VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DVOUDIMENSIONÁLNÍ KONEČNÉ AUTOMATY A JEJICH APLIKACE

TWO-DIMENTIONAL FINITE AUTOMATA AND THEIR APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DOMINIK ŠVAČ**
AUTHOR

VEDOUCÍ PRÁCE **prof. RNDr. ALEXANDER MEDUNA, CSc.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Švač Dominik**
Program: Informační technologie
Název: **Dvoudimensionální konečné automaty a jejich aplikace**
Two-Dimentional Finite Automata and Their Applications
Kategorie: Teoretická informatika

Zadání:

1. Seznamte se s teorií n-dimensionálních konečných automatů dle pokynů vedoucího.
2. Zaveďte nový typ dvou-dimensionálních konečných automatů dle pokynů vedoucího.
3. Dle pokynů vedoucího studujte vlastnosti automatů z bodu 2.
4. Dle pokynů vedoucího aplikujte automaty z bodu 2 ve vhodné oblasti informatiky, např. v počítačové grafice. Implementujte a testujte tuto aplikaci.
5. Diskutujte další vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1
- Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition), Pearson Education, 2006, ISBN 0-321-48681-1

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Meduna Alexander, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Cieľom tejto bakalárskej práce je zadefinovať nový typ dvojrozmerného konečného automatu, ktorý je schopný rozpoznať objekty zo vstupného obrázka. Automat je zameraný na rozpoznávanie dopravných značiek, konkrétne rýchlostné obmedzenia na rýchlosť 50,70 a 80 km/h. Pozostáva z troch častí. Spúšťajú sa postupne. Výsledkom je množina nájdených značiek na obrázku.

Abstract

The goal of this bachelor thesis is to define a new type of two-dimensional finite state automata that is able to recognize objects from the input image. Automata is focused on the recognition of traffic signs, specifically speed limits for speeds of 50, 70 and 80 km / h. It consists of three parts. All parts run sequentially. The result is a set of marks found in the image.

Kľúčové slová

automat, dvojrozmerný konečný automat, dopravné zančky, C++, metódy rozpoznávania objektov

Keywords

automata, two-dimensional finite automata, traffic signs, C ++, object recognition methods

Citácia

ŠVAČ, Dominik. *Dvoudimensionální konečné automaty a jejich aplikace.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexander Meduna, CSc.

Dvoudimensionální konečné automaty a jejich aplikace

Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana RNDr.Alexander Meduna,CSc. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Dominik Švač
21. mája 2020

Poďakovanie

Veľmi rád by som poďakoval pánovi RNDr.Alexandrovi Medunovi,CSc za cenné rady, pomoc a poskytnutý čas, ktorý mi venoval pri riešení tejto práce.

Obsah

Kapitola 1

Úvod a cieľ

Existuje veľa rôznych spôsobov rozpoznávania objektov z obrázka. Každý je špecifický. V tejto bakalárskej práci sa čitateľ dozvie o novom spôsobe rozpoznávania objektov na základe dvojrozmerných konečných automatov. Objekty môžu mať ľubovoľný tvar a veľkosť. Na jednom obrázku ich môže byť aj viac.

Konečný automat prijíma na vstupe reťazec. Prechádza znak po znaku a číta ho. Pomocou prechodovej funkcie, ktorá na základe aktuálneho stavu a prečítaného znaku vchádza do ďalšieho stavu. Hlavný rozdiel medzi takýmto automatom a dvojrozmerným je ten, že dvojrozmerný konečný automat má na vstupe dvojrozmerný reťazec (obrázok). Môže sa pohybovať po vstupe nielen dopredu, ale aj inými smermi. Rozpoznávanie sa začína sa v počiatočnom stave. Na základe pravidiel sa pohybuje, môže vstupovať do ďalších stavov a vykonávať rôzne činnosti. Koniec nastáva vstupom do stavu, ktorý patrí do množiny koncových stavov. Končí úspechom alebo neúspechom, podľa toho, či niečo našiel, alebo nie.

Existuje veľa typov takýchto konečných automatov, ktoré umožňujú rozpoznávanie. Líšia sa spôsobom akým prechádzajú vstup a pravidlami, ktoré dodržujú. O niektorých z nich sa čitateľ oboznámi aj v tejto práci. Sú to napríklad štvorcestné, on-line teselačné, alebo celulárne automaty.

Na základe nich je vytvorený nový typ dvojrozmerného konečného automatu. V nasledujúcich kapitolách je vysvetlené, ako pracuje, funguje a ako sa pohybuje po vstupnom obrázku. Aké využíva prechodové funkcie v jednotlivých jeho častiach.

Je veľa oblastí, kde by sa dal využiť tento automat a objektov, ktoré by mohol rozpoznať. Pre vhodnú ukážku toho ako funguje, zameriame sa na dopravu, presnejšie na dopravné značky. Avšak tých je veľa a pre vysvetlenie a znázornenie nám bude stačiť, ak budeme v celom návrhu automatu a implementácie zameraný na rozpoznávanie kruhových dopravných značiek, ktoré obmedzujú maximálnu povolenú rýchlosť na 50,70 a 80 km/h. Využitie v praxi by mohlo byť napríklad v kamerách do auta, ktoré by pomáhalo vodičom a upozorňovalo by ich na rýchlostné obmedzenia na cestách.

Základ nového typu automatu spočíva v tom, že je rozdelený do troch častí. V prvej časti automat vyhľadáva začiatky potencionálnych objektov, ktoré dokáže rozpoznať. Ako ďalšia nasleduje časť, kde sa pohybuje automat po okrajoch objektu. Kontroluje tak jeho tvar. Ak zistí, že našiel potencionálny tvar značky, ktorý by sedel do množiny značiek, ktoré automat podporuje, tak sa spustí posledná časť. Tá kontroluje vnútro automatu a jej výsledkom je dopravná značka, ktorú automat našiel. Všetky tieto časti sa postupne opakujú kým sa neprejde celý vstupný obrázok.

V druhej kapitole tejto bakalárskej práce sú opísané základné informácie a definície ohľadom konečných automatoch. Opísaná je tu aj teória k rôznym automatom, z ktorých je čerpaná inšpirácia, pomocou ktorej vznikol tento nový typ automatu. Pri každom vybratom automate sa nachádza aj konkrétny príklad. Čitateľ tak lepšie pochopí jeho princíp a fungovanie.

Tretia kapitola je zameraná na opis nového typu automatu. Vysvetlené sú tu formálne definície a základný princíp fungovania automatu. Ako sa pridáva nový objekt, ktorý by mohol automat podporovať. Na konci sa nachádza aj teoretické vysvetlenie princípu jednotlivých častí, z ktorých pozostáva.

Vo štvrtej kapitole sa čitateľ dozvie, ako sú implementačne vyriešené jednotlivé časti. Sú tu opísané funkcie, ktoré sa využívali. Nachádza sa tu aj stručný prehľad použitých prostriedkov.

Piata kapitola je o testovaní, experimentoch a vhodnom použití do oblasti reálneho sveta, poprípade možných vylepšení a optimalizácii. Predstavené sú tu aj niektoré už existujúce spôsoby vyhľadávania a rozpoznávania objektov. Medzi najznámejšie patria metódy strojového učenia.

V poslednej šiestej kapitole je záver, v ktorom sú zhrnuté všetky podstatné veci a výsledok práce.

Cieľom tejto práce je ukázať aj iný pohľad na rozpoznávanie objektov pomocou dvojrozmerných automatov. Aj keď existujú rôzne spôsoby rozpoznávania, každý je špecifický a vhodný pri inej situácii. Tento nový typ automatu, ktorý je opísaný v práci, by mohol rozpoznávať určité objekty rýchlejšie ako ostatné.

Kapitola 2

Konečné automaty

Cieľom tejto kapitoly je stručne vysvetliť a oboznámiť čitateľa ohľadom základnej teórie k dvojrozmerným jazykom a stavovým automatom. Zovšeobecnenie formálnych jazykov do dvoch dimenzií je možné viacerými spôsobmi. My sa tu zameriame hlavne na jazyky, ktoré sú definované konečnými stavovými automatmi.

Opísané sú tu aj niektoré známe druhy dvojrozmerných konečných automatov. Na vstup dostanú obrázok. Pomocou prechodových funkcií sa posúvajú po stavoch, až kým nenarazia na koncový stav.

2.1 Konečný stavový automat

Konečné automaty sú jedným zo základných prostriedkov, ktoré nám umožňujú popis regulárnych jazykov. Základné dva modely sú deterministický konečný automat a nedeterministický konečný automat.

Na začiatku sa automat nachádza v počiatočnom stave. V každom kroku sa prečíta jeden symbol zo vstupu a prejde sa do stavu na základe výsledku prechodovej funkcie. Parametrom takejto funkcie je aktuálny stav a prečítaný symbol. Koniec nastáva v prípade, že sa vstúpi do stavu, ktorý patrí do množiny koncových stavov. Množina všetkých reťazcov, ktorý automat príjme, vytvára regulárny jazyk [9].

2.1.1 Deterministický konečný automat

Deterministický konečný automat [9] (anglicky: Deterministic finit automaton DFA). Je to pätica (Q,Σ,δ,q_0,F),ktorá pozostáva z nasledujúcich častí:

- Konečná množina stavov, často označovaná ako Q
- Konečná množina vstupných symbolov, často označovaná ako Σ
- Prechodová funkcia (Transition function), ktorá berie ako argumenty stav a vstupný symbol a výsledkom je ďalší stav. Prechodná funkcia je najčastejšie označovaná ako δ.
- Počiatočný stav q_0, ktorý je z množiny Q.
- Množina koncových stavov alebo prijatých stavov, označovaná ako F. Kde F je podmnožina Q.

Diagram prechodov

Diagram prechodov [9] (Transition diagram) je graf, ktorý je definovaný nasledovne:

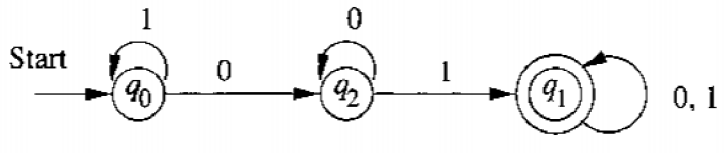
- Pre každý stav z množiny Q existuje uzol.
- Každý stav q patrí do množiny Q, a každý vstupný symbol a patrí do množiny Σ. Existuje prechodová funkcia δ(q,a) = p. Potom existuje prechod z uzla q do uzla p označený ako a. Ak existuje viac vstupných symbolov, ktoré vytvárajú prechod z q do p, potom má prechodový diagram jeden prechod označený zoznamov všetkých týchto symbolov.
- Prechodový diagram obsahuje aj šípku, ktorá vchádza do počiatočného uzlu q_0. Je označená ako Štart a nevychádza zo žiadneho uzla.
- Koncové stavy sú označené dvojitým kruhom a sú z množiny F. Stavy, ktoré nie sú z množiny F, sa značia jedným kruhom.

Príklad 2.1.1

Máme automat DFA, ktorý prijíma reťazce, ktoré obsahujú iba znaky 0 a 1 a musí obsahovať postupnosť 01 niekde v reťazci. Formálne môžeme zapísať tento jazyk ako:

L = { w | w = x01y, x a y pozostávajú iba z 0 a 1}

Obsahuje stavy q_0, q_1, q_2, kde q_0 je počiatočný stav a q_1 je koncový stav [9].



Obr. 2.1: [9]

2.1.2 Nedeterministický konečný automat

Nedeterministický konečný automat [9] (anglicky: Nondeterministic finit automaton NFA). Podobne ako DFA aj NFA má konečnú množinu stavov, konečnú množinu symbolov, počiatočný stav a množinu koncových stavov. Rozdiel medzi nimi je v prechodovej funkcii δ.

Prechodová funkcia pre NFA má ako argumenty stav a vstupný symbol. Výsledkom môže byť jeden stav, žiaden, alebo aj viac stavov.

Definícia 2.1.1

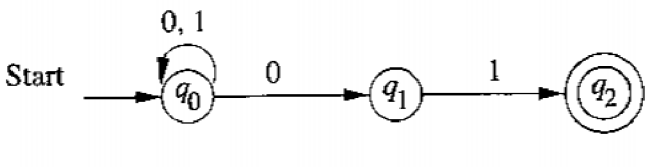
NFA je pätica (Q,Σ,δ,q_0,F) [9] kde:

- Q je konečná množina stavov
- Σ je konečná množina vstupných symbolov
- q_0 je počiatočný stav, kde $q_0 \in$ Q

- F je konečná množina koncových stavov, pričom F $\subseteq$ Q
- δ je prechodová funkcia, ktorá berie stav z Q a vstupný symbol z Σ ako vstupy a výsledkom je podmnožina množiny Q.

Príklad 2.1.2

Máme automat NFA, ktorý spracováva reťazce obsahujúce iba 0 a 1. Reťazec končí postupnosťou 01. Začína v stave q_0. Ak bude v tomto stave na vstupe 0, nevieme určiť, či je to znak, ktorým sa začína koniec reťazca, alebo nie. Preto sú v tomto stave dva prechody, jeden naspäť do seba a druhý do stavu q_1 [9].



Obr. 2.2: [9]

2.2 Základné definície obrázka

Definícia obrázka 2.2.1

Dvojrozmerný reťazec (alebo obrázok) nad vstupnou abecedou Σ je dvojrozmerné obdĺžnikové pole prvkov Σ. Množina všetkých týchto reťazcov nad Σ je označená ako Σ^{**}. Dvojrozmerný jazyk nad Σ je podmnožina Σ^{**} [16].

Definícia veľkosti obrázka 2.2.2

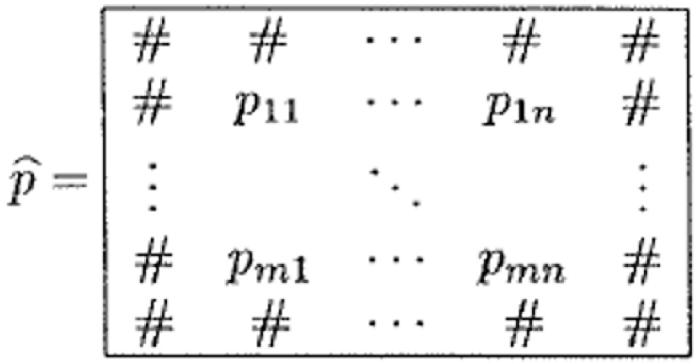
Nech existuje obrázok p $\in \Sigma^{**}$, potom l_1 (p) označíme ako číslo riadkov a l_2 (p) označíme ako číslo stĺpcov obrázka p. Dvojica (l_1 (p), l_2 (p)) je označená ako veľkosť obrázka p. Prázdny obrázok má veľkosť (0,0) a je označený ako λ. Obrázky, ktoré majú veľkosť (0,n) alebo (n,0) kde n > 0 nie sú definované. Množina všetkých obrázkov nad Σ o veľkosti (m,n), kde m,n > 0 je označovaná ako $\Sigma^{m\times n}$. Avšak v prípade, že 1 < i < l_1(p) a 1 < i < l_2(p), potom označenie p(i,j) alebo p_{ij} označuje súradnice (i,j) obrázka p [16].

Definícia bloku 2.2.3

Nech je p obrázok s veľkosťou (m,n). Blok (podobrázok) z obrázka p sa značí p'. Veľkosť obrázka p' označme (m',n'). Musí platiť, že m' < m a súčasne n'< n, potom existujú čísla h,k (h < m-m', k < n-n') tak že p'(i,j) = p (i+h,j+k). Pre všetky i,j musí platiť 0 < i < m' a 0 < j < n' [16].

Definícia obrázka s hranicami 2.2.4

Hranice obrázka [16] identifikujeme použitím špeciálneho symbolu. Pre každý obrázok p veľkosti (m,n) definujeme obrázok $\hat{p}$, ktorý bude mať veľkosť (m+2,n+2). Ten je získaný obklopením obrázka p špeciálnym hraničným symbolom #, ktorý nepatrí do Σ. # $\notin \Sigma$. Obrázok $\hat{p}$ s definovanými hranicami je znázornený na nasledujúcom obrázku 2.3.



Obr. 2.3: [16]

2.3 Existujúce automaty

V tejto časti sa zameriame na viac druhov automatov, ktoré čítajú dvojrozmernú pásku. Prvý sa volá „štvorcestný automat“ (anglicky: four-way automaton). Je to sekvenčný automat na rozdiel od druhého, ktorý je špecifickým typom celulárneho (bukového) automatu „dvojrozmerný on-line teselačný automat“ (anglicky: on-line tesselation automaton). Predstavíme si aj základy celulárneho automatu. Ako príklad si uvedieme hru Life, ktorá je najznámejším typom tohto automatu.

2.3.1 Štvorcestný automat

Jedna z prvých definícii na rozpoznávania obrázkových jazykov bola popísaná M.Blum a C. Hewitt. Oni v roku 1967 predstavili model konečného automatu, ktorý číta dvojrozmernú pásku. Štvorcestný konečný automat [16] je definovaný ako rozšírenie dvojcestného automatu, ktoré rozpoznáva reťazce riadením pohybu do štyroch smerov: Vľavo, vpravo, hore a dole. Formálne opísaný tento automat je nižšie [16].

Definícia 2.3.1

Nedeterministický (Deterministický) štvorcestný konečný automat [16] označovaný ako 4NFA (DFA) je sedmica $(\Sigma, Q, \Delta, q_0, q_a, q_r, \delta)$ kde

- Σ je vstupná abeceda
- Q je konečná množina všetkých stavov
- Δ je množina povolených smerov {L,R,U,D}
- q_0 je počiatočný stav, $q_0 \in$ Q
- q_a, q_r sú „accepting“(prijaté) a „rejecting“(odmietnuté) stavy, $q_a, q_r \in$ Q
- δ : Q $\setminus \{q_a, q_r\} \times \Sigma \to 2^{Q \times \Delta}$ ($\delta : Q \setminus \{q_a, q_r\} \times \Sigma \to Q \times \Delta$) je prechodná funkcia

Netreba tu špecifikovať, či ide o determinizmus alebo nedeterminizmus. Štvorcestný automat označujeme ako 4FA. Ak sa nachádza 4FA v stave z množiny Q, číta vstupný obrázok. Riadenie pohybov závisí na prechodných funkciách δ, ktoré sú dané aktuálnym

stavom a symbolom na aktuálnej pozícií. Výsledok nového stavu pohybu je nová pozícia, ktorá sa posunie o jednu polohu podľa smeru výstupu. V prípade, že sa pohybom dostane do stavu q_a, alebo q_r, 4FA, sa pozastaví. Pretože v týchto stavoch nie sú definované prechodné funkcie. Používame konvenciu takú, že 4FA prečíta obrázok s hranicami $\hat{p}$, a keď sa pohne na pozíciu, ktorá obsahuje symbol #, vráti sa naspäť pri nasledujúcom ťahu. Rovnako by sa dalo povedať, že riadenie je citlivé na hranice. Riadenie vie, kedy sa k hraniciam blíži a nikdy neprejde za hranice mimo obrázka p [16].

4FA rozpoznáva obrázok p $\in \Sigma^{**}$. Ak začína na pozícii (1,1) a nachádza sa v počiatočnom stave, jeho možné ťahy sú pohnutie sa niekde na miesto v okolí. Nakoniec sa zastaví v prijatom stave q_a. Všimnite si, že počas procesu rozpoznávania, 4FA nie je nútený, aby prešiel a prečítal všetky pozície vstupného obrázka. Avšak sa môže vracať na danú pozíciu toľkokrát, koľko je to potrebné [16].

Príklad 2.3.1:

Nech $\Sigma = \{0,1\}$ je abeceda a nech L $\subseteq \Sigma^{**}$ je jazyk obrázkov, ktorého prvý stĺpec je rovnaký ako posledný stĺpec. Potom L je rozpoznaný 4DFA. Prehľadáva obrázok p riadok po riadku, zľava doprava a postupuje smerom zhora nadol. Zároveň kontroluje v každom čase, či všetky pozície obsahujú znak (symbol) z abecedy Σ. Znak prvý zľava v riadku musí byť rovnaký ako znak posledný sprava [16].

Jazyk L je formálne definovaný ako: L = { p | p(i,1) = p(i,$l_2(p)$), i = 1,2 ,..., $l_2(p)$ }

Príklad 2.3.2:

Nech $\Sigma = \{0,1\}$ je abeceda a nech L $\subseteq \Sigma^{**}$ je jazyk štvorcov. Potom jazyk L môže byť rozpoznaný 4DFA. Prehľadáva obrázok p. Začína na pozícii (1,1) a pohybuje sa po diagonále. Hýbe sa tak, že urobí jeden krok doprava a jeden krok dole. Ak prekročí spodný pravý roh, potom je obrázok p štvorec. V tomto prípade kompletne prehľadá celý obrázok p a skontroluje, či všetky pozície obsahujú znak (symbol) z abecedy Σ [16].

Jazyk L je formálne definovaný ako: L = {p | p $\in \Sigma^{**}$ a l_1 (p) = l_2 (p)}.

Teraz uvažujme, že jazyk $L_1 \subseteq \Sigma^{**}$. Pozostáva zo štvorcov nepárnych dĺžok strán s „1“ na pozícii v strede. Jazyk L_1 môže byť rozpoznaný 2NFA. Máme obrázok p. Prehľadávanie sa začína na pozícii (1,1), pohybuje sa po diagonále. V nejakom bode sa uplatní nedeterminizmus, automat si „pamätá“ znak (symbol) na diagonále a začne sa pohybovať dole po druhej diagonále. Pohybuje sa raz doľava a v druhom kroku dole. Ak narazí na spodný ľavý roh, potom obrázok p je štvorec s nepárnou dĺžkou strany. Jeho stredná pozícia je v zapamätanom znaku. Ak zapamätaný znak bol „1“, tak automat skončil úspešne, v opačnom prípade automat bol neúspešný [16].

2.3.2 On-line Tesselačný automat

Tento typ automatu je podobný celulárnym automatom. Hovoríme tu o konkrétnom modely dvojrozmernom celulárnom automate (2CA), ktorý sa volá on-line teselačný automat (2OTA). Bol predstavený K. Inoue a A. Nakamura.

Obsahuje pole buniek. Každá je v každom čase v nejakom stave. Bunky 2OTA nevytvárajú prechody v každom kroku, ale vytvárajú ich v takzvaných „prechodových vlnách“.

Prechádzajú diagonálne naprieč celým poľom. Každá bunka mení svoj stav v závislosti na dvoch susedov. Jeden zhora a ďalší zľava [16].

Definícia 2.3.2

Nedeterministický (Deterministický) dvojrozmerný on-line teselačný automat [16] označovaný ako 2OTA (2-DOTA) je kompletne definovaný ako pätica (Σ,Q,q_0,F,δ) kde :

- Σ je vstupná abeceda
- Q je konečná množina všetkých stavov
- I $\subseteq$ Q (I = {i} $\subseteq$ Q) je množina počiatočných stavov
- F $\subseteq$ Q, je množina konečných (povolených) stavov
- δ : Q $\times$ Q $\times$ $\Sigma \rightarrow 2^Q$ (δ : Q $\times$ Q $\times\Sigma \rightarrow$ Q) je prechodná funkcia

Priebeh automatu 2OTA na obrázku p kde p $\in \Sigma^{**}$ pozostáva zo stavov (ktoré sú z množiny Q) pre všetky pozície (i,j) z obrázka p. Také stavy sú dané prechodovou funkciou δ a závisia na stavoch na pozíciách (i-1,j) a (i, j-1) a na symbole p(i,j).

#	#	#	#	#	#	#	#
#							#
#							#
#					$p(i-1,j)$		#
#				$p(i,j-1)$	$p(i,j)$		#
#							#
#	#	#	#	#	#	#	#

Obr. 2.4: [16]

V čase time = 0 počiatočný stav q_0 je priradený na všetky pozície prvého riadka a prvého stĺpca z obrázka $\hat{p}$. 2OTA pozostáva z $l_1(p) + l_2(p)$ -1 krokov. Začína sa v čase time = 1 prečítaním p(1,1) priradením stavu $\delta(q_0, q_0, p(1,1))$ na pozíciu (1,1). V čase time = 2 sú stavy priradené na pozície (1,2) a (2,1) a podobne na ďalšie diagonály. V čase time = k sú stavy priradené na všetky pozície (i,j), tak že k = i + j – 1. Automat 2OTA rozpoznáva obrázok p dovtedy, kým nie je stav priradený v koncovom stave na pozícii $(l_1(p), l_2(p))$ [16].

Príklad 2.3.3:

Nech existuje Σ = {a} a nech L je podmnožina Σ^{**}. L je jazyk všetkých obrázkov nad Σ s nepárnymi číslami stĺpcov. Platí, že L = {p | $l_2(p)$ je nepárne číslo}. 2OTA môže rozpoznávať obrázok L stavmi „1“ a „2“ na pozície všetkých párnych a nepárnych stĺpcov. Obrázok je rozpoznaný, ak pozícia najpravejšieho stĺpca obsahuje stav „1“. Viac formálne, L je rozpoznaný automatom 2OTA definovaný nasledovne [16]:

- Q = {0,1,2}

- I = {0}
- F = {1}
- δ(0,0,a) = δ(0,2,a) = δ(1,0,a) = δ(1,2,a) = 1
- δ(0,1,a) = δ(2,1,a) = 2

Príklad 2.3.4:

Nech existuje Σ = {a} a nech L je podmnožina Σ^{**}. L je jazyk všetkých štvorcov nad Σ. Počet riadkov a stĺpcov sa musí rovnať. L = {p | p $\in \Sigma^{**}$ a $l_1(p) = l_2(p)$}. 2OTA dokáže rozpoznať obrázok L pomocou stavu „1“ na pozícii v hlavnej diagonále a stavu „2“ a „3“ na pozícii pod alebo nad diagonálou. Obrázok je rozpoznaný, ak pozícia spodného pravého rohu obsahuje stav „1“. Viac formálne, L je rozpoznaný automatom 2OTA definovaný nasledovne [16]:

- Q = {0,1,2,3}
- I = {0}
- F = {1}
- δ(0,0,a) = δ(2,3,a) = 1
- δ(0,1,a) = δ(0,2,a) = δ(2,1,a) = δ(2,2,a) = 2
- δ(1,0,a) = δ(3,0,a) = δ(1,3,a) = δ(3,3,a) = 3

2.3.3 Celulárny automat

Celulárne automaty (CA po anglicky Celullar Automaton) sú diskrétne systémy. Skladajú sa z jednoduchých komponentov (buniek). Tie sú väčšinou identické. Každá bunka má svoj stav, ktorý sa mení v čase. Stav buniek CA je možné počítať v jednotlivých krokoch v diskrétnom čase pomocou pravidiel, ktoré počítajú nasledujúci stav zo stavu buniek a jeho okolia.

Existujú rôzne varianty CA. Líšia sa inou definíciou pravidiel, okolia. Použitie siaha od jednoduchých hier (Life) cez simuláciu dopravy, šírenie epidémií, chemických reakcií, rast kryštálov, až po modely umelého života [17].

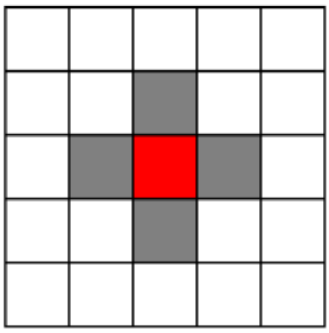
Definícia 2.3.3

CA je štvorica A = (d,S,N,f). Je zložená z nasledujúcich zložiek [17]:

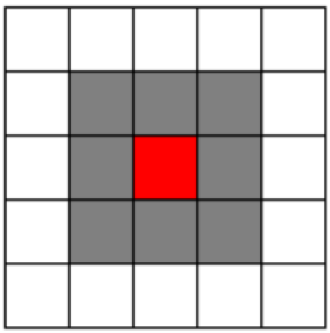
- Dimenzia d - Pole buniek je všeobecne n-rozmerné, obvykle 1D alebo 2D, môže byť konečné alebo nekonečné, všetky bunky sú rovnaké.
- Konečná množina stavov S.
- Susedstvo N (the Neighbourhood), počet a pozícia okolitých buniek, s ktorými bunka pracuje
- Pravidla f, ktoré popisujú chovanie bunky v čase.

Štruktúra buniek je najčastejšie definovaná ako n-dimenzionálna mriežka. Typicky je to pre n = 2. Často sa používajú susedstvá, von Neumann a Moore.

Von Neumann susedstvo obsahuje 4 okolité bunky a susedstvo Moore obsahuje 8 okolitých buniek. Na základe nich sa vypočíta nový stav aktuálnej bunky [17].



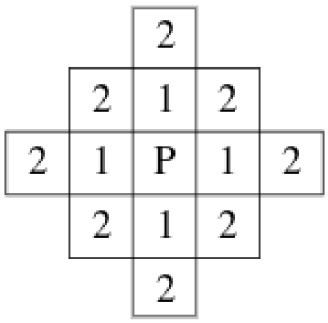
Obr. 2.5: Von Neumannove susedstvo [17]



Obr. 2.6: Moore susedstvo [17]

Na obrázku 2.5 sa nachádza Von Neumannove susedstvo a na obrázku 2.6 sa nachádza Moore susedstvo. Červená bunka je aktuálna bunka a okolo nej sú sivé, ktoré sa nachádzajú v jej susedstve.

Existuje aj rozšírená verzia Von Neumannoveho susedstva, kedy je vzdialenosť od aktuálnej bunky väčšia ako jedna, r > 1. Výsledok okolitých susedov pripomína diamantový tvar. Počet buniek, ktoré sú v okolí sa dá vypočítať pomocou vzorca 1 + 2r(r+1). Na obrázku 2.7 možme vidieť susedstvo tejto rozšírenej verzie [18].



Obr. 2.7:

Príklad 2.3.5: Hra život

Najznámejším príkladom pre celulárne automaty je hra Life [1]. Hru život (anglicky: game of Life) vynašiel John Horton Conway. Na začiatku tejto hry sa určí počiatočná podmienka a hra pokračuje sama.

Automat pri tejto hre je charakteristickým tým, že obsahuje stavy a jednoduché pravidlá. Stavy poukazujú na bunku, či žije, alebo nie. Pravidlá menia stav bunky na základe

stavov okolitých buniek, susedov. Tie sa zisťujú na základe mooroveho susedstva, ktoré obsahuje 8 buniek, ako je to na obrázku 2.6

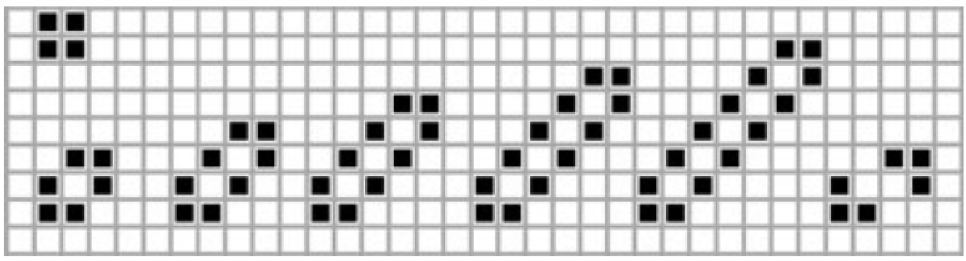
Každá bunka sa riadi nasledujúcimi pravidlami [1]:

Pravidlá:

- Každá živá bunka, ktorá má menej ako dvoch živých susedov, zomrie.
- Každá živá bunka s dvoma, alebo troma živými susedmi ostáva žiť.
- Každá živá bunka s viac ako troma živými susedmi zomrie.
- Každá mŕtva bunka, ktorá má práve troch živých susedov, ožije.

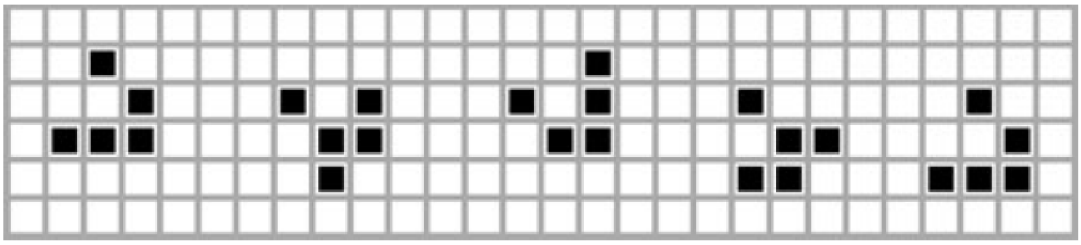
V tejto hre existujú vzory, ktoré sa dajú rozdeliť do kategórií podľa toho ako sa správajú v priebehu času, v každom kroku. Kategórii je veľa. Nasledujúce skupiny, ktoré sú tu opísané, sú len výberom niektorých zaujímavých [1].

- Nemenné (Still Life) [1] – je to skupina vzorov ktoré obsahujú tvary, ktoré sa z časom nemenia. Sú stabilné. Obsahujú konečný počet živých buniek.
- Oscilátor (Oscillators) [1] – sú to tvary, ktoré menia svoj tvar. Sú nestabilné, menia sa s časom, ale vždy po určitom počte krokov sa vrátia do svojho pôvodného tvaru, v ktorom začínali. Na rozdiel od ďalšej skupiny (Posúvajúce sa vzory) sa neposúvajú, pôvodný tvar sa objaví vždy na rovnakom mieste.
- Posúvajúce sa vzory [1] – je skupina vzorov, ktoré sa pohybujú. Objavujú sa po určitom čase v nezmenenej podobe na inom mieste. Jedným z najznámejších prípadov takéhoto vzorca je Krídlo (Glider).
- Delá (Glider Gun) [1] – sám o sebe je takýto vzor delo, stabilný. V určitých časových intervaloch generuje posúvajúce sa vzory.



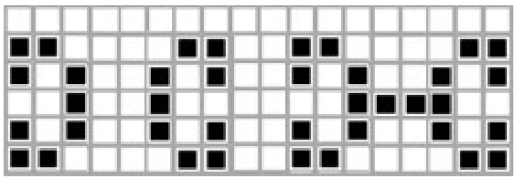
Obr. 2.8: [1]

Na obrázku 2.8 je zopár tvarov, ktoré patria do množiny Nemenné (Still Life). V priebehu celého času hry ostáva ich tvar rovnaký, nemení sa.



Obr. 2.9: [1]

Obrázok 2.9 zobrazuje pohyb tvaru s názvom Krídlo (Glider). Obsahuje 5 tvarov. Prvý a posledný je rovnaký. Krídlo opakuje svoj tvar v každom štvrtom kroku. Každú piatu časovú jednotku tak posunie o jednu pozíciu smerom doprava a dole. Chodí po diagonále.



Obr. 2.10: [1]

Na obrázku 2.10 sa nachádza ukážka tvaru z kategórie Oscilátor (Oscillators). V každom druhom kroku sa dostáva naspäť do svojho pôvodného tvaru.

Kapitola 3

Nový typ dvojdimenzionálneho automatu

Nový typ dvojrozmerného deterministické konečného automatu je schopný rozpoznať obrázok zo vstupu. Presnejšie povedané, dokáže určiť a pomenovať objekty, ktoré sa na vstupnom obrázku nachádzajú. Pre zjednodušenie sa zameriame iba na objekty, ktoré reprezentujú kruhové dopravné značky.

Táto kapitola je rozdelená do dvoch hlavných častí. Prvá je zameraná na všeobecné definície automatu a množiny α a β, ktoré reprezentujú objekty. Na základe nich je schopný automat rozpoznávať jednotlivé objekty, v našom prípade kruhové dopravné značky. V druhej časti je opísané a vysvetlené ako automat funguje.

3.1 Formálne definície

Tento automat prijíma na vstup dvojrozmerný obrázok. Na začiatku ho prevedie do formátu, s ktorým bude pracovať. Každý bod vstupného obrázka je určený jeho farbou a hodnotou 0, ktorá reprezentuje, že daný bod ešte nebol spracovaný.

Rozpoznávanie začína v počiatočnom stave a postupne prechádza celý obrázok, kým sa nedostane do koncového stavu. Výsledkom je množina objektov, ktoré sa našli v obrázku.

3.1.1 Definícia

Formálna definícia pre všetky časti automatu. Automat A = $(\Sigma, \alpha, \beta, Q, \Delta, F, q_0, q_r, \delta)$

- Σ je vstupná abeceda
- α je množina vodorovných reťazcov objektu.
- β je množina zvislých reťazcov objektu.
- Q je konečná množina všetkých stavov
- Δ je množina povolených smerov {L,R,U,D}
- F $\subseteq$ Q, je množina konečných (povolených) stavov, v ktorých automat našiel riešenie.
- q_0 je počiatočný stav, $q_0 \in$ Q
- q_r je koncový stav, v ktorom automat nenašiel riešenie „rejecting“, $q_r \in$ Q

- δ je prechodová funkcia, kde $\delta(Q,\Sigma,\Delta) \rightarrow Q$

Vstupná abeceda Σ

Vstupnú abecedu predstavuje obrázok, ktorý sa snažíme rozpoznať. Je tvorená dvoma znakmi, farbou a hodnotou 0, alebo , ktorá značí či automat už prešiel cez daný znak, alebo nie. Na začiatku sú všetky hodnoty nastavené na 0.

Na rozpoznávanie si určíme značky, ktoré budú reprezentovať farbu pixelu d, d $\in \Sigma$.

- B - Čierna (Black)
- W- Biela (White)
- R – Červená (Red)
- X – Ostatné farby
- # - Hranice

Σ = {B0, W0, R0, X0, #0, B1, W1, R1, X1, #1}

Množiny α a β

Sú množiny vodorovných a zvislých reťazcov objektu, ktorý vyhľadávame. Sú tvorené značkami farieb, ktoré sa nachádzajú vo vstupnej abecede. Každý znak farby reprezentuje postupnosť pixelov rovnakej farby na riadku (pre množinu α) alebo stĺpci (pre množinu β). Pre každý riadok, poprípade stĺpec, sa vytvorí takýto reťazec. Ak sa dva reťazce v množine idúce po sebe rovnajú, vymaže sa ľubovoľný jeden z nich.

Príklad:
Chceme vytvoriť množiny reťazcov α a β pre znak 0. Predpokladajme, že všade okolo znaku je biele pozadie.

$$\alpha = \{ \text{W}, \text{WBW}, \text{WBWBW}, \text{WBW}, \text{W} \}$$

$$\beta = \{ \text{W}, \text{WBW}, \text{WBWBW}, \text{WBW}, \text{W} \}$$

Nad znakom je čisto biela farba a tak prvý reťazec je W. Nasleduje vrcho,l ktorého reťazec je WBW. Pokračuje sa stredom nuly WBWBW a na konci je opäť WBW a W.V tomto prípade je rovnaká množina aj pre stĺpce.

Množina Q

Sú to všetky stavy, cez ktoré automat prechádza, kým nenarazí na stav q_r, alebo q_a, pričom $q_a \in$ F. Každá časť automatu má iné stavy. q_0 je stav, v ktorom sa začína každá časť automatu. V stave q_a sa nachádza automat v prípade, že úspešne skončila aktuálna časť, ináč sa končí v stave q_r.

Množina Δ

Sú to všetky smery, do ktorých sa môže pohnúť automat. L značí pohyb doľava, R pohyb doprava, U pohyb hore a D pohyb dole. V prípade, ak sa nedá pohnúť do žiadneho smeru v okolí aktuálnej pozície, automat končí a to buď úspešne, alebo nie podľa toho, v ktorej časti sa nachádza a podľa definovania prechodových funkcií. Pri posune do smeru sa hodnota aktuálneho pixelu mení nasledovne:

$$\begin{gathered} \text{B0} \rightarrow \text{B1} \\ \text{R0} \rightarrow \text{R1} \\ \text{W0} \rightarrow \text{W1} \\ \text{X0} \rightarrow \text{X1} \end{gathered}$$

Prechodová funkcia δ

Sú to pravidlá na základe, ktorých sa bude vyberať nasledujúci stav podľa aktuálneho stavu, vstupného symbolu a smeru. Pravidlá sú v tvare:

d – Δ → q

Kde d ∈ Σ je nasledujúci pixel podľa smeru z množiny Δ. q je stav, do ktorého sa dostane. Smer z množiny Δ nesmie byť zakázaný.

3.1.2 Množiny α a β pre značky

V týchto reťazcoch sa nenachádzajú biele znaky, ktoré by boli ináč medzi každým znakom. Je to preto, aby sprehľadnili reťazce a neboli také dlhé.

Značka obmedzenie rýchlosti na 50km/h

$\alpha =$
{ R,
RR,
RBR,
RBBR,
RBBBR,
RBBBBR,
RBBBR,
RBBBBR,
RBBBR,
RBBBBR,
RBBR,
RR,
R }

$\beta =$
{ R,
RR,
RBBR,
RBBBR,
RBBR,
RR,
RBR,
RBBR,
RBR,
RR,
R }

Značka obmedzenie rýchlosti na 70km/h

.

$\alpha =$
{ R,
RR,
RBBR,
RBBBR,
RBBR,
RR,
R }

$\beta =$
{ R,
RR,
RBR,
RBBR,
RBR,
RR,
RBR,
RBBR,
RBR,
RR,
R }

Značka obmedzenie rýchlosti na 80km/h

$\alpha =$
{ R,
RR,
RBBR,
RBBBBR,
RBBBR,
RBBBBR,
RBBR,
RR,
R }

$\beta =$
{ R,
RR,
RBBR,
RBBBR,
RBBR,
RR,
RBR,
RBBR,
RBR,
RR,
R }

3.2 Automat

Automat je rozdelený do troch častí, ktoré spolu súvisia. Po úspešnom skončení jednej sa spúšta druhá. Ak skončí aj posledná časť úspešne, objekt bol rozpoznaný.

Prvá časť - Vyhľadávanie - hľadá bod rovnakej farby ako objekt, ktorý vyhľadávame.

Druhá časť - Kruhový automat - kontroluje obrysy objektu. V našom prípade je to kruh. Ak by bol objekt, ktorý chcem vyhľadať iný ako kruhový, táto časť by bola zmenená.

Posledná tretia časť - Vnútorný automat - prehľadáva vnútro objektu z predchádzajúcej časti. Výsledkom je rozpoznaný objekt. Na konci sa automat opäť vracia do prvej časti a hľadá sa ďalší objekt.

3.2.1 Vyhľadávanie

Je to prvá časť automatu, kde sa vyhľadávajú potencionálne objekty. Značka, ktorú sa snažíme rozpoznať, má červený kruh a vo vnútri je na bielom podklade napísané číslo čiernou farbou.

Automat sa pohybuje po riadkoch zhora dole, zľava doprava. Povolené sú tu len smery doprava a dole. Takto sa prejde celý vstup. Keď vstupný znak bude mať červenú farbu a ešte nebol spracovaný, tak sa spustí druhá časť automatu. Začína sa na pozícii (1,1)

Obmedzenia na pohyb

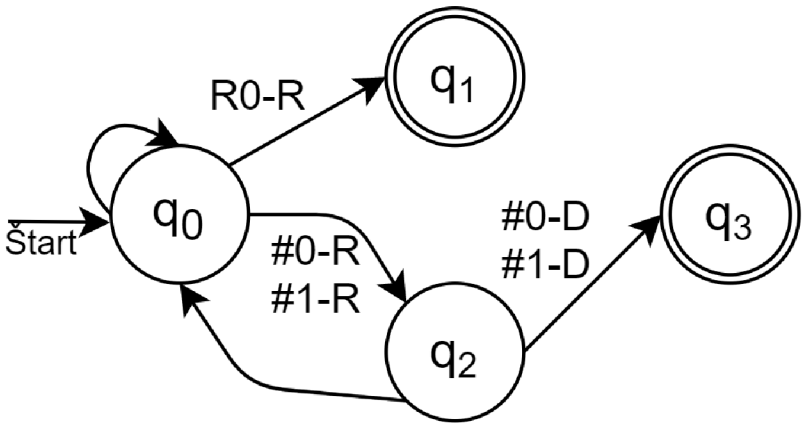
Pohyb po vstupnej abecede nie je v tejto časti obmedzený. Výnimkou sú iba symboly hraníc (#0, #1). Ak automat prečíta takýto pixel d, kde d $\in \Sigma$ tak sa automaticky presúva na pozíciu, ktorú definuje aktuálny stav.

Pravidlá na prechodové funkcie

Stav q_1	Stav q_2
B0-R $\rightarrow q_0$	B0-D $\rightarrow q_0$
W0-R $\rightarrow q_0$	W0-D $\rightarrow q_0$
X0-R $\rightarrow q_0$	X0-D $\rightarrow q_0$
R0-R $\rightarrow q_1$	R0-D $\rightarrow q_0$
#0-R $\rightarrow q_2$	#0-D $\rightarrow q_3$
B1-R $\rightarrow q_0$	B1-D $\rightarrow q_0$
W1-R $\rightarrow q_0$	W1-D $\rightarrow q_0$
X1-R $\rightarrow q_0$	X1-D $\rightarrow q_0$
R1-R $\rightarrow q_0$	R1-D $\rightarrow q_0$
#1-R $\rightarrow q_2$	#1-D $\rightarrow q_3$

Na základe týchto pravidiel je vytvorený aj diagram prechodov. q_0 je počiatočný stav. V stave q_1 sa zavolá druhá časť automatu, kde $q_1 \in$ do F. Do stavu q_2 sa dostane automat, keď narazí na hranice obrázka. Posunie sa na začiatok riadku. q_3 je stav, kde sa rozpoznávanie končí. Či sa objekt našiel závisí na ostatných častiach automatu, ktoré budú opísané v ďalších sekciách.

Diagram prechodov



Obr. 3.1:

3.2.2 Kruhový automat

V druhej časti automatu sa pohybuje automat do kruhu. Kontroluje či skúmaný objekt má okolo seba červený kruh.

Obmedzenia na pohyb

Ak pri pohybe automatu do určitej strany sa nasledujúci pixel d, kde d $\in \Sigma$ rovná symbolu hraníc (#0,#1), alebo symbolu (B1,W1,R1,X1,B0,W0,X0) ,tak potom tento pohyb je zakázaný. Povolené sú iba pohyby po pixeloch d, ktoré obsahujú symbol R0. Tým pádom sa v tejto časti posúva automat iba po červenej farbe.

Vysvetlenie pohybu

Pohyb po kruhu je rozdelený do štyroch častí. Každá z nich reprezentuje jednu štvrtinu kruhu. Líšia sa iba smermi, do ktorých sa pohybujú.

Každá štvrtina kruhu pozostáva z troch úsekov. Nasledujúce vysvetlenie sa bude týkať iba prvej štvrtiny, pri ktorej sa automat posúva smerom doprava a dole. Ostatné sú rovnaké, iba sa vymenia smery. V druhej časti sa posúva dole a doľava. V tretej časti doľava a hore a v poslednej hore a doprava.

Štvrtina z kruhu

Pri pohybe v prvom úseku sa kruh iba mierne posúva smerom dole. Prevláda pohyb doprava. V druhom úseku sa pohyby dole a doprava vyvažujú. V poslednej časti prevláda pohyb dole a iba mierne sa pohybuje doprava. Polkruh končí, keď sa dostaneme do dolnej časti pravej strany kruhu.

Pravidlá na prechodové funkcie

Stav q_0
R0-R $\rightarrow q_0$
R0-U $\rightarrow q_r$
R0-D $\rightarrow q_2$

Stav q_2
R0-R $\rightarrow q_3$
R0-D $\rightarrow q_4$

Stav q_3
R0-R $\rightarrow q_5$
R0-D $\rightarrow q_6$

Stav q_4
R0-R $\rightarrow q_3$
R0-D $\rightarrow q_r$

Stav q_5
R0-R $\rightarrow q_5$
R0-D $\rightarrow q_2$

Stav q_6
R0-R $\rightarrow q_3$
R0-D $\rightarrow q_7$

Stav q_7
R0-R $\rightarrow q_8$
R0-D $\rightarrow q_7$
R0-L $\rightarrow q_a$

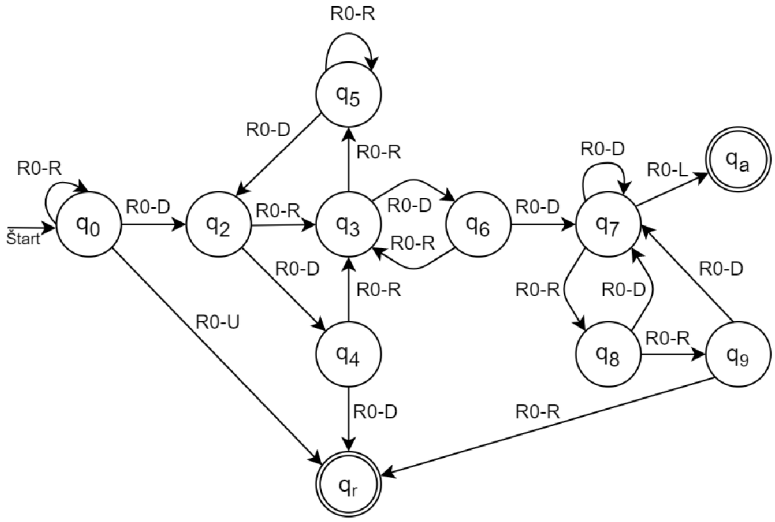
Stav q_8
R0-R $\rightarrow q_9$
R0->D $\rightarrow q_7$

Stav q_9
R0-R $\rightarrow q_r$
R0-D $\rightarrow q_7$

Fáza I.

V počiatočnom stave q_0 sa začína posúvanie tohto automatu. Chodí iba po pixeloch p, ktoré majú červenú farbu. Stavy q_2, q_3 a q_5 zabezpečujú pohybovanie v prvej fáze, keď sa iba mierne pohybuje kruh dole. Stav q_4 detekuje náhle posunutie dole, čo je v kruhu zakázané. Slučka so stavmi q_3 a q_6 sa vykonáva pri druhej fáze, kedy by sa mali striedať smery. V poslednej fáze sa väčšinou striedajú stavy q_7 a q_8. V tejto fáze by už nemal nastať prudký posun smerom doprava. Detekuje to stav q_9.

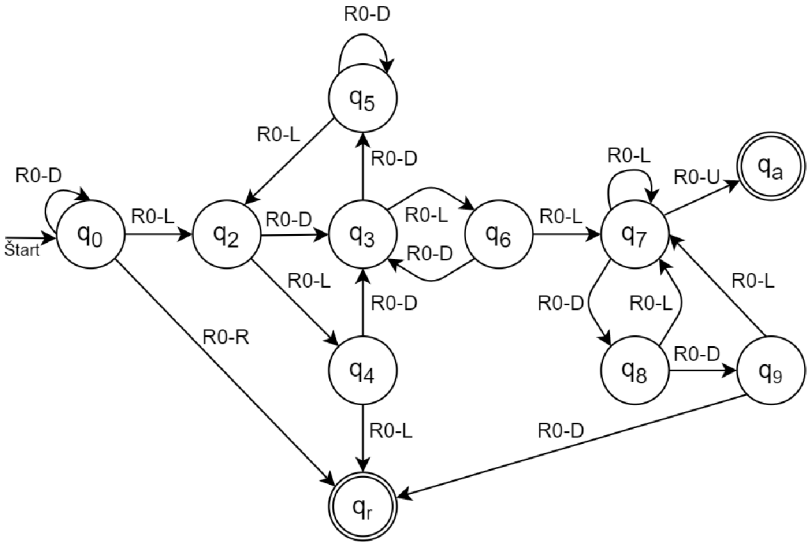
Koniec nastáva vtedy, ak sa v poslednej tretej časti automat dostane do stavu q_a, pričom $q_a \in$ F. A spustí sa ďalšia časť. Keď sa nedá aplikovať žiadne pravidlo, automat sa automaticky dostáva do stavu q_r ,v ktorom sa neúspešne končí rozpoznávanie a spúšťa sa opäť časť automatu Vyhľadávanie.



Obr. 3.2:

Fáza II.

V druhej fáze sa začína v pravom dolnom rohu a posúva sa smerom dole do ľavej časti kruhu. q_0 je počiatočný stav. Ak sa dostane automat do stavu q_a, úspešne sa rozpoznala aktuálna časť a začína sa tretia fáza. Keď sa nedá aplikovať žiadne pravidlo, automat sa automaticky dostáva do stavu q_r, v ktorom sa neúspešne končí rozpoznávanie a spúšťa sa opäť časť automatu Vyhľadávanie.

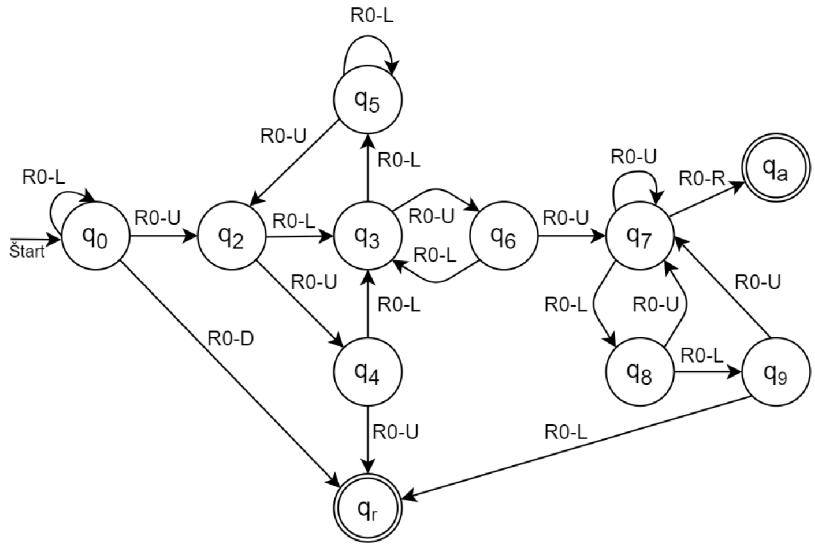


Obr. 3.3:

Fáza III.

Tretia fáza sa spúšťa z ľavého dolného rohu a postupuje hore, pričom sa drží na ľavej strane. V každom stave sa aplikuje prechodová funkcia. Pri dosiahnutí stavu q_a sa vstupuje do poslednej časti kruhu. Keď sa nedá aplikovať žiadne pravidlo, automat sa automaticky

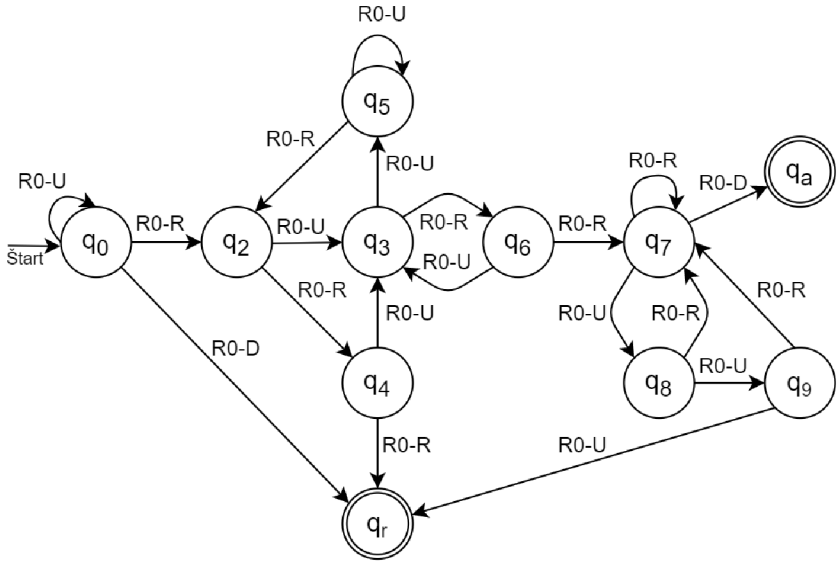
dostáva do stavu q_r ,v ktorom sa neúspešne končí rozpoznávanie a spúšťa sa opäť časť automatu Vyhľadávanie.



Obr. 3.4:

Fáza IV.

Štvrtá fáza začína v ľavom hornom rohu. Používajú sa prechodové funkcie, ktoré sú takmer rovnaké ako predchádzajúce, ale využívajú sa tu iné smery. Automat sa posúva smerom nahor. Pri úspešnom dosiahnutí stavu q_a sa definitívne potvrdilo, že objekt, ktorý sa rozpoznáva, obsahuje okolo seba červený kruh. Tým pádom sa táto časť automatu úspešne končí a začína sa posledná časť, v ktorom sa vyhodnotí vnútro kruhu. Keď sa nedá aplikovať žiadne pravidlo, automat sa automaticky dostáva do stavu q_r, v ktorom sa neúspešne končí rozpoznávanie a spúšťa sa opäť časť automatu Vyhľadávanie.



Obr. 3.5:

3.2.3 Vnútorný automat

Táto časť automatu je posledná. Predpokladá sa, že všetky časti pred touto skončili úspešne. Aktuálna pozícia sa nachádza vo vnútri kruhu, ktorú rozpoznala predchádzajúca časť.

Už vieme, že rozpoznávaný objekt obsahuje okolo seba červený kruh. Teraz si ukážeme ako sa spracováva vnútro a zistí sa konkrétna značka.

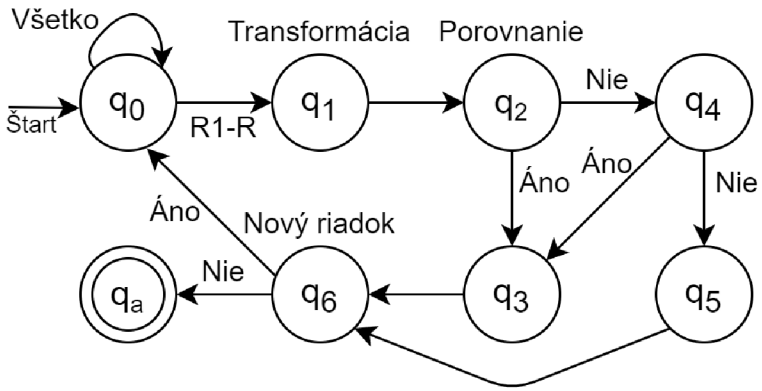
Aj táto časť je rozdelená na dve fázy. Prvá prechádza vnútro po riadkoch zľava doprava smerom zhora dole. Druhá prechádza vnútro po stĺpcoch zhora dole smerom zľava doprava.

Obmedzenia na pohyb

Obmedzenie je definované iba na pohyby, ktoré by viedli do hraničného pixela (#0, #1). Z dôvodu, mohlo vnútro kruhu prejsť v dvoch fázach, tak pri pohybe v prvej fáze sa nebudú meniť hodnoty pixelov, v ktorých sa automat nachádza z hodnoty 0 na hodnotu 1.

Graf

Nasledujúci graf ukazuje, ako funguje rozpoznávanie vnútra pre obidve fázy Vnútorného automatu.



Obr. 3.6:

V stave q_0 sa prechádza riadok (Fáza I), alebo stĺpec (Fáza II) z vnútra kruhu zo vstupného obrázka. Vzniká tak nový reťazec, ktorý obsahuje iba farby z pixelov, cez ktoré sa prešlo.

Do stavu q_1 sa vstupuje, ak sa narazí na pixel s hodnotou R1. Na novo vzniknutý reťazec sa aplikujú nasledovné pravidlá:

$$\begin{aligned} \mathrm{RR} &\rightarrow \mathrm{R} \\ \mathrm{WW} &\rightarrow \mathrm{W} \\ \mathrm{BB} &\rightarrow \mathrm{B} \\ \mathrm{XX} &\rightarrow \mathrm{X} \end{aligned}$$

Môžeme povedať, že vďaka týmto pravidlám sa všetky postupnosti rovnakých farieb spoja do jedného reťazca.

Automaticky sa prechádza do ďalšieho stavu q_2, v ktorom sa porovná reťazec s reťazcom r, kde r $\in \alpha$ (Fáza I) alebo s reťazcom s, kde s $\in \beta$ (Fáza II). Pri úspešnom porovnaní sa vojde do stavu q_3, ináč sa vojde do stavu q_4.

V stave q_4 sa do reťazca r, alebo s (podľa toho v akej sme fáze) priradí nasledujúci reťazec z patričnej množiny a znova sa vykoná porovnávanie ako v stave q_2. Na začiatku je v reťazci r a s prvý prvok.

Stavy q_3 a q_5 počítajú koľkokrát sa cez ne prešlo a z oboch stavov sa dostaneme do q_6. V tomto stave sa vykoná posun na nový riadok (poprípade stĺpec). Ak sa dá posunúť ide sa do počiatočného stavu q_0 ináč sa ide do koncového stavu, v ktorom sa vypočíta percentuálna úspešnosť rozpoznaných riadkov na základe vzorca:

$$\mathrm{U} = \frac{\text{hodnota } q_3}{\text{hodnota } q_3 + \text{hodnota } q_5} * 100$$

Ak je hodnota U vyššia ako 90, tak sa objekt rozpoznal. V opačnom prípade sa nenašiel a pokračuje sa ďalej prvou častou Vyhľadávanie.

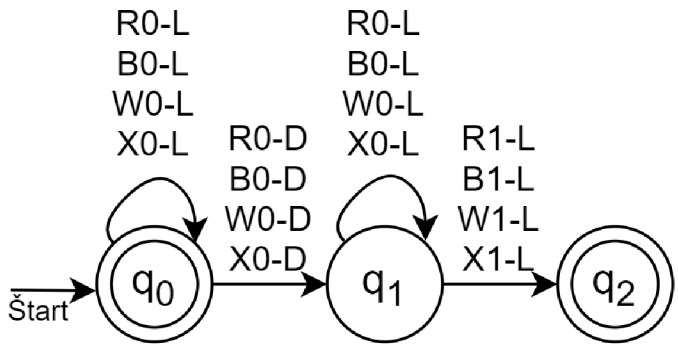
Fáza I.

V prvej fáze sa vnútro kruhu prechádza po riadkoch zľava doprava smerom zhora dole. Každý riadok sa spracuje a porovná, ako je to opísané vyššie. Na posun na nový riadok sa používajú nasledujúce pravidlá:

Stav q_0	Stav q_1
R0-D $\rightarrow q_1$	R1-L $\rightarrow q_2$
B0-D $\rightarrow q_1$	B1-L $\rightarrow q_2$
W0-D $\rightarrow q_1$	W1-L $\rightarrow q_2$
X0-D $\rightarrow q_1$	X1-L $\rightarrow q_2$
R0-L $\rightarrow q_0$	R0-L $\rightarrow q_1$
B0-L $\rightarrow q_0$	B0-L $\rightarrow q_1$
W0-L $\rightarrow q_0$	W0-L $\rightarrow q_1$
X0-L $\rightarrow q_0$	X0-L $\rightarrow q_1$

Stav q_0 je počiatočný stav a zároveň aj koncový v prípade, že sa nedá aplikovať žiadne pravidlo. V tom prípade sa prvá fáza končí a začína sa druhá.

Stav q_2 je taktiež koncový a pozícia, na ktorej sa nachádza reprezentuje ďalší riadok vo vnútri kruhu.



Obr. 3.7:

Na konci sa pripraví pozícia na druhú fázu. Automat sa presunie na miesto, v ktorom sa skončila 3. fáza Kruhového Automatu (Do najľavejšej hornej časti kruhu).

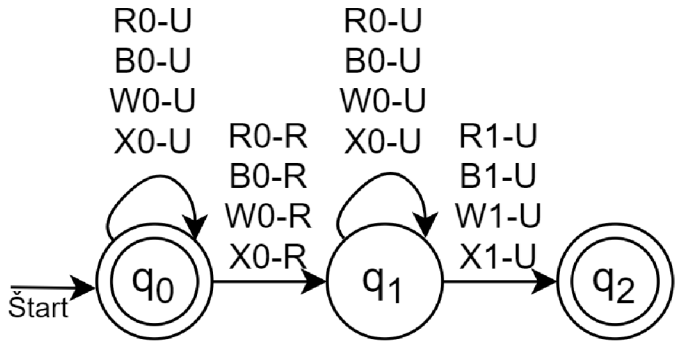
Fáza II.

V druhej fáze sa vnútro kruhu prechádza po stĺpcoch zhora dole smerom zľava doprava. Každý stĺpec sa porovná a spracuje ako je to opísané vyššie. Na posun na nový riadok sa používajú nasledujúce pravidlá:

Stav q_0	Stav q_1
R0-R $\rightarrow q_1$	R1-U $\rightarrow q_2$
B0-R $\rightarrow q_1$	B1-U $\rightarrow q_2$
W0-R $\rightarrow q_1$	W1-U $\rightarrow q_2$
X0-R $\rightarrow q_1$	X1-U $\rightarrow q_2$
R0-U $\rightarrow q_0$	R0-U $\rightarrow q_1$
B0-U $\rightarrow q_0$	B0-U $\rightarrow q_1$
W0-U $\rightarrow q_0$	W0-U $\rightarrow q_1$
X0-U $\rightarrow q_0$	X0-U $\rightarrow q_1$

Stav q_0 je počiatočný stav a zároveň aj koncový v prípade, že sa nedá aplikovať žiadne pravidlo. V tom prípade sa končí aj druhá fáza.

Stav q_2 je taktiež koncový a pozícia, na ktorej sa nachádza, reprezentuje ďalší stĺpec vo vnútri kruhu.



Obr. 3.8:

Kapitola 4

Implementácia

Implementácia tohto dvojrozmerného konečného automatu je rozdelená do troch častí. V prvej časti sa vyhľadáva začiatok potencionálneho objektu. Druhá časť je zameraná na rozpoznávanie obrysu. Posledná súčasť porovnáva a kontroluje vnútro objektu. Popis implementácie je znázornený na kruhových značkách. Použité značky sú obmedzenie maximálnej povolenej rýchlosti na rýchlosť 50,70 a 80 km/h.

4.1 Inicializácia

Inicializácia prebieha tak, že sa na začiatku vytvorí dvojrozmerné pole objektov *Dot*, ktoré reprezentuje vstupný obrázok. Jeho rozmery sú totožné z rozmermi vstupu. Každý pixel zo vstupného obrázka predstavuje objekt *Dot*. Ten pozostáva z nasledujúcich hodnôt:

- *Value* - farba pixela zo vstupného obrázka
- *Live* - hodnota života, nadobúda hodnoty true alebo false a udáva, či automat prešiel cez daný bod alebo nie
- *Pozcia* - udáva pozíciu x a y bodu zo vstupného obrázka

4.1.1 Načítanie obrázka

Načítanie obrázka je implementované pomocou základných knižníc. Vytvorí sa objekt *QImage*, do ktorého sa vloží vstupný obrázok.

Vstupné obrázky sú v adresári „images“ ktorý sa nachádza na rovnakej ceste ako spustiteľná aplikácia. Rozpoznávanie bolo testované na obrázkoch s príponami .png a .jpg. Na rozmeroch obrázka nezáleží. Ale čím má obrázok väčšie rozlíšenie, tým môže byť rozpoznávanie pomalšie. Kontroluje sa viac bodov.

Naplní sa dvojrozmerné pole *Dot*. Každému bodu sa nastavia patričné súradnice a hodnota života na „true“. Prevedieme objekt *QImage* na *QColor* a z neho získame RGB hodnoty. Tie sa vložia ako argumenty pre funkciu *GetColor* a výsledok sa zapíše do hodnoty *Value*.

4.1.2 Funckia GetColor

Táto funkcia na základe troch hodnôt RGB (Red, Green, Blue) roztriedi pixely do skupín podľa farby. Tým sa docieli to, že sa zjednotia rôzne odtiene tej istej farby do jednej. Vďaka

tomu môže rýchlejšie prechádzať vstupný obrázok. Pri dopravných značkách nám netreba poznať veľa farieb.

Rozlišujú sa 3 farby, biela, červená a čierna. V prípade, že hodnoty sa nerovnajú ani jednej z implementovaných farieb, tak sa vráti biela hodnota. Je to z dôvodu urýchlenia vyhľadávania a ničomu to nevadí.

Podmienky na rozdelenie farieb do skupín ,:

- Ak sú všetky hodnoty Red, Green, Blue väčšie ako 200, tak je farba rovná bielej.
- Ak sú všetky hodnoty Red, Green, Blue menšie ako 70, tak je farba rovná čiernej.
- Ak je hodnota Red väčšia ako súčet hodnôt Green a Blue a súčasne, ak sú hodnoty Green a Blue menšie ako 60, tak je farba rovná červenej.

4.1.3 Definícia hraníc a identifikátory značiek

Na konci inicializácie sa zavolá funkcia *CreateBorderPicture*. Tá vytvorí hranice okolo dvojrozmerného poľa objektov *Dot*. Je to preto, aby sa automat nemohol pohnúť mimo povolenú plochu pri pohybovaní, keď sa bude snažiť rozpoznať dopravnú značku. Body hraníc majú hodnotu Value rovnú -1.

Na ukladanie rozpoznaných značiek sa vytvorí vektor „Signs“, kde sa budú ukladať čísla - identifikátory značiek (ID). Identifikátor jednoznačne určuje rozpoznaný objekt (značku).

- ID s hodnotou 0 – značka maximálna povolená rýchlosť 50
- ID s hodnotou 1 – značka maximálna povolená rýchlosť 80
- ID s hodnotou 2 – značka maximálna povolená rýchlosť 70

4.2 Pohyb automatu

Automat sa dokáže pohybovať v štyroch smeroch. Doprava, doľava, hore a dole. Pohybuje sa v dvojrozmernom poli objektov *Dot*. Ľavý horný bod tohto poľa má súradnice pre x a y rovné nula. Pre každý smer sú implementované dve funkcie. Líšia sa v tom, že jedna naozaj posunie aktuálnu pozíciu do požadovaného smeru a druhá len skontroluje farbu a či už automat bol v danom bode alebo nie.

- Pri pohybe hore sa mení súradnica y, zmenší sa o hodnotu jedna.
- Pri pohybe dole sa mení súradnica y, zväčší sa o hodnotu jedna.
- Pri pohybe doľava sa mení súradnica x, zmenší sa o hodnotu jedna.
- Pri pohybe doprava sa mení súradnica x, zväčší sa o hodnotu jedna.

4.2.1 Funkcie Right, Left, Up, Down

Tieto funkcie sú takmer totožné, líšia sa iba smerom, do ktorého sa posúvajú. Začínajú tým, že získajú hodnotu *Value* od bodu, do ktorého sa majú premiestniť.

Pozrieme sa, či sa táto hodnota rovná hodnote mínus jedna. Ak áno, znamená to, že sme narazili na hranice poľa. Sme na konci. V takom prípade sa neposúvame nikde a funkcia vráti hodnotu „false“. Inak pokračujeme ďalej.

Do aktuálneho bodu, v ktorom sa automat nachádza, prepíšeme hodnotu *Live* na „false“. Automat prešiel cez daný pixel a vo väčšine prípadov sa netreba k nemu vracať. Zmení sa aktuálna pozícia x, alebo y podľa toho, do ktorej stany sa automat posúva. Vráti sa hodnota „true“. Znamená to, že sa úspešne zmenila poloha a automat sa posunul.

```
bool Automat::Up(Dot **p)
{
    if (p[x][y - 1].Value == -1)
        return false;

    p[x][y].Live = false;
    y--;
    Color = p[x][y].Value;
    return true;
}
```

Obr. 4.1:

Na obrázku 4.1 môžme vidieť kód implementácie pohybu hore.

4.2.2 Funkcie CheckRight, CheckLeft, CheckUp, CheckDown

Tieto funkcie nemenia aktuálnu pozíciu na novú pozíciu. Iba ju kontrolujú. Z dôvodu sprehľadnenia kódu sa vytvorí dočasná premenná *next*. Do nej sa uloží potenciálna nová pozícia (jeden bod z dvojrozmerného poľa *Dot*).

Z premennej *next* sa porovná hodnota *Value*. Ak sa rovná hodnote mínus jedna, tak sme narazili na hranice poľa. V takom prípade sa neposúvame nikde a funkcia vráti hodnotu „false“. Inak pokračujeme ďalej.

Hodnota farby v bode, v ktorom sa nachádzame, sa musí rovnať hodnote farby v bode nasledujúcom. Keď to tak nie je, tak funkcia vráti hodnotu „false“. Ak to vyhovuje, tak ešte treba skontrolovať, či automat už bol v bode, ktorý kontrolujeme. Vieme to docieliť tým, že sa pozrieme na hodnotu *Live* z premennej *next*.

Výsledok celého kontrolovania je, že tieto funkcie skontrolujú, či sa neposúvame mimo poľa, posúvame sa po rovnakej farbe a ešte sme v danom bode neboli. Funkcie sa používajú hlavne v časti Kruhového automatu.

```
bool Automat::CheckUp(Dot** p)
{
    Dot next = p[x][y - 1];
    if (next.Value == -1)
        return false;

    if (next.Value != p[x][y].Value)
        return false;
    else
        return next.Live;
}
```

Obr. 4.2:

Na obrázku 4.2 môžme vidieť kód implementácie overenia pohybu hore.

4.3 Vyhľadávanie

Rozpoznávanie dopravnej značky začína tým, že sa spustí prvá časť automatu, ktorá vyhľadáva pravdepodobný začiatok značky. Na začiatku sa definuje farba, ktorú bude hľadať. Všetky značky s obmedzením na rýchlosť, ktoré daný automat podporuje, majú červený kruh. Farba sa tak nastaví na červenú.

4.3.1 Funkcia FindNextPoint

Táto funkcia vyhľadáva pixel, ktorý by mohol byť začiatkom značky. Začína od bodu so súradnicami pre x rovné jednej a y rovné jednej. Postupne sa pohybuje smerom doprava. Každý bod, do ktorého vstúpi, kontroluje. Pozerá sa na hodnoty *Live* a *Value*.

Ak je hodnota *Live* nastavená na „true“, čo znamená že automat doteraz neprešiel cez tento bod, tak sa ešte skontroluje, či v hodnote *Value* nieje uložená hodnota červenej farby. Ak áno, funkcia končí a vráti sa hodnota true. Našiel sa potencionálny objekt.

Ak je hodnota *Live* nastavená na „false“, čiže automat cez daný bod už prešiel, tak sa už hodnota *Value* nekontroluje, nemá to zmysel. Vieme, že to už bolo raz kontrolované. Automat sa tým pádom posunie ďalej na ďalší bod smerom doprava.

Primárny pohyb v tejto časti je smerom doprava. V každom bode sa musí kontrolovať, či sme nenarazili na koncové hranice. Keď už sme na konci, tak súradnicu x zvýšime o jednotku a z automatom sa posunieme smerom dole. Prechádzame tak riadok po riadku a kontrolujeme body.

V prípad,e že sa automat pri posunutí dole dostal do hraničného bodu, tak sa vyhľadávanie končí. Prešli sme celý vstupný obrázok. Vráti sa hodnota „false“.

4.3.2 Výsledok vyhľadávania

Funkcia *FindNextPoint* pripraví pozíciu automatu na rozpoznanie dopravnej značky. V cykle sa volá táto funkcia spolu aj s ostatnými časťami kruhový a vnútorný automat, kým jej návratová hodnota je „true“. Výsledkom posledného dielu je identifikátor rozpoznanej značky alebo hodnota mínus jedna v prípade neúspechu.

Po skončení cyklu sa užívateľovi vhodným spôsobom interpretuje výsledok. Vypíšu sa názvy dopravných značiek, ktoré boli nájdené.

4.4 Kruhový automat

Táto časť automatu kontroluje okraje objektu. Pohybuje sa do kruhu. Na začiatku sa uloží počiatočný bod, v ktorom sa začalo rozpoznávanie. Celé kontrolovanie je rozdelené do štyroch hlavných častí. Každý diel prechádza jednu štvrtinu z kruhu. Reprezentujú to funkcie *MostRight*, *MostDown*, *MostLeft* a *MostUp*. Volajú sa tieto funkcie v poradí ako sú napísané. Kontroluje sa ich návratová hodnota. Ak aspoň jedna z nich vráti negatívnu hodnotu „false“, tak sa rozpoznávanie končí a kruhový automat skončí s rovnakou návratovou hodnotou.

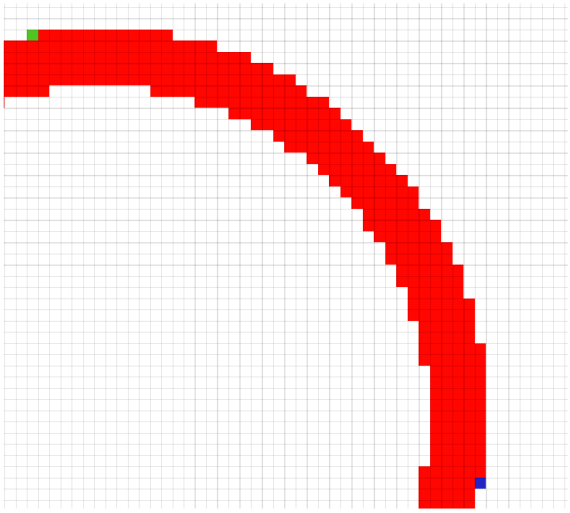
Každá táto funkcia pozostáva z troch fáz. Všetky funkcie sú popísané v nasledujúcich sekciách. To, kde sa automat ďalej pohne, v ktorom ďalšom bode bude, určuje stavový diagram, ktorý bol opísaný v sekcii 3.2.2. Implementovaný je pomocou automatu swtich case v jazyku c++.

4.4.1 Funkcia MostRight

Automat sa pohybuje smerom doprava a dole. Využíva pri tom funkcie na pohyb *Right* a *Down*. V každom bode sa kontrolujú okolité body a posúva sa stále po jednej farbe. Kontroluje to funkciou *CheckRight* a *CheckDown*. Ak sa stane, že sa automat už nevie posunúť doprava, funkcia *CheckRight* vracia hodnotu „false“, zavolá funkciu *CheckDown* a v prípade úspechu sa posunie dole. Opis týchto funkcii je v sekcii 4.2.

V prvej fáze sme v hornej časti kruhu. Docielila to funckia *FindNextPoint* v sekcii 4.3.1. Musí prevládať pohyb doprava a posúvať sa môže o jeden, maximálne dva body naraz dole. V druhej fáze by sa mali pohyby striedať a nemali by byť veľké rozdiely medzi nimi. V poslednej fáze sa už blížime do najpravejšej dolnej časti kruhu. Prevládať by tu mal pohyb dole a doprava by sa automat mal posúvať len nepatrne o jeden maximálne dva body doprava. Pri porušení týchto fáz nastáva chyba a funkcia vracia hodnotu „false“. Diagram prechodov, ktorý využíva funkcia *MostRight*, je na obrázku 3.2.

Táto časť končí, ak sa už nedá ďalej posunú ani doprava, ani doľava. V tom prípade vracia hodnotu „true“, čo znamená, že sa mu podarilo dostať do najpravejšej časti potencionálneho kruhu.



Obr. 4.3:

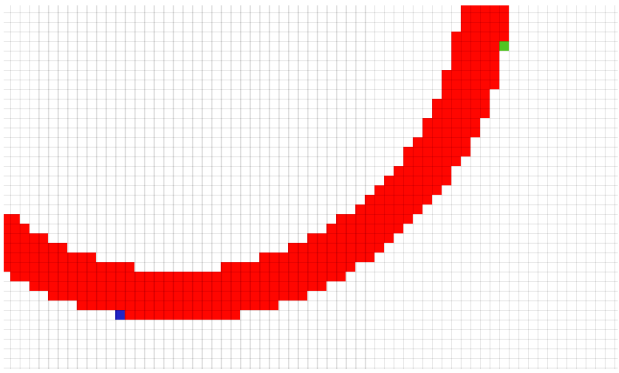
Na obrázku 4.3 je znázornená časť kruhu, ktorú prejde funkcia MostRight. Začína sa v zelenom bode a končí v modrom bode.

4.4.2 Funkcia MostDown

Automat sa pohybuje smerom dole a doľava. Využité sú pritom funkcie na pohyb *Down* a *Left*. V každom bode sa kontrolujú okolité body. Posúva sa stále po jednej farbe. Kontroluje sa to funkciou *CheckDown* a *CheckLeft*. Ak sa stane, že sa automat už nevie posunúť dole, funkcia *CheckDown* vracia hodnotu „false“, volá sa funkcia *CheckLeft* a v prípade úspechu sa posunie doľava. Opis týchto funkcii je v sekcii 4.2

V prvej fáze sme v najpravejšej dolnej časti kruhu. Dostali sme sa tu pomocou funkcie *MostRight*. Musí tu prevládať pohyb dole a posúvať sa môže o jeden, maximálne dva body naraz doľava. V druhej fáze by sa mali pohyby striedať a nemali by byť veľké rozdiely medzi nimi. V poslednej fáze sa dostávame do najspodnejšej ľavej časti kruhu. Prevládať by tu mal pohyb vľavo a iba o pár bodov by sa mal posúvať dole. Veľké skoky by tu nemali byť. V prípade, že tieto fázy sú porušené, nastáva chyba a funkcia vracia hodnotu „false“. Diagram prechodov, ktorý využíva funkcia *MostDown* je na obrázku 3.3.

Táto časť končí, ak sa už nedá ďalej posunú ani dole, ani doprava. V tom prípade sa vracia hodnota „true“, čo znamená, že sa podarilo dostať do najspodnejšej časti potenciónálneho kruhu.



Obr. 4.4:

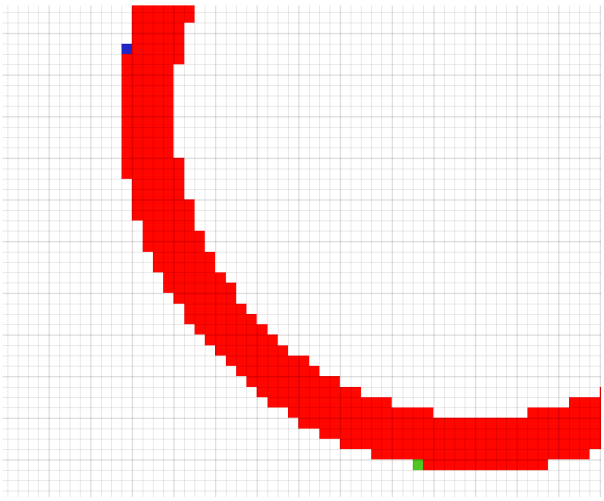
Na obrázku 4.4 je znázornená časť kruhu, ktorú prejde funkcia *MostDown*. Začína sa v zelenom bode a končí v modrom bode.

4.4.3 Funkcia MostLeft

Automat sa pohybuje smerom doľava a hore. Nato sa využíva funkcie *Left* a *Up*. V každom bode sa kontrolujú okolité body. Posúva sa stále po jednej farbe. Kontroluje sa to pomocou funkcii *CheckLeft* a *CheckUp*. Ak sa stane, že automat sa už nevie posunúť doľava tým, že funkcia *CheckLeft* vracia hodnotu „false“, volá sa funkcia *CheckUp* a v prípade úspechu sa posunie hore. Opis týchto funkcii je v sekcii 4.2

V prvej fáze sa automat nachádza v najspodnejšej ľavej časti kruhu. Dostali sme sa tu vďaka funkcii *MostDown*. Musí tu prevládať pohyb vľavo a posúvanie je možné o jeden, maximálne dva body naraz doľava. V druhej fáze by sa mal pohyb doľava a hore striedať rovnomerne. V poslednej fáze sa dostávame do najľavejšej časti kruhu. Prevládať by tu mal pohyb smerom dohora a o iba pár bodov naraz sa posúvať doľava. Veľké skoky by tu nemali byť. V prípade, že tieto fázy sú porušené, nastáva chyba a funkcia vracia hodnotu „false“. Diagram prechodov, ktorý využíva funkcia *MostLeft*, je na obrázku 3.4.

Táto časť končí, ak sa už nedá ďalej posunúť ani doľava, ani hore. V tom prípade sa vracia hodnota „true“, čo znamená, že sa podarilo dostať do najspodnejšej časti potencionálneho kruhu.



Obr. 4.5:

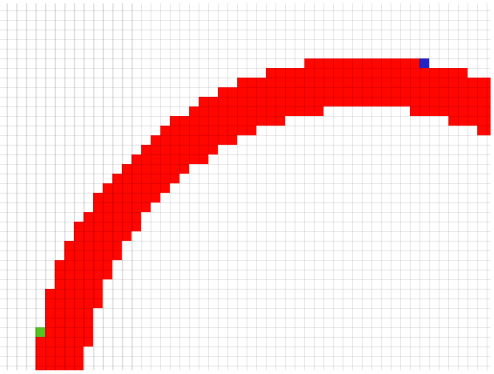
Na obrázku 4.5 je znázornená časť kruhu, ktorú prejde funkcia *MostLeft*. Začína sa v zelenom bode a končí v modrom bode.

4.4.4 Funkcia MostUp

V poslednej časti kontrolovania obrysov objektu sa automat pohybuje smermi hore a doprava. Využívajú sa tu funckie *Up* a *Right* na pohyb. V každom bode sa kontrolujú okolité body. Posúva sa stále po jednej farbe. Pred každým posunom sa kontroluje nasledujúci bod pomocou funkcii *CheckUp* a *CheckRight*. Ak sa stane, že sa automat nevie posunúť dohora, tým že funckia *CheckUp* vracia hodnotu „false“, kontroluje sa bod na pravej strane funkciou *CheckRight* a prípade úspechu sa tam aj posunie. Opis týchto funkcií je v sekcii 4.2

V prvej fáze sa automat nachádza v najľavejšom hornom bode potencionálneho kruhu. Dostali sme sa tu pomocou funkcie *MostLeft*. Mal by tu prevládať pohyb smerom hore a posúvať sa môže naraz o jeden maximálne dva body doprava. V druhej fáze by sa mali pohyby hore a vpravo striedať rovnomerne. Posledná fáza v tejto funkcii *MostUp* sa líši od ostatných tretích fáz predchádzajúcich funkcii. Mali by sme sa nachádzať už na rovnakej časti ako na začiatku v prvej fáze funkcie *MostRight*. Kontrolujeme tu teda, či sa bod smerom nahor od aktuálnej polohy rovná počiatočnému bodu, ktorý sme si inicializovali na začiatku pri spustení kruhového automatu. V prípade že áno, uložíme si do premennej *isCircle* hodnotu „true“, ktorá bude značiť, že sme rozpoznali kruh. Pokračujeme ďalej smerom doprava, aby sme sa dostali do najpravejšieho bodu rozpoznaného kruhu. V prípade, že tieto fázy sú porušené, nastáva chyba a funkcia vracia hodnotu „false“. Diagram prechodov, ktorý využíva funkcia *MostUp*, je na obrázku 3.5.

Táto časť končí, ak sa už nedá posunúť ďalej ani hore ani doprava. V tomto prípade sa vráti hodnota premennej *isCircle*, ktorá nám hovorí, či sme našli kruh, alebo nie.



Obr. 4.6:

Na obrázku 4.6 je znázornená časť kruhu, ktorú prejde funkcia *MostUp*. Začína sa v zelenom bode a končí v modrom bode.

4.5 Vnútorný automat

Vnútorný automat rozpoznáva vnútro už nájdeného kruhového objektu. Ten sme získali pomocou Kruhového automatu. V tejto časti už vieme, že sme našli kruh a pravdepodobne to bude značka, no zatiaľ nevieme aká. V tejto sekcii je opísaná implementácia rozpoznávania už konkrétnych značiek. A to sú obmedzenie maximálnej povolenej rýchlosti na rýchlosť 50, 70 a 80 km/h. Pohyb v tejto časti automatu tiež zabezpečujú funkcie opísané v sekcii 4.2

Samotné rozpoznávanie prebieha v dvoch častiach. Raz sa prechádza objekt po riadkoch horizontálne a druhý- krát po stĺpcoch vertikálne. Výsledok častí je vektor obsahujúci všetky spracované riadky. Reprezentujú ich funkcie *ReadHorizontalLine* a *ReadVerticalLine*. Na začiatku sa vytvoria dva vektory. Jeden vektor reťazcov, do ktorého sa budú ukladať spracované riadky. Do druhého sa budú ukladať body, cez ktoré automat prejde.

Vykoná sa funkcia *NewHorizontalLine*, ktorá posúva aktuálnu pozíciu vždy o riadok nižšie na začiatok ďalšieho riadku objektu. Podrobnejší popis implementácie tejto funkcii bude v podkapitole 4.5.1. Do vektoru bodov sa pridá aktuálny bod. Vykoná sa funkcia *ReadHorizontalLine*, ktorá vráti už upravený reťazec aktuálneho riadku skúmaného objektu. Táto časť sa volá v cykle, kým nám funkcia *NewHorizontalLine* vracia hodnotu „true“. V prípade že vráti hodnotu „false“, znamená to, že sme prešli všetky vodorovné riadky. Ukladáme si tu aj aktuálnu polohu, aby sme sa vedeli dostať ku pozícii začiatku posledného riadka.

```
std::vector<std::string> Lines;

while (NewHorizontalLine() != false)
{
    list->push_back(p[a->x][a->y]);
    Lines.push_back(ReadHorizontalLine());
    x = a->x;
    y = a->y;
}
```

Obr. 4.7:

Na obrázku 4.7 je zobrazená časť kódu, ktorá spracováva riadky zo vstupného obrázka. *List* je vektor bodov *Dot*, p je dvojrozmerné pole vstupných bodov *Dot*. a->x a a->y sú aktuálne súradnice, v ktorých sa nachádza automat.

Po úspešnom naplnení vektora, ktorý obsahuje spracované riadky sa začína s porovnávaním. Volá sa tu funkcia *CompareGrammer* s parametrom s hodnotou „true“. Vďaka nej funkcia dokáže rozpoznať či sa porovnávajú riadky, alebo stĺpce kruhového objektu. V prípade že táto funkcia vráti hodnotu -1, znamená to že spracované riadky sa nerovnajú žiadnej z doposiaľ implementovaných gramatík značiek. Môžeme prestať porovnávať a vraciame taktiež hodnotu -1, čo znamená že sme nerozpoznali značku. Opis implementácie funkcie bude v podkapitole 4.5.3.

Všetky body, cez ktoré sme v tejto časti prešli sa ukladali do vektora. Majú nastavenú hodnotu života *Live* na „false“. Tým pádom sa po nich už nedá prejsť. Pri pohybe automatu sa volali funkcie, ktorých implementácia je vysvetlená v kapitole 4.2 Potrebuje prejsť cez vnútro kruhu ešte raz a tak všetkým bodom vo vektore nastavíme hodnotu života *Live* na „true“. Vyčistíme vektor spracovaných riadkov a pripravíme ho na naplnenie od ďalšej funkcie.

Automat sa presunie na novú pozíciu, z ktorej sa dá získať prvý stĺpec značky. Je to najľavejšia časť kruhu. Aktuálnu pozíciu posunie na to miesto funkcia *FirstVerticalLine*. Odtiaľ sa bude *NewVerticalLine* volať a posúvať sa po stĺpcoch kruhu. Vracia hodnotu „true“. Do vektoru spracovaných riadkov sa uloží výsledok funkcie *ReadVerticalLine*. Opakuje sa to kým sa dá posúvať po stĺpcoch. Znamená to, že sa stále budeme nachádzať vo vnútri kruhu.

Ak *NewVerticalLine* vráti hodnotu „false“, prešli sme všetky stĺpce v kruhu, ktorého vnútro rozpoznávame a cyklus končí.

Pred tým ako sa druhýkrát zavolá funkcia *CompareGrammer* na porovnanie stĺpcov, musia sa informácie získané z prvého volania presunúť k druhému. Je to preto, aby sa nadviazalo rozpoznávanie na prechádzajúce. Ak sa už pri rozpoznávaní riadkov nepotvrdil výskyt nejakej značky, tak už nemá zmysel ju teraz kontrolovať. Celkové rozpoznávanie sa tým urýchli.

Inforámcie, ktoré sa presunú sú:

- indexGrammerLine - určuje pozíciu v poli gramatík pre konkrétnu značku

- MaxHorizontalGrammerLine - počet riadkov gramatík v poli
- MaxVerticalGrammerLine - počet stĺpcov gramatík v poli

Funkcia *CompareGrammer* je volaná s parametrom „false“, aby vedela že sa idú porovnávať stĺpce. Výsledok tejto funkcie je už identifikátor konkrétnej rozpoznanej značky, alebo hodnota mínus jedna v prípade, že kruhový objekt nebol rozpoznaný.

4.5.1 Získavanie riadkov

Získavanie a spracovanie riadkov v kruhovom objekte zabezpečujú funkcie *NewHorizontalLine* a *ReadHorizontalLine*. Prvá funkcia posúva automat stále na nový riadok a druhá funkcia ho prečíta a spracuje.

NewHorizontalLine

Prvýkrát sa volá táto funkcia, keď sa aktuálna pozícia automatu nachádza v pravej hornej časti. Tam, kde sa skončilo rozpoznávanie Kruhového automatu.

Pokúsime sa posunúť o riadok nižšie a potom stále do ľavej strany. Samostatne sa tu musí kontrolovať hodnota života *Live* v bode, do ktorého sme sa dostali. Pohybovať sa môžeme iba v bodoch, v ktorých sme ešte neboli, čo znamená, že hodnota *Live* je rovná „true“.

Ak sa dole nedá posunúť, posúvame sa doľava s tým, že kontrolujeme bod pod nami, či sa tam už dá dostať. V prípade, že dá dostať dole, zavolá sa rekurzívne aktuálna funkcia. Tým sa posunieme dole a pohybujeme sa doľava až na začiatok riadku.

Výsledok funkcie je hodnota „true“ v prípade, že sa aktuálna pozícia zmenila na pozíciu nového riadku rozpoznávaného objektu.

ReadHorizontalLine

Táto funkcia sa volá vždy po funkcii *NewHorizontalLine*. Spracuje aktuálny riadok na základe gramatík (množiny α a β), ktoré sú popísané v kapitole 3.1.2.

V cykle posúva automat stále smerom doprava, kým nenarazí na bod, ktorý má hodnotu života *Live* nastavenú na „false“. Znamená to, že sme prešli celý jeden riadok. Pri pohybe sa kontroluje aj farba bodov, cez ktoré sa prechádza. Vytvorí sa pomocná premenná *Color*, do ktorej sa ukladá farba z predchádzajúcich bodov. Vždy pri zmene farby sa pridá do výsledného reťazca znak. Pri červenej farbe je to písmeno R (Red) a pri farbe čiernej písmeno B (Black).

Výsledný reťazec je návratová hodnota.

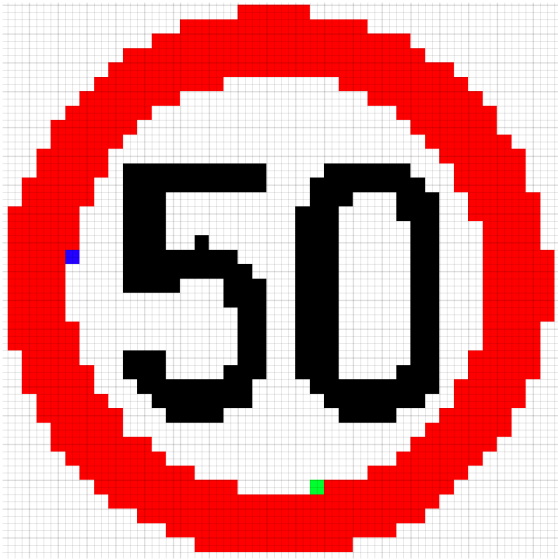
4.5.2 Získavanie stĺpcov

Získavanie a spracovanie stĺpcov v kruhovom objekte zabezpečujú funkcie *FirstVerticalLine*, *NewVerticalLine* a *ReadVerticalLine*.

FirstVerticalLine

Aktuálne sme sa dostali do spodného dolného rohu kruhového objektu, ktorý rozpoznávame. Potrebujeme sa dostať do najľavejšieho bodu objektu, odkiaľ sa budeme posúvať po stĺpcoch smerom doprava a čítať ich.

V cykle sa posúvame doľava a keď sa už ďalej nedá ísť, život Live bodu sa rovná „false“, posunieme sa vyššie. Opakujeme to, kým sa nedá ísť ani doľava, ani hore.



Obr. 4.8:

Na obrázku 4.8 je značka 50, na ktorej sú vyznačené dva body. Funkcia *FirstVerticalLine* presunie aktuálnu pozíciu zo zeleného na modrý bod.

NewVerticalLine

Funkcia sa volá, až keď prebehne funkcia *FirstVerticalLine*. Nachádzame sa v ľavej hornej časti.

Posunieme sa do pravej strany a stále hore. Pritom musíme samostatne kontrolovať hodnotu života *Live* v bode, do ktorého sme sa dostali. Pohybovať sa môžeme iba v bodoch, v ktorých sme ešte neboli, čo znamená že hodnota *Live* je rovná „true“.

Ak sa do pravej strany nedá pohnúť, posúvame sa hore s tým, že kontrolujeme bod napravo od aktuálneho bodu, či sa tam už dá dostať. V prípade, že to ide, zavolá sa rekurzívne aktuálna funkcia. Tým sa posunieme doprava a pohybujeme sa hore až na začiatok riadku.

Výsledok funkcie je hodnota „true“ v prípade, že sa aktuálna pozícia zmenila na pozíciu nového riadku rozpoznávaného objektu.

ReadVerticalLine

Prechádza po stĺpcoch, spracováva, kontroluje a číta ich. Spracovanie aktuálneho stĺpca je na základe gramatík (množiny α a β), ktoré sú popísané v kapitole 3.1.2

V cykle posúva automat stále smerom dole, kým nenarazí na bod, ktorý má hodnotu života *Live* nastavenú na „false“. Znamená to, že sme prešli cely jeden stĺpec. Pri pohybe sa kontroluje aj farba bodov, cez ktoré sa prechádza. Vytvorí sa pomocná premenná *Color*, do ktorej sa ukladá farba z predchádzajúcich bodov. Vždy pri zmene farby sa pridá do výsledného reťazca znak. Pri červenej farbe je to písmeno R (Red) a pri farbe čiernej písmeno B (Black).

Výsledný reťazec je návratová hodnota.

4.5.3 Porovnávanie

V tejto funkcii sa porovnáva spracovaný riadok ,alebo stĺpec s gramatikou danej značky. Gramatika je pole všetkých reťazcov, ktoré reprezentujú dopravnú značku (množiny α a β). Volá sa dvakrát, ale z iným parametrom, ktorý určuje, či ide o rozpoznávanie po riadkoch, alebo po stĺpcoch.

Pre každú značku sa ukladá index, určuje na akej aktuálnej pozícií poľa gramatík sa nachádzame a počet riadkov gramatík v horizontálnom a počet stĺpcov vo vertikálnom smere. Ak index obsahuje hodnotu mínus jedna, znamená to, že rozpoznávanie sa nepodarilo a aktuálny objekt nepatrí danej značke.

Máme zoznam všetkých spracovaných riadkov (stĺpcov) z predchádzajúcich funkcií. Tým pádom máme dve polia, ktoré treba medzi sebou porovnávať. Index pre pole spracovaných riadkov je *indexLine* a pre pole gramatík je to *indexGrammerLine*. Vytiahne sa jeden reťazec z poľa gramatík, podľa značky, s ktorou sa objekt momentálne porovnáva a podľa *indexGrammerLine*. Riadok sa porovná so spracovaným riadkom.

Pri rovnosti sa inkrementujeme *indexLine*. V prípade, že sa nerovnajú, tak skúsime porovnať ešte raz, ale so zvýšenou hodnotou *indexGrammerLine*. Pri úspechu necháme daný index zvýšený, ináč ho necháme na pôvodnej hodnote a pripočítame jednotku k premennej *wrongLine*. Pomocou nej si dokážeme vypočítať koľko riadkov sa rovnalo a koľko nie. Toto sa opakuje, kým neprejdeme všetky spracované riadky.

Ak v tomto cykle sa vystriedajú všetky reťazce gramatiky a v spracovaných riadkov sme sa ešte nedostali na koniec, nastala chyba a objekt nie je totožný s aktuálnu značkou, ktorú porovnávame.

Ďalej sa kontroluje, či premenná *indexGrammerLine* nadobudla maximálnu hodnotu, čiže automat prešiel cez všetky položky v poli gramatík.

Na konci sa vypočíta percentuálna úspešnosť správnych a nesprávnych riadkov. Keď je nad 90 %, tak značka bola rozpoznaná. Tolerujeme malú odchýlku pri rozpoznávaní.

Výsledok funkcie je index značky, ktorá bola rozpoznaná.

```
int percent = (indexLine - wrongLine) / (double)indexLine * 100;
if (percent < 90)
    SignItems[i].indexGrammerLine = -1;
```

Obr. 4.9:

Na obrázku 4.9 je kód, ktorým sa vypočíta percentuálna úspešnosť rozpoznania riadkov (stĺpcov).

4.6 Použité prostriedky

V tejto sekcii sú uvedené technológie, ktoré boli použité pri vytvorení programovej časti tejto bakalárskej práce. Tieto znalosti sú potrebné pre lepšie pochopenie opísanej implementácie. Použité prostriedky a technológie sú vybrané tak, aby bolo nasledujúce pokračovanie pri vývoji aplikácie čo najľahšie.

Výsledný program bol vyvíjaný na platforme Windows 10 a testovaný aj na platforme Ubuntu 18.04 v prostredí Qt vo verzii 5.99. Na úspešné preloženie a spustenie je nutné mať nainštalované všetky technológie.

4.6.1 C++

Jazyk C++ je stále veľmi populárnym jazykom. Od obdobia, kedy boli publikované prvé štandardy jazyka C++, sa veľmi rýchlo posunul na top pozície a tam si drží dlhodobo rovnakú líniu. V súčasnosti patrí jazyk C++ medzi najrozšírenejšie a najpoužívanejšie programovacie jazyky. Vznikol ako rozšírenie z jazyka C. Medzi jeho prednosti patria výkonný kompilátor, rýchly spustiteľný a ľahko udržiavateľný kód [10].

4.6.2 Qt

Pri vývoji programovej časti tejto práce je využíva knižnica Qt. Patrí medzi jedny z najpopulárnejších knižníc pre vytváranie programov s grafickým užívateľským rozhraním. Vybral som si ju aj preto, lebo má podporu na rôznych platformách. Je to multiplatformová knižnica. Podporované desktopové platformy sú Windows, Linux aj OS X. Dá sa použiť aj na mobilné platformy ako Android, iOS, BlackBerry. Veľkou výhodou je prehľadne spracovaná dokumentácia.

Kapitola 5

Testovanie a Experimenty

Piata kapitola obsahuje rôzne príklady využitia opísaného automatu v predchádzajúcich kapitolách, ako sa správa pri špecifických situáciách. Čitateľ sa tu dočíta aj už o existujúcich spôsoboch rozpoznávania obrázkov nielen pomocou automatov. Rôzne vylepšenia a optimalizácie, ktoré neboli aplikované, ale v budúcnosti by sa dali použiť.

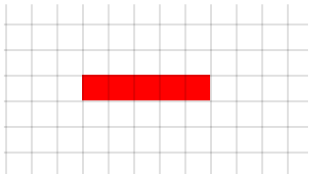
5.1 Testovanie

V tejto sekcii je vybratých pár špecifických objektov a značiek. Tie sa vložia na vstup automatu, ktorý sme si definovali v sekcii 3.1.1 a implementácia zodpovedá s implementáciou v kapitole 4. Pri každom vstupnom obrázku je aj popis toho ako sa automat zachová. Prvú časť Vyhľadávanie môžeme z tohto testovania vynechať, keďže ona iba vyhľadáva prvý bod rovnakej farby. Testovať sa bude druhá a tretia časť, Kruhový automat a Vnútorný automat.

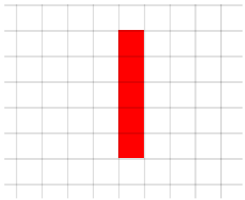
5.1.1 Základné útvary

Nasledujúce vstupné obrázky obsahujú jednoduché základné útvary. Tvorené sú jednou farbou. Kvôli prehľadnosti je farba okolia pri všetkých týchto obrázkoch biela. Pozadie na testovanie nemá vplyv. Aj keby bolo inej farby a v pozadí rôzne útvary a objekty, prvá časť automatu Vyhľadávanie by našla začiatok útvaru a rozpoznávanie by pokračovalo ako pri bielom pozadí.

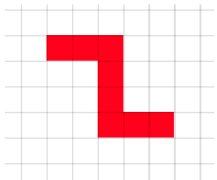
Čiary



Obr. 5.1:



Obr. 5.2:



Obr. 5.3:

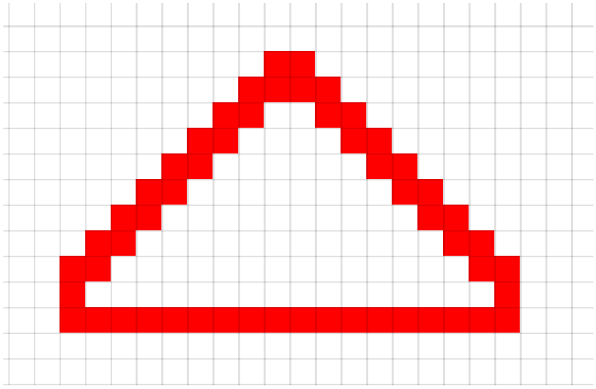
Obrázok 5.1 obsahuje vodorovnú čiaru. Kruhový automat ju prejde celú, na konci sa zastaví

a skončí v chybovom stave, keďže sa už nebude môcť nikde pohnúť.

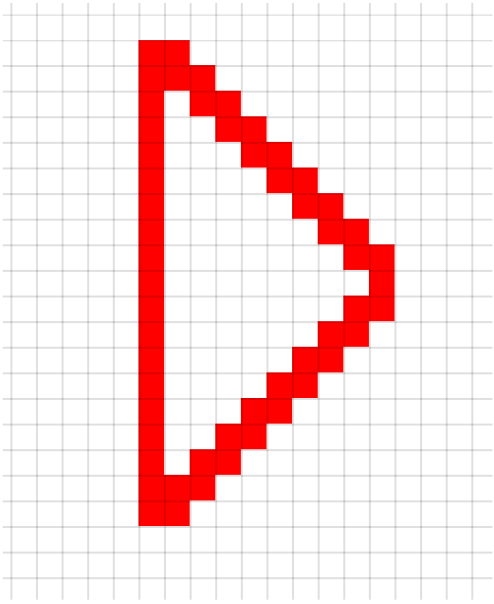
Na obrázku 5.2 sa nachádza zvislá čiara. Od začiatku sa automat pohybuje smerom dole a už v polovici čiary zistí, že nerozpoznáva kruh a tým pádom vojde do chybového stavu.

V obrázku 5.3 je vyobrazený objekt, ktorý obsahuje prudký spád nadol. Ani takýto útvar nie je povolený a automat pri jeho prehľadávaní skončí rovnako ako pri predchádzajúcich.

Trojuholník



Obr. 5.4:

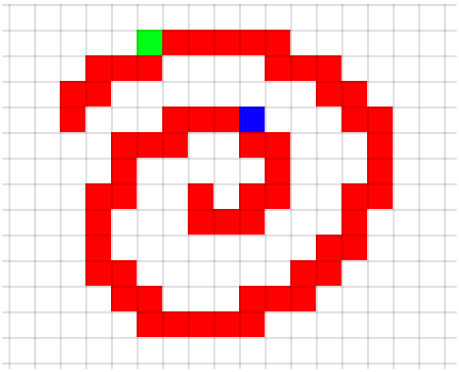


Obr. 5.5:

Ďalší tvar je trojuholník, ktorý sa nachádza na obrázku 5.4. Automat sa pohybuje z horného vrcholu smerom nadol. Prvá fáza Kruhového automatu prebehne v poriadku a na jej noci sa aktuálna pozícia nachádza v pravom vrchole trojuholníka. Chyba nastáva pri druhej fáze. Očakáva sa, že sa pôjde dole a mierne doľava. Ale ako môžeme vidieť na obrázku 5.4, nasledujúce pohyby sú možné iba do ľavej strany, čo je chyba. Automat to zistí a vojde sa do chybového stavu.

Kebyže sa aj stalo, že je trojuholník otočený ako môžeme vidieť na obrázku 5.5, rozpoznaný kruh by nebol. Prvé dve fázy by skončili úspešne, až by sa automat dostal do spodného vrcholu a tam by tretia fáza skončila neúspechom.

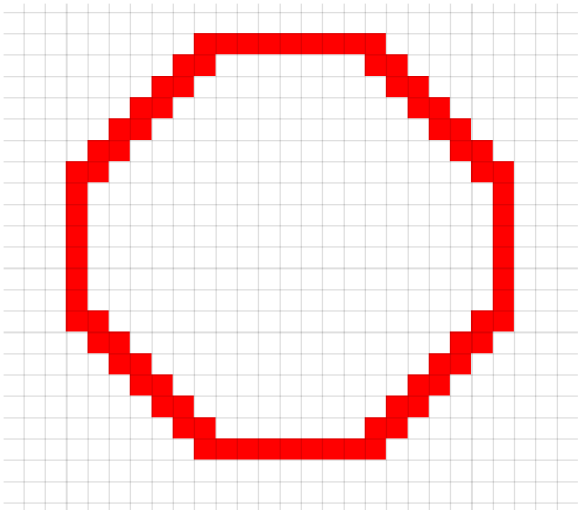
Špirála



Obr. 5.6:

Obrázok 5.6 obsahuje špirálu. Zelený bod ukazuje miesto, kde sa začína pohyb Kruhového automatu a pokračuje sa smerom doprava po červených bodoch. Prejdú sa všetky štyri fázy a automat skončí v bode, ktorý je označený modrou farbou. V poslednej fáze sa kontroluje či bod, cez ktorý prechádza automat, nehraničí s počiatočným bodom (zelený bod). Toto kontrolovanie je popísané v kapitole 4.4.4. Chyba rozpoznávania tak nastala v poslednej štvrtej fáze a Kruhový automat nerozpoznal tak kruh, čo sa aj predpokladalo.

Mnohouholník



Obr. 5.7:

Na obrázku 5.7 je zobrazený pravidelný mnohouholník. Kruhový automat začne rozpoznávanie ako obvykle v jeho hornej časti a postupne prejde celý tento mnohouholník dookola. Všetky fázy skončia úspechom.

Výsledok tohto testu ukazuje, že okrem kruhu Kruhový automat skončí úspešne aj pri mnohouholníkoch, ktoré majú všetky uhly väčšie ako 90 stupňov. Automat takýto uhol vyhodnotí ako súčasť kruhu.

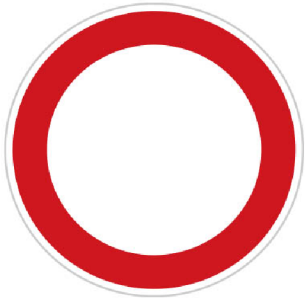
5.1.2 Dopravné značky

Nasledujúce objekty na testovanie rozpoznávania budú dopravné značky. Testovať sa bude ako aj Kruhový automat, tak aj Vnútorný automat. Všetky pozadia kvôli prehľadnosti budú biele. Tak isto ako pri predchádzajúcej sekcii, ničomu to nebude vadiť.

Zákaz vjazdu

Množiny α a β pre značku zákaz vjazdu sú:

$\alpha = \{$ R, RR, R $\}$
$\beta = \{$ R, RR, R $\}$



Obr. 5.8:

Obrázok 5.8 obsahuje dopravnú značku zákaz vjazdu. Táto značka sa nenachádza v množine značiek, ktoré dokáže automat rozpoznať. Dala by sa jednoducho pridať vložením množín α a β, ale v rámci testovania predpokladajme, že nie.

Kruhový automat nájde kruh a spustí sa Vnútorný automat. Porovná riadky a stĺpce vnútra kruhu. Zo začiatku sa bude zdať, že rozpoznávanie prebieha v poriadku. Prvé dva reťazce r_1 a r_2 z množín α a β sú rovnaké pre všetky kruhové značky($r_1 = \{\text{R}\}$, $r_2 = \{\text{RR}\}$). Problém nastáva pri ďalších reťazcoch, ktoré sa už nerovnajú. Vznikajú chybné reťazce a rozpoznávanie končí neúspechom.

Orezané značky

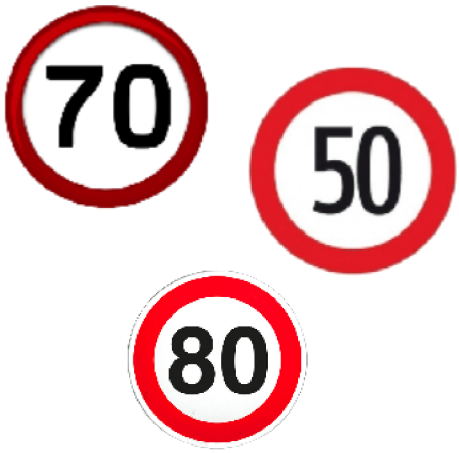
Obr. 5.9:

Obr. 5.10:

Na obrázku 5.9 sa je vidieť značku 50. Nachádza sa v množine značiek, ktoré dokáže automat rozpoznať. Na dvoch stranách chýba časť červenej farby. Automat si dokáže poradiť aj z takouto značkou. Kruhový automat rozpozná kruh a zavolá sa Vnútorný automat, ktorý skontroluje vnútro a výsledkom je rozpoznaná značka obmedzenie rýchlosti na 50 km/h.

Obrázok 5.10 obsahuje takisto značku 50 km/h, ale tá už nie je celá. Chýba z nej kúsok. Pri rozpoznávaní pomocou Kruhového automatu nastáva chyba, keďže chýbajú pixely červeného kruhu. Kruhový automat skončí chybou a značka sa nedala rozpoznať.

Podporované značky



Obr. 5.11:

Na obrázku 5.11 sú tri značky. Automat postupne celý obrázok zhora dole. Ako prvé sa rozpozná značka 70, prejde sa na značku 50 a posledná nájdená značka je 80. Výsledkom sú všetky 3 značky, ktoré boli na obrázku.

Výsledok týchto testov je, že tento nový typ automatu zatiaľ nedokáže rozpoznať objekt, v ktorom chýba veľká časť objektu.

5.2 Existujúce spôsoby rozpoznávania

Existuje veľa spôsobov rozpoznávania obrázkov. V tejto sekcii sa vyberú len pár, stručne sa vysvetlia a porovnajú sa vlastnosti s novým typom dvojrozmerného automatu, ktorý je opísaný v tejto práci.

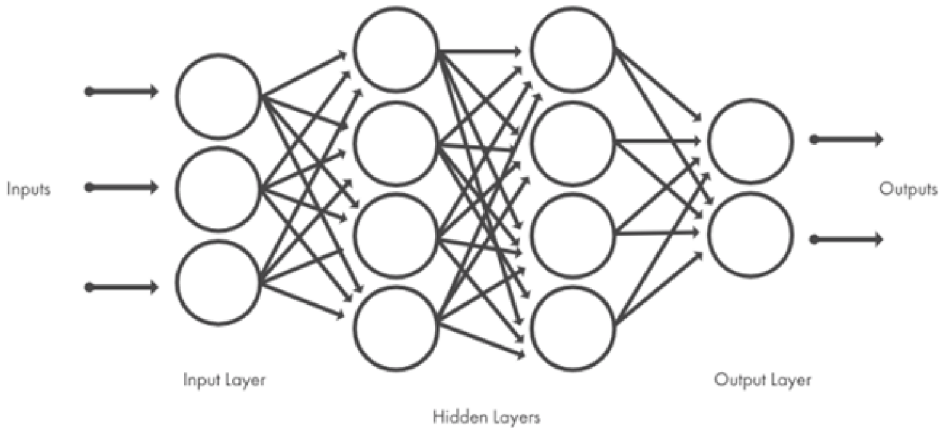
5.2.1 Neurónové siete

Neurónové siete sú počítačové systémy, ktoré sú určené na rozpoznávanie rôznych vzorov. Ich architektúra je inšpirovaná štruktúrou ľudského mozgu. Skladajú sa z troch typov vrstiev:

- vstupná vrstva
- skryté vrstvy
- výstupná vrstva

Vstupná vrstva prijíma signál, vstupné dáta vyhodnocovanie. Úlohou skrytých vrstiev je spracovanie a transformácia vstupných dát, signálu tak, aby ich výstupná vrstva dokázala finálne vyhodnotiť. Počet skrytých vrstiev je voliteľný. Výstupná vrstva robí rozhodnutie alebo prognózu vstupných údajov, podáva výsledky.

Každá sieťová vrstva sa skladá z uzlov (umelé neuróny). Je ich konečný počet. Sú medzi sebou poprepájané, pričom prepojenia majú svoje ohodnotenie – váhy. V každom takomto uzle sa vykonáva výpočet [2].



Obr. 5.12: [2]

Na obrázku 5.12 vidieť 3-vrstvovú neurónovú sieť. Obsahuje jednu vstupnú vrstvu, dve skryté a jednu výstupnú.

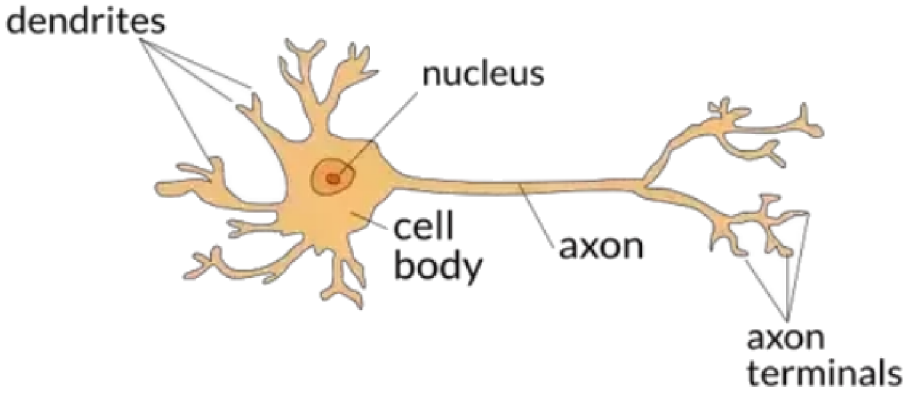
Rozdelenie neurónových sietí

Neurónové siete rozdeľujeme podľa toho, ktorým smerom sa šíri signál na [3]:

- Dopredné (feed-forward) – signál sa tu šíri iba od vstupných neurónov cez skryté až ku výstupným. Takáto neurónová sieť sa nachádza aj na obrázku 5.12.
- Rekurtné neurónové siete – signál sa tu môže pohybovať aj smerom naspäť, od výstupu ku skrytým neurónom, dokonca niekedy aj ku vstupným.

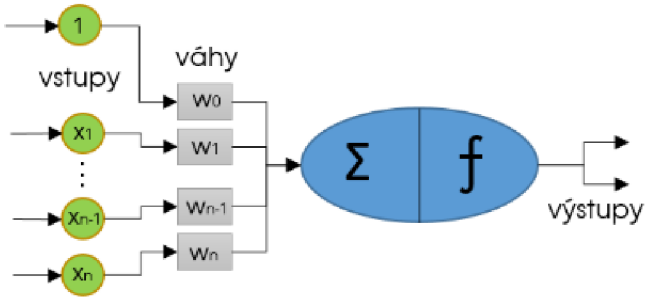
Model neurónu

Biologický model neurónu pozostáva z tela(somy) k nemu sú pripojené dendrity a jeden axon. Rozhranie medzi dendritom jedného a axonom druhého neurónu sa nazýva synapsia. Slúžia na prenos signálu z neurónu, pričom sila prenášaného signálu závisí na sile danej synapse [14].



Obr. 5.13: [15] Neurón

Na základe týchto neurónov sa vytvorili umelé modely neurónov. Signál sa váhou synapsie vynásobí a to určuje jeho silu. Prijaté signály sa sčítajú a výsledkom je hodnota, ktorá sa vloží na vstup aktivačnej funkcii. Tá rozhodne, čo sa stane so signálom, či neurón vyšle signál a s akou silou [14].



Obr. 5.14: [14] Umelý neurón

Na obrázku 5.14 je znázornený umelý počítačový neurón - percepton. Sčíta vstupy X_i vynásobené hodnotou váh W_i a po porovnaní výsledkov s prahovou hodnotou generuje výstup s hodnotou 0 alebo 1. prahová hodnota T sa označuje ako bias , jej hodnota je b, pričom b sa rovná b= -T. Matematicky je to vyjadrené nasledovne [14]:

$$\text{výstup} = \begin{cases} 1 & \text{ak } z = \sum(x_i * w_i) + b > 0 \\ 0 & \text{ak } z = \sum(x_i * w_i) + b \leq 0 \end{cases}$$

Výhody

- Väčšinou neurónové siete majú schopnosť sa učiť.
- Návrh takýchto sieti sa dá ľahko prispôsobiť.

Nevýhody

- V niektorých prípadoch môže byť učenie sa zložité.
- Vzhľadom na vnútornú zložitosť operácií siete nie je všeobecná cesta prístupu k týmto operáciám.
- Zložitá predpoveď výkonnosti budúcej siete.

5.2.2 Algoritmy strojového učenia

Algoritmy strojového učenia poskytuje systému schopnosť automaticky sa učiť a zlepšovať sa, na základe existujúcich príkladov z minulosti alebo z vlastných skúseností. Proces učenia sa začína pozorovaním alebo skúmaním dát, prípadne získavaním nových skúseností. Následne v nich program dokáže nájde vzory a pomocou nich vie zlepšiť svoje rozhodnutia a dosiahnuť svoj cieľ. Hlavným cieľom je dokázať, aby sa počítač učil samostatne a bez toho, aby bol pri učení odkázaný na ľudskú pomoc. To, čo sa naučil, by mal vedieť aj prakticky aplikovať [6].

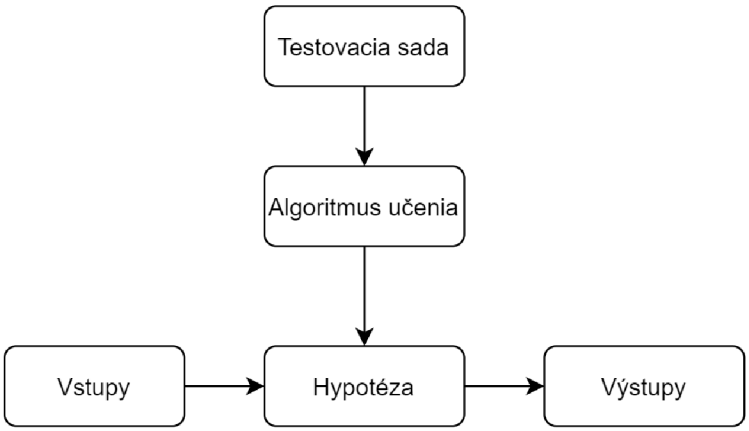
Podľa toho, ako jednotlivé procesy učenia prebiehajú, môžeme algoritmy strojového učenia rozdeliť do troch nasledujúcich skupín:

- učenie s učiteľom (Supervised machine learning)
- učenie bez učiteľa (Unsupervised machine learning)
- učenie formou odmeňovania (Reinforcement learning)

Učenie s učiteľom (Supervised machine learning)

Aktuálne najviac používaných algoritmov strojového učenia patrí do tejto skupiny. Pri takomto učení sa, v testovacej fáze, musí byť prítomný učiteľ. Musí mať prístup k správnym príkladom. Pre každý vstup (objekt, ktorý sa snaží naučiť, aby ho algoritmus rozpoznal) musí poznať správny výstup [5].

Napríklad by sme chceli algoritmus naučiť rozpoznávať čísla od 0 po 9, ktoré sú na obrázku napísané rukou. Forma vstupu by bola obrázok s napísaným číslom, plus aké konkrétne číslo je na obrázku. Čím viac takýchto vstupov sa aplikuje z testovacej sady čísel, tým by algoritmus bol presnejší. Na základe týchto obrázkov sa algoritmus naučí ako vyzerajú jednotlivé čísla, a keď dostane ako vstup nový obrázok, ktorý počas tréningu „nevidel“, vie na základe toho čo sa naučil vyhodnotiť, o aké číslo ide. Učenie s učiteľom je teda silne závislé na dostatočnom množstve kvalitných dát [5].



Obr. 5.15:

Naučiť sa, ako takýmto spôsobom vykonávať úlohy, je na základe explicitného príkladu relatívne ľahko zrozumiteľné. Môžeme to urobiť iba vtedy, ak budeme mať prístup k množine správnych párov vstup-výstup. Pri našom príklade na čísla písané rukou to znamená, že v určitom okamihu musíme poslať človeka, aby klasifikoval obrázky v testovacej sade [5].

Zoznam algoritmov [7]

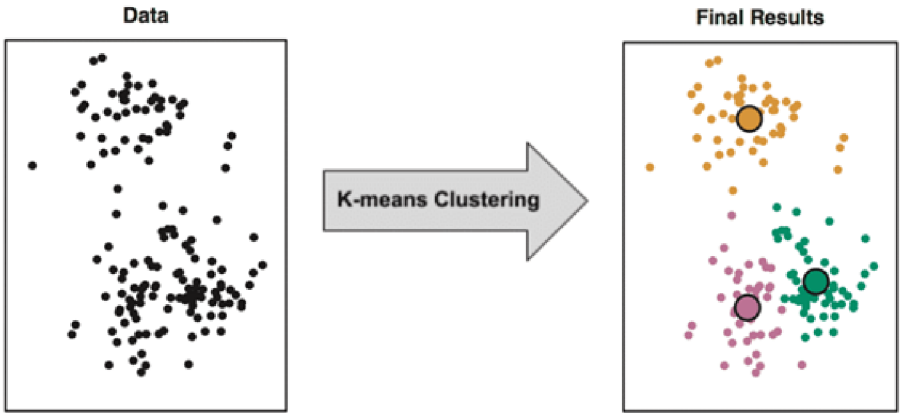
- Najbližší sused (Nearest Neighbor)
- Rozhodovacie stromy (Decision Trees)
- Lineárna regresia (Linear Regression)
- Neurónové siete (Neural Networks)

Učenie bez učiteľa (Unsupervised machine learning)

Učenie bez učiteľa (Unsupervised machine learning) sa používa pri dátach, ktoré neboli vopred klasifikované. Chýba nám druhá časť vstupu „správna odpoveď“, ktorá označovala, čo sa na obrázku nachádza, ako to bolo pri predchádzajúcom spôsobe učenia.

Takéto algoritmy sú obzvlášť užitočné v prípadoch, kedy človek nevie, čo v údajoch hľadať, načo sa zamerať. Snažia sa na základe vstupných údajoch a pravidiel, zisťovať vzory, sumarizovať a zoskupovať údajové body, ktoré pomáhajú odvodiť zmysluplné informácie a lepšie opísať údaje používateľovi.

Výsledkom algoritmu teda nieje správny výstup. Cieľom je preskúmať dáta a odhaliť skrytú štruktúru v týchto neoznačených dátach. Najčastejšími problémami, ktoré rieši učenie bez učiteľa, sú zhlukovanie a asociácia [7] [12].



Obr. 5.16: [11]

Na obrázku 5.16 je ukážka zhlukovania pomocou k-means algoritmu. Vstupné dáta sa rozdelia do tried podľa určitých podobností.

Úlohou zhlukovania je spájať dáta do skupín, ktoré „majú niečo spoločné“. Bežnou aplikáciou je marketing, kde chceme segmentovať zákazníkov do skupín s podobnými preferenciami, správaním. Pri zoskupovaní je často ťažké alebo nemožné vedieť, koľko zoskupení by malo existovať alebo ako by zoskupenia mali vyzerať [5].

Cieľom asociácie je zas nájsť tzv. asociačné pravidlá, ktoré popisujú skupiny dát a vzťahy medzi nimi. Napríklad ak si človek kúpi vec X, tak potom si bude chcieť kúpiť aj vec Y [12].

Zoznam algoritmov [7]

- Asociační analýza (Association rule)
- Algoritmus k-means (k-means clustering)

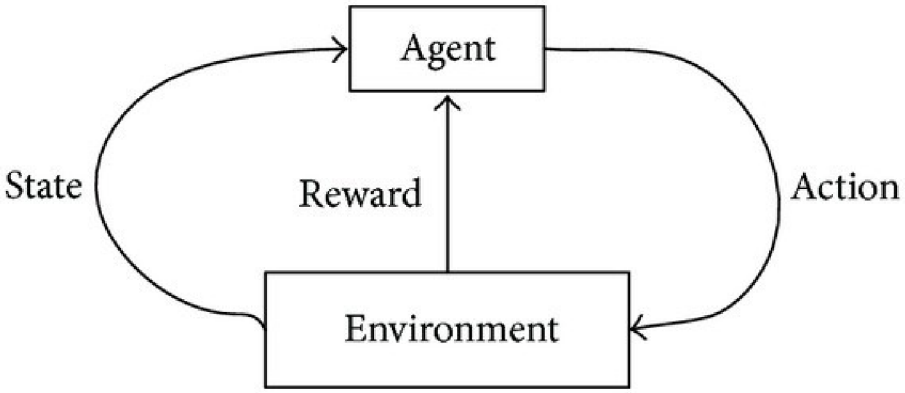
Učenie formou odmeňovania (Reinforcement learning)

Samostatnou kategóriou algoritmov strojového učenia je učenie formou odmeňovania (Reinforcement learning). V tomto prípade už na trénovanie modelu nepoužijeme žiadne označené, či neoznačené trénovacie príklady tak, ako v predchádzajúcich formách učenia [13].

Učenie tu prebieha tak, že vytvoríme systém s agentom, ktorého nasadíme do prostredia a necháme ho nech sa učí prostredníctvom interakcie s prostredím. Agent sa učí zo svojich skúseností, kým nepreskúma celý rad možných stavov [13].

Jediné čo mu musíme určiť sú pravidlá a odmeňovaciu funkciu. Pravidlá určujú ako sa v danom prostredí môže správať. Pomocou odmeňovacej funkcie vie agent vyhodnotiť či rozhodnutie, ktoré práve vykonal bolo preňho prospešné alebo nie. Podobne ako človek skúša jednotlivé možnosti a tým sa učí ako sa má ideálne správať v jednotlivých situáciách. Využíva na to jednoduchú metódu pokus-omyl. [13].

Na to, aby sa agent dozvedel, ktorá akcia je najlepšia, je potrebná jednoduchá spätná väzba.



Obr. 5.17: [7]

Na obrázku 5.17 je znázornený spôsob takéhoto učenia formou odmeňovania. Agent vykoná nejakú akciu (Action) do okolia, prostredia (Environment). Na základe tejto akcie sa dostane agent do nového stavu (State) a získa odmenu (Reward).

Jedným z príkladov pre učenie formou odmeňovania môže byť hra šach. Vytvoríme si tu agenta. Definujeme mu pravidlá na pohyb pre každú figúrku. Nastavím mu aj pravidlo, pri ktorom môže vyhrať hru. Pri vyhodení súperovej figúrky alebo výhre nastáva odmena. Naopak potrestáme ho, ak mu súper vyhodí figúrku, alebo prehrá. Následne ho necháme, nech si zahrá sám proti sebe niekoľko (miliónov) partií. Na základe svojich chýb sa tak učí [13].

Zoznam algoritmov [7]

- Q-Learning
- Temporal Difference (TD) Learning
- Deep Adversarial Networks

5.3 Optimalizácie, vylepšenia

V tejto kapitole budú opísané rôzne optimalizácie a vylepšenia, ktoré by sa dali aplikovať na tento nový typ automatu.

5.3.1 Paralelné procesy

Vytvorením paralelných procesov by sa dalo urýchliť rozpoznávanie. Hlavne v časti Kruhového a Vnútorného automatu.

Kruhový automat

Jeden proces by začal prechádzať kruh tak ako doteraz a druhý proces by sa vybral opačnou stranou, smerom nadol. Koniec potom môže nastať v troch prípadoch.

- Oba procesy sa dostanú na rovnakú pozíciu v obrázku. Kruh by bol nájdený.
- Pozícia jedného z procesov by sa dostala za pozíciu druhého. Kruh by sa nenašiel.
- Automat na jednom z procesov by vošiel do chybového stavu. Kruh by sa nenašiel.

Vnútorný automat

V tretej časti automatu, kde sa rozpoznáva vnútro, by sa mohli tiež využiť dva procesy. Jeden by prechádzal vnútro zhora dole, a druhý zľava doprava. Výsledok oboch procesov by bola množina značiek, ktoré zodpovedajú riadku alebo stĺpcu. Na konci by sa vytvoril prienik týchto množín a výsledkom by bola rozpoznaná značka.

Časť automatu Vyhľadávanie

Tak, ako aj v predchádzajúcich častiach automatu, by sa dali aplikovať vylepšenia a optimalizácie aj v tejto časti. Vyhľadávanie potencionálnych počiatočných bodov značky je implementované tak, že prehľadáva vstup po riadkoch smerom z hora nadol.

Jedno z riešení ako by sa dali aplikovať paralelné procesy je, žeby prechádzal vstupný obrázok naraz aj zhora a aj zdola. Vždy keby jeden z procesov našiel potencionálny bod, vyhľadávanie by sa prerušilo. Koniec by nastal po prejdení celého obrázka.

5.3.2 Aplikácia na mobilné zariadenia

Jedným z vylepšení by mohla byť implementácia automatu na mobilnú platformu. Ako vstup by bol obrázok priamo z mobilného zariadenia. Užívateľ by mohol obrázok priamo aj odfotiť. Následne by sa spustil automat, ktorý by prešiel vstup, rozpoznal objekty na obrázku a nejakým vhodným spôsobom by prezentoval výsledok. Napríklad v textovej podobe, alebo vo forme zvuku.

Objekt, ktorý by sa rozpoznával, by nemusel byť iba dopravná značka. Mohli by to byť rôzne iné objekty. Napríklad štátne poznávacie značky aut. Na základe informácii, ktoré by získal zo vstupu, by určil z akého štátu je, poprípade okresu.

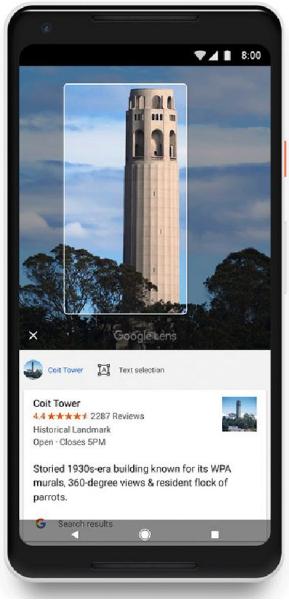
Výsledným produktom by mohla byť aj kalkulačka, ktorá by prečítala vstupný obrázok. Na ňom by bol byť vzorec alebo príklad a výsledkom rozpoznávania by bol výsledok výrazu.

Podobné aplikácie, ktoré by rozpoznávali vstupné obrázky, už existujú. Napríklad je to aplikácia od spoločnosti Google, ktorá sa volá Google Lens. Je dostupná pre zariadenia s operačným systémom Android a iOS. Obsahuje niektoré užitočné funkcie ako sú [4]:

- Dokáže naskenovať čiarový, alebo QR kód
- Z fotky nájde a rozpozná text. Následne ho vie preložiť do iných jazykov. Preklad prebieha rýchlo.
- Tým, že aplikácia Google Lens vie veľmi dobre pracovať s textom, dokáže rozpoznať aj vizitky. Naskenuje potrebné údaje ako meno, číslo a vie ich uložiť rovno do mobilného zariadenia k ostatným kontaktom.

- Taktiež dokáže buď na fotke, alebo priamym namierením fotoaparáta na objekt ho rozpoznať a pridá k nemu pár informácii. Dokáže rozpoznať sochy, budovy, zvieratá rastliny.

Jednou z nevýhod tejto aplikácie je, že aby fungovala správne, musí byť dané zariadenie pripojené k internetu.



Obr. 5.18: [8]

Na obrázku 5.18 je ukážka toho, ako Google Lens funguje. Z fotky určil, že objekt na obrázku je veža Coilt Tower a rovno ukázal aj zopár zaujímavosti k objektu.

Kapitola 6

Záver

Cieľom tejto bakalárskej práce bolo zadefinovať nový typ dvojrozmerného konečného automatu a následne ho implementovať. Pri čítaní tejto práce sa čitateľ dozvedel informácie o rôznych typoch automatov. Na začiatku boli zhrnuté podstatné informácie všeobecne o konečných automatoch. Následne boli vybraté niektoré na ukážku, hlavne tie, na základe ktorých bol definovaný vlastný typ. Ku každému automate bol uvedený aj konkrétny príklad, kvôli lepšiemu pochopeniu a oboznámeniu sa s automatom.

Najdôležitejšou časťou práce bola druhá kapitola, ktorá bola zameraná na nový typ dvojrozmerného konečného automatu. Na začiatku boli uvedené základné definície a charakteristiky automatu. V súčastnosti je veľa objektov na rozpoznávanie, preto bolo nutné zamerať sa iba na jednu skupinu objektov. Stali sa nimi tak dopravné značky, ktoré mali kruhový tvar. Konkrétne to boli obmedzenia na maximálnu povolenú rýchlosť na 50, 70 a 80 km/h.

Rozpoznávanie bolo rozdelené do troch hlavných častí. Každej z nich bola venovaná osobitná pozornosť. V prvej časti pod názvom Vyhľadávanie sa hľadal začiatok objektu, v tomto prípade začiatok značky. Pri úspešnom ukončení vyhľadávania sa automaticky prešlo do ďalšej, ktorou bol Kruhový automat. V nej sa kontrolovalo okolie. Automat prechádzal po okraji a opisoval pri tom kružnicu. Pri neúspechu sa rozpoznávanie vracalo opäť na prvú časť. Keď sa skončila aj táto časť úspešne, pokračovalo sa do poslednej tretej. Tretia časť kontrolovala vnútro. V prípade úspechu bola výsledkom informácia o rozpoznanej značke. Tieto časti sa spúšťali kým sa neprešiel a neskontroloval celý vstupný obrázok.

V ďalších kapitolách bola opísaná implementácia a testovanie. Testovanie bolo prevedené spôsobom, že sa vybralo zopár vstupných obrázkov, ktoré by mohli byť problémom pri rozpoznávaní. Pre každý vstup bolo znázornené a vysvetlené to, ako sa zachová automat v danej situácii.

Implementácia automatu do aplikácie potvrdila, že rozpoznávanie bolo pomerne rýchle a aj presné. Automat čítal vstupný obrázok, prechádzal po pixeloch, Na základe jeho farieb a prechodových funkcií sa dostával do stavov, ktoré určovali aký bude nasledujúci stav a ako bude pokračovať rozpoznávanie.

Na záver boli uvedené aj niektoré už existujúce spôsoby rozpoznávania obrázkov, ktoré nevyužívajú konečné automaty. Jedným z nich sú aj základy algoritmov strojového učenia, ktoré sú tiež vhodné pri problematike procese rozpoznávania.

V rámci budúceho vývoja tohto nového typu automatu by sa dali implementovať aj rôzne vylepšenia. Tým, že pozostáva z troch nezávislých častí, dá sa ľahko rozšíriť a upraviť. Jednotlivé časti by sa dali jednoducho vymeniť za iné, poprípade pridať, alebo odobrať podľa potreby.

Namiesto druhej časti, ktorá je zatiaľ zameraná iba na rozpoznávanie kruhu, by sa mohlo napríklad pridať rozpoznávanie aj na iné objekty ako štvorec, obdĺžnik, alebo trojuholník. Takýmto spôsobom by sa mohlo docieliť rozpoznávanie viacerých druhov dopravných značiek, poprípade iných objektov.

Dôležitou časťou automatu je práve proces vyhľadávania začiatku značky, ktorý sa spúšťa najčastejšie a práve ten volá ostaté časti. Ak by sa na každom pixeli spúšťalo rozpoznávanie, bolo by to neefektívne. Naopak ak by sa vynechala nejaká podstatná časť, objekt by nemusel byť nájdený. Preto je vhodné správne zvoliť algoritmus, ktorý by vyhľadal začiatok objektu. Mohol by napríklad chodiť do kruhu a prechádzať najprv okraje, alebo rovno by išiel smerom do stredu obrázka, kde by sa pravdepodobne skôr mohol objekt nachádzať.

Prechod automatu treťou časťou by sa dal urýchliť. Namiesto kontrolovania každého riadka a stĺpca vo vnútri objektu by kontroloval každý druhý riadok, poprípade stĺpec. Znížila by sa tak presnosť určenia konkrétneho objektu, ale rýchlosť rozpoznania by sa viditeľne zvýšila. Takéto vylepšenie by bolo vhodné pre vstupné obrázky s vysokým rozlíšením.

Do budúcna by bolo možné preniesť automat na mobilnú platformu ako je Android alebo iOS. Postupne by sa pridávali podporované objekty a nielen dopravné značky. Pomocou fotoaparáta by sa získal vstupný obrázok. Užívateľ by namieril mobilné zariadenie na daný objekt a automat by začal proces rozpoznávania. Podobne ako je to v aplikácii Google Lens. Vedel by som si predstaviť, že takýto automat by mohol mať uplatnenie v rôznych odvetviach, hlavne v doprave. Mohol by sa nachádzať v kamerách do áut a upozorňovať vodiča na rýchlostné obmedzenie. Prispelo by to k lepšej bezpečnosti na cestách.

Literatúra

[1] Adamatzky, A.: *Game of Life Cellular Automata*. Springer, 2010. ISBN 978-1-84996-216-2.

[2] AltexSoft: *Image Recognition with Deep Neural Networks and How it's Used* [online]. [cit. 2019-04-30]. Dostupné z: https://www.altexsoft.com/blog/image-recognition-neural-networks-use-cases/.

[3] Babjak, J.: *Neurónové siete sú čiernou skrinkou - Root.cz* [online]. [cit. 2020-04-29]. Dostupné z: https://www.root.cz/clanky/neuronove-siete-su-ciernou-skrinkou/.

[4] Borko, M.: *Google Lens: 5-skvelých funkcií, ktoré urobia z Vášho smartfónu "chytrejšie" zariadenie. Poznáte ich?* [online]. [cit. 2019-04-30]. Dostupné z: https://vosveteit.sk/google-lens-5-skvelych-funkcii-ktore-urobia-z-vasho-smartfonu-chytrejsie-zariadenie-poznate-ich/.

[5] BrainStation: *Machine Learning 101 | Supervised, Unsupervised, Reinforcement & Beyond* [online]. [cit. 2020-04-29]. Dostupné z: https://brainstation.io/blog/machine-learning-supervised-unsupervised-reinforcement.

[6] ExpertSystem: *What is Machine Learning? A definition - Expert System* [online]. [cit. 2020-04-29]. Dostupné z: https://expertsystem.com/machine-learning-definition/.

[7] Fumo, D.: *Types of Machine Learning Algorithms You Should Know* [online]. [cit. 2020-04-29]. Dostupné z: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861.

[8] Hillen, B.: *Google Lens will soon be available in several Android camera apps: Digital Photography Review* [online]. [cit. 2019-04-30]. Dostupné z: https://www.dpreview.com/news/2946014197/google-lens-will-soon-be-available-in-several-android-camera-apps.

[9] Hopcroft, J. E.; Motwani, R.; Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation*. 2. vyd. Addison Wesley, 2000. ISBN 0-201-44124-1.

[10] Learn2code: *C++ Elementary* [online]. [cit. 2020-04-30]. Dostupné z: https://www.learn2code.sk/kurzy/c-plus-plus-elementary.

[11] Luangrath, N.: *Machine Learning Crash Course, Part II: Unsupervised Machine Learning* [online]. [cit. 2020-04-29]. Dostupné z: https://www.iotforall.com/machine-learning-crash-course-unsupervised-learning/.

[12] Muráň, J.: *Algoritmy strojového učenia II. – Učenie bez učiteľa* [online]. [cit. 2020-04-29]. Dostupné z: https://umelainteligencia.sk/algoritmy-strojoveho-ucenia-ii-ucenie-bez-ucitela/.

[13] Muráň, J.: *Algoritmy strojového učenia III. – Učenie formnou odmeňovania* [online]. [cit. 2020-04-29]. Dostupné z: https://umelainteligencia.sk/algoritmy-strojoveho-ucenia-iii-ucenie-formnou-odmenovania/.

[14] Muráň, J.: *Úvod do neurónových sietí 2.časť* [online]. [cit. 2020-04-29]. Dostupné z: https://umelainteligencia.sk/uvod-do-neuronovych-sieti-2-cast/.

[15] Nagyfi, R.: *The differences between Artificial and Biological Neural Networks* [online]. [cit. 2019-04-30]. Dostupné z: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7.

[16] Rozenberg, G.; Salomaa, A., (editori): *Handbook of Formal Languages: Volume 3 Beyond Words*. Springer, 2012. ISBN 9783642638596.

[17] Schiff, J.: *Introduction to Cellular Automata* [online]. [cit. 2020-04-30]. Dostupné z: http://psoup.math.wisc.edu/pub/Schiff_CAbook.pdf.

[18] WolframMathWorld: *Von Neumann Neighborhood* [online]. [cit. 2020-04-30]. Dostupné z: https://mathworld.wolfram.com/vonNeumannNeighborhood.html.

Príloha A

Obsah CD

- **src/** - adresár so zdrojovými kódmi aplikácie
- **xsvacd00.pdf** - text bakalárskej práce
- **doc/** - adresár so zdrojovými kódmi textu bakalárskej práce
- **README** - stručný manuál ku spusteniu