Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20.Б11-мм

Муравьев Илья Владимирович

Разработка настольной версии «Конструктора вычислителей»

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования,
к.т.н. Литвинов Юрий Викторович

Санкт-Петербург
2022

Оглавление

1. Введение **3**

2. Постановка задачи **5**

3. Обзор литературы и существующих решений **6**

3.1. Обзор используемых аналогами способов отладки недетерминированных вычислителей 6

3.2. Используемые технологии 10

4. Архитектура **12**

4.1. Ядро модели 12

4.2. Вся система 12

5. Реализация **15**

5.1. Отладка недетерминированных вычислителей 15

5.2. Автоматы с дополнительными свойствами состояний . . 19

5.3. Сохранение и открытие автоматов 19

5.4. Отмена и повтор действий 20

6. Тестирование и апробация **22**

6.1. Тестирование 22

6.2. Апробация 22

7. Заключение **23**

Список литературы **24**

1 Введение

На курсах «Алгоритмы и анализ сложности» и «Теория автоматов и формальных языков» распространенной задачей является проектирование абстрактных вычислителей и симуляция их работы. По протоколу оценки сформированности компетенций СВ.5006.2017 от 12 апреля 2021 года уровень усвоения материала на курсе «Теория автоматов и формальных языков» объективно низкий. Поэтому с целью упрощения восприятия материала на данных курсах и, как результат, повышения уровня его усвоения, было принято решение разработать приложение, позволяющее моделировать, исполнять, отлаживать и анализировать различные виды абстрактных вычислителей. Разрабатываемый инструмент помимо облегчения образовательного процесса также призван упростить моделирование реальных систем и создание автоматов для анализа текста.

Более подробно необходимость создания данного приложения была обоснована в «Создание приложения для симуляции конечных автоматов и других вычислителей» [9] и «Разработка сервиса визуального редактирования и симуляции конечных автоматов» [10], также в [9] и [10] был проведён обзор аналогов, выявивший, что самым развитым аналогом является JFLAP [8], и показавший отсутствие существующих решений удовлетворяющих функциональным требованиям, сформулированным в техническом задании на проект «Конструктор вычислителей» [14].

В связи с этим, была начата разработка веб [10], [11] и настольной [9], [13] версий «Конструктора вычислителей». Данная работа посвящена совершенствованию разработанного в [13] приложения, что включает:

- создание визуального представления недетерминированных вычислений, так как это направление мало развито аналогами, но является существенным при изучении и отладке НКА, МП-автомата, недетерминированной МТ и других видов недетерминированных вычислителей;

- поддержку сохранения вычислителей, необходимую для длительной работы над одним проектом;
- поддержку автоматов с дополнительными свойствами состояний, что желательно завершить до начала работы над сериализацией и необходимо для реализации автомата Мура;
- поддержку отмены и повтора совершённых действий, нужную для упрощения редактирования автоматов.

2 Постановка задачи

Целью данной работы стало усовершенствование настольной версии «Конструктора вычислителей». Для её выполнения были поставлены следующие задачи:

1. Произвести обзор используемых аналогами способов отладки недетерминированных вычислителей.
2. Упростить отладку недетерминированных вычислителей в «Конструкторе вычислителей».
3. Поддержать автоматы с дополнительными свойствами состояний, в частности автомат Мура.
4. Реализовать сохранение и открытие автоматов.
5. Реализовать отмену и повтор совершённых действий, согласованные с групповыми операциями.
6. Обновить пользовательскую документацию.
7. Провести тестирование и апробацию.

3 Обзор литературы и существующих решений

3.1 Обзор используемых аналогами способов отладки недетерминированных вычислителей

Из инструментов, решающих схожие задачи, для рассмотрения способов отладки недетерминированных вычислителей были выбраны следующие:

- JFLAP [8], являющийся наиболее полным инструментом согласно «Fifty years of automata simulation: a review» [7];
- Automata-Visualizer [1], поддерживающий ДКА, НКА, эпсилон-НКА, ДМП-автомат, МП-автомат и МТ;
- Automaton Simulator [2], поддерживающий ДКА, НКА, эпсилон-НКА, ДМП-автомат и МТ;
- FSM simulator [4], поддерживающий ДКА, НКА и эпсилон-НКА;
- automatonsimulator.com [6], поддерживающий ДКА, НКА, эпсилон-НКА, ДМП-автомат и МП-автомат.

Данный выбор мотивирован тем, что каждый из этих инструментов поддерживает некоторые виды недетерминированных вычислителей.

3.1.1 JFLAP

JFLAP [8] во время отладки в нижней части экрана показывает состояния исполнения (см. рис. 1) и позволяет:

- замораживать и размораживать состояния исполнения,
- производить одновременный шаг исполнения для всех незамороженных состояний,
- перезапускать исполнение,

- просматривать трассировки для выбранных состояний исполнения (см. рис. 2),
- удалять состояния исполнения.

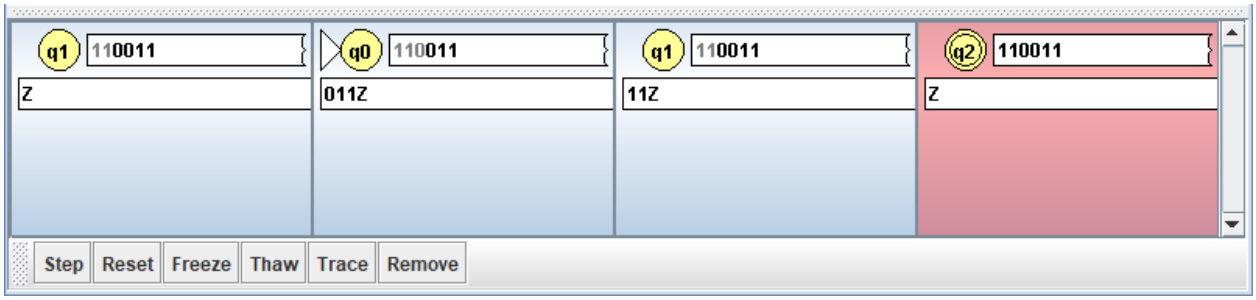

Рис. 1: Панель отладки в JFLAP

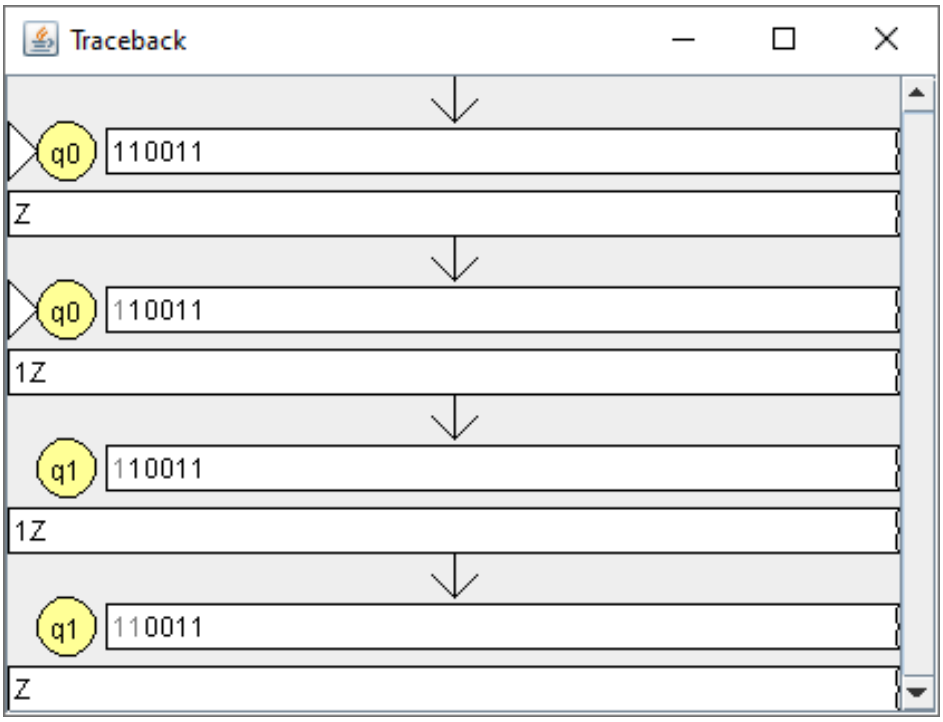

Рис. 2: Трассировка исполнения в JFLAP

Также во время отладки JFLAP выделяет цветом текущие состояния автомата (см. рис. 3).

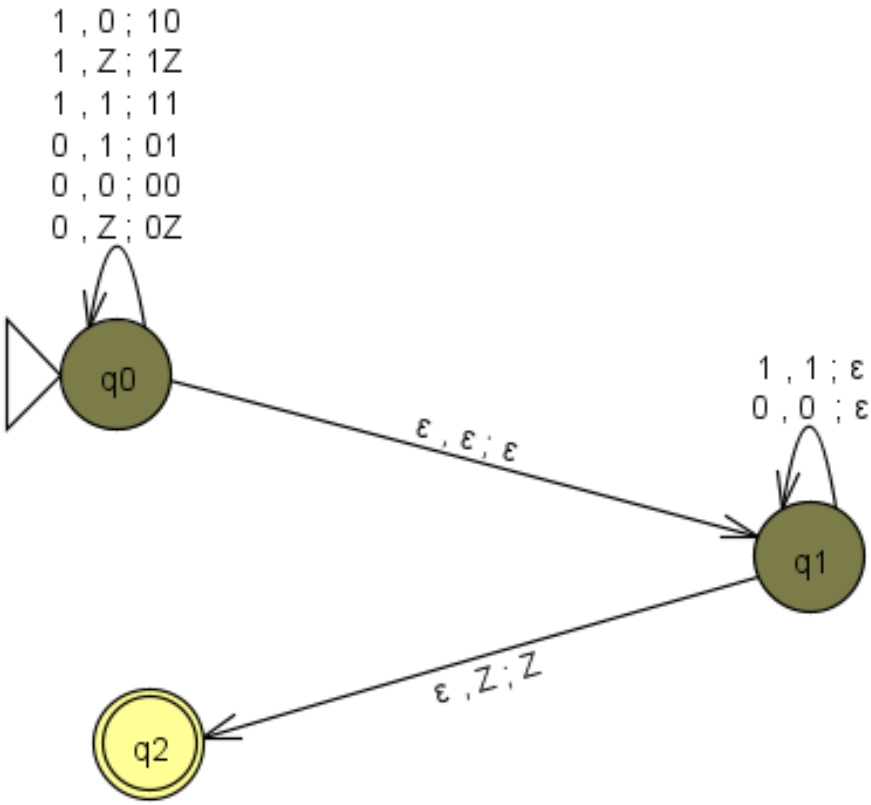

Рис. 3: Выделение текущих состояний автомата в JFLAP

Данный способ представления недетерминированных вычислений и управления ими хотя и предоставляет широкие возможности, но имеет ряд недостатков:

1. После каждого шага исполнения порядок, в котором отображаются состояния исполнения, непредсказуемым образом меняется, затрудняя процесс отладки.

2. Для изучения одного конкретного пути исполнения необходимо после каждого шага замораживать все ответвления.

3. При изучении трассировок для нескольких состояний исполнения приходится несколько раз просматривать совпадающие префиксы этих трассировок.

4. В трассировках не показывается, по какому из переходов автомат перешел из одного состояния в другое.

5. Нет возможности просматривать все варианты содержимого памяти, соответствующие выбранному состоянию автомата.

3.1.2 Automata-Visualizer

Automata-Visualizer [1] поддерживает только мгновенное исполнение и показывает только его результат без какой-либо отладочной информации, что значительно затрудняет отладку автоматов.

3.1.3 Automaton Simulator и FSM simulator

Automaton Simulator [2] и FSM simulator [4] из недетерминированных вычислителей поддерживают только НКА и эпсилон-НКА, что позволяет данным инструментам обойтись выделением цветом всех текущих состояний автомата и текущего обрабатываемого символа на входной ленте, что возможно благодаря одинаковой скорости обработки ввода всеми путями исполнения с помощью стратегии вычисления с шагом по замыканию. К сожалению, такая синхронизация памяти автомата для всех путей исполнения невозможна для более сложных видов вычислителей.

3.1.4 automatonsimulator.com

Отладка НКА и эпсилон-НКА в automatonsimulator.com [6] реализована аналогично рассмотренным ранее Automaton Simulator [2] и FSM simulator [4]. Для МП-автомата данный инструмент рядом с состояниями автомата показывает варианты содержимого стека, но эти данные отображаются на заднем плане и перекрываются стрелками переходов, что препятствует их восприятию (см. рис. 4). Возможности просматривать трассировки и производить исполнение только для выбранных путей исполнения нет.

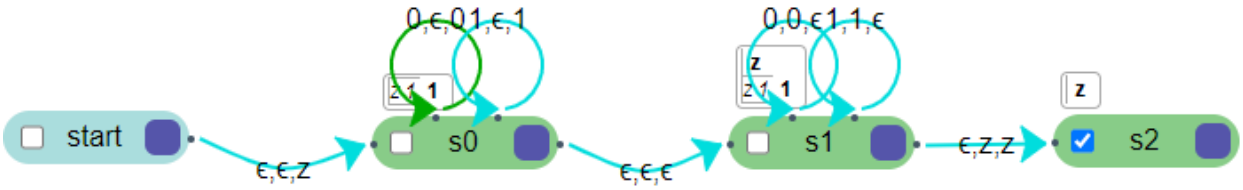

Рис. 4: Отображение содержимого стека рядом с состояниями в automatonsimulator.com

3.2 Используемые технологии

Разработка настольной версии ведётся на платформе JavaFX с использованием фреймворка TornadoFX на языке Kotlin, используется система сборки Gradle, тестирование производится с помощью фреймворка JUnit5. Выбор данных технологий был обоснован в предшествующей работе [13].

В рамках данной работы для упрощения установки «Конструктора вычислителей» пользователями:

- был произведён переход с JavaFX, встроенной в JDK и JRE до Java 11, на OpenJFX, доступную в виде отдельной библиотеки для Java 11+;
- была настроена сборка приложения в один исполняемый JAR-файл с помощью плагина Shadow, выбранного так как он интегрируется с уже используемым Gradle-плагином application.

Для сериализации вычислителей было решено использовать человеко-читаемый, а не бинарный формат, так как объем сохраняемых данных достаточно мал, чтобы преимущества человеко-читаемых форматов перевесили незначительный выигрыш в производительности, получаемый при использовании бинарных форматов.

Выбор производился из трёх библиотек для сериализации в человеко-читаемые форматы, поддерживающих полиморфизм и пользовательские сериализаторы, что требует структура сохраняемых объектов:

- Jackson,

- kotlinx.serialization,
- Gson.

Было выявлено, что все перечисленные библиотеки предоставляют необходимую функциональность, но так как kotlinx.serialization является официальной[1] Kotlin библиотекой для сериализации, то было решено использовать именно её.

Сериализация производилась в JSON формате, потому что он поддержан самой библиотекой kotlinx.serialization, а не её дополнительными расширениями сообществом[2].

[1]kotlinx.serialization находится в разделе «Official libraries» документации Kotlin — https://kotlinlang.org/docs/serialization.html (дата обращения 12.05.2022).

[2]Форматы, поддерживаемые kotlinx.serialization — https://github.com/Kotlin/kotlinx.serialization/blob/master/formats/README.md (дата обращения 13.05.2022).

4 Архитектура

4.1 Ядро модели

Основная идея, стоящая за архитектурой ядра модели (см. рис. 5), разработанной и более подробно описанной в [13], заключается в абстрагировании модулей исполнения, анализа, отладки и т.д. и их визуализаций от конкретных видов автоматов для предотвращения многократного переписывания одного и того же кода для каждого нового вида автоматов. Это достигается за счёт того, что каждый автомат задается своими элементами памяти, динамически определяющими, какие свойства (*фильтры*[3] и *побочные эффекты*[4]) должны быть у его переходов, а, начиная с этой работы, ещё и у состояний, в то время как общий код работает с абстрактными автоматами с памятью неизвестного ему вида и некоторым набором свойств переходов и состояний. Идея такой архитектуры была взята из «Dynamic Object Model» [3].

4.2 Вся система

Приложение в целом было спроектировано в [13] в соответствии с паттерном MVC и состоит из четырех компонентов (см. рис. 6):

- утилит, включающих несколько не являющихся специфичными для предметной области функций и классов, упрощающих реализацию других компонентов;
- модели автомата, предоставляющей методы для работы с ним и позволяющей подписываться на изменения;
- контроллера, обрабатывающего события взаимодействия пользователя с приложением и вызывающего методы модели;

[3] *Фильтром* в данной работе называется компонента аргумента δ-функции переходов в формальном определении автомата (например, символ, который должен быть на входной ленте для осуществления некоторого перехода).

[4] *Побочным эффектом* в данной работе называется компонента значения δ-функции переходов в формальном определении автомата (например, символ, записываемый на выходную ленту автомата Мура при переходе в некоторое состояние).

- представления, определяющего, что должно быть показано пользователю и реагирующего на изменения модели.

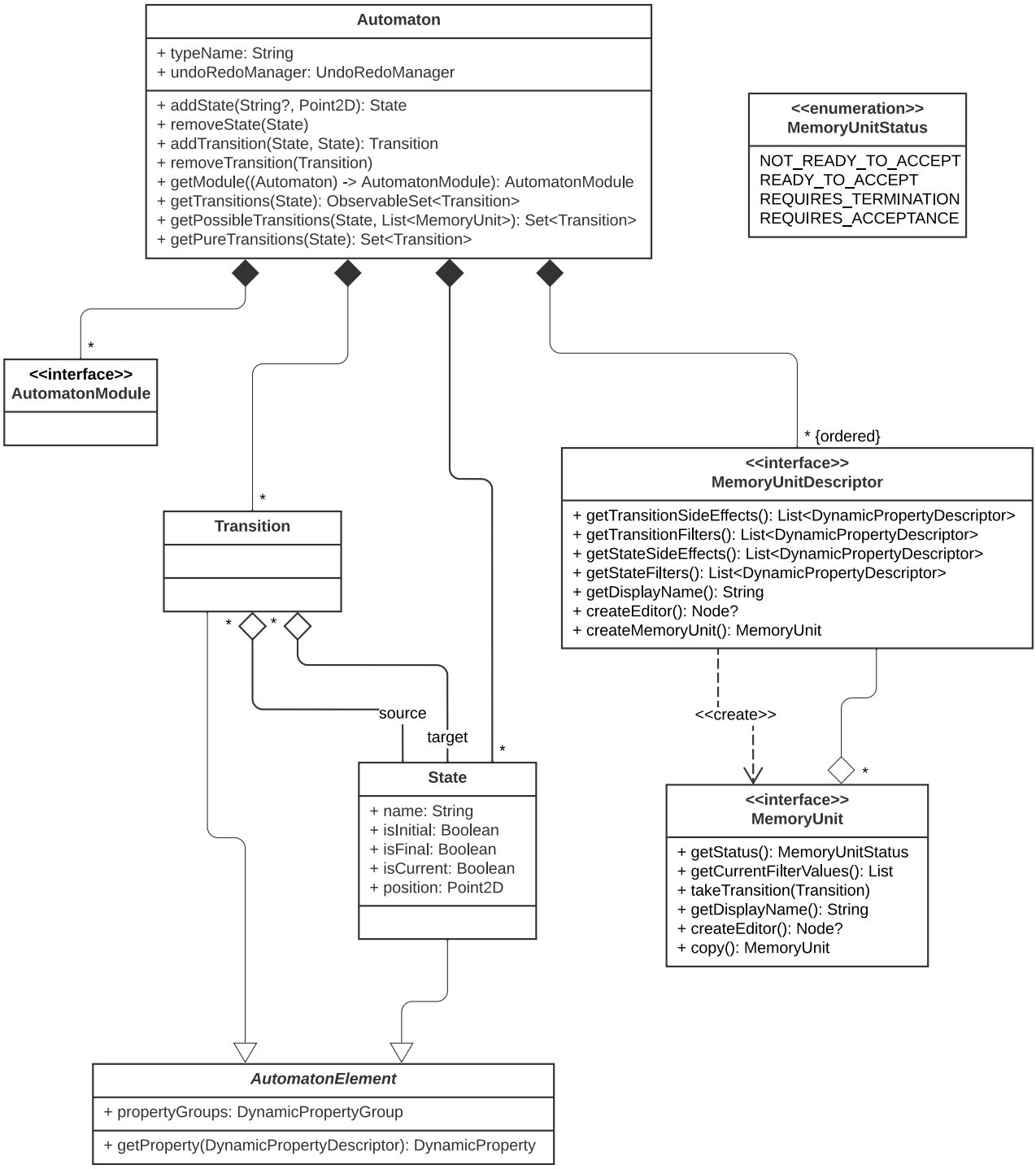

Рис. 5: Диаграмма классов ядра модели

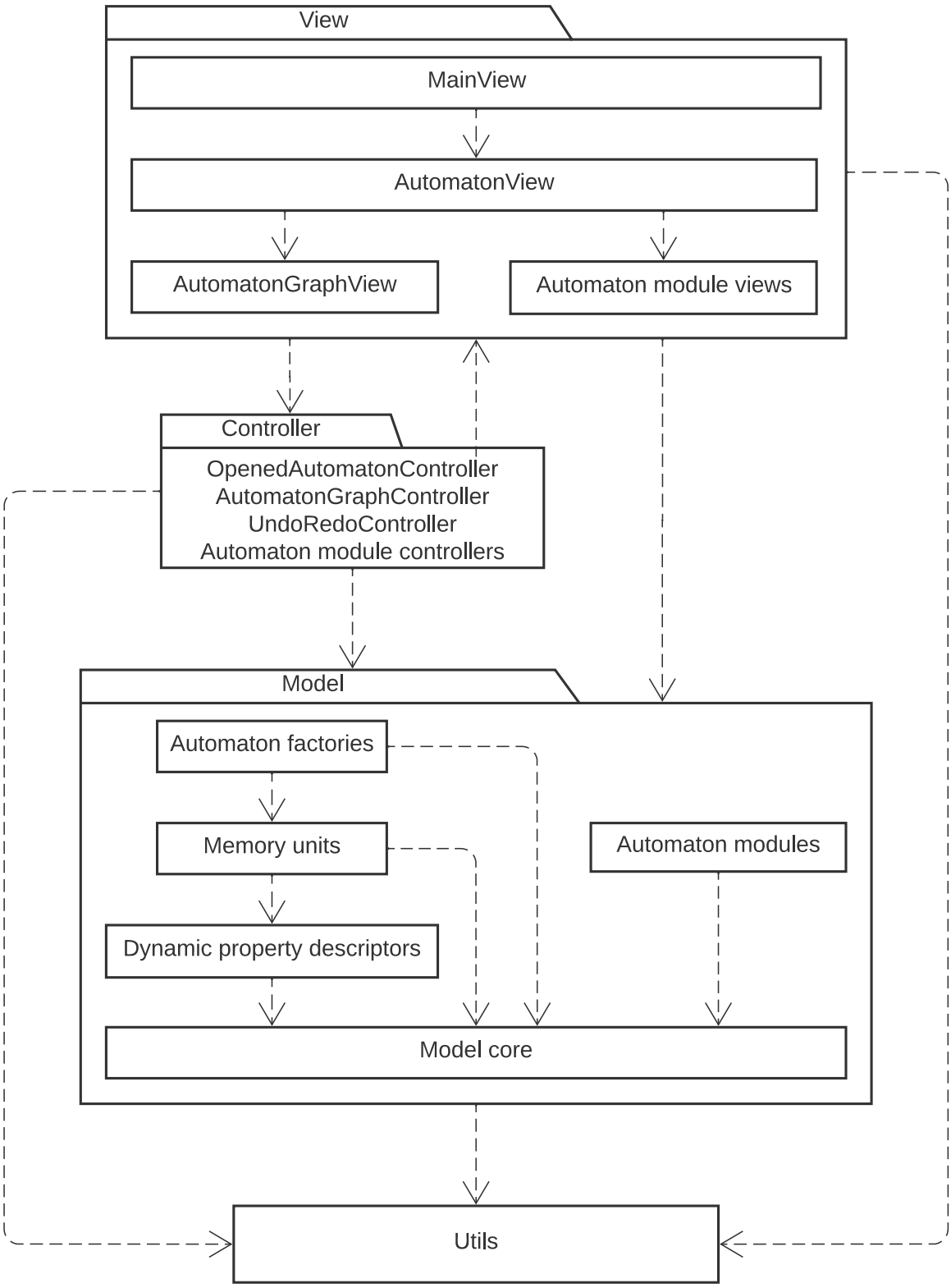

Рис. 6: Диаграмма зависимостей между компонентами «Конструктора вычислителей»

5 Реализация

5.1 Отладка недетерминированных вычислителей

Для упрощения процесса отладки недетерминированных вычислителей были добавлены всплывающие подсказки, появляющиеся при наведении курсора мыши на состояния автомата и показывающие все варианты содержимого памяти автомата для данного состояния (см. рис. 7). Таким образом, благодаря тому, что эти подсказки показываются не всё время, стала возможной демонстрация их на переднем плане без постоянного перекрытия ими самого графа автомата и была решена проблема, выявленная во время обзора automatonsimulator.com [6].

Аналогично JFLAP [8] была реализована возможность замораживать состояния исполнения, что позволило пользователям не тратить время на изучение не интересующих их путей исполнения. Пара кнопок «Freeze» и «Thaw», используемая в JFLAP, для удобства была заменена одним флажком «Frozen».

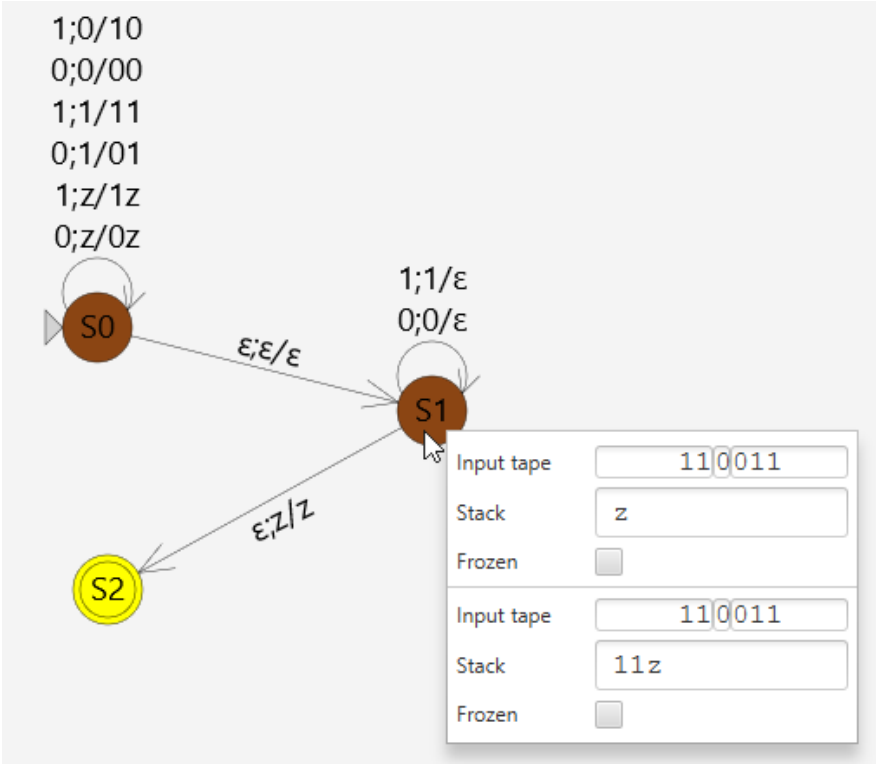

Рис. 7: Всплывающая подсказка с вариантами содержимого памяти для данного состояния

5.1.1 Интерактивное дерево исполнения

Чтобы решить проблему необходимости многократно просматривать совпадающие префиксы трассировок при изучении истории выполнения для нескольких состояний исполнения и проблему непредсказуемости порядка, в котором показываются состояния исполнения в виде списка, было принято решение визуализировать недетерминированное исполнение в виде дерева (или леса, если начальных состояний несколько), представленного на рис. 8, что является подходом, используемым в частности в книге «Введение в теорию автоматов, языков и вычислений» [5], рекомендуемой при изучении предмета «Алгоритмы и анализ сложности».

Функциональность

Дерево исполнения аналогично графу автомата поддерживает увеличение и уменьшение колёсиком мыши и прокрутку левой кнопкой мыши. Также возможно передвижение разделителя для регулировки того, какую часть экрана занимает дерево исполнения, а какую — граф автомата.

При наведении курсора мыши на состояние исполнения, показывается всплывающая подсказка с содержимым памяти автомата в этом состоянии исполнения и флажком для заморозки, видимым для листовых активных состояний исполнения.

Чтобы позволить пользователям изучать отдельные, интересующие их, пути исполнения, была реализована возможность производить шаг исполнения из произвольного листового состояния исполнения с помощью левого клика по нему. Также было поддержано обратное действие — сворачивание поддерева левым кликом по его корню.

Чтобы не дезориентировать пользователя, было реализовано выравнивание, происходящее после любых изменений дерева и обеспечивающее неподвижность последнего кликнутого состояния исполнения или неподвижность корня дерева в случае, если последнее действие — одновременный шаг по состоянию или по замыканию из всех незамороженных состояний исполнения.

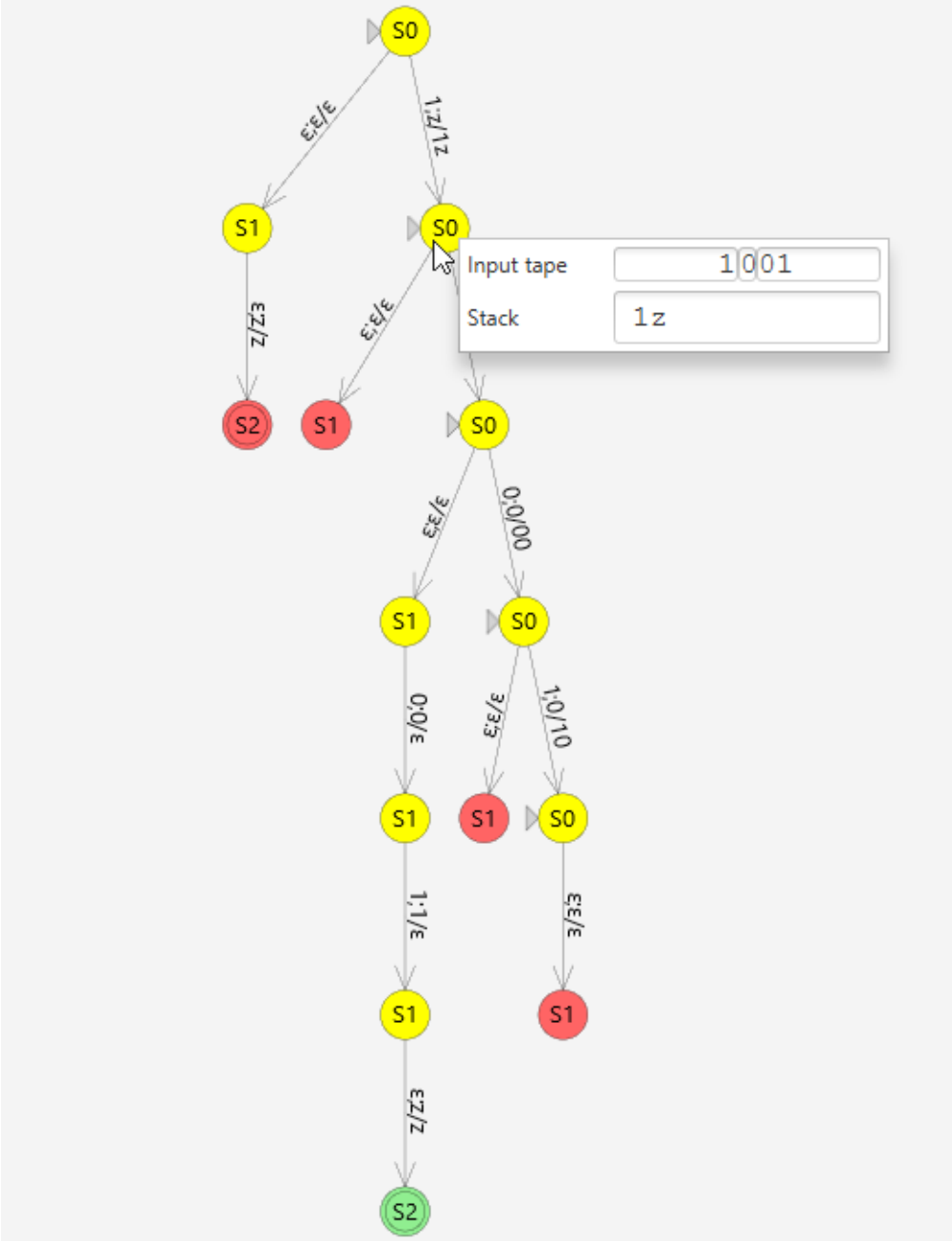

Рис. 8: Интерактивное дерево исполнения

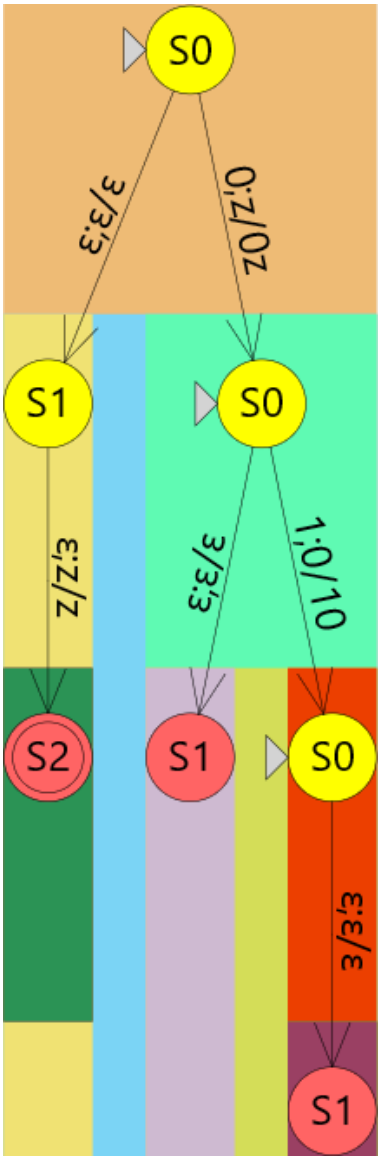

Рис. 9: Интерактивное дерево исполнения с фонами контейнеров раскладки, замененными на случайные цвета

Детали реализации

Для генерации дерева используется древовидная структура из вложенных контейнеров раскладки `HBox` и `VBox`, предоставляемых JavaFX (см. рис. 9).

Для визуализации состояний исполнения и переходов между ними переиспользуются представления состояний и переходов, разработанные ранее для визуального редактора графа автомата.

5.2 Автоматы с дополнительными свойствами состояний

Чтобы поддержать автомат Мура, было реализовано динамическое добавление дополнительных свойств состояний автомата, что было сделано с помощью вынесения уже существующей аналогичной функциональности для переходов автомата в общего предка классов `Transition` и `State` (см. рис. 5).

Для реализации самого автомата Мура, а именно его ленты вывода, был частично переиспользован код ленты вывода автомата Мили, общая для обеих лент вывода часть которого была вынесена в общего предка классов `MealyOutput` и `MooreOutput`.

5.3 Сохранение и открытие автоматов

Для открытия автоматов, сохранения изменений и сохранения автоматов по выбранному пути в меню «File» были добавлены пункты «Open (Shortcut+O)», «Save (Shortcut+S)» и «Save As... (Shortcut+Shift+S)» соответственно, где Shortcut — клавиша-модификатор, обычно используемая в сочетаниях клавиш ОС пользователя (например, Ctrl на Windows и Meta на Mac). Также было добавлено показываемое при закрытии измененного автомата предупреждение, предлагающее сохранить изменения. По умолчанию приложение предлагает сохранять новые автоматы в папку «{user.home}/Documents/automaton-constructor» в файл с именем «Untitled {тип автомата}».

Для сериализации используется библиотека kotlinx.serialization и применяется предлагаемый авторами данной библиотеки паттерн «Суррогат»[5]: во время сериализации сначала сохраняемый автомат конвертируется в ацикличный суррогатный объект, который в свою очередь сериализуется уже автоматически сгенерированным кодом. Десериализация происходит аналогично: автоматически сгенерированный код десериализует суррогатный объект, который конвертируется в откры-

[5]Composite serializer via surrogate — https://github.com/Kotlin/kotlinx.serialization/blob/master/docs/serializers.md#composite-serializer-via-surrogate (дата обращения 12.05.2022).

ваемый автомат.

5.4 Отмена и повтор действий

Для отмены и повтора совершённых действий в меню «Edit» были добавлены пункты «Undo (Shortcut+Z)» и «Redo (Shortcut+Shift+Z)» соответственно.

Действия, которые возможно отменять и повторять:

- создание и удаление состояний и переходов,
- изменение свойств состояний и переходов,
- передвижение состояний.

Отмена и повтор действий согласованы с групповыми операциями, то есть любая групповая операция отменяется и повторяется полностью.

Для согласования с групповыми операциями, в том числе теми, которые состоят из других более мелких групповых операций, например, для удаления нескольких состояний, удаление каждого из которых влечет удаление еще нескольких переходов, реализован `UndoRedoManager`, позволяющий писать код, как в листинге 1.

При вызове функции `f` из листинга 1 в одно композитное действие группируются два действия `action1` и `action2`, при вызове функции `g` — действия `action1`, `action2` и `action3`, а при вызове функции `h` на стек совершённых действий кладутся два действия: одно композитное действие, состоящие из `action1`, `action2` и `action3`, и одно обычное действие `action4`.

По принципу работы `UndoRedoManager` похож на `ReentrantLock`: перед заходом в группу и перед выходом из нее `UndoRedoManager` соответственно инкрементирует и декрементирует свой внутренний счетчик числа объемлющих групп и объединяет все действия внутри одной группы верхнего уровня и её подгрупп в одно композитное действие.

```
fun f() = group {
  perform(action1)
  perform(action2)
}

fun g() = group {
  f()
  perform(action3)
}

fun h() {
  g()
  perform(action4)
}
```

Листинг 1: Пример использования `UndoRedoManager`

6 Тестирование и апробация

6.1 Тестирование

В рамках данной работы было произведено unit-тестирование модели. Покрытие по инструкциям и по веткам[6] составило 90%.

6.2 Апробация

Для проведения апробации была написана пользовательская документация [12], руководствуясь которой потенциальные пользователи приложения, студенты математико-механического факультета СПбГУ, выполнили предложенные им задания в разработанном приложении и оценили его по методике System Usability Scale. Средняя оценка по результатам шести анонимно заполненных опросов составила 75.8, что признается хорошим результатом.

[6]Coverage Counters – https://www.jacoco.org/jacoco/trunk/doc/counters.html (дата обращения 13.05.2022).

7 Заключение

В результате проведенной работы были решены следующие задачи:

1. Произведен обзор используемых аналогами способов отладки недетерминированных вычислителей.
2. Для упрощения отладки недетерминированных вычислителей добавлены всплывающие подсказки и реализовано представление недетерминированных вычислений в виде интерактивного дерева.
3. Поддержаны автоматы с дополнительными свойствами состояний, в частности автомат Мура.
4. Реализованы сохранение и открытие автоматов.
5. Реализованы отмена и повтор совершённых действий, согласованные с групповыми операциями.
6. Обновлена пользовательская документация.
7. Проведены тестирование и апробация.

Репозиторий проекта на GitHub:
https://github.com/spbu-se/KotlinAutomataConstructor.

Список литературы

[1] Alejandro Ferrera. Automata-Visualizer. — URL: https://github.com/AleKiller21/Automata-Visualizer (online; accessed: 2022-18-05).

[2] Carl Burch. Automaton Simulator. — URL: http://www.cburch.com/proj/autosim/ (online; accessed: 2022-18-05).

[3] Dirk Riehle, Michel Tilman, Ralph Johnson. Dynamic Object Model // Pattern Languages of Program Design 5: (Software Patterns) / Ed. by Dragos Manolescu, Markus Völter, James Noble. — Addison-Wesley, 2006. — ISBN: 0321321944. — URL: https://riehle.org/computer-science/research/2005/plopd-5.pdf (online; accessed: 2022-18-05).

[4] Ivan Zuzak, Vedrana Jankovic. FSM simulator. — URL: http://ivanzuzak.info/noam/webapps/fsm_simulator/ (online; accessed: 2022-18-05).

[5] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. Introduction to Automata Theory, Languages, and Computation (2nd Edition). — Addison-Wesley, 2000. — ISBN: 0201441241. — URL: https://www-2.dc.uba.ar/staff/becher/Hopcroft-Motwani-Ullman-2001.pdf (online; accessed: 2022-18-05).

[6] Kyle Dickerson. automatonsimulator.com. — URL: https://automatonsimulator.com/ (online; accessed: 2022-18-05).

[7] Pinaki Chakraborty, P. C. Saxena, C. P. Katti. Fifty years of automata simulation: a review. — 2011. — URL: https://doi.org/10.1145/2038876.2038893 (online; accessed: 2022-18-05).

[8] Susan H. Rodger, Thomas W. Finley. JFLAP – An Interactive Formal Languages and Automata Package. — Sudbury, MA, USA : Jones & Bartlett Publishers, Inc., 2006. — ISBN: 0763738344. — URL: https:

//www.jflap.org/jflapbook/jflapbook2006.pdf (online; accessed: 2022-18-05).

[9] А.Е.Плоскин. Создание приложения для симуляции конечных автоматов и других вычислителей. — 2021. — URL: https://se.math.spbu.ru/thesis/texts/Ploskin_Aleksandr_Evgen%27evich_Bachelor_Report_2021_text.pdf (online; accessed: 2022-17-05).

[10] А.Р.Усманов. Разработка сервиса визуального редактирования и симуляции конечных автоматов (отчёт по весенней практике). — 2021. — URL: https://se.math.spbu.ru/thesis/texts/Usmanov_Artur_Radikovich_Bachelor_Report_2021_text.pdf (online; accessed: 2022-18-05).

[11] А.Р.Усманов. Разработка сервиса визуального редактирования и симуляции конечных автоматов (отчёт по осенней практике, не опубликован). — 2021.

[12] И.В.Муравьев. Пользовательская документация настольной версии «Конструктора вычислителей». — URL: https://docs.google.com/document/d/1jhqQSpF-SMvZJMpAzzRWi49u15uQ_wBPstUS369gO-Y (online; accessed: 2022-18-05).

[13] И.В.Муравьев. Проектирование и реализация конструктора вычислителей с абстрактной памятью. — 2021. — URL: https://drive.google.com/file/d/1sPX4CGHwINEX7otoTbet_GFub7ILjjjm.

[14] Техническое задание на проект «Конструктор вычислителей». — URL: https://docs.google.com/document/d/1MG2lUpvM-c7l8idzfboZMIwJK24I4dQdTPD3zrPxhgU (online; accessed: 2022-18-05).