FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Evoluce webového rozhraní knihovny algoritmů
Student:	Hana Litavská
Vedoucí:	Ing. Jan Trávníček, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

- Seznamte se se současnou implementací webového rozhraní ALT, s již reportovanými nedostatky a výsledky uživatelského testování webového rozhraní ALT publikovanými v předchozí práci [1].
- Doplňte seznam existujících nedostatků o zjištěné v uživatelském testování z předchozí práce a navrhněte jak stávající i nové nedostatky odstranit.
- Implementujte změny do kódu webového rozhraní ALT a po jednotlivých funkčních celcích provádějte nasazení vylepšovaného webového rozhraní ALT.
- Ověřte odstranění známých i nově nalezených nedostatků opětovným provedením uživatelského testování.

[1] Michael Vrána. Knihovna algoritmů ALT - webové rozhraní. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2020.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 30. ledna 2022 v Praze.

Bakalářská práce

EVOLUCE WEBOVÉHO ROZHRANÍ KNIHOVNY ALGORITMŮ

Hana Litavská

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Trávníček, Ph.D.
11. května 2022

Odkaz na tuto práci: Litavská Hana. *Evoluce webového rozhraní knihovny algoritmů.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Obsah

Seznam obrázků

Seznam výpisů kódu

V první řadě děkuji vedoucímu práce Ing. Janu Trávníčkovi, Ph.D. za odborné vedení, ochotu a čas, který mi v průběhu zpracování práce věnoval. Mé poděkování dále patří mé mámě, bratrům, příteli, kocourovi Mourovi a také sobě samé za nekonečnou podporu a trpělivost při celém studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při 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. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, 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 11. května 2022 .....

Abstrakt

Práce se zaměřuje na další rozvoj webového rozhraní knihovny ALT věnované datovým strukturám a algoritmům z oblasti stringologie, arbologie a částečně grafů. Původní verze webového rozhraní je nejprve analyzována se zaměřením na typické případy užití, uživatelské rozhraní, použité technologie i implementaci.

Praktická část práce obsahuje analýzu, návrh a samotnou implementaci řešení existujících nedostatků webového rozhraní, především chybějících funkcionalit a chyb. Veškeré změny jsou prováděny v souladu s existujícím řešením s využitím knihoven React a Redux. Konečná verze je otestována pomocí uživatelského testování s důrazem na testování použitelnosti, díky kterému je získána zpětná vazba k novým změnám ale i celkově k webovému rozhraní knihovny ALT.

Klíčová slova Algorithms Library Toolkit, evoluce webového rozhraní, uživatelská přívětivost, testování použitelnosti, React, Redux, Typescript

Abstract

This bachelor thesis is dedicated to providing a clear view of improvements related to a web interface of a library named ALT which is dedicated to data structures, algorithms related to stringology, arbology, and partly graphs. The original version of the web is analyzed with a strong focus on use cases, user interface, used technology, and implementation.

Content of the practical part of this bachelor thesis includes analysis, solution proposals of improvements related to existing bugs, new features, and limitations of existing implementation and implementation in general. All the changes are completely aligned with the used technology using React and Redux. The final version is tested using usability testing methodology to receive valuable feedback related to implemented changes and feedback in general related to the web interface of the ALT library.

Keywords Algorithms Library Toolkit, evolution of the web user interface, user experience, usability testing, React, Redux, Typescript

Seznam zkratek

ALT	Algorithms Library Toolkit
BI-AAG	bakalářský předmět Automaty a gramatiky vyučovaný na FIT ČVUT
DKA	deterministický konečný automat
DOM	Document Object Model
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
NKA	nedeterministický konečný automat
SVG	Scalable Vector Graphics
UI	user interface neboli uživatelské rozhraní
UX	user experience neboli uživatelská zkušenost
XML	Extensible Markup Language

Úvod

Webová rozhraní umožňují přístup k nejrůznějším aplikacím z internetových prohlížečů bez nutnosti instalace samotného programu a jeho závislostí. Zmíněná dostupnost uživatele motivuje k jeho častějšímu používání. Tato skutečnost hraje velkou roli v případě aplikací využívaných k výuce ve školách či samostudiu. Pro studenty je především atraktivní rychlý přístup, jednoduché zadání vstupu a přívětivé uživatelské prostředí.

Webové rozhraní knihovny algoritmů zpřístupňuje používání knihovny Algorithms Library Toolkit (ALT). ALT je knihovna algoritmů vyvíjená na Fakultě informačních technologií ČVUT. Umožňuje pracovat s datovými strukturami jakými jsou například konečné automaty, gramatiky či regulární výrazy. Zároveň obsahuje algoritmy, které nad těmito datovými strukturami operují.

Bakalářská práce navazuje na práci Knihovna algoritmů ALT – webové rozhraní Michaela Vrány [1], v rámci které byla implementována první funkční verze. Webové rozhraní je aktuálně využíváno během výuky předmětu BI-AAG, kde vznikla potřeba webové rozhraní vylepšit na základě posbírané zpětné vazby.

Výsledek práce bude prospěšný především pro studenty jakožto podpůrný nástroj při studiu předmětů věnujících se formálním jazykům. Vyřešení aktuálních nedostatků nabídne studentům jednodušší zadávání vstupu, srozumitelnější vizualizaci informací o algoritmech, nové funkcionality, jednodušší orientaci a eliminaci chyb. Především pomoc dalším studentům mě motivovala ve výběru tématu bakalářské práce.

Práce se zabývá analýzou aktuálně nasazené verze, řešením jejích problémů a chybějících funkcionalit a uživatelským testováním nové verze.

První část práce se věnuje analýze stavu webového rozhraní před implementací vylepšení. Po zjištění původního stavu následuje část zaobírající se nedostatky nalezenými uživateli a nedostatky z manuálního testování při analýze původního stavu. V této části jsou nedostatky nejprve analyzovány, následuje návrh řešení na základě zjištěných faktů z analytické části a nakonec navazuje popis implementace zvoleného návrhu. Nová verze webového rozhraní je v poslední části otestována pro ověření nového stavu.

Cíl

Cílem bakalářské práce je vyřešit aktuální nedostatky webového rozhraní knihovny ALT a vylepšit tak jeho uživatelskou přívětivost. Prvním dílčím cílem práce je analýza aktuálně nasazeného řešení z předchozí bakalářské práce Knihovna algoritmů ALT – webové rozhraní Michaela Vrány [1], ověření reportovaných nedostatků nalezených během používání tohoto řešení a případný sběr dalších problémů z manuálního testování a prvotní analýzy. Dále je cílem tyto nedostatky analyzovat, navrhnout adekvátní řešení a toto řešení implementovat. Implementovaná řešení budou průběžně nasazována. Po dokončení implementace je kladeno za cíl ověřit výsledný stav webového rozhraní pomocí uživatelského testování.

Bakalářská práce má přínos především pro studenty, kteří tento nástroj využívají při studiu předmětu BI-AAG. Webové rozhraní bude pro uživatele eliminovat předpoklad znalosti formátů vstupu datových struktur specifických pro ALT, výstupy budou vizualizované ve formátu blízkém uživateli, informace o algoritmech budou na webu jednoduše dohledatelné a budou eliminovány nalezené chyby.

Kapitola 1
Pojmy řešené domény

Specifická doména, které se webové rozhraní věnuje, nejprve vyžaduje definici používaných pojmů. Zadefinovány budou především základní termíny z oblasti formálních jazyků, které budou v následujících částech zmiňovány.

1.1 Graf

▶ **Definice 1.1.** *Orientovaný graf G je uspořádaná dvojice (V, E), kde*

- *V je neprázdná konečná množina vrcholů (nebo také uzlů) a*
- *E je množina orientovaných hran (nebo také šipek).*

Orientovaná hrana $(u, v) \in E$ je uspořádaná dvojice různých vrcholů $u, v \in V$. Říkáme, že u je předchůdce v a v je následník u. Platí tedy $E \subseteq V \times V$. [2, s. 34]

V kontextu této bakalářské práce se setkáme s orientovanými grafy při vyjádření konečného automatu pomocí stavového diagramu. Následně je jako graf reprezentován objekt, který vytváří uživatel pomocí webového rozhraní za účelem získání výsledku po jeho vyhodnocení.

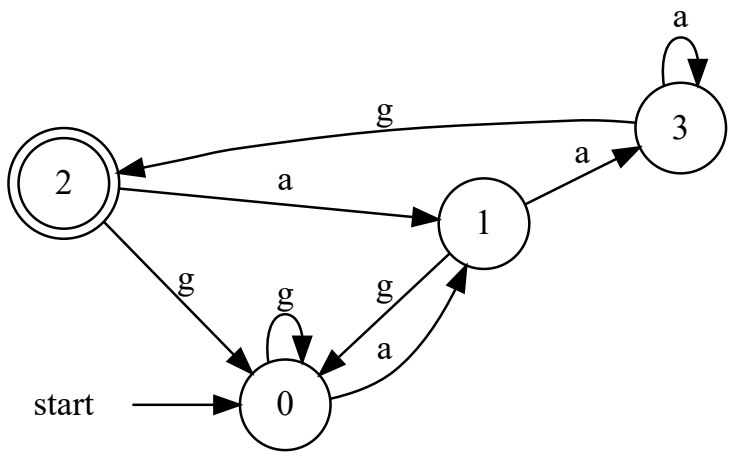
1.2 Konečný automat

▶ **Definice 1.2.** *Konečný automat definujeme jako uspořádanou pětici $M = (Q, \Sigma, \delta, q_0, F)$, kde*

- *Q je konečná neprázdná množina stavů,*
- *Σ je konečná vstupní abeceda,*
- *δ je přechodová funkce,*
- *$q_0 \in Q$ je počáteční stav,*
- *$F \subseteq Q$ je podmnožina koncových stavů.* [3, s. 22]

▶ **Definice 1.3** (Přechodová funkce DKA)**.** *Zobrazení δ z množiny $Q \times \Sigma$ do množiny stavů Q (tj. $\delta : Q \times \Sigma \to Q$) nazýváme přechodovou funkcí deterministického konečného automatu.* [3, s. 23]

▶ **Definice 1.4** (Přechodová funkce NKA)**.** *Zobrazení δ z množiny $Q \times \Sigma$ do množiny všech podmnožin (potenční množiny) množiny Q (tj. $\delta : Q \times \Sigma \to P(Q)$) nazýváme přechodovou funkcí nedeterministického konečného automatu.* [3, s. 23]



Obrázek 1.1 Ukázka stavového diagramu konečného automatu

▶ **Definice 1.5** (Přechodová funkce NKA s ε-přechody)**.** *Zobrazení δ z množiny $Q \times (\Sigma \cup \{\varepsilon\})$ do množiny všech podmnožin (potenční množiny) množiny Q (tj. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$) nazýváme přechodovou funkcí nedeterministického konečného automatu s ε-přechody.* [3, s. 23]

Konečný automat si můžeme představit jako výpočetní model, výpočet začíná v počátečním stavu. Automat následně čte jednotlivé symboly řetězce ze vstupní pásky a dle přečtených symbolů, společně s přechodovou funkcí, mění aktivní stav, dokud nepřečte všechny symboly. Konečný automat řetězec přijme, pokud z pásky již všechny symboly přečetl a zároveň skončí v některém koncovém stavu. [3, s. 22–24]

DKA má na rozdíl od NKA tedy jednoznačně dané, do jakého stavu bude dle přechodové funkce pokračovat. NKA s epsilon přechody navíc může přejít dle funkce do jiného stavu i bez přečtení symbolu.

Konečný automat lze znázornit více způsoby, které se liší v zápisu přechodové funkce. Jedná se o reprezentaci ve formě:

- formálního zápisu,

$$\begin{aligned} \delta: \quad & \delta(0,a) = 1 \\ & \delta(0,g) = 0 \\ & \delta(1,a) = 3 \\ & \delta(1,g) = 0 \\ & \delta(2,a) = 1 \\ & \delta(2,g) = 0 \\ & \delta(3,a) = 3 \\ & \delta(3,g) = 2 \end{aligned}$$

- stavového diagramu (viz obrázek 1.1),
- tabulky (viz ukázka formátu 2.1). [3, s. 24]

1.3 Gramatika

▶ **Definice 1.6.** *Gramatiku definujeme jako uspořádanou čtveřici* $G = (N, \Sigma, P, S)$ *kde*

- N *je konečná neprázdná množina neterminálních symbolů (zkráceně neterminálů).*
- Σ *je abeceda symbolů, které nazýváme terminálními symboly (zkráceně terminály). Jedná se o abecedu jazyka generovaného gramatikou. Platí, že* $N \cap \Sigma = \emptyset$
- P *je konečná množina přepisovacích pravidel ve tvaru:*

$$\alpha A \beta \to \gamma \quad (\alpha, \beta, \gamma \in (N \cup \Sigma)^*, A \in N)$$

- $S \in N$ *je počáteční neterminální symbol gramatiky.* [3, s. 10–11]

Gramatika slouží k popisu jazyka, jelikož z počátečního symbolu se aplikováním jednotlivých přepisovacích pravidel generují řetězce jazyka.

Obecná gramatika je pro většinu aplikací příliš komplikovaná a proto jsou zavedena zjednodušení.

Dle Chomského klasifikace gramatik můžeme na základě tvaru přepisovacích pravidel vymezit následující čtyři typy gramatik:

▶ **Definice 1.7.** *Regulární gramatiku s pravidly tvaru:*

$$A \to \alpha \text{ nebo } A \to aB \quad (\alpha \in \Sigma, A, B \in N)$$

nebo $S \to \varepsilon$*, pouze za předpokladu, že počáteční neterminál* S *se nevyskytuje na pravé straně žádného z pravidel*

▶ **Definice 1.8.** *Bezkontextovou gramatiku s pravidly tvaru:*

$$A \to \alpha \quad (\alpha \in (N \cup \Sigma)^*, A \in N)$$

▶ **Definice 1.9.** *Kontextovou gramatiku s pravidly tvaru:*

$$\gamma A \delta \to \gamma \alpha \delta \quad (\gamma, \delta \in (N \cup \Sigma)^*, A \in N, \alpha \in (N \cup \Sigma)^+)$$

nebo $S \to \varepsilon$*, pouze za předpokladu, že počáteční neterminál* S *se nevyskytuje na pravé straně žádného z pravidel*

▶ **Definice 1.10.** *Obecnou (neomezenou) gramatiku s pravidly tvaru:*

$$\alpha A \beta \to \gamma \quad (\alpha, \beta, \gamma \in (N \cup \Sigma)^*, A \in N)$$

[3, s. 11]

Jazyk generovaný regulární gramatikou nazýváme regulární jazyk a můžeme ho popsat také regulárním výrazem.

▶ **Definice 1.11.** *Regulární výraz definujeme rekurzivně nad abecedou* Σ *takto:*

1. $\emptyset, \varepsilon, a$ *jsou regulární výrazy pro všechna* $a \in \Sigma$*.*
2. *Jsou-li* x, y *regulární výrazy nad* Σ*, pak* $(x + y)$*,* $(x \cdot y)$ *a* $(x)^*$ *jsou regulární výrazy nad* Σ*.* [3, s. 63]

▶ **Příklad 1.12** (Převzato z [3, s. 90])**.** Regulární gramatika $G = (\{S, A\}, \{0, 1, 2\}, P, S)$, kde

$$P = \{S \to 0S \mid 1A \mid 1 \mid 0,$$
$$A \to 2A \mid 1A \mid 0\}$$

generuje regulární jazyk, který můžeme popsat pomocí regulárního výrazu

$$V = 0^*1(2 + 1)^*0 + 0^*(1 + 0).$$

1.4 Parsování

Parsování je proces rozpoznání struktury věty podle dané gramatiky. Definováno je takto abstraktním způsobem za cílem možnosti interpretace v nejrůznějších významech. Větou totiž můžeme rozumět lineární reprezentaci datové struktury, počítačový program nebo jakoukoliv lineární sekvenci, která má daným způsobem definováno možné pořadí symbolů. [4, s. 1]

O parsování tedy nemusíme mluvit pouze v případě syntaktické analýzy zdrojového kódu. V této práci bude parsování zpravidla zmiňováno v kontextu vytvoření vnitřní reprezentace datové struktury z jejího formátu.

Kapitola 2

Analýza původního stavu

Pro pokračování ve vývoji webového rozhraní je stěžejní prvotní analýza původního stavu. Tato kapitola se tedy věnuje seznámení s ALT knihovnou, popisu a analýze webového rozhraní a jeho zdrojového kódu ve stavu „as is“ a je vytvořena po důkladném manuálním testování webové uživatelského rozhraní. Rozebírány jsou použité technologie a jejich zajímavá specifika, struktura zdrojového kódu, podrobně jsou popsány komplexnější části jako například vyhodnocování algoritmového grafu a v neposlední řadě jsou zavedeny jednotné pojmy pro UI komponenty v současné verzi uživatelského rozhraní.

2.1 ALT

ALT je knihovna vyvíjená na FIT ČVUT, která umožňuje manipulaci s datovými strukturami z oblasti stringologie, arbologie a částečně i grafů. Podporuje různé datové struktury, jako například automaty, gramatiky, stromové struktury a algoritmy nad těmito strukturami. Využívána je jako učební nástroj pro oblast formálních jazyků. Dostupná je v podobě příkazové řádky, verze s grafickým rozhraním a webovým rozhraním. [5]

Používání ALT knihovny je možné pomocí příkazové řádky (aql2 program), grafického uživatelského rozhraní (agui2 program) a webového rozhraní. Webové rozhraní knihovny algoritmů ALT je dostupné na adrese `https://alt.fit.cvut.cz/webui/`. První funkční verze tohoto webového rozhraní vznikla v rámci bakalářské práce Michaela Vávry [1], která navazuje na projekt Statemaker. Statemaker je dynamický webový editor konečných automatů vytvořený v rámci bakalářské práce Petra Svobody [6]. V současné chvíli je webové rozhraní využíváno při výuce předmětu BI-AAG na FIT ČVUT.

2.1.1 Nativní textová reprezentace datových struktur používaná v ALT

V následujících podkapitolách bude často analyzováno zadávání vstupu a zobrazování výstupu. Je tedy nutné zmínit formát, který ALT používá pro textovou reprezentaci datových struktur. Formát je odlišný pro různé datové struktury a vychází z jejich existujících způsobů zápisu, v případě automatu ze zápisu přechodové funkce pomocí tabulky, u gramatiky vychází členění z definice. Pomocí tohoto formátu lze předat složitější datovou strukturu jako vstup pro ALT i webové rozhraní.

Pro jednodušší orientaci bude tento formát dále označován jako text-fit formát, jako tomu je ve zdrojových kódech a v původní bakalářské práci zaobírající se editorem automatů Statemaker [6].

Ukázka formátu 2.1 text-fit formát pro DKA, dle dokumentace `https://alt.fit.cvut.cz/docs/parse/`

```
DFA a g
>0 1 0
1 3 0
<2 1 0
3 3 2
```

Ukázka formátu 2.2 text-fit formát pro gramatiku, převzato z `https://alt.fit.cvut.cz/docs/parse/`

```
CFG (
{C, A, B, D},
{a, b},
{
        A -> a A b |,
        B -> a a A b b | B,
        C -> b B | D,
        D ->
},
A)
```

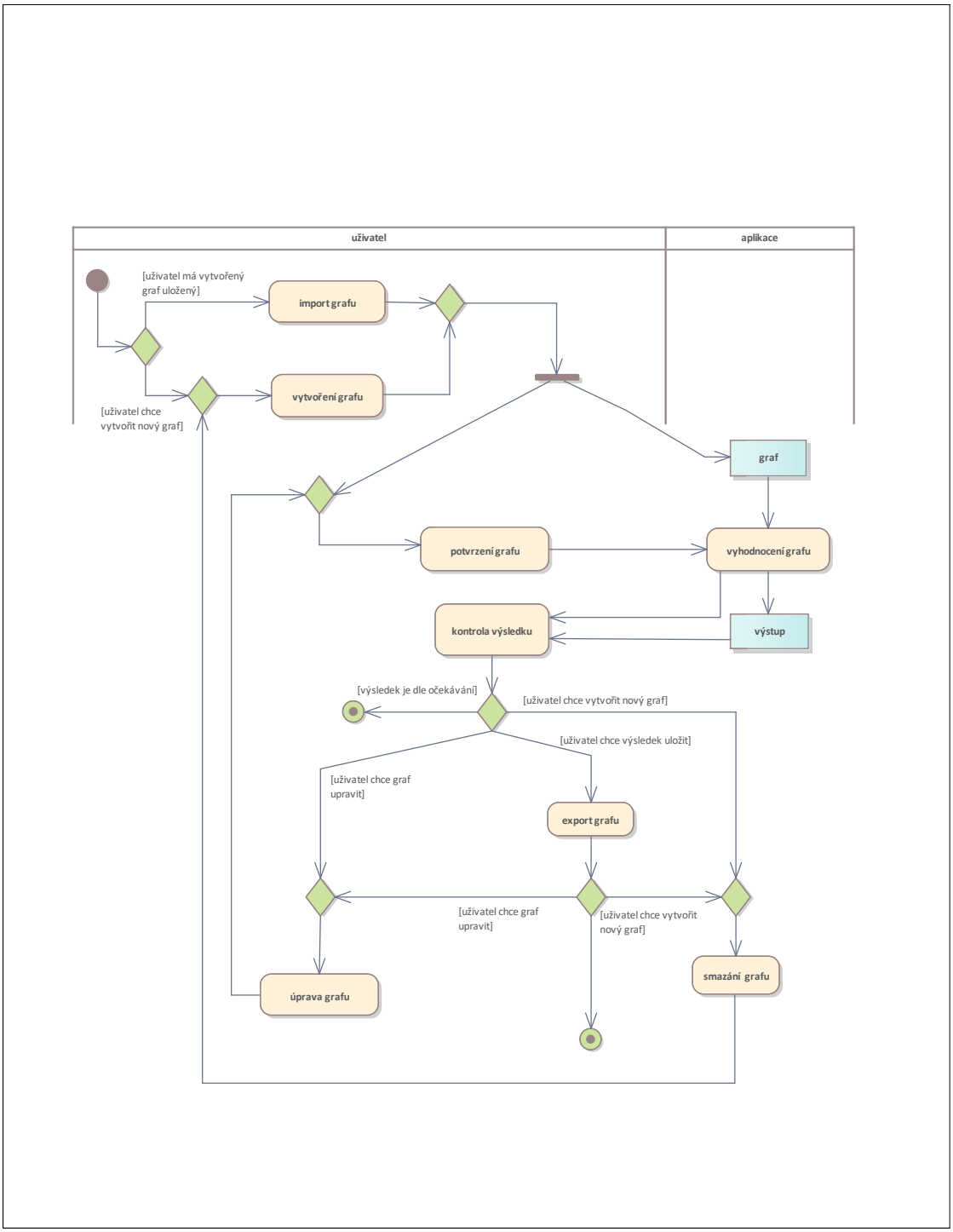
2.2 Současný stav

Pomocí diagramu aktivit 2.1 je přehledně zobrazen pracovní proces uživatele, jehož je webové rozhraní součástí. Diagram popisuje používání webového rozhraní ve stavu před provedením změn.

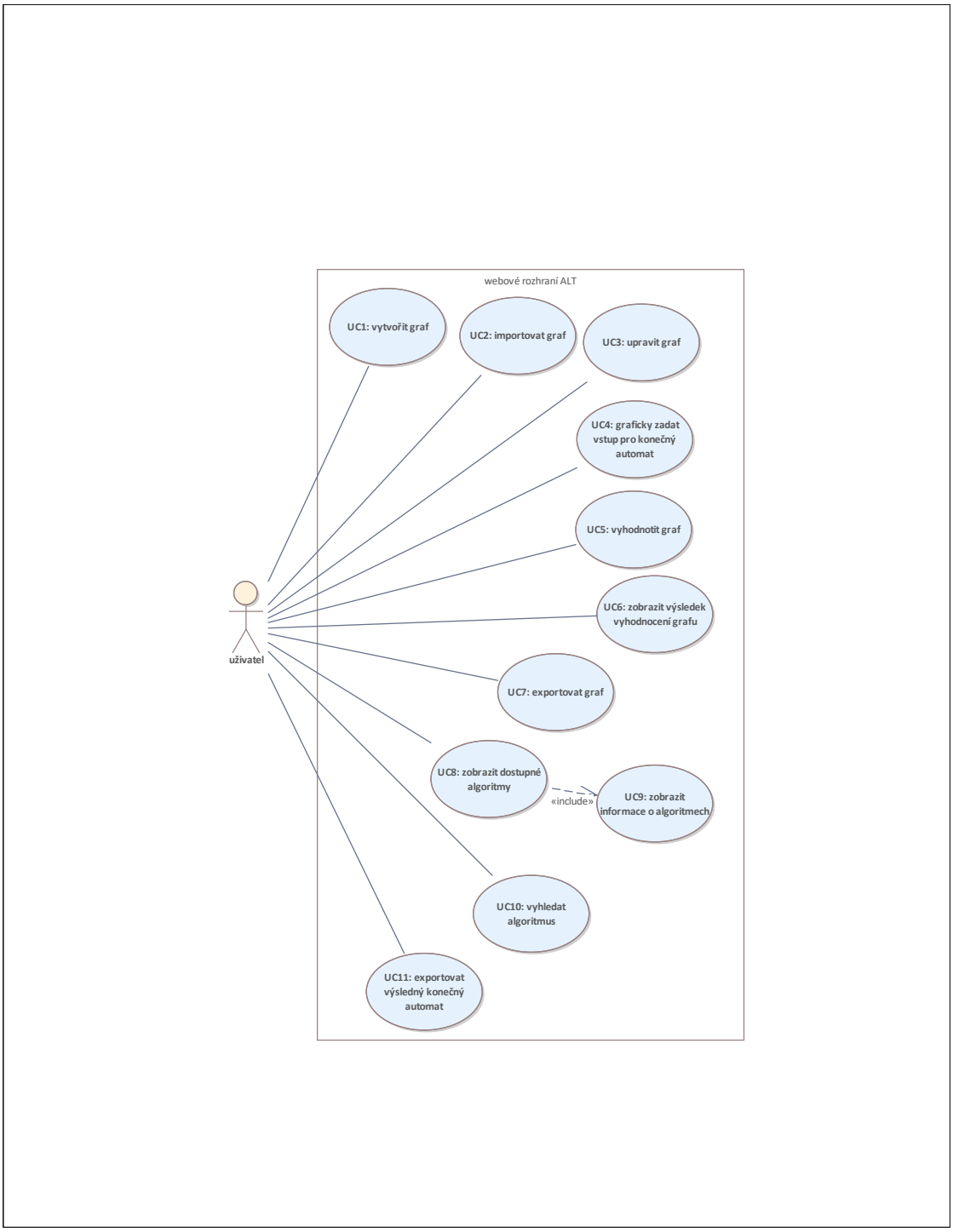
Jako běžného uživatele si lze představit studenta využívajícího ALT k ověření svého postupu při vypracování příkladu z oblasti formálních jazyků. Uživateli je umožněno vytvořit graf pomocí komponent webového rozhraní nebo importovat graf, který byl pomocí webového rozhraní dříve vytvořen a exportován do souboru. Následně je možné tento graf vyhodnotit a zkontrolovat výsledek. Dle získaného výsledku může uživatel pokračovat různě. Pokud vyhodnocení proběhlo v pořádku a nestojí o vytvoření dalšího grafu, již dále nepokračuje nebo graf exportuje do souboru, aby nad ním mohl vyhodnocení spouštět opakovaně. Další možnost nastává, když vyhodnocení grafu neproběhlo, jak by si uživatel přál (zobrazení chybové hlášky, chyba při zadání příkladu) nebo zkrátka má zájem o změnu grafu nebo o pokračování s jiným grafem. V tomto případě graf poupraví nebo smaže a vytvoří nový.

Diagram je nápomocný k analýze aktuálního stavu pro přehlednější identifikaci rizikových míst. Lze z něj jednoznačně vyčíst, že při tvorbě grafu nastává mnoho podmínek, vstup je možné opakovaně upravovat, tyto činnosti by tedy měly být uživatelsky co nejpřívětivější.

Jednotlivé případy užití, které současná verze webového rozhraní splňuje, jsou zobrazené v modelu případů užití 2.2. Modely případů užití se v praxi používají jako podklady k akceptačnímu testování, poslouží tak i při závěrečném testování, během kterého bude potřeba pokrýt typické situace, na které může uživatel při používání narazit. Jednotlivé případy užití jsou dále popsány pro jasné vymezení funkcionalit původní verze. Vzhledem k faktu, že se jedná o analýzu aktuálního stavu, nebyl zvolen popis pomocí podrobných scénářů, ale popis splněných požadavků pro detailnější vytyčení vlastností a implementovaných požadavků. Části, jichž se týká vylepšení, jsou následně podrobně analyzovány ve vlastní kapitole.



Obrázek 2.1 Diagram aktivit



Obrázek 2.2 Model případů užití

UC1: Vytvořit graf

K vytvoření grafu uživatel vybírá vstupní vrcholy, výstupní vrcholy a algoritmy z levého panelu na plátno. Pomocí napojovacích bodů je spojuje hranami. Do vstupních vrcholů lze zadat textový vstup, číslo - celé nebo s desetinnou čárkou nebo vybrat pravdivostní hodnotu.

V druhém typu vstupního vrcholu je možné zadat graficky konečný automat pomocí Statemakeru.

UC2: Importovat graf

Lze importovat soubor ve formátu JSON s reprezentací vytvořeného grafu.

JSON je oblíbený formát pro výměnu dat, i přes název (JavaScript Object Notation) se jedná o notaci nezávislou na programovacím jazyku, z JavaScriptu pouze vychází syntaxe pro zápis objektů. [7, s. 1-6]

UC3: Upravit graf

V průběhu práce s grafem lze editovat vstupní hodnoty a přidávat či ubírat vrcholy a hrany. Vstupní hodnoty se po stisknutí tlačítka znovu zobrazí v dialogu s editorem vstupu.

UC4: Graficky zadat vstup pro konečný automat

Konečný automat lze zadat pomocí kreslení stavového diagramu automatu v nástroji Statemaker. Zde uživatel vybírá na plátně v dialogovém okně jednotlivé stavy, které spojuje hranami.

UC5: Vyhodnotit graf

Graf lze vyhodnotit pomocí tlačítka.

UC6: Zobrazit výsledek vyhodnocení grafu

Výsledek uživatel dostává ve třech možných formách v závislosti na výběru výstupního vrcholu. Výsledek lze zobrazit pomocí tlačítka na zeleně označeném výstupním vrcholu po vyhodnocení grafu. V případě obecného výstupu je po stisknutí tlačítka zobrazen dialog, kde je výsledek vypisován jako textový řetězec. Textový řetězec pro složitější datové struktury zobrazuje textový formát daného objektu, číslo jako text a pravdivostní hodnoty jako 0 a 1.

Pro konečné automaty lze zvolit speciální výstupní vrchol, kde se po stisknutí tlačítka zobrazí dialog a výsledek ve Statemakeru. Třetím výstupním vrcholem je „Dot výstup“ s vykreslením automatu v DOT formátu. DOT je jazyk, který popisuje graf pomocí tří hlavních elementů: grafu, vrcholu a hrany. [8]

UC7: Exportovat graf

Vytvořený graf lze exportovat a stáhnout jako JSON soubor.

UC8: Zobrazit dostupné algoritmy

Algoritmy jsou zobrazeny v levém panelu členěné do kategorií dle částí plně kvalifikovaného jména. Pro každý algoritmus jsou vyjmenována a nabízena k výběru na plátno všechna přetížení algoritmu.

UC9: Zobrazit informace o algoritmech

Jednotlivá přetížení v seznamu algoritmů obsahují informaci o vstupních parametrech, návratové hodnotě a u většiny obsahují i krátký dokumentační popisek.

UC10: Vyhledat algoritmus

Algoritmus je možné vyhledávat pouze dle jména, či části jména. Díky tomu, že kategorie představují jednotlivé části/jmenné prostory plně kvalifikovaného jména algoritmu, lze takto filtrovat i kategorie. Nelze naopak filtrovat podle dalších parametrů algoritmu.

UC11: Exportovat výsledný konečný automat do souboru

Výsledný konečný automat lze exportovat a stáhnout jako soubor ve formátu DOT a SVG. SVG je jazyk určený pro vektorovu grafiku, jedná se o značkovací jazyk s hierarchickou strukturou vycházející z formátu XML. [9]

2.3 Komponenty uživatelského rozhraní

Webové rozhraní se skládá z jedné obrazovky, na které uživatel vytváří graf a následně ho vyhodnocuje. Tvorba grafu je ovládána vybráním vrcholu a napozicováním na plátně. Veškeré vstupy s výstupy jsou zobrazovány uživateli pomocí dialogových oken. Čtenář se v této podkapitole seznámí s jednotlivými částmi webového rozhraní a s terminologií, která se bude vyskytovat v následujících kapitolách.

2.3.1 Plátno

Plátno pokrývá většinu obrazovky webového rozhraní, jedná se o místo určené k tvorbě grafu, kam uživatel přenáší jednotlivé vrcholy a spojuje je pomocí hran reprezentovaných šipkami.

Po pravé straně se nachází panel nástrojů pro manipulaci s grafem, nápověda pro klávesové zkratky a tlačítko pro vyhodnocení. Pro plátno lze také volitelně zapnout mřížku pro jednodušší pozicování vrcholů grafu, plátno lze přiblížit, oddálit a zvolit mód pro výběr, či pro pohyb po plátně.

Z hlediska implementace je plátno reprezentováno SVG komponentou.

2.3.2 Seznam algoritmů

Po levé straně obrazovky se nachází panel, z kterého uživatel vybírá vstupy. V horní části uživatel vybírá ze vstupů s označením `Input` (vstup pro textový formát, číslo či pravdivostní hodnotu) a `Automaton Input` (vstup v pomocí Statemakeru). `Input` po přenesení na plátno otevírá dialogové okno se `select menu` datových typů pro vstup a textovým polem, všechny složitější datové struktury je tedy potřeba zadávat ve formě formátu text-fit a pak parsovat algoritmem `string::Parse`. `Automaton Input` otevírá dialog s nástrojem Statemaker, pomocí kerého lze interaktivně vytvářet konečný automat pomocí grafických nástrojů.

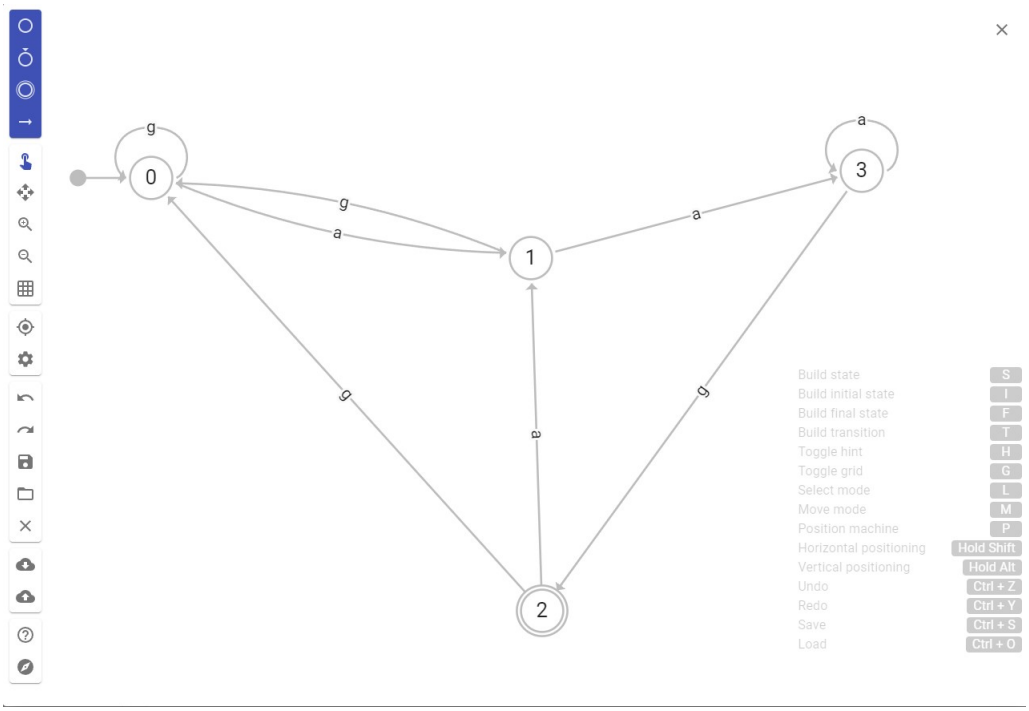
Pod vstupy pokračuje nabídka výstupů `Output`, `Automaton Output` a `Dot Output`. Output slouží pro všechny výstupy, které vypisuje jako text, `Automaton Output` vykresluje konečný automat ve Statemakeru a `Dot Output` zobrazuje konečný automat pomocí knihovny `d3-graphviz`, výsledek lze uložit v DOT a SVG formátu pomocí tlačítek ve spodní části dialogu.

V poslední části panelu se nachází seznam algoritmů s vyhledáváním. Vyhledávat lze pouze podle jména algoritmu. Algoritmy jsou roztříděné pomocí jmenných prostorů ALTu a má podobu vnořených seznamů, které lze pomocí šipek postupně rozbalovat. Při rozbalení celého názvu algoritmu je každé přetížení algoritmu zobrazeno jako box se vstupními parametry, návratovou hodnotou a dokumentačním popiskem.

Pro seznam algoritmů je rekurzivně použitá komponenta obalující `List` a jednotlivé `ListItem`.

2.3.3 Graf

V kontextu této práce bude pojem graf zmiňován v souvislosti s grafickým výsledkem zadání vstupu uživatele. Jedná se o objekt, který uživatel vytvoří na plátně pomocí vstupních vrcholů, algoritmových vrcholů, výstupních vrcholů a hran.



Obrázek 2.3 Snímek obrazovky Statemakeru

2.3.4 Vrchol grafu

Vrcholy grafu vznikají po výběru vstupů, výstupů či algoritmů z levého panelu a přenesení na plátno. Při pozicování vrcholu boxu je naznačen šedou barvou a po umístění na plátno jej lze napojovat pomocí hran s ostatními vrcholy. Pro napojení mají vrcholy napojovací body, které uživateli napovídají počet vstupních či výstupních parametrů. Vrcholy můžeme kategorizovat do třech typů. Vstupní vrcholy, do kterých uživatel zadá vstup. Algoritmové vrcholy představující algoritmus. A výstupní vrcholy sloužící pro zobrazení výstupu.

Vrcholy jsou reprezentované jako boxy a pro jejich vykreslení byla použita SVG komponenta `rect`.

V rámci této práce budou jako vrcholy označovány objekty i se svými daty, pro označení grafické reprezentace bude použit termín box ať už vstupní, výstupní či algoritmový.

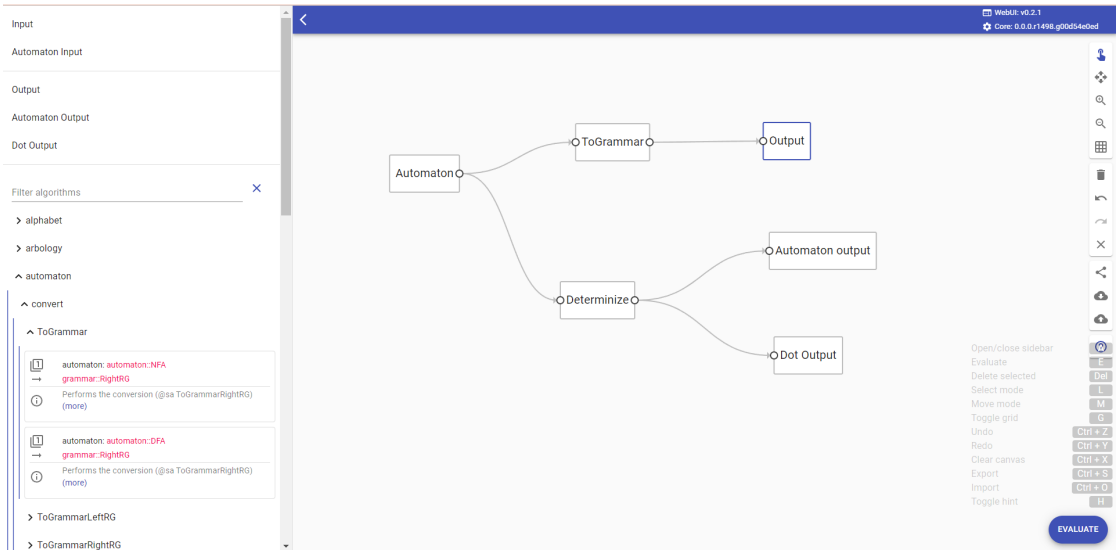
2.4 Technologie

2.4.1 GitLab

GitLab je webový nástroj sloužící k týmové spolupráci a zjednodušuje tak proces konfiguračního řízení díky poskytovanému Git repozitáři. GitLab nabízí i další řešení pro ostatní činnosti softwarového procesu.

Tento nástroj je využíván pro správu zdrojových kódů webového rozhraní, které se zde nachází v podobě monorepozitáře.

Monorepozitář je repozitář obsahující více projektů, zjednodušuje tak složitost projektu, umožňuje znovupoužití částí kódu, snadnou spolupráci týmu či efektivnější řešení závislostí. [10] V případě webové aplikace se jedná o projekty webui (prezentační vrstva), worker a server, které spolu úzce souvisí. Pro tento projekt je navíc GitLab využíván pro správu úkolů a chyb,



Obrázek 2.4 Snímek obrazovky uživatelského rozhraní

lze tak jednoduše sledovat řešené problémy přímo v jednotlivých verzích zdrojového kódu.

V rámci projektu je využívána průběžná integrace a průběžné nasazování. Větev `master` je nasazena na adrese `https://alt.fit.cvut.cz/webui-staging/`, kde je možné otestovat nové změny před konečným nasazením, které je prováděno při vytvoření tagu verze pomocí gitu.

2.4.2 Npm

Npm je balíčkovací systém pro prostředí Node.js vytvořené v roce 2009 jako open source projekt pro sdílení modulů zdrojových kódů. [11]

Npm je tvořeno třemi hlavními částmi. Webovými stránkami, na kterých mohou vývojáři procházet existující balíčky, přístupem pomocí příkazové řádky a registrem, který slouží jako databáze všech balíčků. [12]

Pomocí npm jsou ve webui spravovány závislosti. V souboru `package.json` jsou definovány závislosti, podle tohoto souboru npm stáhne a nainstaluje příslušné knihovny a technologie v požadovaných verzích.

2.4.3 TypeScript

Typescript je programovací jazyk vyvinutý společností Microsoft. Jedná se o nadstavbu programovacího jazyka JavaScript se statickým typováním. Výhodou je kompatibilita s již existujícím kódem v JavaScriptu, kód je totiž následně kompilovaný právě do JavaScriptu. TypeScript nenabízí navíc pouze možnost statického typování, přináší také principy typické pro objektově orientované programování – třídy, rozhraní a moduly. Umožňuje tak lepší strukturování kódu a kontrolu typů. [13, s. 1–29]

2.4.4 React

React je knihovna pro programovací jazyk JavaScript využívaná k vývoji uživatelských rozhraní. Používá deklarativní způsob programování. Rozhraní je vyvíjeno pomocí jednotlivých komponent, které zapouzdřují vlastní tzv. `state` neboli stav. Na základě změn stavu se následně komponenta dokáže vykreslit. [14]

Další typickou součástí komponenty jsou `props` neboli vlastnosti. Jedná se o JavaScriptové objekty, které ovlivňují podobu vykreslené komponenty. `Props` jsou informace, které jsou komponentě předány z vnějšku, zatímco `state` je vnitřní stav. [15]

React má několik typů komponent, například třídu `React.Component`. Kód připomíná XML formát, stejně jako XML totiž používá tagy pro deklarování podoby komponenty. Pokud se data v komponentě změní, React je sám efektivně aktualizuje a znovu vykreslí podobu komponenty.

Syntaxi je možné používat na základě tříd – pak komponenta dědí od `React.Component` třídy nebo je využívána JSX syntaxe, která psaní komponenty zjednodušuje po syntaktické stránce. JSX využívá síly JavaScriptu, stačí vložit javascriptový výraz do složených závorek. Každý React element je javascriptový objekt, který je možné uchovávat v proměnné a předávat v programu. Pomocí React komponent můžeme tvořit složitá uživatelská rozhraní z komponent menších a zároveň znovupoužitelných. [16]

Vytvářet komponenty lze tedy s třídními nebo funkcionálními komponentami. Webové rozhraní knihovny ALT využívá implementaci pomocí funkcionálních komponent, je tedy vhodné se zaměřit na paradigma pro state management neboli správu stavu, a to na hooks. Hooks slouží k zapouzdření stavu, využívají pouze schopností jazyka JavaScript. Využívají deklarativní způsob programování a je možné je volat pouze ve funkčních komponentách. Hlavním cílem hooks je oddělení logiky pro správu stavu od logiky vykreslení komponent. Ukázka useState hooku pro změnu stavu:

useState
Vrací hodnotu stavu a funkci pro aktualizaci hodnoty stavu. [17]

Ukázka formátu 2.3 Ukázka useState hook

```
import { useState } form 'react'

const [ state, setState ] = useState(initialState)
```

V případě knihovny React je také vhodné zmínit, jakým způsobem je předána prohlížeči informace o vykreslení komponent. DOM neboli Document Object Model, česky objektový model dokumentu, je rozhraní pro HTML a XML dokumenty. Manipulace s DOM modelem je jádro moderních interaktivních webových stránek s dynamicky měnícím se obsahem. Model je reprezentován stromovou strukturou a její jednotlivé vrcholy umožňují přístup k příslušným elementům. React využívá k rychlejšímu výkonu virtuální DOM. Jedná se o programovací koncept, při kterém je ideální reprezentace UI udržována v paměti a je synchronizována s reálnou reprezentací DOM modelu. Všechny změny, které mají nastat totiž React nejdříve vytvoří pomocí virtuálního DOM modelu, jedná se o jakousi simulaci. Je totiž kopií reálného HTML DOM modelu, v případě změny React aplikuje změny ve virtuálním DOM modelu a pak ve struktuře HTML DOM modelu mění jen to, co je nutné. Díky stromové struktuře lze tyto změny provádět velmi efektivně a není tak třeba celý DOM model vytvářet znovu.[18, s. 6–8]

2.4.5 MaterialUI

Material UI je React framework nabízející řadu komponent pro tvorbu uživatelského rozhraní [19]. Material UI je open-sourcová implementace designového systému Material design [20], navrženým společností Google, design komponent je inspirován reálným světem a jeho texturami, především tím, jak reálné objekty odráží světlo a vrhají stín [21].

2.4.6 SVG

SVG je standard založený na XML formátu, který definuje grafické objekty. Definovány jsou tři základní typy objektů – vektorové tvary, rastrové obrazy a textové objekty.

Objekty jsou vykreslovány pomocí vektorové grafiky, která umožňuje kvalitní vykreslení, další výhodou tohoto formátu je také škálovatelnost. Objekty lze bez ztráty kvality libovolně zmenšovat a zvětšovat. SVG podporuje CSS styly a dokonce lze na libovolný grafický element aplikovat velmi flexibilní bitmapové efekty, díky kterým lze například jednoduše napsat pro grafický objekt filtr. [22]

2.4.7 Redux

Redux je open-source JavaScriptová knihovna, která slouží ke správě stavu aplikace. Zjednodušuje sdílení dat a komunikaci mezi komponentami. Většinou je využívána v aplikacích používajících knihovnu React, ale lze jej využít i s dalšími knihovnami a frameworky (nebo-li aplikačními rámci) založenými na jazyce JavaScript jako například Angular či Vue.js.

Většina těchto knihoven má vlastní způsoby řešení správy stavu, nicméně pro komplexní aplikace je Redux ideální pro přehlednější správu složitého stavu. Při používání Reduxu je stav vždy předvídatelný, pro stejný vstup pro reducer vrací vždy stejný výstup, nevytváří žádné vedlejší efekty. Stav je v uchováván v tzv. store. Ke store lze v rámci celé aplikace přistupovat a to pomocí actions a reducers. Actions jsou události, která data odesílají směrem do store. Jsou volané pomocí `dispatch` funkce. Reducer je funkce bez vedlejších efektů, která přijímá aktuální stav a akci a vrací stav nový. Může jich být v aplikaci více a každý se může spravovat příslušný stav části aplikace. [23]

2.5 Struktura projektu

V této kapitole následuje analýza jednotlivých částí navazují na přechozí informace. Analyzovány jsou vzájemné závislosti částí projektu a je modelována jejich komunikace.

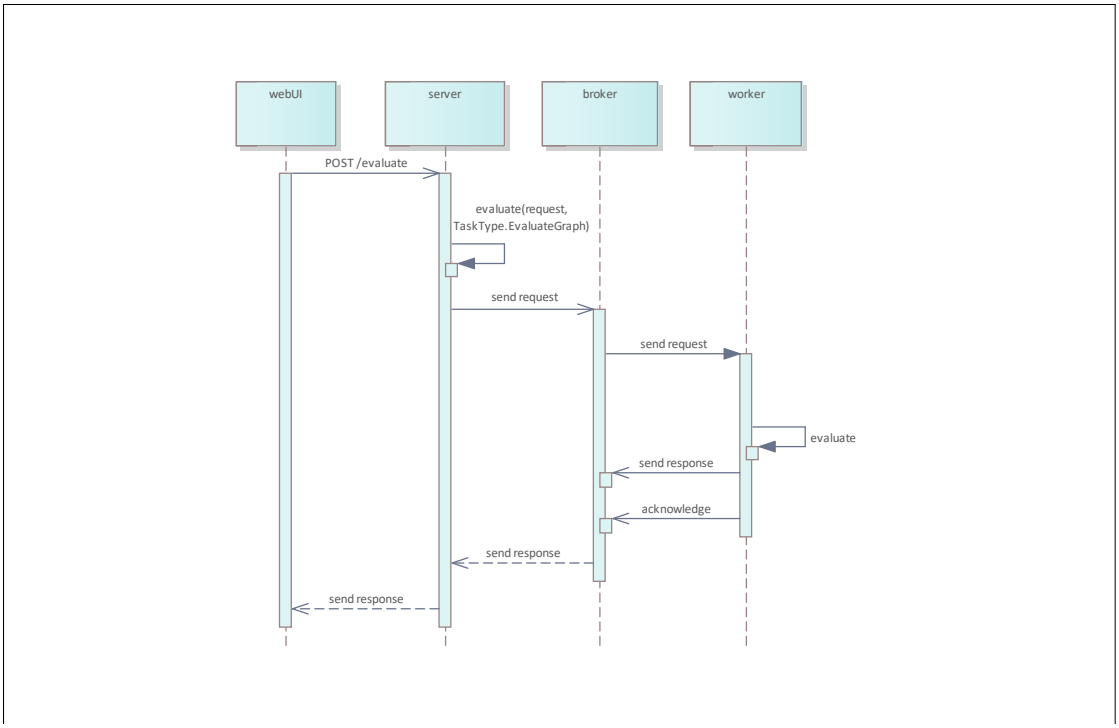
2.5.1 WebUI

Webui je část projektu představující prezentační vrstvu. Zdrojové kódy jsou psány v jazyce TypeScript, komponenty jsou vytvářeny pomocí knihovny MaterialUI, React a SVG a především globální stav je spravován pomocí knihovny Redux. Využíván je funkcionální zápis komponent, který umožňuje využití hooks. Pro správu stavu komponent jsou převážně využity právě hooks. Redux má využívá několik reducerů, které se starají o jednotlivé části stavu rozhraní.

Součástí webového rozhraní je Statemaker, který vykresluje stavy a hrany pomocí SVG komponent, stejně jako webové rozhraní vrcholy a jejich hrany. Tvorba grafu vytváří ze stylu pipe and filter stejně jako agui2.

Pipe and Filter je architektonický styl využívající dva hlavní elementy komponenty (filtr) a konektory mezi nimi (pipe, česky roura). Základním principem je propojení na sobě nezávislých filtrů pomocí pipe, která výstup jednoho filtru pošle do vstupu dalšího. Místa vstupu se nazývají porty. Filtry konzumují data na vstupu, transofmují je a výsledek posílají do jednoho či více portů. Výhodou je, že filtry lze přepoužívat a nedochází tedy k duplikaci kódu, také je jednouché přidání dalších, navzájem jsou na sobě nezávislé. Styl je využíván k transformaci dat, používá se například při implementaci kompilátoru nebo v UNIX shellu. [24]

Při prvním vykreslení Web UI je pomocí `useFetch` hooku získán seznam dostupných algoritmů, ty jsou dále kategorizovány dle plně kvalifikovaného jména a zobrazena (viz diagram 4.1). Vyhodnocení grafu má na starosti třída `Algorithm Service`, která posílá požadavek na koncový bod serveru `/evaluate` (viz diagram 2.5).



Obrázek 2.5 Sekvenční diagram vyhodnocení grafu

2.5.2 Server

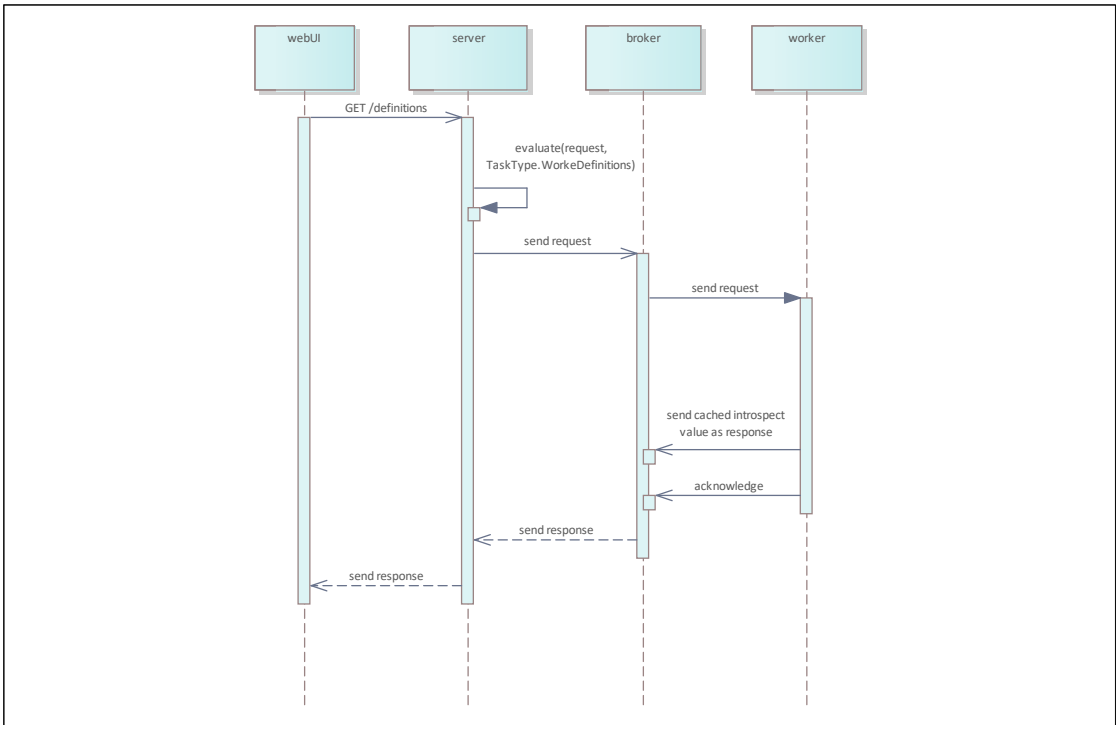
Server je podprojekt psaný v programovacím jazyce Kotlin s využitím frameworku pro vytváření asynchronních klient–server aplikací Ktor [25]. Server spouští vestavěného Apache ActiveMQ message brokera. Message broker je software, který umožňuje komunikaci mezi různými aplikacemi, v případě tohoto projektu zajišťuje posílání dat mezi webovým rozhraním a knihovnou ALT.

Server vystavuje dva koncové body. `/evaluate` pro HTTP metodu POST, kde přijímá graf z webového rozhraní v těle ve formátu JSON a `/definitions` pro HTTP metodu GET, pomocí které žádá webové rozhraní o seznam dostupných algoritmů. Kromě hlavní funkce obsahuje pouze třídu `EvaluateHandler`, která obstarává spuštění message brokera. Požadavky dále řadí do fronty a vytváří také frontu pro odpovědi.

2.5.3 Worker

Worker především zpracovává požadavky, které přichází z webového rozhraní přes server. Je psaný v jazyce C++, stejně jako knihovna ALT. Vyhodnocuje dva různé úkoly a to zjistit seznam algoritmů společně s kompatibilním typováním a verzí ALTu a vyhodnotit graf. Tyto dva úkoly jsou rozlišeny pomocí typu úkolu, který přichází již od brokera ze strany serveru.

Graf je vyhodnocen metodou `evaluate`, která nejdříve řadí vrcholy pomocí algoritmu `topsort`. Vrcholy jsou reprezentovány třídou `AbstractNode`, která je společným předkem pro jednotlivé třídy představující dané vrcholy. Vrcholy jsou postupně vyhodnocovány ALT knihovnou a dle hran jsou výsledky předány potomkovi v grafu, který ho využije ke svému vyhodnocení.



Obrázek 2.6 Sekvenční diagram získání definic algoritmů

Kapitola 3
Analýza

Tato kapitola je věnována analýze všech nedostatků, které byly již reportovány uživateli během používání webového rozhraní a těm, které byly nalezeny v průběhu počátečního testování. Jednotlivé problémy jsou členěny do samostatných podkapitol, každá podkapitola pojednává o jednom problému tzv. issue, které bylo v rámci praktické části bakalářské práce spravováno pomocí systému pro verzování a sledování chyb GitLab. Stejné členění je dodrženo i v následujících kapitolách zabývajících se návrhem a implementací.

3.1 Nápověda u vstupních parametrů

Nápověda u vstupních parametrů některých algoritmů zobrazuje nápovědu pro návratové hodnoty.

V seznamu algoritmů se u každého algoritmu nachází informace o vstupních parametrech a návratové hodnotě daného algoritmu. Každý parametr je z důvodu přehlednosti označen ikonou s číslem parametru nebo znakem výstupního parametru. Tyto ikony zobrazují po najetí myší nápovědu pomocí MaterialUI komponenty `tooltip`. U vstupních parametrů je zobrazeno číslo parametru, u návratové hodnoty nápis „return value“. U části algoritmů je ale pro vstupní parametry zobrazována nápověda určená pro návratovou hodnotu.

Ve zdrojovém kódu se zobrazení `tooltipu` pro vstupní parametry nebo návratovou hodnotu rozhoduje na základě podmínky, která ověřuje, že je daný parametr pojmenovaný. Nepojmenovaný parametr je vizualizován ve webovém rozhraní pouze datovým typem, zatímco pojmenované mají ještě před datovým typem jméno. Pojmenování parametru ale s nápovědou nijak nesouvisí.

3.2 Parametr šablonovaného algoritmu

Šablonované algoritmy nezobrazují v boxu reprezentujícím algoritmový vrchol na plátně vybraný parametr.

Seznam algoritmů nabízí šablonované algoritmy, které nabízí více variant podle toho, pro jaké datové typy slouží. Příkladem takového algoritmu je algoritmus `string::Parse`, který může mít za parametr například datové typy `automaton::Automaton` představující konečný či jiný automat nebo `grammar::Grammar` reprezentující gramatiku. V současné podobě webu si uživatel volí, který parametr použije v seznamu algoritmů výběrem přetížení. Nicméně ve chvíli, kdy je algoritmus vybrán a přenesen na plátno, ztrácí se informace o tom, která verze algoritmu byla zvolena. V rámci plátna ani algoritmového boxu se parametr nikde nezobrazuje, a tak pokud uživatel použije verzi pro automat, například pro gramatiku, je upozorněn chybovou hláškou. Na plátně chybí vizualizace parametru, který se aktuálně používá.

3.3 Dokumentace algoritmů na plátně

Po výběru algoritmu ze seznamu a vytvoření algoritmového vrcholu na plátně uživatel ztrácí podrobné informace o algoritmu. Chybí celý název algoritmu, vstupní parametry, návratová hodnota a u některých algoritmů dokumentace. Pokud potřebuje uživatel informace najít, musí je vyhledat v seznamu algoritmů, který obsahuje v mnoha případech několik přetížení jednoho algoritmu.

3.4 Vizualizace podrobného výstupu

Webové rozhraní aktuálně dokáže zobrazit podrobný výstup u algoritmů, které ho podporují. Podrobný výstup informuje uživatele o dílčích krocích, které provedl algoritmus. Například pro algoritmus `minimize`, který minimalizuje DKA, to jsou tabulky přechodové funkce automatu v jednotlivých iteracích. Zobrazení informace o tom, že vrchol obsahuje podrobný výstup je ale silně matoucí, jelikož je označen zeleným ohraničením, které je totožné se signalizací obyčejného výstupního vrcholu. Zároveň pro výpis je používáno proporcionální písmo, které nedokáže tabulky ze znaků správně zobrazit.

3.5 Inovace vstupů a výstupů pro komplexní datové typy

3.5.1 Vstup

Nabízené formáty vstupu, kromě Statemakeru pro grafické znázornění automatu, jsou textový řetěz (string), číslo (number) a pravdivostní hodnota (bool). Tyto možnosti jsou velmi nedostatečné, jelikož využití knihovny ALT tkví především v operacích nad složitějšími datovými strukturami. Pro vstup takové datové struktury je uživatel povinen použít zadání vstupu ve formě textového řetězce. Text musí být ve formátu text-fit 2.1. Zadání obsáhlé tabulky přechodů je pro uživatele časově náročným procesem, během kterého je jednoduché udělat mnoho chyb. Pro získání výsledné datové struktury (např. konečného automatu), kterou očekává na vstupu algoritmus spojený hranou, je navíc nutné ještě použít algoritmus `string::Parse`, který text v tomto formátu naparsuje na danou datovou strukturu. Uživatele bez větší znalosti o ALT knihovně tak nemusí ani napadnout, že je třeba textový vstup dále zpracovávat.

3.5.2 Formulář pro gramatiku

Z výše uvedených důvodů je vhodné nezůstat pouze u zadání vstupu pomocí text-fit formátu. Zachování formátu má jednoznačně svůj smysl díky kompatibilitě s celou ALT knihovnou. Nicméně pro běžného uživatele může být přepis složitého datového typu do striktního textového formátu nepohodlné a příliš zdlouhavé. Tento problém se týká i zadání vstupu gramatiky, pro kterou bude vytvořen formulář. Vhodné je také co nejvíce minimalizovat počet údajů, které musí uživatel vyplnit.

3.5.3 Tabulkový vstup pro automat

Další datová struktura, pro kterou je potřeba vytvořit formulář, je konečný automat. Zadání vstupu je aktuálně možné ve formě stavového diagramu ve Statemakeru. Formou tabulky jej lze zadat pouze v textovém formátu. Navíc reprezentace v podobě tabulky je velmi častá (v zadaných příkladech v BI-AAG tato forma výrazně převažuje) a přepis do text-fit formátu je zdlouhavý a nabízí nespecifické chybové hlášky v případě nedodržení formátu.

3.5.4 Výstup

Veškeré výstupy, kromě výstupu automatu při použití výstupu do Statemakeru, uživatel dostane pouze v textovém formátu. Pokud je výstupem konečný automat a uživatel sám nevybere použití algoritmu `string::Compose`, který automat převede do text-fit formátu, pak se zobrazí pouze nenaformátovaný textový výpis objektu. Tento výpis je velmi nesrozumitelný, je tedy žádoucí všechny komplexní datové typy formátovat jako text-fit. Velmi příhodné je toto použít i pro datový typ bool, který se aktuálně vypisuje pouze jako hodnoty 0 a 1 místo `false` a `true`.

3.5.5 Výstup gramatiky

Po implementaci výstupu z předchozí podkapitoly bude výsledná gramatika zobrazována ve formátu text-fit. Jedná se rozhodně o zlepšení, nicméně textový výstup je především pro gramatiky s pravidly s textově obsáhlejší pravou stranou velmi nepřehledný. Cílem je tedy vizualizovat výsledek tak, aby byly přehledně zobrazeny všechny součásti gramatiky a především přepisovací pravidla.

3.5.6 Tabulkový výstup pro automat

Přechodová funkce konečného automatu může být reprezentována pomocí tabulky. Tento formát je nejpoužívanějším formátem v předmětu BI-AAG. Aktuální verze webového rozhraní nabízí výpis v textovém formátu, DOT a SVG export a grafický výstup ve Statemakeru. Chybí zde možnost přehledné vizualizace přechodové funkce pomocí formátované tabulky, na kterou jsou studenti zvyklí. Aktuálně lze zobrazit automat ve formě tabulky pouze při použití algoritmu `string::Compose`, stále se ale jedná pouze o textový formát.

3.6 Vyhledávání v algoritmech

V seznamu algoritmů lze aktuálně vyhledávat pomocí textového pole, které během uživatelova psaní zároveň souběžně provádí vyhledávání. Hlavním nedostatkem tohoto řešení je, že vyhledávání se uskutečňuje pouze na úrovni kategorie. Porovnává se pouze, zda nějaká kategorie, tedy část názvu algoritmu, obsahuje vyhledávaný podřetězec. V jednotlivých přetíženích, která obsahují informace o parametrech, návratových hodnotách a dokumentaci, nevyhledává. Nelze tedy vyfiltrovat pouze algoritmy, které mají jako vstupní parametr třeba DKA. Hlavními požadavky je, aby bylo možné filtrovat i ve vstupních parametrech, návratové hodnotě a dokumentačním popisku.

Z analýzy zdrojového kódu bylo zjištěno, že aktuální implementace využívá rekurzivní algoritmus, vyhledávání totiž probíhá nad vnořenou datovou strukturou kategorií. Kategorie je zobrazena, pouze pokud obsahuje hledaný řetězec nebo ho obsahuje jeho podkategorie. Zdrojový kód filtrování se nachází ve stejném souboru jako komponenta seznamu kategorií, z tohoto důvodu bude potřeba řešit jak vyčlenit i stávající filtrování do samostatné funkce, která bude navíc i lépe testovatelná.

3.7 Seznam přetížení algoritmů

Jak již bylo zmíněno, seznam algoritmů je tříděn podle kategorií, kde jsou nakonec vypsány všechna existující přetížení algoritmu dostupná v ALT knihovně. U některých algoritmů se jedná o velmi dlouhý seznam, například u algoritmu `string::Compose`, který z mnoha datových struktur vytváří textovou reprezentaci. V důsledku toho musí uživatel při výběru algoritmu k přenesení na plátno hledat příslušné přetížení.

Pokud uživatel vybere jiné přetížení a napojí nesprávně hranu z jiného algoritmu s nekompatibilním typem, zobrazí se mu chybová hláška. Pokud i přesto uživatel graf vyhodnotí, tak i navzdory chybovému hlášení proběhne vyhodnocení úspěšně. Vyhodnocení používá pouze algoritmus a přetížení použije knihovna ALT dle vstupů. Seznam i chybové hlášky jsou pro uživatele velmi matoucí.

3.8 Filtrování algoritmů

Webové rozhraní před zobrazením algoritmů získává seznam od workera. Aktuálně jsou zobrazované všechny existující algoritmy, a to i ty, které ve verzi webového rozhraní nejsou potřebné. Aktuálně se jedná o algoritmy z jmenného prostoru `example`.

3.9 Vlastní funkce

ALT v aql2 verzi pro příkazovou řádku umožňuje vytváření vlastních funkcí využívající funkce knihovny. Uživatel tak může používat sekvenci algoritmů bez nutnosti vypisování dílčích kroků. Funkci navíc může využívat opakovaně. Ukázka vstupu funkce v aql2 viz Pro verzi webového rozhraní to navíc znamená zjednodušení vytvořeného grafu na plátně.

Ukázka formátu 3.1 Funkce provádějící minimalizaci pro nedeterministický konečný automat

```
function MinimizeNFA ( auto $automaton ) returning auto begin
    execute automaton::determinize::Determinize $automaton |
    automaton::simplify::Trim - |
    automaton::simplify::Minimize - > $result;
return $result;
end
```

Kapitola 4
Návrh

4.1 Nápověda u vstupních parametrů

Řešením problému je zobrazit u vstupních parametrů nápovědu se správným pořadovým číslem parametru a u návratových hodnot zobrazit nápovědu informující, že se jedná o „return value“.

V kódu je třeba rozhodovat o zobrazení komponenty `tooltip` dle typu parametru.

4.2 Parametr šablonovaného algoritmu

V rámci přípravy návrhu vizualizace parametru na boxu algoritmového vrcholu vyvstalo několik možností, které obsahovaly další vylepšení z hlediska celkové uživatelské přívětivosti.

4.2.1 Výběr parametru v seznamu algoritmů

Seznam algoritmů bude nabízet uživateli pouze jeden algoritmus pro všechny varianty s různými parametry, u kterého se bude nacházet `select menu` s výběrem parametru.

Na plátně dojde pouze k vizualizaci parametru pod jménem algoritmu v algoritmovém boxu. Výhodou tohoto řešení je zkrácení seznamu algoritmů, který aktuálně obsahuje pro každé přetížení a pro každou variantu šablonovaného algoritmu jeden box k výběru. Nevýhodou se naopak stává nemožnost změny parametru přímo na plátně. Nevýhodou z hlediska implementace je sjednocení algoritmů s různými parametry do jednoho. Specializace algoritmu pomocí šablony totiž není z pohledu formátu ALTu to samé co přetížení. Problematické by bylo zobrazovat algoritmus s obecnou dokumentací jako jeden a ostatní varianty se všemi detaily v algoritmovém vrcholu ukládat.

4.2.2 Výběr parametru na plátně

Druhá varianta také zobrazuje pouze jednoho zástupce algoritmu za všechny varianty. Volba parametru bude možná v `select menu` přímo v algoritmovém boxu na plátně.

Výhodou je uživatelská přívětivost. Uživatel používající algoritmus si v průběhu skládání operací může pouze překlikávat varianty algoritmu na jednom vybraném boxu.

Nevýhody zobrazení jednoho algoritmu sdílí s předchozím návrhem. Navíc nastává mnoho komplikací z implementačního hlediska, jelikož bude třeba řešit hlavně zvolení správného přetížení algoritmu a případná změna přímo v rámci plátna. Dále je potřeba použít SVG element `<foreignObject>`, aby se v rámci SVG boxu mohla vykreslit komponenta knihovny MaterialUI.

Tato komponenta ale není podporována v prohlížeči Internet Explorer [26]. Dalším problémem je vizualizace napojovacích bodů pro vstupní a výstupní parametry, pokud by byl parametrizovaný algoritmus v budoucnu přetížen.

4.2.3 Zobrazení parametru na plátně a přidání nových vrcholů

Poslední možnost nabízí ponechání seznamu algoritmů v současném stavu a po výběru na plátně zobrazit pod názvem algoritmu v boxu i jeho parametr. Tento problém se dotýkal především používání algoritmu `string::Parse`, pomocí kterého je aktuálně možné zadávat datové struktury ve formátu text-fit.

Po konzultaci s vedoucím práce bylo rozhodnuto o přidání vrcholu pro vstup a výstup, který při zadání textu automaticky použije algoritmus `string::Parse`, čímž bude uživatel plně odstíněn od použití algoritmu jako samostatného vrcholu v grafu.

Výhodou je přehlednost vizualizace parametru, navíc zobrazení algoritmů v seznamu je v souladu s formátem, který webové rozhraní dostane z ALTu.

Ne příliš kritickou nevýhodou je, že seznam algoritmů zůstává pro šablonované algoritmy stejný a stále je potřeba pro každou variantu vybrat nový box v případě změny. Nicméně vytvoření možnosti zadání vstupu bez použití algoritmu `string::Parse` uživatelskou přívětivost kompenzuje.

Nakonec byl zvolen třetí způsob řešení, u kterého jednoznačně převažují výhody. Parametr zobrazuje přehledně a nenaskytne se žádný problém při zobrazování algoritmu jak v seznamu tak na plátně. Navíc nabízí pro uživatele obecně jednodušší zadání vstupu a odstínění od parsování vstupního formátu.

4.3 Dokumentace algoritmů na plátně

Informace o algoritmu lze zobrazovat mnoha způsoby. Nutností však je vázat ovládací prvek dokumentace k boxu, jelikož na plátně se v jednu chvíli může nacházet boxů více. Přístup k dokumentaci bude zajišťovat ikona sloužící jako tlačítko, která se bude nacházet na straně boxu. Pro zobrazení dokumentace byly zvažovány následující dvě MaterialUI komponenty.

4.3.1 Dialog

`Dialog` je vhodná komponenta k zobrazení dat, zejména pokud se jedná o delší text. Informace o algoritmu aktuálně obsahují celý název algoritmu, vstupní a výstupní parametry a dokumentační popisek. Zobrazení tohoto dialogu by blokovalo větší část plátna a mohlo znepřehlednit vytváření grafu.

4.3.2 Popover

`Popover` je komponenta používaná k zobrazení obsahu přes jiné komponenty [27]. Vzhledem k velikosti zobrazovaných dat splňuje požadavek na nezakrytí celého plátna při otevření dokumentace během tvorby grafu. Data navíc může přímo zobrazit u tlačítka na boxu, takže se důležité informace pro uživatele nachází přímo v místě manipulace s boxem.

4.4 Vizualizace podrobného výstupu

Pro odlišení vrcholů s klasickým výstupem a podrobným výstupem je třeba odlišit barvu ohraničení boxu po vyhodnocení. Při výběru byly zvažovány veškeré kolize s dalšími barvami ohraničení

tak, aby nemystifikovaly uživatele, kupříkladu tím, že se jedná o chybu. Původním návrhem byla žlutá barva, která nakonec nebyla zvolena z důvodu nespolehlivého zobrazování s různými hodnotami jasu na monitoru. V závěru byla zvolena tmavě zelená barva indikující jiný typ výstupu.

Pro text podrobného výstupu je třeba použít neproporcionální písmo. K zachování konzistence je navrženo využití písma `Roboto Mono`, které zachovává konzistenci aktuálním fontem `Roboto`.

4.5 Inovace vstupů a výstupů pro komplexní datové typy

4.5.1 Vstup

Prvotní návrh uvažoval o dvou možnostech reprezentace tohoto vstupu z hlediska toho, v jakém vstupním vrcholu by se měl nacházet.

1. Vytvořit k již existujícímu vstupnímu vrcholu další vstupní vrchol `input&parse`.

   Výhody

   - jasně vymezené zadání primitivních typů (string, number, bool) a komplexních typů

   Nevýhody

   - uživatel musí mít znalost o rozdílech těchto dvou vstupů, při tvorbě grafu tedy musí rozhodovat o tom, který vstupní vrchol zvolit
   - při úpravě grafu na plátně může uživatel vyžadovat změnu datového typu vstupu, v tomto případě je potřeba vybraný vstupní vrchol na plátně smazat a vybrat vstupní vrchol nový

2. Sloučit tento typ vstupu do existujícího vstupního vrcholu.

   Výhody

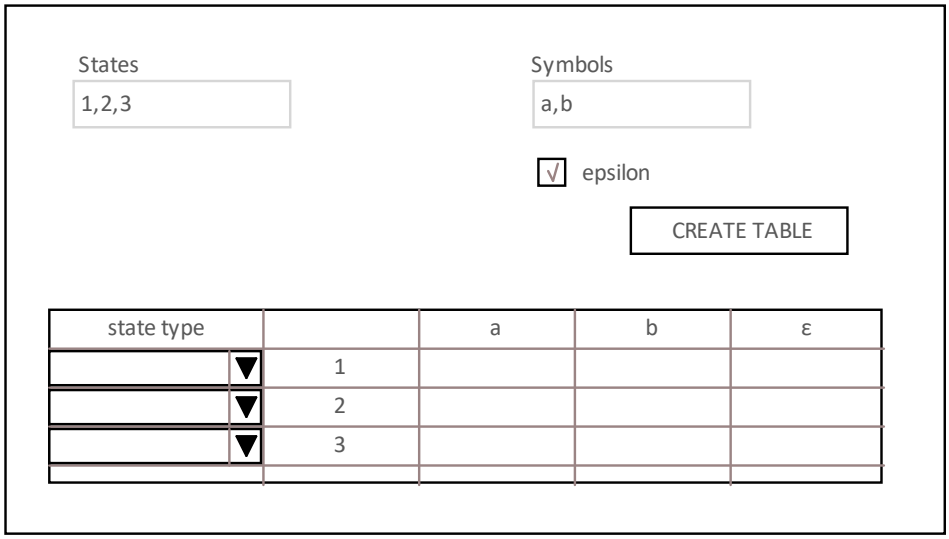
   - uživatel nemusí přemýšlet o významu či typu vstupního vrcholu
   - změnu datového typu vstupu lze provést přímo na plátně v jednom jediném vstupním vrcholu
   - znovupoužitelnost dialogu pro zadání vstupu
   - zachování přehlednosti v seznamu algoritmů

Vzhledem k jasné převaze výhod druhého návrhu bylo zvoleno použití jednoho typu vstupního vrcholu pro všechny vrcholy. Informace o tom, do kterého datového typu bude vstup parsován, půjde zvolit pomocí `select menu` s datovými typy, které budou odděleny hlavičkou jako primitivní a komplexní typy.

4.5.2 Formulář pro gramatiku

Prvním zásadním rozhodnutím je místo, kde bude možné díky formuláři zadat vstup. Pro zachování snadné orientace při zadávání vstupu bylo navrženo ponechat jeden vstupní vrchol (a tím pádem i dialog). Pro oddělení textového a formulářového vstupu je vhodné použít komponentu `tab` neboli záložku, kterou se lze rychle navigovat mezi možnými způsoby vstupu zvoleného datového typu.

Mezi ovládací prvky v prvotním návrhu patří `select menu` pro výběr typu gramatiky a dynamický formulář pro pravidla. Dynamický formulář při otevření bude nabízet řádek s textovými poli pro levou a pravou stranu pravidla. Další řádky bude možné přidat pomocí tlačítka. Umožněno bude také mazat jednotlivé řádky. Řešení zadání počátečního symbolu bylo v tomto návrhu realizováno tak, že první pravidlo bude vždy obsahovat počáteční symbol. Toto řešení počátečního symbolu bylo zamítnuto vzhledem k tomu, že ne vždy obsahuje první pravidlo počáteční symbol a gramatika může být z pohledu pravidel opravdu obsáhlá.



Obrázek 4.1 Návrh tabulky s generováním velikosti

Další návrh obsahoval povinné textové pole pro jednoznačné zadání počátečního symbolu.

Po schválení návrhu grafické stránky je třeba zvážit, jakým způsobem zadávat vstupy do textového pole. Pravidla s totožnou levou stranou se často píší v rámci jednoho pravidla s více pravými stranami. Toto je třeba umožnit i ve formuláři, jelikož přidávání stále nových řádků pro jednotlivá pravidla by bylo časově náročné a z hlediska ovládání nepřívětivé. V tomto ohledu je navrhována konzistence s text-fit formátem, který pravidla odděluje znakem svislé čáry |.

Dalším úskalím při zadání vstupu jsou epsilon pravidla, kde je také navrhována konzistence s text-fit formátem. Pro rozpoznání terminálů a neterminálů z pravidel je nutné zavést pravidla pro symboly. Aby bylo uživateli umožněno tyto znaky zadávat, je třeba každý symbol oddělovat mezerou.

4.5.3 Tabulkový vstup pro automat

Hlavním cílem návrhu tabulkového formuláře pro konečný automat je umožnit rychlé zadání vstupu. Tabulka může obsahovat mnoho řádků a sloupců, jejichž počet není na začátku znám a je třeba řešit její dynamické zvětšování.

Při návrhu podoby tabulky je potřeba myslet i na implementaci tabulkového vstupu z důvodu konzistence vzhledu a zvolit komponentu z Material UI knihovny, která půjde použít v obou případech. Z tohoto důvodu byla zvolena komponenta `Table`.

K řešení ovládání pohybu a dynamického zvětšování tabulky byly vzneseny následující návrhy:

Generování velikosti dle abecedy

Uživatel nejprve zadá pomocí textového pole abecedu a stavy, podle kterých se vygeneruje počet sloupců tabulky. Řádky budou přidávány pomocí tlačítka.

Výhodou tohoto přístupu je, že se uživatel nemusí při vyplňování tabulky pozastavovat nad přidáváním sloupců.

Nevýhodou je vlastně nadbytečný krok z pohledu UX, kdy je potřeba nejprve vyplnit textové pole.

Dynamická změna velikosti pomocí tlačítek

Další možností je použití tlačítek pro přidání sloupce i řádku.

Výhodou je na první pohled srozumitelné ovládání, nevýhodou zase nutné používání tlačítka.

Dynamická změna velikosti pomocí kláves

Posledním návrhem je umožnit pohyb pomocí kláves po řádku tabulky a po dosažení posledního pole v řádku vytvořit řádek další. Uvažováno bylo využití klávesy tabulátoru a enter klávesy. Možné je přidat také podobné přidání sloupce, to už by ale pro uživatele mohlo být matoucí a v tomto případě návrh zůstává u využití tlačítka.

Výhodou je rychlý pohyb i přidávání nových řádků při vyplňování tabulky, nevýhodou je, že je možné zvolit takovou klávesu, která pro uživatele nebude dostatečně intuitivní.

Při volbě následného postupu, byly zvažovány všechny výše uvedené přístupy i jejich nejrůznější kombinace. Nakonec byl zvolen poslední způsob, který využívá klávesové zkratky. Přidání řádku bude provedeno po stisknutí klávesy Enter. Sloupce budou přidávány pomocí tlačítka. Smazání řádku bude stejné jako u formuláře po stisku tlačítka a smazání sloupce bude možné provést stisknutím klávesy Delete. Označení stavu jako počátečního či konečného bude realizováno pomocí `select menu` s názornými ikonami šipek.

4.5.4 Výstup

Požadovaný cíl je, aby výpis komplexních datových typů a pravdivostních hodnot byl ve formátu text-fit. Stejně jako byl v případě vstupních vrcholů použit algoritmus `string::Parse`, u výstupního vrcholu bude využit algoritmus `string::Compose` převádějící datové struktury do textového formátu.

4.5.5 Tabulkový výstup pro automat

U tabulkového výstupu bylo původně zvažováno, do kterého výstupního vrcholu by měl být zahrnut. Po implementaci výstupních vrcholů s parsováním bylo vhodné použití tohoto výstupu přímo v nich.

Hlavním výstupem by měla být tabulka, nicméně textový formát je důležité zobrazovat také z důvodu zachování kompatibility s ALT knihovnou i přes příkazovou řádku. Pro tyto dva formáty bylo navrženo použití tlačítka pro zobrazení text-fit formátu pod tabulkou.

Vybraným řešením se ale nakonec stalo použití záložek, stejně jako tomu je u vstupních vrcholů z důvodu konzistence.

Pro zobrazení tabulky je možné použít Material UI komponentu `Data Grid` a `Table`. Komponenta `Data Grid` slouží pro zobrazování více dat, nabízí různé možnosti řazení a filtrování. Nabízí i verzi, ve které je možné data editovat, kterou lze použít při zadání vstupu. Nicméně úpravy jsou zamýšleny spíš pro menší změny než pro zadání vstupu. Tabulka je v našem případě vizuálně přehlednější a v kombinaci s drobnými úpravami také použitelná pro získání vstupu. Navíc žádné řazení nebo filtrování v tomto případě není žádoucí.

4.5.6 Výstup gramatiky

I zde je třeba zachovat textový výstup, takže je z důvodu konzistence příhodné použít překlikávání obsahu dialogu pomocí záložek. Komponenta záložky pro výstup je navíc navržena tak, aby šla přepoužít a byl dodržen tzv. „DRY princip“, tedy vyhýbání se kopírování kódu. V dialogu bude použita MaterialUI komponenta `Card` zobrazující typ gramatiky, terminály, neterminály, počáteční symbol a pravidla gramatiky.

4.6 Vyhledávání v algoritmech

Úplně prvotním grafickým návrhem bylo využití filtrování jako u komponenty `Data Grid`. Po bližším zkoumání zdrojových kódu ale bylo zjištěno, že toto řešení nelze aplikovat na rekurzivní seznam, kterým aktuálně je komponenta představující seznam algoritmů.

Hlavním bodem návrhu z pohledu uživatelské přívětivosti bylo zvolení prohledávaných kategorií. Zvažováno bylo `select menu` s možností výběru více možností nebo například zaškrtávací `checkbox`.

Po konzultaci s vedoucím bakalářské práce bylo zvoleno řešení s jednotlivými `checkboxy`, díky čemuž bude možno prohledávané kategorie kombinovat a hledat v nich zároveň.

Z implementačního hlediska byl hlavní záměr použít rekurzivní algoritmus, který zachová funkcionalitu filtrování během psaní uživatele a dokáže tyto filtry kombinovat. Filtry by měly být jednoduše rozšiřitelné, pokud by se přidávala další informace o algoritmu. Jednotlivé filtry by tedy měly jít rozšiřovat a dodržovat stejné rozhraní pro filtrovací funkci. Algoritmus nebude vyhledávat pouze na úrovni názvu kategorie, ale bude třeba filtrovat i v podstruktuře, která obsahuje jednotlivá přetížení.

4.7 Seznam přetížení algoritmů

Navrženy byly dva velmi odlišné přístupy. Hlavní cíle, které návrh musí splnit, je zkrácení seznamu algoritmů, vizualizace dostupných parametrů, zobrazení správného počtu napojovacích bodů na boxu a správná kontrola parametrů.

4.7.1 Výběr přetížení uživatelem

Algoritmus bude reprezentován jedním boxem v seznamu, který lze vybrat na plátno. Před umístěním na plátno bude zobrazen dialog pro výběr přetížení. Výhodou je zkrácení seznamu algoritmů, nevýhodou je, že uživatel musí absolvovat krok navíc. Ověření typů vstupních parametrů navíc zůstane řešené stejně – validace by opět musela probíhat dle zvoleného přetížení, avšak ve vyhodnocení by byl použit pouze daný algoritmus.

4.7.2 Sloučení přetížení do jedné reprezentace algoritmu

V druhém návrhu jsou sloučena přetížení se stejným počtem vstupních parametrů do jednoho boxu. Box bude nabízet zobrazení všechna dostupná přetížení ve formě dialogu či komponenty `popover`. Po výběru na plátno budou parametry kontrolovány podle povolené množiny parametrů. V případě tohoto návrhu převažují jeho výhody. Uživatel nemusí dlouho vyhledávat daná přetížení, informace o nich ale neztrácí, jelikož je bude možné kdykoliv zobrazit. Kontrola typů sice bude benevolentnější, ale uživatel nebude varován v případě, kdy algoritmus lze vyhodnotit. Z výše uvedených důvodů byl tento postup zvolen k implementaci.

4.8 Filtrování algoritmů

S výhledem na budoucí rozšíření bylo navrženo použití konfiguračního souboru, který bude obsahovat regulární výrazy. Využit by byl způsob třídění pomocí plně kvalifikovaného jména a bylo by jednoduché vyřadit jak celé kategorie, tak jednotlivé algoritmy v seznamu. Filtrování bude aplikováno po získání algoritmů ve webovém rozhraní před jejich kategorizací.

Po následné konzultaci bylo zvoleno přímočařejší řešení, a to filtrování přímo na straně workera před odesláním seznamu algoritmů webovému rozhraní. Využito bude opět regulárních výrazů, tentokrát ale přímo pomocí proměnné v části worker.

4.9 Vlastní funkce

Zadání vstupu pro vytvoření vlastní funkce bylo zvažováno v grafické či textové podobě. Pro prvotní zavedení této novinky ve webovém rozhraní byla zvolena nejprve textová podoba. Vytvoření vlastní funkce proběhne pomocí formuláře, ve kterém bude zadáno jméno, parametry, návratová hodnota a tělo funkce. Z dostupných parametrů se vytvoří připojovací body.

Z hlediska vyhodnocení bude potřeba pozměnit formát těla požadavku odeslaného na koncový bod `/evaluate`. Vzhledem k tomu, že je neprve nutné funkci vytvořit před tím, než se začne používat, bude JSON soubor začínat seznamem funkcí a následně bude obsahovat informace o grafu.

Přepoužití vlastní funkce bude ve webovém uživatelské rozhraní umožněno pomocí kopírovací ikony umístěné přímo na boxu na plátně.

Kapitola 5
Implementace

5.1 Nápověda u vstupních parametrů

V komponentě `Param` byla opravena podmínka rozhodující o zobrazovaném obsahu `tooltipu`. Nově se rozhoduje na základě atributu `order`, který udává pořadí vstupního parametru nebo indikuje, že se jedná o návratovou hodnotu. Využito je zde nahlížení na datové typy v Typescriptu jako na množiny, atribut `order` může být typu číslo nebo typ `'return'`. Zjišťuje tedy, o kolikátý vstupní parametr se jedná, nebo zda má hodnotu `return`.

5.2 Parametr šablonovaného algoritmu

Nápis na boxu byl změněn pro všechny šablonované algoritmy. Pod názvem se nově zobrazuje název vybraného parametru, toto bylo vyřešeno v komponentě `Node.tsx` reprezentující box v grafu na plátně. V rámci implementace bylo třeba zajistit správnou velikost boxu, který dříve nebyl navržen tak, aby zobrazoval víceřádkový text.

Prvotní implementace měla využít atributu `dominantBaseline` pro SVG element `text`, která měla centrovat nápis doprostřed boxu. SVG nenabízí žádné prostředky, jak vycentrovat text doprostřed SVG obdélníku. Nastavení pozice textu na řádku pomocí `dominantBaseline` si nedokáže poradit s víceřádkovým textem a navíc jeho kompatibilita s prohlížeči není z dokumentace známá [28].

V tomto případě nakonec bylo použito stejné řešení jako u víceřádkového textu ve stavech ve Statemakeru.

Vytvořen byl nový hook `useNodeSize`, který podle velikosti textu změní velikost boxu, zároveň se rozhoduje na počtu vstupních parametrů, aby boxy s více vstupními parametry dokázaly vykreslit napojovací body, tak aby byly ovladatelné.

5.3 Dokumentace algoritmů na plátně

Do interface `AlgorithmNode` zapouzdřujícího data o boxu byla přidána informace o dokumentačním popisku. Následně bylo do boxů, které reprezentují algoritmy přidána ikona do pravého spodního rohu sloužící jako tlačítko pro zobrazení popoveru. Popover i tlačítko bylo vyčleněno do samostatné třídy `DocsPopover`.

5.4 Vizualizace podrobného výstupu

Implementováno bylo zobrazení tmavě zeleného ohraničení boxu po vyhodnocení na základě typu boxu a toho zda obsahuje výstup.

Písmo `Roboto Mono` je zobrazeno pouze pro podrobný výstup a ne pro klasický.

5.5 Inovace vstupů a výstupů pro komplexní datové typy

5.5.1 Vstup

Rozšíření z hlediska UI se týkalo především komponenty `InputDialog`. Typ vstupu uživatel vybírá pomocí `select menu` komponenty, která je rozdělená pomocí hlaviček na primitivní typy – string, bool, number a parsované typy. Všechny vrcholy jsou vnitřně popsány typem vrcholu. V tomto případě bylo nutné vytvořit další typy vrcholů, aby bylo možné tento vstup zpracovat pomocí algoritmu `std::Parse` ve správné variantě.

K nastavování parametrů nově vznikajícího vrcholu, k otevírání dialogu či znovu otevírání dialogu určitého vrcholu je využíván Redux. Díky ukládání informacích o všech vrcholech bylo možné znovuotevřený dialog zobrazit i poslední zadanou hodnotou.

U znovu otevírání dialogu již umístěného vrcholu bylo třeba změnit, na základě čeho se objeví správný typ. Původní verze spoléhala na datový typ vstupu jako takového. Nyní byl tento přístup změněn a rozhoduje se na základě samotného typu vrcholu.

Pro serializaci vytvořeného grafu byl implementován nový filtr dle typu vrcholu. Pro všechny parsované typy je společný a na základě daného typu vyhledá, jaký parametr je třeba ke spuštění `string::Parse` algoritmu použít.

Aktuálně tedy není potřeba používat `string::Parse` algoritmu pro získání daného datového typu, ale výstup vstupního boxu je přímo objekt.

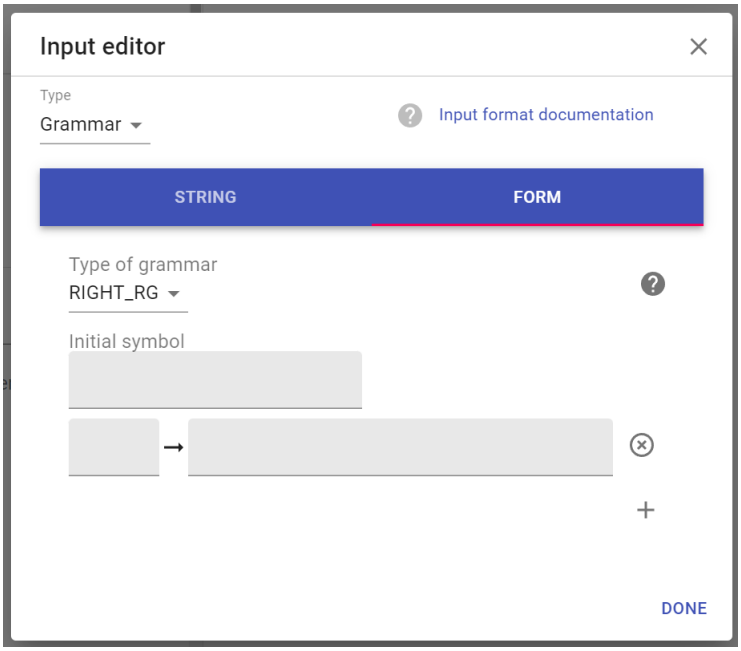
V rámci řešení tohoto problému byl objeven další nedostatek předchozí verze. Jednalo se o kontrolu kompatibility datových typů při změně datového typu uvnitř vrcholu ve chvíli, kdy je již vstupní vrchol napojen jako vstupní parametr nějakého algoritmového vrcholu. Pro tyto případy byla přidána kontrola všech napojených hran při změně typu vstupu s již napojenou hranou. Při nekompatibilitě datových typů je hrana zbarvena červeně a uživatel je upozorněn pomocí upozornění v komponentě `snackbar`.

5.5.2 Formulář pro gramatiku

Formulář pro gramatiku byl vyvíjen v rámci nové React komponenty `InputGrammarField`, která je vložena do existujícího dialogu `InputDialog`. Závislost mezi těmito dvěma komponentami byla nakonec vyřešena tak, že vnější dialog předává pomocí `props` funkce, kterými se po akci ve vnitřní komponentě odešlou data zpět do vnějšího dialogu. Toto řešení bylo zvoleno z důvodu toho, že vnější dialog potvrzuje data ke zpracování, tato odpovědnost mu byla ponechána. Navíc tak bylo možné implementovat varování uživatele při nesprávném vstupu a následně nezavření dialogu s nesprávným vstupem.

V `InputGrammarField` byla použita Material UI komponenta `Tabs`, která je vykreslena jako dvě záložky „STRING“ a „FORM“, pomocí kterých uživatel překlikává mezi vstupem v textovém formátu text-fit a vstupem ve formě formuláře. Formulář byl implementován jako dynamický formulář, je tedy možné jednotlivé řádky s textovými poli přidávat a ubírat.

Formulář jako takový byl původně řešen pomocí hooku `useState`. Během lokálního manuálního testování byl ale objeven problém. Pokud byl stav aktualizován pomocí `useState` hooku a následně bylo k proměnné přistupováno, často byla vrácena předchozí hodnota a ne aktuální. Zmíněné chování je zapříčiněno asynchronním chováním hooku, k vyřešení situace byla nakonec zvolena funkce obalující `useSyncState` hook [29]. Další silně zvažovanou možností bylo použití



Obrázek 5.1 Formulář pro gramatiku

Reduxu. Vzhledem k tomu, že se nejedná o velmi komplexní formuláře z pohledu stavu a Redux je v rámci celého projektu používán k řešení globálnějších problémů, nakonec tento způsob zvolen nebyl.

Kvůli upřesnění formátu vstupu byla přidána nápověda s instrukcemi pro uživatele, která byla také vyčleněna do samostatné komponenty. Původní záměr byl zobrazovat nápovědu na odkazu mířícím na dokumentaci text-fit formátu. Toto řešení se ale jevilo jako matoucí. Nakonec byla pro nápovědu zvolena klasická informační ikona, která se nachází přímo v rámci formuláře.

K uchovávání informací o gramatice byl vytvořen nový interface `GrammarData`, jelikož webové rozhraní dosud s gramatikami na této úrovni nepracovalo. Implementováno bylo i vytvoření této struktury z textových polí.

Také bylo třeba přidat nový typ vrcholu, který indikuje, jaký je obsah vstupního vrcholu. Tyto typy jsou využívány také ke znovuotevření dialogu na plátně, kdy uživatel očekává, že zadaná data budou zobrazena ve stavu, v jakém byla uložena. Veškeré ukládání a znovuzískání dat je řešeno pomocí Reduxu a nově implementovaná akce `setInputGrammar`, kde je dle vstupu klasifikován i výstupní typ vrcholu.

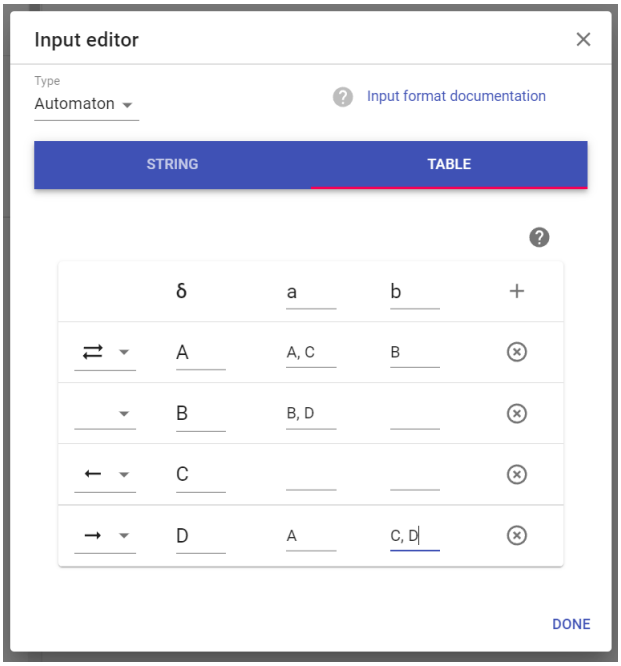
Zpracování vrcholu s takto zadaným vstupem se serializuje na algoritmus `string::Parse` s textovým vstupem. Nutné tedy bylo implementovat serializaci nového typu vrcholu a převod z interní struktury dat do text-fit formátu.

Formulář uživatele upozorňuje na nevyplněný vstup, upozorní uživatele i na fakt, že počáteční symbol není neterminál.

Pro funkce s `GrammarData` strukturou byly napsány jednotkové testy.

5.5.3 Tabulkový vstup pro automat

Vytvořen byl nový typ vstupního vrcholu. Pro tabulkový vstup byl vytvořen další obsah vstupního dialogu. Opět je zachován formát ve formě text-fit podobě pomocí záložek. Komponenta je implementovaná pomocí Material UI komponenty.



Obrázek 5.2 Formulář pro automat

Dále bylo nutné přidat novou datovou strukturu pro tabulku. `Row` představuje řádek přechodů z jednoho stavu, řídí se podle pořadí symbolů zadaných v hlavičce tabulky. Implementována byla také konverze do `StateData` formátu, který webui používá pro reprezentaci automatu. Konverze se také nachází před vyhodnocením, kde je provedena do text-fit formátu.

Řešeno také bylo znovuotevření vstupního dialogu, kdy nejprve docházelo k přeměně na `StateData` hned po uložení dialog. Po znovuotevření ale nebyl dialog v naprosto stejném stavu jako předtím.

Stav komponenty pro vstup automatu je řízen z dialogu, který předává funkce pomocí `props`.

Pohyb pomocí klávesy Enter byl implementován pomocí useRef hooku, který umožňuje přístup k jednotlivým textovým polím. Pomocí indexů je tedy možné se vždy posunout na další pole. Po dosažení konce řádku je přidán nový řádek a využit hook useRef, který při změně délky tabulky posune foxus na nový řádek. Mazání řádku je umožněno stisknutím tlačíka, stejně tak přidání sloupce. Z důvodu velikosti tabulky byla také odstraněna statická velikost na dynamicky zvětšovanou s dolní i horní mezí. Tlačítkem Delete lze odstranit celý sloupec.

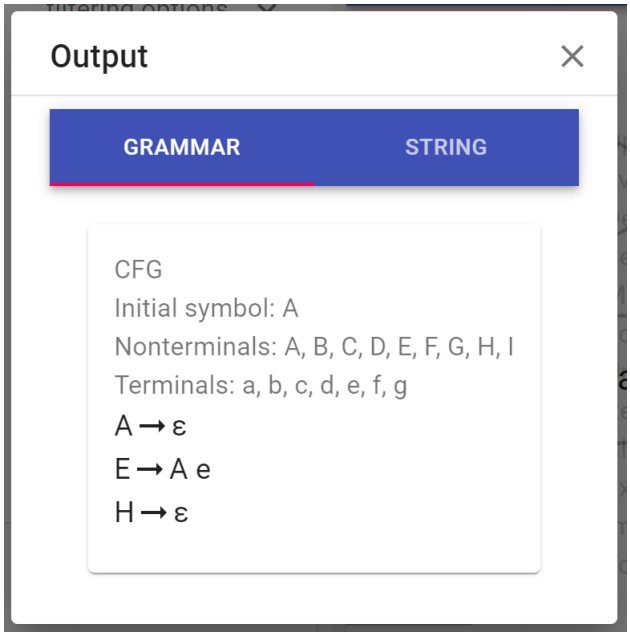
K tabulce byla přidána také krátka nápověda především kvůli zadefinování oddělovačů a epsilon symbolu.

5.5.4 Výstup

V tomto případě byl přidán pro výstup komplexních struktur algoritmus `string::Compose`, který z nich vytvoří textový formát. Aplikace tohoto algoritmu je přidána před všechny výstupy datových typů, které je třeba ještě přetransformovat do text-fit formátu. Děje se tak při serializaci.

5.5.5 Výstup gramatiky

Do typů výstupních vrcholů byl přidán další typ, tentokrát pro gramatiku. Výstupní vrcholy nově ukládají svůj typ do Reduxu, aby bylo možné správně vizualizovat typ – typ výstupu je



Obrázek 5.3 Výstup gramatiky

v tomto případě vždy text. Dialog je rozdělen stejně jako v případě vstupů pomocí záložek, v jedné vizualizovaná gramatika a v druhé textový formát.

5.5.6 Tabulkový výstup pro automat

Stejně jako u gramatiky, i zde byl přidán další typ výstupního vrcholu, podle kterého je řízen obsah dialogu. Pro komponentu tabulky byla z důvodu konzistence zvolena Material UI komponenta `table`, v tomto případě bez textových polí.

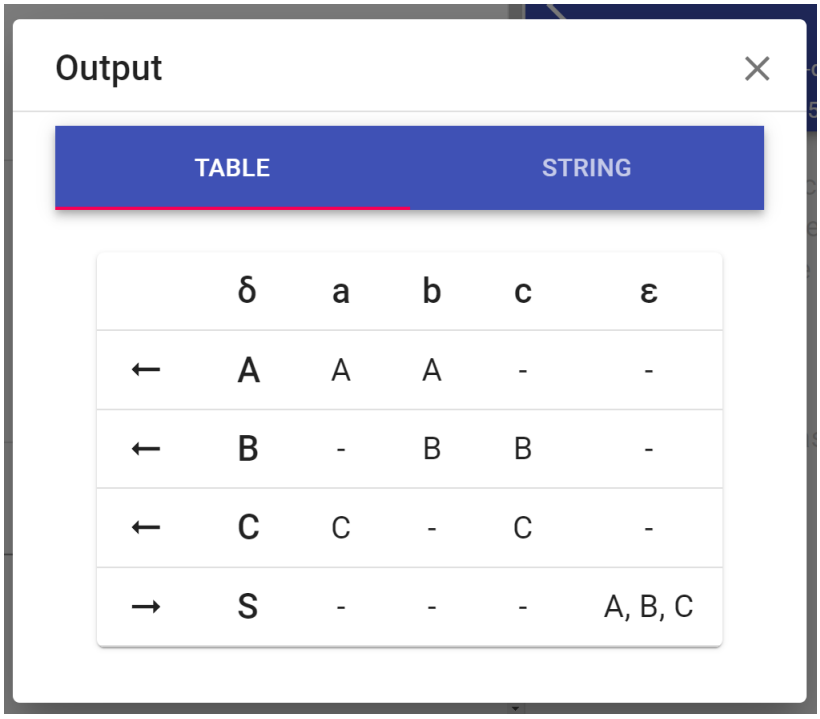
Dialog je rozdělen stejně jako v případě vstupů pomocí záložek, v jedné tabulka přechodové funkce automatu a v druhé reprezentace v text-fit formátu. Z důvodu většího obsahu dialog bylo také potřeba změnit změnu velikosti dialogu, která byla prozatím fixní.

Z datové struktury automatu bylo třeba získat abecedu, která je seřazena a použita jako hlavička tabulky. Pro vizualizaci počátečních a koncových bodů jsou opět voleny ikony šipek.

5.6 Vyhledávání v algoritmech

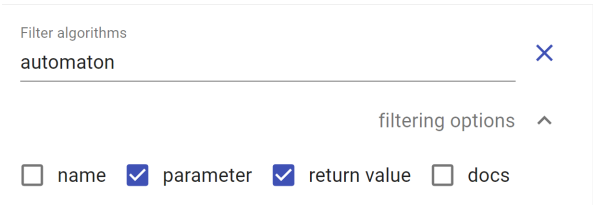
Do levého panelu byly přidány `checkbox` komponenty pro každou kategorii, ve kterých je možné vyhledávat – „name“, „parameter“, „return value“, „docs“. Základní nastavení bez výběru je vyhledávání ve všech kategoriích.

Pro komponenty seznamu algoritmů jsou pomocí `props` předané informace o vyhledávané hodnotě i parametrech, ve kterých má vyhledávání proběhnout. Parametry filtru jsou reprezentovány pomocí datových struktur `NameFilter`, `ParamFilter`, `ReturnValueFilter` a `DocsFilter`, které implementují `Filter` udávající rozhraní metody `filter`. Tu již každý filtr implementuje různě, `filter` metoda je určena pro vyhledávání v přetížení. V `ParamFilter` slouží k vyhledávání řetězce v názvu či typu parametru, v `NameFilter` například tato metoda vždycky vrátí `true`, jelikož nemá vyhledávání ve jméně na jednotlivá přetížení vliv a pokud je zobrazena kategorie, pak jsou zobrazeny i všechny její přetížení.



Obrázek 5.4 Výstup automatu ve formě tabulky

Funkcionalita vyhledávání byla přesunuta do samostatného souboru v `utils filtering`, zde se nachází rekurzivní algoritmus a pomocné filtrovací funkce. Funkce prohledává rekurzivně podkategorie. Prohledává vždy aktuální kategorii – pokud se ve filtrech nachází filtrování dle jména, prohledá jméno kategorie. Pokud jsou v kategorii již přetížení, prohledává se v přetíženích a následně se prohledává subkategorie. Pokud se v kategorii nenachází vyhledávaný výraz, nevykreslí se, pokud se zde v ní nachází, je vykreslena a předává filtr podkategoriím, tak aby byly vykresleny ty, ve kterých se filtr nachází také. Pro dílčí filtrovací funkce byly napsány jednotkové testy.



Obrázek 5.5 Vyhledávání v algoritmech

5.7 Seznam přetížení algoritmů

Nejdříve byla přidáno do kategorizace algoritmů sloučení přetížení dle počtu vstupních parametrů. Algoritmový vrchol tedy aktuálně představuje několik přetížení, o kterých si drží informace v Reduxu kvůli kontrolám typu a dokumentaci. `Popover` s dokumentací se nyní nachází i v seznamu algoritmů, kde je seznam jednotlivých algoritmů, stejným způsobem se zobrazuje i na plátně. Typová kontrola hledá zda se oba typy nachází v množině výstupu prvního a v množině daného parametru vstupu druhého vrcholu. Dále bylo opraveno filtrování, kde je vyhledávání re-

alizováno i v jednotlivých přetíženích, které vrchol představuje. Zobrazení typů je koncipováno, aby uživatel získal co nejvíce informací již na první pohled. Pokud mají tedy všechna sloučená přetížení stejný typ parametru nebo výstupu, tento typ je zobrazen. Pokud ne, ale typy mají společný jmenný prostor, je tato část názvu použitá jako jméno pro parametru a typ je obecný „overload type“. Tím je uživatel případně odkázán na bližší informace v `popover` komponentně, která se zobrazí po stisknutí informačního tlačítka.

5.8 Filtrování algoritmů

Regulární výrazy představující algoritmy, které nemají být ve výsledku seznamu algoritmů pro webové rozhraní, byly přidány do pole v `introspect.cpp`, která tvoří JSON reprezentaci dostupných algoritmů. Všechny algoritmy, které se shodnou s nějakým regulárním výrazem, jsou při tvorbě JSON reprezentace přeskočeny.

5.9 Vlastní funkce

Pro vytvoření vlastního algoritmového vrcholu bylo nutné přidat další typ vrcholu do levého panelu s výběrem vrcholů, který po umístění na plátno otevře dialog s formulářem pro vytvoření funkce. Dialog je ve webovém rozhraní jinak modální, při otevření lze ovládat pouze obsah dialogu, dokud není zavřen. U vlastních algoritmových vrcholů je ale žádoucí umožnit uživateli vyhledávat informace o algoritmech a zobrazovat si jejich dokumentaci, z tohoto důvodu je tedy možné interagovat i s komponentami na hlavní obrazovce, i když je dialog otevřen. Na straně workera byla přidána registrace nové funkce, která je dostupná pro celé vyhodnocení grafu. Funkce lze v grafu používat na více místech, možné je tedy vrcholy množit pomocí kopírovacího tlačítka přímo na boxu na plátně.

Kapitola 6
Testování

Testování je důležitou činností v oblasti softwarového inženýrství. Hlavním cílem testování je ověřit splnění požadavků a spolehlivost softwaru. V případě webového rozhraní se z velké části testování věnuje i uživatelské přívětivosti.

Testování v rámci této práce zahrnovalo jednotkové testy v případě menších úseků kódu. Využit byl framework Jest, který byl používán v projektu již od implementace Statemakeru [1, s. 39]. Dále probíhalo manuální testování zejména ve fázi před konečným nasazením a uživatelským testováním. Cílem závěrečného uživatelského testování není pouze otestovat funkčnost nových rozšíření, ale otestovat také použitelnost. V implementační části práce došlo k rozšíření o další možnosti zadání vstupu, u kterých je třeba ověřit, zda jsou uživatelsky přívětivé. Testovací plán tohoto typu testu byl koncipován tak, aby pokryl oba dva požadavky.

Testování použitelnosti slouží k identifikaci existujících problémů, zjištění příležitostí pro zlepšení a získání informací o chování uživatele. Účastníkem takového testování je facilitátor a tester. Facilitátor zadává úkoly testerovi bez dalšího ovlivňování, zároveň zaznamenává jeho chování a získává další informace. Tester plní úkoly dle instrukcí a poskytuje zpětnou vazbu. Často je po uživatelích požadováno myslet nahlas – pro získání relevantních informací. [30]

6.1 Testovací plán uživatelského testování

Plán pro uživatelské testování obsahuje následující požadavky.

Požadavky na testery

- tester absolvoval předmět BI-AAG
- tester má povědomí o knihovně ALT, webovém rozhraní ALT nebo jej aktivně využíval při studiu (optimální jsou různé stupně zkušenosti u jednotlivých testerů)
- minimální počet testerů je 3

Požadavky na úkoly

- úkoly pro testery budou podobné reálným příkladům z předmětu BI-AAG
- úkoly budou formulovány tak, aby se tester mohl co nejlépe vžít do reálné situace
- úkoly nebudou zadávány příliš detailně, aby bylo možné sledovat chování uživatele kvůli zjištění přehlednosti webu
- úkoly budou tvořené v souladu s diagramem případů užití kvůli ověření zachování existujících funkcionalit

- testováno bude minimálně pomocí 5 scénářů/úkolů, které budou zaměřeny na nově implementované řešení a jeho kompatibilitu s existujícím řešením

Požadavky na prostředí

- testování proběhne z minimálně 3 nejpoužívanějších prohlížečů pro verifikaci kompatibility

Požadavky na průběh testování

- testování proběhne osobně
- testeři budou vyzváni k přemýšlení nahlas nad plněním úkolů v průběhu celého testování
- testeři vyplní po provedení úkolu dotazník

Požadavky na vyhodnocení

- vyhodnoceno bude, zda úkol měl očekávaný výstup (neskončil např. chybou)
- popsána bude orientace testerů v průběhu plnění úkolů
- budou vyhodnoceny odpovědi na otázky z dotazníku

6.2 Průběh testování

Testování absolvovali tři testeři v různých prohlížečích (Google Chrome – tester č. 1, Mozilla Firefox – tester č. 2 a Microsoft Edge – tester č. 3) na různých prohlížečích. Všichni zúčastnění absolvovali kurz BI-AAG, během kterého dva příležitostně využili aql2 přes příkazovou řádku (tester č. 1, tester č. 2) a dva webové rozhraní (tester č. 1, tester č. 3). Testování probíhalo při osobním setkání, testeři dostali úkoly vytištěné na papíře a byli požádání o sdílení myšlenek nahlas. V průběhu byli všichni sledováni a po skončení došlo formou rozhovoru k zodpovězení otázek z dotazníku níže.

Všechny úkoly byly do scénářů převzaty ze Sbírky řešených příkladů [3], se kterými se testeři mohli během studia setkat. Úkoly jsou podány postupně od detailněji zadaných k zadaným volnějším způsobem kvůli zjištění orientace uživatele v prostředí webového rozhraní.

6.3 Testovací případ 1

Případ se zaměřuje na testování formuláře pro gramatiku a výstupu konečného automatu reprezentovaného tabulkou. Zároveň se jedná o regresní test pro export do Statemakeru, export do grafu do souboru, jeho vyhodnocení a jeho import a znovuvyhodnocení. Sledováno je, jak se testerovi ovládá nový formulář pro gramatiku, jak se orientuje ve vstupních a výstupních vrcholech.

6.3.1 Testovací scénář

Byl Vám zadán úkol na předmět BI-AAG a chcete si ověřit správný výsledek.

Následující regulární gramatiku převeďte na automat a na regulární výraz. Automat si chcete zobrazit jako tabulku, stavový diagram a také chcete zobrazit grafický výstup výsledného diagramu z `dot` formátu. Vytvořte dle zadání graf a vyhodnoťte ho.

Graf si vyexportujte, graf z plátna celý smažte a následně znovu importujte a znovu ho vyhodnoťte.

Zadaná gramatika: $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde

$$\begin{aligned} P = \{S &\to aA \mid bB \mid \varepsilon, \\ A &\to aA \mid aB \mid c, \\ B &\to cA \mid b\} \end{aligned}$$

6.3.2 Očekávaný výsledek

Graf se vyhodnotí a výstupem bude konečný automat ve 3 různých výstupních vrcholech a regulární výraz v 1 výstupním. Export proběhne bez chybové hlášky, import vytvoří stejný graf jako ten původní a vyhodnocení proběhne i podruhé bez potíží.

6.3.3 Výsledek

Ve všech prohlížečích proběhl test bez potíží a s očekávaným výsledkem.

Všichni zúčastnění využili pro zadání vstupu nový formulář pro gramatiku. U jednoho z testerů došlo pouze ke zmatení, jaký typ gramatiky má zvolit. Kvůli nekompatibilitě zvoleného typu gramatiky a vstupu algoritmu dokonce hledal algoritmus pro převod na regulární gramatiku. Pro běžné studenty BI-AAG se jeví také jako matoucí rozdělení typů regulárních gramatik na pravou a levou.

Každý ze zúčastněných si po chvíli zadávání vstupu zobrazil nápovědu s formátem vstupu a již napsaná zadaná pravidla museli opravit. Jeden z testerů si stěžoval na příliš dlouhé instrukce. Na obrazovce s prohlížečem Google Chrome se nápověda nezobrazila celá, a to z důvodu nastaveného přiblížení v prohlížeči. Při zobrazení na hodnotu 100 % podobný problém nenastal. V nápovědě byly dobře hodnoceny vhodně zvolené příklady u jednotlivých instrukcí.

K vyhledání algoritmu využili 2 ze 3 testerů intuitivně vyhledávání.

Při napojování výstupních vrcholů došlo u všech zúčastněných k menšímu či většímu zmatení kvůli výběru výstupních vrcholů. Komentáře se shodovaly především v tom, že výstupních vrcholů je příliš mnoho. Nejsou si tedy jisti, v kterém vrcholu najdou jaký výstup a proč musí do grafu přidávat několik výstupních vrcholů.

6.4 Testovací případ 2

Testován je formulář pro tabulku, výpis podrobného výstupu a také úprava předchozího grafu. Zároveň je regresně testován export do `svg` formátu, export, import grafu a jeho znovuvyhodnocení.

6.4.1 Testovací scénář

Smažte graf z předchozího úkolu.

Následující nedeterministický konečný automat chcete zminimalizovat (zdeterminizovat, odstranit zbytečné a nedosažitelné stavy, zminimalizovat). U minimalizace Vás zajímají i jednotlivé kroky (podrobný výstup). Výsledek chcete zobrazit jako tabulku a stavový diagram, který budete exportovat do `svg` formátu. Podle tohoto zadání vytvořte graf a vyhodnoťte ho.

Graf poté exportujte.

Importujte graf, smažte původní z plátna a znovu ho vyhodnoťte.

	δ	a	b
$\leftrightarrow$	A	A, C	B
	B	B, D	
$\leftarrow$	C		
$\rightarrow$	D	A	C, D

6.4.2 Očekávaný výsledek

Graf bude smazán z plátna. Vyhodnocení nového grafu proběhne bez chybových hlášek, v případě boxu `Minimilize` bude možné zobrazit podrobný výstup. Výstup bude zobrazen jako tabulka a export do `svg` formátu proběhne bez problémů. Po následném exportu a importu se graf opět vyhodnotí bez chybových hlášek.

6.4.3 Výsledek

Hned při prvním kroku se ani v jednom z prohlížečů nepodařilo smazat importovaný a vyhodnocený graf. Bylo zjištěno, že nijak neupravovaný importovaný graf nelze smazat pomocí akce „Clear canvas“. Nejdříve bylo nutné graf poupravit a pak již akce proběhla. Samotné vyhodnocení vyšlo bez problémů. Jedinou drobnou chybou je nedostatečná velikost dialogu pro širší tabulky u podrobného výstupu. Import, export a znovuvyhodnocení proběhlo dle očekávání.

Všichni ze zúčastněných u zadání vstupu využili formulář pro tabulku. Formulář byl uživateli velmi dobře přijat. Tester č. 1 měl námitky k chybějícímu tlačítku pro přidání řádku. Nelíbí se mu, že musí stisknout tlačítko Enter, aby se dostal na konec řádku i když třeba buňky v řádku nevyplňuje. Podobný názor zopakoval i tester č. 3. Tester č. 2 také zmiňoval přidání tlačítka pro přidání řádku, navrhoval také použít klávesu Delete spíše pro mazání řádku a pro sloupce přidat tlačítko pro smazání.

6.5 Testovací případ 3

Tento scénář ověřuje orientaci uživatele a správné vyhodnocení grafu při vytvoření a použití vlastního algoritmového vrcholu.

6.5.1 Testovací scénář

Předchozí graf upravte tak, že místo jednotlivých tří algoritmů pro minimalizaci nedeterministického konečného automatu využijete vlastní vytvořenou funkci z těchto tří.

Po vytvoření grafu pokračujte s vyhodnocením.

Dále přidejte ještě jeden graf na plátno, opět využijte svůj vlastní algoritmový vrchol a jako vstup vygenerujte náhodný konečný automat. K tomu využijte algoritmus `RandomAutomatonFactory`.

Opět vyhodnoťte.

6.5.2 Očekávaný výsledek

Výsledek by měl být kromě chybějícího podrobného výstupu stejný. Vyhodnocení vlastního algoritmového vrcholu proběhne bez chybových hlášek. V případě použití náhodného vytvoření

automatu, záleží jakou abecedu zvolí uživatel. Pokud by například zvolil pro reprezentaci stavů či abecedy čísla, měl by na to být ve formuláři upozorněn.

6.5.3 Výsledek

Vyhodnocení dopadlo dle očekávání stejně jako v předchozím případě. Druhý graf se také v pořádku vyhodnotil.

Všem zúčastněným bylo třeba nejprve ukázat syntaxi pro zadání vstupu, jelikož nedokázali správně zadat tělo funkce. U použití algoritmu `RandomAutomatonFactory` nebylo testerům jasné jak zadat datový typ `ext::set`.

6.6 Testovací případ 4

Testován je formulář pro gramatiku a výstup gramatiky se správným vyhodnocením. V tomto případě je také sledováno, zda tester pro sekvenci algoritmů nevyužije vlastní algoritmový vrchol.

6.6.1 Testovací scénář

Předchozí graf smažte celý.

V následující bezkontextové gramatice odstraňte zbytečné symboly, ε-pravidla a jednoduchá pravidla. Zadaná gramatika: $G = (\{S, A, B, C, D\}, \{a, b, c\}, P, S)$, kde

$$\begin{aligned} P = \{&S \to aSbA \mid \varepsilon, \\ &A \to aBbA \mid bCB \mid CD, \\ &B \to bbBa \mid aS, \\ &C \to aAaA \mid \varepsilon, \\ &D \to SC \mid aABb\} \end{aligned}$$

Graf vyhodnoťte.

6.6.2 Očekávaný výsledek

Vyhodnocení proběhne bez chybové hlášky, ve výstupním dialogu bude gramatika vizualizována. Pokud uživatel využije vytvoření vlastního algoritmového vrcholu, pak se také vyhodnotí bez potíží.

6.6.3 Výsledek

Vyhodnocení opět proběhlo bez problémů.

Testeři opět využili zadání vstupu ve formě gramatiky. První z testerů navrhoval ještě větší zjednodušení neterminálů a terminálů, například vše z levé stany pravidel považovat za neterminály. Tento přístup ovšem není možný pro všechny typy gramatiky. Po diskuzi, zda by spíše preferoval textové pole pro zadání obou abeced terminálů a neterminálů, uznal stávající způsob za nejvhodnější. Jedinou výtkou stále zůstalo zadávání typu gramatiky. Nikdo ze zúčastněných nevyužil vlastní algoritmový vrchol.

6.7 Testovací případ 5

Testována je především orientace uživatele na webovém rozhraní. Dále je testována funkčnost filtrování. Podle volby uživatele jsou také testovány buď parametry šablonovaných algoritmů nebo parsovaný vstup. Především u tohoto případu je sledováno chování uživatele během plnění úkolu. Zároveň je testován export a import parsovaného vstupu.

6.7.1 Testovací scénář

Předchozí graf smažte celý.

Následující regulární výrazy budou vstupy pro Vámi vybrané algoritmy.

$$(a^*\emptyset^*c + b^*a\varepsilon^*)^*(b + \emptyset + a\emptyset)^*$$

$$(a + b)^*ab(a + b)^*$$

Algoritmy vybírejte libovolně po vyhledání všech algoritmů, které přijímají jako vstupní parametr regulární výraz.

Graf vyhodnoťte.

Graf exportuje, importujte a vyhodnoťte znovu.

6.7.2 Očekávaný výsledek

Vyhodnocení grafu proběhne v pořádku. Uživatel dokáže dohledat jakým způsobem zadat speciální znaky regulárního výrazu, dále dokáže využít nové rozšířené filtrování a správně samostatně napojit veškeré vstupy a výstupy. Export, import a znovuvyhodnocení proběhne v pořádku.

6.7.3 Výsledek

Vyhodnocení proběhlo bez potíží.

Všichni testeři zvládli ihned najít místo pro zadání regulárních výrazů. K syntaxi regulárního výrazu bylo od jednoho účastníka navrženo vytvořit podobnou nápovědu jako u nových formulářů alespoň pro základní speciální znaky vyskytující se v regulárních výrazech.

Testeři dokázali využít vyhledávání pomocí parametrů, ale uvítali by checkboxy s možnostmi hned pod textovým polem pro vyhledávání. Napojení algoritmů se vstupy i výstupy zvládli bez potíží.

6.8 Dotazník

1. Byla pro Vás orientace v nabídce vstupních, výstupních a algoritmových vrcholů snadná?
2. Jak se Vám ovládal formulář pro gramatiku? Jak byste si představoval/a zadání gramatiky pomocí pravidel? Bylo Vám jasné v jakém formátu je třeba gramatiku zadat?
3. Jak jste byl/a spokojený/á se zadáním automatu pomocí tabulky? Zdržoval Vás některý krok? Jak byste si představoval/a zadání tabulky?

4. Bylo pro vás snadné v průběhu plnění úkolů nalézt veškeré informace o algoritmech, které jste potřeboval/a?
5. Jak hodnotíte nové vyhledávání dle různých parametrů?
6. Jak hodnotíte výběr algoritmů, respektive jednotlivých přetížení? Orientuje se Vám v nabízených přetížených dobře, věděl/a jste, jaké datové typy máte jak napojit?
7. Bylo Vám jasné, jakým způsobem vytvořit vlastní algoritmový vrchol? Měl/a jste potíže se syntaxí?
8. Jaké jsou podle Vás silné a slabé stránky webového rozhraní?

6.9 Shrnutí výsledku

Průběh testování splnil všechny naplánované požadavky testovacího plánu. Pozitivním výstupem z uživatelského testování je kompatibilita webového rozhraní s třemi použitými prohlížeči. Vyhodnocení grafu probíhalo s očekávaným výsledkem.

Díky správně zvoleným testovacím scénářům bylo odhaleno mnoho drobných chyb. Značná část se vyskytovala v oblastech webového rozhraní, kam nové změny nezasahovaly – problematický je z funkčního i uživatelského hlediska především tzv. move mode, tedy mód během kterého lze posouvat plátno. Následující chyby byly nalezeny během testování a nápady na zlepšení získány jako zpětná vazba od testerů:

- eliminovat volbu typu gramatiky ve formuláři pro gramatiku
- pro posun na plátně je nutné se přepnout do „Move mode“, který je nutné aktivovat stisknutím tlačítka či klávesovou zkratkou, posun by měl jít mimo graf bez přepnutí do módu
- u levého panelu s nabídkou by měla jít zmenšit jeho šířka
- ze seznamu algoritmů by měl jít algoritmus přetáhnout na plátno (aktuálně se jedná o kliknutí na algoritmus a na plátno)
- spojit všechny výstupní vrcholy do jednoho, sjednotit exporty ze Statemakeru a z Dot výstupního vrcholu
- zmenšit šířku dokumentace algoritmového boxu
- při vybrání vrcholu, který ještě není umístěn na plátno, tzv. ghost vrchol, by měl jít zrušit výběr Escape klávesou nebo pravým tlačítkem myši
- ghost vrchol bylo v jednu chvíli nemožné umístit na plátno, pomohlo obnovit stránku (chybu se bohužel nepodařilo zreprodukovat)
- po importu grafu nefunguje akce „Clear canvas“
- možnost použít `ctrl + c`, `ctrl + v` na vrcholy
- možnost vybrat více vrcholů na plátně pomocí myši pro společný přesun či smazání
- invertovat gesto pro přiblížení plátna, většině testerům připadalo opačně
- po zapnutí módu pro pohyb se v jednu chvíli podařilo vytvořit hranu, při snaze napojení na další vrchol se na konec hrany přidala další hrana (chybu se bohužel nepodařilo zreprodukovat)
- po zapnutí módu pro pohyb je možné vytvořit hranu kliknutím na vstupní napojovací bod vrcholu, avšak hrana vychází z posledního místa, kde byla tvořena, může se tedy jednat o náhodné místo na plátně

- vyhledávací kategorie s checkboxy by měly být zobrazeny pořád pod textovým polem

Z nahlas vyřčených myšlenek testerů při plnění úkolů a z dotazníku bylo zjištěno, že nové změny byly velmi kladně přijaty. Jako význačný posun byl nejčastěji zmiňován jeden vstupní vrchol s dialogem pro všechny vstupní typy, a to i komplexní datové typy. Jeden z testerů prohlásil, že v předchozí verzi webového rozhraní s textovými vstupy by měl značné potíže úkoly splnit. Především byl chválen formulář pro gramatiku, u které byla jako jediný problém zmiňována volba typu gramatiky. U automatu, jak bylo výše zmíněno, je spíše preferováno využití tlačítek než klávesových zkratek. Ty jsou ale i přesto testery označeny jako příjemný způsob zadání vstupu navíc k tlačítkům.

Výstupní vrcholy byly komentovány hůře, většina zúčastněných navrhovala použití pouze jednoho výstupního vrcholu, kde bude možné exportovat výsledek do všech dostupných formátů a také zobrazovat jednotlivé reprezentace. V tomto případě často argumentovali právě sjednoceným vstupním vrcholem pro více typů.

Orientace testerů na webovém rozhraní byla dobrá, čím pozdější testovací scénář nastal, tím rychleji dokázali testeři informace najít. Navrhováno ještě bylo, aby vstupní vrchol dokázal po zadání automatu ve formě tabulky zobrazit obsažený typ, tedy rozlišit například DKA, NKA a další typy a informovat o tom uživatele. Chválena byla dokumentace pro algoritmy, byla jí pouze vyčtena moc velká šířka. Delší dokumentační popisky je třeba od určité velikosti zalomit. Testeři také intuitivně ihned využívali vyhledávání v algoritmech. Jediné co bylo od dvou testerů vyčteno je, že by checkboxy pro volbu místa vyhledávání neměly být schované pod tlačítkem „filter options“, ale měly by být rovnou k dispozici pod vyhledávacím polem.

Seznam algoritmů se sloučenými přetíženími byl také hodnocen velmi kladně. Jeden z testerů zmiňoval, že ho typ ve chvíli, kdy algoritmus vybírá, nezajímá a popover s dokumentací mu naprosto stačí.

U algoritmového vrcholu byla u všech účastníku vyčtena syntaxe. V tomto případě je možné v budoucnu odstínit uživatele od syntaxe aql2 například vytvořením dialogu s dalším plátnem, kde se opět napojí jednotlivé algoritmové vrcholy a následně dojde ke sloučení do vrcholu jednoho.

Jako silné stránky byly označeny především vstupní vrchol, zkrácený seznam algoritmů a možnost využít formuláře pro komplexní datové typy. Jako slabé stránky byla označena syntaxe pro vlastní algoritmový vrchol a nutnost explicitního zvolení módu pro pohyb plátnem.

Závěr

Hlavním cílem práce bylo implementovat nové funkcionality a vyřešit nalezené chyby během používání webového rozhraní knihovny algoritmů ALT. Původní verze webového rozhraní byla nejprve analyzována. Došlo k popisu jeho role při typickém pracovním procesu uživatele, analyzovány byly případy použití, které aktuálně webové rozhraní nabízí. Analýza se také věnuje jednotlivým komponentám grafického uživatelského rozhraní a technologiím, pomocí kterých je uživatelské rozhraní vytvořeno. Analyzovány jsou i jednotlivé podprojekty a je modelována jejich komunikace.

Jednotlivé problémy, které byly řešeny, nejdříve prošly analýzou. Následně byla u každého navržena řešení a nakonec bylo zvolené řešení implementováno. Uživatel nyní k používání webového rozhraní nepotřebuje znalost nativního formátu knihovny ALT. Ta je sice zachována jako možnost vstupu i výstupu, ale aktuálně nabízí formuláře pro nejvyužívanější datové typy.

Seznam algoritmů je nyní zkrácen, nepozbývá ale podrobné informace o přetíženích, které uživatel najde nově i na plátně. Uživatel také nyní ví i s jakou variantou šablonovaného algoritmu na plátně pracuje. Byla vylepšena vizualizace podrobného výstupu. Umožněno je prohledávání informací o algoritmech, což usnadňuje uživatelům orientaci. Také lze nyní přidávat a využívat vlastní algoritmové vrcholy. Eliminovány byly chyby, jako například kontrola typů u přetížení algoritmu.

V rámci uživatelského testování byla získána cenná zpětná vazba. Poprvé došlo k testování ve více prohlížečích. Testována byla nejen funkčnost ale i uživatelská přívětivost. Díky formě testování navíc byla zjištěna zpětná vazba pro budoucí vývoj webového rozhraní.

Jak potvrdilo i uživatelské testování, webové rozhraní knihovny algoritmů je pro studenty atraktivním nástrojem při studiu předmětů z oblasti formálních jazyků. Do budoucna lze webové rozhraní rozšířit o další editory pro ostatní komplexní datové typy či interaktivní nástroje pro práci s regulárními výrazy. Z hlediska vývoje je také možným dalším krokem blíže se zaměřit na optimalizaci prezentační vrstvy, například optimalizovat vykreslování komponent. Mnoho cenných nápadů pro pokračování ve zlepšování uživatelské přívětivosti webového rozhraní nabízí výsledky provedeného testování.

Bibliografie

1. VRÁNA, Michael. *Knihovna algoritmů ALT - webové rozhraní*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.
2. KNOP, Dušan et al. *Souvislost, DFS, stromy, orientované grafy* [online] [cit. 2022-04-16]. Dostupné z: `https://courses.fit.cvut.cz/BI-AG1/media/lectures/bi-ag1-p2-handout.pdf`. Soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém USB disku.
3. ŠESTÁKOVÁ, Eliška. *Automaty a gramatiky: sbírka řešených příkladů*. 1. vydání. Praha: České vysoké učení technické, 2017. ISBN 978-80-01-06306-4.
4. GRUNE, Dick; JACOBS, Ceriel J. H. *Parsing Techniques: A Practical Guide* [online]. New York, NY: Springer New York, 2008 [cit. 2022-04-17]. ISBN 978-0-387-68954-8. Dostupné z DOI: `10.1007/978-0-387-68954-8_1`. Překlad autorky.
5. *Algorithms Library Toolkit* [online]. 2022-03-27 [cit. 2022-03-28]. Dostupné z: `https://alt.fit.cvut.cz/`.
6. SVOBODA, Petr. *Webový editor konečných automatů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.
7. BASSETT, Lindsay. *Introduction to JavaScript object notation: a to-the-point guide to JSON*. O'Reilly Media, Inc., 2015. ISBN 978-1-491-92948-3.
8. GANSNER, Emden; KOUTSOFIOS, Eleftherios; NORTH, Stephen. *Drawing graphs with dot* [online]. Technical report, AT&T Research, 2006 [cit. 2022-04-18]. Dostupné z: `https://labs.onb.ac.at/gitlab/wobweger/h_howto/-/raw/8e8aed73e99899f2f2447098343e83cfd6481bba/0_gen/x_gv/dotguide.pdf`.
9. DUCE, David; HERMAN, Ivan; HOPGOOD, Bob. Web 2D Graphics File Formats. *Computer Graphics Forum* [online]. 2002, roč. 21, č. 1, s. 43–64 [cit. 2022-04-18]. Dostupné z DOI: `https://doi.org/10.1111/1467-8659.00565`.
10. *GitLab, Product Direction - Monorepos* [online] [cit. 2022-04-20]. Dostupné z: `https://about.gitlab.com/direction/monorepos/`.
11. *About npm* [online] [cit. 2022-04-20]. Dostupné z: `https://www.npmjs.com/about`.
12. *npm docs, About npm* [online] [cit. 2022-04-19]. Dostupné z: `https://docs.npmjs.com/about-npm`.
13. WOLFF, Ivo G. d.; VANE, Vilic; JANSEN, Remo H. *TypeScript: Modern JavaScript Development*. Packt Publishing, 2016. ISBN 1787289087;9781787289086;
14. *React, What si React?* [Online] [cit. 2022-04-19]. Dostupné z: `https://reactjs.org/`.

15. *React Component State* [online] [cit. 2022-04-19]. Dostupné z: https://reactjs.org/docs/faq-state.html.
16. *React* [online] [cit. 2022-04-19]. Dostupné z: https://reactjs.org/tutorial/tutorial.html#setup-for-the-tutorial.
17. BUGL, Daniel. *Learn React Hooks*. Packt Publishing, 2019. ISBN 1838641440;9781838641443;
18. ELROM, Elad. *React and libraries: your complete guide to the React ecosystem*. New York: Apress, 2021. ISBN 9781484266960;148426696X;
19. *Material-UI Installation* [online] [cit. 2022-04-19]. Dostupné z: https://v4.mui.com/getting-started/installation/.
20. *Material Design Web* [online] [cit. 2022-04-19]. Dostupné z: https://material.io/develop/web.
21. *Material Design Components* [online] [cit. 2022-04-19]. Dostupné z: https://material.io/design/introduction#components.
22. HEJRAL, Martin. *Průvodce SVG* [online] [cit. 2022-04-25]. Dostupné z: https://www.interval.cz/clanky/pruvodce-svg/.
23. MOSTAFAEI, Arvin. *Everything about Redux* [online] [cit. 2022-04-25]. Dostupné z: https://medium.com/codex/everything-about-redux-ccfe3c93ec85.
24. *Pipe and Filter Architecture* [online] [cit. 2022-05-04]. Dostupné z: https://syedhasan010.medium.com/pipe-and-filter-architecture-bd7babdb908.
25. *Ktor* [online] [cit. 2022-04-25]. Dostupné z: https://ktor.io/.
26. *MDN web docs <foreignObject>* [online] [cit. 2022-04-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/SVG/Element/foreignObject.
27. *Material-UI Popover* [online] [cit. 2022-04-18]. Dostupné z: https://v4.mui.com/components/popover/.
28. *MDN web docs dominant-baseline* [online] [cit. 2022-04-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/dominant-baseline.
29. DAVIS, Adam Nathaniel. *Synchronous State With React Hooks* [online]. DEV Community [cit. 2022-04-19]. Dostupné z: https://dev.to/bytebodger/synchronous-state-with-react-hooks-1k4f.
30. MORAN, Kate. *Usability testing 101* [online]. Nielsen Norman Group, 2019-12 [cit. 2022-03-28]. Dostupné z: https://www.nngroup.com/articles/usability-testing-101/.

Obsah přiloženého média

- `src` — zdrojové kódy
  - `webui` — zdrojové kódy pro webui
  - `worker` — zdrojové kódy pro worker
  - `server` — zdrojové kódy pro server
- `text` — text práce
  - `thesis.pdf` — text práce ve formátu PDF
  - `thesis` — zdrojová forma práce ve formátu LaTeX
- `sources` — literární zdroje