы управления группой лифтов

В зависимости от принятой системы управления лифты могут работать индивидуально, в паре или в группе.

При индивидуальной работе действие одного лифта не влияет на действие дру-

гого. Для индивидуально работающих лифтовых установок характерны свойства:

– системы вызовов на таких установках действуют только на один лифт;

– одиночное управление, как правило, применяется для обслуживания зданий с небольшим числом этажей.

При парной или групповой работе лифты действуют зависимо друг от друга. Такая система предназначена для тех случаев, когда одиночная установка лифта недостаточна для обеспечения пассажиропотока и в здании устанавливают два лифта, обслуживающие одни и те же этажи. Системы парного управления обеспечивают такую согласованную работу двух лифтов с общей системой вызовов, при которой достигается максимальная производительность их и минимальное время ожидания.

Групповая система управления применяется при групповой установке лифтов в крупных общественных и административных зданиях, когда имеют место напряженные пассажиропотоки и парные установки не обеспечивают требуемой производительности. Групповая установка применяется с числом лифтов от трех до четырех.

В общем виде программа работы одного или группы лифтов сводится к отработке кабинами отдельных команд: вызовов, приказов и др. Программы работ в разных режимах для наиболее массовых типовых пассажирских лифтов жилых зданий приводятся в техническом описании принципа действия их электрических схем.

Система вызовов на групповых лифтах так же, как и на парных, общая для всей группы совместно работающих лифтов, то есть устанавливается только одна вызывная кнопка на этаже. Кроме того, групповые лифты оборудованы специальным автоматическим устройством организации совместной работы лифтов.

В системах группового управления предусматриваются утренний, дневной и вечерний режимы работы. Эти режимы задаются диспетчером или устанавливаются автоматически в зависимости от направленности и напряженности пассажиропотока в здании.

1.6 Алгоритмы управления лифтами

Алгоритм работы системы управления состоит из основного алгоритма, алгоритма подпрограмм, реализующих различные режимы работы системы управления (ревизии, деблокировки, управления из машинного помещения, нормальной работы, пожарной опасности), и алгоритмов дополнительных подпрограмм, реализующих типовые действия, производимые в режиме нормальной работы (движение лифта по приказу, остановка кабины на этаже).Алгоритм работы системы управления представлен на рисунке 1.3.

Рисунок 1.3 – Алгоритм работы системы управления

Алгоритм начинается с включения лифта и работу (блок 1), после чего начинается постоянный контроль цепи безопасности (2). Если цепь разомкнута, происхо-

дит аварийная остановка лифта (3). В зависимости от причины аварийной остановки применяется режим деблокировки (5), если кабина лифта установилась на ловители или конечные выключатели, либо производится определение и устранение другого рода сбоя в системе (6). Блоки 7...9 определяют необходимость включения того или иного режима работы лифта, блоки 10...12 реализуют соответствующие подпрограммы. Программа продолжает свою работу до тех пор, пока не будет выполнен принудительный останов лифта. Схема алгоритма подпрограммы, реализующей режим нормальной работы, приведена на рисунке 1.4.

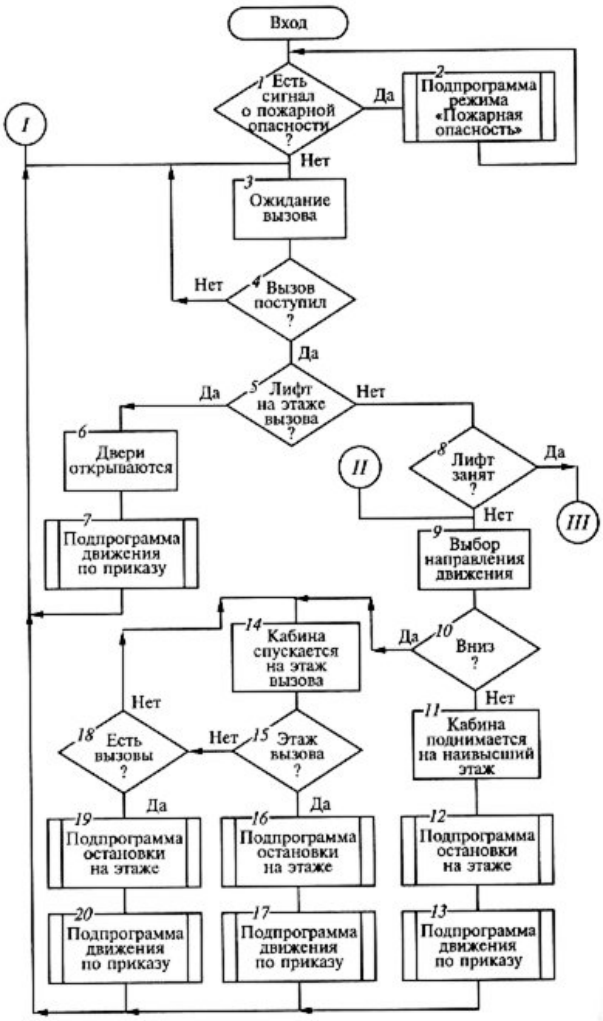

Рисунок 1.4 – Алгоритм подпрограммы

В этом режиме производятся контроль пожарной безопасности (2), регистрация и выполнение всех вызовов и приказов, контроль загруженности кабины. Этот алгоритм составлен с учетом работы системы с собирательным управлением вниз, т.е. выполняются попутные вызовы при движении кабины вниз (если загрузка ме-

нее 90 % от номинальной), Таким образом, в подпрограмме реализуются ожидание и регистрация вызова (3, 4), проверка нахождения кабины лифта на этаже вызова (5). В зависимости от этого осуществляется открытие дверей кабины с последующей работой лифта по приказу (6, 7) или проверяется условие занятости кабины (8). Если кабина свободна, то блоки 9-20 осуществляют выбор направления движения кабины и в зависимости от этого после получения приказа выполняются попутные вызовы при движении вниз (если они зарегистрированы) (14-20)или движение кабины на наивысший из этажей, с которых поступили вызовы, а затем после получения приказа собирательное управление для движения вниз.

Если при регистрации вызова кабина занята, вызов выполняется при попутном следовании кабины при условии, что она загружена менее чем на 90 % номинальной загрузки. В противном случае, представленном на рисунке 1.5, ожидают, пока кабина не освободится или не проследует в попутном направлении, загруженная менее чем на 90 % (21-29).

Рисунок 1.5 – Алгоритм при загрузке кабины более чем на 90 %

Система управления лифтом обеспечивает выполнение требований пассажиров – приказы из кабины или запросы с этажей, при этом решая спектр задач, которые связаны с определением направления движения в зависимости от взаимного расположения этажа, на котором находится кабина лифта, и требуемого этажа, с остановкой кабины на указанном этаже, с необходимостью обеспечения безопасного для пассажиров работы лифта, а также, связанных с различными режимами работы лифта.

Первые лифты не были рассчитаны на обработку одновременно нескольких запросов. Выполнение запросов производилось только последовательно, каждый последующий запрос мог быть произведен только после выполнения предыдущего, даже если вызовы являлись попутными. Такой вид управления подъёмником называется последовательным. Эта схема управления обеспечивает наиболее простую реализацию схемы управления, но пропускная способность лифта при этом невелика. Несмотря на это, в некоторых случаях её реализация производится и в настоящее время. Например, такая схема управления применяется в грузовых и больничных лифтах, а также в пассажирских лифтах для зданий малой и средней этажности. Данная схема управления подразумевает, что все запросы пользователей фиксируются и исполняются последовательно. В случае поступления одновременных запросов команды, которые поступают из кабины лифта являются приоритетными, то есть лифт в первую очередь доставляет пассажира на требуемый этаж и только после этого перемещается на этаж, с которого был произведен следующий вызов (запрос).

Следующим этапом развития в данной области стала собирательная схема управления. Она получила обширное распространение в жилых зданиях, в которых применяется односторонняя собирательная система при движении вниз. Лифт, производящий движение вниз, опускаясь на первый этаж, делает остановки в соответствии с запросами от пассажиров с этажей, совершая, таким образом, сбор пассажиров. При движении в обратном направлении (снизу-вверх) обрабатываются только запросы из кабины, а запросы с этажей игнорируются. Это связано с тем, что необходимость в перемещении жителей домов с одного этаж на другой возникает очень редко. На ап-

парате вызова лифта с такой схемой управления располагается только одна кнопка.

Несколько позже в офисных зданиях и гостиницах стала применяться двусторонняя собирательная система, которая обрабатывает запросы с этажей при движении в обоих направлениях. Вызывной аппарат лифтов, работающих по данной схеме управления имеет две кнопки – вверх и вниз. При вызове производится не только указание необходимого этажа, но и требуемое направления движения. Данное усложнение схемы управления обосновывается увеличением пропускной способности лифта. Сегодня такая схема управления очень распространена в различных типах зданий. В зданиях с большим количеством этажей и интенсивными пассажиропотоками устанавливают несколько лифтов (группу лифтов). При этом возникает потребность согласования работы лифтов в группе по вызовам, задачами которого являются повышение производительности лифтов, уменьшение времени ожидания кабины пассажирами, сокращение (или полное отсутствие) количества холостых пробегов и связанное с этим уменьшение износа лифтов и расхода энергии. Данные задачи решаются системами группового управления лифтами с применением диспетчеризации.

Стоит также отметить, что групповое размещение даёт возможность для повышения качества лифтового обслуживания при существенной экономии капитальных затрат за счет использования общего машинного помещения. Также существенно снижаются затраты на проведение технического обслуживания. На протяжении длительного периода времени без собирательные, односторонние и полные собирательные схемы управления применялись в большинстве зданий, какие-либо принципиально новые решения в системе управления не производились. В это время совершенствование схем управления двигалось по пути, как уже говорилось, минимизации времени ожидания кабины лифта, что производилось за счет оптимизации схем работы лифтовых групп. Известно, что существовали решения на базе нейросетевых технологий. Данные решения приносили определенный эффект.

Один из прогрессивных алгоритмов управления группой лифтов был разработан в американской компании Otis Elevator, подразумевающий, что каждый пасса-

жир перед посадкой указывает необходимый для него этаж. В соответствии с этим система сообщает в какой лифт садиться и через какой промежуток времени пребудет кабина. При этом на каждой посадочной площадке устанавливается приказная панель, аналогичная той, которая расположена в кабине лифта. При оптимизации движении лифтов в качестве целевой функции принимается количество остановок, которые делает кабина лифта и данную величину необходимо минимизировать. Таким образом, чем меньше кабина де- лает остановок, тем быстрее она возвращается на главный посадочный этаж, при этом за счёт меньшего количества попутных остановок происходит существенная экономия электроэнергии и сокращение времени кругового рейса кабины лифта.

Посадка пассажиров близких этажей производится в один лифт с целью минимизации времени поездки и уменьшения количества остановок.

Достаточно большое распространение в системах управления группой лифтов получил круговой алгоритм управления. Его преимуществами является простота реализации, равномерное распределение нагрузки между лифтами в группе, а также обеспечение выполнения требований пассажиров на приемлемом уровне при сравнительно не интенсивном пассажиропотоке. Основной целью круговой системы при диспетчерском управлении группой лифтов является достижение равной загрузки каждого лифта. Вызовы распределяются по мере их поступления последовательным образом по отдельным лифтам. Вызов 0 назначается к обслуживанию кабиной 0, вызов 1 – кабиной 1, вызов L–кабиной 0, вызов L+1 – кабиной 1 и так далее.

Частным случаем кругового алгоритма представляет собой диспетчерское управление при максимальном потоке пассажиров вверх, отличием которого является использовании особой стратегии движения. Данная стратегия заключается в том, что, если групповой контроллер обнаруживает простаивающий лифт, он производит инициализацию вызова на первый этаж для уменьшения времени ожидания будущих пассажиров.

Алгоритм трех переходов, представленный Ронгом и Хаконеном в 2003 году, используется для определения последовательности обслуживания вызовов с этажей.

Алгоритм разбивает все вызовы на 3 категории:

– вызовы, которые лифт может обслужить без изменения направления движения;

– вызовы, которые лифт может обслужить после того, как один раз изменит направление движения;

– вызовы, которые лифт может обслужить после того, как дважды изменит направление движения.

Приоритет выполнения вызовов убывают от первой группы к третьей.

В некоторых случаях используют идею зонирования высотных зданий по вертикали, которая заключается в разделении здания на несколько прилегающих друг к другу зон и каждый из лифтов обрабатывает вызовы с этажей только зоны, назначенной для обслуживания данным лифтом.

Как правило, разделение здания на зоны с целью оптимизации движения лифтов производится в высотных административных зданиях, но данная концепция может также применяться и для обычных жилых здания. Оптимизация данной стратегии проводится при помощи правильного определения размеров выделяемых зон, при этом могут быть учтены такие параметры как численность жильцов на этажах, а также некоторые приоритеты обслуживания каких-либо этажей, которая определяется на основе набора условий, определяемых отдельно в каждом конкретном случае.

Также применение идеи зонирования позволяет использовать технические этажи для размещения машинных помещений для лифтов, что сокращает потери полезной площади здания. Размещение оборудования на технических этажах позволяет уменьшить затраты на решение проблемы защиты от шума и вибраций, хотя на текущий момент данная потребность не является актуальной, так как в настоящее время производители лифтового оборудования выпускают изделия, отличающееся достаточно низким уровнем шума.

В процессе развития алгоритмов управления лифтов в составе группы кроме детерминированных классических стратегий появились динамические стратегии автоматизированного управления лифтами. В рамках данных стратегий существу-

ют подходы, которые позволяют произвести переход от жестких схем управления к гибким, способным подстраиваться к изменениям условий в процессе функционирования (изменение интенсивности пассажиропотока или выход из строя лифтов в составе лифтовой группы).

Одной из таких является стратегия на основе поиска, сущность которой заключается в нахождении оптимального решения при каждом новом возникающем событии (например, вызове с этажа), при этом формируется определённая реакция на данное событие.

Алгоритм поиска производит решение задачи назначения, которая заключается в распределении зафиксированных вызовов между лифтами в группе с учётом команд, которые произвелись из кабины лифта. Если количество свободных лифтов больше числа произведённых вызовов, то алгоритм распределит их в соответствии с определенными критериями. В случае, если количество вызовов превышает количество свободных лифтов, то из множества заявок будут выделены те, обслуживание которых обеспечит минимум целевой функции, например, среднее время ожидания.

Недостатком данной стратегии является статичность в интервале формирования решений, что означает тот факт, что новые вызовы, которые поступили после назначения не имеют возможности изменить решение системы управления. Для нивелирования данного недостатка применяется переопределение управляющего воздействия, которое подразумевает корректировку решения согласно с новой текущей ситуации. Стоит отметить, что данное решение существенно повышает вычислительную сложность алгоритма. По этой причине данный метод целесообразно применять при достаточно высокой плотности пассажиропотока и когда другие подходы не дают существенного повышения эффективности.

Большое количество алгоритмов управления, которые реализуются на базе контроллеров, относятся к стратегии, описываемой набором правил в терминах «IF-THEN» логики. Такие правила составляются на основе опыта, полученного в процессе функционирования системы или сформулированных по результатам соответствующих научных исследований. Данный решение, как правило, используется в

области искусственного интеллекта, но может быть применено также в системах управления лифтовым оборудованием, оснащенной множеством датчиков.

Использование нечеткой логики может существенность расширить возможности методов, основанных на определении правил, и повысить уровень вариативности системы управления, увеличивая число степеней свободы путем повышения уровня детализации условия в сравнении со стандартной бинарной логикой.

Наряду с этим имеется возможность решать задачи управления при помощи применения генетических алгоритмов, которые подразумевают имитацию процесса эволюции – гены родителя наследуются потомками.

В системах с применением искусственного интеллекта генетические алгоритмы рассматриваются как совокупность отдельных шагов. Для имитации процесса эволюции требуется создать множество правил, на основе которых формируются промежуточные решения.

Из этого множества может быть сформирован алгоритм управления, при этом начальный родительский алгоритм-ген может быть сформирован случайным образом. Сам же процесс эволюции в этом случае представляет собой «слияние» генов получение в итоге гена-потомка – определенного решения. Перед «слиянием» производится определение перспективности смешиваемых алгоритмов путем проведения экспериментов, направленных на определение эффективности алгоритмов и обеспечения взвешенного выбора правил. Таким образом, алгоритмы с каждой итерацией будут совершенствоваться, приближаясь к некоторому идеалу и после окончания итераций будет получен некоторый усовершенствованный алгоритм управления.

Один из недостатков генетических алгоритмов заключается в их высокой трудоемкости, обусловленная итерационным характером реализации данных алгоритмов. Использование для вычисления параллельных структур, таких как, например, программируемых логических интегральных схем (ПЛИС), снижает влияние данного недостатка. Но более критичным недостатком является тот факт, что эффективность алгоритма определяется исходным набором правил, если в числе которого

нет некоторого правила, то данное правило никогда не появится в порожденных алгоритмах.

Стоить сделать замечание, что при применении динамических стратегий управления их эффективность во многом будет зависеть от числа «степеней свободы», которыми обладает система, например, количество лифтов в составе лифтовой группы.

2 Постановка задачи

Целью данного исследования является разработка программного решения, подтверждающего принципиальную возможность и целесообразность построения и использования системы логического вывода в интеллектуальном управлении группой лифтов.

2.1 Автоматическое доказательство теорем

Одним из подходов к интеллектуализации систем управления является разработка алгоритмов обработки информации, основанных на моделировании процесса рассуждений. Наиболее формализованные подходы базируются на автоматическом построении логического вывода в некоторой системе формализованных знаний (логического описания предметной области). Программные системы для поиска логического вывода называют системами автоматического доказательства теорем (АДТ), поскольку утверждения, для которых существует логический вывод, являются теоремами (в заданном исчислении).

Для оценки систем АДТ используются следующие формальные критерии эффективности поиска логического вывода:

– время решения задачи. Какое время затрачено системой АДТ для поиска логического вывода, либо установления факта невыводимости формулы;

– количество шагов вывода. Количество шагов, из которых состоит найденный логический вывод (количество формул в цепочке);

– расход памяти. Сколько памяти компьютера затрачено системой АДТ в процессе решения задачи;

– ширина класса решаемых задач. Объединение класса новых задач решаемых системой и класса задач, для которых предложен новый способ решения.

Очевидно, что повышение эффективности поиска логического вывода заключается в уменьшении численных характеристик критериев 1-3, и увеличении 4.

2.2 Позитивно-образованные формулы

В качестве теоретического базиса разработанной системы АДТ выбрано исчисление позитивно-образованных формул (ПО-формул), созданное и исследованное под руководством академика С. Н. Васильева, канд. физ.-мат. наук А. К. Жерлова, на основе языка типово-кванторных формул, изначально использовавшегося для формализации свойств динамических систем и синтеза теорем в методе векторных функций Ляпунова. Логическое исчисление состоит из трех компонент: язык, аксиомы и правила вывода. Далее представлены аксиомы и правила вывода. Исчисление ПО-формул имеет единственную аксиому и единственное правило вывода. Как и в методе резолюций, для доказательства формулы мы будем пытаться опровергнуть ее отрицание, поэтому аксиома исчисления ПО-формул — тождественно ложная ПО-формула вида (согласно принятым ранее сокращениям). Для того, чтобы алгоритмы адекватно использовали возможности ПО-формализма необходимо выявить его совместимые полезные свойства, а также свойства, несовместимые с адаптируемыми алгоритмами. Существуют некоторые особенности формализма ПО-формул, которые необходимо разрешить для эффективной реализации системы АДТ. Анализ исчисления ПО-формул показал три основные проблемы (с указанием влияния на критерии эффективности, изложенные во введении):

– поиск ответов на вопросы с неограниченными переменны- ми. Данная процедура требует выбора подставляемого терма для переменной вопроса из эрбранова универсума, который, в общем случае, т. е. при наличии функциональных символов, является счетным (бесконечным) множеством. Изначально неизвестно, какой именно терм необходимо выбрать, причем формально любая подстановка является корректной, хотя и не обязательно приводящей в итоге к опровержению формулы. В данном случае, решение проблемы направлено на повышение эффективности согласно критериям 1, 2;

– язык ПО-формул в [4] характеризуется как «достаточно однородный, но в то же время хорошо структурированный», а соответствующее исчисление «хорошо усваивает эвристики», т. е. базовая стратегия исчисления достаточно легко настраивается

под конкретную задачу. Стоит заметить, что представление ПО-формул использует большее разнообразие синтаксических структур, чем, например, представление дизъюнктов, которыми оперирует метод резолюций, в силу использования разнородных сущностей в структуре формулы: база, вопросы, консеквенты вопросов, два типа кванторов. То есть требуются применение специальных методов доступа к данным неоднородным частям ПО-формулы, а также разработка специальных структур данных представления ПО-формул. Решение этой проблемы направлено на повышение эффективности согласно критериям 1-3;

– несмотря на то, что изначально представление формулы языка предикатов в языке ПО-формул более компактно чем КНФ, применение правила вывода в процессе построения логического вывода при наличии дизъюнктивного ветвления, в общем случае, приводит к бóльшему усложнению структуры формулы, чем при применении правила резолюции. Таким образом, через некоторое количество шагов вывода размер ПО-формулы может оказаться в разы больше чем размер соответствующей КНФ. Другим фактором, присущим для всех методов и систем АДТ является неограниченная планка сложности решаемых задач. Некоторые формализации изначально являются весьма объемными. Отсюда вытекает потребность в обеспечении экономного использования оперативной памяти, а также обеспечение функционирования системы в режиме полного заполнения доступной памяти. Решение данной проблемы направлено на повышение эффективности согласно критерию 3;

– для решения прикладных задач необходимо обеспечить настраиваемость системы АДТ для учета особенностей конкретной задачи.

2.3 Дерево состояний вывода

Одним из основополагающих средств реализации поиска логического вывода в разработанной системе АДТ является дерево состояний вывода, которое строится, в основном, при помощи структур данных, базирующихся на чанках (рисунок 2.1).

В структурах дерева состояний вывода, с одной стороны, хранится вся совокупность шагов вывода, по которой можно распознать какие действия были произведены на каком шаге. С другой стороны, дерево состояний вывода представляет те-

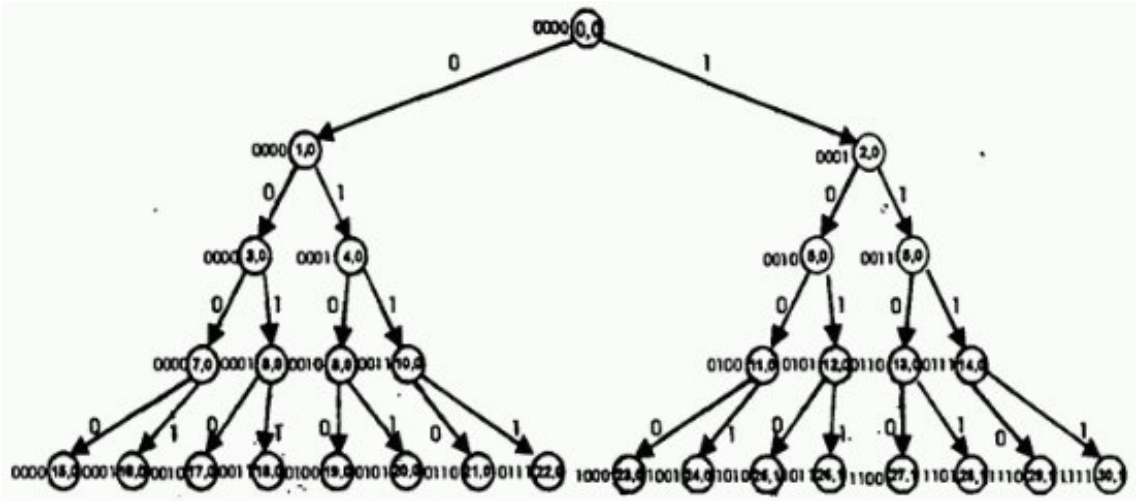

Рисунок 2.1 – Дерево состояний

кущую опровергаемую формулу. Основная задача дерева состояний вывода заключается в том, чтобы строго зафиксировать все события, произошедшие на каждом шаге логического вывода. Примером фиксируемого события является факт применения подстановки в базовой подформуле к некоторому вопросу. Такая фиксация событий позволяет:

– использовать больше информации о выполненных действиях, тем самым анализировать процесс логический вывод, а значит эффективно (в смысле большего разнообразия вариантов управления) внедрять эвристики в базовую стратегию поиска логического вывода;

– производить поиск логического вывода с возвратом (backtracking) в процессе построения логического вывода;

– реализовать стратегию разделения данных (data sharing) для случая расщепления базовых подформул после ответа на вопрос с дизъюнктивным ветвлением;

– производить эффективное (в смысле удобства реализации системы АДТ и производительности) управление оперативной памятью.

2.4 Системные предикаты и вычисляемые термы

Разработка алгоритмов программной системы включает средства автоматизации анализа количественных и структурных характеристик процесса поиска логического вывода, а также средств управления логическим выводом. В результате ре-

ализации создана программная система АДТ и среда поддержки разработки систем АДТ, ориентированные как на практические задачи, так и на абстрактные задачи, традиционно считающиеся математическими.

В логических языках программирования, например, Прологе, введены так называемые системные предикаты (встроенные предикаты, built-in predicates), особенность которых заключается в том, что они в процессе построения ЛВ или выполняют некоторое побочное действие, например, вывод на экран, чтение файла и др., или их истинность вычисляется из значений параметров. Например, var(X) в Прологе определяет, является ли терм X переменной. Будем называть системным предикатом такой предикат, атомы которого входят в конъюнкты дерева ПО-формулы, но не участвуют непосредственно в ЛВ. Системные предикаты не имеют прямого отношения к формализации задачи, но они обладают одним из следующих свойств: они влияют на процесс ЛВ в результате реализации некоторых побочных действий; их истинностные значения вычисляются; они выводят некоторую системную информацию.

В системе реализованы следующие системные предикаты:

Next(L) — переход к вопросу, помеченному идентификатором L. Предикат помещается в конъюнкт корневой вершины консеквента вопроса. Если на данный вопрос будет произведен ответ, то следующим вопросом, для которого будет производится выбор ответа, будет вопрос (множество вопросов), помеченный идентификатором L. Таким образом, при помощи данного предиката задаются варианты порядка ответа на вопрос.

OffQuestion(L)/OnQuestion(L) — отключение/включение вопроса с идентификатором L. Отключенный вопрос объявляется неактивным, т. е. не принимает участие в логическом выводе, при этом в любой момент может быть заново включен.

RemQuestion(L) удаляет вопрос с идентификатором L. RemFact(L) и RemPatternFact(L). Удаление факта помеченного идентификатором L, а также удаление всех основных примеров L, если L — терм.

OffFact(L)/OnFact(L) — отключение и включение факта с именем L. Поведение

подобно включению и отключению вопросов.

Write(T) — печать терма T на стандартный вывод.

Save(L) — пометить состояние процесса поиска ЛВ в данной базовой подформуле идентификатором L.

Rollback(L) — откатить (backtrack) состояние вывода в базовой подформуле до состояния L с утерей более поздних по отношению к L маркировок.

Commit(L) — фиксировать состояние базы L как неоткатываемое, при этом фиксируются все состояния, помеченные ранее, чем L .

2.5 Упрощения при реализации

Поскольку в исследовании выявляется возможность и целесообразность построения и использования систем логического вывода, то при поиске решения допускаются все упрощения, которые неизбежны при построении моделей и при переходе к реальному приложению.

Первым основным упрощением является дискретность времени с заданной величиной интервала между соседними моментами времени (тактами), содержащее начальный момент и бесконечно продолжимое вправо. Высота этажей считается одинаковой, и скорость движения лифта с одного этажа на другой полагаем равной одному такту. Длительность остановки кабин для входа-выхода пассажиров равняется одному такту. Так же не рассматриваются случаи переполнения кабин.

Считается, что любой пассажир придерживается следующим правилам:

– для вызова лифта он нажимает на этаже кнопку вызова и ждёт кабину, без ложных и ошибочных вызовов;

– войдя в кабину, пассажир задаёт ей команду, для чего он нажимает кнопку нужного этажа, который вносится в маршрут данной кабины, без ложных и ошибочных команд.

Простейшим алгоритмом принятия решения является поиск ближайшей кабины к месту вызова. Однако, термин «ближайшая» требует уточнения и рассмотрения примера.

Допустим, есть система из k = 2 кабин, способных перемещаться по n = 5 этажам на рисунке 2.2. Пусть кабины находятся на 1-м и 2-м этажах, первая пуста и находится в покои, а второй предстоят остановки на 3-м и 4-м этажах. Поступает вызов с пятого этажа, и первая кабина получается ближайшей, так как её требуется 4 такта, а второй кабине требуется 5 тактов. Получается дистанция – это количество тактов, которое необходимо кабине, чтобы добраться до этажа, выполняя уже сформированный маршрут.

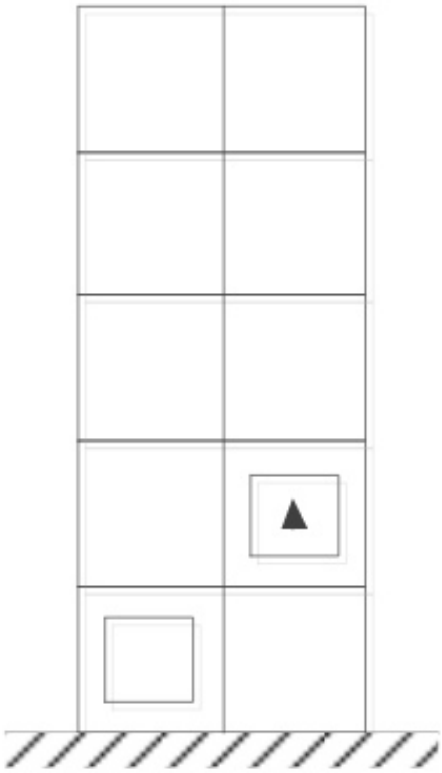

Рисунок 2.2 – Интеллектуальное управление группой лифтов

Есть и другой подход, который основывается на исключении худших альтернатив на основе логического вывода. И если после сокращения допустимых альтернатив их останется несколько, то выбор может быть случайным или основываться на каких-либо критериях. В этом и предыдущих подходах одним из основных критериев является средняя длительность ожиданий.

2.5.1 Логическая модель

Выстраивая логическую модель, получаем тройку (F, S, V), где F – это позитивно образованная формула (ПОФ), описывающая состояние лифта и принципы, по которым функционирует лифт, S – порядок ответов на запросы при логическом выводе, V – внешние воздействие, в данном случае имитация пассажиропотока. Так же следует отметить, что одними из основных будут предикаты связанные со временем: T(t) – момент времени t и N(t, t') – следующий за t момент времени t'.

Основными объектами в данной модели являются кабина Cab и человек Man. В момент времени t кабина имеет вид Cab(i, e, S, t), где i – идентификатор кабины, e – этаж, а S - маршрут кабины, список этажей. Человек имеет вид Man(e, d, τ, t), где e – этаж, d – целевой этаж, который добавляется в маршрут S в момент входа человека в кабину и $d \neq e$, τ – длительность ожидания человеком кабины. Дистанцией же будет Dist(e, S, i, t, α), где α – это дистанция от кабины i на этаже e с маршрутом S, где произошёл вызов. И связь i кабины с вызовом с e этажа Conn(i, e).

В каждый момент времени t_0 принятия решения формула F имеет вид

$$\exists A(t_0) \begin{cases} \forall T(t) \exists T(t'), N(t, t'), \\ \Phi \\ \Psi \end{cases}$$

$A(t_0)$ – коньюнкт, описывающий состояние системы в момент времени t_0. Если $A(t_0)$ содержит Man, то появление человека необходимо связать с вызовом определённой кабины. И группа формул Φпорождает все варианты связи и имитирует движение кабин совместно с формулой времени $\forall T(t) \exists T(t'), N(t, t')$ для некоторого количества тактов. А за счёт формул Ψпроисходит фильтрация некоторых вариантов.

Оставшиеся варианты оцениваются оцениваются и выбирается один из самых наилучших.

2.5.2 Группа формул Ψ

$\forall Man(e, d, , t), N(t, t')\exists Dist(e, S_1, 1, t, \alpha_1), ...,$

$$Dist(e, S_k, k, t, \alpha_k) \begin{cases} \exists Conn(1, e) \\ ... \\ \exists Conn(k, e), \\ \exists Man(e, d, +1, t') \end{cases}$$

– формула вычисляющая дистанцию до каждой кабины при новом появлении человека

$$\forall Cab(i, e, S, t), Conn(i, e'), N(t, t') \begin{cases} \forall e' < e\exists Cab(i, e-1, S/e', t') \\ \forall e < e'\exists Cab(i, e+1, S/e', t') \end{cases}$$

$$\forall Cab(i, e, S, t), S = S(e', S_1), N(t, t') \begin{cases} \forall e = e'\exists Cab(i, e-1, S_1, t') \\ \forall e' < e\exists Cab(i, e-1, S, t') \\ \forall e < e'\exists Cab(i, e+1, S, t') \end{cases}$$

$\forall Cab(i, e, null, t), N(t, t')\exists Cab(i, e, null, t')$

– формулы реализующие движение, где S/e является операцией вставки этажа в маршрут.

2.5.3 Группа формул Φ

Формулы из группы Фреализуют дополнительные ограничения, которые следует учитывать при построении логического вывода. В дальнейших работах они использоваться не будут. Но упомянуть о них крайне необходимо, так как они дают возможность данной модели быть более гибкой к различным ситуациям. А также их добавление в разрабатываемую систему не будет сложной задачей.

Например, вот формула, которая запрещает откладывать связывание вызова лифта человеком белее, чем на 4 такта:

$\forall Man(e, d, 4, t)\exists False$

Это правили необходимо, в том случае если в модели будет возможна отсрочка принятия решения на вызов лифта.

Таким образом, определяется функция F. Её реализация будет выполнена на языке Prolog. Перевод из приведённых выше формул возможен по примеру из теоретической части.

3 Моделирование и анализ

3.1 Используемые технологии

Логическое исчисление состоит из трёх компонент: язык, аксиомы и правила вывода.

Позитивно–образованные формулы (ПОФ)

Пусть, X — множество переменных, и A — конъюнкт.

– $\exists_X A$ и $\forall_X A$ есть $\exists$–ПОФ и $\forall$–ПОФ соответственно;

– если $F = \{F_1, \ldots, F_n\}$ есть $\forall$–ПОФы, тогда $\exists_X A \colon F$ есть $\exists$–ПОФ;

– если $F = \{F_1, \ldots, F_n\}$ есть $\exists$–ПОФы, тогда $\forall_X A \colon F$ есть $\forall$–ПОФ;

– любая $\exists$–ПОФ или $\forall$–ПОФ есть ПОФ.

Данный вид формул называется позитивно-образованным, поскольку для их записи используются только позитивные типовые кванторы. Иными словами, формула не содержит оператора отрицания.

Любая формула языка предикатов первого порядка может быть представлена как позитивно–образованная формула. Таким образом ПО–формула есть особый вид записи классических формул языка предикатов, подобно КНФ, ДНФ и др.

ПО–формула начинающаяся с $\forall\emptyset$ называется ПО–формулой в *каноническом виде*.

Очевидно, что любая ПОФ может быть приведена к каноническому виду. Действительно, допустим F — это $\exists$–ПОФ, тогда выражние вида $\forall\emptyset \colon F$ по определению является $\forall$–ПОФ, и соответствует записи $true \rightarrow F$ тождественной F. Если F — это неканоническая $\forall$–ПОФ, тогда $\forall\emptyset \colon \{\exists\emptyset \colon F\}$ соответствует записи $true \rightarrow \{true \& F\}$, тождественной F. Типовые кванторы $\forall\emptyset$ и $\exists\emptyset$ называются *фиктивными*, поскольку не влияют на истинность формулы и не связывают никаких переменных, а только лишь служат конструкциями сохраняющими корректную запись ПОФ.

Для удобства читаемости формул, будем представлять их в древовидной форме

следующим образом:

$$Q_X A\colon \{F_1, \ldots, F_n\} \equiv Q_X A \begin{cases} F_1 \\ \ldots \\ F_n \end{cases},$$

где Q некоторый квантор, и пользоваться соответствующей терминологией: узел, корень, лист, ветвь и т.д. Учитывая что квантору $\forall$ соответствует дизъюнкция формул $\{F_1, \ldots, F_n\}$, а квантору $\exists$ соответствует конъюнкция, то будем говорить что $\forall$ узлу соответствует *дизъюнктивное ветвление*, а $\exists$ узлу *конъюнктивное*.

Некоторые части канонической ПО–формулы имееют специальные названия:

– корневой узел имеет вид $\forall\emptyset$ и назвается *корень ПО–формулы*;

– дочерние узлы корня ПО–формулы имеют вид $\exists_X A$ и называются *базами ПО–формулы*, конъюнкт A называется *базой фактов*, а вся подформула начинающаяся с базового узла называется *базовой подформулой*;

– дочерние узлы баз имею вид $\forall_Y B$ и называются *вопросами* к родительской базе. Если вопрос является листовым узлом $\forall_Y B \equiv \forall_Y B\colon false$, то он называется *целевым вопросом*.

– поддеревья вопросов называются *консеквентами* или *следствиями*. Следствием целевого вопроса является $false$.

На примере аналогии с языком Пролог можно более доступно рассмотреть ПО-формулы.

– база фактов ПОФ, являющаяся конъюнкцией атомов, соответствует множеству фактов пролога.

– правила пролога имеют форму `P0:- P1,..., Pn`. В данном случае, `P1,...,Pn` соответствует конъюнкту вопроса, а `P0` его консеквенту (следствию). Отметим, что язык ПО–формул несколько богаче чем язык пролога, поскольку допускает использование в качестве консеквента дизъюнкцию произвольных формул, пролог же допускает лишь одну атомарную формулу.

– целевой вопрос ПО–формулы вида $\forall_Y B$ соответствует запросу пролога.

Рассмотрим следующую программу на языке Пролог и запрос к ней:

```
in(a,b).
in(b,c).
it(X,Y):-in(X,Z),in(Z,Y).
?- it(a,X).
```

Соответствующая данной программе ПО–формула:

$$\forall\emptyset\colon \exists in(a,b), in(b,c)\colon \begin{cases} \forall_{x,y,z} in(x,z), in(z,y)\colon \exists it(x,y) \\ \forall_x it(a,x) \end{cases}.$$

Исходя из вышеописанного и учитывая удобство адаптации языка к ПО-формулам для реализации вычисления позитивно образованных формул подошёл декларативный язык программирования общего назначения Prolog. При помощи этого языка,можно реализовать рассмотрение множества вариантов обхода дерева формулы.

3.1.1 SWI-Prolog

Prolog (Programming in logic) – один из старейших и все еще один из наиболее популярных языков логического программирования, хотя он значительно менее популярен, чем основные императивные языки. Это язык высокого уровня, не алгоритмический, предназначенный для написания программ с использованием концепций и методов логического программирования. В основе Пролога лежит раздел математической логики, называемый исчисление предикатов. Его базис составляет процедура доказательства теорем методом резолюции для хорновских дизъюнктов. Пролог включает механизм вывода, который основан на сопоставлении образцов, с помощью подбора ответов на запросы он извлекает хранящуюся (известную) информацию.

Области применения языка Prolog и декларативных языков в целом

- реализация систем искусственного интеллекта
- создание экспертных систем и оболочек экспертных систем
- разработка систем помощи принятия решений
- разработка систем обработки естественного языка
- построение планов действий роботов и т.п.

Особенности языка

– описание проблемы и правил ее решения

– нахождение всех возможных решений с помощью механизма поиска с возвратом (backtracking)

– легкий синтаксис

Prolog был создан под влиянием более раннего языка Planner и позаимствовал из него следующие идеи:

– обратный логический вывод (вызов процедур по шаблону, исходя из целей);

– построение структура управляющей логики в виде вычислений с откатами;

– принцип “отрицание как неудача”;

– использование разных имен для разных сущностей.

Главной парадигмой, реализованной в языке Prolog, является логическое программирование. Как и для большинства старых языков, более поздние реализации, например, Visual Prolog, добавляют в язык более поздние парадигмы, например, объектно-ориентированное или управляемое событиями программирование, иногда даже с элементами императивного стиля.

Prolog использует один тип данных, терм, который бывает нескольких типов:

– атом — имя без особого смысла, используемое для построения составных термов;

– числа и строки такие же, как и в других языках;

– переменная обозначается именем, начинающимся с прописной буквы, и используется как символ-заполнитель для любого другого терма;

– составной терм состоит из атома-функтора, за которым следует несколько аргументов, каждый из которых в свою очередь является атомом.

Программы, написанные на чистом Prolog, описывают отношения между обрабатываемыми сущностями при помощи клауз Хорна. Клауза — это формула вида Голова :- Тело., которая читается как “чтобы доказать/решить Голову, следует доказать/решить Тело”. Тело клаузы состоит из нескольких предикатов (целей клаузы), скомбинированных с помощью конъюнкции и дизъюнкции. Клаузы с пустым телом

называются фактами и эквивалентны клаузам вида Голова :- true. (true — не атом, как в других языках, а встроенный предикат).

Другой важной частью Prolog являются предикаты. Унарные предикаты выражают свойства их аргументов, тогда как предикаты с несколькими аргументами выражают отношения между ними. Ряд встроенных предикатов языка выполняют ту же роль, что и функции в других языках

В качестве реализации языка Prolog был выбран SWI-Prolog. В данной реализации имеется необходимы функционал, а именно предикаты nb_getval, nb_setval и nb_current. Эти предикаты позволяют использовать другие предикаты в качестве глобальных переменных, что позволяет существенно упростить написание правил для вычисления.

SWI-Prolog — это свободная (открытая) реализация языка программирования Prolog, часто используемая для преподавания и приложений Semantic Web. Эта реализация представляет богатый набор возможностей, библиотеки для constraint logic programming, многопоточности, юнит-тестирования, интерфейс к языку программирования Java, ODBC и т. д., поддерживает литературное программирование, содержит реализацию веб-сервера, библиотеки для SGML, RDF, RDFS.

Ниже приведены некоторые особенности юиблиотеки SWI-Prolog:

– является мощной средой разработки с набором графических инструментов XPCE. Ядро системы лицензировано под GNU LGPL, библиотеки — под GNU GPL с дополнительным условием, позволяющим использование в проприетарных приложениях.

– система довольно популярна, в основном благодаря удобной среде и переносимой библиотеке для создания графического интерфейса. SWI-Prolog, как почти все реализации языка, по большей части реализует Edinburgh Prolog, но также содержит отдельные элементы ISO Prolog.

– SWI-Prolog включает в себя быстрый компилятор, профилировщик, набор библиотек и удобный интерфейс для подключения C-модулей. Он реализован для ряда UNIX-платформ, таких, как HP, IBM Linux, для NeXT, OS/2, Sun и Sparc.

3.1.2 SimPy

Так же было разработано программное решение на языке python с использование фреймворка SimPy. Который позволил за короткий срок на реализовать программное решение для проведения различных тестов.

SimPy - это Python фреймворк процессо-ориентированной дискретно-событийной системы моделирования. Его диспетчеры событий основаны на функциях-генераторах Python.

Также они могут использоваться для создания асинхронных сетей или для реализации мультиагентных систем (с как моделируемым, так и реальным взаимодействием).

Процессы в SimPy - это просто Python генераторы, которые используются для моделирования активных компонентов, например, таких как покупатели, транспортные средства или агенты. SymPy также обеспечивает различные виды общих ресурсов для моделирования точек с ограниченной пропускной способностью (например, серверов, касс, тоннелей). Начиная с версии 3.1, SimPy также будет обеспечить возможности мониторинга для помощи в сборе статистических данных о ресурсах и процессах.

SymPy представляет собой открытую библиотеку символьных вычислений на языке Python. Цель SymPy - стать полнофункциональной системой компьютерной алгебры (CAS), при этом сохраняя код максимально понятным и легко расширяемым. SymPy полностью написан на Python и не требует сторонних библиотек.

SymPy можно использовать не только как модуль, но и как отдельную программу. Программа удобна для экспериментов или для обучения. Она использует стандартный терминал IPython, но с уже включенными в нее важными модулями SymPy и определенными переменными.

3.1.3 Дополнительный инструментарий

В процессе реализации было создано порядка десяти файлов с описанием правил логического вывода общей суммой порядка тысячи строк. Такое большое коли-

чество правил обусловлено наличием деталей и нюансов.

Как и в любом развивающемся программном решении возникает потребность вести учёт версий данного программного решения. Для тих целей использовалась система контроля версий Git. Что позволило не только вести учёт версий программной реализации, но и размещать разрабатываемые материалы на публичном репозитории.

В качестве среды разработки использовался Vim. Данный инструмент позволяет адаптировать его под любой язык программирования, так же в свободном доступе имеются расширения для Prolog, Python, Git и так далее.

3.2 Реализация программного решения

Для реализации поставленной цели была необходимость в решении двух задач:

– реализация среды, в которой должна происходить работа системы управления;

– реализация системы, которая должна посредством доказательства теоремы выполнять управление группой лифтов, тем самым обслуживая поступаемые в течении времени запросы.

Выполнение обоих задач на языке высокого уровня было бы не корректно, так как подобные языки не являются декларативными, что делает наделение системы управления интеллектуальностью значительно усложняет задачу. Но и реализация среды на языке Prolog является не совсем удачно идеей, так как это увеличит количество потребляемых ресурсов. Однако реализовывать эти задачи на разных языках, а потом настроить совместную работу тих решений, является непростой задачей, которая займёт достаточно много времени. Поэтому было принято решение обе задачи реализовать на языке Prolog, так как основной целью работы является разработка интеллектуальной системы управления. Более того, правила для описания среди можно использовать в построении логического вывода.

В процессе работы поставленные задачи были решены разными способами:

– реализация на языке Python;

– реализация но языке Prolog.

Хоть и целесообразность использования языка Prolog понятна, реализация на

Python будет необходима для сравнения обоих решений. А сам язык Python достаточно прост и обладает большим количеством решений для моделирования различных сред, в данном случае используемым решением является библиотека SimPy.

3.2.1 Описание реализации на SimPy

Данная реализация системы управления группой лифтов с ограниченным количеством кабин и несколькими людьми, которые приходят для попадания с одного этажа на другой. В данной системе используется ресурс для моделирования ограниченного количества кабин. Он также определяет процесс выбора кабины и перевозку ей человека.

Когда человек появляется рядом с шахтой лифта, он вызывает первую из свободных кабин. Как только он его кабина забирает, время ожидания человека прекращается. Он, наконец, добирается до нужного этажа и уходит.

После старта системы начинается генерация людей, они появляются после случайного интервала времени, пока продолжается симуляция.

При том, что данная программа является многопоточной, стоит отметить тот факт, что здесь используются общие ресурсы. Эти ресурсы могут использоваться для ограничения количества процессов, использующих их одновременно. Процесс должен запрашивать право использования ресурса. Как только право на использование больше не требуется, оно должно быть выпущено. Данная система смоделирована как ресурс с ограниченным количеством кабин. Люди прибывают к кнопке вызова кабины и просят выполнить перевозку на другой этаж. Если все кабины заняты, человек должен ждать, пока один из пассажиров не закончит поездку и не освободит кабину.

Данная реализация получилась достаточно простой, весь исполняемый скрипт поместился в один файл. Листинг того файла размещён в приложении А. Из листинга видно, как с помощью библиотеки SimPy было выполнено управление общими ресурсами (кабинами). Это выполнялось при помощи функции Resourse. Так же можно заметить как выполнялся запрос на резервирование общих ресурсов:

```
with elev.cab.request() as request:
```

```
        yield request
        info('%s is moving' % (man.name), env)
        yield env.process(elev.moving(man))
```

Таким образом было реализовано и ожидание ресурса, если тот на время появления запроса занят.

3.2.2 Ведение журнала о выполнении программы

Так как разрабатываемые реализации не обладают графическим интерфейсом, очень важно обратить внимание на разработку журнала программы. Посредством разработанных журналов можно детально изучить особенности выполнения программных решений и сделать выводы о корректности работы.

Информация в журнале разделяется на следующие типы:

– INFO – некоторая информация о ходе выполнения или некоторые оповещения;

– TRACE – трассировка программы, позволяет отследить, через какие состояния прошла программа;

– DEBUG – подробная информация о протекающих процессах, данная информация достаточно объемная, поэтому подобная информация выводилась только при отладке программ;

– WARNING – важные предупреждения, которые помогают выявить критичные участки кода, которые могут вести к некорректной работе;

– ERROR – информация о критичных ошибках в выполнении программы. Эта информация помогает быстро исправлять ошибки в коде.

Так же реализована возможность задавать уровень ведения журнала, которая позволяет не захламлять журнал, если нет необходимости в подробной информации. Например, для иллюстрации журнала в отчётах отладочная информация не нужна, или для проведения тестов достаточно получать предупреждения и информацию о результатах проведения теста.

В процессе разработки ведения журнала было выделено четыре уровня:

– вывод только предупреждений и информации об ошибках;

– предупреждения, ошибки и оповещения;

– предупреждения, ошибки, оповещения и трассировка;
– предупреждения, ошибки, оповещения, трассировка и отладочная информация;

Сама же запись в журнал имеет следующую структуру.

```
23 May 2018 22:48:41 [40:4] TRACE Manage people
```

Первые четыре столбца в журнале – это реальное время журналирования момента моделируемого процесса. Следующим столбцом идёт связка двух чисел, первое число - это номер процесса, он необходим для идентификации сессии, а второе число показывает момент времени моделируемого процесса. А дальнейшая информация передаёт сообщения записываемые программой.

3.2.3 Пример работы реализации на SimPy

Данный пример иллюстрирует работу группы лифтов, где их количество равно 2, в здании с 30 этажами. В ходе работы системы появится люди в случайные моменты времени на случайных этажах этажах, и у каждого человека целью является добраться на другой этаж.

Изначально первая кабина находится на первом этаже e_0, а вторя на втором e_1. Ниже приведён лог показывающий работу системы:

```
19 May 2018 16:00:58 [51:null] Session end
23 May 2018 21:41:15 [23:null] Start session
23 May 2018 21:41:15 [23:0] Man 0 appears on floor '0', target '24'
23 May 2018 21:41:15 [23:0] Man 0 is moving
23 May 2018 21:41:15 [23:12] Man 1 appears on floor '14', target '5'
23 May 2018 21:41:15 [23:12] Man 1 is moving
23 May 2018 21:41:15 [23:24] Man 0 has reached by Cab 0
23 May 2018 21:41:15 [23:24] Man 2 appears on floor '2', target '10'
23 May 2018 21:41:15 [23:24] Man 2 is moving
23 May 2018 21:41:15 [23:33] Man 3 appears on floor '3', target '1'
23 May 2018 21:41:15 [23:34] Man 2 has reached by Cab 0
23 May 2018 21:41:15 [23:34] Man 3 is moving
23 May 2018 21:41:15 [23:35] Man 1 has reached by Cab 1
23 May 2018 21:41:15 [23:39] Man 3 has reached by Cab 0
23 May 2018 21:41:15 [23:null] Session end
9 May 2018 16:00:58 [51:null] Session end
```

Изучив выше изложенный журнал, можно увидеть, что за время симуляции появилось 4 человека, каждый человек доставлен.

3.2.4 Реализация на языке Prolog

Перед реализацией системы, которая будет осуществлять управление посредством выполнения логического вывода необходимо реализовать среду, в которой будет протекать процесс, нуждающийся в автоматическом обслуживании. Аналогично реализации на языке Python была выполнена модель на языке Prolog с некоторыми различиями:

– всё действие выполняется в один поток;
– все случайные значения генерируются заранее.

Следует пояснить какие значения задаются случайным образом:

– этаж, которого поступает запрос на обработку;
– этаж, который будет выбран в качестве ли в процессе обработки запроса;
– момент времени, в который поступает запрос.

Моделирование процесса происходит за счёт выполнения логического вывода, что облегчает наделять систему автоматического управления интеллектуальностью. Более того, правила описанные для моделируемого процесса будут использованы для построения решения системой управления.

В приложении А представлен основной файл, в котором реализована инициализация логического вывода. Данный код иллюстрирует реализацию формулы времени. Благодаря функционалу SWI-Prolog в дальнейшем предикаты отражающие время можно будет указывать в правилах только при необходимости.

Реализации правил manage_people и manage_elevators вынесены в другие файлы, как как являются весьма громоздкими. Manage_people отвечает за внешний фактор (случайное появление нуждающихся в лифте людей). А manage_elevators включает в себя формулы движения лифтов.

Что касается инициализации моделируемого процесса, то из журнала можно узнать все детали начального состояния модели.

```
23 May 2018 22:48:41 [40:null] DEBUG Show var: Steps '20' Elev '2'
   Floors '5' People '2'
23 May 2018 22:48:41 [40:null] TRACE Start modulation
23 May 2018 22:48:41 [40:null] DEBUG Init people apprears '[2,4]'
```

```
23 May 2018 22:48:41 [40:null] DEBUG Init people floors '[1,0]'
23 May 2018 22:48:41 [40:null] DEBUG Init people targets '[4,4]'
23 May 2018 22:48:41 [40:null] DEBUG Init people waiting '[0,0]'
23 May 2018 22:48:41 [40:null] DEBUG Init people states '[0,0]'
23 May 2018 22:48:41 [40:null] DEBUG Init elevators floors '[0,1]'
```

Из данной части журнала можно выяснить следующие вещи:

– моделируемый процесс будет протекать в течении 20 единиц времени;

– количество кабин равняется двум;

– количество этажей равняется пяти;

– количество поступаемых запросов равняется двум;

– первый запрос появится во после истечения второй единицы времени, а второй после четвёртой;

– запросы будут со второго и первого этажа соответственно;

– целью обоих запросов является этаж под номером 4;

– кабины расположены на первом и втором этажах.

Во время инициализации моделируемого процесса нет информации о текущей единице времени, поэтому пишется null.

За время выполнения моделируемого процесса будет в журнале будет саккумулирована вся промежуточная информация. Это позволяет более ясно представить картину о произошедших действиях. Особенно такой функционал важен при отладке программного решения, так как в момент внеплановой остановки программы информации предоставляемой инструментарием SWI-Prolog может не хватать. Тем более программа может вести себя некорректно и без внезапных остановок.

После завершения моделируемого процесса необходимо собрать данные о результатах, которые необходимы для оценки разработанного программного решения.

Разработанное ведение журнала позволяет записывать различную информацию о принципах проходимого процесса.

```
23 May 2018 22:48:41 [40:18] TRACE Start step
23 May 2018 22:48:41 [40:18] TRACE Finish step
23 May 2018 22:48:41 [40:19] TRACE Start step
23 May 2018 22:48:41 [40:19] TRACE Finish step
23 May 2018 22:48:41 [40:null] TRACE Finith modulation
```

```
23 May 2018 22:48:41 [40:null] TRACE Show statistics
23 May 2018 22:48:41 [40:null] INFO People status results: '[3,3]'
23 May 2018 22:48:41 [40:null] INFO People waiting results: '[1,1]'
```

С помощью приведённой чисти журнал, можно собрать некоторую статистику о проделанной работе. Например, по результатам показанной части журнала можно сделать вывод, что все пассажиры доставлены и ожидание каждого их них не составило дольше одной единицы времени.

В приложении Б при ведён журнал, вмещающий в себя информацию о деталях работы моделируемого процесса от запуска до завершения.

3.2.5 Блок управления на языке Prolog

Вот время работы кабины имеют график, в котором отражено какие этажи будут посещены донной кабиной. Таким образом каждая камина обладает списком задач, который необходимо выполнить. Если этот список пустует, то кабина находится в покое. В момент поступления запроса система управления должна принять решение, в чью очередь задач должен попасть этаж, с которого поступил запрос. То есть система должна вычислить какая кабина ближе к этажу, откуда поступил запрос, с учётом очереди задач кабины.

Ниже приведена часть программы, которая отвечает за поиск наиболее подходящей кабины:

```
get_dist(Pos, [], Floor, Res) :- Res is abs(Pos - Floor).
get_dist(Pos, [H | T], Floor, Res) :-
        (H = -1 ->
                get_dist(Pos, T, Floor, Dist),
                Res is Dist + 1
        ;
                get_dist(H, T, Floor, Dist),
                Res is Dist + abs(Pos - H)
        ).

fill_dist(0, _, Dist, Dist).
fill_dist(Ind, Floor, Dist, Res) :-
        NextInd is Ind - 1,
        var_getvalue(elevators_floors, ElevFloors),
        get_elem(ElevFloors, NextInd, 0, ElevPos),
```

```
        atom_concat('elev_rmap_', NextInd, MapName),
        term_string(MapTerm, MapName),
        var_getvalue(MapTerm, Map),
        get_dist(ElevPos, Map, Floor, D),
        set_elem(Dist, NextInd, 0, D, NewDist),
        fill_dist(NextInd, Floor, NewDist, Res).

find_available_elev(Floor) :-
        nb_getval(n_elevators, NElev),
        zero_list(NElev, Dist),
        fill_dist(NElev, Floor, Dist, Distances),
        swritef(ElevDistLog, 'Current distances are \'%t\'', [Distances
           ]),
        logdebug(ElevDistLog),
        get_min_dist_id(Distances, MinDist, Elev),
        swritef(ElevMinDistLog, 'Current min dist \'%t\' with id \'%t
           \'', [MinDist, Elev]),
        logdebug(ElevMinDistLog),
        append2map(Elev, Floor).

find_in_list([], _, Res) :- Res = false.
find_in_list([H | T], Val, Res) :-
        (H = Val ->
                Res = true
        ;
                find_in_list(T, Val, Res)
        ).

find_in_elev_lists(_, Ind, Res) :- Ind =< 0, Res = false.
find_in_elev_lists(Floor, Ind, Res) :-
        Ind >= 0,
        NextInd is Ind - 1,
        atom_concat('elev_rmap_', NextInd, ListName),
        term_string(ListTerm, ListName),
        var_getvalue(ListTerm, List),
        find_in_list(List, Floor, FRes),
        (FRes ->
                Res = true
        ;
                find_in_elev_lists(Floor, NextInd, Res)
        ).

find_floor(Floor, Res) :-
        nb_getval(n_elevators, NElev),
```

```
        find_in_elev_lists(Floor, NElev, Res).

elev_call(Floor) :-
        find_floor(Floor, IsFloorInMaps),
        (IsFloorInMaps ->
                logdebug('Floor is in maps')
        ;
                logtrace('Putting floor to map'),
                find_available_elev(Floor)
        ).
```

Представленный выше код от иллюстрирует алгоритм, по которому происходит поиск решения. Данная реализация является более развитой, чем реализация представленная на языке Python, но всё равно поиск производится по достаточно простому принципу.

3.2.6 Интеллектуальная реализация на языке Prolog

Как уже было замечено, при поступлении запроса система управления имеет целый перечень вариантов решения. И почти каждое решение влечёт за собой различные варианты развития событий. Возможны даже ситуации, когда с очки зрения уже про иллюстрированного подхода вариант решения является не подходящем, это решение может оказать самым наилучшим ввиду меняющихся условий с течением времени.

В приложении А размещён код, который иллюстрирует реализацию формулы реализующей следующие функции:

- обход возможных вариантов будущего;
- сбор статистики с каждого варианта;
- выбор наиболее подходящего варианта будущего по признаку.

В данном случае интересующим признаком является среднее ожидание человеком кабины. За время поиска наиболее подходящего варианта решения, про исходит обход дерева возможных состояний системы, рассмотрев варианты и произведя оценку система выбирает более более подходящую кабину. Часть приведённая ниже ниже как раз демонстрирует этот выбор.

```
do_simulate(Floor, Elev, H) :-
        nb_getval(current_sim, SimPrefix),
        append2map(Elev, Floor),
        manage_elevators,
        var_getvalue(step, Step),
        Next is Step + 1,
        var_setvalue(step, Next),
        simulate_loop(SimPrefix),
        sim_getval(SimPrefix, people_waiting, PeopleWaitingList),
        sum_list(PeopleWaitingList, H),
        swritef(SimLog, 'Waiting sum is \'%t\'', [H]),
        logdebug(SimLog).
simulate(Floor, Elev, H) :-
        (nb_current(current_sim, CurrentPrefix) ->
                atom_concat(CurrentPrefix, Elev, SimPrefix),
                atom_concat(SimPrefix, '_', Prefix),
                init_sim(Prefix),
                nb_setval(current_sim, Prefix),
                do_simulate(Floor, Elev, H),
                nb_setval(current_sim, CurrentPrefix)
        ;
                atom_concat('r_', Elev, SimPrefix),
                atom_concat(SimPrefix, '_', Prefix),
                init_sim(Prefix),
                nb_setval(current_sim, Prefix),
                do_simulate(Floor, Elev, H),
                nb_delete(current_sim)
        ).
```

Таким образом данная система является более интеллектуально в сравнении с предыдущем вариантом.

3.2.7 Ведение журнала для интеллектуальной реализации

Учитывая тот факт что поиск решения усложнился, следует полагать, что возросла потребность в более информативных записях в журнале. Для записей из такого журнала нужно добавить информацию как проходил обход имеющихся решений.

Изучив изложенный приложении Б журнал, можно увидеть, что происходит построение дерева формулы и её обход. Для того чтобы было нагляднее следует прокомментировать строку лога.

```
with elev.cab.request() as request:
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Start step
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Do loop step
```

Первые четыре столбца в логе - это реальное время журналирования момента моделируемого процесса. Следующим столбцом идёт связка двух чисел, первое число - это номер процесса, он необходим для идентификации сессии, а второе число показывает момент времени моделируемого процесса. Ещё одним столбцом является связка строки и числа, число - это так же момент времени в данной ветки формулы, А строка отражает индекс чанка формулы в момент вывода, r означает корень выводимой формулы, а дальше серез нижние подчёркивание перечислены индексы кабин, которые участвуют в логическом выводе в данные момент.

Так же следует упомянуть, что отдельное внимание требует инициализация обработки одного из решений:

```
24 May 2018 06:49:49 [71:2] TRACE Copy var 'step' into 'r_0_step' with
   val '2'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'people_targets' into '
   r_0_people_targets' with val '[4,4]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'people_floors' into '
   r_0_people_floors' with val '[1,0]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'people_waiting' into '
   r_0_people_waiting' with val '[0,0]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'people_states' into '
   r_0_people_states' with val '[1,0]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elevators_floors' into '
   r_0_elevators_floors' with val '[0,1]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elev_rmap_1' into '
   r_0_elev_rmap_1' with val '[]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elev_people_1' into '
   r_0_elev_people_1' with val '[]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elev_rmap_0' into '
   r_0_elev_rmap_0' with val '[]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elev_people_0' into '
   r_0_elev_people_0' with val '[]'
```

Рассмотрев от журнал можно сделать вывод, что при инициализации рассмотрения варианта решения происходит копирование переменных окружения. Это необходимо для того, что бы основные данные при моделировании небыли испорчены.

3.2.8 Реализация на языке Prolog с элементом вероятности

Очевидно, что очень сильно упрощена. Более того в реальной жизни не всегда известно в какое время ожидать запрос на обработку.

Добавление правила, приближающую текущую реализацию к реальной жизни, подчеркнёт гибкость разработанного программного продукта. И это докажет что данная система легко настраиваемая под разного рода задачи.

С помощью пары правил добавляется элемент вероятности к появлению запроса:

```
(nb_current(current_sim, _) -> TheVer = Prob ; TheVer = Ver),
(TheVer >= 50 ->
        get_people_elem(people_floors, Id, Floor),
        swritef(AppearLog, 'A man appears with id \'%t\' on floor with
           id \'%t\'', [Id, Floor]),
        loginfo(AppearLog),
        set_people_elem(people_states, Id, 1),
        elev_call(Floor)
;
        set_people_elem(people_states, Id, -1),
        loginfo('A man does not appear')
)
```

Такие изменения повлекут за собой ложные запросы, которые будут рассмотрены системой управления.

Получив такой вариант программы, можно сделать вывод, что данная реализация легко может быть адаптирована под любую систему или среду.

3.2.9 Добавление ограничений на логический вывод

В ходе выполнения тестирования обнаружился интересный факте, более интеллектуальные реализации прекращали работу до её завершения из-за недостатка ресурсов.

Естественно, вопрос заключается в том, что когда программа проходит возможные варианты, программное решение выполняет слишком много операций.

В данном случае решением является правило ограничивающие то, на сколько много последовательных запросов в будущем система должна учитывать.

```
simulate_loop(SimPrefix) :- nb_getval(deep, Deep), string_length(
   SimPrefix, CurrentDeep), Deep =< CurrentDeep.
simulate_loop(SimPrefix) :-
        nb_setval(current_sim, SimPrefix),
        sim_getval(SimPrefix, step, Step),
        nb_getval(n_steps, Steps),
        Step >= Steps,
        swritef(SimLog, 'Simulation \'%t\' is finished', [SimPrefix]),
        logtrace(SimLog),
        show_stat.
simulate_loop(SimPrefix) :-
        nb_setval(current_sim, SimPrefix),
        var_getvalue(step, Step),
        nb_getval(n_steps, Steps),
        Step < Steps,
        logtrace('Start step'),
        simulate_loop_step,
        logtrace('Finish step'),
        Next is Step + 1,
        var_setvalue(step, Next),
        simulate_loop(SimPrefix).
```

После такой реализации более интеллектуальное решение стало чаще ошибаться. В результате имеется два варианта: иметь дорогое высокопроизводительное оборудование или пользоваться менее интеллектуальной программой.

3.3 Сравнительный анализ

Как было упомянуто выше одним из основных критериев является средняя длительность ожиданий. Так же можно сравнить время выполнения моделируемого процесса. С помощью среднего времени ожидания можно оценить эффективность разработанного решения, и, используя время выполнения программ можно произвести оценку ресурсозатратности разработанного решения.

Проведения анализа необходимо провести тестирование, имеющихся программных решений. По результатам данного тестирования можно будет произвести оценку эффективности и ресурсозатратности.

3.3.1 Условия проведения тестирования

Основными исходными данными являются: количество кабин, количество людей участвовавших при моделировании, точнее количество поступивших запросов, и количество обслуживаемых этажей. Все упомянутые данные можно менять и производить большое количество разноплановых тестов.

В данном тестировании в качестве констант были взяты количество лифтов и количество людей. С помощью такого рода тестов можно произвести оценку не только упомянутых свойств системы, но и увидеть какова типа инфраструктура нуждается или не нуждается в разработанных решениях.

Так же был выбран ряд значений для количества обслуживаемых этажей: 30, 15 и 5. Для зданий со сложной инфраструктурой подходит значение в 30 этажей, для зданий с менее развитой инфраструктурой 15 этажей и для ещё менее 5 этажей.

Для сбора дынных производился многочисленный запуск моделируемого процесса и был произведён подсчёт средних значений. Результаты сбора данных приведены в таблицах 3.1-3.4. Для упрощения представления полученных данных были даны названия реализациям: более интеллектуальным способом реализации является I тип, а более тривиальным является II тип.

3.3.2 Сравнение реализаций

Невзирая на тот факт, что модели типов I и II сильно упрощены, уже можно произвести оценку их эффективности и ресурсозатратности по данным в таблицах 3.1 и 3.2.

Таблица 3.1 – Сравнение по среднему времени ожидания

Способ управления	30 этажей	15 этажей	5 этажей
I тип	7.5	6	2.7
II тип	9.6	7.1	3.2

По результатам из таблицы 3.1 можно сделать вывод, что более интеллектуальная реализация оказалась эффективнее тривиальной во всех условиях.

Таблица 3.2 – Сравнение по времени выполнения

Способ управления	30 этажей	15 этажей	5 этажей
I тип	20.516 с	3,711 с	0,527 с
II тип	0,153 с	0,134 с	0,112 с

Однако, по результатам из таблицы 3.2 очевидно, что более интеллектуальная реализация оказалась в десятки раз медленнее, чем более простая реализация.

3.3.3 Сравнение реализаций с элементом вероятности

Для тестирования реализаций с элементом вероятности появления запроса, было добавлено 5 человек (запросов), имеют низкую вероятность появления.

При таких условиях возникает вопросы:

– будет ли система I типа всё ещё эффективной;

– на сколько сильно процесс проведения моделируемого процесса затянется, если затянется.

Конечно, было бы нечестно реализации I типа не дать возможность оценки вероятности появления запроса. Поэтому было добавлено правило учитывать вероятность появления запроса.

Таблица 3.3 – Сравнение по среднему времени ожидания

Способ управления	30 этажей	15 этажей	5 этажей
I тип	7.8	5.5	3.4
II тип	9.6	7.1	3.2

По результатам из таблицы 3.3 можно сделать вывод, что более интеллектуальная реализация остаётся всё ещё эффективнее тривиальной, но при небольшом числе этажей интеллектуальная реализация оказалась менее эффективной.

Таблица 3.4 – Сравнение по времени выполнения

Способ управления	30 этажей	15 этажей	5 этажей
I тип	56.791 с	40.739 с	32.957 с
II тип	0.171 с	0.149 с	0.127 с

Более того, по результатам из таблицы 3.4 видно, что более интеллектуальная реализация оказалась ещё медленнее, в то время как тривиальная реализация в по-

казателях не изменилась.

Полученные результаты можно оправдать тем, что интеллектуальная реализация тратила время на обработку запросов, которые по факту не произошли, и это не помешало её превзойти более тривиальную реализацию на больших данных.

3.3.4 Сравнение реализаций на языках Prolog и Python

Входе выполнения экспериментов на выполнение моделируемого процесса с ещё большими числами не хватило вычислительной мощи оборудования для реализаций на Prolog. Что иллюстрирует потребность в ресурсах у данных подходов к решению задачи.

Стоит отметить, что программное решение было выполнено и на языке Python, которое ни обладало особой интеллектуальностью и не не несло в себе какого-то нового алгоритма. Поведённые тесты показали следующий результат: среднее время ожидания – 12.2 и время выполнения – 0.065 секунд.

Хоть и результат на языке Python, и является более быстрым, но в то же время имеет наихудший результат по среднему времени ожидания. Наверняка, попытка вынести функции интеллектуальной системы на более быструю платформу будет целесообразным.

3.4 Выводы

Получив результаты изложенные выше в таблицах 3.1-3.4, можно рассмотреть целесообразность разработки интеллектуальной системы управления. Сравнительный анализ показал, что среднее время ожидания человеком значительно меньше, чем у более тривиального решения. Также результаты можно сравнить с помощью диаграммы на рисунке 3.1, где отчётливо видно, что продвинутое программное решение является более эффективном.

Более того, продвинутое решение требует небольшого улучшения, которое позволит с лёгкостью пройти тест на небольшом количестве этажей. Улучшение будет собой представлять не большое правило, которое бы могло увеличить глубину обхода дерева вариантов решений.

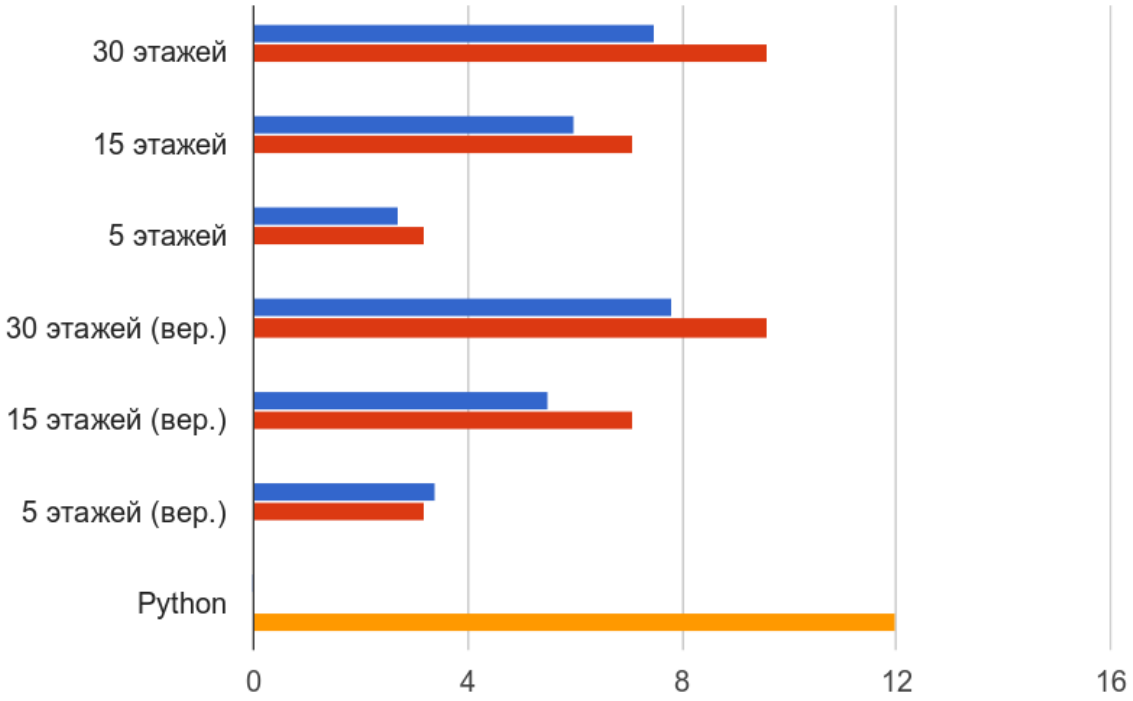

Рисунок 3.1 – Сравнение по среднему времени ожидания

Но не стоит забывать, что чем сложнее требуемый логический вывод, тем больше ресурсов необходимо системе для функционирования. Поэтому необходимо обратить внимание на диаграмму, которая представлена на рисунке 3.2.

Из диаграммы видно, что время, которое затрачено на построение логического вывода достаточно велико. Это показывает, что для своевременного решения необходимо достаточно большое количество ресурсов. Несмотря на тот факт, что на стройка разработанного программного обеспечения не будет занимать длительное время, что делает продукт более дешёвым, поддержание данной системы становится дорогостоящим процессом, за счёт потребности в дорогостоящем оборудовании.

Исходя из этого имеется необходимость заняться оптимизацией данного программного решения, которое позволит снизить потребность в большом объёме ресурсов.

И тем не менее, в наше время высокопроизводительное оборудование быстро удешевляется, развивается и становится более доступным. Более того, для мест со сложной инфраструктурой затраты на разработанное программное решение будет вполне целесообразным.

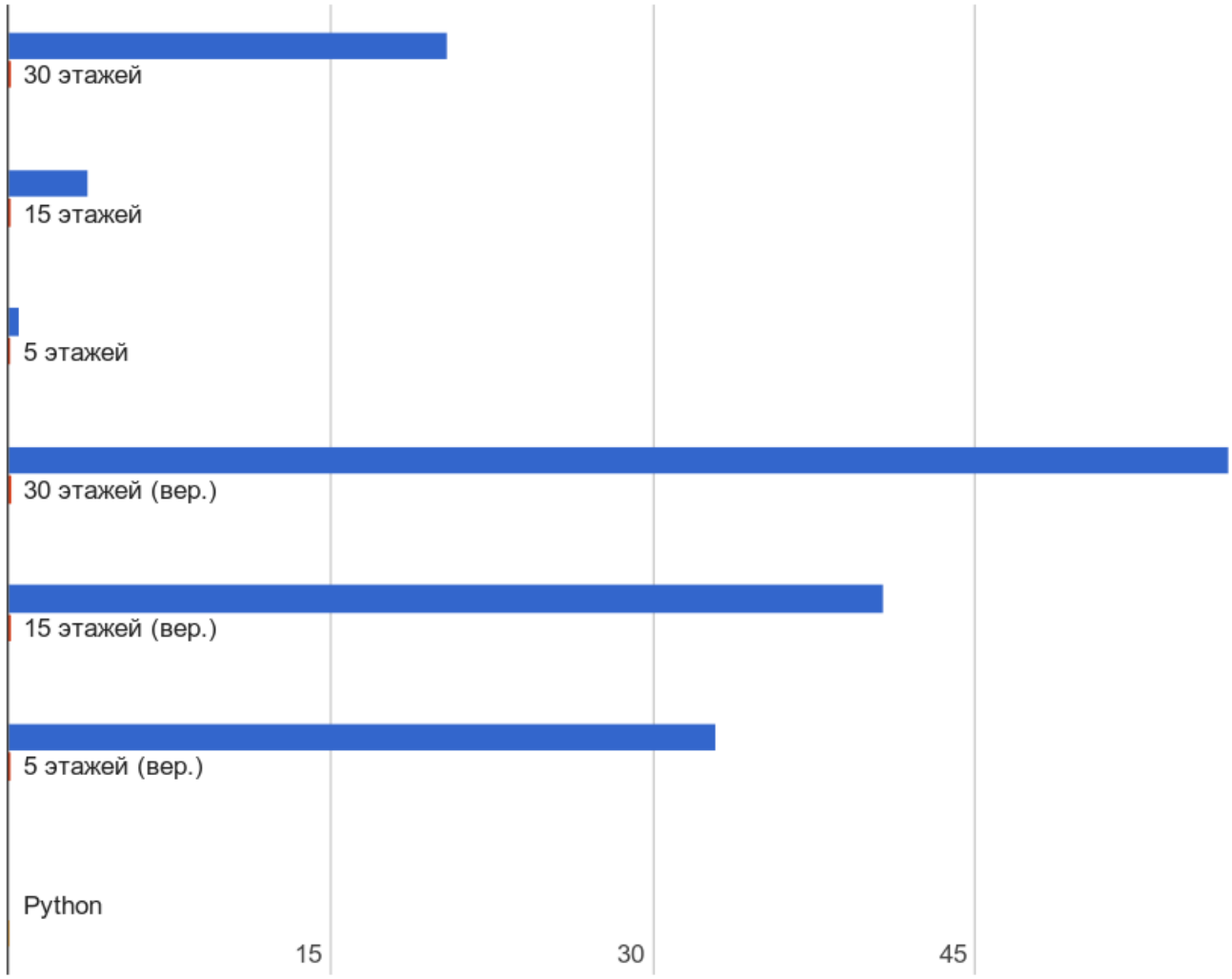

Рисунок 3.2 – Сравнение по времени выполнения

Важно подчеркнуть ещё тот, что данное программная реализация является решением двух задач: моделирование среды и построение логического вывода. А это значит, что если выделить блок отвечающий за моделирование среды и реализовать его на оптимизированном движке, то сразу будет получен большой прирост прирост в производительности.

Таким образом, разработав данное программное решение, была получена платформа для разработки новых программных решений, где можно будет выделить у данной системы два блока, с которыми будет возможно вести независимую разработку. Это позволит ускорить процесс развития данной системы и появится возможность заменять эти блоки на более сложные решения.

Заключение

Несмотря на тот факт, что более сложные решения в управлении лифтовыми группами нуждаются в большом объёме ресурсов, интеллектуальные системы управления позволяют своевременно справляться с пассажиропотоком в среде с быстро меняющимися условиями. А также подобная система позволит учитывать вновь возникшие условия и принципы среды, в которой происходит управление, без существенных изменений в программной реализации, что снижает стоимость поддержки данной системы. Плюс, упрощается процесс разработки сложных алгоритмов управления.

Полученные при программном моделировании результаты показали, что посредством добавления дополнительных условий можно улучшать интересуемые показатели, например: среднее время ожидания кабины пассажиром и характеристики построения логического вывода.

Принцип рассмотрения вариантов возможного будущего, позволяет сделать систему управления более гибкой в отношении к особенностям пассажиропотока, зависящими от времени суток или другим временным факторам.

Разработанные программные решения могут быть использованы для моделирования более сложных сред для проведения экспериментов и построения гипотез.

Следующим шагом в развитии данной работы будет разбиение разработанного программного решения на два бока: логический и физический. Где логический блок будет отвечать за принятие решения, а физический блок за физические процессы, протекаемые в построенной модели. После чего можно будет вести разработку двух блоков независимо друг от друга. Что приведёт к использованию новой оптимизированной интеллектуальной системы управления и замене физической модели на реальную систему лифтов.

Список использованных источников

1. А. А. Ларионов. Программные технологии для эффективного поиска логического вывода в исчислении позитивно-образованных формул / А. А. Ларионов, Е. А. Черкашин – Иркутск : Изд-во ИГУ, 2013. 104 с.
2. Sutcliffe G., Suttner C. The TPTP problem library – CNF release v. 1.2.1 / Journal of Automated Reasoning. 1998. С. 177-203 104 с.
3. ГОСТ 34.602.89. Информационная технология. Комплекс стандартов на автоматизированные системы.Техническое задание на создание автоматизированной системы. - Введ. 01-01-1990 - М.: Госстандарт России : Изд-во стандартов, 2016. 11 с.
4. ГОСТ 34.601.90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания. - Введ. 01-01-1992 - М.: Госстандарт России : Изд-во стандартов, 2016. 5 с.
5. ГОСТ 34.603.92. Информационная технология. Виды испытаний автоматизированных систем. - Введ. 01-01-1993 - М.: Госстандарт России : Изд-во стандартов стандартов, 2016. 5 с.
6. ГОСТ 2.114-95. Единая система конструкторской документации. Технические условия. - Введ. 01-01-1996 - М.: Госстандарт России : Изд-во стандартов, 2016. 12 с.
7. ГОСТ Р 53736-2009. Изделия электронной техники. Порядок создания и постановки на производство. Основные положения. - Введ. 01-01-2011 - М.: Госстандарт России : Изд-во стандартов, 2016. 50 с.
8. СанПиН 2.2.2/2.4.1340-03. О введении в действие санитарно-эпидемиологических правил и нормативов. - Введ. 01-01-2003 - М.: Федеральный центр гигиены и эпидемиологии Роспотребнадзора, 2017. 6 с.
9. Требования к оформлению текстовой и графической документации. Нормоконтроль. - И.: ИрГУПС, 2017. 46 с.

Приложение А

Листинги программ

Листинг реализации на SimPy:

```
import simpy
import random
import datetime
import os
LOGFILE = 'project.log'
T_INTER = 5
SIM_TIME = 40
NUM_FLOORS = 30
def info(message, env = None):
    now = datetime.datetime.now()
    time = 'null'
    if env is not None :
        time = '%.2f' % (env.now)
    file = open(LOGFILE, 'a')
    #  file.write('%s %s\n' % (NOW.strftime("%Y-%m-%d %H:%M"), message)
       )
    file.write('%s [%s:%s] %s\n'
               % (now.strftime("%d %b %Y %X"), os.getpid(), time,
                  message))
    file.close()
class Man:
    def __init__(self, ind):
        self.name = 'Man %d' % ind
        self.floor = random.randint(0, NUM_FLOORS // 2)
        self.target = self.floor
        while (self.target == self.floor):
            self.target = random.randint(0, NUM_FLOORS)
        info('%s appears on floor \'%s\' with target \'%s\''
             % (self.name, self.floor, self.target), env)
class Cabine(object):
    def __init__(self):
        self.state = 0
        self.floor = 0
class Elevator(object):
    def __init__(self, env, num_machines):
        self.env = env
        self.cab = simpy.Resource(env, num_machines)
        self.cabs = [Cabine() for i in range(num_machines)]
        self.n_cabs = num_machines
```

```
    def get_free_cab(self):
        for i in range(self.n_cabs):
            if (self.cabs[i].state == 0):
                return i
    def moving(self, man):
        cid = self.get_free_cab()
        self.cabs[cid].state = 1
        moving_time = abs(man.target - man.floor)
        waiting = abs(man.floor - self.cabs[cid].floor) + moving_time
        yield self.env.timeout(waiting)
        self.cabs[cid].state = 0
        info('%s has reached by Cab %d' % (man.name, cid), env)

def call(env, ind, elev):
    man = Man(ind)
    with elev.cab.request() as request:
        yield request
        info('%s is moving' % (man.name), env)
        yield env.process(elev.moving(man))
def people_generator(env):
    elev = Elevator(env, 2);
    i = 0
    env.process(call(env, i, elev))
    while True:
        yield env.timeout(random.randint(T_INTER - 2, T_INTER + 2))
        i += 1
        env.process(call(env, i, elev))
info('Start session')
env = simpy.Environment()
env.process(people_generator(env))
env.run(until=SIM_TIME)
info('Session end')
```

Основной файл, инициализация логического вывода:

```
:- ensure_loaded('conf.pl').
:- ensure_loaded('log.pl').
:- ensure_loaded('util.pl').
:- ensure_loaded('peopletool.pl').
:- ensure_loaded('elevatortool.pl').
:- ensure_loaded('maintool.pl').
:- ensure_loaded('init.pl').
do_loop_step :-
        manage_people,
```

```
        manage_elevators.
do_loop :-
        (nb_current(step, _) ->
                false
        ;
                logwarn('Var \'Step\' has null value'),
                loginfo('Set first step \'0\''),
                nb_setval(step, 0)
        ),
        do_loop.
do_loop :-
        var_getvalue(step, Step),
        nb_getval(n_steps, Steps),
        Step >= Steps.
do_loop :-
        var_getvalue(step, Step),
        nb_getval(n_steps, Steps),
        Step < Steps,
        logtrace('Start step'),
        do_loop_step,
        logtrace('Finish step'),
        Next is Step + 1,
        var_setvalue(step, Next),
        do_loop.
run:-
        show_var,
        logtrace('Start modulation'),
        init,
        do_loop,
        logtrace('Finith modulation'),
        show_stat.
% GOAL
:- run.
```

Обход возможных вариантов обработки запроса:

```
init_sim(SimPrefix) :-
        copy_var(SimPrefix, step),
        copy_var(SimPrefix, people_targets),
        copy_var(SimPrefix, people_floors),
        copy_var(SimPrefix, people_waiting),
        copy_var(SimPrefix, people_states),
        copy_var(SimPrefix, elevators_floors),
        nb_getval(n_elevators, NElev),
        copy_elev_list(SimPrefix, NElev).
```

```
simulate_loop_step :-
        logtrace('Do loop step'),
        manage_people,
        manage_elevators.
simulate_loop(SimPrefix) :-
        nb_setval(current_sim, SimPrefix),
        sim_getval(SimPrefix, step, Step),
        nb_getval(n_steps, Steps),
        Step >= Steps,
        swritef(SimLog, 'Simulation \'%t\' is finished', [SimPrefix]),
        logtrace(SimLog),
        show_stat.
simulate_loop(SimPrefix) :-
        nb_setval(current_sim, SimPrefix),
        var_getvalue(step, Step),
        nb_getval(n_steps, Steps),
        Step < Steps,
        logtrace('Start step'),
        simulate_loop_step,
        logtrace('Finish step'),
        Next is Step + 1,
        var_setvalue(step, Next),
        simulate_loop(SimPrefix).
simulate :-
        (nb_current(current_sim, CurrentPrefix) ->
                SimPrefix = CurrentPrefix
        ;
                SimPrefix = 'r_'
        ),
        init_sim(SimPrefix),
        simulate_loop(SimPrefix).
do_simulate(Floor, Elev, H) :-
        nb_getval(current_sim, SimPrefix),
        append2map(Elev, Floor),
        manage_elevators,
        var_getvalue(step, Step),
        Next is Step + 1,
        var_setvalue(step, Next),
        simulate_loop(SimPrefix),
        sim_getval(SimPrefix, people_waiting, PeopleWaitingList),
        sum_list(PeopleWaitingList, H),
        swritef(SimLog, 'Waiting sum is \'%t\'', [H]),
        logdebug(SimLog).
simulate(Floor, Elev, H) :-
```

```
        (nb_current(current_sim, CurrentPrefix) ->
                atom_concat(CurrentPrefix, Elev, SimPrefix),
                atom_concat(SimPrefix, '_', Prefix),
                init_sim(Prefix),
                nb_setval(current_sim, Prefix),
                do_simulate(Floor, Elev, H),
                nb_setval(current_sim, CurrentPrefix)
        ;
                atom_concat('r_', Elev, SimPrefix),
                atom_concat(SimPrefix, '_', Prefix),
                init_sim(Prefix),
                nb_setval(current_sim, Prefix),
                do_simulate(Floor, Elev, H),
                nb_delete(current_sim)
        ).
calculating(_, Elev, []) :-
        nb_getval(n_elevators, NElev),
        Elev >= NElev.
calculating(Floor, Elev, [H | T]) :-
        nb_getval(n_elevators, NElev),
        Elev < NElev,
        simulate(Floor, Elev, H),
        Next is Elev + 1,
        calculating(Floor, Next, T).
calculate(Floor, ResList) :-
        calculating(Floor, 0, ResList).
do_elev_call(Floor, Elev) :-
        calculate(Floor, ResList),
        min_list(ResList, Min),
        nth0(Elev, ResList, Min).
```

Приложение Б

Журналы выполнения программ

Журнал, отражающий выполнение программы от старта до завершения:

```
23 May 2018 22:48:41 [40:null] TRACE Start modulation
23 May 2018 22:48:41 [40:null] WARNING Var 'Step' has null value
23 May 2018 22:48:41 [40:null] INFO Set first step '0'
23 May 2018 22:48:41 [40:0] TRACE Start step
23 May 2018 22:48:41 [40:0] TRACE Finish step
23 May 2018 22:48:41 [40:1] TRACE Start step
23 May 2018 22:48:41 [40:1] TRACE Finish step
23 May 2018 22:48:41 [40:2] TRACE Start step
23 May 2018 22:48:41 [40:2] INFO A man appears with id '0' on floor
   with id '1'
23 May 2018 22:48:41 [40:2] TRACE Putting floor to map
23 May 2018 22:48:41 [40:2] TRACE moving_elevators NElev '2' Id '0'
23 May 2018 22:48:41 [40:2] TRACE moving_elevators NElev '2' Id '1'
23 May 2018 22:48:41 [40:2] TRACE Manage people
23 May 2018 22:48:41 [40:2] INFO Going out people '[]' from elev '1'
23 May 2018 22:48:41 [40:2] TRACE Trace getting_people_in '2'
23 May 2018 22:48:41 [40:2] TRACE Trace getting_people_in '1'
23 May 2018 22:48:41 [40:2] INFO Coming people '[0]' to elev '1'
23 May 2018 22:48:41 [40:2] TRACE Finish step
23 May 2018 22:48:41 [40:3] TRACE Start step
23 May 2018 22:48:41 [40:3] TRACE moving_elevators NElev '2' Id '0'
23 May 2018 22:48:41 [40:3] TRACE moving_elevators NElev '2' Id '1'
23 May 2018 22:48:41 [40:3] TRACE Elevator '1' is closing the door
23 May 2018 22:48:41 [40:3] TRACE Change ElevPos '1' Id '1'
23 May 2018 22:48:41 [40:3] TRACE Finish step
23 May 2018 22:48:41 [40:4] TRACE Start step
23 May 2018 22:48:41 [40:4] INFO A man appears with id '1' on floor
   with id '0'
23 May 2018 22:48:41 [40:4] TRACE Putting floor to map
23 May 2018 22:48:41 [40:4] TRACE moving_elevators NElev '2' Id '0'
23 May 2018 22:48:41 [40:4] TRACE Manage people
23 May 2018 22:48:41 [40:4] INFO Going out people '[]' from elev '0'
23 May 2018 22:48:41 [40:4] TRACE Trace getting_people_in '2'
23 May 2018 22:48:41 [40:4] TRACE Trace getting_people_in '1'
23 May 2018 22:48:41 [40:4] INFO Coming people '[1]' to elev '0'
23 May 2018 22:48:41 [40:4] TRACE moving_elevators NElev '2' Id '1'
23 May 2018 22:48:41 [40:4] TRACE Change ElevPos '2' Id '1'
23 May 2018 22:48:41 [40:4] TRACE Finish step
23 May 2018 22:48:41 [40:5] TRACE Start step
```

```
23 May 2018 22:48:41 [40:5] TRACE moving_elevators NElev '2' Id '0'
23 May 2018 22:48:41 [40:5] TRACE Elevator '0' is closing the door
23 May 2018 22:48:41 [40:5] TRACE Change ElevPos '0' Id '0'
23 May 2018 22:48:41 [40:5] TRACE moving_elevators NElev '2' Id '1'
23 May 2018 22:48:41 [40:5] TRACE Change ElevPos '3' Id '1'
23 May 2018 22:48:41 [40:5] TRACE Finish step
23 May 2018 22:48:41 [40:6] TRACE Start step
23 May 2018 22:48:41 [40:6] TRACE moving_elevators NElev '2' Id '0'
23 May 2018 22:48:41 [40:6] TRACE Change ElevPos '1' Id '0'
23 May 2018 22:48:41 [40:6] TRACE moving_elevators NElev '2' Id '1'
23 May 2018 22:48:41 [40:6] TRACE Manage people
23 May 2018 22:48:41 [40:6] INFO Going out people '[0]' from elev '1'
23 May 2018 22:48:41 [40:6] TRACE Trace getting_people_in '2'
23 May 2018 22:48:41 [40:6] TRACE Trace getting_people_in '1'
23 May 2018 22:48:41 [40:6] INFO Coming people '[]' to elev '1'
23 May 2018 22:48:41 [40:6] TRACE Finish step
23 May 2018 22:48:41 [40:7] TRACE Start step
23 May 2018 22:48:41 [40:7] TRACE moving_elevators NElev '2' Id '0'
23 May 2018 22:48:41 [40:7] TRACE Change ElevPos '2' Id '0'
23 May 2018 22:48:41 [40:7] TRACE moving_elevators NElev '2' Id '1'
23 May 2018 22:48:41 [40:7] TRACE Elevator '1' is closing the door
23 May 2018 22:48:41 [40:7] TRACE Finish step
23 May 2018 22:48:41 [40:8] TRACE Start step
23 May 2018 22:48:41 [40:8] TRACE moving_elevators NElev '2' Id '0'
23 May 2018 22:48:41 [40:8] TRACE Change ElevPos '3' Id '0'
23 May 2018 22:48:41 [40:8] TRACE moving_elevators NElev '2' Id '1'
23 May 2018 22:48:41 [40:8] TRACE Finish step
23 May 2018 22:48:41 [40:9] TRACE Start step
23 May 2018 22:48:41 [40:9] TRACE moving_elevators NElev '2' Id '0'
23 May 2018 22:48:41 [40:9] TRACE Manage people
23 May 2018 22:48:41 [40:9] INFO Going out people '[1]' from elev '0'
23 May 2018 22:48:41 [40:9] TRACE Trace getting_people_in '2'
23 May 2018 22:48:41 [40:9] TRACE Trace getting_people_in '1'
23 May 2018 22:48:41 [40:9] INFO Coming people '[]' to elev '0'
23 May 2018 22:48:41 [40:9] TRACE moving_elevators NElev '2' Id '1'
23 May 2018 22:48:41 [40:9] TRACE Finish step
23 May 2018 22:48:41 [40:null] TRACE Finith modulation
23 May 2018 22:48:41 [40:null] TRACE Show statistics
23 May 2018 22:48:41 [40:null] INFO People status results: '[3,3]'
23 May 2018 22:48:41 [40:null] INFO People waiting results: '[1,1]'
```

Часть результата журналирования, в котором отражен обход возможных вариантов обработки запроса:

```
24 May 2018 06:49:49 [71:null] TRACE Start modulation
24 May 2018 06:49:49 [71:null] WARNING Var 'Step' has null value
24 May 2018 06:49:49 [71:null] INFO Set first step '0'
24 May 2018 06:49:49 [71:0] TRACE Start step
24 May 2018 06:49:49 [71:0] TRACE Finish step
24 May 2018 06:49:49 [71:1] TRACE Start step
24 May 2018 06:49:49 [71:1] TRACE Finish step
24 May 2018 06:49:49 [71:2] TRACE Start step
24 May 2018 06:49:49 [71:2] INFO A man appears with id '0' on floor
   with id '1'
24 May 2018 06:49:49 [71:2] TRACE Putting floor to map
24 May 2018 06:49:49 [71:2] TRACE Copy var 'step' into 'r_0_step' with
   val '2'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'people_targets' into '
   r_0_people_targets' with val '[4,4]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'people_floors' into '
   r_0_people_floors' with val '[1,0]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'people_waiting' into '
   r_0_people_waiting' with val '[0,0]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'people_states' into '
   r_0_people_states' with val '[1,0]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elevators_floors' into '
   r_0_elevators_floors' with val '[0,1]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elev_rmap_1' into '
   r_0_elev_rmap_1' with val '[]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elev_people_1' into '
   r_0_elev_people_1' with val '[]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elev_rmap_0' into '
   r_0_elev_rmap_0' with val '[]'
24 May 2018 06:49:49 [71:2] TRACE Copy var 'elev_people_0' into '
   r_0_elev_people_0' with val '[]'
24 May 2018 06:49:49 [71:2] [r_0_:2] TRACE moving_elevators NElev '2'
   Id '0'
24 May 2018 06:49:49 [71:2] [r_0_:2] TRACE Change ElevPos '0' Id '0'
24 May 2018 06:49:49 [71:2] [r_0_:2] TRACE moving_elevators NElev '2'
   Id '1'
24 May 2018 06:49:49 [71:2] [r_0_:3] TRACE Start step
24 May 2018 06:49:49 [71:2] [r_0_:3] TRACE Do loop step
24 May 2018 06:49:49 [71:2] [r_0_:3] TRACE moving_elevators NElev '2'
   Id '0'
24 May 2018 06:49:49 [71:2] [r_0_:3] TRACE Manage people
```

```
24 May 2018 06:49:49 [71:2] [r_0_:3] INFO Going out people '[]' from
   elev '0'
24 May 2018 06:49:49 [71:2] [r_0_:3] TRACE Trace getting_people_in '2'
24 May 2018 06:49:49 [71:2] [r_0_:3] TRACE Trace getting_people_in '1'
24 May 2018 06:49:49 [71:2] [r_0_:3] INFO Coming people '[0]' to elev
   '0'
24 May 2018 06:49:49 [71:2] [r_0_:3] TRACE moving_elevators NElev '2'
   Id '1'
24 May 2018 06:49:49 [71:2] [r_0_:3] TRACE Finish step
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Start step
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Do loop step
24 May 2018 06:49:49 [71:2] [r_0_:4] INFO A man appears with id '1' on
   floor with id '0'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Putting floor to map
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'step' into '
   r_0_0_step' with val '4'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'people_targets'
   into 'r_0_0_people_targets' with val '[4,4]'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'people_floors'
   into 'r_0_0_people_floors' with val '[1,0]'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'people_waiting'
   into 'r_0_0_people_waiting' with val '[1,0]'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'people_states'
   into 'r_0_0_people_states' with val '[2,1]'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'elevators_floors'
   into 'r_0_0_elevators_floors' with val '[1,1]'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'elev_rmap_1' into
   'r_0_0_elev_rmap_1' with val '[]'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'elev_people_1'
   into 'r_0_0_elev_people_1' with val '[]'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'elev_rmap_0' into
   'r_0_0_elev_rmap_0' with val '[-1,4,-1]'
24 May 2018 06:49:49 [71:2] [r_0_:4] TRACE Copy var 'elev_people_0'
   into 'r_0_0_elev_people_0' with val '[0]'
24 May 2018 06:49:49 [71:2] [r_0_0_:4] TRACE moving_elevators NElev '2'
    Id '0'
24 May 2018 06:49:49 [71:2] [r_0_0_:4] TRACE Elevator '0' is closing
   the door
24 May 2018 06:49:49 [71:2] [r_0_0_:4] TRACE Change ElevPos '1' Id '0'
24 May 2018 06:49:49 [71:2] [r_0_0_:4] TRACE moving_elevators NElev '2'
    Id '1'
```