Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20.Б11-мм

Олейников Андрей Павлович

Реализация поддержки формализмов и дерева вычислений в веб-приложении «Конструктор 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

Конечный автомат — абстрактный автомат [18], имеющий конечное число состояний. Конечные автоматы активно применяются в решении множества задач, например: в сканировании текстовых массивов для поиска заданных шаблонов, в лексических анализаторах, в разработке протоколов. Автоматы являются важной темой в области компьютерных наук, уровень сложности изучения и преподавания которой снижается при использовании утилит, симулирующих исполнение автоматов [10].

На кафедре системного программирования был реализован MVP «Конструктора вычислителей» [19] – сервис визуального редактирования и симуляции конечных автоматов. Продукт представляет ценность для студентов, проходящих курсы «Теория автоматов и формальных языков», «Программирование», «Проектирование программного обеспечения», а также для преподавателей. Инструмент позволяет демонстрировать работу вычислителей и экспериментировать с вычислителями, также он облегчает создание вычислителей, решающих определенные задачи, в конечном итоге, повышая уровень освоения материала.

Текущая версия веб-приложения поддерживает отображение и исполнение ДКА, НКА и ε-НКА. Для увеличения количества тем, проходимых студентами в перечисленных выше курсах, в которых можно было бы использовать «Конструктор вычислителей», требуется поддержка большего количества формализмов, а для упрощения и ускорения процесса поиска ошибки в вычислителе, созданном пользователем, необходим инструмент, позволяющий наблюдать трассировку исполнения, в качестве такого инструмента предполагается использование дерева вычислений.

Таким образом, данная работа посвящена расширению функциональности сервиса поддержкой новых формализмов, преобразований и реализацией отображения дерева вычислений.

2. Постановка задачи

Целью работы является расширение функциональности «Конструктора вычислителей» инструментом, позволяющим отслеживать ход вычислений, поддержкой новых формализмов и их преобразований. Для её выполнения были поставлены следующие задачи:

1. Провести анализ существующих решений по визуализации трассировки исполнения вычислителей и отображению вычислителей и преобразований, перечисленных ниже.
2. Реализовать поддержку следующих вычислителей:
   (a) автомат с магазинной памятью,
   (b) детерминированный автомат с магазинной памятью,
   (c) машина Тьюринга,
   (d) автомат Мили,
   (e) детерминированный автомат Мили,
   (f) автомат Мура,
   (g) детерминированный автомат Мура.
3. Реализовать следующие преобразования автоматов:
   (a) минимизация ДКА,
   (b) преобразование из НКА в ДКА,
   (c) преобразование из автомата Мили в автомат Мура,
   (d) преобразование из автомата Мура в автомат Мили.
4. Реализовать отображение дерева вычислений.
5. Провести тестирование и апробацию реализованных пользовательских возможностей.

3. Обзор

3.1. Обзор существующих решений

На сегодняшний день уже существуют проекты, частично реализующие визуализацию возможностей, описанных в поставленных задачах. Рассмотрим сервисы, наиболее удовлетворяющие запросу к данной работе:

- Automaton simulator [2] имеет поддержку автомата с магазинной памятью, представляет его в виде графа, но не демонстрирует изменение стека при исполнении.

- Симулятор вычислителей Калифорнийского университета в Дейвисе, поддерживающий машину Тьюринга [1], отображает ленту машины и демонстрирует ее изменения на каждом шаге, отображая головку машины.

- Симулятор машины Тьюринга ВМК МГУ им. М.В. Ломоносова [20] - имеет аналогичное представление ленты с лентой машины Тьюринга в симуляторе, описанном выше, но не поддерживает отображение головки вычислителя.

- JFLAP [5] имеет минимизацию ДКА, преобразование НКА к ДКА, поддерживает отображение машин Мили и Мура, машины Тьюринга, автомата с магазинной памятью и позволяет увидеть трассировку исполнения, демонстрируя состояния стека и переходы по ребрам до одного выбранного состояния.

3.2. Используемые технологии

В проекте «Конструктор вычислителей» используется JavaScript библиотека React [11], разработка ведется на языке TypeScript [14], используется библиотека визуальных компонентов Material-UI [8], а также обертка над библиотекой Vis-network [17] – React-graph-vis [12] для отображения и визуального редактирования графа.

4. Ход работы

4.1. Симуляция автоматов

Для реализации поддержки вычислителей были разработаны классы **PDA**, **DPDA**, **TM**, **Mealy**, **DMealy**, **Moore**, **DMoore**.

В них реализованы методы минимизации ДКА [9], преобразование НКА к ДКА [7], преобразования автомата Мили к автомату Мура [3], преобразования автомата Мура к автомату Мили [4]. Схема классов приведена на рис. 1. Данные классы имеют методы для пошагового и мгновенного исполнения вычислителя, которые возвращают объект содержащий данные текущего состояния и историю переходов до этого состояния, используемый для отображения дерева вычислений и истории состояний. Методы, исполняющие ДКА, бросают исключение, если построенный автомат является недетерминированным, что позволяет отобразить ошибку пользователю. Поведение метода минимизации ДКА аналогично, при невозможности минимизации.

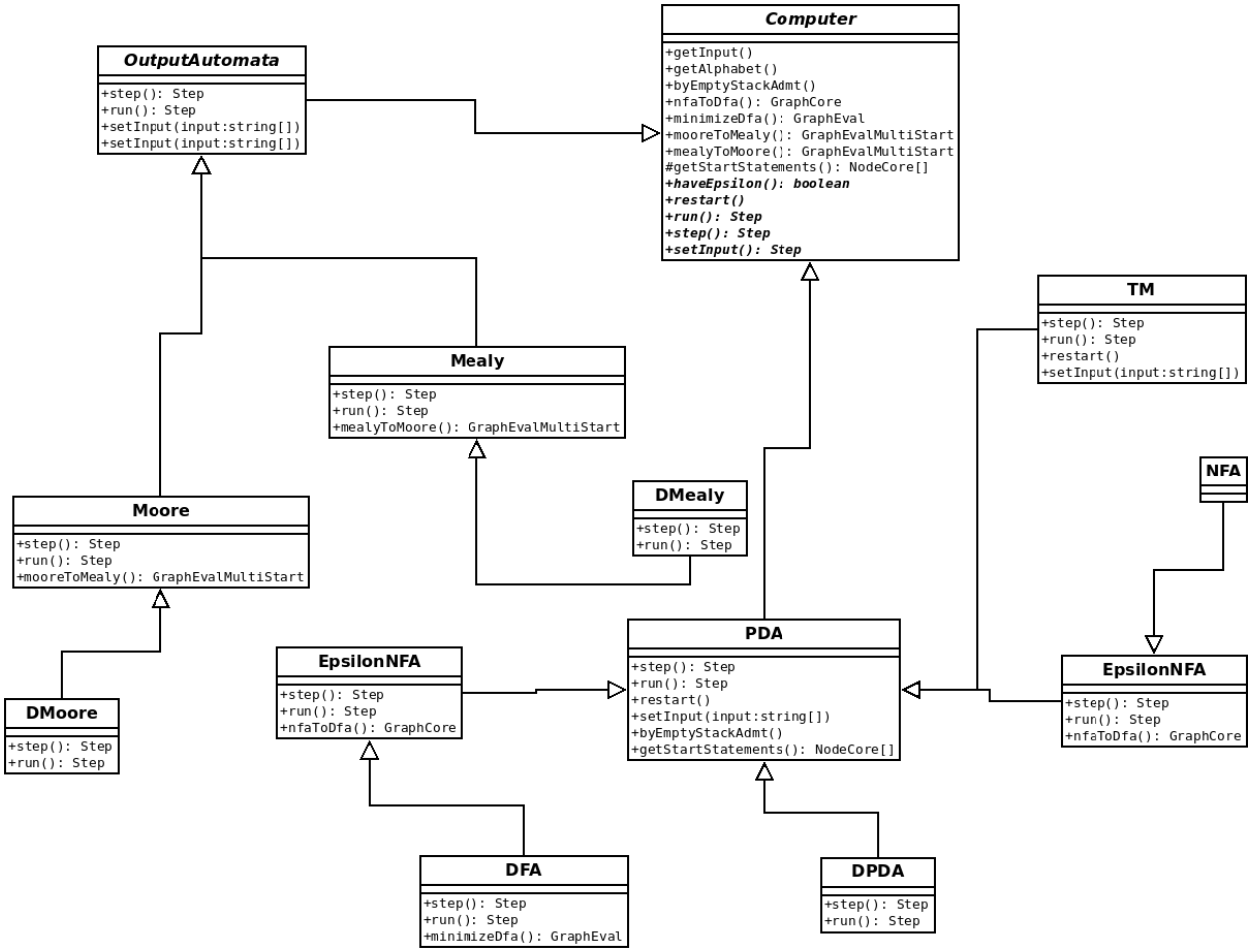

Рис. 1: Диаграмма классов-вычислителей

4.2. Отображение автоматов

MVP поддерживает отображение переходов по буквам из алфавита, что недостаточно для отображения вычислителей, чья поддержка реализуется в данной работе. Для расширения возможности визуализации ребер был спроектирован интерфейс **TransitionParams** (рис. 2).

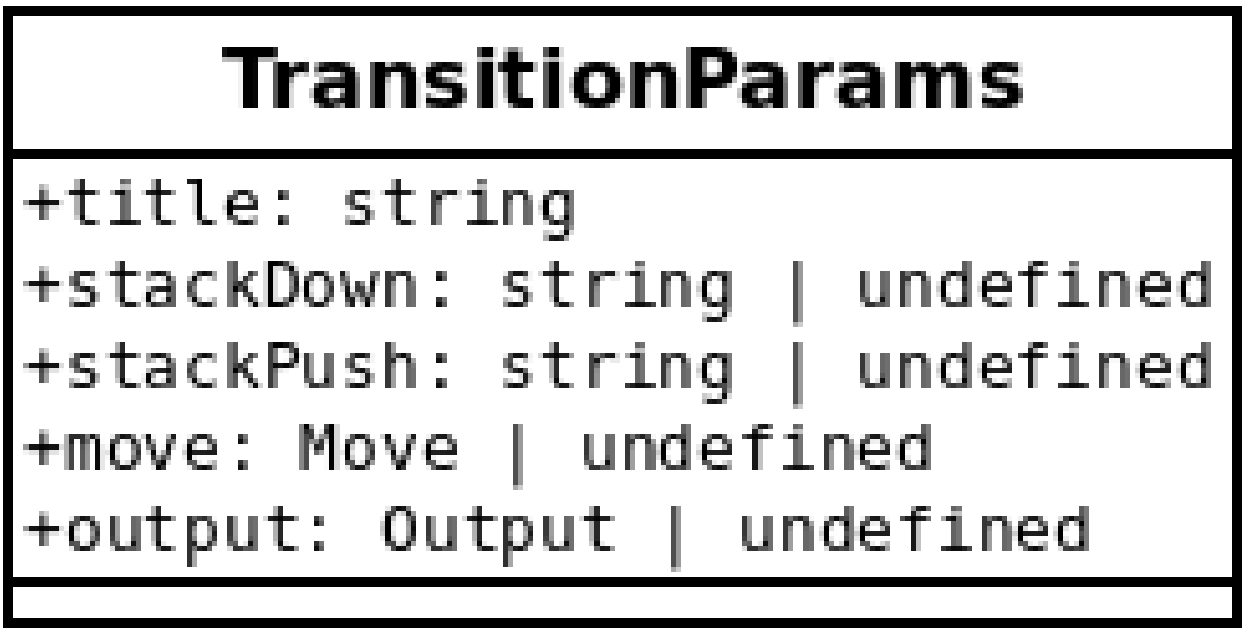

Рис. 2: Методы интерфейса TransitionParams

4.2.1. Отображение автомата с магазинной памятью

Для поддержки автомата с магазинной памятью требовалось отображение стека. Идея визуализации стека заключается в отображении состояния стека в том или ином состоянии на каждом этапе исполнения в блоке истории (рис. 3).

Рис. 3: Отображение стека

Для изменения режима допуска входной последовательности был реализован элемент интерфейса, изображенный на рис. 4.

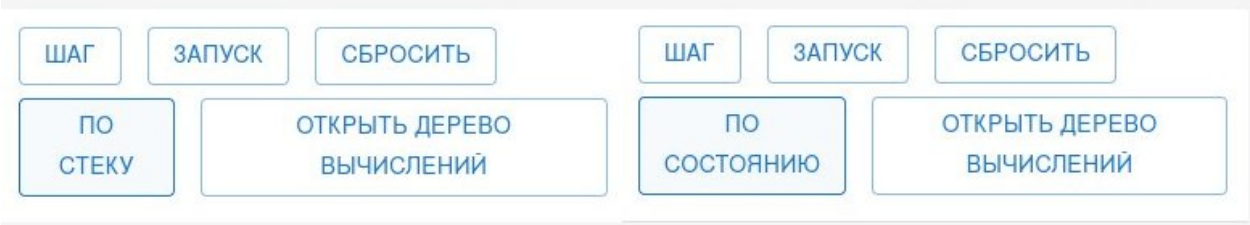

Рис. 4: Переключение допуска

Ребра автомата с магазинной памятью показывают информацию о требуемом алфавитном символе на входе и вершине стека для перехода на следующее состояние, а также список элементов в обратном порядке, которые окажутся на стеке при выполнении условия перехода (рис 5).

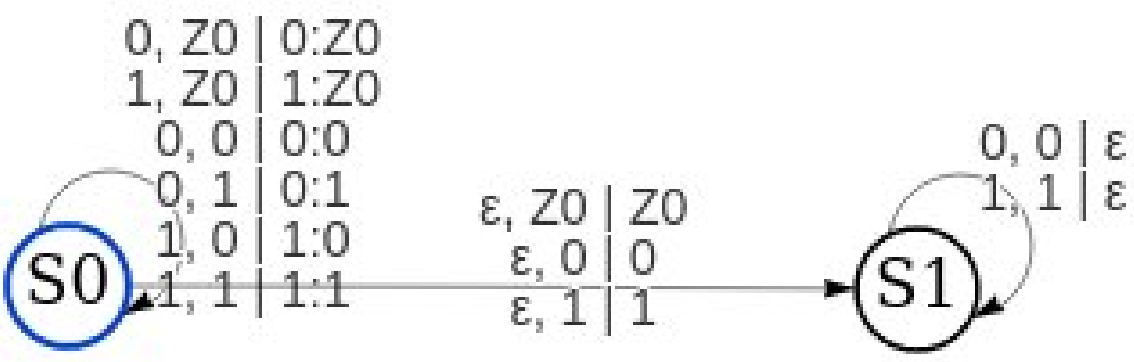

Рис. 5: Ребра автомата с магазинной памятью

4.2.2. Отображение машины Тьюринга

Поддержка машины Тьюринга подразумевала отображение изменений на ленте. Реализацией отображения ленты (рис. 6) является растущий при исполнении список элементов ленты, с которыми взаимодействовала головка машины Тьюринга и одного пустого символа в конце. По мере исполнения окно, отображающее кусок ленты, автоматически смещается вслед за головкой.

Рис. 6: Лента машины Тьюринга

Ребра машины Тьюринга показывают информацию о требуемом элементе памяти, считываемым головкой машины для перехода на следующее состояние, об элементе, записываемом на ленту, и о направлении движения головки при переходе на следующее состояние (рис 7).

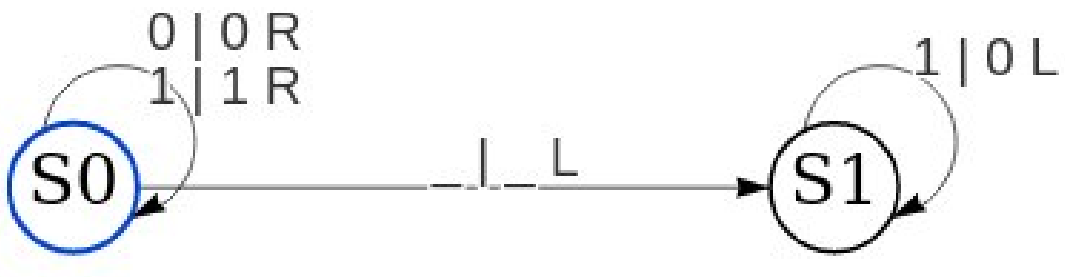

Рис. 7: Ребра машины Тьюринга

4.2.3. Отображение автомата Мили

Отображение ребер (рис 8) автомата Мили показывают информацию о символе ввода, требуемом для перехода на следующее состояние и сигнале, отображаемом в блоке истории (рис 9) после каждого посещенного состояния в процессе исполнения.

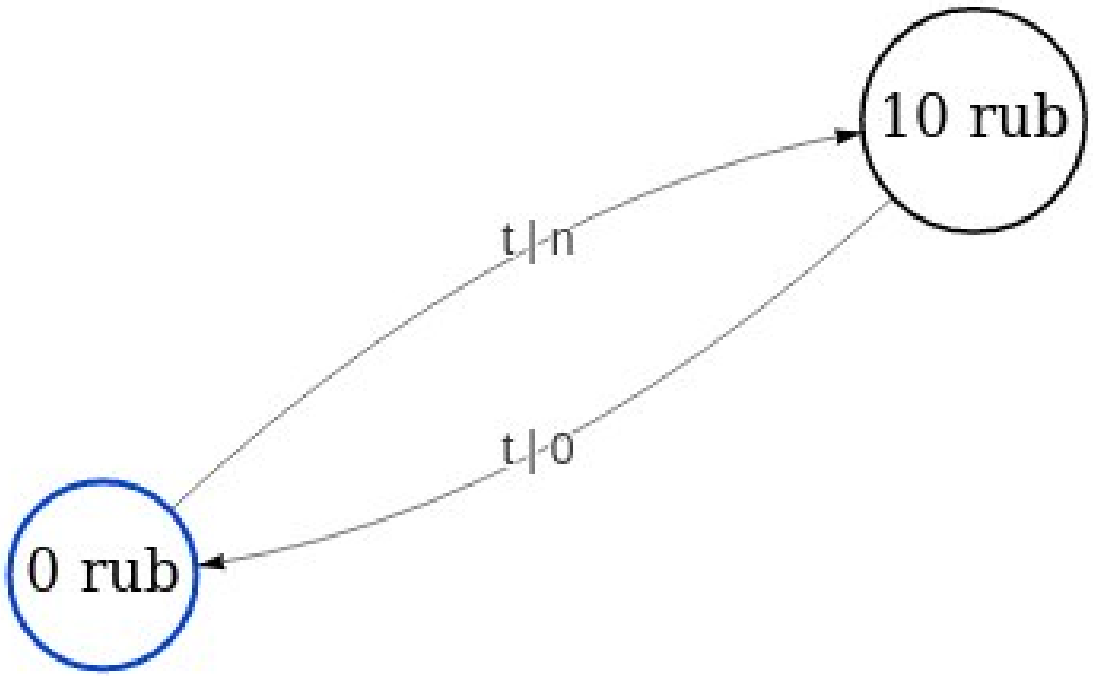

Рис. 8: Ребра автомата Мили

Рис. 9: Сигналы автомата Мили

Для возможности создания автомата Мура по, построенному пользователем, автомату Мили был реализован элемент интерфейса изображенный на рис. 10.

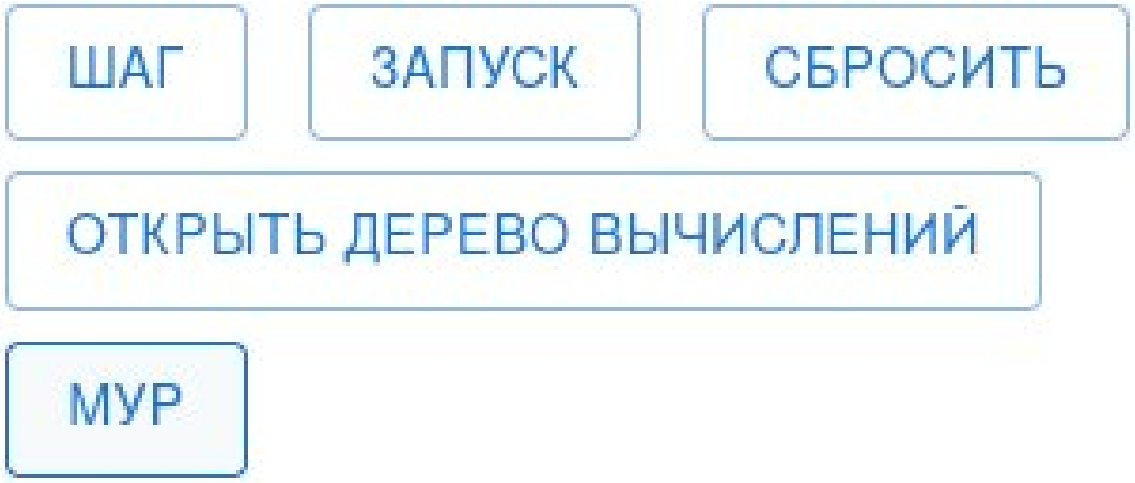

Рис. 10: Кнопки автомата Мили

4.2.4. Отображение автомата Мура

Ребра машины Мура не хранят ничего, кроме символа алфавита для перехода на следующее состояние. Однако в этом вычислителе необычное, по сравнению с остальными, представление вершин. Вершины (рис. 11) демонстрируют выходной сигнал, который отображается в блоке истории, как в автомате Мили.

Задача интерфейса преобразования автомата Мили к автомату Мура решена аналогичным способом, описанном в прошлом параграфе.

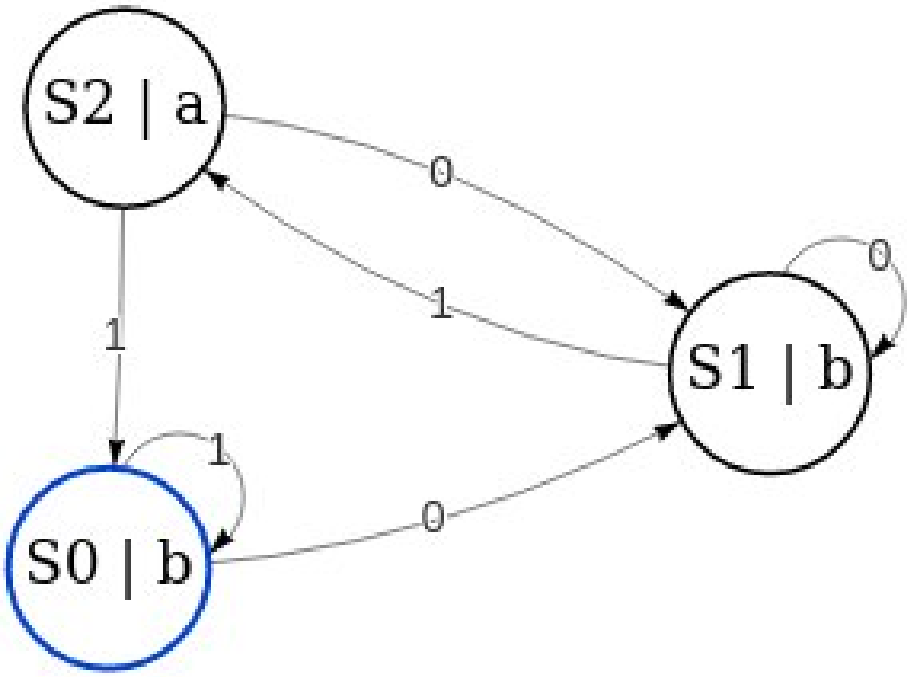

Рис. 11: Вершины автомата Мура

4.3. Оповещение об ошибке

Исполнение детерминированных вычислителей может привести к ошибке, если построенный автомат является недетерминированным, а минимизация ДКА является невозможной, если существует хотя бы одно состояние с меньшей степенью исходящих вершин, чем мощность алфавита. Поэтому было реализовано предупреждение об ошибке (рис. 12), показываемое пользователю при попытке исполнить недетерминированный автомат, начав работу с любым детерминированным автоматом или при попытке минимизировать ДКА, не удовлетворяющий условию, описанному выше.

Рис. 12: Уведомления об ошибках

4.4. Отображение дерева вычислений

Дерево вычислений является трассой исполнения вычислителя. Поскольку в процессе исполнения недетерминированный вычислитель может находиться во множестве вершин одновременно, такое дерево способно стремительно расширяться, что порождает необходимость в навигации по нему. Эта задача уже решена для визуализации вычислителя с помощью React-graph-vis. Однако при взаимодействии с вычислителем возникает неожиданное изменение положения вычислителя на экране, такая же проблема возникла и при визуализации дерева с помощью React-graph-vis.

React-graph-vis является оберткой над библиотекой Vis-network. Vis-network позволяет отображать графы, используя холст HTML. Создание сети происходит с помощью конструктора Network, принимающего следующие параметры:

- container – ссылка на HTML-элемент представляющий контейнер для сети,
- data – данные сети.

Важно заметить, что в качестве параметра data, может быть передан DataSet [16] из библиотеки Vis-data [15], позволяющий работать сети с динамическими данными.

Просмотрев исходники React-graph-vis стало понятно, что данная обертка пересоздает сеть при каждом её обновлении, что приводит к изменению положения вычислителя на экране. Для исправления такого поведения и переиспользования решения для отображения дерева вычислений было принято решение отказаться от обертки. Такое решение

привело к освобождению от ограничения количества функций взаимодействия с сетью и элементарной реализации поддержки контекстного меню (рис. 13).

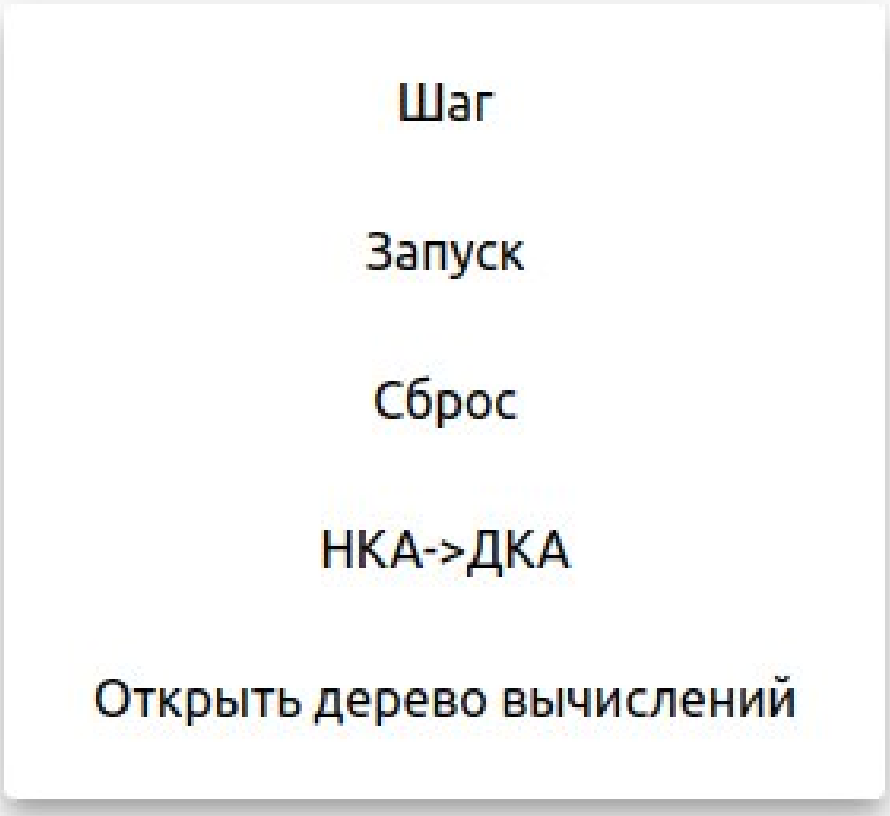

Рис. 13: Пример контекстного меню

Реализация отображения дерева вычислений представляет трассировку исполнения с посчитанными ε-замыканиями [6] (рис. 14). Для взаимодействия с окном дерева вычислений был реализован элемент интерфейса изображенный на рис. 15.

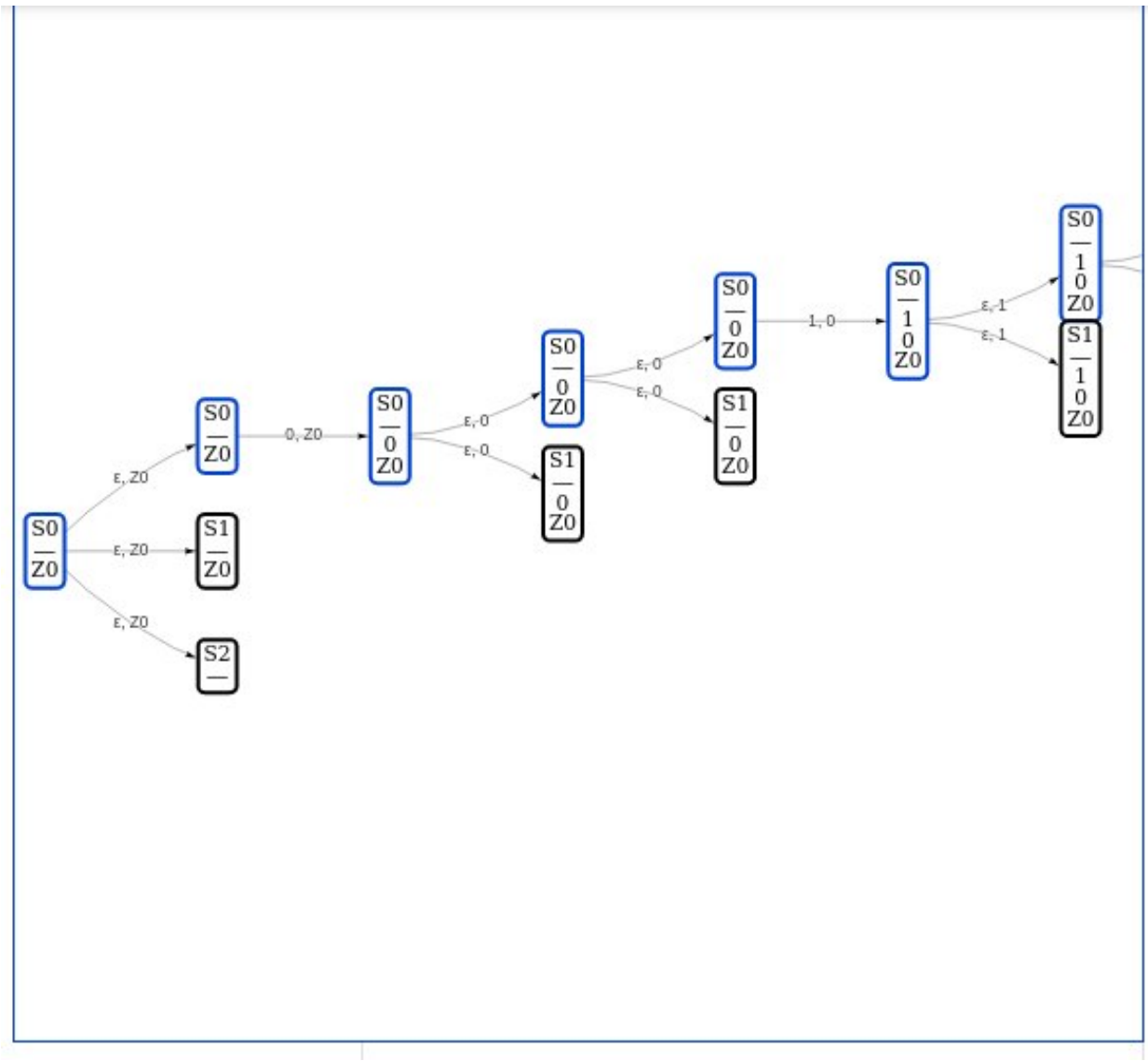

Рис. 14: Дерево вычислений

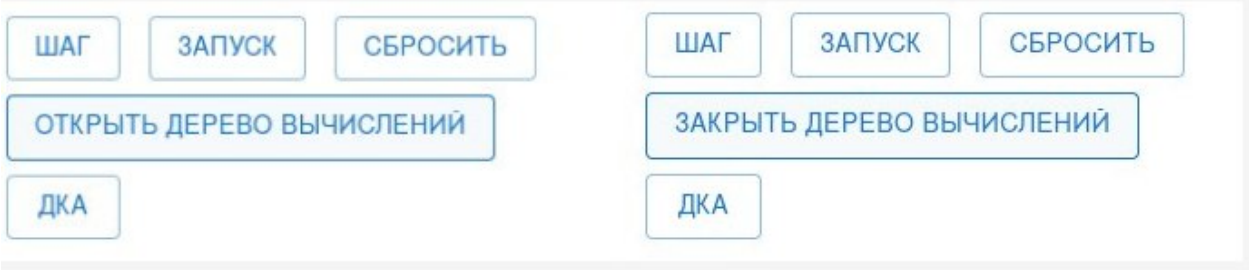

Рис. 15: Кнопка взаимодействия с окном дерева вычислений

5. Тестирование и апробация

5.1. Тестирование

В процессе разработки для проверки корректности работы методов, не влияющую на визуализацию, были созданы юнит-тесты. В тестировании вычислителей были использованы классические задачи, такие как инкремент на машине Тьюринга, проверка на полином в автомате с магазинной памятью, автомат, выдающий сдачу, на автомате Мили.

5.2. Апробация

После реализации новых возможностей сервиса была проведена их апробация с использованием методики оценивая пользовательского интерфейса SUS [13]. Девять респондентов получили анкету, состоящую из заданий, связанных с применением пользовательского интерфейса и десяти вопросов, характеризующих удобство пользования сервисом. После был проведен анализ ответов, в результате которого был высчитан средний бал SUS – 73.1. Полученный результат эквивалентен оценке B- (хорошо).

Таким образом, взаимодействие пользователя с новыми вычислителями, их преобразованиями и инструментом для отслеживания трассы исполнения является удобным, но имеет место для улучшений.

6. Заключение

В результате работы были решены следующие задачи:

1. Проведен анализ существующих решений по визуализации новых вычислителей, преобразований и трассировки исполнения.
2. Реализована поддержка следующих вычислителей:
   (a) автомат с магазинной памятью,
   (b) детерминированный автомат с магазинной памятью,
   (c) машина Тьюринга,
   (d) автомат Мили,
   (e) детерминированный автомат Мили,
   (f) автомат Мура,
   (g) детерминированный автомат Мура.
3. Реализованы следующие преобразования автоматов:
   (a) минимизация ДКА,
   (b) преобразование из НКА в ДКА,
   (c) преобразование из автомата Мили в автомат Мура,
   (d) преобразование из автомата Мура в автомат Мили.
4. Реализовано отображение дерева вычислений.
5. Проведено тестирование и апробация реализованных функциональностей.

Веб-приложение:
https://spbu-se.github.io/WebAutomataConstructor/
Исходный код:
https://github.com/spbu-se/WebAutomataConstructor/

Список литературы

[1] Automaton Simulator. — URL: https://web.cs.ucdavis.edu/~doty/automata/ (online; accessed: 2022-05-23).

[2] Automatonsimulator. — URL: https://automatonsimulator.com/ (online; accessed: 2022-05-23).

[3] Conversion From Mealy to Moore Machine. — URL: https://cstaleem.com/conversion-from-mealy-to-moore-machine (online; accessed: 2022-05-23).

[4] Conversion From Moore to Mealy Machine. — URL: https://cstaleem.com/conversion-from-moore-to-mealy-machine (online; accessed: 2022-05-23).

[5] JFLAP. — URL: https://www.jflap.org/ (online; accessed: 2022-05-23).

[6] JOHN E. HOPCROFT RAJEEV MOTWANI JEFFREY D. ULLMAN. Что такое ε-замыкание // Introduction to Automata Theory, Languages, and Computation SECOND EDITION. — 2008. — P. 91.

[7] JOHN E. HOPCROFT RAJEEV MOTWANI JEFFREY D. ULLMAN. Эквивалентность детерминированных и недетерминированных конечных автоматов // Introduction to Automata Theory, Languages, and Computation SECOND EDITION. — 2008. — P. 77–83.

[8] Material-UI: A popular React UI framework. — URL: https://material-ui.com/ (online; accessed: 2022-05-23).

[9] Minimize DFA. — URL: https://research.cs.vt.edu/AVresearch/openalgoviz/JFLAP/trunk/DOCS/gui.minimize.MinimizePane.html (online; accessed: 2022-05-23).

[10] P. Chakraborty P.C. Saxena C.P. Katti. Fifty Years of Automata Simulation: A Review. — 2011. — URL: https://users.cs.duke.edu/

~rodger/jflappapers/ChakrabortyX2011.pdf (online; accessed: 2022-05-23).

[11] React - A JavaScript library for building user interfaces. — URL: https://reactjs.org/ (online; accessed: 2022-05-23).

[12] React-graph-vis. GitHub. — URL: https://github.com/crubier/react-graph-vis (online; accessed: 2022-05-23).

[13] The System Usability Scale How It's Used in UX. — URL: https://xd.adobe.com/ideas/process/user-testing/sus-system-usability-scale-ux/ (online; accessed: 2022-05-23).

[14] TypeScript: Typed JavaScript at Any Scale. — URL: https://www.typescriptlang.org/ (online; accessed: 2022-05-23).

[15] Vis-data. — URL: https://visjs.github.io/vis-data/data/index.html (online; accessed: 2022-05-23).

[16] Vis-data: DataSet. — URL: https://visjs.github.io/vis-data/data/dataset.html (online; accessed: 2022-05-23).

[17] Vis-network. — URL: https://visjs.github.io/vis-network/docs/network/ (online; accessed: 2022-05-23).

[18] Wikipedia. Abstract machine. — URL: https://en.wikipedia.org/wiki/Abstract_machine (online; accessed: 2022-05-23).

[19] А.Р. Усманов. Разработка сервиса визуального редактирования и симуляции конечных автоматове. — URL: https://se.math.spbu.ru/thesis/texts/Usmanov_Artur_Radikovich_Bachelor_Report_2021_text.pdf (online; accessed: 2022-05-23).

[20] Эмулятор машины Тьюринга. — URL: https://cmcmsu.info/1course/alg.schema.mt.htm (online; accessed: 2022-05-23).