Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный технический университет
имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ »

КАФЕДРА «ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ БАКАЛАВРА НА ТЕМУ:

«Составление расписания занятий в вузе с помощью SMT-солвера»

по направлению подготовки 01.03.02 Прикладная математика и информатика,
профиль – Анализ, порождение и преобразование программного кода

Студент группы ИУ9-82 _____ А. И. Карькин
(Подпись, дата) (И.О.Фамилия)

Руководитель выпускной квалификационной работы _____ С. Ю. Скоробогатов
(Подпись, дата) (И.О.Фамилия)

Нормоконтролер _____
(Подпись, дата) (И.О.Фамилия)

Москва, 2019

АННОТАЦИЯ

Тема данной ВКР: «Составление расписания с помощью SMT-солвера».

Объем выпускной квалификационной работы составляет 53 страницы. В работе приведено 4 рисунка, 8 листингов с исходным кодом и 3 приложения. В ходе написания работы, использовалось 14 источников, каждый из которых приводится в разделе «Список использованных источников».

Цели данной работы: реализовать программу выполняющую составление расписания с помощью SMT-солвера, оценить эффективность выбранного подхода.

Для достижения поставленных целей была изучен методика составления SMT-задач, были рассмотрены и проанализированы современные SMT-солверы, из числа которых был выбран наиболее эффективный. С помощью выбранного SMT-солвера была реализована программа, выполняющая поставленную задачу.

В разделе 1 приводится постановка задач, необходимых для выполнения цели работы.

В разделе 2 обсуждается вопрос проектирования базы данных, необходимой для хранения вспомогательной информации, которая требуется для составления расписания.

В разделе 3 обсуждается вопрос непосредственно формирования расписания с помощью SMT-солвера

В разделах 4, 5 приводится описание реализации программного решения и интерфейса разработанных приложений

И наконец, данная работа завершается разделом 6, где описывается тестирование приложений и описываются полученные результаты.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

Большая часть данной работы посвящена обсуждению вопросов связанных с задачей выполнимости формул в теориях (satisfiability modulo theories — SMT). Для определения данного тремина используется ряд классических понятий из математической логики. Данные понятия можно найти в работах [1], [2], [3].

В источнике [4] рассказывается про концепцию SMT и приводится теоретические основы для построения собственного SMT-решателя. Там же приводится ряд понятий, которые используются в рассматриваемой предметной области.

Область исследования выполнимости формул в одной или нескольких теориях называется *задачей выполнимости формул в теориях*.

SMT-формула — это формула записанная на языке логики первого порядка, в которой некоторые функции и предикатные символы могут иметь интерпретацию в той или иной теории.

Процедуры, выполняющие решение SMT-задач (проверка формул на выполнимость, либо их опровержение, предоставление модели для данной задачи, обоснование невыполнимости формул, упрощение формул и т. д.) называют *SMT-решателями*.

Также в тексте данной работы будут встречаться такие понятия, как интерпретация и модель, поэтому следует прояснить значение данных терминов.

Под *интерпретаций* подразумевают некоторое соответствие между формальной записью формул и их значением (или *семантикой*) символов данной формулы в одной или нескольких теориях. Сами по себе формулы без интерпретации представляют лишь набор символов. В тот момент, когда формуле дается интерпретация, она приобретает содержательный смысл. Тогда о формулах в языке первого порядка можно сказать, что она является истинной или ложной в данной интерпретации. Если существует такая интерпретация, в

которой все рассматриваемые формулы истины, то такая интерпретация называется *моделью* [3].

В работе [2] вводится такое понятие, как сорт. Под *сортом* подразумевается некоторое множество значений, которые могут принимать константы и предметные переменные в языках первого порядка. Аналогом сорта, является тип данных в языках программирования. Данные понятия занимают особую роль при решении SMT-задач, поскольку в них фигурируют символы из разных теорий и с помощью сортов, можно отделять символы одной теории от другой (например, сорт целых чисел в арифметике и сорт массивов в теории массивов).

Наряду с SMT-задачами и SMT-решателями вводятся такие понятия, как SAT-задачи и SAT-решатели (набор процедур, выполняющих решение SAT-задачи). В отличии от SMT, где используются языки первого порядка в *SAT-задачах* (от английского сокращения sat. – satisfiability – выполнимость) для описания формул используется язык пропозициональной логики. Сама SAT-задача, заключается в поиске таких значений булевых переменных в некоторой формуле, при которых формул принимает истинное значение. В отличии от языков высших порядков, в языке пропозициональной логики существует лишь две константы, ИСТИНА и ЛОЖЬ, и переменные (их называют пропозициональными) могут принимать лишь два значения. В пропозициональной логике также отсутствуют кванторы.

Забегая вперед, можно сказать, что методики решения SAT-задач, используются при решении SMT-задач. Зачастую, последние сводятся к более простым SAT-задачам, решение которых менее затратно по числу операций.

ВВЕДЕНИЕ

При составлении расписания занятий университета важно учитывать ряд ограничений для того, чтобы составленное расписание не приводило к парадоксальным ситуациям. К примеру, при неправильно составленном расписании, может оказаться так, что преподаватель ведет два занятия одновременно в разных аудиториях, или, например, аудитория в которой должно проходить занятие для данной группы в некоторый момент времени уже занята другой группой.

В большинстве вузах, расписание составляется людьми. К сожалению, последние могут ошибаться, в результате чего, в начале семестра случаются сбои при проведении занятий, а составленное расписание приходится корректировать и при том в ряде случаях не по одном разу. Чтобы избежать такого рода проблемы, было бы хорошо иметь программу для ЭВМ, которая будет составлять расписание быстрее человека и к тому же, без логических противоречий. В данной работе рассматривается один из вариантов написания такой программы с помощью SMT-решателя.

При выполнении данной работы, были поставлены следующие цели:

– поиск наиболее подходящего SMT-решателя для поставленной задачи;

– разработка программы для составления расписания вуза с помощью выбранного решателя;

– оценка эффективности полученного решения и применимости его на практике.

Также следует отметить, что расписание занятий в вузе для некоторой группы должно удовлетворять учебному плану специализации группы. В учебном плане, указывается число часов, отводимое на конкретный вид занятия по данному предмету в течении одного или нескольких семестров. Учебный план определяет сколько должно быть занятий того или иного типа по определенному предмету для заданной специализации и кафедры.

1 Постановка задачи

Для составления расписания понадобиться ряд вспомогательных данных:

1) информация о структуре университета: факультеты и кафедры, подразделения учебного заведения;

2) список специальностей и специализаций вуза, а также информацию о том, по каким направлением ведет подготовку та или иная кафедра;

3) учебные планы для специализации, которые составляются какой-либо кафедрой (учебный планы по одной и той же специализации, составленной одной кафедрой, может отличаться от такового для другой кафедры);

4) список преподавателей, а также информацию о том, какие занятия ведет преподаватель на той или иной кафедре;

5) информацию об аудиториях в вузе

и т. д.

Данные для составления расписания будем хранить в базе данных. Модель данных вуза легко представить реляционной моделью. Также данные должны храниться в согласованном непротиворечивом виде. Поэтому в качестве базы данных была выбрана реляционная база.

В качестве исходных данных для тестирования приложения мы будем использовать информацию предоставленною на официальном сайте МГТУ им. Н. Э. Баумана, а также данные предоставленные на портале «Электронный университет» МГТУ им. Н. Э. Баумана.

Данные представленные в вышеописанных источниках необходимо привести к удобному для реляционной модели виду, после чего сохранить эти данные в базу.

При проектировании базы, также следует учитывать ограничения, накладываемые на таблицы, представляющие сущности рассматриваемой предметной области (примеры таких ограничений, приведены в разделе «Введение»). Данное требование позволит избежать ряд ошибок при реализации программы для генерации расписания — ограничения

накладываемые на таблицы спроектированной базы данных не дадут возможности сохранить некорректно составленное расписание в базу.

После проектирование базы и её заполнения тестовыми данными, можно переходить к реализации той части, разрабатываемой программы, которая отвечает непосредственно за составление расписания. Для этого, первым делом, следует выбрать подходящий для эффективного решения поставленной задачи SMT-решателя. Затем, необходимо описать все ограничения, накладываемые на расписание на входном языке SMT-решателя. После этого, для заданных входных параметров (расписание обычно составляется для групп, схожих специальностей, поэтому будет естественно, передавать на вход генератору расписания список шифров учебных групп) из базы загружаются сущности, необходимые для составления расписания. Для загруженных сущностей, также описываются ограничения, накладываемое на конкретное расписание.

Также было бы полезным, отображать составленное расписание в удобном для восприятия пользователей виде.

С учетом вышесказанного, для реализации поставленной задачи, понадобиться выполнить следующие этапы:

1) Проектирование базы данных
2) Подготовка тестовых данных для составления расписания.
3) Заполнение базы тестовыми данными.
4) Выбор наиболее подходящего SMT-решателя.
5) Описание ограничений, накладываемых на расписание, на целевом языке SMT-решателя.
6) Создание инструмента для просмотра составляемых расписаний.

2 Проектирование базы данных

Данный раздел посвящен проектированию базы данных, описанной выше.

В первой части исходного раздела приведена концептуальная схема предметной области.

Во второй части, описывается реляционная модель данных.

В третьей — описываются ограничения, которые накладываются на таблицы спроектированной базы, а также говорится о том, каким способом можно добиться выполнение того или иного ограничения.

2.1 Построение модели типа сущность-связь

Перечислим основные сущности, которые будут фигурировать в нашей базе (названия сущностей выделены курсивом и начинаются с большой буквы). Расписание обычно формируется для *Группы*. Группа, обычно занимается по некоторому *Учебному Плану*, который определяется тремя параметрами: *Кафедрой*, годом начала обучения, *Специализацией*. У кафедры может быть несколько специализаций. В то же время программы одной и той же специализации могут преподаваться на разных кафедрах, то есть между *Кафедрой* и *Специализацией* отношение «многие-ко-многим». *Кафедра* может принадлежать какому-либо *Факультету*, при том только одному. На кафедре может быть специализации для разных *Квалификаций* (бакалавриат, специалитет, магистратура, аспирантура). При этом каждая *Специализация* относится только к одной *Квалификации*. Учебный процесс разбит на *Семестры*. Расписание обычно не меняется на протяжении семестра, поэтому расписание *Учебной Группы* также определяется *Семестром*. Расписание каждой *Группы* состоит из *Учебных Дней,* которые определяются *Днем Недели*.

Учебный День состоит из *Занятий. Занятие* определяется *Временем Пары* (её номер, время начала и окончания)*, Аудиторией, Предметом*. Занятие может быть нескольких типов: лабораторная, семинар, лекция и др, поэтому имеет смысл выделить такую сущность, как *Тип Занятия*. Занятие может вести как один преподаватель так и несколько. Другими словами, между *Занятием* и *Преподавателем* есть связь «многие-ко-многим». Для проверки корректности расписания, необходимо сверять его с учебным планом. Поэтому следует выделить сущности для его описания. Сущность *Учебный План* определяется учебным годом, *Кафедрой* и *Специализацией*. Учебный план представляет из себя таблицу, в которой каждой паре из семестра и предмета соответствует ячейка с перечислением *Учебных Часов* по каждому типу занятий. То есть сущность *Учебные Часы* для данного семестра и данного предмета состоит из *Типа Занятия* и часов, выделенных на данный тип занятий.

Описанные сущности и связи между ними можно представить с помощью диаграммы ER-модели в нотации crows-foot, которая приведена в приложении А.

2.2 Построение реляционной модели

ER-модель, построенная в предыдущем разделе, была преобразована в реляционную модель. Схема реляционной модели приведена в приложении Б.

Для нормализации реляционной модели, были введены дополнительные отношения (таблицы). Скажем несколько слов об этих отношениях.

Для осуществления связи типа «многие-ко-многим» между кафедрой (табл. `department`) и специализацией (табл. `specialization`) была введена дополнительная таблица `department_specliazation`. Данная таблица имеет в качестве дочерней таблицу учебных планов (табл. `study_plan`), каждая строка которой может различаться в зависимости от года поступления (в то же время, у `department_specialization` может быть несколько учебных планов одного и того же года).

Для хранения информации о преподавателях была введена таблица tutor, которая содержит имя преподавателя, его электронную почту и ученую степень. Преподаватель может вести занятия по определенным предметам на определенных кафедрах. Все возможные предметы, перечислены в таблице subject. В таблице department_subject хранится информация о том, какие предметы ведет та или иная кафедра, то есть данная таблица описывает отношение между кафедрой и предметом типа «многие-ко-многим». Наконец, для для того, чтобы можно было сказать какой именно предмет ведет преподаватель на той или иной кафедре была создана таблица tutor_subject, которая включает в себя два внешних ключа: tutor_id (ссылка на идентификатор преподавателя в таблице tutor) и subject_id (идентификатор предмета в таблице subject).

Учебный план состоит из списка строк (табл. study_plan_item), каждая строка соответствует определенному предмету на той или иной кафедре, поэтому study_plan_item имеет два внешних ключа: study_plan_id и department_subject_id. Строка состоит из ячеек (табл. study_plan_item_cell). Каждая ячейка определяется родительской строкой (табл. study_plan_item) и семестром (табл. Term, хранит список целых чисел, соответствующих номерам семестра). В каждой ячейке записывается требуемое число часов для каждого типа занятия (табл. hours_per_class). Тип занятия хранится в таблице class_type.

Само расписание описывается следующими таблицами. Каждый элемент расписания для данной группы состоит из дней (табл. schedule_day), которые определяются днем недели (табл. day_of_weak). Дни в свою очередь состоят из списка слотов под конкретное занятие (табл. schedule_item), каждый из которых определяется временем пары (табл. class_time). Каждый элемент в расписании может быть трех типов: числитель, знаменатель, либо и то и другое. Поэтому таблица schedule_item имеет в качестве дочерней таблицы schedule_item_parity, которая содержит текстовое поле, day_parity,

принимающее одно из трёх значений: «ЧС» — числитель, «ЗН» — знаменатель, «ЧС/ЗН» — и числитель, и знаменатель. После того, как мы определили четность дня недели занятия, можно понять в какой аудитории оно находится, его тип (семинар/лекция), предмет и какие преподаватели его ведут. Эти данные описываются с помощью внешних ключей: `classroom_id`, `class_type_id`, `tutor_subject_id`, которые ссылаются на соответствующие таблицы.

2.3 Ограничения, накладываемые на таблицы базы данных

Ниже приведены ограничения, которые накладываются на таблицы проектируемой базы данных. К каждому ограничению приводится краткое описание его реализации на уровне базы данных.

1) Никакая строка из таблицы `study_plan_item_cell` не может содержать повторяющуюся пару, состоящую из идентификатора семестра и внешнего ключа, ссылающегося на таблицу `study_plan_item`.

Реализация: Выполнение данного ограничение можно обеспечит с помощью ограничения `UNIQUE` на таблицу `study_plan_item_cell` для столбцов: `study_plan_item_id` и `term_id`.

2) Название типа занятия должно быть уникально.

Реализация: Добавим ограничение `UNIQUE` на столбец `type_name` для таблицы `class_type`.

3) Номер группы не может дублироваться в рамках одного учебного плана.

Реализация: Добавим ограничение `UNIQUE` на `study_plan_id`, `group_number` для таблицы `study_group`

4) Время начала и конца пар не может пересекаться, при этом время начала должно быть меньше времени конца пары.

Реализация: первое ограничение можно проверить написав триггер на обновление таблицы `class_time`; второе условие можно проверить с помощью ограничения `CHECK (starts_at < ends_at)`.

5) Для каждого дня в расписании не может быть двух занятий с одинаковым временем начала и конца.

Реализация: ограничение `UNIQUE` на `schedule_day_id` и `class_time_id` для таблицы `schedule_item`.

6) Две группы, которые занимаются по одному учебному плану не могут заниматься по одному и тому же предмету в одно и тоже время, если только тип исходного занятие не является лекцией.

Реализация: При обновлении таблицы `schedule_item_parity` сравнивать время (оно формируется из колонок из нескольких таблиц: `schedule_day.title`, `schedule_item.class_time`, `schedule_item_parity .day_parity`) и тип занятия данной группы (`scedhule_item_parity.class_type_id`) с другими группами. Если время и тип совпадают, но различаются шифры групп и различаются их учебные планы, либо совпадают учебные планы, но тип занятия не является лекцией, то откатывать транзакцию и информировать пользователя об ошибке.

7) Один и тот же преподаватель не может вести два различных занятия в одно и то же время

Реализация: Напишем триггер, который будет проверять данное условие при обновлении таблицы `schedule_item_parity`: если при обновлении/вставки записей, существуют записи с одинаковыми значениями колонок `lecturer_id` и `classtime_id`, то откатываем транзакцию и сообщаем об ошибке.

8) Для поля `schedule_item_parity.day_parity` должны выполнятся следующие проверки: во-первых, данное поле может принимать только значения: «ЧС» — числитель, «ЗН» — знаменатель, «ЧС/ЗН» — числитель и знаменатель; во-вторых у таблицы `schedule_item_parity` не может быть двух дочерних записей с одинаковым значение поля `day_parity`.

Реализация: Первое условие можно обеспечить добавив ограничение `CHECK` на столбец `day_parity`. Второе ограничение можно проверять с помощью триггера на обновление таблицы `schedule_item`.

2.4 Выбор СУБД

После того, как спроектирована реляционная модель и описаны ограничения, накладываемые на таблицу, можно переходить к физическому созданию базы.

В качестве реляционной системы управление базами данных (РСУБД) был выбран Postgre SQL. Выбор данной РСУБД обусловлен следующими обстоятельствами. PostgreSQL имеет широкий спектр возможностей для описания разного рода ограничений, обеспечения целостности данных (что играет большею роль при решении поставленной задачи). Разработчики PostgreSQL стремятся соответствовать стандарту ANSI-SQL:2008, данная РСУБД отвечает требованиям ACID (Atomicity, Consistency, Isolation, Durability), поддерживает множество встроенных типов, в том числе пользовательские. Так что функциональности PostgreSQL должно быть более чем достаточно для реализации поставленной задачи.

3 Составление расписания

Как правило расписание составляется для множества групп схожих специальностей. При этом некоторые занятия (как правило, лекции) в расписании групп могут быть общими, если группы имеют идентичные программы по данному предмету в рамках текущего семестра. Однако, совпадение учебных программ по некоторому предмету зачастую трудно определить, из-за отсутствия таковых в открытом доступе. Кроме того, учебные программы представляются в виде текста на естественном языке, поэтому для их автоматического анализа потребуется использовать методики анализа текста на естественном языке, что существенно усложняет поставленную задачу.

Поэтому введем, некоторое упрощение и будем выполнять генерацию расписания только для групп с одинаковым учебным планом. Как следует из раздела 2, такие группы в исходной модели данных принадлежат одной и той же кафедре, имеют одинаковую специализацию и год начала обучения.

Вся информация, необходимая для генерации расписания для заданного списка групп будет хранится в базе данных.

Стало быть, для составления расписания нам лишь потребуется задать список групп, обучающихся по одному и тому же учебному плану, или же указать сам учебный план и номер семестра, если требуется составить расписание для всех групп в рамках одного семестра.

С учетом сказанного, расписание можно составлять по следующей схеме. Сначала для заданных групп, из базы данных необходимо получить следующие данные:

а) учебные планы групп (количество часов, выделенных на каждый предмет в данном семестре);

б) список преподавателей, которые ведут определенные занятия в текущем семестре;

в) список аудиторий для проведения занятий;

г) список предметов в данном семестре;

д) ряд дополнительной справочной информации: дни недели, время начала пар, типы занятий и прочее.

После того, как исходные данные получены, их необходимо представить в форме, понятной для SMT-решателя. Для того чтобы расписание было корректным, необходимо описать ряд ограничений в виде формул на входном языке SMT-решателя (о том о каких именно ограничениях идет речь, будет сказано в разделе 3.1). Далее, с помощью SMT-решателя необходимо проверить непротиворечивость этих формул в рамках предоставленных данных. В том случае, если оказалось, что не существует интерпретации, в которой все описанные формулы принимают истинное значений (то есть при которой выполняются все ограничения), необходимо сообщить пользователю об

ошибке. В противном случае следует запросить у SMT-решателя такую интерпретацию или, другими словами, построить модель. В рамках полученной модели необходимо сформировать множество выражений, отвечающих каждой строке расписания заданных групп для каждого дня недели. И наконец, полученные константы преобразовать в модель данных базы, после чего сохранить их в базу.

3.1 Описание ограничений при составлении модели расписания

Для того, чтобы расписание составлялось корректно, SMT-решателю необходимо задать ряд ограничений в виде формул. Как уже, говорилось ранее, большинство SMT-решателей в той или иной форме «понимает» задачи описанные на языке формул первого порядка в той или иной теории.

Для составления ограничений нам потребуются следующие теории: теории целых и вещественных чисел, теории равенств и неравенств, линейная арифметика, теория индуктивных типов данных. Краткое описание данных теорий можно найти в работе [4].

3.1.1 Обозначения

В данном разделе при описании ограничений, будут использоваться специальные обозначения для сортов, переменных констант, определения индуктивных типов данных.

Множество сортов договоримся обозначать словами и словосочетаниями записанными латинскими буквами и будем выделять курсивом. Каждое слово в названии сорта должно начинаться с заглавной буквы. Если название состоит из нескольких слов, то остальные слова в названии также будут начинаться с заглавной буквы. Примеры обозначений: *Lesson*, *LessonItem*.

Константы также будут обозначатся латинскими символами и выделятся курсивом. Константы начинаются с маленькой буквы и выделяются жирным

шрифтом. В названии константы также могут встречаться индексы Примеры: ***a***, ***b***, ***c***, ***lec***, ***sem,*** $\boldsymbol{x_1}$, $\boldsymbol{x_2}$.

Для обозначения констант, будет использоваться нотация аналогичная константам, но при этом символы в названии констант не будут выделятся курсивом. Примеры обозначений: *a, b, subject*, x_1, x_2.

Поскольку сорты — это по сути дела, множества, для обозначения факта принадлежности переменной или константы тому или иному сорту, можно также использовать символ «∈». Например, если *x* принадлежит сорту *Sort,* то будем записывать: $x \in Sort$.

В теории индуктивных типов данных для определения типов данных используется такое понятие, как *конструктор* [5]. Конструктор описывает поля, которые содержит тот или иной тип данных. Каждый тип данных может состоять из одного и более конструктора, при этом конструктор может быть пустым. Определение конструктора состоит из его названия и перечисления переменных (полей) того или иного сорта, которые задают тип (этот список может быть и пустым, в этом случае обозначение списка будем опускать). *Перечисление полей* одного конструктора для того или иного типа будем обозначать следующим образом: $(x_1 SortName_1, x_2 SortName_2, \ldots, x_N SortName_N)$, где x_i – переменная, а $SortName_i$ – название сорта, к которому принадлежит переменная x_i. *Название конструктора* будет обозначатся словами, записанными латинскими буквами, начинающимися с маленькой буквой и будут выделятся жирным текстом (также, как и константы). Если у типа данных имеется более одного конструктора, то конструкторы друг от друга будем отделять с помощью символа «|». Название типа данных от его определения буем отделять символом «=». Таким образом, для того, чтобы объявить тип данных, будет использоваться нотация вида:

$$
\begin{aligned}
DataTypeName = \quad & \boldsymbol{cons_1}(x_1 SortName_1, x_2 SortName_2, \ldots) \\
& | \ \boldsymbol{cons_2}(y_1 AnotherSortName_1, y_2 AnotherSortName_2, \ldots) \\
& | \ \boldsymbol{empty} \\
& | \ \ldots
\end{aligned}
$$

У конструктора может задаваться предикатная функция или *тестер*, который дает возможность установить тот факт, что элемент рассматриваемого типа был создан именно с помощью данного конструктора [5]. Для обозначения тестера будем использовать приставку *is-*, а в скобках будем указывать рассматриваемый элемент. То есть если элемент *x* типа *DataType* был создан с помощью конструктора $cons_1$, то будем записывать: $is\text{-}cons_1(x)$. Также для конструкторов без определения (констант) будет естественно записывать факт принадлежности через знак «=». Например, если *x* соответствует конструктору ***empty***, будем записывать: $x = \mathbf{empty}$.

Конструктор также может содержать *дистуктор* [5], который позволяет получить доступ к одному или множеству полей, который содержится в определении конструктора. Для обозначения дистркутора для некоторой переменной будем ставить символ «.» после переменной и записывать название поля. Так, например, если переменная *x* типа *DataType* была создана с помощью конструктора вида $cons(rec_1 Sort_{1,} rec_2 Sort_2)$, то для доступа к полю rec_1 будем пользоваться записью: $x.rec_1$.

Индуктивные типы данные с приведенной нотацией позволяют также записать перечисления, как список конструкторов без определений (пустых конструкторов), которые можно рассматривать как константы. Например, мы можем объявить следующее перечисление:

$$Kind = \mathbf{lec} \mid \mathbf{sem} \mid \mathbf{lab}$$

В теориях, которые мы будем использовать вводятся ряд символов и сортов, которые будут обозначатся следующим образом:

– сорт целых чисел: *Int*

– сорт вещественных чисел: *Real*

– сорт символов (переменных и констант) из булевой алгебры: *Bool*

– константы из сорта целых чисел обозначаются с помощью цифр, в записи которых может содержатся знак «+» или «-», например, **-1**, **0**, **1**, **2**, …, **15**, …

– константы из сорта вещественных чисел будем записывать в десятичном виде или в форме дроби, например, **1.54**, $\frac{1}{2}$.

– две константы из сорта *Bool*: **true** – истина, **false** – ложь**.**

3.1.2 Структуры для описания ограничений

Для описания ограничений для расписания, нам понадобиться ввести структуры приведенные ниже. Каждая структура представляет ту или иную сущность рассматриваемой предметной области.

– Тип занятия (*Kind*) — перечисление вида $Kind = \textbf{\textit{lec}} \mid \textbf{\textit{sem}} \mid \textbf{\textit{lab}}$, где константы обозначают следующее: ***lec*** – лекция, ***lab*** – лабораторная работа, ***sem*** – семинар.

– Преподаватель (*Tutor*) — индуктивный тип данных, каждый представитель которого определяется некоторым целочисленным уникальным идентификатором: $Turor = \textbf{\textit{tutor}}(tutorid\ Int)$.

– Аудитория (*Room*) — определяется аналогично *Tutor*:

$Room = \textbf{\textit{room}}(roomid\ Int)$.

– Предмет (*Subject*) — аналогично предыдущему:

$Subject = \textbf{\textit{subject}}(subjectid\ Int)$.

– Учебная группа (*Group*) — аналогично предыдущему:

$Group = \textbf{\textit{group}}(groupid\ Int)$.

– Занятие (*Lesson*) — может быть пустым и непустым. Во втором случае, занятие определяется предметом, типом занятия, преподавателем, который ведет данный предмет и аудиторией. Определить занятие можно так: $Lesson = \textbf{\textit{blank}} \mid \textbf{\textit{lesson}}(subject\ Subject, kind\ Kind, tutor\ Tutor, room\ Room)$, где ***blank*** — конструктор пустого урока, ***lesson*** — конструктор непустого.

– Строка под занятие (*LessonItem*) — в случае, если данное занятие проводится и по числителям и по знаменателям, строка в расписании содержит только один урок, если занятия по числителям и по знаменателям различаются

то строка содержит пару занятий. Для описании строки используется тип данных, который определяется так:

$$LessonItem = \textbf{lesson}(lesson\ Lesson) \mid \textbf{lessonpair}(numerator\ Lesson, denominator\ Lesson).$$

– Слот под строку с занятием (*Slot*) — «искусственный» сорт, предназначенный для обозначения порядка строк с занятиями. Данный тип, представляет из себя перечисление: $Slot = \textbf{a} \mid \textbf{b} \mid \textbf{c} \mid \textbf{d} \mid \textbf{e} \mid \textbf{f} \mid \textbf{g}$.

Конструкция, описывающая расписание занятий для списка групп схожей специальности будет формироваться следующим образом. Для каждой группы (элемента сорта *Group*) и каждого дня недели (элемента сорта *Day*) будет составляется список строк с занятиями (элементов сорта *LessonItem*). Каждая строчка ставится в соответствие некоторому слоту (элементу сорта *Slot*). Или же, другими словами, расписание представляет из себя, отображение ϕ, которое каждой тройке, состоящей из группы, дня недели и слота, ставит в соответствие элемент сорта *LessonItem*, который может содержать либо одно занятие, либо пару из занятий:

$$\phi: (group, day, slot) \rightarrow item,$$

где $group \in Group$, $day \in Day$, $slot \in Slot$ и $item \in LessonItem$.

Если занятие отсутствует в строке с расписанием, то на его месте ставится пустой урок (***blank***).

3.1.3 Описание ограничений

Ниже приведено словесное описания ограничений, которые накладываются на составляемое расписание, с использованием нотации, описанной ранее, в подразделе 3.1.1. (Более детальное описание с использованием структур из конкретного SMT-решателя приведено на языке Java в папке с исходным кодом, которая расположена на диске, прилагающемся к данной работе и более подробно описывается в разделе ?).

1) Количество занятий для одного дня не должно быть сильно большим и не должно быть меньше двух. Оптимальным будет, если число занятий для одного дня будет варьироваться от двух до четырех.

2) Занятия могут начинаться с любой допустимой пары, но при этом между занятиями не должно быть пропусков или, другими словами, должны отсутствовать пустые занятия между непустыми (не должно быть «окон» в расписании).

3) Пара занятий $l_1, l_2 \in Lesson$ для некоторых групп $g_1, g_2 \in Group$ (договоримся, что здесь и далее, все рассматриваемые группы будут заниматься по одному учебному плану) и для некоторого фиксированного дня недели может находится в одному слоте и в занимать одну и ту же позицию в строчке с расписанием, если строка состоит из пары занятий только в с следующих случаях:

а) оба занятия являются пустыми, $l_1 = \textbf{blank} \wedge l_2 = \textbf{blank}$.

б) одно из занятий пусто, другое — нет , и тип непустого занятия не является лекцией (лекции для групп схожих специальностей обычно объединяют для экономии времени преподавателей), то есть без ограничения общности: $l_1 = \textbf{blank} \wedge l_2 \neq \textbf{blank} \wedge l_2.kind \neq \textbf{lec}$

в) оба занятия не являются пустыми, у занятий совпадают: предмет, аудитория и преподаватель, а тип каждого из занятий является лекцией,

$$l_1 \neq \textbf{blank} \ \wedge \ l_2 \neq \textbf{blank} \ \wedge \ l_1.subject = l_2.subject \ \wedge \ l_1.tutor = l_2.tutor$$
$$\wedge \ l_1.room = l_2.room \ \wedge \ l_1.kind = l_2.kind \ \wedge \ l_1.kind = \textbf{lec}$$

г) оба занятия не являются пустыми, тип каждого из занятий не является лекцией, преподаватели, которые ведут занятия не совпадают и не совпадают номера аудиторий,

$$l_1 \neq \textbf{blank} \ \wedge \ l_2 \neq \textbf{blank} \ \wedge \ l_1.kind \neq \textbf{lec} \ \wedge \ l_2.kind \neq \textbf{lec} \ \wedge$$
$$\wedge \ l_1.tutor \neq l_2.tutor \wedge \ l_1.room \neq l_2.room$$

4) На основании ограничений пункта 3, можно записать ограничение, определяющее при каком условии пара групп $g_1, g_2 \in Group$ для заданного дня $d \in Day$ могут занимать один и тот же слот $s \in Slot$. Пусть $i_1 = \phi(g_1, d, s)$, $i_2 = \phi(g_2, d, s)$ — пара элементов сорта *LessonItem*. Данные элементы будут корректны, при следующем условии:

$$is\text{-}lesson(i_1) \wedge is\text{-}lesson(i_2)$$

и $i_1.lesson$, $i_2 lesson$ являются корректным в соответствии с ограничениями п. 3 или же, если:

$$is\text{-}lessonpair(i_1) \wedge is\text{-}lessonpair(i_1)$$
$$\wedge\ i_1.numerator, i_2.numerator\ -\ \text{корректны по п. 3)}$$
$$\wedge\ i_1.denominator, i_2.denominator\ -\ \text{корректны по п. 3)}$$

5) Далее, на основании п. 4. определяется корректность расписания для двух групп $g_{1,}g_2 \in Group$ и заданного дня $d \in Day$, которое формулируется следующим образом: для заданного дня и заданных групп перебираем все возможные значения слотов, после чего для каждого слота из перебираемого набора пары групп и дня говорим, что последние должны быть корректны по п. 4.

6) Аналогично п. 5 формулируется ограничение на корректность расписания занятий для двух групп в течение всей недели.

7) На основании ограничения п. 6 можно сформулировать валидность расписания для всех групп на протяжении всей недели, перебрав все возможные пары из всех допустимых групп и потребовать корректность расписания для каждой такой пары в соответствии с п. 6.

Помимо проверки расписание на непротиворечивость, его также следует проверять на соответствие учебному плану. По учебному плану мы можем определить количество занятий каждого типа по каждому из предметов в данном семестре, которые должны приходится на одну неделю. Учебный процесс в университете, обычно длится 17 недель. На каждое занятие в учебном плане отводится по 2 часа. В учебном плане, обычно, указывается число часов *k,* отводимое на каждый предмет каждого типа для одного или нескольких семестров (некоторые предметы могут преподаваться на протяжении нескольких семестров). Обозначим за *l* количество семестров в течении которых преподаются занятия заданного типа по заданному предмету С сказанного, по формуле приведенной ниже мы можем определить сколько занятий *n* заданного типа $k \in Kind$ должно быть по предмету $s \in Subject$ в течении одной недели, чтобы расписание удовлетворяло учебному плану.

$$n = \frac{k}{2 \cdot 17 \cdot l} \quad (1)$$

В формуле (1), используется деление, а значит количество занятий может оказаться дробным. Учебный план составляется таким образом, чтобы число занятий k либо делилось нацело, либо при делении давало дробную часть равную 0.5. Последнее, в свою очередь, означает, что в расписании данный предмет должен присутствовать только по числителям или знаменателям. Поэтому при подсчете предметов в составляемом расписании, мы будем считать занятия, которые ведутся только по знаменателям и только по числителям за 0.5, а занятия, которые проводятся каждую неделю за 1.

Расчет количества предметов заданного типа в расписании будем производить по следующей схеме. Сначала посчитаем общее число занятий в одной строке занятия, потом подсчитаем общее число занятий в течение дня, затем — для недели. Тем самым для заданного предмета *s*, заданного типа занятия *k* мы получим общее число занятий на неделе, которое можно сверить со значением, полученным по формуле (1).

Отметим одну особенность приведенной выше конструкции расписания. Входе описания ограничений мы пользуемся отображением ϕ, для которого дано лишь формальное определение. Однако мы никак явным образом не описываем, по каким правилам строится данное отображение. Поэтому возникает вопрос, как SMT-решатель построит интерпретацию данной функции. Ведь, интерпретация берется из какой-либо теории, а в данном случае не понятно о какой теории идет речь (мы имеем некоторую абстрактное отображение, только и всего).

Некоторые SMT-решатели могут построить интерпретацию такой функции в том случае, если данный решатель поддерживает *пустую теорию* (от англ. empty theory) или теорию не интерпретированных функций [4]. Поэтому при выбора SMT-решателя нужно также учесть данное обстоятельство.

3.2 Выбор SMT-решателя

При выборе SMT-решателя для решения поставленной задачи, необходимо учитывать два обстоятельства. Во-первых, SMT-решатель должен поддерживать ряд теорий, перечисленных в разделе 3.1. Во-вторых, поскольку весь проект изначально писался на языке программирования Java, SMT-решатель должен иметь поддержку данного языка.

Как оказалось, таким критериям удовлетворяет лишь один более или менее качественный SMT-решатель — это Z3 Theorem Prover (далее, просто Z3) от кампании Microsoft.

Согласно источнику [6], Z3 для решения SMT-задач используют следующую схему. В начале исходная задача записывается в виде формул на некотором «высокоуровневом» языке. Данная запись может содержать кванторы, и другие сложные конструкции, взятые из различных теорий первого порядка. Затем, производится упрощения записанных конструкций, основанное на простых алгебраических преобразованиях. После этого, выражения «высокоуровневого языка», транслируются в некоторые промежуточные удобные для последующей обработки структуры данных, и затем, выполняется поиск интерпретации символов записанных на входном языке в одной или нескольких теориях (символы теорий могут пересекаться). При этом в Z3 некоторые символы могут оставаться и без интерпретации. Преобразованная SMT-задача разбивается на SAT-подзадачи. Полученные SAT-задачи решаются с использованием ряда известных алгоритмов, таких, как DPLL [7]. В том случае, когда при решении задачи используется интерпретация из той или иной теории? используется ряд эвристик, специфичных для конкретных теорий, позволяющий отыскать решение наименее затратным способом.

Если абстрагироваться от деталей, можно заметить, что такая схема очень напоминает устройство компилятора высокоуровневого языка, который преобразует исходный язык в некоторый низкоуровневый (в случае SMT-решателя — исходный язык преобразуется в язык пропозициональной логики). Поэтому по аналогии с компиляторами, вводят такое понятия, как фронтенд (от англ. Font-end — клиентская сторона). Фронтенд выполняет перевод целевого языка SMT-решателя в некоторое абстрактное представление. После чего выполняется ряд оптимизаций и преобразований.

Z3 имеет несколько фронтендов, позволяющих записывать задачу либо в текстовом формате (Z3 поддерживает SMT LIB и Simplify), либо средствами бинарного API, написанного на языке C. К низкоуровневому API для Z3 компания Microsoft предоставляет байндинги для большинства современных языков программирования, на основании которых реализованы высокоуровневые API для этих языков. Для реализации генератора расписания нам понадобиться высокоуровневое API для языка Java.

4 Разработка программного решения

С учетом требований сформулированных в разделе 1, нам понадобится реализовать ряд программных утилит, каждая из которых будет выполнять одну из поставленных задач. Все утилиты, которые были реализованы в результате выполнения работы, были написаны на языке Java.

Но прежде, чем приступать к написанию данных утилит, необходимо создать базу данных и ряд таблиц, речь о которых шла в разделе 2.2. По-мимо создания таблиц, необходимо также описать ряд ограничений, сформулированных в разделе 2.3, которые описываются с помощью триггеров и ограничения `UNIQUE`.

SQL-скрипт `init.sql` (скрипт инициализации) выполняющий вышеописанные действия приведен в папке `./sql`, которая располагается на компакт диске, прилагаемому к данной работе.

Далее, созданную базу необходимо заполнить данными. Для выполнение данной задачи была реализована консольная утилита, под название `dbfill`. Данная утилита считывает данные из справочников, которые представляют из себя csv-файлы. Каждый справочник содержит информацию, необходимую для одной или нескольких таблиц. Программа итеративно считывает данные из каждого файла и сохраняет их в ту или иную таблицу.

Часть из этих справочников формировалась вручную, простым копированием таблиц в программу Excel с сайта университета. Другая часть данных была представлена в неудобном для формирования таблиц виде. Такие данные приводились к нужном виду автоматически с помощью программы. Так, например, для получения списка аудиторий и информации о преподавателях, точнее, информации о том, какой преподаватель ведет тот или иной предмет для данной группы, был написан парсер расписания. Данный парсер последовательно скачивал расписания занятий в виде html-страницы для каждой группы со страницы расписания МГТУ им. Н. Э. Бамана. Затем, выполнялся парсинг каждой из страницы, после чего данные сохранялись в csv-файл. За выполнение данной задачи, отвечает модуль `schedule-parser`, расположенный в папке проекта с аналогичным наименованием.

После того как база сформирована и заполнена данными можно переходить составлению расписания. Для выполнения этой задачи была написана утилита `smtgen`, которая для заданного списка групп считывает вспомогательные данные из базы, необходимые для составления расписания. После этого, для полученных данных утилита `smtgen` формирует SMT-задачу для Z3 с помощью выскоуровнего API на Java. Затем, Z3 предпринимает попытку решить данную задачу. Если решение существует, то у Z3 запрашивается модель расписания. Данная модель преобразуется в модель данных и сохраняется в базу.

Также, для представления расписания в понятном для восприятия человеком виде была написана утилита `pdfgen`, которая для заданной группы считывает данные о расписании из базы и формирует pdf-документ, в котором

представлено расписание в виде таблицы (если таковое расписание существует в базе).

Далее, приводится подробное описание, каждой из утилит.

4.1 Реализация модели данных на языке Java

Для упрощения операций доступа к данным, использовалась такая техника, как объектно-реляционное отображение (от англ.: Object-Relational Mapping — ORM) позволяющая связать сущности базы (таблицы, представления отношения между ними) с концепциями объектно-ориентированного программирования (ООП). Проще говоря, ORM — это такая техника, которая позволяет каждой таблице или представлению поставить в соответствие некоторый класс в контексте ООП.

Для языка программирования Java существует множество фреймворков, реализующих данную технику. По ряду причин, из числа прочих был выбран фреймворк под названием Hibernate ORM — фреймворк с открытым исходным кодом разработанный компанией Red Hat. Данный фреймворк позволяет абстрагироваться от той или иной реляционной базы, что позволяет в случае необходимости заменить реляционную базу без особых трудностей. Он также поддерживает кеширование запросов (при частом выполнении одинаковых запросов, запросы сохраняются в оперативную память, что позволяет улучшить производительность), «ленивую» загрузку данных (данные загружаются из базы только в момент, когда они действительно нужны), а также свой абстрактный язык, под названием HQL для написания запросов. Данный язык схож с языком SQL, за тем исключением, что вместо таблиц в запросах указываются классы сущностей, что позволяет реализовывать сложные запросы (например, с соединением таблиц) абстрагируясь от конкретной базы.

Для настройки подключения к базе, в Hibernate необходимо прописать параметры подключения к базе и информацию о классах, которые используется при отображении. Данная информация прописывается в конфигурационном xml

файле, hibernate.cfg.xml, который должен располагаться в директории с ресурсами проекта.

Для данного проекта, этот файл располагается в следующей директории: ./schedule-core/src/main/resources и имеет вид:

Листинг 1 — Файл конфигурации Hibernate: настройки соединения и ORM

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
             "http://www.hibernate.org/dtd/hibernate-configuration-
3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Параметры полкдючения к базе -->
       <property name="connection.driver_class">
        org.postgresql.Driver
    </property>
    <property name="connection.url">
      jdbc:postgresql://localhost:5432/bmstu_schedule
     </property>
    <property name="connection.username">
      admin
    </property>
    <property name="connection.password">
      admin
    </property>

    <!-- Не изменяем модель данных.
     В случе не совпадения ООП модели
     и модели базы данных получим ошибку -->
    <property name="hbm2ddl.auto">validate</property>

    <!-- Перечисление классов, которые
     отображаются на таблицы БД: -->
    <mapping class="ru.bmstu.schedule.entity.Classroom"/>
    <mapping class="ru.bmstu.schedule.entity.DayOfWeek"/>
    <!-- ... -->
  </session-factory>
</hibernate-configuration>
```

После того, как мы настроили подключение к базе, необходимо задать параметры описывающие, каким классам-сущностям ставятся в соответствии те или иные таблицы и каким полям в этих классах, соответствуют те или иные колонки таблиц в базе. В Hibernate существует два способа осуществить данное отображение: а) через xml файл конфигурации, hibernate.cfg.xml; б) с

помощью аннотаций. Мы будем использовать второй способ. В листинге 2 приводится пример того, как описывается данное соответствие способом б).

Листинг 2 — Пример ORM между таблицей `subject` и классом `Subject`

```
@Entity
@Table(name = "subject")
public class Subject {

    @Id
    @Column(name = "subject_id", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Basic
    @Column(name = "subject_name", nullable = false)
    private String name;

    // определения геттеров и сеттеров
    // ...

    // перепоределение методова hashCode и equals
    // ...

}
```

По-мимо описания соответствия между полями класса и колонками таблицы или представления, в Hibernate можно также описывать отношения между сущностями. Для этого у класса-сущности должна быть ссылка или коллекция ссылок на зависимую сущность, а само отношение описывается посредством добавления одной из аннотаций, в зависимости от типа связи между сущностями: а) аннотация `@OneToOne` описывает связь «один-к-одному»; б) `@OneToMany` — «один-ко-многим»; в) `@ManyToOne` — «многие-к-одному» и г) `@ManyToMany` — «многие-ко-многим».

Процесс создания классов-сущностей и описания отношения между сущностями и таблицами базы, в некотором плане, является «механической» процедурой. Поэтому нет смысла приводить этот процесс для каждой таблицы.

Стоит лишь отметить, что названия сущностей соответствует таковым, описанным в разделе 2.1 (все название переведены на английский язык).

Классы-сущности располагаются в папке с проектом, в модуле `schedule-core`, в пакете `ru.bmstu.schedule.entity`.

4.2 Реализация классов для осуществления доступа к данным

В ходе выполнения работы, был реализован ряд классов, выполняющий базовые операции доступа к данным такие, как создание сущностей, получение одной или несколько сущностей по заданному критерию, свойству или ключу, обновление полей сущности, и удаление сущности из базы, то есть таких классов, которые выполняют CRUD операции (от англ. CRUD — Create, Read Update, Delete operations — операции чтения, записи, обновления и удаления). Все такого рода классы, реализованные в рамках данной работы имеют суффикс DAO (от англ. Data Access Object — объект доступа к данным) и наследуются от одного абстрактного обобщенного класса, который называется `HibernateDao` и реализует ряд примитивных методов, на основании которых можно выполнить любую CRUD-операцию. В листинге 3 приводится реализация некоторых методов данного класса.

Листинг 3 — Generic класс, реализующий базовые операции доступа к данным

```
public abstract class HibernateDao
  <PK extends Serializable, E> {
   /* выполнить функцию func в рамках одной транзакиции, вернуть
  результат функции func типа T, в функцию передается объект
 класса Session, который позволяет осуществлять запросы к базе */

  protected <T> T composeInTransaction(Function<Session, T> func)
  throws HibernateException {
    T entity;
    Session session = getSession();
    session.beginTransaction();
    entity = func.apply(session);
    session.getTransaction().commit();
    return entity;
  }
```

```
  /* то же самое, что и composeInTransaction,
  но для действия без результата */
  protected void consumeInTransaction
  (Consumer<Session> consumer) {
    composeInTransaction(session -> {
      consumer.accept(session);
      return null;
    });
  }

  /* создает объект класса Criteria,
  который позволяет формировать условные запросы */
  protected Criteria createEntityCriteria() {
    return getSession().createCriteria(persistentClass);
  }

  /* поиск объекта по первичному ключу */
  public E findByKey(PK primaryKey) {
    return composeInTransaction(session ->
      session.get(persistentClass, primaryKey)
    );
  }

  /* получить список всех объектов из базы */
  public List<E> findAll() {
    return composeInTransaction(session ->
      (List<E>) createEntityCriteria().list()
    );
  }

  /* создать объект типа E,
  вернет ключ типа PK созданного объекта */
  public PK create(E entity) {
    return composeInTransaction(session ->
      (PK) session.save(entity)
    );
  }

  /* обновление объекта */
  public void update(E entity) {
    consumeInTransaction(session -> session.update(entity));
  }

  /* удаление объекта */
  public void delete(E entity) {
    consumeInTransaction(session -> session.delete(entity));
  }
}
```

Таким образом, для того, чтобы реализовать класс, позволяющий выполнять доступ к объектам той или иной сущности остается лишь

унаследоваться от класса `HibernateDao`, объявить конструктор и при необходимости определить дополнительные методы.

Все классы, осуществляющие доступ к данным располагаются в пакете `ru.bmstu.schedule.dao` в папке модуля `schedule-core`.

4.3 Заполнение базы тестовыми данными

Все необходимые данные можно найти на официальном сайте МГТУ им. Н. Э. Баумана и на портале «Электронный университет» вуза. Большая часть данных представлены в табличном виде, поэтому особых проблем при их преобразовании в формат csv не возникает. В указанных источниках представлены все требуемые данные, за исключением информации об аудиториях вуза и информации о том, какой предмет ведет тот или иной преподаватель. Однако, данную информацию можно «вычленить» из расписания занятий, предоставленных в виде веб-страниц на официальном сайте вуза. Поскольку общее число групп и расписаний к ним (даже в рамках одной кафедры) велико, будет тяжело выполнить извлечение требуемых данных вручную. Поэтому для автоматизации данного процесса нам понадобиться скачать html-страницы с сайта университета и распарсить их. Данный функционал предоставляет библиотека для языка Java — `org.josup`. Эта библиотека поддерживает язык *селекторов* для выбора вершин DOM-дерева. С помощью селекторов можно получить вершину или группу вершин DOM-дерева, которые представляются объектами интерфейса `Element` и `Elements` соответственно. У данных объектов имеется ряд методов для доступа к содержимому вершин и их атрибутов.

Также для облегчения работы с csv файлами была использована библиотека Apache Commons CSV, которая позволяет парсить и формировать csv файлы, записанные в различных нотациях вместе с заголовками.

Таким образом, для заполнения базы, вначале, формируется набор справочников, которые берутся с сайта университета. Затем данные

справочники парсятся средствами библиотеки Apache Commons Csv и сохраняются в базу с помощью классов, описанных в 4.3. Последнюю процедуру выполняет утилита `dbfill`, которая на вход принимает properties-фаулу, где каждый ключ соответствует пути к csv-файлу справочнику (более подробное описание данного файла приводится в разделе 5.2.2). Поскольку все данные скачанные с сайта вуза предварительно сохраняются в csv-файлы, для заполнения базы тестовыми данными нам не понадобиться подключение к сети Интернет для корректной работы утилиты `dbfill`. Это обстоятельство позволяет уменьшить список требований, выдвигаемых при инсталляции приложения.

4.4 Автоматическое составление расписания

Для автоматического составления расписания была реализована утилита под названием `smtgen`. Данная утилита представляет из себя консольное приложение, которое принимает на вход список групп и генерирует для данных групп расписание занятий в соответствии со схемой приведенной в разделе 3. Если в базе содержится достаточно информации для генерации непротиворечивого расписания, то программа выводит сообщение об успешном окончании работы и сохраняет составленные расписания в базу. В противном случае, выводится сообщение об ошибке. Также следует отметить, что если расписание для заданных групп уже существует в базе данных, то данное расписание перезаписывается новым.

Для написания данной утилиты был реализован ряд вспомогательных структур, а также следующие классы, в которых реализуется основная логика работы данного приложения:

– `GenerateSchedule` — точка входа в программу, распознает аргументы командной строки, загружает всю необходимую информацию из базы, составляет расписание для групп и сохраняет его в базу.

– SmtSchedueGenerator — отвечает непосредственно за генерацию расписания, для загруженных данных формирует модель с помощью SmtScheduleModelGenerator; преобразует данную модель в реляционную модель с помощью ModelToScheduleTransformer. Прототип класса SmtSchedueGenerator вместе с описанием аргументов каждого метода приводится в листинге 4.

– SmtScheduleModelGenerator — класс, реализующий взаимодействие с библиотекой Z3, выполняет генерацию модели для загруженных из базы объектов сущностей. Прототип данного класса приведен в листинге 6.

– ModelToScheduleTransformer — преобразуют сгенерированную модель расписания обратно в сущности базы. Прототип этого класса приводится в листинге 5.

Листинг 4 — Прототип класса, генерирующего расписание

```
public class SmtSchedueGenerator {
  // конструктор, принимает объекты из базы:
  SmtScheduleGenerator(
      /* словарь - каждому предмету ставит
       в соответствие число часов по каждому типу занятий */
      Map<Subject, SubjectsPerWeek> totalSubjectsPerWeak,
      /* коллекция - каждый элемент содержит тройку:
      (преподоавтель, премдет, тип занятия) */
      Collection<LecturerSubject> lecturerSubjects,
      /* коллекция аудиторий: */
      List<Classroom> classrooms,
      /* коллеция групп для которых составляется расписание */
      List<StudyGroup> groups
  ) { }

  /* генерирует расписания, если это возможно, возвращает словарь:
  учебная группа -> расписание группы. Если расписание для
  предоставленных данных невозможно сгенерировать,
  выбрасывает RuntimeException: */
  Map<StudyGroup, Schedule> generateSchedule()
  throws RuntimeException
}
```

Параметры командной строки распознаются средствами бибилотеки Apache Commons CLI. Подробное описание всех параметров утилиты приводится в разделе 5.

Листинг 5 — Прототип класса `ModelToScheduleTransformer`

```
public class ModelToScheduleTransformer {
  /* первый аргумент - генератор модели, необходим для получения
  констант и самой модели;
  осстальные аргемнты - списки словарей, которые каждому
  идентификатору ставят в соотвествие объекты сущностей базы */
  public ModelToScheduleTransformer(
    SmtScheduleModelGenerator modelGenerator,
    Map<Integer, Subject> idToSubj,
    Map<Integer, Lecturer> idToLecturer,
    Map<Integer, StudyGroup> idToStudyGroup,
    Map<Integer, Classroom> idToClassroom,
    Map<LessonKind, ClassType> kindToClassType
  ) { }

  /* преобразует модель, полученную от Z3 в модель данных,
  возвращает словарь: каждой группе ставится в соответствие
  составленное расписание для данной группы */
  public Map<StudyGroup, Schedule> transform();
}
```

Загрузка информации из базы данных об указанных группах, учебных планах, аудиториях, предметов и преподавателях осуществляется с помощью классов, описанных в разделе 4.2.

Перед построением модели SMT-задачи все сущности базы преобразуются в константы, каждая из которых представляется целочисленным идентификатором, равному первичному ключу сущности базы. С помощью данных констант описываются ограничения, которые накладываются на конкретное расписание. Затем, список сформированных ограничений передается SMT-решателю с помощью библиотеки Z3 для Java. SMT-решатель проверяет ограничения на непротиворечивость, после чего, если это возможно, выдает модель. В рамках построенной модели формируется список занятий для каждого дня и для каждой группы. Далее данный список, преобразуется обратно в модель данных, после чего сохраняется в базу.

Листинг 6 — Прототип класса, выполняющий решение SMT-задачи и генерацию модели

```
public class SmtScheduleModelGenerator {
  /* список идентификаторов, каждый идентификатор
  - первичный ключ одной из сущностей, передаваемых
  в SmtSchedueGenerator;
  SubjectPerWeek – содержит идентификатор предмета и словарь с
количеством занятий каждого типа;
  TutorForLesson – тройка вида (идентификатор преподоваетеля,
идентификатор предмета, тип занятий), хранит информацию о том
какой преподаватель ведет тот или иной предмет */
  public SmtScheduleModelGenerator(
    Map<Integer, SubjectsPerWeek> totalSubjectsPerWeak,
    List<TutorForLesson> tutorForLessons,
    List<Integer> rooms,
    List<Integer> groups
  ) { }

  /* проверка ограничений на непротиворечивость
(составляет SMT-задачу и выполняет её решение) */
  public boolean satisfies();

  /* опциональный объект, представляющий модель
   Model - класс из библиотеки Z3 */
  public Optional<Model> getSmtModel();

  // возвращает константы с группами
  public Expr[] getGroupsConstants();

  // возвращает константы со слотами
  public Expr[] getSlotsConstants();

  // возвращает константы с группами
  public Expr[] getDaysConstants();
}
```

4.5 Генерация PDF документа

Для составление PDF документа, содержащего расписание занятий учебной группы, была реализована утилита под названием `pdfgen`. Данная утилита принимает на вход путь к выходному PDF-документу и шифр учебной группы. Если в базе имеется расписание для заданной группы, данная утилита сохраняет PDF-документ с расписанием по указанному пути. В противному случае программа выдает сообщение об ошибке. Расписание для заданной группы можно представить как список дней. При этом каждый день можно отобразить в виде таблицы, в первой колонке которой отображается время

начала и окончания пары, а во второй — список занятий. Если занятия различается в зависимости от четности недели, то ячейка во второй колонке разделяется на две — первая под числитель, вторая под знаменатель. Каждая ячейка с занятием должна содержать предмет, тип занятия, номер аудитории и преподавателя, который ведет данное занятие.

Для формирования PDF документа использовалась библиотека iTextPDF.

PDF-документ, который требуется сформировать, можно представить следующими базовыми элементами: параграф, таблица, ячейка таблицы. Каждый из этих элементов в библиотеке iText представляется соответствующими классами: `Paragraph`, `Table`, `Cell`. Для работы с самим PDF-документом в данной библиотеке предоставлен класс `Document`. У класса `Document` есть метод `add()` для добавления дочернего элемента, у `Table` есть метод `addCell(Cell)` для добавления ячейки. Объект класса `Document` принимает путь к выходному файлу при инициализации. После того как документ заполнен, у объекта класса Document необходимо вызвать метод `close()`, который закроет поток записи выходного файла.

5 Руководство пользователя

Реализованное программное решение, представляет из себя набор утилит, для запуска которых потребуется установить следующее:

1) PostgreSQL Server версии 10.8.
2) Java Runtime Environment версии 1.8.
3) Python 2.7.
4) Microsoft Z3 Prover и байндинг библиотеки SMT Lib 2.0 для Java версии 4.8.4 под используемую платформу.
5) Apache Maven версии 3.6 (для сборки приложения).
6) Java Development Kit версии 1.8 (для сборки приложения).

Проект с исходным кодом проекте можно найти на компакт диске, который прилагается к данной работе.

Весь проект состоит из нескольких модулей (консольных утилит и библиотек), которые располагаются в одноименных папках в корневой директории проекта (см. раздел 4).

В папке проекта, содержится директория build/ в которой расположены сборки каждой из утилит (jar-архвивы). Некоторые утилиты могут не запускаться на определенных платформах. Поэтому перед использованием может потребоваться пересобрать весь проект под используемую платформу.

Также, прежде чем, начать использовать утилиты, необходимо настроить и заполнить базу данных PostgresSQL.

5.1 Сборка проекта

Некоторые модули программного решения, например `smtgen`, могут не работать на разных платформах. Для успешной работы утилит может потребоваться их сборка.

Перед сборкой всего проекта, необходимо собрать Z3 из репозитория [8]. В указанном источнике, располагается пошаговая инструкция, описывающая как это сделать.

В Linux для сборки нужно выполнить следующие команды в терминале:

```
$ cd {папка, где будет располагаться z3}
$ git clone https://github.com/Z3Prover/z3.git
$ python scripts/mk_make.py --java
$ cd ./build
$ make && make
$ sudo make install
```

После сборки Z3, байндинг для Java необходимо установить с помощью Maven, выполнив команду из под директории `./build` в папке с библиотекой Z3:

```
$ mvn instal:install-file -Dfile=com.microsoft.z3.jar -
DgroupId=com.microsoft.z3 -DartifactId=javasmt-solver-z3 -
Dversion=4.8.4 -Dpackaging=jar
```

Перед сборкой, может также потребоваться поменять данные подключения к базе PostgreSQL, в том случае, если на этапе настройки базы указываются нестандартные параметры. Параметры подключения указываются

в файле `./schedule-core/src/main/resources/hibernate.cfg.xml`. Пример данного приводился в листинге 1, в разделе 4.1.

И наконец, сам проект можно собрать выполнив команду в директории проекта:

```
$ mvn clean install -DskipTests=true
```

В результате выполнения данной программы, в папке каждого модуля появится директория `target/`, где будет располагаться jar-файл каждой утилиты, которую можно запустить выполнив команды:

```
$ cd {название утилиты}/
$ java -jar ./target/{название утилиты}.jar
```

5.2 Настройка базы и её заполнение тестовыми данными

Для того чтобы «накатить» базу данных с расписанием, пользователю предоставляется два варианта: а) скачать готовый docker-образ с базой, который при первом запуске автоматически произведет все настройки и заполнит базу; б) установить и настроить PostgreSQL вручную, прогнав необходимые скрипты инициализации.

В следующих подразделах приводится каждый из вышеописанных способов.

5.2.1 Установка PostgreSQL с помощью Docker

Если пользователь выбрал данный вариант, ему понадобиться установить Docker. О том как это можно сделать в Linux Ubuntu, подробно описано в [9]. Для Windows придется, выполнить немного другую процедуру, о которой описано в [10].

После того как Docker установлен и запущен, необходимо скачать образ с базой, выполнив команду:

```
$ docker pull karkinai/schedule-db
```

После чего образ можно запустить с помощью команды:

```
$ docker run --rm -p 5432:5432 --name scheduledb
karkinai/schedule-db
```

Примечания:

1 Для запуска образа в режиме демона, можно добавить параметр `-d`.

2 Для того, чтобы данные не стирались при каждом запуске, можно смонтировать папку с данными PostgreSQL в локальную папку. Для этого нужно добавить параметр `-v` при запуске: `-v {путь к папке с данными на локальной машине}:/var/lib/postgresql/pgdata`.

5.2.2 Установка PostgreSQL Server на локальный компьютер

В Linux Ubuntu, PostgreSQL можно установить выполнив команду в терминале (для этого потребуется права root-пользователя):

```
$ sudo apt-get install postgresql postgresql-contrib
```

О том, как установить PostgreSQL для других операционных систем, подробно описано в [11].

Далее, необходимо создать пользователя и поставить пользователю пароль. В Linux это можно сделать выполнив следующие команды в терминале:

```
$ sudo -u postgres createuser <имя пользователя>
$ sudo -u postgres psql
psql=# alter user <имя пользователя> with encrypted password
'<пароль для пользователя>';
```

Примечание – для упрощения процедуры настройки рекомендуется в качестве имени пользователя использовать «`admin`» с паролем «`admin`». Это позволит избежать дополнительных действий в дальнейшем.

После того, как создан пользователь, необходимо прогнать скрипт, создания базы, выполнив команду в директории проекта (в терминале):

```
$ psql --username postgres --dbname postgres -a -f
./sql/create_tables.sql
```

Затем запустить скрипт заполнения базы:

```
$ java -jar ./build/dbfill.jar
./dbfill/src/main/resources/config.properties
# или (если выполнялась процедура, описанная в предыдущем
разделе):
$ java -jar ./dbfill/target/dbfill.jar
./dbfill/src/main/resources/config.properties
```

По умолчанию, утилита `dbfill`, заполняет базу лишь демонстрационными данными (в базу записываются только данные для кафедры ИУ9 МГТУ им. Баумана и для специальности «01.03.02. Прикладная математика и информатика»). Однако набор данных, можно расширять, изменив пути к входным файлам-справочникам. Для этого нужно изменить параметры в файле `./dbfill/src/main/resources/config.properties`, расположенном в директории проекта, после чего пересобрать проект, или же при запуске указать путь к этому или другому properties файлу с аналогичной структурой. В данном файле указываются свойства в формате ключ-значение. Каждое значение соответствует пути к csv-справочнику или папке со справочниками (для учебного плана). Пример такого файла, вместе с описанием ключей приводится в листинге 7.

Листинг 7 — Утилита `dbfill`: пример файла `config.properties`

```
// справлчкик с типами занятий
csv.file.classtypes=/references/class-types.csv
// дни недели
csv.file.week=/references/day-of-weeks.csv
// справочник с преподователями
csv.file.lecturers=/references/lecturers.csv
// коды специализаций, специальностей, их название
csv.file.specs=/references/bmstu_specializations.csv
// время занятий (начало, конец, номер)
csv.file.classtime=/references/class_time.csv
// кафедры
csv.file.departments=/references/bmstu_departments.csv
// папка с календарными планами (содержит csv-файлы)
csv.folder.calendar=/references/demo_iu9/
// факультеты
csv.file.faculties=/references/bmstu_faculties.csv
// группы (для кажд. Группы указ. специализация)
csv.file.groups=/references/bmstu_groups.csv
// соотношение (преподователь, предмет)
csv.file.subjects=/references/lecturer_subjects.csv
```

Примечания:

1 Пример из листинга 7 взят из папки проекта утилиты `dbfill`. По указанным в примере путям (пути располагаются относительно папки `./dbfill/src/main/resources/`) можно найти примеры csv-справочников для каждой таблицы.

2 В файле `config.properties` ключ `csv.folder.calendar` указывает на папку с csv-файлами, соответствующими календарным планам. Название каждого такого файла, должно удовлетворять шаблону: `{Кафедра}__{Код специализации}__{Год начала обучения}.csv`. Каждый такой файл содержит наименование дисциплины, кафедры на которой преподается данная дисциплина и номера семестров, на которых данная дисциплина преподается. Примеры таких файлов можно найти в папке `./dbfill/src/main/resources/references/demo_iu9/`.

5.3 Утилита smtgen

Утилита `smtgen` предоставляет возможность генерировать расписание, либо для списка групп, либо для учебного потока. Во втором случае, задается код специализации, код кафедры и год начала обучения.

В листинге 8 приведен список входных параметров, которые может принимать программа.

Листинг 8 — Список параметров командной строки утилиты `smtgen`

```
-g, --groups <Список учебных групп>
-d, --dept <Шифр кафедры>
-s, --spec <Код специализации>
-y, --year <Год начала обучения, соотвествует году календарном плане>
-t, --term <Номер семестра>
```

Если указывается параметр `-g`, то остальные параметры игнорируются. Если параметр `-g` опущен, то должны быть указаны все остальные параметры. В противном случае, программа выводит сообщение об ошибке.

Пример запуска программы:

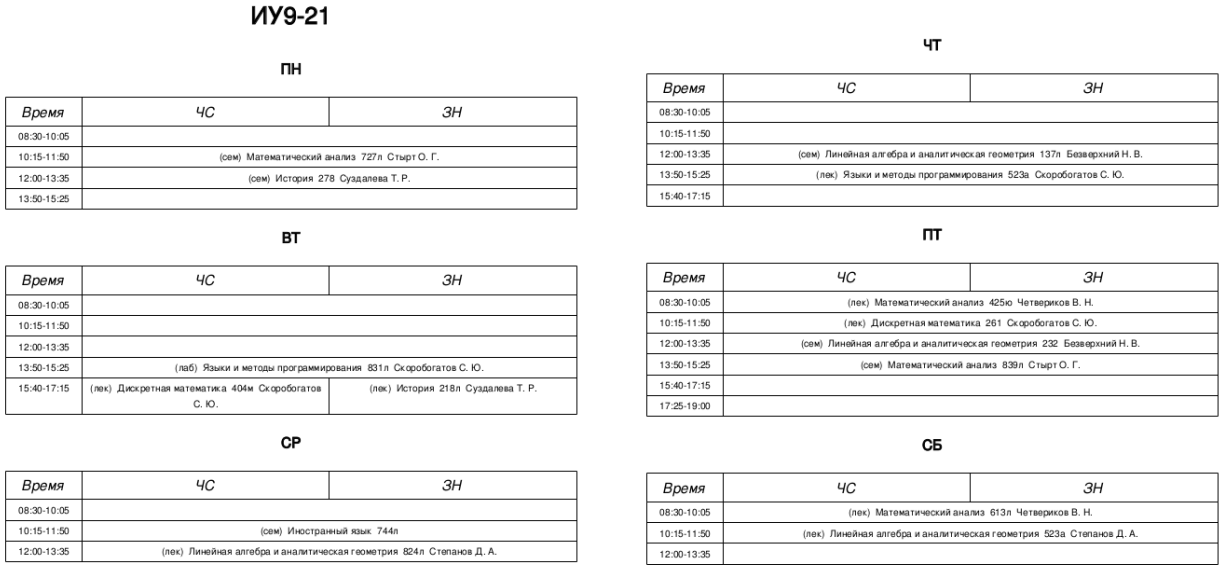
```
$ java -jar ./release/smtgen-1.0.1.jar -g ИУ9-21 ИУ9-22
```

В результате работы, программа генерирует расписание вуза, выводит промежуточный результат в консоль и сохраняет его в базу данных.

В случае, если программе переданы параметры, при которых расписание не может быть сгенерировано с учетом всех ограничений, программа выдает сообщение об ошибке: «Невозможно построить модель для данных параметров».

5.4 Формирование pdf-документа

Утилита `pdfgen` принимает всего два параметра: шифр группы и путь к папке, в которой будет содержатся выходной файл. Если расписания для данной группы отсутствует в базе или группы с данным шифром не существует, то программа выведет сообщение об ошибке. В случае успешного завершения программа сгенерирует pdf-документ. Пример такого документа приведен на рисунке 1.



ИУ9-21

ПН

Время	*ЧС*	*ЗН*
08:30-10:05		
10:15-11:50	(сем) Математический анализ 727л Стырт О. Г.	
12:00-13:35	(сем) История 278 Суздалева Т. Р.	
13:50-15:25		

ВТ

Время	*ЧС*	*ЗН*
08:30-10:05		
10:15-11:50		
12:00-13:35		
13:50-15:25	(лаб) Языки и методы программирования 831л Скоробогатов С. Ю.	
15:40-17:15	(лек) Дискретная математика 404м Скоробогатов С. Ю.	(лек) История 218л Суздалева Т. Р.

СР

Время	*ЧС*	*ЗН*
08:30-10:05		
10:15-11:50	(сем) Иностранный язык 744л	
12:00-13:35	(лек) Линейная алгебра и аналитическая геометрия 824л Степанов Д. А.	

ЧТ

Время	*ЧС*	*ЗН*
08:30-10:05		
10:15-11:50		
12:00-13:35	(сем) Линейная алгебра и аналитическая геометрия 137л Безверхний Н. В.	
13:50-15:25	(лек) Языки и методы программирования 523а Скоробогатов С. Ю.	
15:40-17:15		

ПТ

Время	*ЧС*	*ЗН*
08:30-10:05	(лек) Математический анализ 425ю Четвериков В. Н.	
10:15-11:50	(лек) Дискретная математика 261 Скоробогатов С. Ю.	
12:00-13:35	(сем) Линейная алгебра и аналитическая геометрия 232 Безверхний Н. В.	
13:50-15:25	(сем) Математический анализ 839л Стырт О. Г.	
15:40-17:15		
17:25-19:00		

СБ

Время	*ЧС*	*ЗН*
08:30-10:05	(лек) Математический анализ 613л Четвериков В. Н.	
10:15-11:50	(лек) Линейная алгебра и аналитическая геометрия 523а Степанов Д. А.	
12:00-13:35		

Рисунок 1 — Пример pdf-документа с расписанием занятий (группа ИУ9-21)

6 Тестирование

Для тестирования приложения использовалось два подхода: тестирование отдельных классов с помощью юнит-тестов, тестирование всей системы в целом на реальных данных.

Для написания юнит-тестов использовался фреймворк для языка Java, под названием JUnit Jupiter v 5.5. Данный фреймворк, по-мимо базового функционала, который предоставляет каждый из подобного рода фреймворк (проверка условий с помощью assert-ов, заглушки или моки, проверка методов на выброс исключений и т. д.), JUnit Jupiter позволяет писать тесты с использованием такой методики, как Data-Driven testing (DDT) — написания тестов, управляемых данными [12]. Данная методика заключается в разделении логики тестирования и данных для тестирования. При этом данные представляются в удобной для записи краткой форме, например, в формате csv. Описанный подход активно использовался при написании тестов.

Юнит-тесты, в частности, использовались для тестирования ограничений, накладываемых на базу, которые были описаны в разделе 3.1.3. Для каждого DAO-класса (см. подраздел 4.2) формировался набор искусственных данных. Для данного набора данных выполнялись базовые CRUD-операции и проверялась реакция базы на данные операции. При этом входе написания тестов, помимо стандартных, подбирались такие данные при которых ограничения не выполнялись, в результате чего «выбрасывалась» ошибка на уровне базы, которая приводила к срабатыванию исключений на уровне Hibernate. После завершения группы тестов, все объекты созданные в базе для выполнения тестов, удалялись. В итоге, описанные тесты не оставляли никаких побочных эффектов.

В результате проведения тестирования, все реализованные юнит-тесты были успешно пройдены.

После проведения юнит-тестирования, было проведено тестирование всей системы на реальных данных. Данные для составления расписания, скачанные с

официального сайта, были преобразованы в csv-справочники с помощью методов, описанными в разделе 4.3. Для тестирования были отобраны данные для кафедры ИУ9 и специализации «01.03.02 Прикладная математика и информатика — Анализ, порождение и преобразование программного кода». Эти справочники загружались в базу данных с помощью утилиты `dbfill`. Далее, с помощью утилиты `smtgen` запускалась генерация расписания для каждой группы в отдельности, затем для пары групп, для трех и для четырех. (Все группы, которые передавались программе, были с одного семестра и имели общий учебный план). В результате работы утилиты `smtgen` формировалось расписание, которое сохранялось в базу. Затем, составленные расписания переводилось в PDF-документ с помощью утилиты `pdfgen`. PDF-файлы для двух групп, которые сгенерировала данная программа приводятся в приложении . В ходе проведения данных тестов, каждое из составленных расписаний удовлетворило всем ограничениям для заданного набора групп, описанным в подразделе 3.1.3.

ЗАКЛЮЧЕНИЕ

В данной работе была поставлена задача автоматического составления расписания занятий в вузе. Данная задача, была разбита на ряд подзадач: подготовка данных для составления расписание, заполнение этими данными реляционной базы данных, составление расписания, генерация PDF-документа для представления расписания в читаемом виде. В результате выполнения ВКР, был реализован ряд утилит, каждая из которых решает одну из поставленных подзадач. Каждая утилита была протестирована, в результате чего, было выявлено, что реализованное программное решение успешно справляется с поставленной задачей.

В ходе тестирования, было также выяснено, что на составление расписания у SMT-решателя для трех и более групп уходит порядка получаса. Данный результат является терпимым для составление расписания занятий одного учебного потока. Однако, при составлении расписания для всего вуза, может потребоваться куда больше времени. Возможно, данное время можно сократить увеличив вычислительную мощность компьютера и распараллелив процедуру составления расписания по семестрам и учебным планам.

В то же время, полученный результат наводит на мысль, что возможно, при автоматическом составлении расписания следует использовать другие подходы, не использующие SMT-решателей [13]. В частности, задачу составление расписания можно рассматривать, как задачу целочисленной многокритериальной оптимизации при заданных ограничениях [14].

В ходе выполнения работы, были успешно выполнены все поставленные цели, был выбран наиболее подходящий для составления расписания SMT-решатель, разработано программное решение, позволяющее автоматически составлять расписание занятий, была проведена оценка эффективности полученного решения в рамках данного подхода.

Для дальнейшего улучшения разработанного решения можно предложить, ряд модификаций, к числу которых можно отнести следующие.

1) Добавление дополнительных ограничений при формирования расписания:

- учет свободного времени преподавателей: некоторые преподаватели по-мимо данного учебного заведения могут работать и в других места, и как следствие, они могут появляется в данном учебном заведении только в определенные дни и только в определенное время
- учет вместимости аудиторий: не все аудиторий рассчитаны на проведение занятий для всего учебного потока, как это бывает в случае с лекциями
- учет того факта, что определенного типа занятий по определенным предметам могут проводится только в специализированных аудиториях; так например, лабораторная по физике, должна проводится в аудитории оснащенной специальным оборудованием.

2) Можно также добавить ряд условий оптимальности составляемого расписания. Например, расписание должно создаваться так, чтобы и преподавателям, и для студентам приходилось преодолевать минимальное расстояние в течение дня, и при том, число перемещений между зданиями университета должно быть минимальным.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Успенский, В. А. Вводный курс математической логики / В. А. Успенский, Н. К. Верещагин, В. Е. Пилско. — 2-е изд. — М.: ФИЗМАТЛИТ, 2004. — 128 с.

2 Канатников, А. Н. Математическая логика и теория алгоритмов: учеб. пособие для студ. каф. ИУ9 / А. Н. Канатников — М.: изд-во МГТУ им. Н. Э. Баумана, 2010. — 74 с.

3 Мендельсон Э. Введение в математическую логику [Текст] / Э. Меднельсон; пер. с англ. Ф.А. Какбкова, техн. ред. К.Ф. Брудно — Москва: изд-во "Накука", 1971. – 320 с.

4 Barrett C. Satisfiability Modulo Theories / C. Barret [and others] // Frontiers in Artificial Intelligence and Applications. — IOS Press, Feb. 2009 – vol. 185 – P. 825-885.

5 Burstall. R. M. Proving properties of porgrams by structural induction // Computer Journal — London: 1969. — Vol. 12 — P. 135-154. Access: http://www.cse.chalmers.se/edu/year/2010/course/DAT140_Types/Burstall.pdf

6 De Moura L. Z3: An Efficient SMT Solver / L. de Moura, N. Bjørner // Proceedings from 14th International Conference: Tools and Algorithms for the Construction and Analysis of Systems — Budapest: Springer-Verlag Berlin Heidelberg, 2008 — P. 337-340.

7 Nieuwenhuis R., Oliveras A., Tinelli C.. Abstract DPLL and abstract DPLL modulo theories. // Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning — Montevideo: Springer, 2005. — P. 36-50

8 Microsoft. Z3 Theorem Prover. GitHub Repository [Electronic resource]. — 2017 — Access mode: https://github.com/Z3Prover/z3 (online; accessed: 22.06.2019)

9 Docker docs – Get Docker CE for Ubuntu [Electronic resource]. – 2019. – Access mode: https://docs.docker.com/install/linux/docker-ce/ubuntu/ (online; accessed: 09.06.2019)

10 Docker docs — Install Docker Toolbox on Windows [Electronic resource]. — 2019. — Access mode: https://docs.docker.com/install/linux/docker-ce/ubuntu/ (online; accessed: 09.06.2019).

11 Postgres Guide – Installing Postgres [Electronic resource]. — 2019. — Access mode: http://postgresguide.com/setup/install.html (online; accessed: 09.06.2019).

12 C. J. Nagle. Test Automation Frameworks [Elecronice resource]. — 2015. — Access mode: http://safsdev.sourceforge.net/ (online; accessed: 09.06.2019)

13 Хаусаджиев, А. С. Обобщенный алгоритм составления расписания в вузе с учетом новых требований федеральных государственных образовательных стандартов. / А. С. Хаусаджиев, И. В. Сибикина. // Вестник Астраханского государственного технического университета. — 2016. — №3. — С. 78 – 86.

14 Burke E. K., Marecek J., Parkes J., Andrew J., Rudova H. Penalising Patterns in Timetables: Novel Integer Programming Formulations // Proccedings of the Operations Research — Berlin: Springer, 2008 — P. 409-414

ПРИЛОЖЕНИЕ А

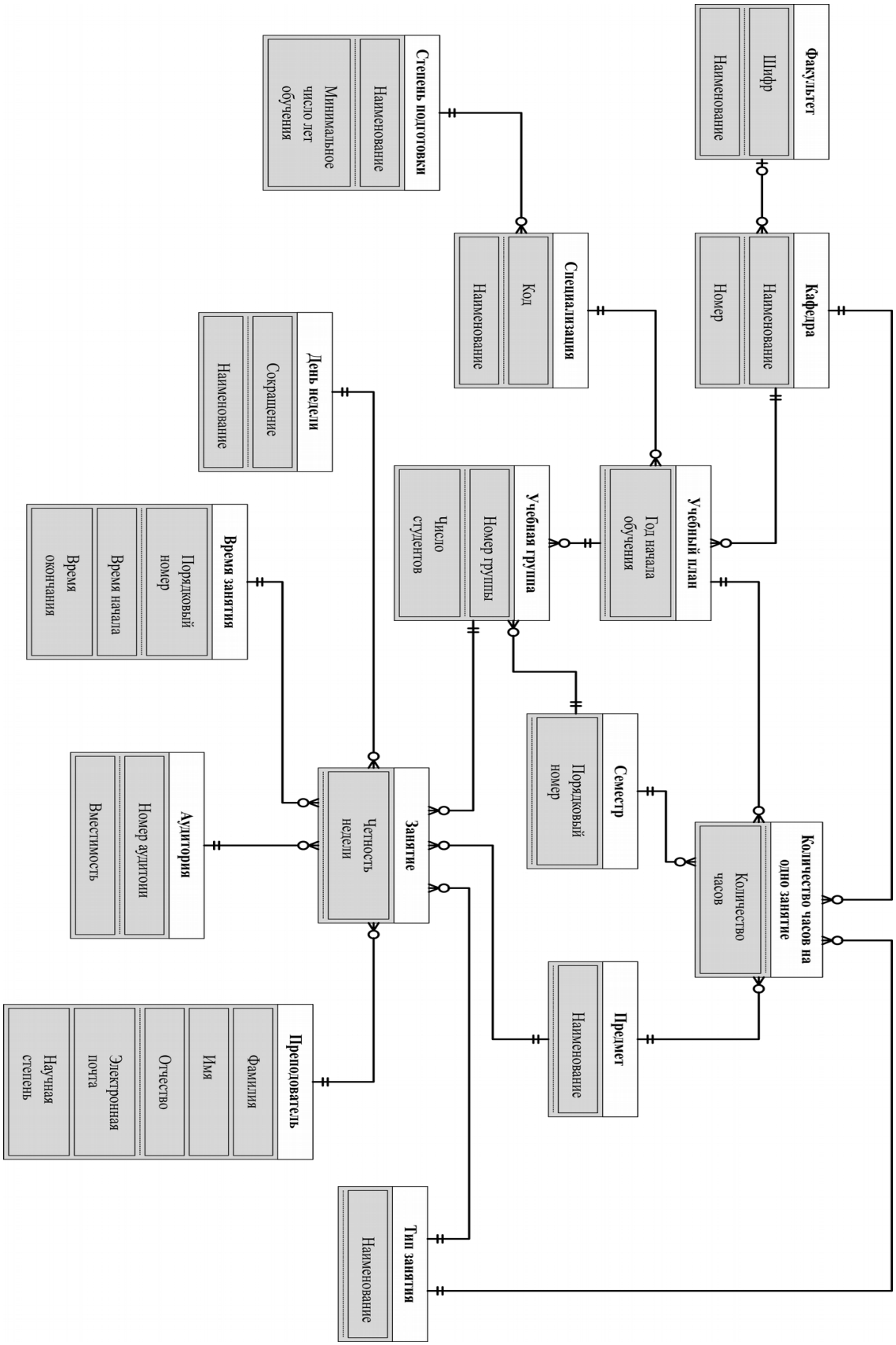


Рисунок 2 – ER-модель базы

ПРИЛОЖЕНИЕ Б

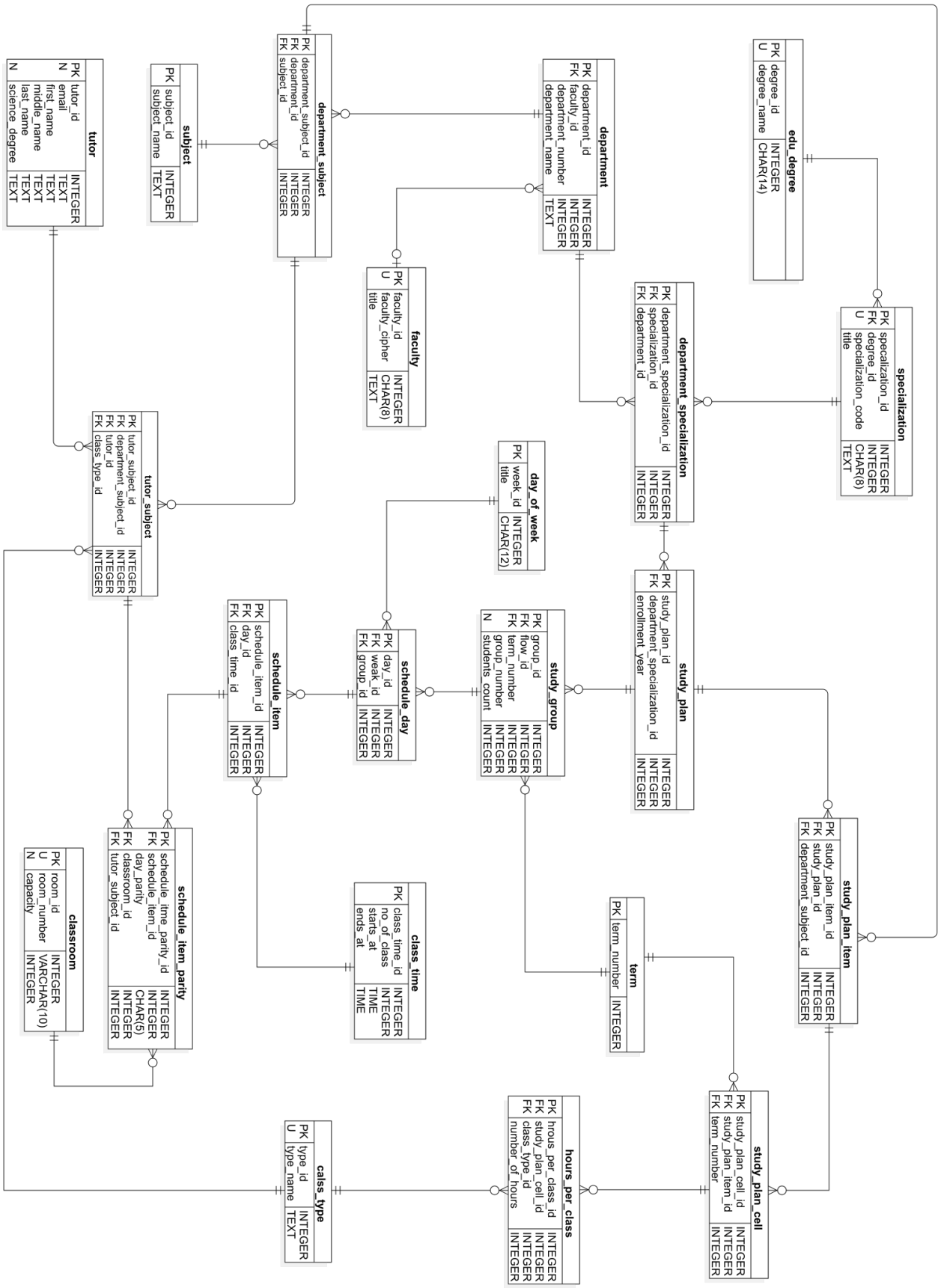


Рисунок 3 – Диаграмма реляционной модели

ПРИЛОЖЕНИЕ В

Понедельник

Время	*ЧС*	*ЗН*
8:30-10:05	(сем) История 218л Суздалева Т. Р.	
10:15-11:50	(сем) Линейная алгебра и аналитическая геометрия 520 Безверхний Н. В.	
12:00-13:25	(сем) Математический анализ 520 Стырт О. Г.	

Вторник

Время	*ЧС*	*ЗН*
8:30-10:05	(сем) Математический анализ 111л Стырт О. Г.	
10:15-11:50	(лек) Математический анализ 534л Четвериков В. Н. В.	
12:00-13:25	(сем) Иностранный язык 224л	

Среда

Время	*ЧС*	*ЗН*
	Занятий нет	

Четверг

Время	*ЧС*	*ЗН*
8:30-10:05	(лек) Языки и методы программирования 534л Скоробогатов С. Ю.	
10:15-11:50	(сем) Линейная алгебра и аналитическая геометрия 1013л Безверхний Н. В.	

Пятница

Время	*ЧС*	*ЗН*
8:30-10:05	(лек) Линейная алгебра и аналитическая геометрия 224л Степанов Д. А.	
10:15-11:50	(лек) История 1013л Суздалева Т. Р.	(лек) Дискретная математика 534л Скоробогатов С. Ю.
12:00-13:25	(лаб) Языки и методы программирования 1139л Скоробогатов С. Ю.	

Суббота

Время	*ЧС*	*ЗН*
10:15-11:50	(лек) Линейная алгебра и аналитическая геометрия 520 Степанов Д. А.	
12:00-13:35	(лек) Дискретная математика 224л Скоробогатов С. Ю.	
13:50-15:25	(лек) Математический анализ Каф Четвериков В. Н.	

Понедельник

Время	*ЧС*	*ЗН*
8:30-10:05	(сем) Линейная алгебра и аналитическая геометрия 534 Безверхний Н. В.	
10:15-11:50	(сем) Математический анализ 534л Стырт О. Г.	
12:00-13:25	(сем) Линейная алгебра и аналитическая геометрия Каф Безверхний Н. В.	

Вторник

Время	*ЧС*	*ЗН*
8:30-10:05	(лаб) Языки и методы программирования 224л Скоробогатов С. Ю.	
10:15-11:50	(лек) Математический анализ 534л Четвериков В. Н.	
12:00-13:25	(сем) История 1013л Суздалева Т. Р.	

Среда

Время	*ЧС*	*ЗН*
	Занятий нет	

Четверг

Время	*ЧС*	*ЗН*
8:30-10:05	(лек) Языки и методы программирования 534л Скоробогатов С. Ю.	
10:15-11:50	(сем) Математический анализ Каф Стырт О. Г.	

Пятница

Время	*ЧС*	*ЗН*
8:30-10:05	(лек) Линейная алгебра и аналитическая геометрия 224л Степанов Д. А.	
10:15-11:50	(лек) История 1013л Суздалева Т. Р.	(лек) Дискретная математика 534л Скоробогатов С. Ю.

Суббота

Время	*ЧС*	*ЗН*
08:30-10:05	(сем) Иностранный язык 1139л	
10:15-11:50	(лек) Линейная алгебра и аналитическая геометрия 520 Степанов Д. А.	
12:00-13:35	(лек) Дискретная математика 224л Скоробогатов С. Ю.	
13:50-15:25	(лек) Математический анализ Каф Четвериков В. Н.	

Рисунок 4 — Расписания для двух групп: слева – ИУ9-21, справа – ИУ9-22