Санкт-Петербургский государственный университет

Кафедра системного программирования

Плоскин Александр Евгеньевич

Создание приложения для симуляции конечных автоматов и других вычислителей

Отчет по учебной практике

Научный руководитель:
доцент кафедры СП, к.т.н. Ю. В. Литвинов

Санкт-Петербург
2021

Оглавление

Введение

Задачи моделирования абстрактных вычислителей и симуляции их работы часто возникают в образовании, например, при преподавании курсов «Программирование», «Проектирование программного обеспечения» и наиболее широко в курсе «Теория автоматов и формальных языков». Согласно протоколу оценки сформированности компетенций СВ.5006.2017 от 12 апреля 2021 года уровень усвоения материала по дисциплине «Теория формальных языков и трансляций», в курсе изучения которой также изучаются абстрактные вычислители, объективно низок. Использование инструмента, иллюстрирующего различные вычислители, могло бы положительно сказаться на результатах преподавания. Так, например, согласно статье «Automaton Simulator» [2] использование симулятора автоматов имело положительный опыт при преподавании теоретической информатики в Словацком Техническом Университете в Братиславе. Помимо использования в образовании, инструмент для симуляции вычислителей может иметь применение в промышленных проектах с целью быстрой разработки конечных автоматов и генерации по ним кода. Ни одно из существующих решений для визуализации вычислителей не предлагает всей функциональности, которая необходима для удобной работы в современном понимании. Так, например, возможность использования пользовательских аккаунтов и полноценной совместной разработки не внедрена ни в одно из представленных на рынке приложений. Из этих соображений, на кафедре системного программирования разрабатывается проект «Конструктор вычислителей», результатом которого должно стать приложение, позволяющее разрабатывать, визуализировать и симулировать исполнение конечных автоматов различного вида, машин Тьюринга и других автоматоподобных формализмов, а так же поддерживающее пользовательские аккаунты, коллаборативную разработку, различные аналитические операции и возможность построения вычислителей из уже разработанных «строительных блоков». Предполагается существование двух, возможно полностью независимых друг от друга, версий инструмента — про-

граммы для операционной системы Windows и веб-приложения. Разработка каждой из версий является совместной. Эта работа посвящена разработке десктопной части[1] проекта «Конструктор вычислителей».

[1]Десктопное приложение — программа, работающая на компьютере пользователя

1. Постановка задачи

Целью данной работы в проекте стало создание минимального полезного продукта — настольного приложения, поддерживающего визуальное задание и симуляцию ДКА, НКА и эпсилон-НКА[2].

Были поставлены следующие задачи:

1. Провести анализ существующих решений и выявить детали интерфейса, общие для всех визуализаторов вычислителей.

2. Изучить библиотеки для визуализации графов, соответствующие критериям, и выбрать наилучший вариант для разработки.

3. Продумать архитектуру приложения и в её рамках разработать компоненты, реализующие требуемую от приложения функциональность, такую как:

   - визуальное задание вычислителя в графовом представлении на сцене с поддержкой увеличения/уменьшения;
   - визуализация пошагового исполнения автомата на строке;
   - диагностика ошибок в вычислителе и вывод его типа;
   - создание автоматических тестов с возможностью как одиночного, так и пакетного запуска;
   - сохранение и загрузка как вычислителей, так и тестов.

4. Разработать пользовательский интерфейс, поддерживающий локализацию на русском и английском языках, предоставляющий пользователю доступ ко всем возможностям программы.

5. Написать пользовательскую документацию.

6. Провести тестирование и апробацию.

[2]Формальное определение различных видов конечных автоматов вы можете найти в статье по ссылке: https://www.geeksforgeeks.org/introduction-of-finite-automata/ (дата обращения: 30.05.2021).

2. Обзор литературы и существующих решений

В этом разделе приведён обзор наиболее интересных приложений, решающих схожие задачи, а так же инструментов и библиотек, использованных во время разработки.

2.1. Обзор аналогов

На данный момент создано и применяется великое множество разнообразных инструментов, решающих схожие задачи. Однако по тем или иным причинам ни один из них не удовлетворяет всем требованиям. Рассмотрим те решения, которые оказали наибольшее влияние на разработку этого проекта.

- JFLAP согласно статье «Fifty Years of Automata Simulation: A Review» [13] является наиболее полным инструментом. Помимо всевозможных абстрактных вычислителей, поддерживает также регулярные выражения и контекстно-свободные грамматики [9]. Не поддерживает ни совместную разработку, ни создание автоматов из «строительных блоков», поэтому не может считаться полным аналогом для проекта «Конструктор вычислителей». Не находится в состоянии активной разработки с 2008 года и на данный момент является явно устаревшим решением. В силу его популярности, одним из технических требований в проекте является поддержка формата сохранений JFLAP;

- AutomataLib — библиотека от Технического университета Дортмунда [10], поддерживает моделирование различных автоматов, сохранение и загрузку в формате GraphViz. Не поддерживает большую часть требуемых в проекте вычислителей, например Машину Тьюринга;

- Automaton Simulator поддерживает ДКА, НКА и МП-автоматы [7]. Позволяет визуально редактировать автоматы, симулировать

пошаговое исполнение, а так же создавать автоматические тесты. Не поддерживает Машину Тьюринга и прочие требуемые в проекте вычислители;

- Automaton Simulator от Калифорнийского университета [17] поддерживает только текстовое задание вычислителя, однако имеет красивую визуализацию пошагового исполнения в табличном виде;

- FSM Simulator позволяет строить конечные автоматы по регулярным выражениям и симулировать работу в пошаговом режиме с визуализацией на автомате в графовом представлении [8]. Не позволяет задавать автоматы на сцене.

Несмотря на то, что вышеперечисленные инструменты не удовлетворяют всем требованиям из разрабатываемого проекта, было необходимо учесть как опыт взаимодействия с пользователем при симуляции, так и наследовать визуальным традициям (например, двойная обводка для принимающих состояний), общим для большинства существующих визуальных представлений вычислителей.

2.2. Используемые технологии

Согласно техническому заданию [21], разработка приложения велась на платформе .NET 5. Эта платформа предоставляет возможность создания приложений с графическим интерфейсом при помощи системы WPF [6]. В силу популярности и наличия хорошей документации был выбран C# как основной язык разработки. Для непрерывной интеграции был выбран веб-сервис Appveyor, так как он предоставляет возможность бесплатного использования.

2.3. WPF

Библиотека WPF позволяет создавать графические пользовательские интерфейсы для программ для операционной системы Windows.

WPF позволяет разрабатывать приложение, используя раздельно разметку графического представления на языке XAML [19] и программную часть, определяющую логику работы приложения на любом из языков, поддерживаемых на .NET. Пользовательский интерфейс состоит из элементов управления, называемых в терминах работы с WPF контролами. Каждый контрол имеет внешний вид, который может быть задан некоторым шаблоном или собственным свойством контрола и изменен внешними действиями благодаря механизму триггеров[3]. Связь между данными и их визуальным отображением реализуется при помощи паттерна MVVM [18], а для поддержки действий пользователя на интерфейсе используется поведенческий шаблон проектирования «Команда» [3].

2.3.1. MVVM

Согласно паттерну MVVM код делится на три части:

- View включает в себя части реализации непосредственно отвечающие за внешнее представление. В данном случае это все страницы, написанные на языке разметки XAML и часть логики, которая непосредственно отвечает за отображение;
- Model содержит в себе данные и логику работы с ними.
- ViewModel благодаря механике привязки данных от свойства модели представления к свойству контрола [4] обеспечивает связь между данными и визуальным представлением.

2.3.2. Команда

Паттерн «Команда» играет важную роль для реализации конкретных модулей приложения. Команды — объекты, реализующие общий интерфейс для взаимодействия и предоставляющие метод, отвечающий

[3]Триггер — действие, изменяющее свойства контрола. Является ответом на некоторое событие. Ознакомиться подробнее можно по ссылке: https://metanit.com/sharp/wpf/10.2.php (дата обращения: 30.05.2021).

за исполнение команды и метод, который позволяет узнать, может ли команда быть исполнена. Каждая модель представления имеет набор команд, которые привязываются к конкретным контролам визуального представления, например, каждая кнопка имеет свойство Command, к которому привязывается команда, которая будет исполнена при нажатии на кнопку.

2.4. Обзор решений для визуализации графов

Наиболее существенной задачей при разработке на данном этапе стало создание визуального редактора для вычислителей в графовом представлении. Для этого было необходимо сделать обзор библиотек для визуализации графов, поддерживаемых на платформе .NET, и выбрать наиболее подходящую. Рассмотрим наиболее интересные варианты:

- MSAGL [11] — активно разрабатываемая Microsoft библиотека для визуализации графов, более ранние версии которой называются GLEE. Позволяет как визуализировать существующие графы, так и манипулировать с объектами на сцене. Могла бы быть использована в проекте, однако, по причине плохой документации и наличия некоторых существенных проблем при использовании (например, полная перерисовка графа после взаимодействия с ним), решение которых заняло бы большое количество времени, была отвергнута;

- Graphviz [1] — хороший инструмент для рисования графа по заданным в некотором формате данным, однако не позволяет работать с объектами визуализации, а поэтому не подходит для создания визуального редактора;

- Graph# [5] поддерживает как визуализацию графа, так и редактирование его на сцене. Использует библиотеку графовых алгоритмов Quickgraph как логическое ядро, что позволяет без дополнительных преобразований использовать множество алгоритмов.

Помимо этого, имеет множество встроенных алгоритмов визуализации графов, а так же предоставляет возможность создания и использования своих алгоритмов;

- GraphX [14] является наследницей Graph# и активно переиспользует её код, а потому, всё сказанное выше относительно Graph# верно и для GraphX. Помимо этого, имеет удобную сцену с поддержкой перемещения и увеличения/уменьшения. Имеет лицензию Apache 2.0 и может быть использована в проекте.

Таким образом, выбор библиотеки для визуализации графов пал на GraphX. Помимо вышеупомянутых соображений, в техническом задании есть прямое указание использовать GraphX, чтобы иметь возможность переиспользовать код проекта REAL.NET [20]. Стоит отметить и то, что GraphX имеет ряд недостатков, например, некорректное отображение рёбер с совпадающими началом и концом, которые должны будут найти решение путём изменения исходного кода библиотеки.

3. Архитектура

Приложение состоит из нескольких модулей, каждый из которых реализует часть функциональности приложения, которая не зависит напрямую от других модулей.

Диаграмма классов приложения представлена на Рис. 3. Можно выделить несколько основных модулей:

- сцена предоставляет пользователю визуальный редактор, на котором он может редактировать граф, который используется другими модулями. Также, на сцене визуализируется пошаговое исполнение автомата на строке. Сцена имеет несколько вспомогательных зависимых компонентов, отвечающих за диагностику ошибок, определение типа вычислителя и выбор инструмента для редактирования;

- исполнитель предоставляет пользователю возможность ввести строку и исполнить вычислитель на ней в мгновенном или пошаговом режиме;

- тестовая панель содержит в себе множество тестов, которые могут быть запущены, а также инструментарий для их создания и удаления.

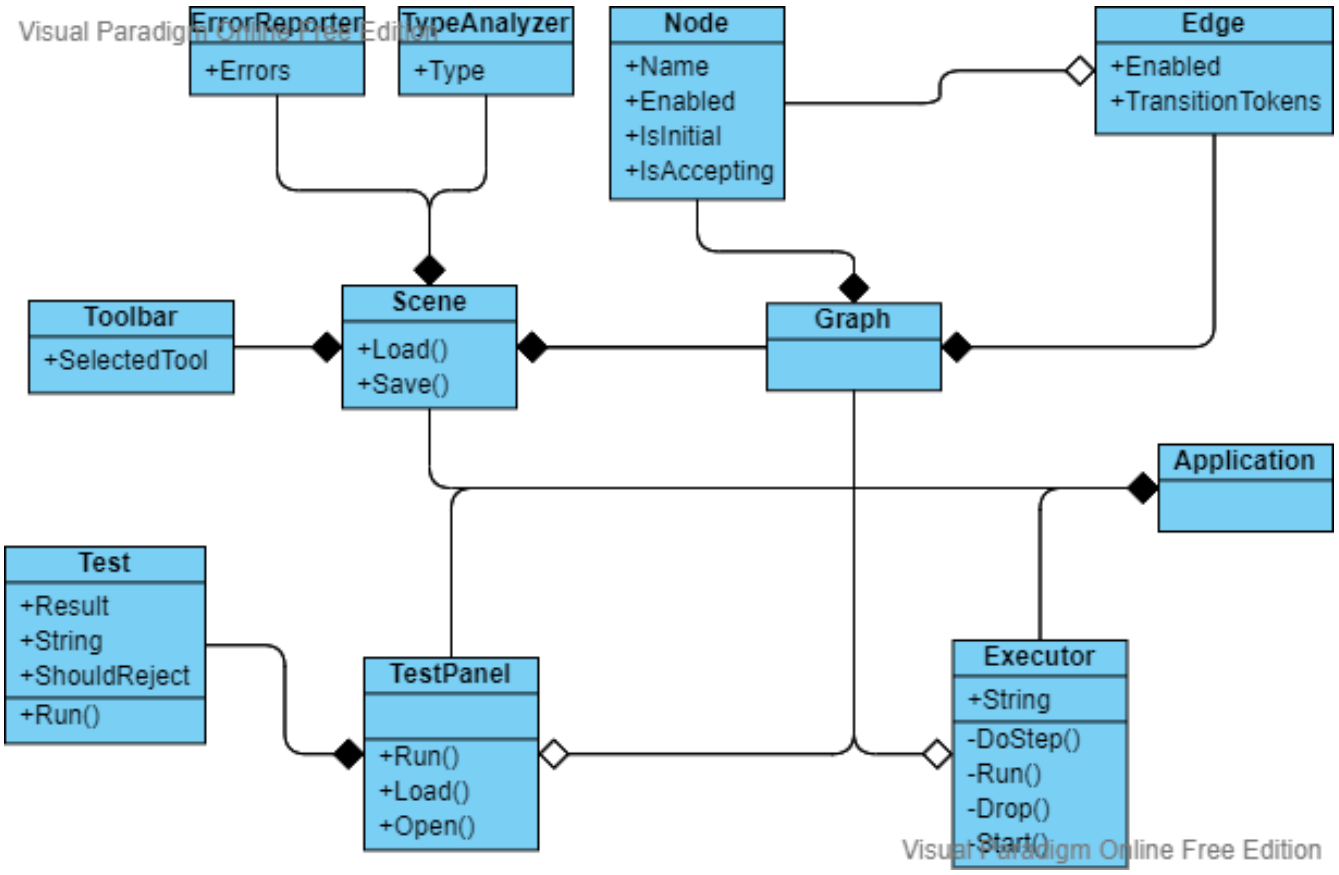

Рис. 1: Диаграмма классов приложения

4. Реализация

В рамках описанной архитектуры были разработаны несколько независимых компонентов, решающих наиболее важные задачи, поставленные перед программой.

4.1. Визуальный редактор

С использованием библиотеки GraphX был реализован визуальный редактор, предоставляющий пользователю возможность размещать на безграничной сцене специфичные для конкретного вычислителя компоненты или удалять их. Атрибуты компонентов вычислителя редактируются прямо на сцене. Сцена работает в трёх режимах: удаление, создание и перемещение, переключение между которыми доступно либо в панели инструментов, либо по горячей клавише. На сцену также выводится информация об ошибках в созданном вычислителе и о его типе.

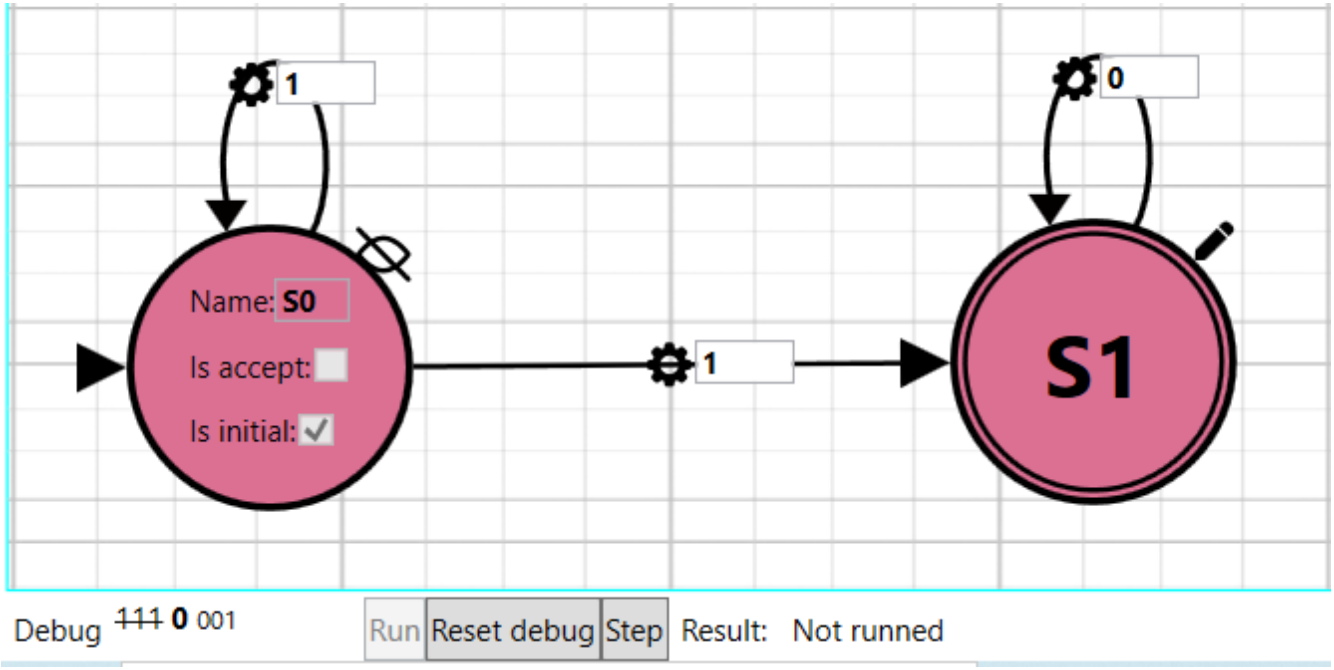

Рис. 2: Визуализация исполнения автомата

При дальнейшей разработке проекта, созданный визуальный редактор может быть применён к любому вычислителю. Для этого, достаточно только поменять контролы вершин и рёбер на соответствующие.

4.2. Симулятор исполнения

Полученный на сцене вычислитель в графовом представлении может быть исполнен на строке как мгновенно с выводом результата, так и пошагово с визуализацией исполнения на сцене. Отметим, что алгоритм исполнения вычислителя на строке принадлежит авторству другого участника проекта. Важная деталь — симуляция исполнения недетерминированных автоматов. Если вычислитель находится в нескольких состояниях, то на сцене в один момент времени отображается все множество состояний вычислителя. Помимо этого, реализована блокировка запуска работы симулятора при наличии ошибок в вычислителе. Пример визуализации исполнения НКА на строке можно увидеть на Рис. 2.

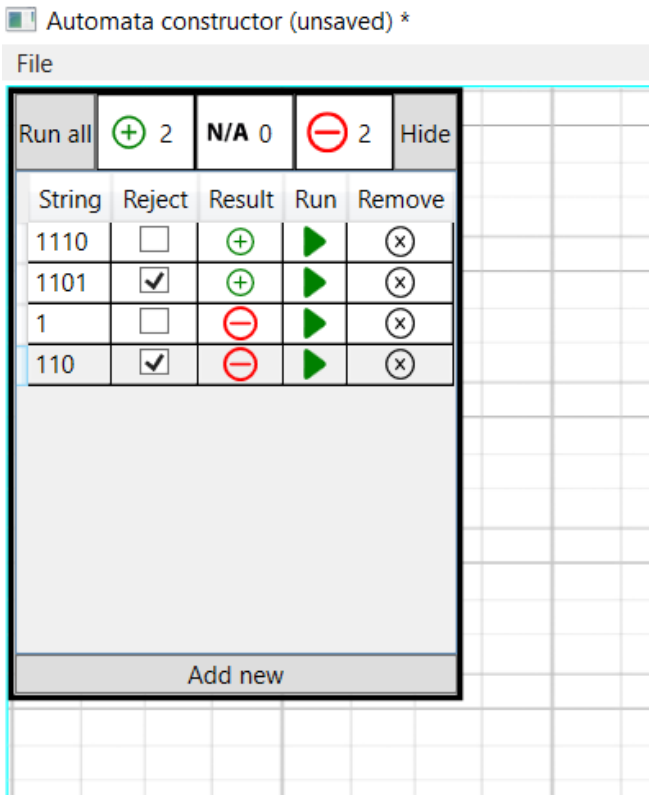

Рис. 3: Тестовая панель

4.3. Тестовая панель

Приложение поддерживает создание автоматических тестов. Пользователь может ввести строку, а так же указать, должна ли строка приниматься вычислителем. Тесты могут быть запущены как в пакетном, так и в одиночном режиме. Пример созданного набора тестов на тестовой панели можно увидеть на Рис. 3.

4.4. Поддержка сохранений

Поддержка сохранений была важной целью разработки на этом этапе. В качестве формата для сохранения как тестов, так и автоматов был выбран XML, так как в будущем планируется поддержка реализованных в формате XML сохранений JFLAP. Сериализация и десериализация реализованы при помощи библиотеки YAXLib [16], которая является традиционной для решения таких задач. На данном этапе развития, на сцене может быть только один автомат, поэтому при загрузке его из

файла сцена предварительно очищается. В дальнейших стадиях разработки, будет реализована возможность загружать автоматы на сцену как «строительные блоки».

4.5. Пользовательский интерфейс

Все разработанные функции приложения были связаны воедино и предоставлены пользователю в графическом интерфейсе, ознакомиться с которым можно на Рис. 4. Через механику ресурсов приложения [15] была реализована поддержка локализации интерфейса на русском и английском языках в зависимости от языка операционной системы, на которой запущена программа. В будущем язык приложения можно будет изменить в настройках. Для того, чтобы новые пользователи могли быстрее начать работу с инструментом после внедрения очередной версии, был написан небольшой обзор функций приложения, ознакомиться с которым можно по ссылке: https://github.com/spbu-se/DesktopAutomataConstructor/wiki/Usage-tutorial.

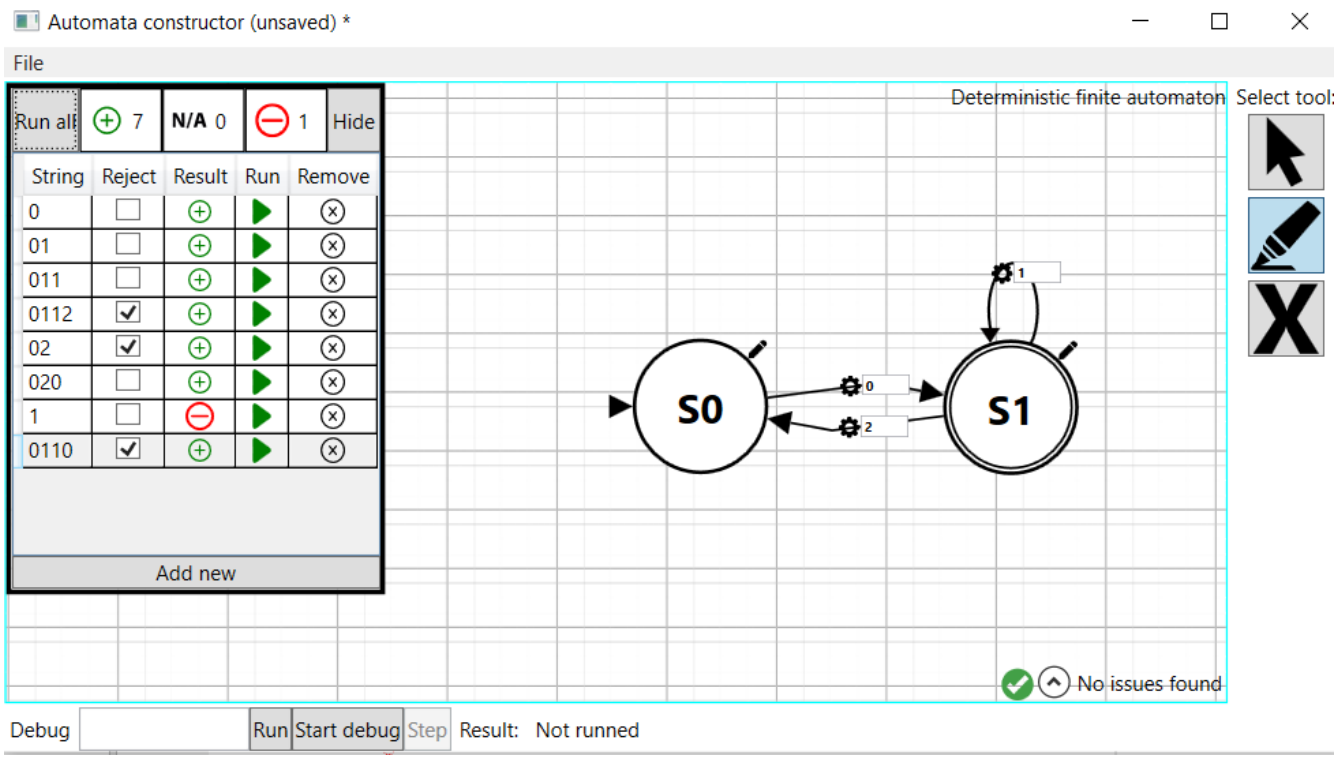

Рис. 4: Пользовательский интерфейс

5. Тестирование и апробация

Разработка проекта, согласно техническому заданию, ведётся инкрементально, каждая фаза должна заканчиваться полным тестированием, пробным внедрением и апробацией на потенциальных пользователях. В связи с этим, тестирование и апробация стали важной частью этой работы.

5.1. Тестирование

Во время разработки все независимые блоки системы, не связанные напрямую с визуальным представлением, были протестированы в рамках модульного тестирования. Помимо этого, были написаны подробные тестовые сценарии, затрагивающие всю функциональность приложения.

5.2. Апробация

После реализации основных функций, приложение было предложено потенциальным пользователям среди студентов математико-механического факультета СПбГУ для апробации по методике System Usability Scale [12]. Помимо этого, пользователи оценили сложность различных действий по предложенным им тестовым сценариям. На момент написания отчёта, число заполненных анкет было равно трём, что недостаточно для того, чтобы делать полноценные выводы, однако на основе имеющихся отзывов System Usability Score равен 78,75, что считается хорошей оценкой.

Заключение

В результате работы были решены следующие поставленные задачи:

- создан визуальный редактор автоматов;
- реализована визуализация исполнения автомата на строке;
- реализована поддержка сохранения/загрузки вычислителей и тестов;
- разработан пользовательский интерфейс, поддерживающий локализацию на русском и английском языках;
- проведено тестирование и апробация приложения;
- написана пользовательская документация.

В дальнейшем существовании проекта большая часть существующего кода может быть переиспользована в рамках плагинной архитектуры для расширения приложения для работы со многими вычислителями. Благодаря написанной документации, новые разработчики смогут легко включиться в проект в рамках летней школы и будущих учебных практик.
Ознакомиться с кодом проекта можно по ссылке: https://github.com/spbu-se/DesktopAutomataConstructor.

Список литературы

[1] AT&T. Graphviz. — 2020. — URL: https://graphviz.org/about/ (дата обращения: 24.05.2021).

[2] Chudá D Rodina D. Automata simulator. — 2010. — URL: https://dl.acm.org/doi/abs/10.1145/1839379.1839449 (дата обращения: 24.05.2021).

[3] Command design pattern. — URL: https://www.dofactory.com/net/command-design-pattern (дата обращения: 24.05.2021).

[4] Data binding overview (WPF .NET). — 2021. — URL: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/data/?view=netdesktop-5.0 (дата обращения: 24.05.2021).

[5] David Schmitt Arthur Zaczek. Graph sharp. — 2012. — URL: https://www.nuget.org/packages/GraphSharp/ (дата обращения: 24.05.2021).

[6] Desktop Guide (WPF .NET). — 2021. — URL: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-5.0 (дата обращения: 24.05.2021).

[7] Dickerson Kyle. Automaton simulator. — 2021. — URL: https://github.com/kdickerson/automatonSimulator (дата обращения: 24.05.2021).

[8] Ivan Zuzak Vedrana Jankovic. FSM Simulator. — 2021. — URL: http://ivanzuzak.info/noam/webapps/fsm_simulator/ (дата обращения: 24.05.2021).

[9] JFLAP. What is JFLAP. — 2005. — URL: http://www.jflap.org/ (дата обращения: 24.05.2021).

[10] LearnLib. Automata lib wiki. — 2018. — URL: https://github.com/LearnLib/automatalib/wiki (дата обращения: 24.05.2021).

[11] Microsoft. MSAGL. — 2021. — URL: https://github.com/microsoft/automatic-graph-layout (дата обращения: 24.05.2021).

[12] Nathan Thomas. How To Use The System Usability Scale (SUS) To Evaluate The Usability Of Your Website. — 2021. — URL: https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usabi (дата обращения: 24.05.2021).

[13] 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 (дата обращения: 24.05.2021).

[14] Panthernet. GraphX wiki. — 2020. — URL: https://github.com/panthernet/GraphX/wiki (дата обращения: 24.05.2021).

[15] Resources in .NET apps. — 2018. — URL: https://docs.microsoft.com/en-us/dotnet/framework/resources/ (дата обращения: 24.05.2021).

[16] Sina Iravanian Julian Verdurmen. YAXLib. — 2021. — URL: https://github.com/YAXLib/YAXLib,urldate=''24.05.2021'', language=''russian''.

[17] UC Davis. Automaton simulator. — 2021. — URL: https://web.cs.ucdavis.edu/~doty/automata/help.html (дата обращения: 24.05.2021).

[18] Wikipedia. Model–view–viewmodel. — 2021. — URL: https://en.wikipedia.org/wiki/Model-view-viewmodel (дата обращения: 24.05.2021).

[19] XAML overview (WPF .NET). — 2021. — URL: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-5.0 (дата обращения: 24.05.2021).

[20] Yurii Litvinov. REAL.NET. — 2019. — URL: https://github.com/yurii-litvinov/REAL.NET (дата обращения: 24.05.2021).

[21] Техническое задание на проект “Конструктор вычислителей”. — URL: https://docs.google.com/document/d/1MG2lUpvM-c7l8idzfboZMIwJK24I4dQdTPD3zrPxhgU (дата обращения: 24.05.2021).