FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Rozpoznání konečného automatu z obrazu
Student:	Lukáš Horák
Vedoucí:	Ing. Jan Trávníček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2021/22

Pokyny pro vypracování

Nastudujte znázornění konečného automatu ve formě diagramu.
Nastudujte možnosti strojového rozpoznání prvků automatu v obrázku.
Navrhněte vhodný přístup převodu automatu ve formě diagramu z obrázku do strojově zpracovávatelného formátu.
Implementujte prototyp rozpoznávání s využitím knihovny Qt.
Otestujte implementaci na obrázích automatů, které dodá vedoucí práce.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 12. října 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Rozpoznání konečného automatu z obrazu

Lukáš Horák

Katedra . . . (Teoretické informatiky)
Vedoucí práce: Ing. Jan Trávníček, Ph.D.

7. ledna 2021

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Ing. Jan Trávníčekovi, Ph.D. za trpělivost a nápady při realizaci.

Prohlášení

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

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

V Praze dne 7. ledna 2021 .....

Odkaz na tuto práci

Horák, Lukáš. *Rozpoznání konečného automatu z obrazu.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato bakalářská práce se zabývá strojovým rozpoznáním diagramů konečných automatů z obrazu. Samotný obraz bude mít omezení, že na snímku bude jediný diagram automatu bez objektů v pozadí. Tento problém je řešen pomocí veřejné knihovny OpenCV, která zprostředkovává funkce pro rozpoznání tvarů a jiných metod rozpoznávání obrazu. Tato práce dále využívá knihovny Tesseract OCR, která zprostředkovává metody pro klasifikaci textu z obrazu. Pro zprostředkování grafického uživatelského rozhraní bude program využívat veřejnou knihovnu Qt.

Klíčová slova rozpoznaní obrazu, OpenCV, program, Qt knihovna, C++, konečný automat

Abstract

This Thesis is centered around image detection of finite automata. The images will be constrained to only images with one finite automata diagram per picture without any background objects. To reach this effect we use open-source libraries OpenCV, which detects objects in images and offers methods for general image detection. We also use Tesseract OCR which handles text classification from images. For graphical user interface the program will use Qt open source framework.

Keywords image recognition, OpenCV, program, Qt library, C++, finite automata

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešním světě je automatizace cílovým prvkem většiny manualních činností mezi které patří mimo jiné i problematika strojového rozpoznánvání obrazu, zvuku a jiných činností nebo objektů. Strojové rozpoznávání je nástroj pro řešení široké škály problémů, u kterých může zjednodušit jejich automatizaci a řešitelnost. Pro tuto problematiku vzniklo mnoho různých nástrojů pro práci s obrazem a následné rozpoznání jednotlivých prvků a vazeb. Tato práce se zabývá analýzou, návrhem a implentací technologie strojového rozpoznávání obrazu ku prospěchu automatizace převodu obrazu konečných automatů do strojově čitelného formátu.

Automatizace tohoto problému je nejenom určená pro výuku, kde se dá použít k rychlému ověřování výsledků diagramů konečných automatů, ale také může pomoci jako mezistupeň pro práci s konečnými automaty. Konečný automat se může ve formě diagramu převést do strojové podoby a poté poslat dále ke zpracovaní, minimalizaci a jiné. Konečný automat je jednoduchý výpočetní model, který přijímá regulární jazyky. Ačkoli je konečný automat jednoduchý model jeho využití je nezměrné např. lexikální analyzátor v překladačích, jednoduché vyhodnocovací modely a spousta dalších.

Kapitola 1

Cíl práce

Cílem rešeršní části práce je seznámení se základnímy principy znázornění Konečných automatů ve formě diagramu, analýza problematiky strojového rozpoznaní diagramů konečných automatů a jeho prvků, a také analýza knihovny určené pro rozpoznaní geometrických tvarů a psaného textu z obrazu. Následně zjištění integrace knihovny do formy UI aplikace pomocí frameworku Qt.

Cílem praktické části prace je návrh a implementace aplikacex, která převede diagram konečného automatu z obrazu do strojově zpracovatelného formátu. Tato aplikace bude obsahovat i jednoduché uživatelské rozhraní, které bude vytvořeno za pomoci Qt frameworku.

Kapitola 2

Analýza a návrh

2.1 Základní pojmy

Seznámení se základnímy pojmy.

- *abeceda* - konečná množina *symbolů* (značeno Σ)
- *řetězec* nad abecedou - konečná posloupnout symbolů abecedy
- prázdná posloupnost = *prázdný* řetězec = ϵ
- Σ^* - množina všech řetězců nad Σ
- Σ^+ - množina všech neprázdných řetězců nad Σ
- $\Sigma^* = \Sigma^+ \cup \epsilon$
- operace *zřetězení* (značeno .)
  - $\forall x, y \in \Sigma^*$, připojením řetězce y za řetězec x vznikne řetězec $x.y$ (zkráceno na xy)
  - je asociativní, t.j. $\forall x, y, z \in \Sigma^* : (xy)z = x(yz)$
  - není komutativní, t.j. $\exists x, y \in \Sigma^* : xy \neq yx$
  - ϵ se chová vzhledem k operaci zřetězení jako neutrální prvek: $x\epsilon = \epsilon x = x$
- délka řetezce x: $|x|$

[1]

2.2 Jazyk

2.2.1 Formální jazyk

Formální jazyk L nad $\Sigma : L \subseteq \Sigma^*$ množina řetězců.
operace:

- množinové operace: *sjednocení, průnik, rozdíl*
- *komplement* (doplněk) jazyka L_1: $\overline{L_1} = \Sigma^* \setminus L_1$
  $(\overline{L_1} \cup L_1 = \Sigma^*, L_1 \cap \overline{L_1} = \emptyset)$.
- *součin* (zřetězení) jazyků: $L = L_1.L_2 = xy : x \subset L_1, y \subset L_2$ (L je definován nad abecedou $\Sigma = \Sigma_1 \cup \Sigma_2$)
- n-tá mocnina jazyka L: $L^n = L.L^{n-1}, L^0 = \epsilon$.
  iterace (*Kleene star*) L^* jazyka L: $L^* = \bigcup_{n=0}^{\infty} L^n$

[1]

2.2.2 Regulární jazyk

Jazyk $L \subset \Sigma^*$ je nazván konečný, pokud $\exists n \in \mathbb{N}$: $|L| < n$
Jazyk $L \subset \Sigma$ je regulární když:

- $L = \emptyset$
- $L = a$, $\forall a \in \Sigma$
- pokud jsou jazyky A, B regulární, pro operace:
  - sjednocení
  - součin
  - iterace

  jsou výsledky také regulární

Nejmenší třída jazyků, která obsahuje všechny konečné jazyky a jazyky vzniklé z konečných jazyků pomocí operaci

- sjednocení
- součin
- iterace
- doplněk

je množína všech regulárních jazyků.
[1]

2.3 Gramatika

Gramatika se skládá z množiny pravidel, pomocí kterých může být každé slovo předepsaným způsobem vygenerováno z předem daného počátečního symbolu. Generování probíhá tak, že vezmeme počáteční symbol, na něj aplikujeme kterékoli z pravidel, na získaný řetězec opět aplikujeme kterékoli z pravidel atd., dokud nevygenerujeme požadované slovo. Pokud je pro každé slovo nejvýše jeden postup generování, gramatika je jednoznačná.

2.3.1 Definice gramatiky

Gramatika je čtvečice $G = (N, \Sigma, P, S)$, kde

- N je konečná množina neterminálních symbolů.
- Σ je konečná množina terminálních symbolů ($\Sigma \cap N = \emptyset$).
- P je množina (přepisovacích) pravidel. Je to konečná podmnožina množiny $(N \cup \Sigma)^*.N.(N \cup \Sigma)^* \times (N \cup \Sigma)^*$, (element (α, β)zP zapíšeme $\alpha \to \beta$ a nazveme pravidlo.)
- $S \in N$ je počáteční symbol gramatiky

[1]

2.3.2 Klasifikace gramatik

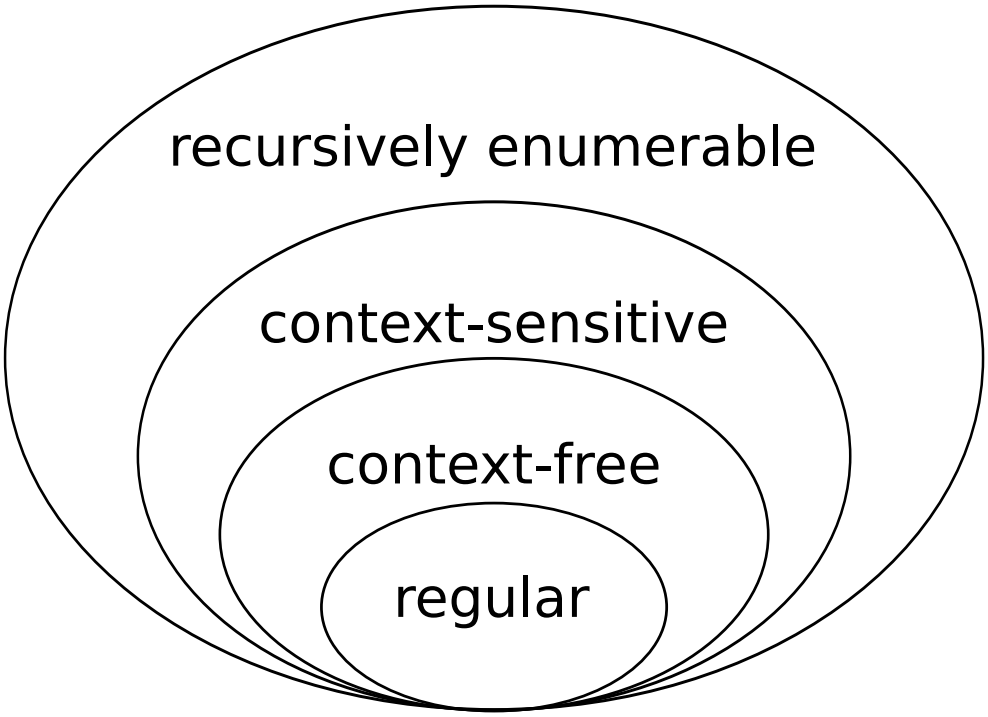
gramatiky typu 0 (neomezené gramatiky), jetliže odpovídá obecné definici gramatiky

gramatiky typu 1 (kontextové gramatiky), jestliže každé pravidlo z P má tvar $\gamma A\delta \to \gamma\alpha\delta$, kde $\gamma, \delta \in (N \cup \Sigma)^*, \alpha \in (N \cup \Sigma)^+, A \in N$, nebo tvar $S \to \epsilon$ v případě, že S se nevystkytuje na pravé straně žádného pravidla

gramatiky typu 2 (bezkontextové gramatiky), jestliže každé pravidlo má tvar $A \to \alpha$, kde $A \in N, alpha \in (N \cup \Sigma)^*$

gramatiky typu 3 (regulární gramatiky), jestliže každé pravidlo má tvar $A \to aB$ nebo $A \to a$, kde $A, B \in N, a \in \Sigma$, nebo tvar $S \to \epsilon$ v případě, že S se nevyskytuje na pravé straně žádného pravidla

[1]



Obrázek 2.1: Chomského hierarchie [2]

2.4 Deterministický Konečný automat

Konečný automat je teoretický výpočetní model popisující velice jednoduchý systém, který může nabývat jednoho z několika stavů. Tento model přechází mezi jednotlivými stavy pomocí symbolů z abecedy přechodů. Tento atutomat se nazívá konečným právě proto, že jeho množina stavů je také konečná. Krom informace o aktuálním stavu nemá automat paměť/informace o jiných stavech, které navšťívil/nenavšťívil. Konečný automat dokáže přijímat pouze regulární jazyky a proto se používají při vyhodnocování regulárních výrazů. Konečný automat se dále dělí na Deterministický konečný automat (dále jen DKA), Nedeterministický konečný automat (dále jen NKA), Mealyho automat, Mooreův automat. V této práci se nadále budeme zabívat DKA.

2.4.1 Definice DKA

DKA M je uspořádaná pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde:

- Q je konečná množina vnitřních stavů
- Σ je je konečná vstupní abeceda
- δ je zobrazení z $Q \times \Sigma$ do Q
- s_0 počáteční stav, $q_0 \in Q$
- F množina koncových stavů automatu, $F \subseteq Q$

[3]

2.4.2 Konfigurace konečného automatu

Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Dvojici $(q, w) \in Q \times \Sigma^*$ nazveme *konfigurací* konečného automatu M. Konfiguraci (q_0, w) nazveme *počáteční konfigurací* konečného automatu M, konfiguraci (q, ϵ), kde $q \in F$, nazveme *koncovou konfigurací* konečného automatu M. [3]

2.4.3 Přechod DKA

Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je DKA. Nechť $\vdash_M$ je relace nad $Q \times \Sigma^*$ (podmnožina $(Q \times \Sigma^*) \times (Q \times \Sigma^*)$) taková, že $(q, w) \vdash_M (p, w')$ právě tehdy, když $w = aw'$ a $\delta(q, a) = p$ pro nějaké $a \in \Sigma$, $w \in \Sigma^*$. Prvek relace $\vdash_M$ nazveme *přechodem* v automatu M.

$\vdash_M^k$ - k-tá mocnina relace $\vdash_M$
$(\alpha_0, \beta_0) \vdash_M^k (\alpha_k, \beta_k)$ pokud $\exists(\alpha_i, \beta_i), 0 < i < k : (\alpha_i, \beta_i) \vdash_M (\alpha_{i+1}, \beta_{i+1}), 0 \leq i < k$
$\vdash_M^+$ - tranzitivní uzávěr relace $\vdash_M$
$\vdash_M^*$ - tranzitivní a reflexivní uzávěr relace $\vdash_M$

[3]

2.4.4 Jazyk přijímaný DKA

Řekneme, že řetězec $w \in \Sigma^*$ je *přijat* konečným deterministickým automatem $M = (Q, \Sigma, \delta, q_0, F)$, jestliže $\exists(q_0, w) \vdash_M^* (q, \epsilon)$ pro nějaké $q \in F$. $L(M) = w : w \in \Sigma^*, \exists q \in F : (q_0, w) \vdash^* (q, \epsilon)$ *je jazyk přijímaný automatem* M.
[3]

2.4.5 Vizualizace konfigurace DKA

Konečné automaty jak DKA tak NKA se dají popsat/vizualizovat mnoho způsoby, mezi základní způsoby zobrazení automatů patří zobrazení tabulkou, diagramem, regulárním výrazem, regulární gramatikou a jiné. Následující příklady ukazují zápis stejného DKA jen jinou formou, kde:

$Q = (S_0, S_1, S_2)$.

$\Sigma = (0, 1)$.

$q_0 = S_0$.

$F = S_0$.

δ viz jednotlivá zobrazení.

2.4.5.1 Zobrazení tabulkou

Konečné automaty se dají rychle a jednoduše zapsat pomocí tabulky, která dokáže rychle, i pro člověka, popsat konečný automat, Její nevýhodou je možná nepřehlednost a nutnost pro každý stav (řádek tabulky) popsat všechny symboly (sloupec tabulky), i když daný stav nepřijímá všechny symboly z abecedy.

Tabulka 2.1: příklad DKA tabulka

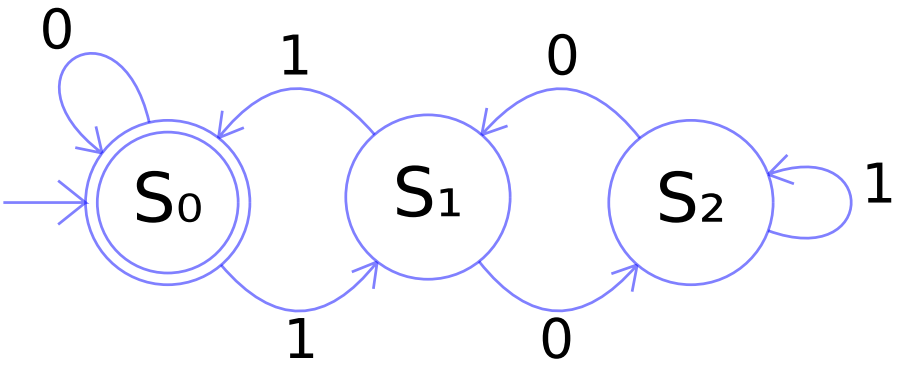
Stav	**0**	**1**
$\boldsymbol{S_0}$	S_0	S_1
$\boldsymbol{S_1}$	S_2	S_0
$\boldsymbol{S_2}$	S_1	S_2

Způsob procházení dané tabulky můžeme simulovat řetězcem **101**. Na začátku jsme ve stavu $\boldsymbol{S_0}$ jelikož $\boldsymbol{q_0 = S_0}$ (je počátečním stavem). První symbol řezězce je **1**. Na řádce se stavem $\boldsymbol{S_0}$ a sloupci se symbolem **1**, přejdeme do stavu $\boldsymbol{S_1}$. Další symbol v řetězci je **0**, který ve stavu $\boldsymbol{S_1}$ přechází do stavu $\boldsymbol{S_2}$. jako poslední symbol máme **1**, ten má přechod do stavu $\boldsymbol{S_2}$ a tedy ve stavu zustáváme. Výsledný stav $\boldsymbol{S_2}$ není koncovým stavem a proto řetězec **101** nepatří do jazyka příjímaného tímto automatem.

2.4.5.2 Zobrazení diagramem

Mezi další často používaná zobrazení patří diagram. Jeho výhodou je snadná přehlednost a jednodušší pochopení struktury zobrazovaného automatu. Mezi jeho výhody patří intuitivní čtení, rychlé orientování pro malý počet stavů/přechodů a narozdíl od tabulky není nutné vypisovat všechny možné kombinace symbolů a stavů i když přechody mezi nimi nemusejí existovat. Jedna z hlavních

nevýhod tohoto typu zobrazení diagramu je prostor nutný pro jeho sestrojení a nepřehlednost pro více stavů/přechodů. Jakožto přechodové funkce nám slouží orientované `hrany` (šipky mezi stavy), které nám udávají směr přechodů a přechodový symbol umístěný v okolí přechodu nás informuje, který symbol je nutný pro přechod. Diagram jakožto nástroj pro snadné čtení se může lišit v mnoha směrech. jako ukázka byl vybrán 2.2.



Obrázek 2.2: Příklad DKA diagram [4]

V diagramu 2.2 je znázorněn počáteční stav $\boldsymbol{S_0}$ příchozí hranou a koncový stav znázorněn dojitým kruhem. Náš aktuální stav je $\boldsymbol{S_0}$, podle odchozích hran vidíme, že symbol **0** náš stav nezmění, zatímco symbol **1** změní náš stav na $\boldsymbol{S_1}$.

2.5 Zpracovnání obrazu

Zpracování obrazu je proces zkládající se z několika kroků. Prvním krokem je předzpracování obrazu. Cílem předzpracování by měla být úprava obrazu za účelem jednoduší práce v dalších krocích. Po předzpracování je užitečným krokem segmentace - rozdělení obrazu na důležité celky a zaměření na informace, které můžeme využít v pozdějších krocích zpracování obrazu. Po segmentaci můžeme využít získané informace k dosažení našich cílů.

2.5.1 Základní pojmy

2.5.1.1 Pixel

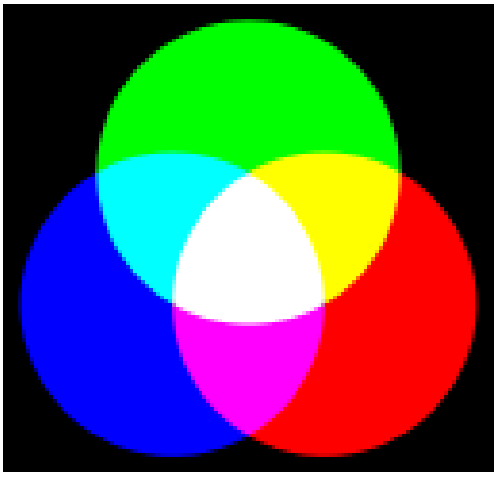
Pixel (zkrácení anglických slov picture element) je nejmenší jednotka digitální rastrové (bitmapové) grafiky. Představuje jeden bod informace v matici. Tento bod je použit jako zdroj informace o barevném stavu. Body v mřížce mohou

využívat různých formátů např. RGB, BGR , CMYK. Pixel můžeme obecně považovat za nejmenší samostatný prvek v obrazu. Každý pixel je možné jednoznačně identifikovat podle jeho souřadnic.

2.5.1.2 Barevné modely

Barevné modely nám udávají informace o formátu a reprezentace dat daného snímku.

- **RGB** - tento model vychází z faktu, že lidské oko obsahuje tři základní druhy buněk citlivých na barvu. Tyto buňky jsou citlivé na vlnové délky, které zhruba odpovídají červené (vlnová délka 630 nm), zelené (530 nm) a modré (450 nm) barvě. Kombinací těchto barev lze získat téměř všechny barvy barevného spektra. Pro model RGB se používá míchání barev (červené, zelené, modré) různých intenzit 2.3. [5]



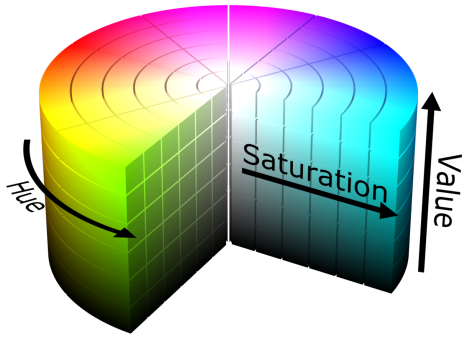
Obrázek 2.3: Grafické zobrazení RGB [6]

- **HSV** - model HSV má tři základní parametry:
  - barevný tón (Hue)
  - sytost (Saturation)
  - jas (Value)

  Kombinací těchto parametrů dostaneme požadované barvy viz. 2.4.

2.5.1.3 Konvoluce

Digitální obraz můžeme popsat jako matici, jejíž každý prvek vyjadřuje jas nějakého okolí příslušného bodu obrazu. Při zpracování obrazu se mohou



Obrázek 2.4: Grafické zobrazení HSV [7]

používat metody diskretní dvourozměrné konvoluce.
Lineární konvoluce je binární operátor definovaný pro funkce $\boldsymbol{f}$ a $\boldsymbol{h}$ na množině $\boldsymbol{M}$ v bodě $\boldsymbol{x}$ vztahem:

$$(\boldsymbol{f} * \boldsymbol{h})(\boldsymbol{x}) = \int_{\boldsymbol{M}} \boldsymbol{f}(\boldsymbol{x} - \boldsymbol{u})(\boldsymbol{hu})\boldsymbol{du}$$

odvozením z lineární konvoluce lze *dvourozměrnou diskrétní konvoluci* vyjádřit jako:

$$\boldsymbol{v}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) * \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{i}=-\boldsymbol{k}}^{\boldsymbol{k}} \sum_{\boldsymbol{j}=-\boldsymbol{k}}^{\boldsymbol{k}} \boldsymbol{f}(\boldsymbol{x} - \boldsymbol{i}, \boldsymbol{y} - \boldsymbol{i})\boldsymbol{h}(\boldsymbol{i}, \boldsymbol{j}).$$

Mějme čtvercovou matici $\boldsymbol{Q}_{(\boldsymbol{2r+1})\times(\boldsymbol{2r+1})} = (\boldsymbol{q_k}, \boldsymbol{l})$, kde $\boldsymbol{r} \in \mathbb{N}$ a matici $\boldsymbol{O}_{\boldsymbol{o1}\times\boldsymbol{o2}} = (\boldsymbol{o}_{\boldsymbol{i},\boldsymbol{j}})$, kde $\boldsymbol{o_1}, \boldsymbol{o_2} >> \boldsymbol{2r+1}$, potom

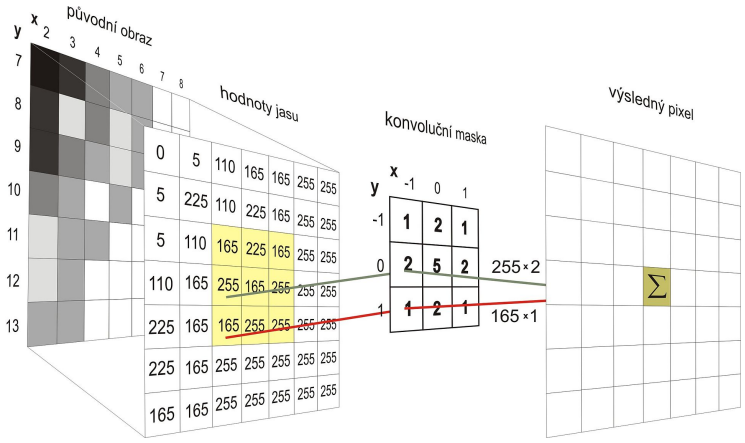
$\boldsymbol{v}(\boldsymbol{i0}, \boldsymbol{j0}) = \boldsymbol{f}(\boldsymbol{i_0}, \boldsymbol{j_0}) * \boldsymbol{Q}(\boldsymbol{i_0}, \boldsymbol{j_0}) = \sum_{\boldsymbol{k}=\boldsymbol{1}}^{\boldsymbol{2r+1}} \sum_{\boldsymbol{l}=\boldsymbol{1}}^{\boldsymbol{2r+1}} \boldsymbol{f}(\boldsymbol{i_0} - \boldsymbol{r} - \boldsymbol{1} + \boldsymbol{k}, \boldsymbol{j_0} - \boldsymbol{r} - \boldsymbol{1} + \boldsymbol{l})\boldsymbol{q_k}, \boldsymbol{l}$,

je dvourozměrná diskrétní konvoluce realizovaná v bodě $\boldsymbol{o}_{\boldsymbol{i_0},\boldsymbol{j_0}}$ původního obrazu $\boldsymbol{O}$ konvoluční maskou $\boldsymbol{Q}$.

$\boldsymbol{f}(\boldsymbol{i_0}, \boldsymbol{j_0})$ reprezentuje původní a $\boldsymbol{v}(\boldsymbol{i_0}, \boldsymbol{j_0})$ výslednou hodnotu jasu v obraze $\boldsymbol{O}$. [8]

2.5.2 Předzpracování obrazu

Důležitou součástí zpracování obrazu je jeho předzpracování. Jedná se o operace, které nám zjednodušší práci s obrazem. Obraz z reálného světa při zachycení vytváří spoustu problémů. Mezi tyto problémy patří např. artefakty - odlesk hladkého povrchu, nevyvážené barvy, špatné nasvícení, jiné úhly fotografie. Těchto defektů se po pořízení snímku těžko zbavujeme. V těchto kapitolách si představíme různé metody pro odstranění artefaktů z obrazu.



Obrázek 2.5: Dvourozměrná diskrétní konvoluce[9]

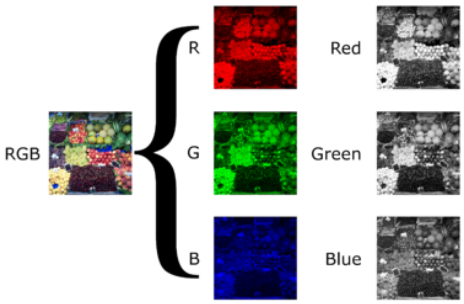
2.5.2.1 Převedení barvy

Základní operace předzpracování je převést snímek na barvu, typ barevného rozložení, které pro nás bude nejvýhodnější. Pro zobrazení diagramu konečného automatu není nutné používat barevný snímek, a proto můžeme použít metodu pro převedení barevného snímku na šedotónový.

Barevný obraz se skládá ze tří barev: červená, zelená, modrá. Šedotónový obraz můžeme získat kombinací hodnot jednotlivých barev.

$$\boldsymbol{Y} \leftarrow \mathbf{0.299} \cdot \boldsymbol{R} + \mathbf{0.587} \cdot \boldsymbol{G} + \mathbf{0.114} \cdot \boldsymbol{B}$$

kde $\boldsymbol{R}$, $\boldsymbol{G}$, $\boldsymbol{B}$ jsou jednotlivé prvky barevného schéma, red (červená), green (zelená) ,yellow (žlutá), $\boldsymbol{Y}$ je výsledný jas šedotónového pixelu viz.2.6.



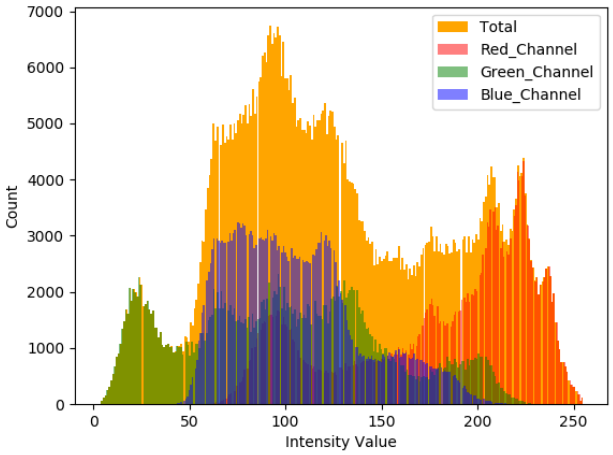
Obrázek 2.6: Rozdělení barev [10]

2.5.2.2 Histogram obrazu

Histogram je graf, který reprezentuje distribuci aproximace kategoricky uspořádaných dat. Histogram se dá vytvořit pomocí metody binning, což je metoda která

vezme data na vstupu a rozdělí je podle intervalů do jednotlivých krabic (anglicky bin). Touto metodou nám vznikne vhodná reprezentace dat, kterou můžeme použít pro vynesení na graf. Tento způsob dále můžeme aplikovat pro náš problém práce s obrazem a jeho barevné odstíny. Histogram, který se používa pro vizualizaci barevných odstínů se nazývá histogram obrázku (Image histogram). Tento histogram nám ukáže distribuci barevných odstínů v obraze, kterou můžeme použít pro další práci. Príklad barevného histogramu 2.7 z obrazu 2.8.

Obrázek 2.7: Obraz pro 2.8 [11]



Obrázek 2.8: Barevný histogram pro 2.7 [11]

2.5.2.3 Vyrovnání histogramu

Jeden z důvodů proč vyrovnávat histogram obrazu je lepší viditelnost tmavých ploch na obrazu, což nám pomůže nejenom lépe vidět zrakově ale usnadní nám to prací se snímkem samotným. Vyrovnávání funguje na principu zvýšení kontrastu v obraze aby došlo k lepší distribuci barev.

Pro diskrétní šedotónový obraz {x}, kde n_i je počet výskytů šedé úrovně i, je pravděpodobnost výskytu pixelu úrovně i v obraze:

$p_x(i) = p(x = i) = \frac{n_i}{n}, 0 \leq i < L$

kde L je celkový počet urovní šedé barvy v obraze (typicky 256), n je celkové množství pixelů v obraze, a kde $p_x(i)$ je výsledná hodnota histogramu pro úroveň pixelu i , normalizována na intervalu $[0, 1]$.

jako další krok definujeme komutativní distribuční funkci pro p_x jako

$cdf_x(i) = \sum_{j=0}^{i} p_x(x = j)$

což je takékomutativné normalizovaný histogram.

S touto funkcí můžeme vytvořit transformaci ve formě $y = T(x)$, která nám vrátí nový obraz y, takový obraz by měl interval hodnot rovný

$cdf_y(i) = iK$

pro nějakou konstantu K. kde transformace se rovná

$cdf_y(y') = cdf_y(T(k)) = cdf_x(k)$

kde k je interval $[0, L)$, kvůli správné transformaci zpět na interval $[0, 1]$, musíme aplikovat

$y' = y * (max\{x\} - min\{x\}) + min\{x\}$

Po aplikaci této transformace dostaneme na výstup vyrovnaný histogram 2.9. Jak můžeme sledovat jedná se o jednoduchou transformaci kde se histogram „roztáhne“ aby pokryl celé spektrum. [12]

2.5.2.4 Rozostření obrazu - konvoluční filtry

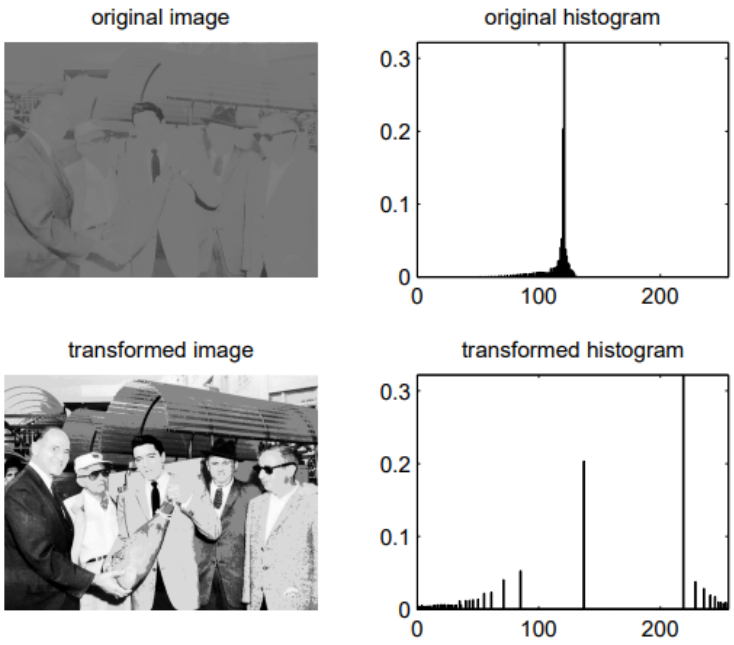
Rozostření obrazu je metoda, která se zbavuje převážně šumu v obraze. Šum se v obraze projevuje zmenou barevných hodnot pixelů. Šum nejčastěji nastává defektem kamery, manipulací s obrazem např. kompresí obrazu. Obraz s částí bez šumu (část uvnitř čtverce v obrazu) a částí se šumem můžeme vidět v obraze 2.10.

Gaussovo rozostření

Gaussovo rozostření je funkce, která konvuluje vstupní matici filtrační maticí např. 2.11, které se skládá vzorcem:

$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e$

kde x je vzdálenost od počátku na ose horizontální, a je standardní deviace σ, tuto funkci použijeme k naplnění konvoluční masky. která je aplikována na originální obraz.



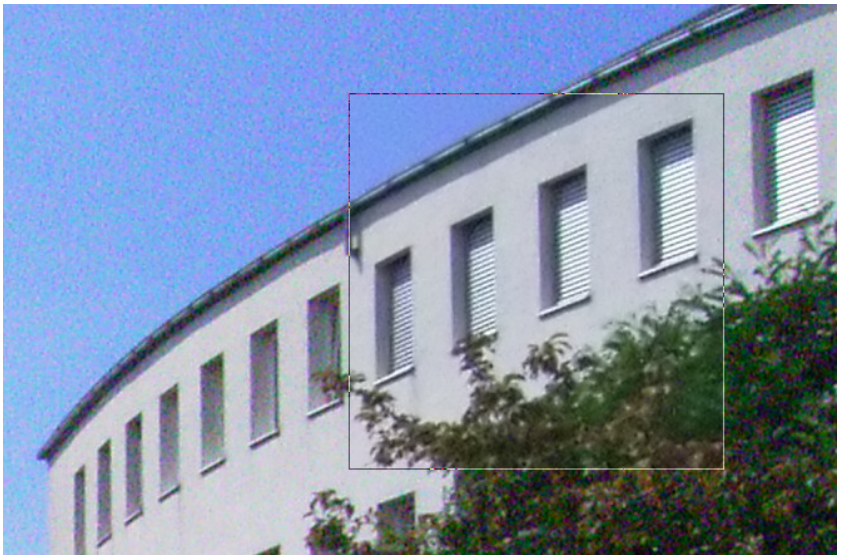
Obrázek 2.9: Příklad vzvážení histogramu [12]

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Obrázek 2.11: Element pro gaussovo rozostření [14]

Průměrové rozostření

Průměrové rozostření používá matice rovnoměrného rozdělení např. 2.12.



Obrázek 2.10: Příklad šumu [13]

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Obrázek 2.12: Element pro průměrné rozostření [14]

Mediánové rozostření

Tato operace rozostření je podobná předchozím operacím, liší se jen v aplikované funkci. Namísto funkce průměru nebo gausse se vybraný pixel sestavý jako medián okolních hodnot. Medián je prostřední hodnota $\boldsymbol{y}$ z nějaké konečné uspořádané množiny $\boldsymbol{M}$, kde pro každé $\boldsymbol{x_n}$, $\boldsymbol{x_{n+1}} \in \boldsymbol{M}$, kde $\boldsymbol{n}$ je pozice v množině $\boldsymbol{M}$, platí že $\boldsymbol{x_n} \leq \boldsymbol{x_{n+1}}$. tedy platí, že $\boldsymbol{y} = \lfloor \boldsymbol{x}_{\frac{N}{2}} \rfloor$, kde $\boldsymbol{N}$ je počet prvků v množině $\boldsymbol{M}$.

2.5.3 Segmentace

Segmentace obrazu jsou metody, které rozdělí obraz podle určitých zvolených chrakteristik. Vetšinou chceme aby tyto charakteristiky popisovaly nejakou

oblast našeho zájmu v obrazu např. segmentace po barevných složkách nebo tvarů.

2.5.3.1 Prahování

Mezi jednu metodu segmentace patří prahování obrazu. Můžeme si to znázornit jako odstranění hodnot z histogramu. Jedná o jednoduchou funkci kde:

$$\boldsymbol{y'} = \begin{cases} \boldsymbol{M} & \text{pokud } \boldsymbol{y} > \boldsymbol{T} \\ \boldsymbol{0} & \text{jinak} \end{cases}$$

kde $\boldsymbol{M}$ je maximální hodnota na kterou se nastaví úroveň pixelu, $\boldsymbol{T}$ je hodnota prahu. Takže pokud intezita úrovně pixelu $\boldsymbol{y}$ je vyšší jak $\boldsymbol{T}$ tak nová hodnota pixelu bude $\boldsymbol{M}$, jinak se nastaví na $\boldsymbol{0}$.

2.5.3.2 Matematická morfologie

Matematická morfologie vychází z vlastností bodových množin. Nejčastěji se aplikuje nad binárnímy obrazy ale dá se zobecnit na šedotónové a barevné obrazy. Morfologické operace se nejčastěji používají pro předzpracování obrazu, zjednodušení tvarů, zdůraznění struktůry. Morfologické operace představují systematický pohyb strukturálního elementu po obrazu a vytvoření výstupního obrazu dle typu operace. Mezi základní typy operace se řadí Dilatace a Eroze. Tyto operace jsou duální. Operace $\boldsymbol{A}$ a $\boldsymbol{B}$ jsou duální, právě když $\boldsymbol{A}(\boldsymbol{X}^{\boldsymbol{C}}) = [\boldsymbol{B}(\boldsymbol{X})]^{\boldsymbol{C}}$, kde $\boldsymbol{X}^{\boldsymbol{C}} = \boldsymbol{E}/\boldsymbol{X}$ označuje doplněk množiny $\boldsymbol{X}$ vzhledem k množině $\boldsymbol{E}$. Pro morfologické operace se používá strukturální element, který může mít jakoukoliv velikost i tvar pokud splňuje, že je menší jak množina obrazu, strukturální element je množina $\boldsymbol{Y} < \boldsymbol{X}$, kde $\boldsymbol{X}$ je množina obrazu. K ukázce operací použijeme obrázek 2.13. [15]

Obrázek 2.13: Ukázkový obrázek pro morfologii [16]

2.5.3.3 Dilatace

Dilatace skládá body dvou množin dohromady pomocí vektorového součtu. Jde o operaci, která rozšíří výstupní obraz o svůj strukturální element. Tato

operace dostane na výstup**1**, pokud alespoň jeden pixel pod maticí strukturálního elementu je **1**.

$\boldsymbol{X} \oplus \boldsymbol{Y} = \bigcup_{\boldsymbol{y} \in \boldsymbol{Y}} \boldsymbol{X_y}$

Obrázek 2.14: Ukázka dilatace [16]

2.5.3.4 Eroze

Eroze je operace opačná od dilatace, oproti dilataci se skládají body dvou množin pomocí vektorového rozdílu. Používá se na ztenčování, zjednodušování struktur objektů. Jedná se o operaci duální k dilataci.

$\boldsymbol{X} \ominus \boldsymbol{Y} = \bigcap_{\boldsymbol{y} \in \boldsymbol{Y}} \boldsymbol{X_{-y}}$

Obrázek 2.15: Ukázka eroze [16]

2.5.4 Zavření a Otevření

Další morfologické operace jsou kombinací dvou předchozích operací. Kombinace eroze a poté dilatace se nazývá otevření. Otevření díky prvně provedené erozy je toto skvělá metoda proti šumu, z obrazu vymizejí objekty menší jak struturální element a poté díky následné dilataci se zbylé objektu slijí zpět. Zavření má opačný účinek a po jeho provedení se slijí „díry“ v objektech, spojí se objekty blízko u sebe a vyhladí obrys.Otevření a zavření jsou duální operace.

Otevření $\boldsymbol{X} \circ \boldsymbol{Y} = (\boldsymbol{X} \ominus \boldsymbol{Y}) \oplus \boldsymbol{Y}$.

Zavření $\boldsymbol{X} \bullet \boldsymbol{Y} = (\boldsymbol{X} \oplus \boldsymbol{Y}) \ominus \boldsymbol{Y}$.

Obrázek 2.16: Ukázka otevření [16]

Obrázek 2.17: Ukázka zavření [16]

2.5.4.1 Sobelův operator

Je operátor používaný (nejen) pro detekci hran. Výsledky, které produkuje tento operator jsou obrazy, které kladou důraz na hrany. Operátor používá dvě matice o velikosti $\mathbf{3} \times \mathbf{3}$. Pro $\boldsymbol{A}, \boldsymbol{G_x}, \boldsymbol{G_y}$, kde $\boldsymbol{A}$ je původní obraz, $\boldsymbol{G_x}$ je první matice a $\boldsymbol{G_y}$ je druhá matice oparátoru, pro které platí:

$$\boldsymbol{G_x} = \begin{bmatrix} \mathbf{+1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{+2} & \mathbf{0} & \mathbf{-2} \\ \mathbf{+1} & \mathbf{0} & \mathbf{-1} \end{bmatrix} \times \boldsymbol{A}$$

$$\boldsymbol{G_y} = \begin{bmatrix} \mathbf{+1} & \mathbf{+2} & \mathbf{+1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{-1} & \mathbf{-2} & \mathbf{-1} \end{bmatrix} \times \boldsymbol{A}$$

kde × značí operaci konvoluce.

Po aplikaci tohoto operátoru můžeme zjistit výslednou sílu gradient hrany jako matici $\boldsymbol{G} = \sqrt{\boldsymbol{G_x^2} + \boldsymbol{G_y^2}}$ a směr $\boldsymbol{\Theta} = \mathbf{arctan}(\frac{\boldsymbol{G_y}}{\boldsymbol{G_x}})$ [17]

2.5.4.2 Canny edge detector

Canny edge detector je robustní metoda pro detekci hran v obraze. Tato metoda dokáže najít hrany v obraze pomocí několika kroků:

Redukce šumu

Šum představuje pro detekci hran závažný problém. Šum v obraze může jednoduše zkreslovat výsledky a proto je důležitým prvním krokem redukce šumu, nejčastěji pomocí gausové filtraci. Důležitým poznatkem s použitím tohoto filtru je zvolení velikosti filtru. Nižší velikost filtrovací matice nám zvýší sesitivitu vůči šumu v algoritmu a naopak příliš velká velikost filtrovací matice nám může zkreslit výsledky zvýšením lokalizačních chyb detekce hran. Hodnoty této matice můžeme doplnit podle:

$\boldsymbol{H_{ij}} = \frac{\mathbf{1}}{\mathbf{2\pi\sigma^2}}\exp(-\frac{\boldsymbol{(i-(k+1))^2}}{\mathbf{2\sigma^2}}); \mathbf{1} \leq \boldsymbol{i, j} \leq \boldsymbol{(2k+1)}$,
kde $\boldsymbol{k}$ velikost matice, $\boldsymbol{H}$ výsledná matice, $\boldsymbol{i, j}$ indexi matice, $\boldsymbol{\sigma}$ směrodatná odchylka.

Nalezení intenzitního gradientu obrazu
Pro získání hodnot síly gradientů jsou použity operátory pro hledání hran např Sobelův operátor. Z tohoto operátoru můžeme zjistit velikost gradientu a směr podle 2.5.4.1.

Non-maximum suppresion - hledá lokace s nejostřejšími hranami

[18]

Obrázek 2.18: Detekce hran [19]

2.5.4.3 Thinning

Thinning je morfologická operace, která se používá pro odstranění pixelů okolo objektů, podobně jako operace eroze. Jedna z nejdůležitějších operací pro nás. Tato operace se dá využít na skeletonizaci obrazu. Skeletonizace znamena že objekty z matice $\boldsymbol{X}$ projdou transformací kde jejich celková struktůra zůstane stejná ale kde kontůra objektu samotná bude minimální podle velikosti struturálního elementu. [20]

2.5.4.4 Hit-and-miss

Jednou z method jak na thinning obrazu je rozšíření hit-and-miss transformace. V teto transformace může strukturální element obsahovat jak **0** tak **1**. Jedná se o jednoduchou transformaci, která se dá vyjádřit jako

$\boldsymbol{thin(X,Y) = X - hit - and - miss(X,Y)}$, kde výsdný obraz je $\boldsymbol{X}$ a $\boldsymbol{Y}$ je strukturální element. Rozdíl mezi maticemi je definován jako $\boldsymbol{X - Y} = \boldsymbol{X} \cap \text{NOT } \boldsymbol{Y}$

Tato metoda nám říká, že strukturální element prochází celý obraz a pokud se jeho matice rovná maticy bodu a jeho okolí středu strukturálního elementu, tak se střed tohoto elementu nastaví na **0**. Element musí obsahovat alespoň jednu **0** aby se projevily změny. Typ elementu mění chování procesu a výsledku. Operace končí pokud po přechodu vstupního obrazu maticí se výsledná matice nezmění. [21]

2.5.4.5 Thinning pomocí eroze a otevření

Druhá možná implementace pro thinning je iterativní aplikování eroze a otevření na obraz. Tato metoda dosahuje výsledku iterativním sjednocením popsané jako

$$\boldsymbol{S_n(A) = (A \ominus nB) - (A \ominus nB) \circ B}$$

kde poslední krok, než obraz eroduje do prázdného výsledku, vypadá takto

$\boldsymbol{S(A) = \bigcup_{n=0} S_n(A)}$

$\boldsymbol{S}$ je výsledný obraz, $\boldsymbol{A}$ je vstupní binární obraz, $\boldsymbol{B}$ je strukturální element, $\boldsymbol{n}$ je počet iterací eroze.

2.5.4.6 Zhang-Suen algoritmus

Tento algoritmus je jednoduchý a rychlý algoritmus pro skeletonizaci binárního obrazu (kde hodnoty pixelů $\boldsymbol{p(i,j) \in \{1,0\}}$).

Konektivita pixelu

Konektivita pixelu udává počet objektů připojených s pixelem. Konektivita se dá určit vzorcem:

$$\boldsymbol{C_n = \sum_{k \in S} N_k - (N_k * N_{k+1} * N_{k+2})},$$

kde $\boldsymbol{N}$ je matice okolí pixelu, $\boldsymbol{N_k}$ je barva pixelu, $\boldsymbol{N_0}$ je středový pixel, $\boldsymbol{N_1}$ je pixel vpravo od středu zkoumaného okolí a zbytek bodů je ve směru hodinových ručiček okolo středu

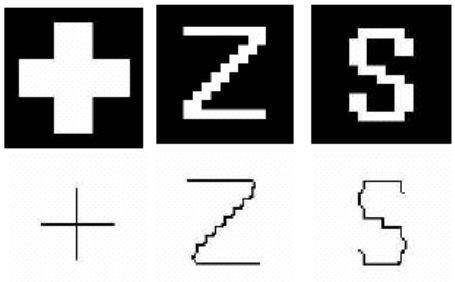
Algoritmus funguje na dvou iteracích, první iterace pixel $\boldsymbol{p(i,j)}$ je vymazán pokud jsou následující podmínky splněny:

- konektivita je jedna
- zkoumaný pixel má alespoň dva černé pixely sousedy ale ne více než šest
- alespoň jeden z pixelů $\boldsymbol{p(i,j+1)}$, $\boldsymbol{p(i-1,j)}$ a $\boldsymbol{p(i,j-1)}$ je bílí
- alespoň jeden z pixelů $\boldsymbol{p(i-1,j)}$, $\boldsymbol{p(i+1,j)}$ a $\boldsymbol{p(i,j-1)}$ je bílí

Ve druhé iteraci jsou podmínky:

- konektivita je jedna
- zkoumaný pixel má alespoň dva černé pixely sousedy ale ne více než šest
- alespoň jeden z pixelů $\boldsymbol{p(i-1,j)}$, $\boldsymbol{p(i,j+1)}$ a $\boldsymbol{p(i+1,j)}$ je bílí
- alespoň jeden z pixelů $\boldsymbol{p(i,j+1)}$, $\boldsymbol{p(i+1,j)}$ a $\boldsymbol{p(i,j-1)}$ je bílí

Pokud jsou splněny podmínky tak na konci iterace se dany pixel vymaže z obrazu. Pokud po konci kterékoliv iterace nebyl vymazán žádný pixel tak algoritmus končí. Příklad algoritmu 2.19. [22]



Obrázek 2.19: Algoritmus Zhang-Suen [23]

2.5.5 Nástroje

Nástroje jsou nedílnou součástí práce na komplikovanějších projektech. Zde můžeme využít předzpracovaných nástrojů pro jednoduší práci a přesnější výsledky. Tyto nástroje nám umožní používat rychlé metody pro zpracování obrazu.

2.5.5.1 Qt

Qt je multiplatformní aplikační framework, který je používán pro vytváření GUI aplikací. Qt je knihovna programovacího jazyka C + + ale existuje i pro jazyky Python, Ruby, C, Perl, Java a jiné. K samotné knivovně Qt je vyvíjen software Qt Creator, který slouží jako IDE centrované přímo pro Qt, a Qt designer, který se používá pro vytváření elementů pomocí jednoduchého uživatelského rozhraní. [24]

2.5.5.2 OpenCV

OpenCV je open-source knihovna poskytující mnoho metod pro zpracování obrazu. Knihovna je multiplatformní a podporuje několik jazyků, mezi nejvíce používané patří C + + a python. V případě C + + se využívá její rychlosti pro real-time strojové zpracování obrazu. OpenCV také podporuje využití neuronových sítí přímo zabudované v jeho knihovně. Opencv podporuje implementace TensorFlow, Torch/PyTorch a Caffe. [25]

2.5.5.3 Tesseract-OCR

Tesseract-OCR (dále jen tesseract) je open-source engine od společnosti Google LLC. Tesseract je engine pro multiplatformní strojové zpracování textu a symbolů. Tesseract je považován za jeden ze tří nejlepších OCR enginů v závisloti na přesnosti na jednotlivých symbolech. Od roku 2006 je tesseract open-source a stále je vyvíjen do dnes. Aktuální verze enginu je 4.1.1 stabilní vydána na konci roku 2019 a verze 5.1.x beta (nestabilní). Ve verzi 2.0 tesseract nepodporoval analýzu struktůry textu a tak nemohl přijímat několika sloupcový text, avšak od verze 3.0 tuto možnost nabízí a jiné, např. pozicionální informace, analýzu rozvržení stránky. V prvních verzích tesseract podporoval pouze anglický jazyk a v pozdějších verzích přídal podporu přes sto jazyků a to i čínštinu, češtinu a mnoho jiných. Jedna z velkých výhod je možnost naučit tesseract vlastní jazyky. Pro správnou detekci textu je nutné aby text byl správně škálován a to alespoň na 20 pixelů $\boldsymbol{x - height}$, což je vzdálenost spodní linie (baseline) po medián (mean line) textu, jako je znázorněno v obrázku 2.20. Rotace, natočení a jiné artefakty musí být odstraněny předem v předzpracování obrazu pro správné výsledky.

Od verze 4.0 podporuje tesseract také model pro strojové učení LSTM, který se od normálních neuronovách sítí, které jsou dopředné liší v tom, že

používá zpětný feedback loop, díky kterému může zpracovávat delší sekvence dat jako jsou videa a zvukové nahrávky. [26]

2.5.5.4 Leptonica

Leptonica [**?**] je open-source knihovna a jedním z balíčků vyžadovaným Tesseract-OCR, tento balíček je hojně využívaný pro podporu metod zpracování obrazu a analýzu. Mezi důležité metody se řadí např. Affiní transformace, morfologie, seedfill (floodfill), maskování a jiné operace pro zjednodušní práce s obrazy. [27]

Obrázek 2.20: Ukázka textové linie

2.5.6 Existující řešení

Nejbližší existující podobné řešení již bylo vypracováno na toto téma, detekce konečných automatů v obraze, na platformu android. Práce měla za úkol detekci stavů, přechodů a klasifikaci textu. Práce byla založena na detekci stavů pomocí metody *hough transform*, kde hlavním problémem detekce stavů byla variabilita předem neznámých velikostí stavů, a proto vznikaly problémy chybné detekce stavů. Detekce přechodů mezi stavy byla realizována pomocí maskování obrazu pro zbavení nedůležitých informací. Poté se detekuje kontůra mezi stavy a vybere se orientace přechodu za pomoci rohového detektoru Harris. Klasifikace textu je pak zpracována pomocí natrénovaného modelu SVM. Vstup této práce se převážně liší v tom, že je vstup získán z kamery mobilního zařízení a nebo jiným způsobem na platformě android, takže se musí dváb větší důraz na předzpracování obrazu, před jeho zpracováním. Výsledný zpracovatelný formát je JSON [28]

Rozdílnost řešení spočívá v použití jiných metod, zatímco v druhé práci byl kladen důraz na detekci stavů pomocí metody Hough transform, v této práci se budeme snažit získat i neprefektní kruhy a klasifikovat je jako stavy pokud budou splňovat určitá kriteria. Tímto přístupem hodlám zvýšit pravděpodobnost správné detekce a klasifikace přechodů. Detekce přechodů je založena na podobné bázi kde se detekují rohy pro zjištění směru. Detekce textu je v této práci řešena pomocí nástroje Tesseract, který by měl nabídnout robustní model pro detekci text z obrazu. Jiná řešení, která se zabívají danou problematikou nebyla nalezena.

Kapitola 3

Realizace

3.1 Použité nástroje

3.1.0.1 Qt

Qt je v aplikaci použit pro rychlé a jednoduché vytvoření GUI elementů jako jsou tlačítko, text box, vykreslení obrazů. Pro vývoj GUI byl použit jak API Qt tak i Qt designer, který usnadnil hlavně pokládání elementů do poziční mřížky, která zpracovává pozice a velikosti elementů, které uchovává. Verze Qt knihovny byla zvolena 5.12.0 jakožto nejnovější stabilní verze v době vzniku projektu. U Qt designeru by verze neměla hrát roli jelikož Qt designer je jen návrhář pro GUI a neměl by hrát roli po zkompilování aplikace.

3.1.0.2 OpenCV

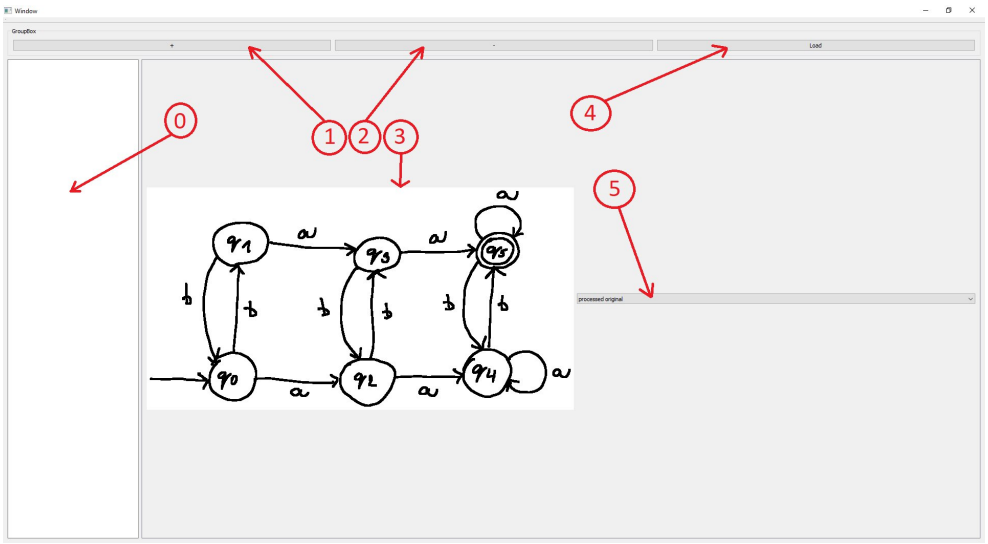
OpenCV je open source knihovna, která zprostředkovává metody pro zpracování obrazu. V práci byla použita verze OpenCV 4.0.1, která byla jako nejnovější *stabilní* verze v době vytvoření aplikace. Všechny verze OpenCV nad verzí 4.0 by měli být kompatabilní s aplikací. Problémy mohou nastat v budoucnu s novější verzí , a nebo pokud se drasticky změní API pro implementované metody v knihovně jako tomu bylo při vydání verze 2.0 v roce 2009.

3.1.0.3 Tesseract-OCR

Hlavní úloha nástroje tesseract je klasifikace textů v obraze, o předzpracování obrazu, segmentaci a detekci se postará OpenCV s jeho širokou škálou metod. Zvolený jazyk pro klasifikaci symbolů je anglický jazyk pro jeho jednoduchost, a naše omezení pro symboly abecedy a symboly stavů, které budou jen ve formátu anglického jazyka, bez speciálních znaků. Zvolená verze tesseract je 4.0, tesseract 4.0 nejnovější stabilní verze.

3.2 Aplikace

3.2.1 Uživatelské rozhraní



Obrázek 3.1: Ukázka GUI

Uživatelské rozhraní je rozvrženo podle obrázku 3.1, kde jednotlivé uživatelské elementy jsou definované takto:

0. Textové pole pro výstup z detekce.

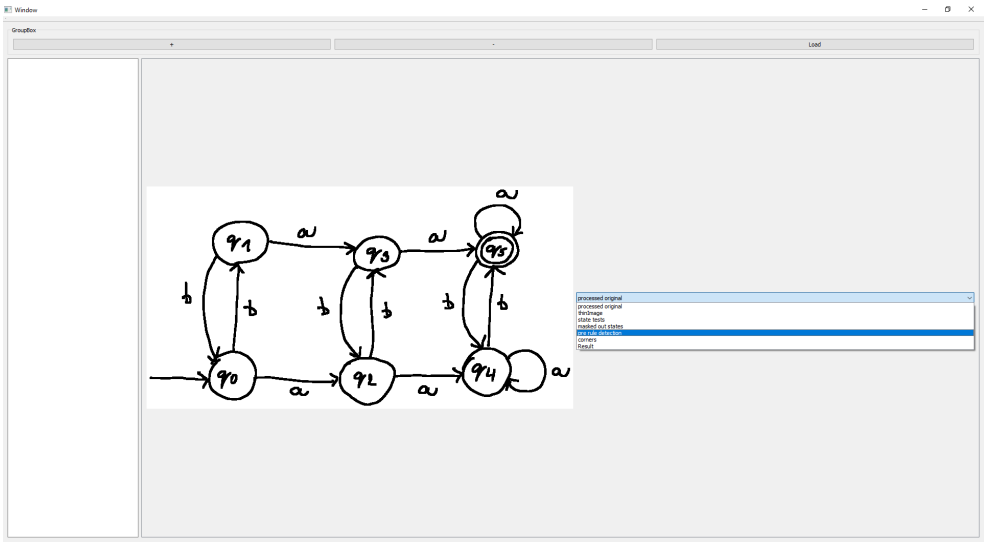
1. Přidání dalšího okna pro obraz. výhodné pro sledování výstupu různých operací na obrazu, viz 3.3.

2. Odebrání okna obrazu. Odebírání postupuje po jednom okně dokud není jejich počet rovný nule.

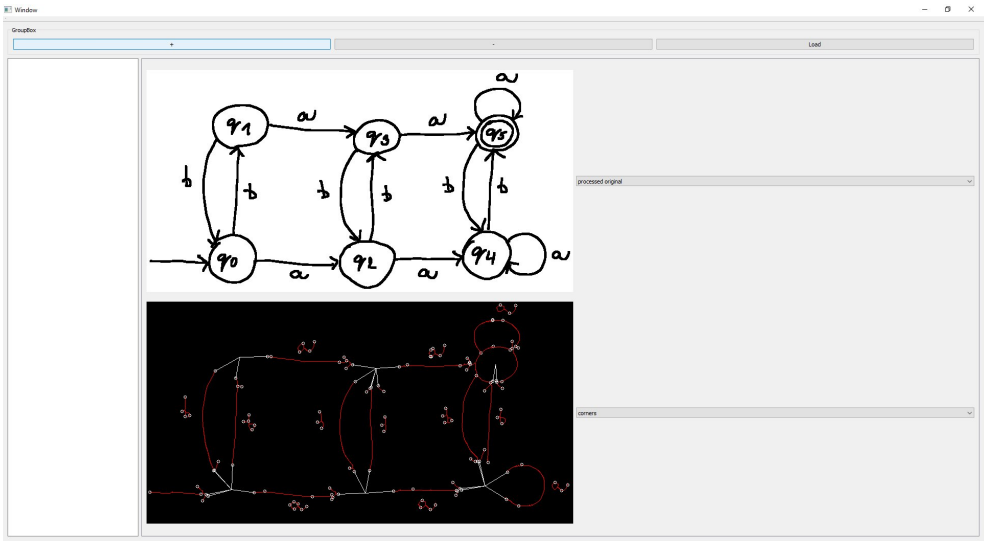
3. Oblast vykreslování obrazů. Pozor, Oblast se škáluje podle velikosti obrazu takže element 5 může zajet za okraje okna, pokud se tot stane prosím použijte scroll bar na spodní liště.

4. Tlačítko pro nahrání obrazu z disku nebo jiného uložiště. Otevře dialogové okno pro vybrání obrazu ke zpracování, Při výběru více jak jednoho objektu se načítá první v pořadí vybraných objektů.

5. Kombo box pro zvolení aktuálního vykreslení v 3, ke kterému je přiřazen.



Obrázek 3.2: Ukázka kombo boxu GUI



Obrázek 3.3: Ukázka přidání image view GUI

3.3 Zpracování obrazu

Samotné zpracování obrazu se skládá z mnoha kroků. Na počátku máme diagram automatu. Tento diagram je na bílém pozadí, kde důležité prvky jsou

zobrazeny barvou nerovnající se bílé. Naším cílem je obraz předzpracovat, segmentovat a poté klasifikovat segmentované části obrazu.

3.3.1 Detekce objektů

Detekce tvarů a objektů je nedílnou součástí práce s obrazem. Metody detekce nám dovolí detekovat a segmentovat vybrané objekty, které jsou pro nás důležité.

3.3.1.1 Kontura objektů

Kontůra je uspořádaná množina bodů. Pokud dokážeme správně segmentovat obraz a získat kontůru objektu, který nás zajímá, můžeme získat informace o vlastnostech objetku a nadále ho klasifikovat. Kontůra objektu nám může prozradit informace např. o poloze, tvaru, a jiné informace, které můžeme využít pro další segmentaci. [29].

3.3.1.2 Hierarchie

Nalezení kontůr nám dává informace přímo o nalezeném objektu, pokud ale chceme zjistit více musíme se soustředit na celkovou hierarchii všech objektů v obraze. Naštěstí tento problém za nás řeší OpenCV, který získává informace o hierarchii již při hledání kontůr a dokáže nám tuto hierarchii vytvořit dle potřeb do různých hierarchických rozpoložení. Pomocí hierarchie můžeme zjistit relativní polohy nalezenáých kontůr a tak získat více informací o dané struktůře celého obrazu. V OpenCV je hierarchie reprezentována pomocí vektoru o čtyřech prvcích **[další, předchozí, první potomek, rodič]** . Potomek a rodič značí aktuální hladinu v hierarchii, kde potomek je kontůra plně obsažená v aktuální kontůře a rodič kontůra, která obsahuje naší aktuální kontůru. Další a předchozí zprostředkovávají informaci o současné hladině hierarchie, to znamena že posouváním prostřednictvým další a předchozí se můžeme dostat ke kontůrám na stejné hladině jako je naše aktuální kontůra.

3.3.1.3 Charakteristiky objektů

Poté co nalezneme kontůry objektů nemusíme skončit pouze u jejich velikosti, pozici ,a nebo tvaru. O každé kontůře můžeme získat jiné charakteristiky jako jsou například

bounding box který nám může podat informaci o minimálním čtverci, který obsahuje kontůru.

$\boldsymbol{Aspect\ ratio} = \frac{\boldsymbol{Width}}{\boldsymbol{Height}}$, kde $\boldsymbol{Width}$ a $\boldsymbol{Height}$ jsou šířka a výška bounding boxu.

$\boldsymbol{Extent} = \frac{\boldsymbol{Contour\ area}}{\boldsymbol{Bounding\ rectangle\ area}}$, kde $\boldsymbol{Contour\ area}$ je velikost oblasti uvnitř kontůry a $\boldsymbol{Bounding\ rectangle\ area}$ je obsah čtverce kontlůry.

$\boldsymbol{Solidity} = \frac{\boldsymbol{Contour\ area}}{\boldsymbol{Convex\ hull\ area}}$, kde $\boldsymbol{Convex\ hull\ area}$ je velikost oblasti konvexní obálky kontůry.

3.3.1.4 Floodfill (seedfill)

[30] Tento algoritmus funguje na principu naplnění spojené komponenty, kde tato komponenta má stejnou barvu, která je určená na vstupu. Floodfill dokáže ze seed (počátečního) bodu vrátit komponentu, která obsahuje tento seed bod a je stejnou barvou. Rozhodnutí, zdali je bod ve stejné komponentě je zjištěno pomocí podobnosti sousedících bodů. Bod $\boldsymbol{(x, y)}$ se považuje za bod této komponenty pokud:

$$\begin{aligned} \boldsymbol{p(x', y')_r - lowDiff_r \leq p(x, y)_r \leq p(x', y')_r + upDiff_r} \\ \boldsymbol{p(x', y')_g - lowDiff_g \leq p(x, y)_g \leq p(x', y')_g + upDiff_g} \\ \boldsymbol{p(x', y')_b - lowDiff_b \leq p(x, y)_b \leq p(x', y')_b + upDiff_b} \end{aligned} \tag{3.1}$$

pro barevné body v obraze, kde $\boldsymbol{p(x, y)}$ je bod, který se testuje na podobnost, $\boldsymbol{p(x', y')}$ je bod, se kterým je testující bod porovnán na podobnost, $\boldsymbol{lowDiff}$ a $\boldsymbol{upDiff}$ jsou dolní a horní hranice přijatelného rozdílu. Bod $\boldsymbol{p(x', y')}$ je nastaven podle vstupu metody, kde buď se porovnává jako počáteční bod, a nebo je nastaven jako již zpracovaný soused v komponentě.

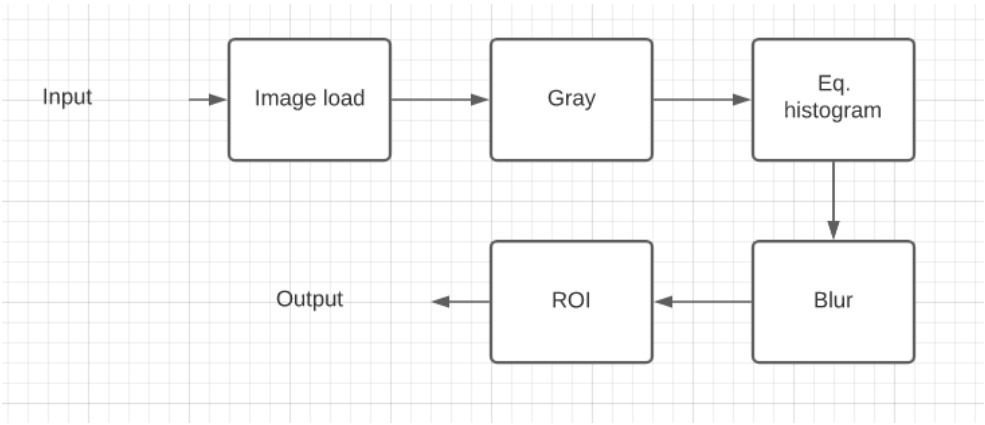
[30]

3.3.1.5 Detekce rohů

Pro detekci rohů byl použit algoritmus Shi-Tomasi algoritmus. [31]

3.3.2 Předzpracování obrazu

Na počátku předzpracování máme neupravený obraz v jednom z často používaných formátů např. png. Naším cílem je úprava obrazu pro jednoduší práci při detekci. Proces předzpracování obrazu, který je použit v aplikaci je zobrazen na 3.4 a popsán níže.



Obrázek 3.4: Předzpracování

1. **Načtení obrazu** - prvním krokem je načtení obrázku do programu. K tomuto účelu můžeme využít metod OpenCV pro práci s obrazy. Tato metoda dokáže pracovat s formáty: .bmp, .jpg, .png, .webp, .jp2, .sr, .tiff, .pbm, .exr.

2. **Převod na šedotónový obraz** - Po načetení obrazu použijeme metodu pro převod do šedotónového obrazu 2.5.2.1. Oproti barevnému snímku nám tato metoda umožní jednoduší práci s prahováním a metodami segmentací.

3. **Ekvalizace histogramu** - Podle 2.5.2.3 provedeme ekvalizaci histogramu a získáme tak lepší vyvážení barev.

4. **Aplikace gausova filtru** - Nyní aplikujeme rozostření s gausovým filtrem 2.5.2.4, které sníží senzitivitu na šum pro naše ostatní kroky. Zbavení se šumu nám zajistí lepší výsledky s vyhledáváním kontůr v dalším kroku.

5. **Segmentace** - Jako poslední krok předzpracování můžeme oříznout obrázek pomocí segmentace. Novou oblast najdeme pomocí metody nalezení kontůr 3.3.1.1, pomocí které zjistíme kde v obraze se nacházejí důležitá místa s informacemi.

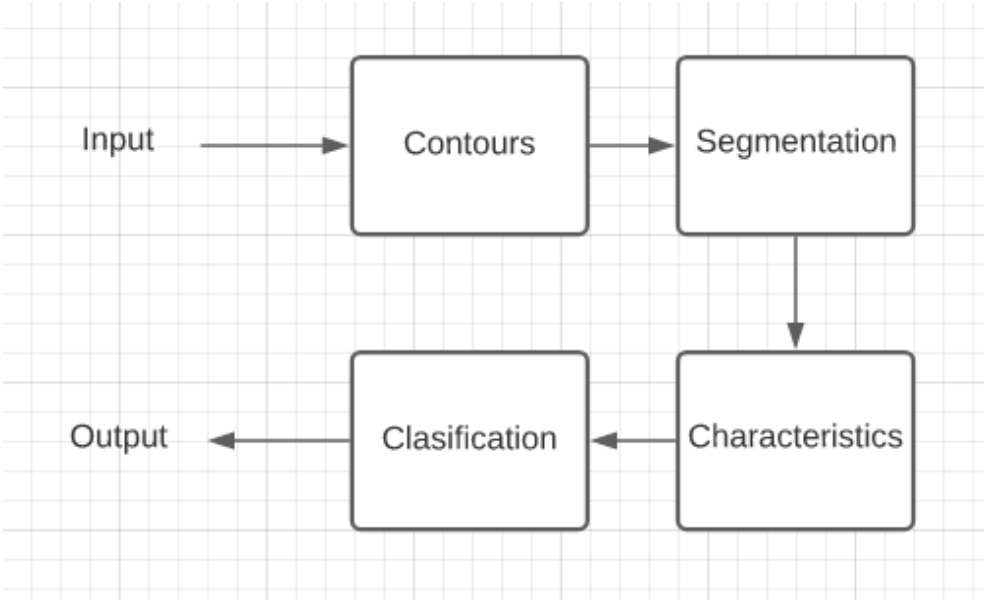
3.3.3 Detekce stavů

Na vstupu detekce je předzpracovaný obraz. Cílem tohoto kroku je segmentovat obraz a klasifikovat stavy 3.5. Tento úkol nám ztěžuje fakt, že stavy mohou mít více klasifikací:

- počáteční stav

- koncový stav
- stav bez přidané informace (stav, který není ani koncový ani počáteční)

Pro počáteční stavy naše detekce probíhá stějně jako u stavu bez přidané hodnoty, jak je popsáno níže. Rozdíl nastává pro koncové stavy. V případě koncového stavu se v kroku kde se zjišťují charakteristiky zjistí, že nalezený objekt je v hierarchii s jiným objektem kterýmu byla zjištěna charakterizace stavu. V momentu kdy se zjistí daná skutečnost se stavy sjednotí a uchová se informace jen o stavu, který je nadřazený a stav se označí jako koncový. Problém nastává pro malé kruhy uprostřed stavů např. symbol **0**. Avšak tento problém je řešen výpočtem a následnou filtrací stavů s velkým rozdílem velikosti oproti ostatním nalezeným stavům (outlier).



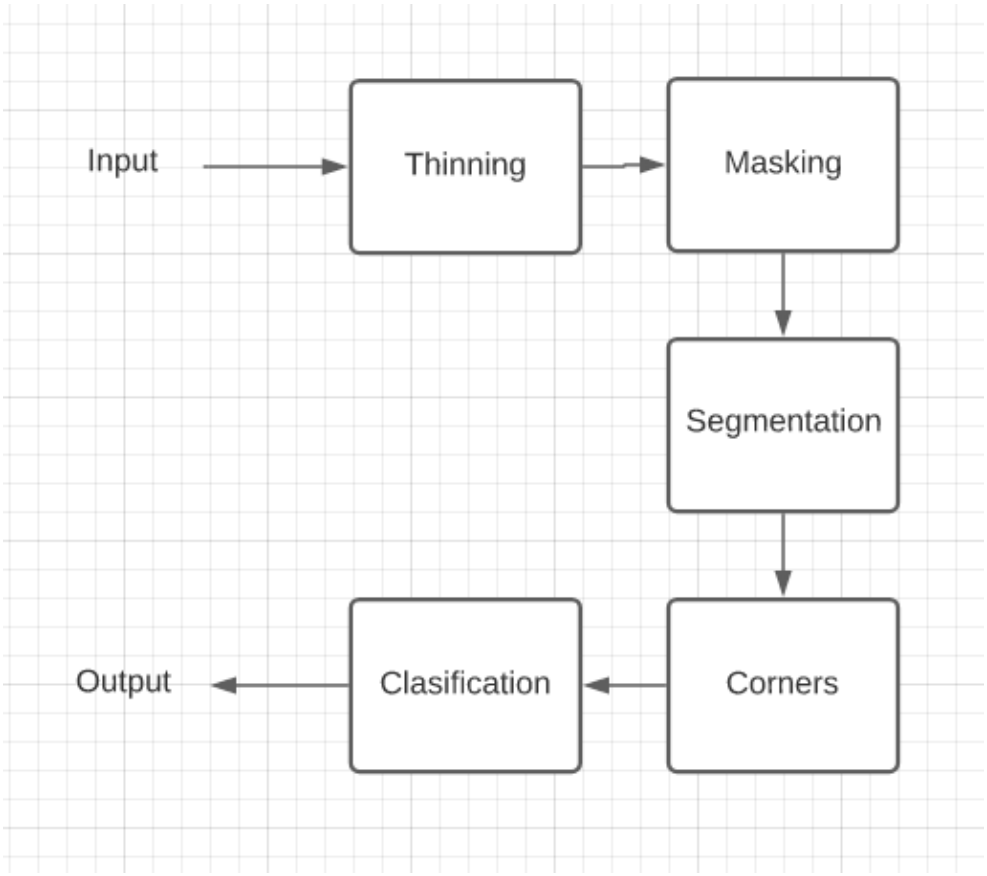
Obrázek 3.5: Detekce stavů

1. **Segmentace** - První krok při detekci je segmentace pomocí kontůr, které získáme z obrazu pomocí metod definovaných v OpenCV 3.3.1.1. U detekce stavů nás budou zajímat kontůry vnitřní. Tyto kontůry získáme s pomocí hierarchie kontůr, vytvořené při nacházení kontůr. Poté zahodíme příliš malé kontůry, které můžeme brát jako šum.
2. **Charakteristiky** - Tento krok se v procesu snaží zjistit informace o kontůrách a najít charakteristiky jako např. aspect ratio, kulatost a nebo jestli je daná kontůra v hierarchii nad jinou kontůrou což by indikovalo existenci jiné kontůry, která může být mimo jiné i symbol tohoto stavu (pokud se jedná o stav).

Tato metoda dosahuje příznivých výsledků i pokud se nejedná o perfektní kruhy, dokáže detekovat i ellipsoidy do určité úrovně podobnosti s kruhy. U detekce stavů nebyla použita metoda Hough transform pro hledání kruhů. Tato metoda často nebyla senzitivní na ellipsoidy, které nebyly blízko perfektnímu kruhu. Jako zlepšení by se dalo spojit metody použité v programu a Hough transform a važit jejich výsledky aby se mohly vzájemně doplňovat v nedostatcích. Hough transform je robustní metoda pro detekci objektů v obraze, avšak je závislá na vysoké podobnosti hledaného objektu. Sensitivita vůči ellipsoidům se s šišatostí značně snižuje.

3.3.4 Detekce přechodů

Na vstupu máme předzpracovaný obraz, stejný jako v předchozím kroku, a informace o možných stavech. Cílem tohoto kroku je zjednodušit informace v obraze pomocí metody thinning **??**, vymazat detekované struktůry z předchozího kroku, segmentovat zbylé objekty, které budou z největší pravděpodobností obsahovat informace o přechodech mezi stavy a jejich klasifikace. Postup pro detekci přechodů je znázorněn na obrázku 3.6.



Obrázek 3.6: Detekce přechodů

1. **Thinning** - V programu je implementován algoritmus pro skeletonizaci 2.5.4.6. Tento algoritmus je rychlý a vytváří malé počty artefaktů ve výsledném skeletu, např. oproti metodě skeletonizace s pomocí morfologických operací 2.5.4.5. Výsledný skelet nám sníží počet informací v obrazu na úkor tvarových charakteristik. V tomto kroku již jich není potřeba zachovávat tvary , a tak není problém použít algoritmus.

2. **Maskování** - Maskování předem nalezených stavů z obrazu nám nadále sníží počet informací, které musíme zpracovat a usnadní nám následnou segmentaci. Hlavní problém tohoto kroku je transformace původního obrazu na jeho skelet. Jelikož se nyní z předchozího kroku zmenila informační stránka struktůr v obraze, tak nemůžeme použít výsledek z předchozí detekce v poměru 1:1. Z tohoto důvodu byl použit algoritmus flood fill 3.3.1.4 a eroze 2.5.3.2.

   Morfologická eroze je použitá na zmenšení masky pokrývající stav detekovaný v předchozích krocích. Velikost strukturálního elementu byla zvo-

lena $\mathbf{7 \times 7}$. Při testování bylo možné pracovat s velikostí elementu $\mathbf{5 \times 5}$. Obě tyto hodnoty byly zvoleny v testovací fázi aplikace. Po erozi masky získané z detekovaných stavů aplikujeme masku na obraz a vymažeme vnitřní prostor detekovaných stavů. Tímto krokem nám vznik vymazaný prostor uprostřed stavu. Je velice důležité aby vymaskovaný prostor byl uzavřen. Nyní můžeme aplikovat metodu flood fill. Tato metoda nám zaplní prostor až k hraně stavu který nám poskytl skelet stavu. Po vyplnění pomocí metody flood fill dostaneme novou masku o velikosti skeletonu stavu, který můžeme vymazan a zbavit se tak přebytečné informace a zjednodušit si práci s klasifikací přechodů.

Tato metoda má dva zásadní problémy. Může se stát, že skeletonizace vytvoří defekty viz. 3.7 (vlevo přetečení flood fill, vpravo skeleton z defektem, který způsobuje problém). Tento defekt může způsobit přetečení metody flood fill. Jako řešení tohoto problému je implementována maska pro maximalní flood fill, která zabrání přetečení za určitý limit, jako limita byla zvolena velikost hraničního boxu nalezeného stavu zvětšena o deset procent velikosti tohoto boxu. Druhý problém nastane pokud je eroze příliš slabá a zanechá uvnitř stavu zbytky objektů, které tento stav obsahoval. Tyto zbytky by byli ale velice malé vzhledem k velikosti elementu použit při erozi, takže se zahodí pro jeho malou velikost.

3. **Segmentace** - Z výsledku předchozího kroku máme na vstupu obraz bez stavů. Můžeme tedy získat kontůry zbylých objetů. Dále se kontůry filtrují velikostně (zahodíme kontůry, které jsou menší jak nejmenší vzdálenost mezi detekovanými stavy). Zbydou nám kandidáti přechodů.

4. **Detekce rohů** - Detekce rohů nám pomůže charakterizovat nalezené kontůry a zjistit zdali obsahují rohy, které můžeme použít pro udání směru přechodu. Pro nalezení rohů použijeme algoritmus Shi-Tomasi **??** pro detekci rohů. Směr přechodu získáme zjištěním okolí bodu pro daný roh jako suma jasu:

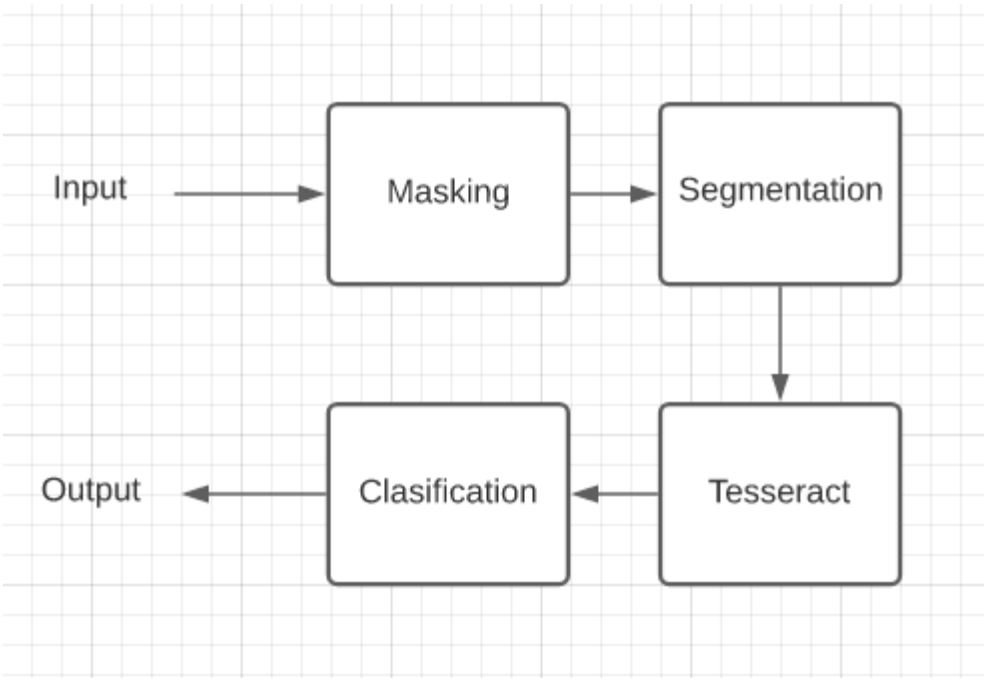
   $\boldsymbol{s_{i,j} = \sum_{x=i-k}^{i+k} \sum_{y=j-k}^{j+k} p(x,y)}$, kde $\boldsymbol{s}$ je suma výsledného jasu okolo pixelu $\boldsymbol{p(i,j)}$, $\boldsymbol{k}$ je velikost testovací matice, $\boldsymbol{2k < i,j}$

5. **Klasifikace** - Po zjištění směru se přechod přiřadí nejbližšímu stavu pro obě svoje strany. Rozdíl nastává při detekci jednoho, nebo žádného stavu v okolí. V případě žádného stavu se kontůra zahazuje. Při detekci jednoho stavu se ke stavu připojuje označuje ho za počáteční stav.

Obrázek 3.7: Defekt

3.3.5 Detekce textu

Vstupem pro tuto metodu nám přichází obraz bez stavů, které byli detekované v předchozích krocích. Cílem toto kroku je získat klasifikace pro text v obrazu a zařadit ho ke správné struktůře.



Obrázek 3.8: Detekce textu

1. **Maskování** - První krok je vymaskování nepotřebných informací z obrazu. Jde zde o totožné operace jako v předchozích krocích, s doplněním

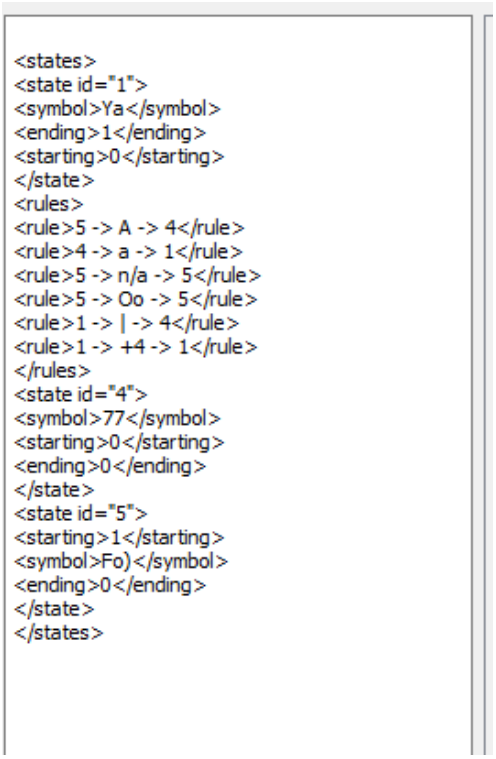
o maskování nalezených přechodů. Maskování těchto přechodů je pouze aplikování přebarvení kontůry přechodu na barvu pozadí (díky skeletonizaci je tato kontůra široká jeden pixel).

2. **Segmentace a Tesseract** - Segmentace se seskládá ze dvou kroků. První krok se stará o segmentaci symbolů ve stavu. Masku pro stav již máme připravenou a po aplikaci inverzní masky nám zůstane objekt uvnitř stavu. Tento objekt můžeme použít v klasifikaci a poslat ho dále do nástroje tesseract.

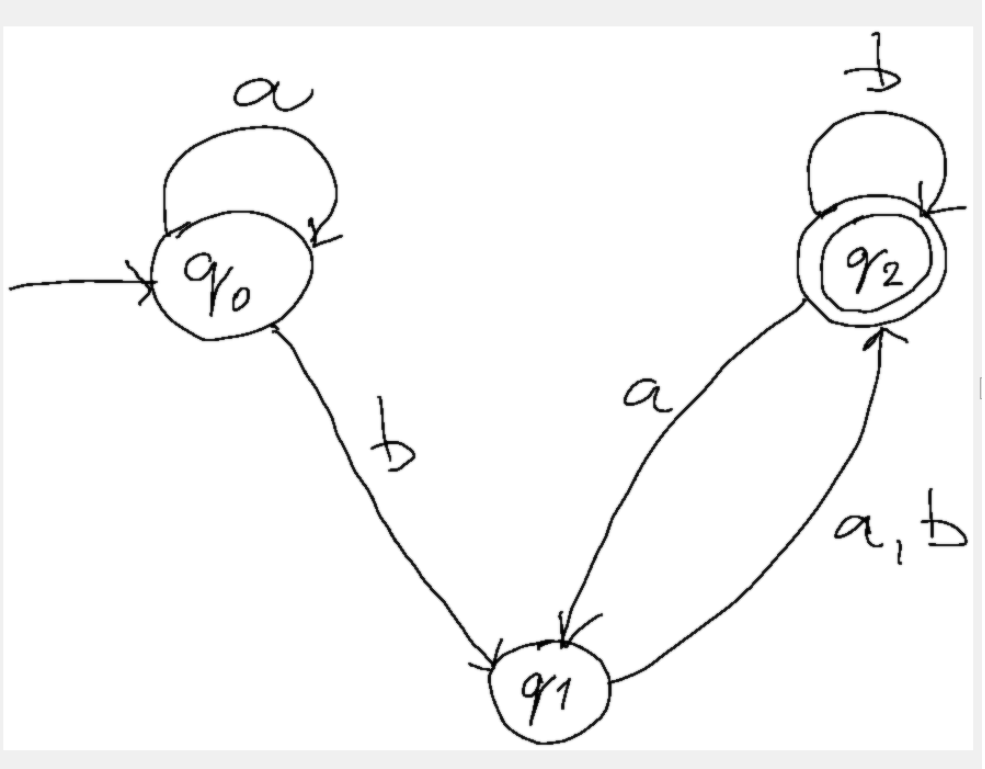
   Druhý krok se skládá ze získání zbylých objektů v obraze a poslání do nastoje tesseract. V tomto případě pokud v obraze zbyl objekt nepatřící mezi symboly, tesseract vrátí prázdný symbol, jelikož ho nezvládne klasifikovat. Toto řešení je slabou stránkou tohoto kroku. Řešení by mohlo spočívat v příkládání váhy pro všechny nalezené symboly zvolit nejlépe hodnocený symbol pro daný přechod/stav.

3. **Klasifikace** - V poslední fázi je čas přiřadit jednotlivé symboly k přechodům (stavy jsou jednoznačné). Přechodům jsou symboly přiřazené podle vzdálenosti od kontůry přechodu, kde nejbližší symbol je přiřazen.

4. **Výstupní forma XML** - Výstup aplikace mimo vizualizace samotné aplikace je formou dokumentu XML příklad výstupu 3.9 k diagramu 3.10. Tento Výstup se dá využít pro připojení k jiné aplikaci v budoucnu.



Obrázek 3.9: Ukázka výstupu XML



Obrázek 3.10: Ukázka výstupu diagram

Závěr

Cílem práce bylo vytvoření aplikace, která dokáže převést obraz diagramu konečného automatu do strojové formy pro jednoduší práci s automaty. Cíl práce se z části splnil. Program dokáže detekovat a klasifikovat objekty v diagramu deterministického konečného automatu a výsledek podává ve forme jazyka XML, která se může použít pro budoucí rozšíření nebo napojení na jiné aplikace. Automatizace detekce diagramu DKA z obrazu je možná a v budoucnu by se dala rozvést. Budování aplikace na detekci diagramů konečných automatu je nelehké, do problému vstupují spousty neznámých a jejich řešením se mohou zabývat celé obory. Naštěstí v dnešní době lze takovéto aplikace postavit jednodušeji díky nástojům jako je OpenCV. Tyto nástroje umožňují budovat na metodách a prokázaném postupu.

Literatura

[1] HOLUB, J.: Automaty a gramatiky: Základní pojmy. 2020, soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém USB disku. Dostupné z: https://courses.fit.cvut.cz/BI-AAG/lectures/bi-aag-01-zakladni_pojmy.pdf

[2] Finkelstein, J.: Graphical representation of the sets of languages included in the w:Chomsky hierarchy, 2010. Dostupné z: https://commons.wikimedia.org/wiki/File:Chomsky-hierarchy.svg

[3] HOLUB, J.: Deterministické a nedeterministické konečné automaty. 2020, soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém USB disku. Dostupné z: https://courses.fit.cvut.cz/BI-AAG/lectures/bi-aag-02-konecne_automaty.pdf

[4] Example of a DFA that accepts binary numbers that are multiples of 3, 2007. Dostupné z: https://commons.wikimedia.org/wiki/File:DFA_example_multiplies_of_3.svg

[5] DOLEŽAL, J.: Základní barevné systémy, 2021. Dostupné z: http://homel.vsb.cz/~dol75/Pgrafika/Prednaska/ZaklBarevneSys.html

[6] Quark67(Modified color by Monami) – Image:Synthese+.svg, C. B.-S. .: Additive color mixing with RGB colors, 2008. Dostupné z: https://commons.wikimedia.org/w/index.php?curid=4798169

[7] The HSV color model mapped to a cylinder. POV-Ray source is available from the POV-Ray Object Collection, 2021. Dostupné z: https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png

[8] Václav Hlaváč, J. K.: Předzpracování v prostoru obrazů: Lokální předzpracování. 2020, soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém USB disku. Dostupné z:

http://cmp.felk.cvut.cz/cmp/courses/33DZOzima2005/slidy/lokalniPredzpracovani.pdf

[9] grt: Princip výpočtu dvourozměrné diskrétní konvoluce, 2006. Dostupné z: https://commons.wikimedia.org/w/index.php?curid=969064

[10] Grayscale, 2021. Dostupné z: https://commons.wikimedia.org/wiki/File:Beyoglu_4671_tricolor.png

[11] Histograms in Image Processing with skimage-Python, 2021. Dostupné z: https://towardsdatascience.com/histograms-in-image-processing-with-skimage-python-be5938962935

[12] Histogram Equalization, 2021. Dostupné z: https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf

[13] „Smial“: Odstranění šumu, 2021. Dostupné z: https://commons.wikimedia.org/w/index.php?curid=3055755

[14] Marcel Jiřina, L. B., Jakub Novák: Filtrace v prostorové a frekvenční oblasti. 2020, soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém USB disku. Dostupné z: https://courses.fit.cvut.cz/BI-SVZ/lectures/files/bi-svz-07-filtrace-v-prostorove-a-frekvencni-oblasti.pdf

[15] Dougherty, E.: *Mathematical morphology in image processing*. CRC press, 2018.

[16] Morphological Transformations, 2021. Dostupné z: https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

[17] Kanopoulos, N.; Vasanthavada, N.; Baker, R. L.: Design of an image edge detection filter using the Sobel operator. *IEEE Journal of solid-state circuits*, ročník 23, č. 2, 1988: s. 358–367.

[18] Canny, J.: A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, , č. 6, 1986: s. 679–698.

[19] Canny edge detector, 2021. Dostupné z: https://www.mathworks.com/matlabcentral/fileexchange/40737-canny-edge-detector

[20] Marcel Jiřina, L. B., Jakub Novák: Morfologie a tvarové charakteristiky. 2020, soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém USB disku. Dostupné z: https://courses.fit.cvut.cz/BI-SVZ/lectures/files/bi-svz-10-morfologie-a-tvarove-charakteristiky.pdf

[21] Dougherty, E. R.: *An Introduction to Morphological Image Processing*. SPIE Optical Engineering Press, 1992, ISBN 0-8194-0845-X. Dostupné z: https://books.google.cz/books?id=1kvxAAAAMAAJ&q=isbn:081940845X&dq=isbn:081940845X&hl=cs&sa=X&ved=2ahUKEwjOqOTB74ruAhVCKuwKHcKRDSkQ6AEwAHoECAAQAg

[22] Chen, W.; Sui, L.; Xu, Z.; aj.: Improved Zhang-Suen thinning algorithm in binary line drawing applications. In *2012 International Conference on Systems and Informatics (ICSAI2012)*, IEEE, 2012, s. 1947–1950.

[23] Finkelstein, J.: Graphical representation of the sets of languages included in the w:Chomsky hierarchy, 2010. Dostupné z: https://commons.wikimedia.org/wiki/File:Chomsky-hierarchy.svg

[24] Qt: Qt 5.12.0 [software]. 2020-01-01. [cit. 2020-02-14]. https://www.qt.io/. Linux, Windows.

[25] OpenCV: OpenCV 4.0.1 [software]. 2020-01-01. [cit. 2020-02-14]. https://opencv.org/. Linux, Windows.

[26] Google: Tesseract 4.0.0 [software]. 2018-09-29. [2020-02-14]. https://github.com/tesseract-ocr/tesseract. Linux, Windows, and macOS (x86).

[27] Leptonica: Leptonica 5.0 [software]. 2020-01-01. [cit. 2020-02-14]. http://www.leptonica.org/. Linux, Windows.

[28] Hamák, O.: Rozpoznání konečného automatu z obrazu. 2019.

[29] Kumar, A. H.: Contours in Images. *The Startup*. Dostupné z: https://medium.com/swlh/contours-in-images-a58b4c12c0ff

[30] Vandevenne, L.: Lode's Computer Graphics Tutorial. Dostupné z: https://lodev.org/cgtutor/floodfill.html

[31] Jiambo Shi, C. T.: Good Features to Track. 1994. Dostupné z: http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf

Příloha **A**

Seznam použitých zkratek

SVM Support vector machine (Metoda podpůrných vektorů)

OCR Optical character recognition (Optické rozpoznávání znaků)

LSTM Long short-term memory

API Application programming interface (rozhraní pro programovací jazyk)

GUI Graphical user interface (Grafické uživatelské prostředí)

IDE Integrated eevelopment environment (Vývojové prostředí)

Příloha B

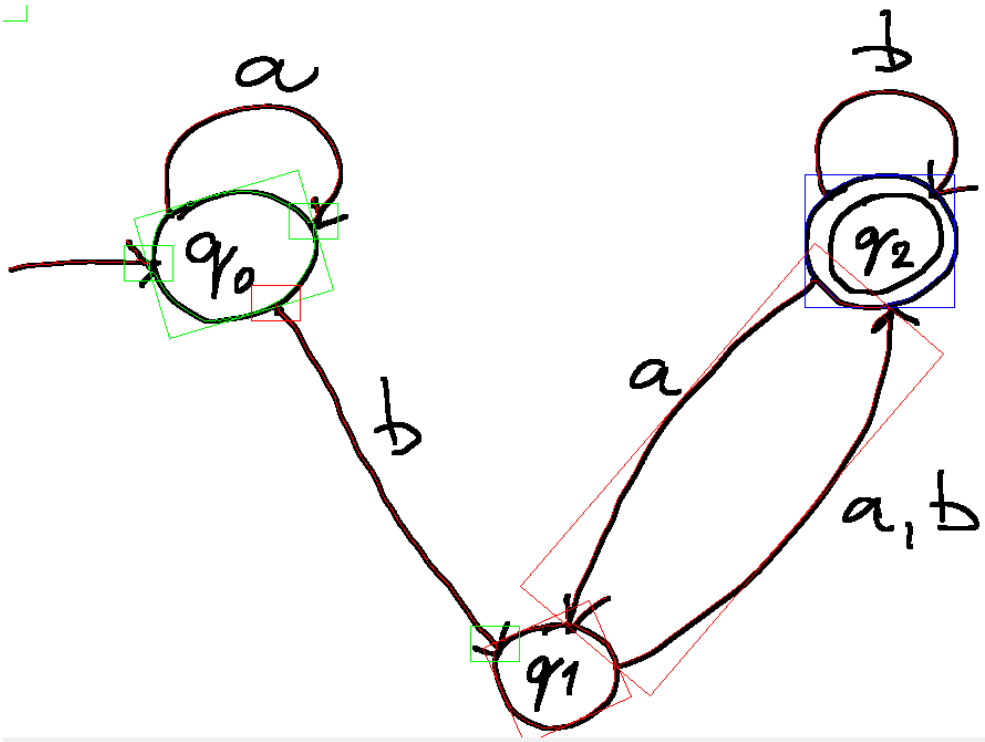
Ruční testy

V těchto testech jsou výstupy z datasetu FAdataset, poskytnutým panem Ing. Jan Trávníčekem, Ph.D. Kde malé čtverce na stavech značí směr detekovaných stavů, zelená barva stavu značí startovní stav, modra barva koncový stav a červená regulární stav. Pokud je detekce a klasifikace symbolů funkční tak poté je tato detekce znázorněna modrou barvou a klasifikace je napsaná uvnitř této detekce.

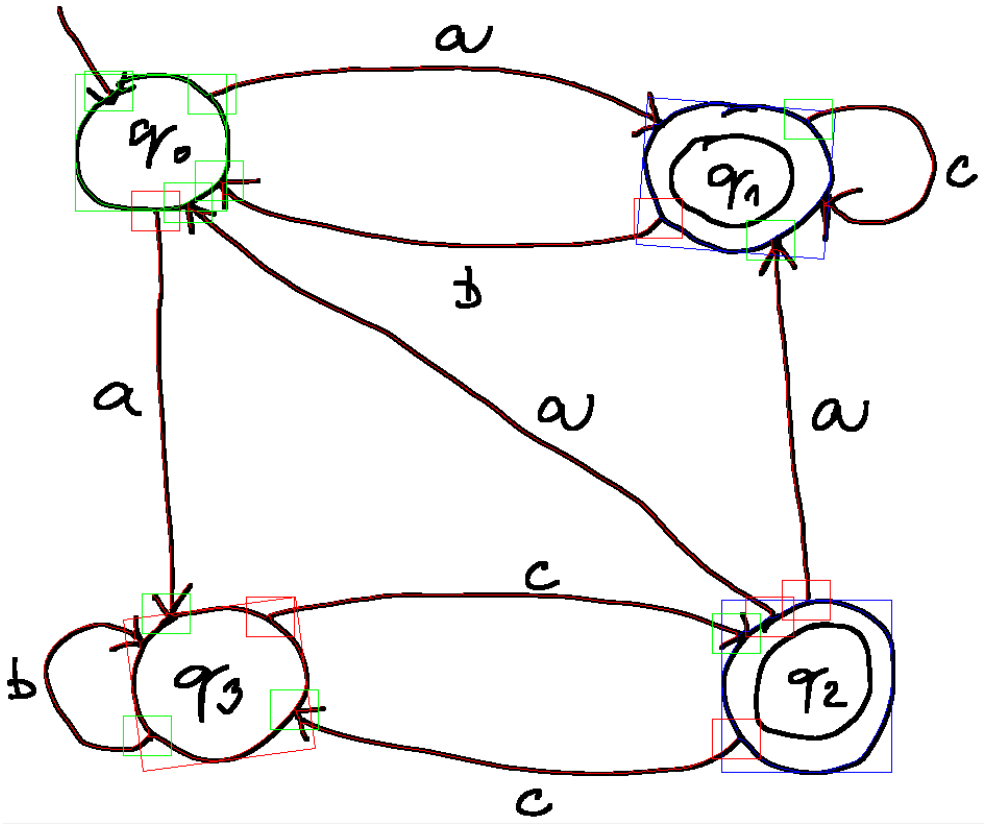
Testování proběhlo ručně nad náhodně vybraným vstupem, bylo vždy zjištěno celkový počet detekovaných objektů v obraze, kolik detekcí proběhlo celkově a jestli se detekovali správné objekty.

Tabulka B.1: Výsledky ručních testů

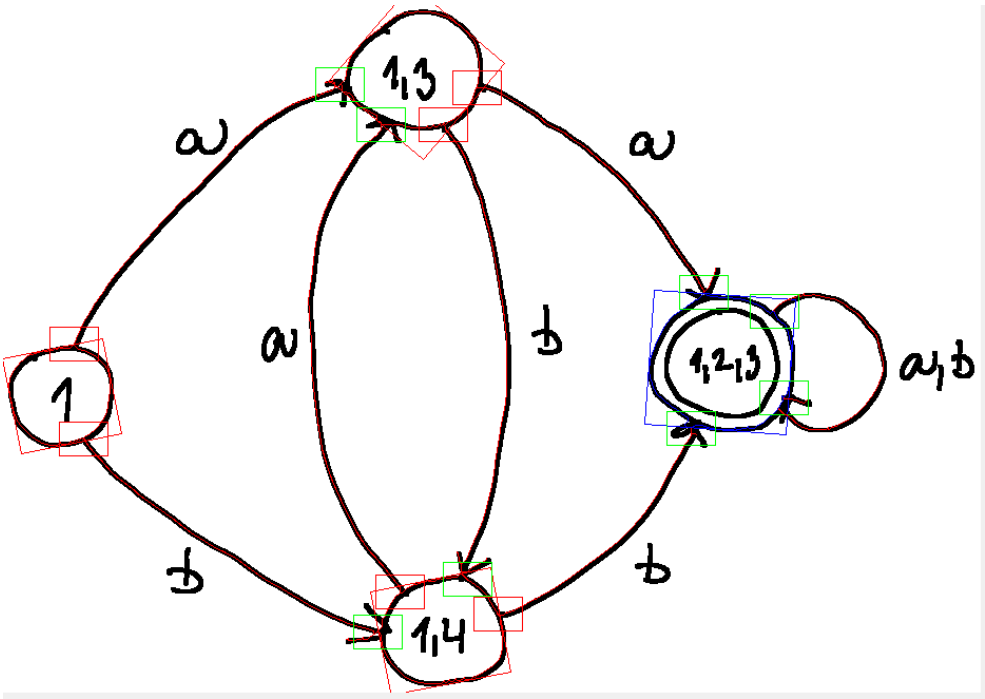
ok	**Počet objektů**	**Počet detekcí**	**Počet správných detekcí**
Stavy	77	56(72%)	34(44%)
Přechody	137	97(70%)	66(48%)
Symboly	209	63(17%)	22(10%)



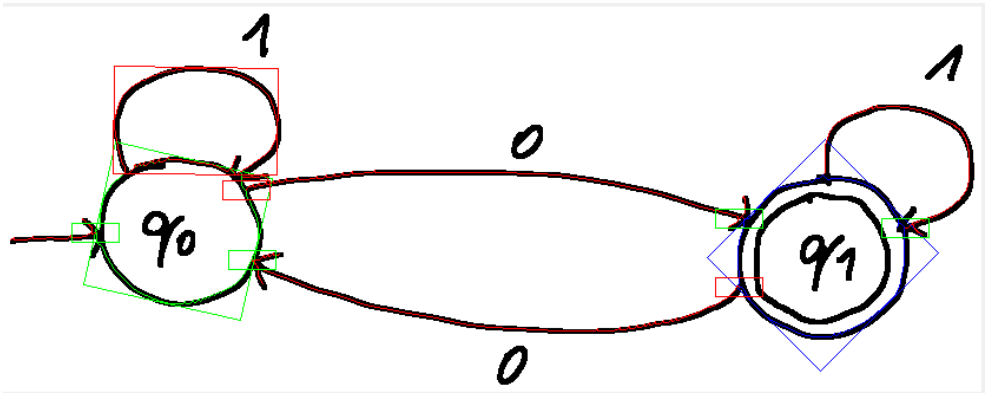
Obrázek B.1: test01



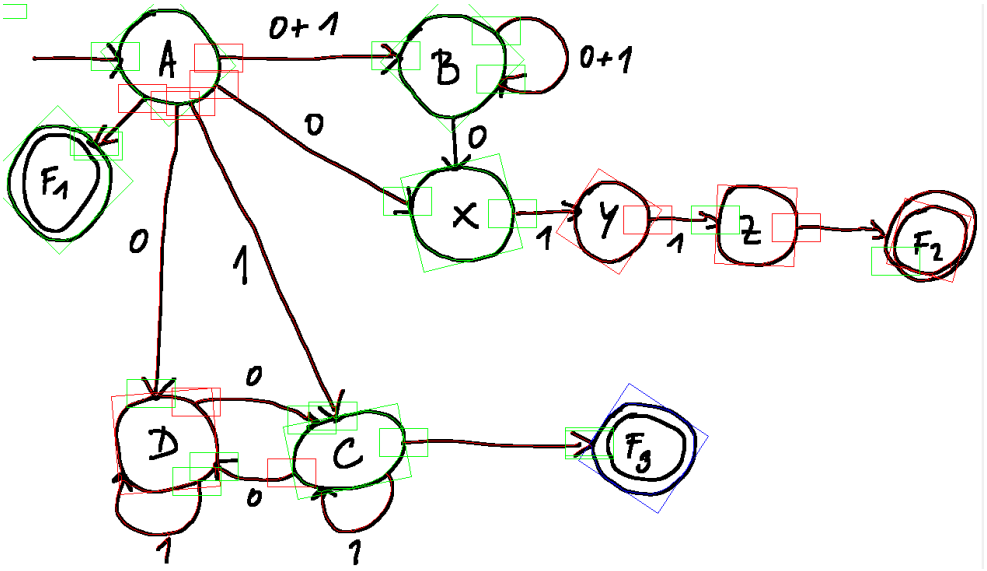
Obrázek B.2: test02



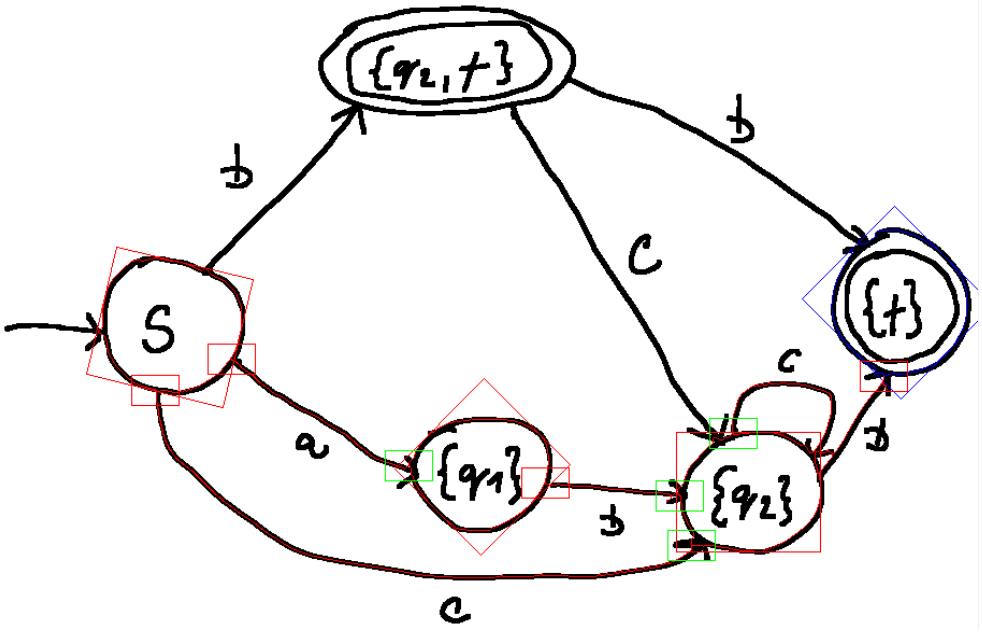
Obrázek B.3: test03



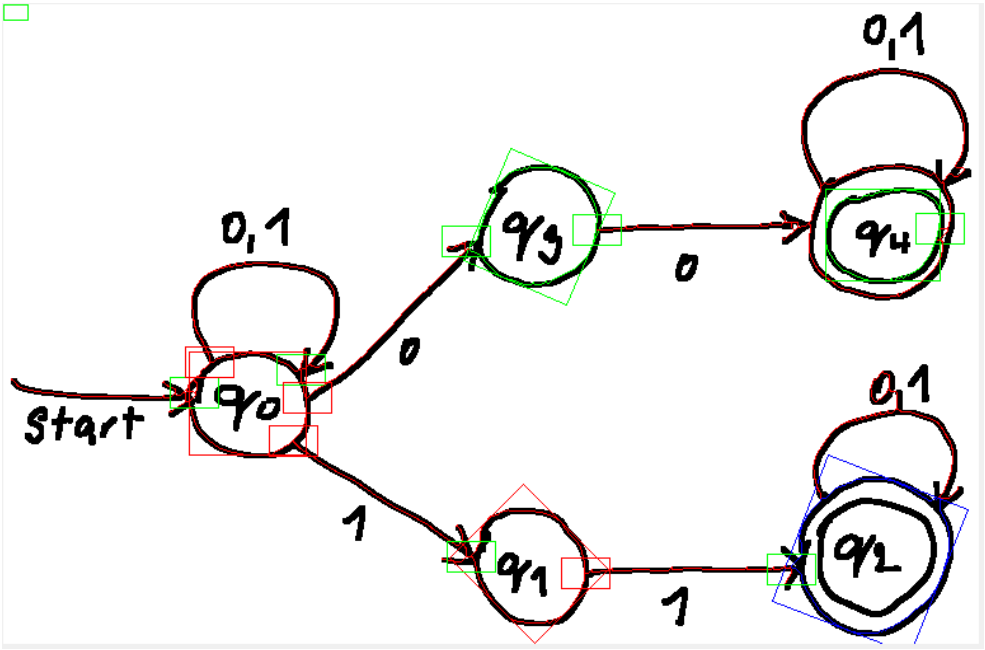
Obrázek B.4: test04



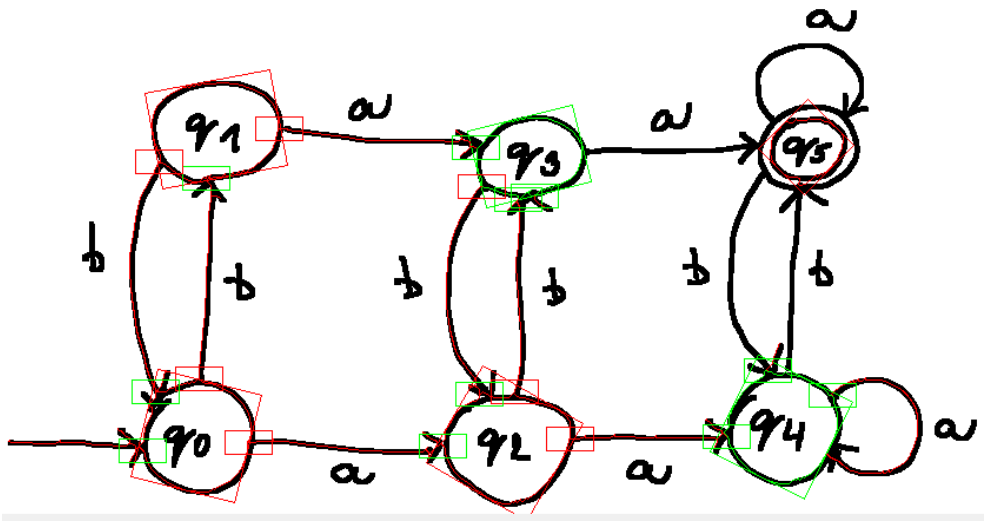
Obrázek B.5: test05



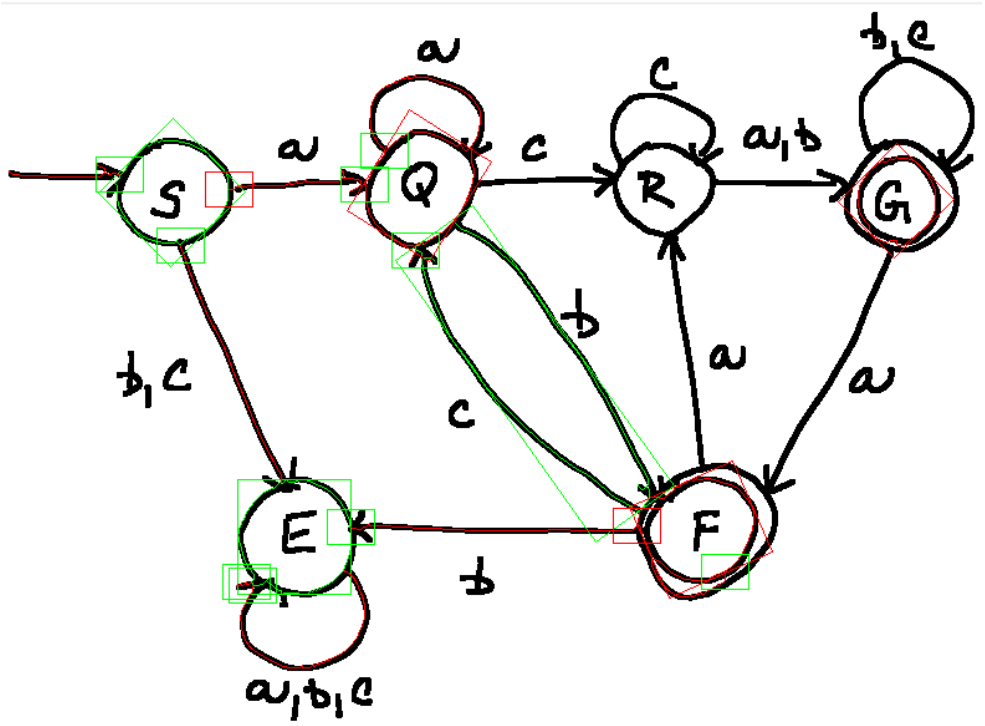
Obrázek B.6: test06



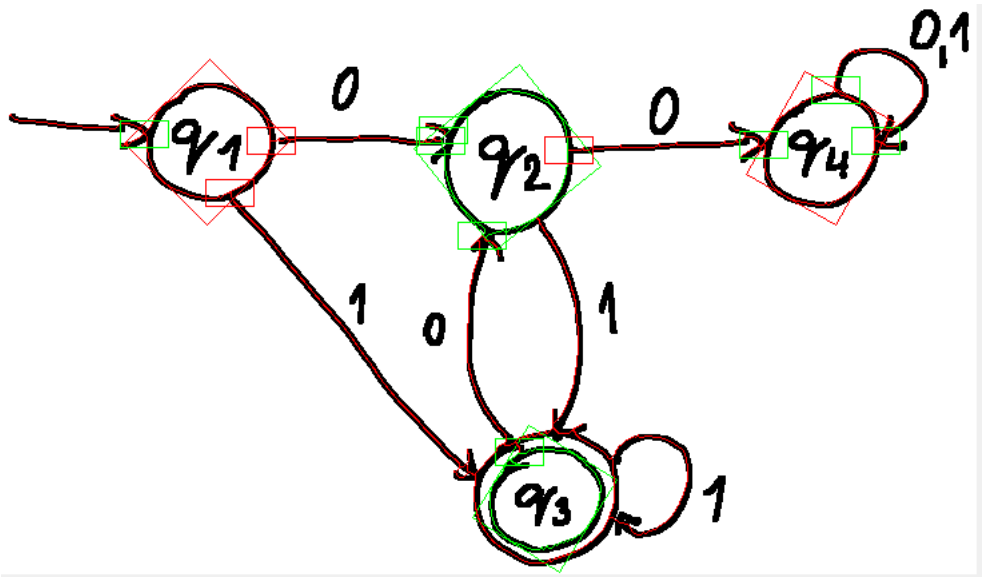
Obrázek B.7: test07



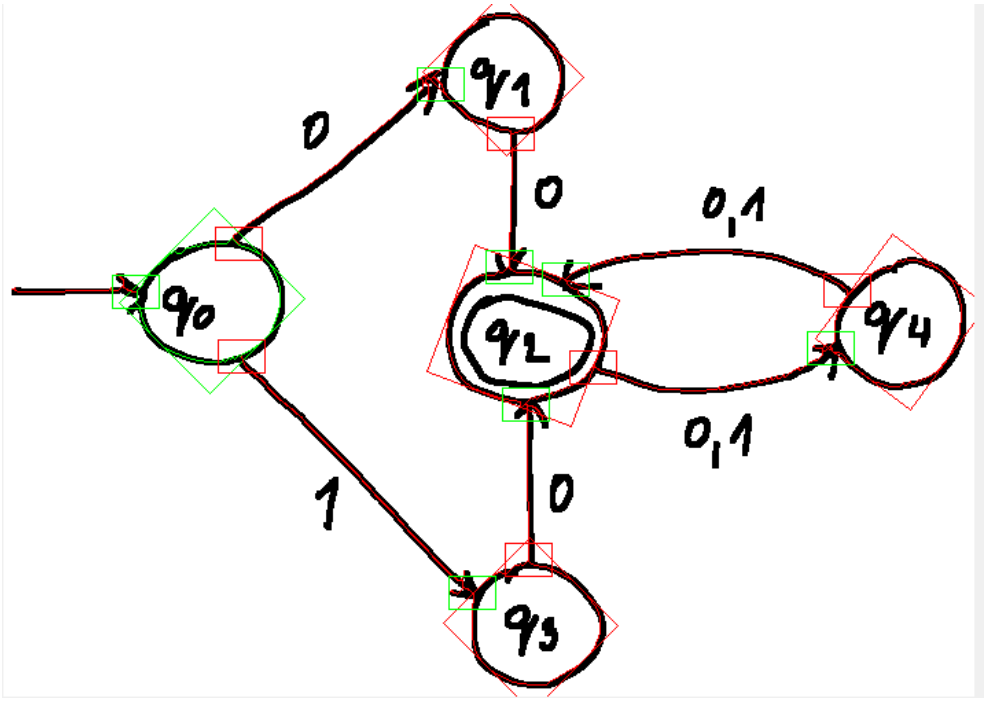
Obrázek B.8: test08



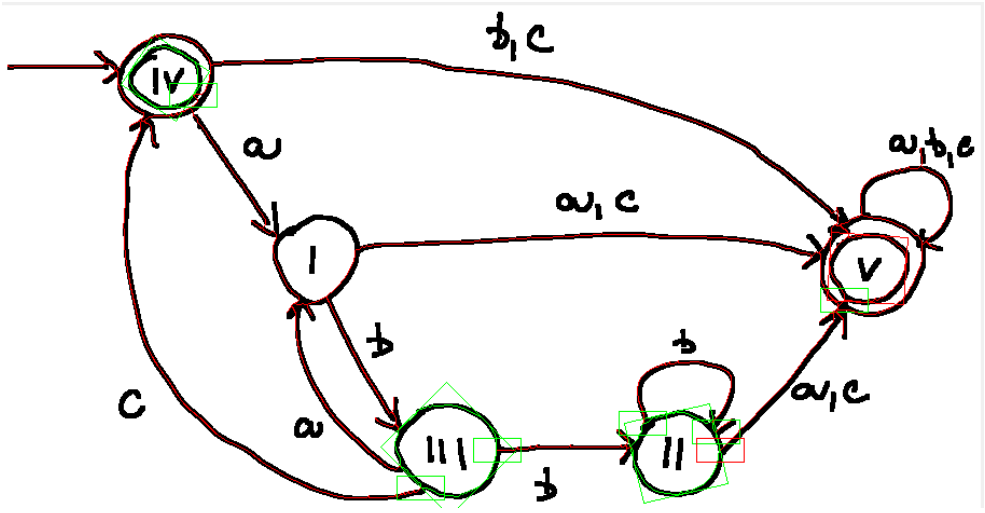
Obrázek B.9: test09



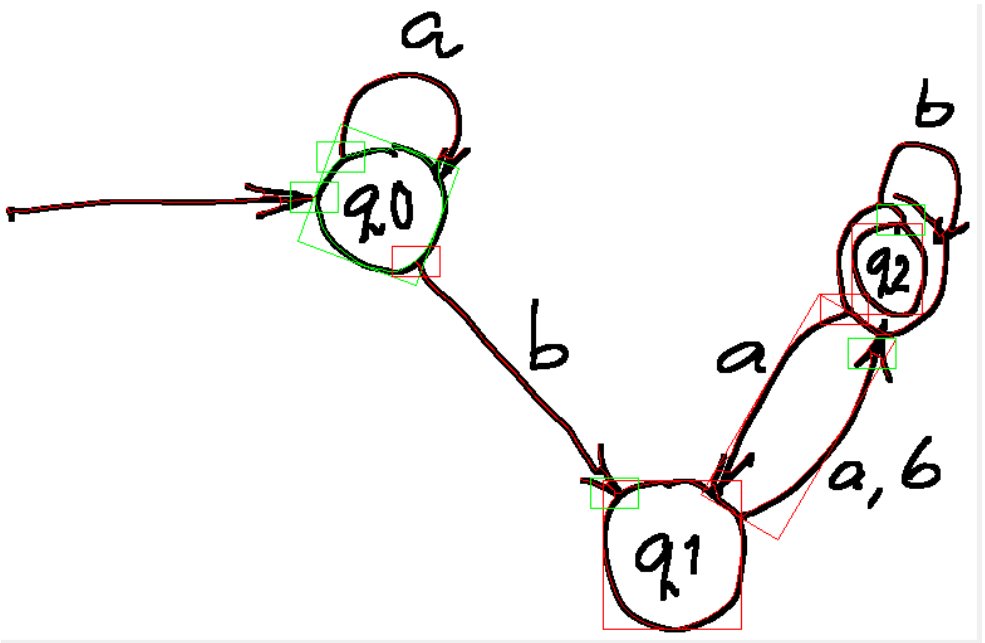
Obrázek B.10: test10



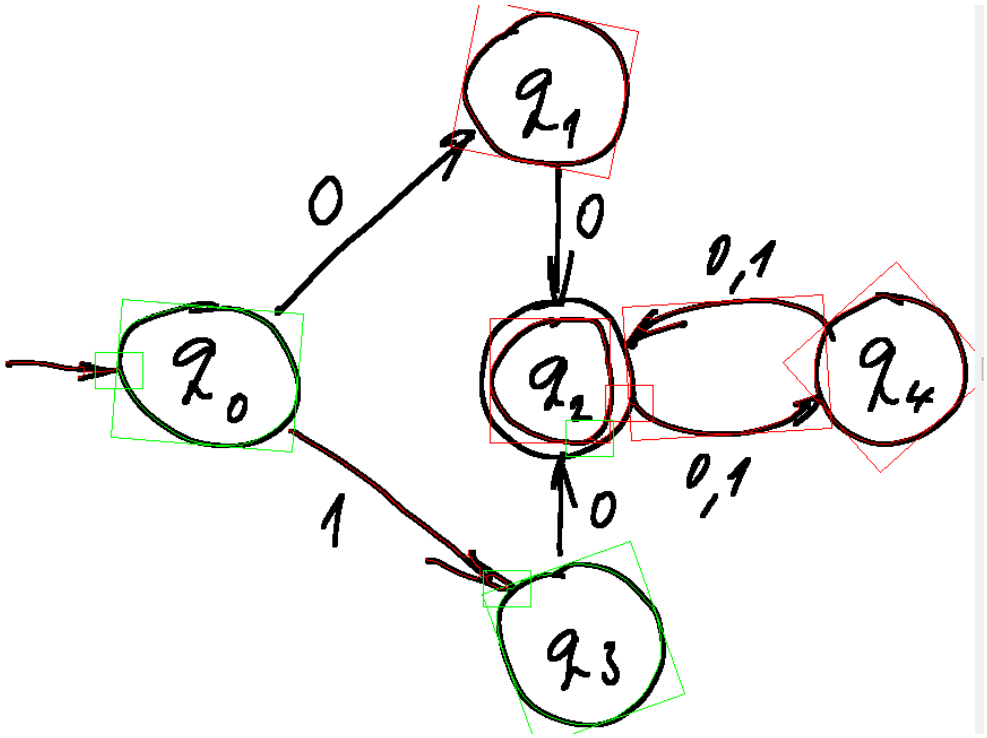
Obrázek B.11: test11



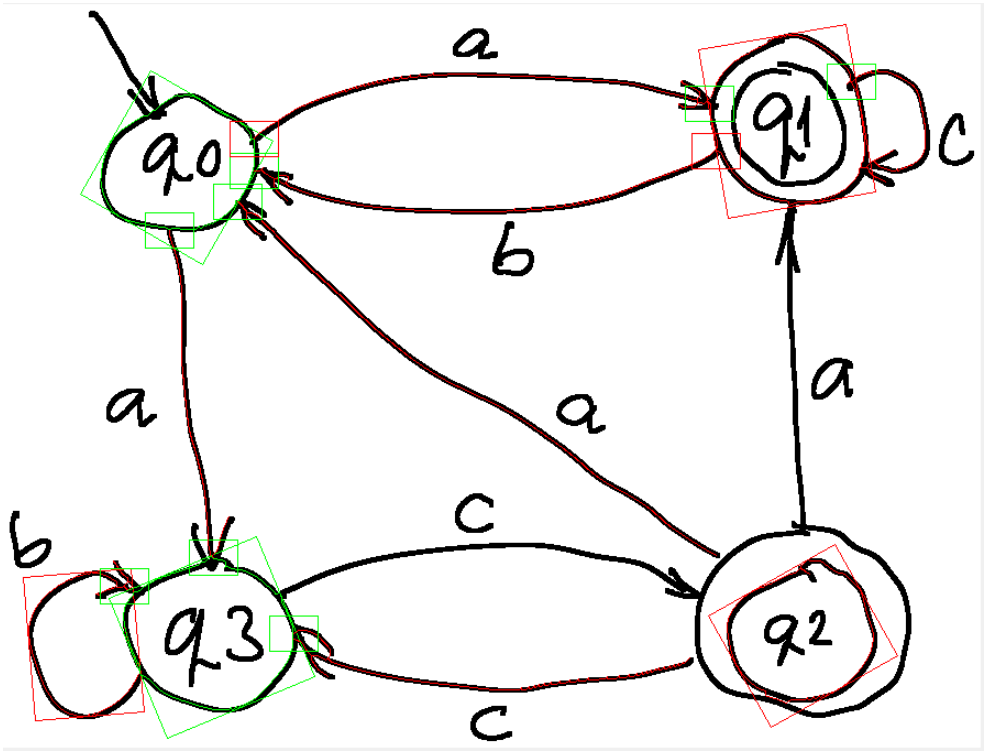
Obrázek B.12: test12



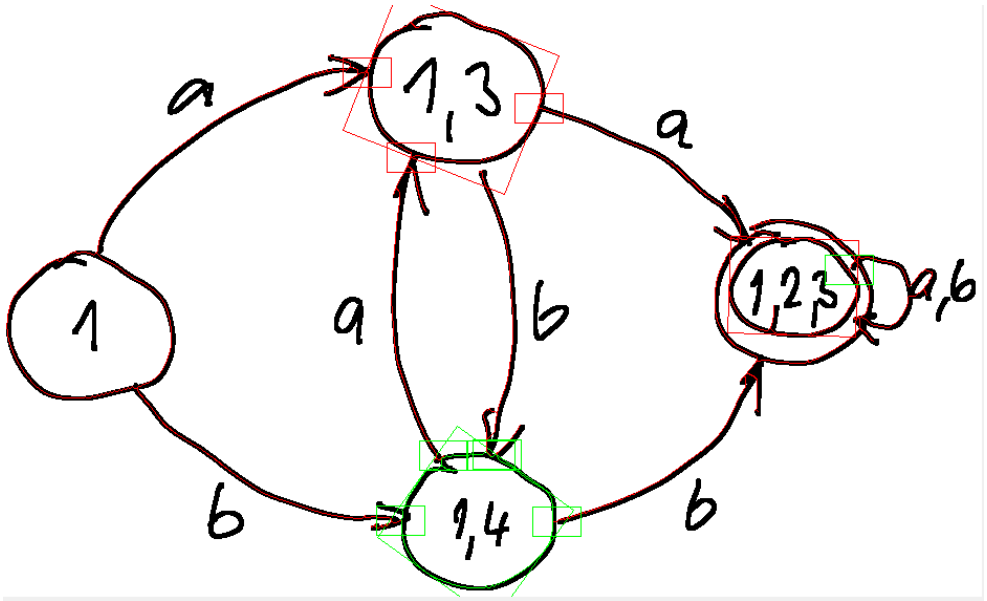
Obrázek B.13: test13



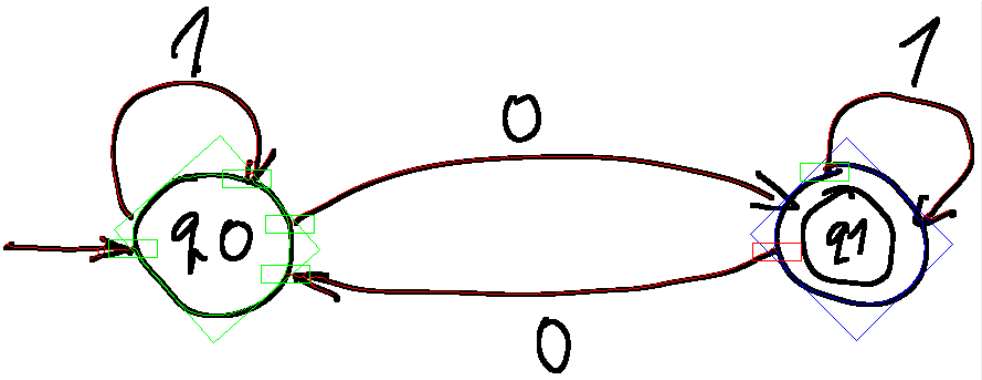
Obrázek B.14: test14



Obrázek B.15: test15



Obrázek B.16: test16



Obrázek B.17: test17

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `core` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `doxygen` ..... dokumentace ke zdrojovým kódům
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF