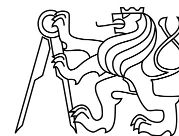
ESKÉ VYSOKÉ U ENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMA NÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁ SKÉ PRÁCE

Název: GUI k automatové knihovn ALIB
Student: Václav Mareš
Vedoucí: Ing. Ond ej Guth, Ph.D.
Studijní program: Informatika
Studijní obor: Softwarové inženýrství
Katedra: Katedra softwarového inženýrství
Platnost zadání: Do konce letního semestru 2017/18

Pokyny pro vypracování

Vytvo te grafické uživatelské rozhraní ke knihovn ALIB. Aplikace bude podporovat podmnožinu sou asné funk nosti knihovny, která se týká operací s kone nými automaty. Prove te rešeršní rozbor existujících aplikací na práci s automaty, analýzu požadavk , návrh a implementaci, která bude ši itelná jako svobodný software (opensource) a realizovaná pomocí Qt. Hotové ešení vhodnými prost edky otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
d kan

V Praze dne 1. února 2017

Poděkování

Děkuji svému vedoucím za cenné připomínky a pomoc při psaní této práce. Děkuji panu Trávníčkovi za pomoc s projektem ALIB. Děkuji své rodině a přátelům za podporu. Další díky také patří testerům: Ondrovi, Frézovi a proděkanovi pro pedagogickou činnost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2017 . . . .

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2017 Václav Mareš. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Mareš, Václav. *GUI k automatové knihovně ALIB*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

// Abstrakt

Abstrakt

Tato práce se zaměřuje na návrh a implementaci grafického uživatelského rozhraní pro automatovou knihovnu ALIB. Text obsahuje popis knihovnou nabízených funkcí a analýzu podmnožiny funkcí, které budou nabízeny grafickým rozhraním. Dále obsahuje rešerši existujících aplikací pracujících s automaty a jejich porovnání, návrh vlastního grafického rozhraní a architektury aplikace. Cílem práce bylo vytvořit pro uživatele zajímavé grafické rozhraní k ovládání automatové knihovny ALIB. Hlavní myšlenkou navrženého rozhraní je poskytnout uživateli stavební bloky, funkce knihovny, a nechat jej poskládat řešení jeho problému. Původně bylo možné využít funkce knihovny pouze z příkazové řádky. Při implementaci byl využit programovací jazyk C++ a framework Qt. Výstupem práce je vlastní program s grafickým rozhraním zpřístupňující funkce knihovny ALIB. Vzniklý kód je pod opensource licencí.

Klíčová slova ALIB, grafické uživatelské rozhraní, konečné automaty, Qt, C++

Abstract

This bachelor's thesis describes the design and implementation of application providing grafical user interface for automata library ALIB. The text contains overview of ALIB functions and analysis of functions subset working with finite automata. Research on existing applications focusing on finite automata, design of GUI and application's architecture. Application's goal is to provide intuitive grafical interface for functions from automata library. Main idea behind the GUI is provide algorithms as pieces and let the users build their solution. Originally interface of project ALIB was on the command line. The implementation uses C++ and framework Qt. The product of this thesis is an application providing GUI to subset of ALIB functions. Created code is licenced as opensource.

Keywords ALIB, graphical user interface, finite automata, Qt, C++

Obsah

Úvod **1**
Cíl práce 1
Struktura práce 2

1 Analýza **3**
1.1 Současný stav řešení problematiky 3
1.2 Řešení 4
1.3 Analýza knihovny ALIB 6
1.4 Požadavky 9

2 Návrh **13**
2.1 Návrh uživatelského rozhraní 13
2.2 Návrh aplikace 19

3 Realizace **25**
3.1 Framework Qt 25
3.2 Propojení s knihovnou ALIB 26

4 Testování **29**
4.1 Metodika 29
4.2 Průběh 29
4.3 Výsledky 31
4.4 Výhled do budoucna 33

Závěr **35**

Literatura **37**

A Seznam použitých zkratek **39**

B Obsah přiložené SD karty **41**

Seznam obrázků

2.1 Návrh hlavního okna 14
2.2 Příklad uživatelem vytvořeného diagramu algoritmů 15
2.3 Návrh dialogu zadání vstupu 17
2.4 Návrh dialogu nastavení výstupu 18
2.5 Diagram třídy `WrapperBox` 20
2.6 Diagram použití vzoru *facory method* 21
2.7 Diagram hierarchie podtříd třídy `GraphicsBox` 22
2.8 Diagram hierarchie podtříd třídy `ModelBox` 23

Seznam tabulek

1.1 Tabulka vstupů a výstupů algoritmů 10

4.1 Výsledky kvantitativních otázek 32

Úvod

Na katedře teoretické informatiky Fakulty informačních technologií je pod vedením Ing. Jana Trávníčka vyvíjen projekt ALIB (Automata library) [1]. Projekt je určen k práci s automaty, gramatikami a dalšími strukturami spojenými s matematickou implementací algoritmů. Sloužit má jak studentům, tak vyučujícím, avšak jeho obsluha je v současnosti obtížná. Jediné rozhraní, které je knihovnou ALIB poskytováno, je v příkazové řádce. Příklad použití ALIBu najdeme v sekci 1.3.3. Ovládání z příkazové řádky klade na uživatele značné nároky, a proto vznikl požadavek vytvoření grafického uživatelského rozhraní. Toto rozhraní má poskytovat vybranou podmnožinu funkcí, které nabízí knihovna ALIB, zaměřenou na práci s konečnými automaty. Rozhraní značně usnadní použití projektu.

Cíl práce

Hlavním cílem práce je navrhnout a implementovat grafické uživatelské rozhraní k automatové knihovně ALIB. Přínosem tohoto rozhraní je usnadnit využití funkcí knihovny ALIB. K implementaci aplikace má být využit jazyk C++ a framework Qt. S ohledem na hlavní cíl práce jsou dílčí cíle stanoveny takto:

- Seznámit se s existujícími aplikacemi, které pracují s konečnými automaty.
- Zorientovat se ve zdrojovém kódu projektu ALIB.
- Analyzovat podmnožinu funkcí, které bude aplikace podporovat.
- Stanovit požadavky na aplikaci.
- Seznámit se s frameworkem Qt a navrhnout jeho využití.

- Navrhnout architekturu aplikace.
- Implementovat navrženou aplikaci za pomoci C++ a Qt.
- Vhodně otestovat výsledné řešení.

Struktura práce

Tato práce je rozdělena do čtyř kapitol. V první kapitole se věnuji analýze existujících aplikací pro práci s konečnými automaty, které poskytují grafické rozhraní. Dále představuji automaty a algoritmy z knihovny ALIB, jejich vztahy, a ovládání z příkazové řádky. Nakonec této kapitoly stanovuji požadavky pro naši aplikaci.

Druhá kapitola obsahuje návrh aplikace. Návrh je rozdělen na část grafického rozhraní a architekturu aplikace. Představuji zde podobu jednotlivých dialogů a způsob, kterým bude uživatel pracovat s algoritmy knihovny ALIB.

Třetí kapitola popisuje realizaci. Najdeme zde rozhodnutí o způsobu využití frameworku Qt. Dále kapitola obsahuje výpisy kódu implementující zajímavé části návrhu aplikace.

Poslední kapitolou je testování aplikace. Kapitola obsahuje popis metodiky, průběh testování a jeho vyhodnocení.

Kapitola 1

Analýza

1.1 Současný stav řešení problematiky

V současnosti neexistuje grafické rozhraní ke knihovně ALIB. Můžeme se však podívat na jiné aplikace pracující s automaty, několik si jich představit a rozdělit je do skupin podle jejich funkcí.

První skupinu tvoří aplikace, které slouží pouze k navrhnutí grafů automatů a nijak nepracují s jejich logikou. Představíme si pouze jediného zástupce *Finite State Machine Designer* [2]. Tento nástroj je velmi snadno dostupný a použitelný. Výsledný graf automatu je pak možné exportovat ve formátech PNG (Portable Network Graphics), SVG (Scalable Vector Graphics) a LaTeX.

Druhou skupinu tvoří aplikace, které jsou schopny provést jednoduchou simulaci nad automaty. Tedy vyhodnotit, zda je daný vstup automatem přijat, nebo odmítnut. Do této skupiny zařadím online nástroj *FSM Simulator* [3], kde FSM znamená finite state machine, a desktopovou aplikaci *Automaton Simulator* [4]. Prve zmíněný *FSM Simulator* simuluje krok po kroku funkci DFA (deterministic finite automaton), NFA (nondeterministic finite automaton) a NFA-ε (nondeterministic finite automaton with epsilon transitions). Je aktivně vyvíjen a na svých stránkách odkazuje na další nástroje pracující s automaty, gramatikami a regulárními výrazy. Nevýhodou je, že výsledky není možné nijak exportovat. Aplikace *Automaton Simulator* dokáže kromě simulace DFA a NFA navíc i simulaci deterministického zásobníkového automatu a Turingova stroje. Dokáže také exportovat navržené automaty ve formátu JSON (JavaScript Object Notation). Bohužel aplikace již není aktivně vyvíjena a poslední verze vznikla v roce 2008.

Třetí skupinu tvoří aplikace schopné složitějších operací nad automaty. Například odstranění epsilon přechodů či minimalizaci DFA. Do této skupiny zahrnuji *jFAST – the Finite Automata Simulator* [5], *Automata editor* [6] a *Finite-*

Automata[7]. Všechny tři tyto nástroje jsou desktopové aplikace. Ovládání je jednoduché – ve všech třech případech spočívá v umisťování prvků na plochu. Tímto způsobem je sestaven automat, který lze poté upravit například minimalizací nebo jinou nabízenou funkcí. *jFAST* a *Automata editor* dokáží také automaty exportovat, a to jak v podobě obrázků, tak také textově.

Všechny zde představené aplikace a nástroje působí prostě a jednoduše, některé až stroze. U některých je to dáno jejich stádiem vývoje, *jFAST* je stále v beta verzi. U jiných je to ukončením vývoje, například *Automaton Simulator*. Nejvíce zaujal *Automata editor* a to nejen mě, ale i uživatele, alespoň podle hodnocení a počtu stažení aplikace na stránkách projektu.

Všechny aplikace se tedy ve větší míře věnují možnostem tvorby grafu automatu. Nabízené algoritmy pro transformace konečných automatů, pokud vůbec jsou nabízeny, jsou jen základní a zanechávají pocit že jsou vedlejší funkcí aplikace. Naše aplikace se zabývá hlavně algoritmy nad konečnými automaty, které knihovna ALIB nabízí. Uživatel nemá možnost si automat nakreslit, může ho zadat pouze textově. Inspirací z uživatelských rozhraní představených aplikací je jednoduchost ovládání. Ideálně všechny možnosti na jednom hlavním okně. Všechny aplikace také nabízí plochu, nebo plátno pro vytvoření automatu. Tento prvek pro sestavení celku z částí na grafické ploše byl přenesen i do naší aplikace.

1.2 Řešení

Zadání bakalářské práce specifikuje technologie C++ a framework Qt. Technologie jsou tedy dané. Potřebuji ještě určit podmnožinu funkcí nabízených knihovnou ALIB a dále jakým způsobem bude vrstva uživatelského rozhraní komunikovat s knihovnou. Knihovna ALIB se skládá z mnoha jednotlivých menších programů, které se ovládají přes příkazovou řádku. Výstup jedno programu se předává dalšímu na vstupu, a tak je poskládán požadovaný celek.

Společnou částí pro všechny možnosti řešení je zvolení vhodné podmnožiny funkcí s automaty, které bude výsledné GUI nabízet. Této části se věnuji v sekci 1.3. Z pohledu na již existující programy vidíme, že je vhodné umět přijímat více formátů vstupu. Například textový zápis automatu, regulární výraz a zápis gramatikou. Dále nabídnout operace jako odstranění epsilon přechodů NFA a minimalizaci DFA. V neposlední řadě exportovat výstupy, a to jak v textové podobě (ALIB nabízí XML (Extensible markup language)), tak v podobě obrázku, nejlépe formátu SVG. ALIB umí poskytnou výstup v DOT formátu (textový popis grafu) pro aplikaci *Graphviz*[8].

1.2.1 Možnosti řešení

Propojení GUI a knihovny lze udělat třemi způsoby.

1.2.1.1 Oddělení fungování GUI od fungování ALIBu

V tomto případě by GUI zastoupilo příkazovou řádku a volalo jednotlivé programy knihovny. Toto řešení nabízí izolaci od knihovny, která se stále živě vyvíjí. Dobře by tedy odolalo změnám uvnitř knihovny. Hlavní nevýhodou bude rychlost běhu tohoto řešení. Volání jednotlivých programů a předávání výsledků znamená opětovanou inicializaci objektů uvnitř knihovny.

1.2.1.2 Zakomponovat knihovnu a GUI do jednoho programu

Vzhledem ke sdílenému programovacímu jazyku C++ je možné přímo vytvářet objekty z knihovny a vyhnout se opakování některých kroků. Tento přístup je zajisté rychlejší než předchozí možnost řešení. Nevýhodou tohoto přístupu je nerobustnost GUI ve vztahu ke knihovně, kterou s sebou nese přímá závislost na jejím kódu, například volání metod jejích tříd.

1.2.1.3 Spolupráce knihovny ALIB a GUI

V tomto případě by knihovna byla schopna nabídnout přehled všech funkcí, které obsahuje. Uživatelské rozhraní by bylo schopné reagovat na tento přehled a nabízet dostupné funkce uživateli. Tato varianta byla zamítnuta z důvodu nepřipravenosti ALIBu. Knihovna se stále vyvíjí a v tuto chvíli nedokáže reportovat nabízené funkce. Implementování této funkčnosti by vydalo na samostatnou bakalářskou práci.

1.2.2 Zvolené řešení

Zvoleným řešením je druhá varianta, tedy začlenit knihovnu ALIB jako část aplikace. Toto řešení je vzhledem k současným možnostem preferované i tvůrci projektu ALIB. Aplikace může vytvářet přímo instance tříd z knihovny a volat funkce ze sdílených objektů ALIBu. Výsledná aplikace díky tomuto přístupu bude dostatečně výkonná. Musíme si dát pozor na změny v rozhraních, které budou využívány, aby nedošlo k problémům při kompilaci. Jak již bylo řečeno, projekt ALIB je stále živě vyvíjen a tato situace může nastat. Toto řešení má také výhodou pro svou rozšiřitelnost, teoreticky je možné řešení upravit na třetí variantu, v případě, že by knihovna v budoucnu poskytovala přehled funkcí.

1.3 Analýza knihovny ALIB

Knihovna ALIB je aktivní projekt na katedře teoretické informatiky pod vedením Ing. Jana Trávníčka. Knihovna nabízí algoritmy pracující se strukturami, které existují nad regulárními, bezkontextovými a kontextovými jazyky. Tyto struktury jsou implementovány soubory ve složce alib2data a algoritmy pracující s těmito strukturami jsou ve složce alib2algo. Tato práce se však věnuje pouze oblasti konečných automatů. V následující části práce si rozebereme, jaké typy automatů v ALIBu existují a jaké operace s nimi můžeme provádět.

1.3.1 Třídy

Konečné automaty jsou v knihovně zastoupeny třídami `DFA`, `NFA`, `MultiInitialStateNFA`, `EpsilonNFA`, `CompactNFA` a `ExtendedNFA`. Automaty tvoří v tomto pořadí řadu, kde dříve zmíněný jednodušší typ může být reprezentován později zmíněným složitějším typem. ALIB nám dokonce přímo nabízí k tomu určené konstruktory. Třída `NFA` tedy například nabízí konstruktor přijímající `DFA`, `MultiInitialStateNFA` lze zkonstruovat z `DFA`, nebo `NFA` a podobně. Třídy však nejsou vždy zcela přesnou reprezentací jednoduššího typu. Jednoduchost typu automatu pro potřeby této práce určíme jako pořadí ve zmíněné řadě. Nejjednodušší typ je tedy `DFA`, nejsložitější je `ExtendedNFA`. Speciálním případem je `MultiInitialStateNFA`, který jako jediný typ dokáže reprezentovat více počátečních stavů. Složitější typy však poskytují konstruktor akceptující `MultiInitialStateNFA`. Dojde ke konverzi původních počátečních stavů a vznikne nový počáteční stav s epsilon přechody na původní počáteční stavy. `CompactNFA` umožňuje mít na přechodech řetězce a `ExtendedNFA` dovoluje na přechodech dokonce regulární výrazy.

`DFA` – Deterministický konečný automat, implementován dle definice

`NFA` – Nedeterministický konečný automat, implementován dle definice

`MultiInitialStateNFA` – Nedeterministická konečný automat s více počátečními stavy

`EpsilonNFA` – Nedeterministický konečný automat s epsilon přechody

`CompactNFA` – Nedeterministický konečný automat, který umožňuje na přechodech řetězce včetně prázdného

`ExtendedNFA` – Nedeterministický konečný automat, který umožňuje na přechodech regulární výrazy

1.3.2 Algoritmy

ALIB rozděluje algoritmy nad automaty do tří skupin: *transform*, *simplify* a samostatný algoritmus *determinize*. Algoritmy ze skupiny *transform* mohou měnit přijímaný jazyk, nebo reprezentovat automat složitějším typem. Můžeme je dále rozdělit na algoritmy přijímající pouze jeden automat a algoritmy přijímající dva automaty. Algoritmy, které přijímají jeden automat, jsou `Reverse`, `Compaction` a `Iteration`. Algoritmy přijímající dva automaty jsou `Union`, `Concatenation` a `Intersection`.

`Reverse` je algoritmus, který je implementovaný pro objekty `DFA`, `NFA` a `MultiInitialStateNFA`. Výstupem je `NFA`, který přijímá doplněk jazyka přijímaného automatem na vstupu.

`Compaction` je algoritmus, který přijímá objekty `DFA`, `NFA` a `CompactNFA`. Výstupem je `CompactNFA`, který přijímá stejný jazyk jako původní automat.

Algoritmus `Iteration` nabízí dvě varianty `Iteration` a `IterationEpsilonTransition`. Vstupem pro `Iteration` je jeden `DFA` nebo `NFA` a výstupem je `NFA`, který přijímá iteraci jazyka přijímaného původním automatem. Varianta `IterationEpsilonTransition` akceptuje kterýkoliv objekt automatu a vrací `EpsilonNFA`, tedy automat s novým počátečním stavem a epsilon přechodem do původního počátečního stavu a epsilon přechody z konečných stavů do původního počátečního.

`Union` také nabízí dvě varianty. Jednou je `UnionCartesianProduct` a druhou `UnionEpsilonTransition`. Obě varianty přijímají dva automaty a vrací jeden. První algoritmus přijímá dva `DFA` a vrací jeden `DFA`, nebo přijímá dva `NFA` a vrací `NFA`. Výsledný automat přijímá sjednocení původních jazyků. Tento algoritmus se také označuje jako *sjednocení paralelní činností*. Druhý algoritmus přijme na vstupu dva `DFA`, `NFA`, nebo `EpsilonNFA` výstupem je vždy `EpsilonNFA` s novým počátečním stavem a epsilon přechody z tohoto nového stavu do původních počátečních stavů automatů na vstupu.

Algoritmus `Concatenation` má také dvě provedení: `Concatenation` a `ConcatenationEpsilonTransition`. Algoritmy přijímají na vstupu dva automaty stejného typu a vrací jeden automat. Jedná se o algoritmy součinu dvou jazyků. První akceptuje objekty `DFA` a `NFA`. Jeho výstupem je `NFA`. Druhé provedení navíc akceptuje `EpsilonNFA` a návratovým objektem je vždy `EpsilonNFA`.

Algoritmus `Intersection` také označován za *cartesian product* nebo *průnik paralelní činností*. Přijímá dva `DFA` a vrací jeden `DFA`, nebo dva `NFA` a vrací jeden `NFA`. Výsledný automat přijímá průnik jazyků, které jsou přijímány automaty na vstupu.

Druhou skupinu *simplify* tvoří algoritmy, které zjednodušují automat avšak nemění přijímaný jazyk. Tyto algoritmy jsou `EpsilonRemover`, `Minimize`, `Normalize`, `Rename`, `SingleInitialState`, `Total`, `UnreachableStateRemover`, `UselessStateRemover` a `Trim`.

Algoritmus `EpsilonRemover` nabízí dvě verze `EpsilonRemoverIncoming` a `EpsilonRemoverOutgoing`. Jak je z názvu patrné, algoritmy odstraňují epsilon přechody. Oba tedy umí přijmout `EpsilonNFA` a vrátit odpovídající `NFA`. Obě verze také umí přijmout objekty jednodušší než `EpsilonNFA` ty vrací nezměněné, protože již epsilon přechody neobsahují.

Algoritmus `Minimize`, jak název napovídá, provádí minimalizaci. Přijímá tedy `DFA` a vrací minimální `DFA`.

Algoritmus `Normalize` přijímá `DFA` a vrací normalizovaný `DFA`. Normalizací dojde k deterministickému přejmenování stavů. Nové stavy jsou přirozená čísla od nuly. Algoritmus najde uplatnění například po determinizací, kdy jsou nové stavy ALIBem označeny množinou původních stavů.

`Rename` je algoritmus velmi podobný `Normalize`. `Rename` přejmenuje stavy na přirozená čísla od nuly, ale na rozdíl od `Normalize` nedeterministicky. Vstupem je opět `DFA` a návratový objekt je také `DFA`.

`SingleInitialState` slouží k převedení automatu s více počátečními stavy na automat s jedním počátečním stavem. Užitečné uplatnění je tedy předložit na vstupu `MultiInitialStateNFA`, výstupem pak je `NFA`. Algoritmus také umí přijmout všechny ostatní typy, které pouze vrátí, protože již splňují podmínku jednoho počátečního stavu.

Algoritmus `Total` slouží k doplnění chybového stavu. `Total` přijme objekt typu `DFA` nebo `NFA`, avšak objektem popsaný automat musí být deterministický. Návratový objekt je stejného typu jako ten na vstupu. Výsledný automat je úplně určený.

Algoritmy `UselessStateRemover` a `UnreachableStateRemover` odstraňují příslušně zbytečné a nedosažitelné stavy. Algoritmus `Trim` provádí oba tyto algoritmy. Všechny tři algoritmy přijímají na vstupu kterýkoliv objekt a vrací příslušný objekt stejného typu.

Zvlášť v rozdělení ALIBu ještě stojí algoritmus `Determinize`. Přijímá objekt typu `DFA` nebo `NFA` a vrací příslušný `DFA`. Jedná se o algoritmus determinizace, tak jak je popsán například v předmětu AAG.

Vstupy a výstupy pro všechny algoritmy jsou uspořádány do tabulky 1.1. Na řádcích najdeme jednotlivé algoritmy a v příslušných sloupcích počet auto-

matů daného typu, které algoritmus přijímá. Většina algoritmů vrací výsledek pouze jednoho typu. Má tedy jedinou jedničku ve sloupci `out` pod daným typem. Pokud je na řádku více jedniček ve sloupcích `out`, zachovává se typ automatu ze vstupu. Speciální případ pak tvoří algoritmy, které mají ve sloupci `in` hvězdičku. Najdeme sloupec `out` s hvězdičkou a tento typ bude mít výsledný automat.

1.3.3 Ovládání

Jediné uživatelské rozhraní, které projekt ALIB nabízí, je na příkazové řádce. Funkce nabízené knihovnou jsou dostupné v podobě jednotlivých programů. Pokud tedy uživatel chce složit několik funkcí a dosáhnout řešení složitějšího problému, nezbývá mu než využít systému *pipes and filters* – předávat výstup předchozího programu na vstup dalšího. Zřejmě proto, že je projekt stále ve vývoji, neexistuje žádná textová dokumentace a jednotlivé programy toho po předání přepínače *help* také mnoho neprozradí. Tento stav platí v době psaní této práce. Uživatel je tedy postaven do situace zkoušet a nevyhnutelně se mýlit, dokud se mu nepodaří sestavit takový příkaz, který splní jeho požadavek.

Pojďme si ukázat konkrétní použití knihovny. Uživatel má dva `NFA` ve formátu XML v souborech a.xml a b.xml. Přeje si graf deterministického automatu ve formátu SVG do souboru c.svg. Výsledný automat přijímá součin jazyků jeho automatů.

Příkaz pro tuto operaci, za předpokladu, že vše je ve složce, kde se právě nachází, vypadá takto.

Výpis kódu 1.1: Příklad použití ALIBu z příkazové řádky

```
./alangop2 -i a.xml -j b.xml -a concatenation \
| ./adeterminize2 | ./aminimize2 \
| ./anormalize2 -l automaton \
| ./aconvert2 --automaton_to_dot | dot -Tsvg -oc.svg
```

Na první pohled je zřejmé, že je od uživatele vyžadováno jisté úsilí a znalost knihovny, aby byl schopen svůj požadavek uspokojit. Zjednodušit a zpříjemnit plnění takových požadavků je hlavním cílem uživatelského rozhraní.

1.4 Požadavky

Požadavky na aplikaci vyplývají z analýzy knihovny ALIB a ze zadání práce, byly konzultovány s vedoucím práce a stejně tak i s vedoucím projektu ALIB. Všechny nefunkční požadavky vyplývají ze zadání práce.

	DFA		NFA		MultiInitialStateNFA		EpsilonNFA		CompactNFA		ExtendedNFA	
	in	out	in	out	in	out	in	out	in	out	in	out
Reverse	1	0	1	1	1	0	0	0	0	0	0	0
Compaction	1	0	1	0	0	0	0	0	1	1	0	0
Iteration	1	0	1	1	0	0	0	0	0	0	0	0
IterationEpsilon.	1	0	1	0	1	1	1	0	1	0	1	0
UnionCartesian.	2	1	2	1	0	0	0	0	0	0	0	0
UnionEpsilon.	2	0	2	0	0	0	2	1	0	0	0	0
Concatenation	2	0	2	1	0	0	0	0	0	0	0	0
ConcatenationEpsilon.	2	0	2	0	0	0	2	1	0	0	0	0
Intersection	2	1	2	1	0	0	0	0	0	0	0	0
EpsilonRemover	1	1	1	1*	1	1	1*	0	0	0	0	0
Minimize	1	1	0	0	0	0	0	0	0	0	0	0
Rename	1	1	0	0	0	0	0	0	0	0	0	0
SingleInitialState	1	1	1	1*	1*	0	1	1	1	1	1	1
Total	1	1	1	1	0	0	0	0	0	0	0	0
Trim	1	1	1	1	1	1	1	1	1	1	1	1
Unreachable	1	1	1	1	1	1	1	1	1	1	1	1
Useless	1	1	1	1	1	1	1	1	1	1	1	1
Determinize	1	1	1	0	0	0	0	0	0	0	0	0

Tabulka 1.1: Tabulka vstupů a výstupů algoritmů. Pro každý algoritmus je zaznamenáno kolik objektů daného typu automatu přijímá a kolik jich vrací. Pokud je možností více, zachovává se typ automatu. Hvězdičkou jsou označeny zvláštní případy, kde označený vstup má označený výstup.

- Aplikace nabídne uživatelské rozhraní, přes které bude moci uživatel ovládat podmnožinu funkcí knihovny ALIB.
- Aplikace bude napsána v jazyce C++ a využije framework Qt.
- Aplikace bude spolupracovat s projektem knihovny ALIB, bude vytvářet instance a volat statické metody tříd knihovny.

Všechny funkční požadavky vyplývají z hlavní myšlenky této práce. Nabídnout uživateli snadno ovladatelné grafické rozhraní, které zpřístupní algoritmy nad konečnými automaty. Funkčními požadavky jsou tedy tyto algoritmy:

- Determinizace konečného automatu
- Odstranění zbytečných stavů automatu
- Odstranění nedosažitelných stavů automatu
- Minimalizace konečného automatu
- Odstranění epsilon přechodů z automatu
- Převedení na automat s jedním počátečním vstupem
- Doplnění na úplný automat
- Součin dvou automatů
- Sjednocení dvou automatů
- Průnik dvou automatů
- Iterace automatu

Více jsem se o implementaci těchto algoritmů knihovnou ALIB zmínil v sekci 1.3.2. Z těchto požadavků také přímo vyplývají případy užití. V této práci tedy nenajdeme diagramy případů užití. Za dostatečný popis užití považuji analýzu algoritmů a jejich kombinace.

Kapitola 2

Návrh

Návrh a implementace aplikace byly provedeny iterační metodou. V každé iteraci jsem rozšiřoval existující aplikaci a snažil se vyhovět vybraným požadavkům z analýzy. Níže popsaný návrh je výsledkem tohoto procesu. Z tohoto důvodu je v něm také vidět konkrétní využití frameworku Qt. Tomu se blíže věnuji v sekci 3.1. Tato kapitola je rozdělena na část popisující návrh uživatelského rozhraní a na část popisující návrh aplikace.

2.1 Návrh uživatelského rozhraní

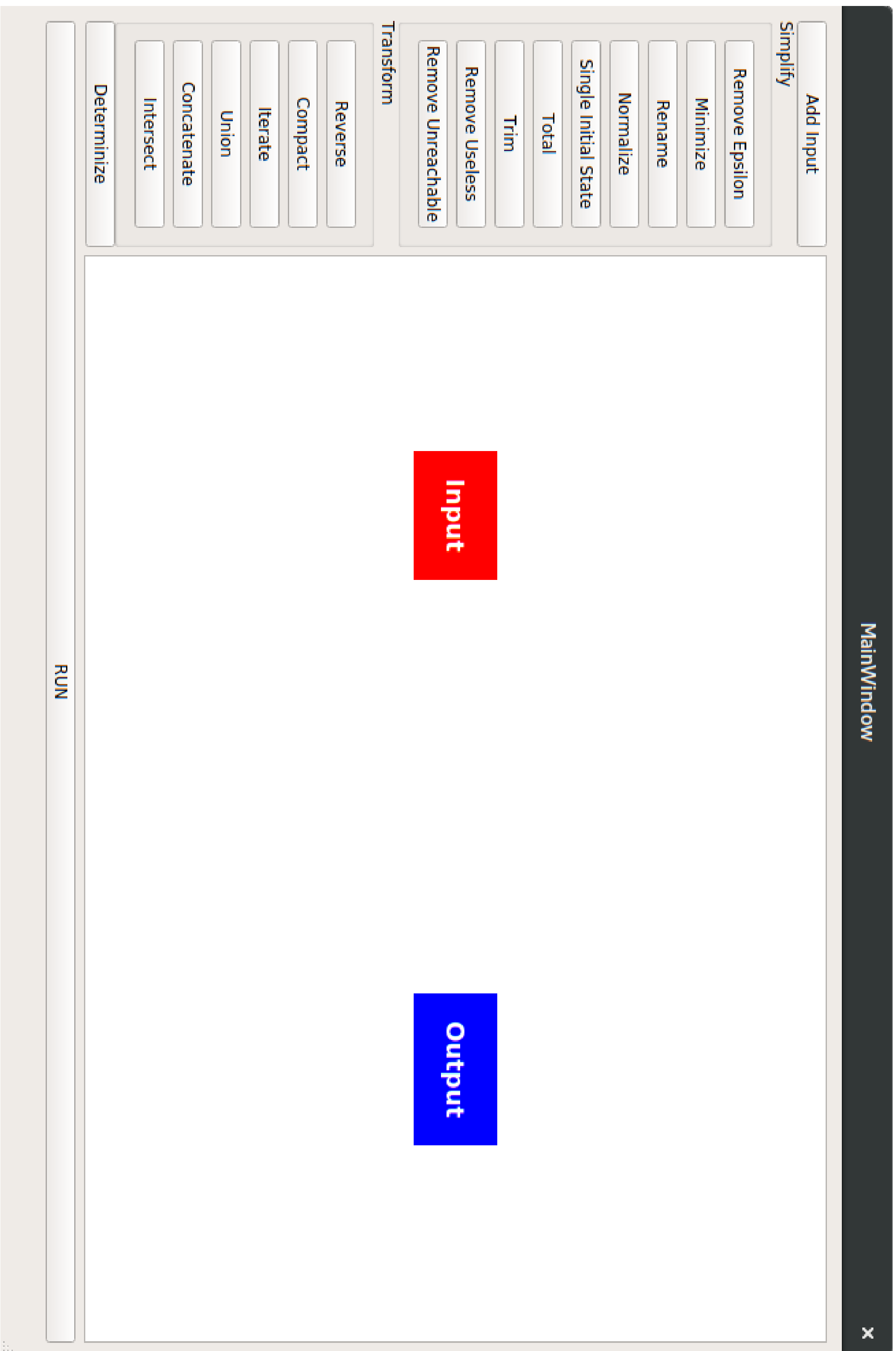
Návrh rozhraní vychází z analýzy a struktury ALIBu. Cílem je nabídnout uživateli snadné a intuitivní ovládání, které umožní poskládat jednotlivé algoritmy do větších celků. Dále byl kladen důraz na jednoduchost vstupu a výstupu. Uživatel má několik možností při zadávání automatu. Stejně tak má možnost uložit výsledek v textové či v grafické podobě.

Zvolený způsob manipulace s algoritmy se podobá UML diagramu aktivit. Jednotlivé uzly jsou algoritmy ALIBu, hrany znázorňují předání výstupu na vstup. Pomyslný token či tokeny, které se grafem pohybují, jsou objekty konečných automatů. Tento koncept je dále označován jako diagram algoritmů, viz sekce 2.1.2.

Nyní si představíme jednotlivé části uživatelského rozhraní. Zdůvodním některá rozhodnutí, která vedla k výsledné podobě, a nakonec si projdeme obecné použití aplikace, jako jsou společné kroky, které uživatel provede, nehledě na konkrétní algoritmus nebo algoritmy, které chce použít.

2.1.1 Hlavní okno

Hlavní okno aplikace neboli mainwindow (obrázek 2.1) je prvním prvkem grafického rozhraní, který uživatel uvidí. Hlavní okno můžeme rozdělit na několik



Obrázek 2.1: Návrh hlavního okna.

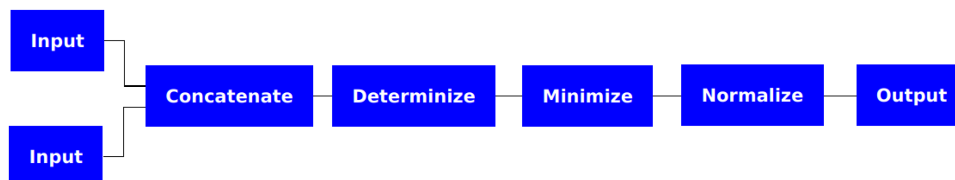
částí. V levé části najdeme dostupné algoritmy, rozdělené do skupin tak, jak je tomu uvnitř knihovny ALIB. Dále je zde plátno pro práci s diagramem algoritmů. Kliknutím na tlačítko algoritmu v levé části dojde k vytvoření uzlu diagramu na plátně. Po kliknutí na některá tlačítka algoritmů se nám otevře kontextové menu. Zde můžeme zvolit variantu daného algoritmu, kterou chceme použít.

Jednotlivé uzly se na plochu plátna umisťují v podobě obdélníků znázorňujících jednotlivé algoritmy. Automaticky jsou na něm umístěny povinné uzly *Input* a *Output*. Přes kontextové menu je možné s uzly pracovat – propojovat je hranami nebo nastavovat jejich chování. V neposlední řadě je na hlavním okně tlačítko *RUN*, kterým se spustí průchod připraveným diagramem algoritmů.

Toto uspořádání bylo zvoleno na základě struktury ALIBu. Levá část přímo kopíruje rozdělení uvnitř knihovny. Myšlenka interakce uživatele s plátnem a grafické reprezentování algoritmů a jejich propojení je odvozena z existujících aplikací, zmíněných v sekci 1.1. Samotný diagram algoritmů je přímo inspirován nabízeným ovládáním ALIBu, výpis kódu 1.1, přestože naše aplikace komunikuje přímo s funkcemi knihovny.

2.1.2 Diagram algoritmů

Hlavní částí uživatelského rozhraní je komponenta plátna. Zde uživatel manipuluje s diagramem algoritmů, obrázek 2.2. Sestavením tohoto diagramu uživatel nastavuje, které algoritmy ALIBu a v jakém pořadí se vykonají. Diagram se podobá jednoduššímu UML diagramu aktivit. Obsahuje povinně jeden počáteční a koncový uzel. Případně další vnitřní a počáteční uzly. Existují dva typy vnitřních uzlů podle typů algoritmů. První dovoluje pouze jednu vstupní hranu, druhý typ dovoluje vstupní hrany dvě. Všechny vnitřní uzly mají právě jednu výstupní hranu. Token, případně tokeny, toku v diagramu jsou objekty konečných automatů.



Obrázek 2.2: Příklad uživatelem vytvořeného diagramu algoritmů

2.1.3 Input

Input, nebo také vstup, je povinným počátečním uzlem diagramu algoritmů. Jedná se o počáteční uzel. Na plátně je reprezentován obdélníkem s popisem Input. Kliknutím pravým tlačítkem myši se lze otevřít kontextové menu.

Zde najdeme možnost *Connect* pro připojení výstupní hrany k dalšímu uzlu. Dále možnost *Set Input*, kterou otevřeme dialog *Input settings*, obrázek 2.3. V tomto okně můžeme nastavit vstupní automat. Můžeme nahrát automat ze souboru, a to ve formátu XML nebo čistě textovém, dle specifikace ALIBu. Pokud je zadaný vstup validním automatem, nabídne se nám jeho grafické zobrazení. Pokud vstup validní není, dialog na to uživatele upozorní. Špatné nebo chybějící nastavení uzlu *Input* je také reprezentováno červenou barvou obdélníku v diagramu algoritmů.

Vstupních uzlů může být na plátně více než jen jeden. Další vstupní uzel přidáme tak, že klikneme na tlačítko *Add Input* v levé části hlavního okna. Tímto způsobem uspokojíme požadavek algoritmů se dvěma vstupy. Zároveň má uživatel možnost mít více nastavených vstupů a jednoduchým přepojením jedné hrany změnit automat, který projde diagramem algoritmů. Správa vstupů je tedy plně v rukou uživatele.

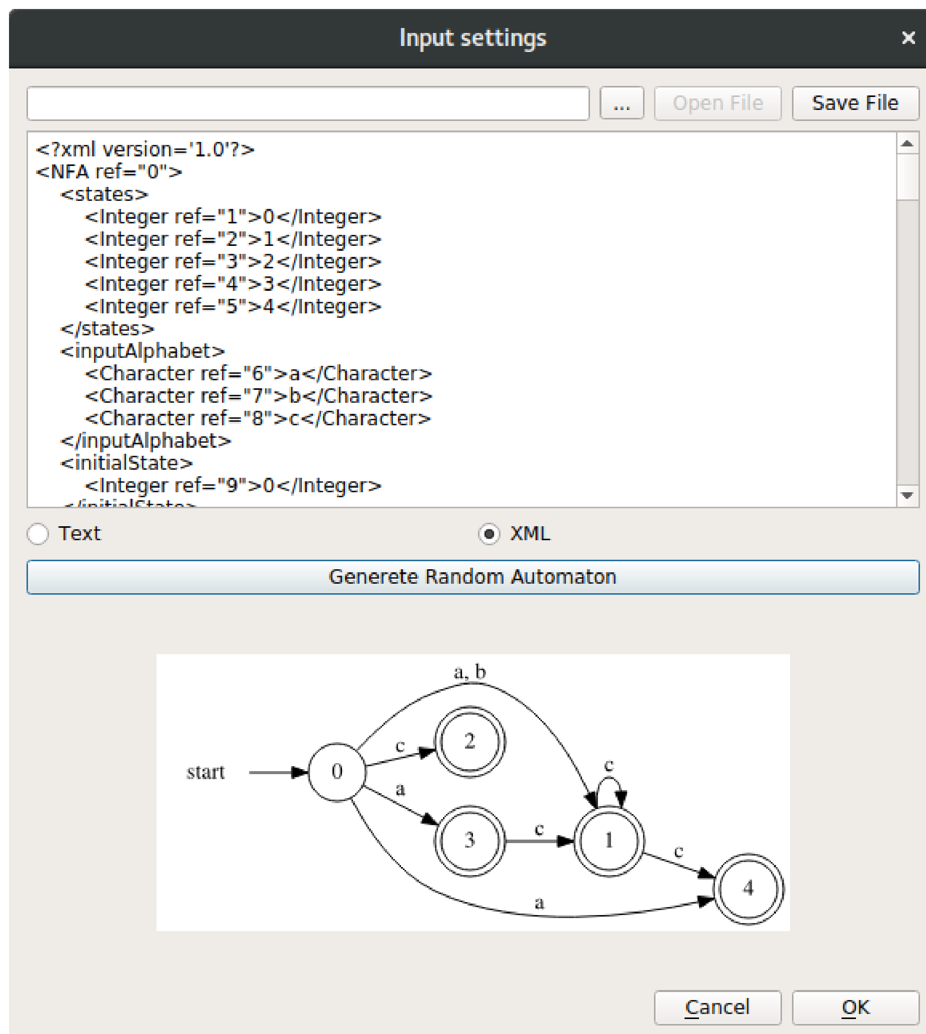
Poslední možností v kontextovém menu je *Delete*. Uživatel tedy může snadno odstranit vstupní uzly, které už nepotřebuje. Toto rozhodnutí, svěřit přidávání a odebírání vstupních uzlů uživateli, nebyl původní návrh uživatelského rozhraní. Původní myšlenka byla přidávat nové *Input* uzly spolu s algoritmy, které je požadují. Tím bychom zaručili, že máme vždy dostatek vstupních uzlů pro sestavení platného diagramu algoritmů. Během návrhu jsem narazil na problém – vnitřní uzly můžeme z plátna mazat. V nejhorším případě je uzel propojen se dvěma nastavenými vstupy, možná i přes další algoritmy. V této situaci nejsem schopen rozhodnout, který *Input* odstranit spolu s mazaným algoritmem. Výsledný návrh tedy je *Input* nepřidávat ani neodstraňovat a nechat rozhodnutí na uživateli. Z tohoto důvodu nabízí počáteční uzel možnost *Delete* ve svém kontextovém menu a v levé části hlavního okna je tlačítko *Add Input*.

2.1.4 Output

Output nebo také výstup je druhým povinným uzlem diagramu algoritmů. Jedná se o uzel koncový. Na plátně je reprezentován obdélníkem s popisem Output. Otevřením kontextového menu se nám nabídne jediná možnost a to *Set Output*. Touto možností otevřeme dialog *Output settings* obrázek 2.4. V tomto dialogu můžeme vybrat formát výstupu XML, PNG, SVG, nebo textový strukturovaný, dle ALIBu. Dále můžeme zvolit, zda se má výsledek pouze zobrazit, nebo uložit do souboru. Samozřejmě můžeme také nastavit cestu k tomuto souboru.

2.1.5 Vnitřní uzly s jedním vstupem

Tyto vnitřní uzly reprezentují algoritmy ALIBu. Konkrétně algoritmy, které přijímají jeden automat jako svůj vstup. Na plátně jsou reprezentovány ob-

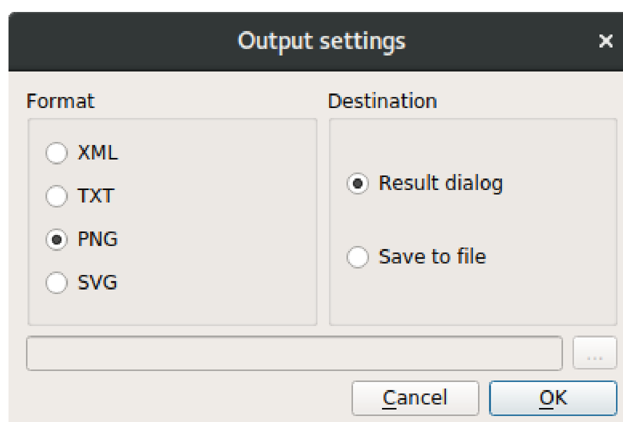


Obrázek 2.3: Návrh dialogu zadání vstupu

délníkem s názvem daného algoritmu. Otevřením kontextového menu máme možnosti *Connect* případně *Reconnect* a *Delete*. Možností *Connect* vytvoříme výstupní hranu z tohoto uzlu. Možnost *Reconnect* je dostupná pouze v případě, že tento uzel již je propojen výstupní hranou s dalším uzlem. Zvolením této možnosti zrušíme původní hranu a jsme v procesu vytváření nové výstupní hrany. Možnost *Delete* smaže tento uzel a všechny hrany se kterými je propojen.

2.1.6 Vnitřní uzly se dvěma vstupy

Tyto vnitřní uzly reprezentují algoritmy ALIBu, které přijímají dva automaty jako svůj vstup. Jedná se výhradně o algoritmy operující s jazykem, který au-



Obrázek 2.4: Návrh dialogu nastavení výstupu

tomat přijímá. V uživatelském rozhraní jsou téměř totožné jako uzly s jedním vstupem až na to, že mají právě dva.

2.1.7 Hrany

Hrany jsou čistě grafickým prvkem, avšak velmi důležitým. Propojují jednotlivé uzly diagramu algoritmů a předávají uživateli informaci o tom, jak vypadá logický model uvnitř aplikace. S hranami nejde nijak přímo interagovat, jejich vytváření a mazání je umožněno pouze přes uzly. Během návrhu bylo otázkou, jak velké úsilí investovat do vedení hran, myšleno snaze vyhnout se ostatním objektům na plátně. Zvolen byl způsob jednoduchých hran a spolehnout se, že uživatel ve vlastním zájmu bude chtít přehledný diagram algoritmů. Hrany jsou tedy vždy pravoúhlé, snaží se spojit uzly co nejkratší cestou, bez zbytečných zalomení. Hrany jsou schopny reagovat na situaci, kdy začátek hrany (výstupní bod z uzlu) je více vpravo než konec hrany (vstupní bod do uzlu) a vytvořit příslušný tvar podobný písmenům S nebo Z. Na složitější situace však hrany nereagují, podobně ignorují kolize a překrývání s obdélníky uzlů. Jak již bylo zmíněno přehledný diagram je v zájmu uživatele a pozice uzlů na plátně jsou plně v jeho režii.

2.1.8 Prezentace výsledku

Pro prezentaci výsledku je navržen jednoduchý dialog *Result*. Tento dialog obsahuje buď obrázek výsledného automatu ve formátu, který si uživatel zvolil, nebo komponentu obsahující text reprezentující automat, opět ve formátu, který si uživatel zvolil.

2.1.9 Obecné použití grafického rozhraní

Nyní si ukážeme, jak je zamýšleno použití celého uživatelského rozhraní. Uživatel otevře aplikaci, kde ho uvítá hlavní okno. Na plátně hlavního okna jsou připraveny dva uzly. Nenastavený vstupní uzel, *Input*, a výstupní uzel, *Output*. Na plátně není přítomna žádná hrana. Záleží na uživateli, kolik a jaké algoritmy chce použít. Kliknutím na tlačítko algoritmu v levé části hlavního okna a případným výběrem konkrétní varianty dojde k umístění vnitřního uzlu na plátno. Uživatel spojí uzly diagramu hranami. Stiskem pravého tlačítka myši otevře kontextové menu nad uzlem jehož výstup chce předat, vybere možnost *Connect*, poté klikne na uzel, který má předání přijmout. Pokud vše učinil správně, zobrazí se grafická hrana spojující uzly. Dalšími důležitými body je nastavení vstupního a výstupního uzlu celého diagramu. K tomuto účelu slouží dialogy *Input Settings* a *Output Settings*. V nastavení vstupu si uživatel může vybrat, jakým způsobem chce zadat automat. Během editace se mu nabízí zobrazení grafu automatu. Poté co uživatel korektně nastaví automat a potvrdí stiskem tlačítka *OK*, obdélník *Input* v diagramu algoritmů zmodrá. V nastavení výstupu si uživatele vybere formát výsledku a způsob prezentace. Nakonec uživatel stiskne tlačítko *RUN* v dolní části hlavního okna a spustí běh vyhodnocení diagramu algoritmů. Výsledek je uložen nebo prezentován, dle zvoleného nastavení.

2.2 Návrh aplikace

Architektura aplikace se odvíjí od zvoleného řešení komunikace s knihovnou ALIB, jak již bylo zmíněno dříve v sekci 1.2.2. Hlavní myšlenkou je volat funkce algoritmů přímo ze sdílených objektů ALIBu. Naplnění tohoto požadavku vedlo k rozdělení aplikace na dvě vrstvy. Vrstvu modelu starající se o logiku a komunikaci s ALIBem a vrstvu grafické části, která prezentuje a přijímá akce od uživatele. Velkou část tříd tvoří reprezentace algoritmů. Ty mají jak logickou, tak grafickou část. Další částí jsou dialogy zastupující grafickou část. Ve vrstvě modelu je ještě několik tříd se specifickou zodpovědností.

2.2.1 Zapouzdření modelu a grafiky uzlů diagramu algoritmů

Uzly jsou v aplikaci tvořeny instancí jednou z podtříd třídy `ModelBox`, logická část, a stejně odvozenou podtřídou třídy `GraphicsBox`, protože tyto dvě instance vždy existují společně a jsou zapouzdřeny v objektu `WrapperBox`. `WrapperBox` se stará o předání volání funkci obou částem. Tímto je také dosažena konzistence stavu modelu a grafiky. Instance podtříd `ModelBox` a `GraphicsBox` nemají přímý odkaz jedna na druhou, mají však přístup ke společnému objektu `WrapperBox`, obrázek 2.5.

Pro jednodušší vytváření objektů třídy `WrapperBox` byla navrhnuta třída

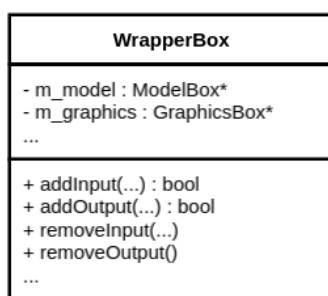
WrapperFactory. Jejím jediným úkolem je na požadavek vytvoření daného objektu WrapperBox s konkrétním uzlem, případně algoritmem z knihovny ALIB, vrátit příslušnou instanci. K tomuto účelu byl využit návrhový vzor *factory method*[9]. Pro konkrétní požadavek, vytvořit uzel *Input*, vypadá situace takto. Třída WrapperFactory vytvoří příslušné instance modelu a grafiky. Vytvoří a vrátí objekt WrapperBox, který obsahuje právě vytvořené instance, viz obrázek 2.6

2.2.2 Grafická vrstva

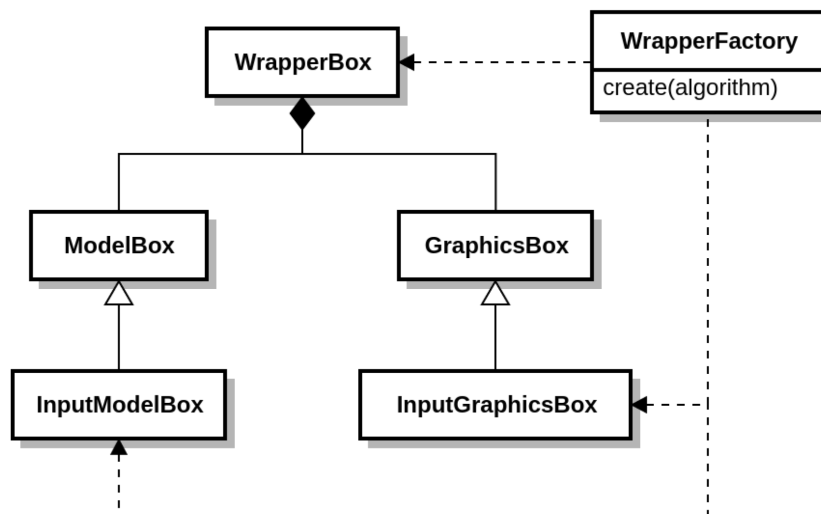
Tato vrstva se stará o prezentaci obsahu uživateli a o předání uživatelových vstupů do modelu. Sestává se ze tříd dialogů, které jsou potomky dialogů frameworku Qt. Druhou významnou částí této vrstvy je hierarchie tříd třídy GraphicsBox. Objekty těchto tříd reprezentují obdélníky uzlů diagramu algoritmů na plátně hlavního okna. Posledním zajímavým prvkem této vrstvy je třída GraphicsConnection, která reprezentuje hrany v diagramu algoritmů.

2.2.2.1 Dialogy

Aplikace nabízí tři dialogy, se kterými uživatel může manipulovat. Tyto jsou hlavní okno obrázek 2.1, *Input settings* obrázek 2.3 a *Output settings* obrázek 2.4. Pro vytvoření dialogů byl využit framework Qt, konkrétně každý dialog je potomkem třídy QDialog. Dialogy předávají své nastavení do vrstvy modelu, za předpokladu že je uživatel akceptuje. Speciální pozornost si zaslouží dialog *Input settings* a jemu odpovídající třída InputDialog. Tento dialog prezentuje náhled grafu konečného automatu, pokud je vstup zadaný do textové komponenty platnou reprezentací automatu. Toto vyhodnocení náhledu se děje s každou změnou textu. K reakci na změnu je využit systém signálů frameworku Qt a vyhodnocení proběhne pomocí volání metod tříd z vrstvy modelu aplikace.



Obrázek 2.5: Diagram třídy WrapperBox



Obrázek 2.6: Diagram použití vzoru *facory method*

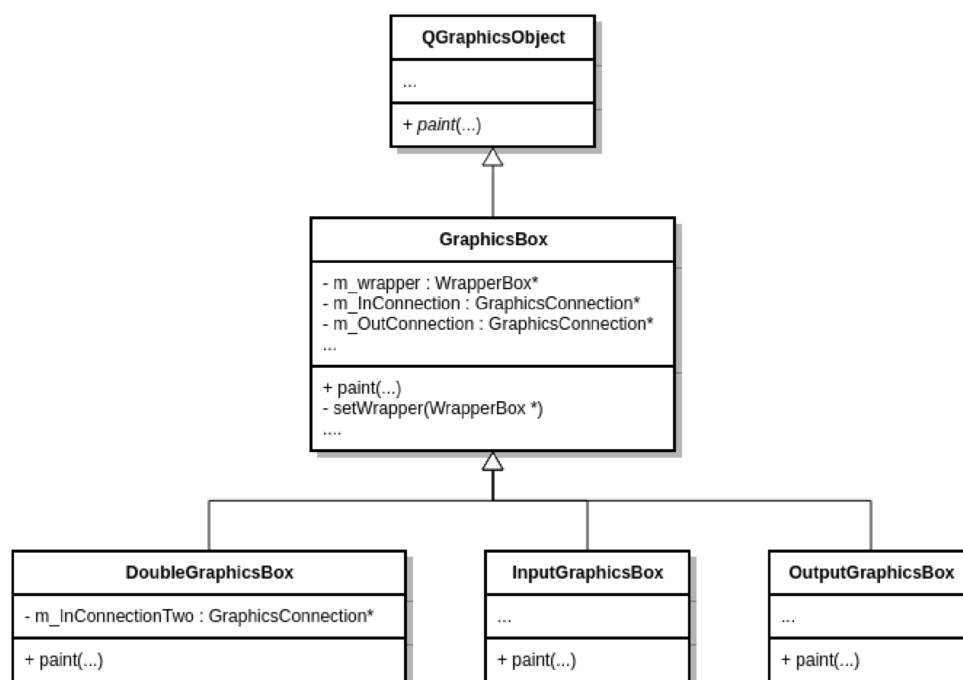
2.2.2.2 Grafická část uzlů diagramu algoritmů

Grafická část uzlů diagramu algoritmů je tvořena třídou `GraphicsBox` a jejími podtřídami, obrázek 2.7. Tato třída je sama potomkem `QGraphicsObject` frameworku Qt, díky tomu mohou být instance této hierarchie umístěny na plátno a reagovat na uživatelovy akce. Touto dědičností se také třídy zavazují k implementaci metody `paint(...)`, která je volána pro vykreslení objektů na plátno hlavního okna. Reakce na uživatelovy akce je umožněna využitím systému signálů framworku Qt. Objekty tak mohou nabídnout kontextové menu s možností vyvolání dialogů nebo vytvoření hran.

Objekty této hierarchie také komunikují s objekty hran `GraphicsConnection`. Předávají jim informace o bodech ke kterým je hrana připojena a případně informaci o tom, že je hrana odpojena a je potřeba ji smazat.

2.2.2.3 Hrany

Hrany jsou tvořeny instancemi třídy `GraphicsConnection`, která dědí z třídy Qt frameworku, `QGraphicsObject`. Jsou tedy vykreslovány na plátno a k tomu slouží poděděná metoda `paint(...)`. Během navrhování existovala fáze, kdy objekt hrany byl atributem třídy `GraphicsBox`. To znamenalo, že každý uzel si spravoval svou výstupní hranu. Tento návrh byl však vyměněn za hrany, které jsou nezávislé a instance grafické části uzlů drží pouze odkaz na připojené hrany. Tímto rozhodnutím se zpřehlednil systém mazání hran.



Obrázek 2.7: Diagram hierarchie podtříd třídy `GraphicsBox`

Vytvoření nové hrany je reakce na dvě uživatelovy po sobě jdoucí akce. Za tímto účelem, byla navrhnuta třída `ConnectionHelper` ve vrstvě modelu. Samotné objekty hran nemají ekvivalent ve vrstvě modelu.

2.2.3 Vrstva modelu

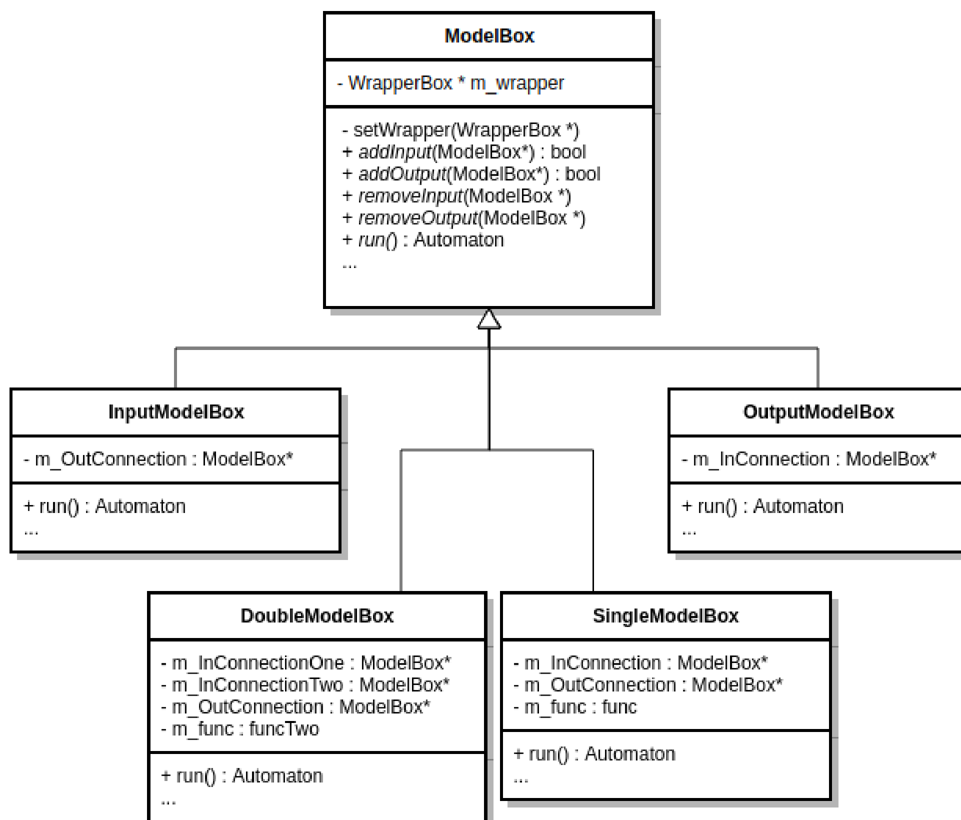
Ve vrstvě modelu najdeme hierarchii tříd odvozené od `ModelBox`, které reprezentují logickou část diagramu algoritmů a přímo volají funkce ze sdílených objektů knihovny ALIB. Dále zde najdeme třídy se specifickými zodpovědnostmi, které jsou využívány z celé aplikace, například třídy `Converter` a `ConnectionHelper`. Tato tenká vrstva je prostředníkem mezi uživatelským rozhraním a funkcemi ALIBu, nenajdeme zde žádné složité algoritmy.

2.2.3.1 Logická část uzlů diagramu algoritmů

Návrh logické části uzlů je podobný jako návrh části grafické. Najdeme zde hierarchii tříd, tentokrát odvozenou od třídy `ModelBox`, obrázek 2.8. Instance těchto tříd tvoří logický model diagramu algoritmů. Třídy `SingleModelBox` a `DoubleModelBox` implementují uzly reprezentující algoritmy knihovny ALIB, s jedním a se dvěma vstupy, příslušně. Všechny podtřídy třídy `ModelBox` implementují metodu `run()`, která provede akci pro daný uzel. Zavolá daný al-

goritmus z ALIBu, vytvoří objekt automatu, zobrazí či uloží výsledek, podle toho o jaký uzel se jedná.

Toto je také mechanizmus, kterým se celý diagram algoritmů vyhodnotí. Vyhodnocení začne u výstupního uzlu, který zavolá metodu `run()` na uzlu svého vstupu `m_InConnection`. Ten zavolá `run()` na svých vstupech atd., poslední volané jsou vstupní uzly. Instance třídy `InputModelBox`, předá objekt uživatelem nastaveného automatu. Každý objekt této řady volání dostane od svého předchůdce automat. Zavolá nad ním metodu algoritmu z knihovny ALIB a výsledek předá jako návrat volání metody `run()`. Nakonec se výsledný automat zobrazí, nebo uloží, podle uživatelovy volby, v instanci třídy `OutputModelBox`.



Obrázek 2.8: Diagram hierarchie podtříd třídy `ModelBox`

2.2.3.2 Pomocné třídy se specifickou zodpovědností

Z návrhu předchozích částí vyplynulo, že některé akce by měly mít vlastní třídu se zodpovědností za požadované akce. Takovými třídami jsou například `ConnectionHelper` a `Converter`.

V prvním případě se jedná o pomocnou třídu pro vytvoření objektů `Graphics-`

`Connection`. Tyto objekty reprezentují grafické hrany diagramu algoritmů. Protože můžeme v jednom okamžiku tvořit pouze jednu hranu, byl pro tuto třídu použit návrhový vzor *singleton*[10]. Pro úspěšné vytvoření nové hrany musí uživatel provést dvě akce v grafickém rozhraní. Vybrat dva uzly, odkud a kam hrana povede. Po zahájení první akce je tato informace uchována právě v instanci třídy `ConnectionHelper`. V případě, že uživatel neučiní platný druhý krok, je stav resetován. Pokud však uživatel učiní platný druhý krok, objekt vytvoří novou instanci `GraphicsConnection` a předá jí příslušné informace, které uzly spojuje.

Třída `Converter` sdružuje znalost o konvertování objektů automatů knihovny ALIB do textové formy, například XML formátu, stejně tak vytvoření nového automatu z textu. Tato třída vystavuje všechny své metody jako statické.

Další pomocnou třídou je `GraphvizIntegrator`, která, jak jméno napovídá, zodpovídá za propojení aplikace s knihovnou programu *Graphviz*[8]. Tuto třídu využívají dialogy *Input sttings* a *Result*. Třída nabízí dvě metody, vytvoření obrázku v paměti, nebo přímé uložení obrázku do souboru.

Kapitola 3

Realizace

3.1 Framework Qt

Framework Qt je multiplatformní aplikační rámec využívaný pro tvorbu aplikací s grafickým uživatelským rozhraním podporující více softwarových a hardwarových platforem.

Použití frameworku Qt je přímo součástí zadání. Není však určeno, jak přesně Qt využít. Dokumentace[11] frameworku popisuje více než jeden způsob, jak implementovat uživatelské rozhraní. Jednou možností je Qt Quick s modulem QML. Tento přístup je zaměřený na nové technologie, dotykové ovládání a moderní vzhled. Nabízí možnosti animací a vlastních grafických efektů; relativně snadné vytvoření rozhraní s unikátním vzhledem a chováním. GUI za pomoci tohoto přístupu je implementováno jazyky QML, JavaScript.

Druhou možností je využít Qt Widgets, která ve frameworku existuje podstatně déle. Tento přístup je doporučen pro aplikace desktopového charakteru. Umožňuje přirozený vzhled aplikace na systému, kde je spuštěna. Nabízí paletu komponent uživatelského rozhraní, které jsou v tomto odvětví standartem. GUI pomocí tohoto přístupu je implementováno jazykem C++.

Při přípravě realizace jsem zkusil oba přístupy. Ukázalo se, že využití Qt Quick vyžaduje jisté znalosti JavaScriptu, kterými nedisponuji. Dále se projevil fakt, že tato část frameworku se v poslední době měnila a dokumentace plně neodpovídá posledním změnám. Narazil jsem na příklady použití z minulých verzí, které již nejsou podporovány. Rozhodnutí Qt Quick a modul QML nevyužít nakonec stvrdila skutečnost, že tento přístup nenabízí přednastavené komponenty, které jsou v GUI očekávané. Tento přístup by byl tedy pracnější, náročnější na nové znalosti a dokumentace frameworku ho pro naše užití ani nedoporučuje.

Výpis kódu 3.1: Třída GraphicsBox a metoda paint()

```
class GraphicsBox : public QGraphicsObject
{
    Q_OBJECT
public:
    GraphicsBox(QString text, ...);
    virtual ~GraphicsBox();
    virtual void paint(QPainter *painter, ...);
    ...
}

...

void GraphicsBox::paint(QPainter *painter, ...)
{
    painter->setFont(m_font);
    painter->setPen(Qt::white);
    painter->fillRect(m_boundRect,m_color);
    prepareGeometryChange();
    painter->drawText(m_boundRect, Qt::AlignCenter,
                      m_text, &m_boundRect );
    m_boundRect.adjust(-BOX_MARGIN, -BOX_MARGIN,
                        BOX_MARGIN, BOX_MARGIN);
}
```

Implementace uživatelského rozhraní tedy byla provedena pomocí Qt Widgets. Tento přístup využívá C++ třídy nabízené Qt moduly *core*, *gui*, *widgets* a *xml*. Vytvořili jsme tedy vlastní třídy, které dědí z vybraných Qt tříd a implementují naše GUI. Jako příklad uvádím výpis kódu 3.1, který zachycuje dědění z třídy Qt frameworku. Třída `QGraphicsObject` umožňuje jejím instancím umístění na plátno v grafickém rozhraní aplikace. Třída `GraphicsBox` implementuje metodu vykreslení objektu na plátno, `paint()`. Vidíme, že metoda popisuje nastavení fontu, barvy obdélníku, vykreslení jména uzlu a nastavení okrajů okolo textu. Tato metoda je zavolána objektem `QPainter` kdykoliv komponenta plátna vyžadá překreslení obsahu.

3.2 Propojení s knihovnou ALIB

Způsob komunikace aplikace s knihovnou již byl prezentován v sekcích 1.2.2 a 2.2.3.1. Nyní se podíváme na jeho implementaci. Logická část každého uzlu diagramu algoritmů je instancí jedné z podtříd třídy `ModelBox`. Implementuje

tedy virtuální metodu `run()`. Podíváme se, jak je tato metoda implementována ve třídě `SingleModelBox`, výpis kódu 3.2.

Výpis kódu 3.2: Metoda run()

```
automaton::Automaton *SingleModelBox::run()
{
  automaton::Automaton *res = NULL;
  if(m_InConnection)
    res = m_InConnection->run();
  if(res)
  {
    try{
      res->setData(m_func(*res).getData());
    } catch (exception::CommonException e) {
      QString mes =
        QString::fromStdString(e.getCause());
      AlibExceptionHandler::getInstance().
        setMessage(mes);
      res = NULL;
    }
  }
  return res;
}
```

První podmínkou zkontroluji, že k této instanci je připojen uzel, na kterém můžu zavolat `run()`. Pokud ano, volání provedu. Pokud ne, diagram algoritmů není validní a vrátím `NULL`, jako ukazatel na výsledný automat. Druhá podmínka kontroluje, že uzel před námi provedl metodu `run()` v pořádku a vrátil ukazatel na automat; pokud vrátil `NULL`, předám ho dál. Pokud obě podmínky platí, zavolám funkci ALIBu – algoritmus tohoto uzlu – a změním data objektu automatu podle výsledku. V případě, že došlo k výjimce, předám informace objektu `AlibExceptionHandler` a vrátím neplatný ukazatel.

Výpis kódu 3.3: Metoda run()

```
typedef automaton::Automaton(*func)
        (const automaton::Automaton&);
```

Funkce ALIBu, kterou mám zavolat, je uložena v instanční proměnné `m_func`. Typ této proměnné je ukazatel na metodu, viz výpis kódu 3.3. Tato proměnná je naplněna při vytváření instance modelu ve statické metodě třídy `WrapperFactory`, výpis kódu 3.4. Příklad uvádí případ pro `SingleModelBox`, obdobně je tomu pro `DoubleModelBox`. Modely vstupního a výstupního uzlu nepotřebují ukazatel na metodu z ALIBu. Chování jejich metody `run()` si uživatel nastavil v příslušných dialozích.

Výpis kódu 3.4: Metoda run()

```
WrapperBox *WrapperFactory::create
        (WrapperFactory::Algorithms algorithm, ...)
{
  ...
  switch( algorithm )
  {
  ...
  case DETERMINIZE:
    return new WrapperBox(
           new SingleModelBox( &automaton::
             determinize::Determinize::determinize ),
           new GraphicsBox( "Determinize", ... )
           );
  ...
  }
}
```

Model každého uzlu, který reprezentuje algoritmus, má tedy ukazatel na funkci z knihovny ALIB. Při vyhodnocování diagramu algoritmů nastane okamžik, kdy se na tomto objektu zavolá metoda `run()`. Instance si nejdříve zažádá o automat, nebo automaty od svých předchůdců. Zavolá funkci z ALIBu a výsledný automat vrátí.

Kapitola 4

Testování

Předchozí části této práce jsou o návrhu a implementaci uživatelského rozhraní. Není tedy překvapením, že v této části se věnuji *usability testing* – uživatelskému testování – implementované aplikace. Cílem tohoto testování je ověřit, že jsem GUI navrhl rozumným způsobem a že je použitelné pro reálné uživatele. Samozřejmě chci také odhalit co nejvíce chyb a nedokonalostí. Dále chci dostat zpětnou vazbu uživatelů a návrhy pro budoucí změny.

4.1 Metodika

Zvolený způsob testování aplikace je moderované uživatelské testování. To znamená, že jsem osobně přítomen během testování, uvádím ho a ptám se na otázky hodnotící aplikaci. Otázky a úkoly jsou dopředu připravené. Stejné otázky a úkoly jsou zadávány ve stejném pořadí všem uživatelům. Snažím se neovlivnit výsledek, během jednotlivých úkolů testování jsem pouze pozorovatelem. Aplikaci testuje vždy jen jeden uživatel. Natáčím záznam obrazovky zařízení, na kterém uživatel interaguje s aplikací. V našem případě natáčím také zvuk toho, co uživatel říká, požádal jsem ho, aby uvažoval nahlas a popisoval, co a proč právě dělá. Toto uvažování nahlas je jedna z technik uživatelského testování *Concurrent Think Aloud*[13].

4.2 Průběh

Uskutečnil jsme celkem tři testovací sezení. Všechna sezení probíhala stejným způsobem, se stejným počítačem. To vše za účelem stejných podmínek pro každého uživatele. Sezení můžu rozdělit na čtyři části. Úvod sezení, úvodní otázky, testovací úlohy a závěrečné otázky.

Na úvod sezení jsem představil uživateli o jakou aplikaci se jedná, abych sjednotil jejich očekávání. Vysvětlil jsem mu metodiku testování a požádal ho, aby

mi oznámil, pokud považuje úkol za splněný, případně že ho splnit nedokáže. Vyžádal jsem si také souhlas s natáčením uživatele.

Úvodními otázkami jsem zjistil od uživatele jeho znalosti a očekávání.

- Jak byste ohodnotil své znalosti konečných automatů na škále 1–5?
  (1 – nevím co jsou to KA, 5 – dokážu sepsat definici KA i pseudokód algoritmů nad KA)
- Slyšel jste někdy o projektu automatové knihovny ALIB na ČVUT FIT?
- Pokud ano, jaká je vaše znalost knihovny (1–5)?
  (1 – pouze jsem slyšel název, 3 – dokáži ALIB použít k čemu potřebuji, 5 – jsem členem vývojového týmu)
- Co od uživatelského rozhraní očekáváte? Jak by zhruba mělo fungovat?

Po těchto otázkách jsme uživatele posadili k počítači s otevřenou aplikací a zadali mu postupně sedm úkolů. Po každém úkolu jsme položili následující otázky.

- Jak složitý pro vás byl tento úkol (1–5)?
  (1 – příliš, nedokončil jsem ho; 5 – snadný, zvládl jsem ho na první pokus)
- Co způsobilo problémy, proč byl úkol těžký?
- Chcete k úkolu dodat komentář?

V závěrečné části jsme položili další otázky shrnující celou uživatelovu zkušenost. Po těchto otázkách většinou následovala krátká debata, popřípadě návrat k některým úkolům a jejich opětovné řešení s novými informacemi.

- Jak byste celkově hodnotil ovladatelnost aplikace (1–5)?
  (1 – neovladatelné, 5 – snadné a intuitivní)
- Jaká část GUI vám dělala největší problém, nejvíc vás zmátla, byla nepříjemná?
- Postrádáte nějakou konkrétní funkci GUI, kterou využíváte v jiných programech?
- Co se vám na GUI líbilo, nelíbilo a co byste doporučil změnit?

4.2.1 Testovací úkoly

1. Vygenerujte náhodný automat a zobrazte jej ve formátu TXT.
2. Změňte abecedu generovaného automatu, místo znaku **a** generujte **f**.
3. Vygenerujte náhodný automat, převeďte ho na normalizovaný deterministický, a uložte do souboru ~/alib_examples/DFA.xml ve formátu XML.
4. Odstraňte z plochy vše krom uzlu výstupu.
5. Načtěte automaty ze souborů ~/alib_examples/NFA1.txt a NFA2.xml. Proveďte jejich konkatenaci pomocí epsilon přechodů. Odstraňte epsilon přechody metodou odchozích hran. Výsledný automat zobrazte ve formátu PNG.
6. Změňte metodu odstranění epsilon přechodů na metodu příchozích hran. Výsledek opět zobrazte jako obrázek formátu PNG.
7. Součin automatů NFA1.txt a NFA2.xml pronikněte s automatem ze souboru NFA3.xml. Výsledek převeďte na úplný deterministický automat, přejmenujte stavy a výsledek uložte jako obrázek DFA.svg ve formátu SVG.

4.3 Výsledky

Výsledky testování můžu rozdělit na několik oblastí. První oblastí jsou výsledky kvantitativních otázek, druhou odpovědi na kvalitativní otázky. Třetí částí je mé pozorování a jím odhalené nedostatky a chyby aplikace.

Výsledky kvantitativních otázek jsou zpracovány v tabulce 4.1. Pro přehlednost jsou otázkám přiděleny zkratky – U1 až U3 jsou otázky z úvodní části testovacího sezení. U2 je otázka na znalost knihovny ALIB. Odpověď ANO je reprezentována číslem jedna, odpověď NE číslem nula. T1 až T7 jsou otázky na uživatelův dojem z jednotlivých úkolů a Z1 je otázka na celkový dojem z aplikace.

Z hodnocení od uživatelů vidím, že první dojem z aplikace je v polovině bodové škály. Toto se také shoduje s mým pozorováním; uživatel ze začátku trochu tápe, než přijde na ovládání plátna. Následuje série postupně složitějších úkolů. Hodnocení se zlepšuje. Poslední úkol vyžaduje znalost anglických názvů algoritmů. Toto byl problém a ani jeden uživatel úkol nesplnil, přestože si to někteří mysleli.

otázka	uživatel 1	uživatel 2	uživatel 3	průměr
U1	3	2	3	2,67
U2	1	0	1	0,67
U3	1	-	1	1
T1	3	3	3	3
T2	5	5	5	5
T3	4	3	4	3,67
T4	5	5	5	5
T5	4	4	4	4
T6	5	5	5	5
T7	3	1	4	2,67
Z1	4	3	4	3,67

Tabulka 4.1: Výsledky kvantitativních otázek

Na kvalitativní otázky jsem dostal mnoho odpovědí a komentářů. Některé opakující se si představíme a vyjádřím se k nim.

- **Zadání automatu pomocí grafu** Tato funkcionalita není u této aplikace zamýšlena, při představování aplikace jsem tento fakt měl zmínit.
- **Označení možných vstupů a výstupů uzlů** Grafickou reprezentaci vstupů a výstupů uzlů jsem již uvažoval pro další vývoj.
- **Označení skupiny uzlů a akce nad skupinou (přesun, smazání)** I o této funkcionalitě jsem uvažovali pro další vývoj.
- **Informování uživatele o úspěšném uložení výsledku do souboru** Tato funkce GUI znatelně chybí a v návrhu mi unikla.
- **Nepřehledný dialog *Input settings* a nestandardní otevírání souboru** Tento dialog doplatil na postupný vývoj a vyžaduje přeorganizování.

Celkové hodnocení aplikace rezonuje se skutečností, že je ve velmi raném stádiu. Uživatelé odhalili i několik chyb aplikace. Nejvíce připomínek se týkalo dialogu *Input settings* a plátna s diagramem algoritmů. Diagram algoritmů byl ale také oceněn jako elegantní řešení pro spojení algoritmů do větších celků.

Z pozorování interakce uživatele s aplikací si odnáším několik poznatků. Aplikace postrádá *tooltipy*, popisky, které by více objasnily tlačítka algoritmů a uzly diagramu na plátně. Propojování hran není dostatečně intuitivní. Vytvářená hrana by například mohla dočasně končit na kurzoru. Červená barva obdélníku *Input* je velmi intuitivní prvek, který snadno předává informaci o

jeho nastavení. Podobný prvek by mohl být použit pro označení uzlů, které postrádají vstup.

4.4 Výhled do budoucna

Implementovaná aplikace je v raném stádiu, avšak se znalostí knihovny ALIB použitelná. Testování odhalilo slabá místa uživatelského rozhraní a uživatelé mi předali zajímavé návrhy na změny a vylepšení. Zde uvádím přehled návrhů pro další vývoj.

- Přeorganizování dialogu *Input settings*
- Export zadaného vstupu samostatně z kontextového menu uzlu
- Detekce formátu textu vkládaného do dialogu *Input settings*
- Zvolení algoritmu (zamáčknutí tlačítka) a přidání uzlu na přesné místo na plátně (klik myší)
- Posouvání plátna na *drag* akci myší
- Označení skupiny uzlů a akce nad skupinou (posunutí, smazání)
- Grafická reprezentace vstupů a výstupů uzlů
- Grafická reprezentace uzlů, kterým chybí vstup
- Grafické znázornění stavu vytváření nové hrany
- Lépe uživateli sdělit informaci o nekompatibilním automatu a algoritmu
- Akce s uzly pouze za pomoci klávesnice (smazání na stisk delete)
- Informovat uživatele o úspěšném uložení výsledku do souboru
- Nahrazení uzlu uzlem, přepojení hran

Další oblastí pro rozvoj aplikace je rozšíření funkcí zpřístupněných z knihovny ALIB, přidání dalších objektů, nejen automatů. Implementované řešení je navíc možné v budoucnu upravit a uskutečnit dynamickou spolupráci uživatelského rozhraní a knihovny ALIB, viz. třetí možnost řešení v sekci 1.2.

Závěr

V bakalářské práci jsem se zabýval návrhem a implementací aplikace poskytující grafické uživatelské rozhraní pro knihovnu ALIB. Funkce zpřístupněné pomocí GUI byly již v zadání omezeny na oblast konečných automatů. Úmyslem celé aplikace je zjednodušit a zpříjemnit používání knihovny. Práce byla rozdělena do čtyř kapitol, které se věnují dílčím cílům a skrze ně plní hlavní cíl práce.

V první kapitole jsem analyzoval existující aplikace pro práci s konečnými automaty. Dle očekávání jsem zjistil, že žádná existující aplikace nespolupracuje s knihovnou ALIB. Dokonce neexistuje žádná aplikace, která by za hlavní cíl měla skládání dílčích algoritmů s automaty. Dále jsem provedl analýzu projektu ALIB. Identifikoval oblast pracující s konečnými automaty a popsal její třídy a algoritmy. Na konci této kapitoly jsem popsal ovládání projektu ALIB z příkazové řádky, které posloužilo nejen jako motivace pro celý projekt, ale také jako inspirace pro diagram algoritmů. Výsledkem této kapitoly jsou požadavky na návrh a implementaci aplikace.

V druhé kapitole jsem navrhl podobu uživatelského rozhraní. V této části jsem využil poznatky z existujících aplikací, rozboru ALIBu i jeho textového ovládání. Výsledkem je okenní aplikace s plátnem pro práci s diagramem algoritmů. Dále jsem se v této kapitole věnoval návrhu tříd aplikace. Rozdělení do dvou vrstev grafické a modelové.

Ve třetí kapitole jsem se věnoval implementaci a testování. Představuji zde využití frameworku Qt požadovaného ze zadání, implementaci tříd a metod z návrhu a vybrané propojení aplikace GUI a projektu ALIB.

Poslední kapitola popisuje uživatelské testování, jeho vyhodnocení. Kapitola také obsahuje sekci s návrhy pro další vývoj, které vznikly na základě poznatků z testování.

Výsledkem této práce je funkční aplikace grafického rozhraní ke knihovně ALIB, omezená na práci s konečnými automaty. Ačkoliv je aplikace použitelná a splňuje požadavky, které jsem si stanovil, jedná se spíše o ověření konceptu, než o úplné řešení. Pro ideální využití aplikace studenty a vyučujícími je potřeba dodat do GUI více informací a upravit ovládání, aby bylo intuitivnější. Aplikace však směřuje správným směrem a nápady pro další vývoj jsou zmíněny v poslední sekci poslední kapitoly.

Literatura

[1] Ing. Trávníček, J.: Automata library [online]. [cit. 2016-12-04]. Dostupné z: https://gitlab.fit.cvut.cz/travnja3/automata-library

[2] Wallace, E.: Finite State Machine Designer [online]. [cit. 2016-12-04]. Dostupné z: http://madebyevan.com/fsm/

[3] Zuzak, I.; Jankovic, V.: FSM Simulator [online]. [cit. 2016-12-04]. Dostupné z: http://ivanzuzak.info/noam/webapps/fsm_simulator/

[4] Burch, C.: Automaton Simulator [online]. [cit. 2016-12-04]. Dostupné z: http://www.cburch.com/proj/autosim/index.html

[5] White, T. M.: jFAST - the Finite Automata Simulator [online]. [cit. 2016-12-04]. Dostupné z: http://jfast-fsm-sim.sourceforge.net/

[6] Žďárek, J.; Mi-La: Automata editor [online]. [cit. 2016-12-04]. Dostupné z: https://sourceforge.net/projects/automataeditor/

[7] Çuka, E.: Finite-Automata [online]. [cit. 2016-12-04]. Dostupné z: https://sourceforge.net/projects/finiteautomataconverter/

[8] Graphviz - Graph Visualization Software [online]. [cit. 2016-12-04]. Dostupné z: http://www.graphviz.org/

[9] Gamma, E.; Helm, R.; Johnson, R.; aj.: *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, první vydání, 1994, ISBN 0-201-63361-2, 107-116 s.

[10] Ref. 9, 127-134 s.

[11] User Interfaces | Qt 5.8 [online]. [cit. 2017-05-05]. Dostupné z: https://doc.qt.io/qt-5/topics-ui.html

[12] Scholten, R.: Drupal - Usability testing [online]. [cit. 2017-05-02]. Dostupné z: https://www.drupal.org/docs/develop/usability-testing

[13] Usability.gov - Improving the User Experience [online]. [cit. 2017-05-03]. Dostupné z: https://www.usability.gov/

Příloha **A**

Seznam použitých zkratek

ALIB Automata library – Knihovna pro práci s automaty, gramatikami a dalšími strukturami spojenými s matematickou implementací algoritmů

GUI Graphical user interface – Grafické uživatelské rozhraní

PNG Portable Network Graphics – Formát obrázků vhodný pro přenos po síti

SVG Scalable Vector Graphics – Vektorový formát obrázků

FSM Finite state machine – Konečný stavový automat

DFA Deterministic finite automaton – Deterministický konečný automat

NFA Nondeterministic finite automaton – Nedeterministický konečný automat

NFA-ε Nondeterministic finite automaton with epsilon transitions – Nedeterministický konečný automat s epsilon přechody

JSON JavaScript Object Notation – Strukturovaný formát pro výměnu dat

XML Extensible markup language – Strukturovaný formát pro výměnu dat

DOT Čistě textový, strukturovaný popis grafu

UML Unified Modeling Language – grafický jazyk pro vizualizaci systémů

Příloha B

Obsah přiložené SD karty

```
readme.txt .................... stručný popis obsahu SD karty
src
  automata-library ............ zdrojové kódy knihovny ALIB
  impl ........................ zdrojové kódy implementace
  thesis ...................... zdrojová forma práce ve formátu LaTeX
text .......................... text práce
  thesis.pdf .................. text práce ve formátu PDF
```