TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

SYSTÉM PRO ROZPOZNÁVÁNÍ A ZPRACOVÁNÍ TEXTOVÉ INFORMACE Z OBRAZU

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Matěj Bartoš**
Vedoucí práce: doc. Ing. Josef Chaloupka, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

SYSTEM FOR RECOGNITION AND PROCESSING OF TEXTUAL INFORMATION FROM A DIGITAL IMAGE

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Matěj Bartoš**
Supervisor: doc. Ing. Josef Chaloupka, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2014/2015

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Matěj Bartoš**
Osobní číslo:	**M12000104**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Systém pro rozpoznávání a zpracování textové informace z obrazu**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Vytvořte aplikaci, která by realizovala OCR převod obrázků s matematickými rovnicemi.
2. Rozpoznaný text upravte na základě zvyklostí použitých při psaní matematických rovnic.
3. Navrhněte aplikaci pro vyhodnocování některých rozpoznaných matematických rovnic.
4. Výslednou aplikaci otestujte na menší databázi (minimálně 100) digitalizovaných obrázků s matematickými rovnicemi.

Rozsah grafických prací:	**Dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 30 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Mark L. Murphy: Android 2 - Průvodce programováním mobilních aplikací, COMPUTER PRESS, EAN:9788025131947**

[2] **Komatineni, S.: Pro Android New York: Apress, c2011, xxii, 1175 s. ISBN 978-1-4302-3222-3**

[3] **Steele, J., Nelson, T.: The Android developer's cookbook: building applications with the Android SDK. Upper**

[4] **Saddle River, NJ: Addison-Wesley, c2011, xv, 339 p. Developer's library. ISBN 0-321-74123-4.**

[5] **Morris, J.: Android user interface development: beginner's guide ; quickly design and develop compelling user interfaces for your android applications. Birmingham, UK: Packt Open Source, c2011, vi, 287 p. ISBN: 978-1-849514-48-4.**

Vedoucí bakalářské práce:	**doc. Ing. Josef Chaloupka, Ph.D.** Ústav informačních technologií a elektroniky
Konzultant bakalářské práce:	**Ing. Karel Paleček** Ústav informačních technologií a elektroniky
Datum zadání bakalářské práce:	**12. září 2014**
Termín odevzdání bakalářské práce:	**15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
vedoucí ústavu

V Liberci dne 12. září 2014

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14. 5. 2015

Podpis:

Poděkování

Chtěl bych poděkovat vedoucímu práce panu doc. Ing. Josefu Chaloupkovi, PhD za užitečné rady při konzultacích, rady a připomínky k této práci a také za pomoc při vývoji aplikace.

Anotace

Bakalářská práce: „Systém pro rozpoznání a zpracování textové informace z obrazu “ se zabývá vytvořením aplikace pro mobilní zařízení běžícím na systému Android, která by rozpoznala textovou informaci z obrazu a poté ji vyhodnotila dle daných matematických pravidel. Práce se zaměřuje na rozpoznání textu z obrazu pomocí optického rozpoznávání znaků a vytvoření vyhodnocovacího algoritmu, který by byl podobný dnešním CAS. CAS je zkratkou pro Computer Algebra System a mezi představitele patří např. Axiom, Maple, Maxima, Reduce atd.

Klíčová slova

OCR, Pattern Matching, Android, CAS, AST

Abstract

This bachelor thesis: “System for recognition and processing of textual information from a digital image” concerns with the creation of an application for mobile devices powered by Android system, which recognizes textual information from a digital image and then evaluates it according to given mathematical principles. The thesis aim is to recognize text from a digital image with the help of optical character recognition. Furthermore it deals with the development of evaluation algorithm, which would be similar to contemporary CAS, where CAS stands for Computer algebra system. As the main representatives for CAS can be mentioned for example Axiom, Maple, Maxima, Reduce, etc.

Key words

OCR, Pattern Matching, Android, CAS, AST

Obsah

Seznam ilustrací

Seznam tabulek

Seznam zkratek

API – Application Programming Interface, aplikační rozhraní knihovny, které může programátor používat.

BOM – Byte order mark, značí endianitu souboru.

CAS – Computer Algebra System, systém pro symbolické počítání.

hOCR – otevřený standard pro výstup formátovaného textu z OCR systémů.

JNI (Java Native Interface) – rozhraní umožňující propojit kód běžící na virtuálním stroji s nativním kódem jazyka C a C++

Leptonica Image Processing Library (Leptonica) – knihovna pro práci s obrazem a obrazovými formáty.

OCR – Optical Character Recognition, soubor metod pro optické rozpoznávání znaků.

1 Úvod

Během posledních let vzrostl zájem o systémy pro automatickou analýzu. Automatická analýza je jedním ze způsobů jak digitalizovat data pro zpracování do počítače. Tradičním způsobem je zadávání dat pomocí klávesnice. Existují různé technologie pro daná vědní odvětví jako např. rozpoznání řeči, kamerové systémy, čárové kódy a optické rozpoznání znaků, které analýzu zprostředkovávají.

Optické rozpoznávání znaků (Optical character recognition, OCR), které poskytuje metody pro konvertování velkých dat do digitální podoby automaticky, přispělo k popularizaci automatické analýzy nejvíce. I přestože se OCR nepovažuje za nejmladší vědní obor je tato tématika nadále zkoumána, jelikož stále neexistuje řešení pro reálná data, které by podávalo stoprocentní výsledky. U reálných dat je potřeba počítat s velkou škálou fontů, s nepřesným zaměřením dokumentů, se zašuměním apod. Dnes se stále zkoumají metody pro robustní a škálovatelné aplikace metod OCR.

V první kapitole jsou nastíněny obecné principy fungování Optického rozpoznávání znaků. V následující kapitole je rozebrán OCR framework Tesseract, což je systém zvolený pro rozpoznávání v aplikaci. Třetí kapitola zahrnuje implementační detaily při vytváření CAS. Následují implementační detaily aplikace pro Android a použití aplikace.

Cílem práce je navrhnout a vytvořit aplikaci pro mobilní telefon s operačním systémem Android, která by rozpoznávala matematické úlohy a prováděla výpočty na rozpoznaných datech. Nutností je nalézt OCR framework pro systém Android a zapracovat jej do aplikace. Dalším bodem je vypracování programu, který řeší transformaci textu do syntaktického stromu, a následně vyřešit či vyhodnotit chyby při nesprávné syntaxi, nebo neschopnosti řešit zadanou problematiku. Propojení transformačního a vyhodnocovacího programu s rozpoznávacím softwarem je posledním bodem při řešení funkčnosti aplikace. Přidruženým cílem je vytvořit databázi nafocených příkladů, na které by se dala funkčnost aplikace po zhotovení vyzkoušet.

Během rešerše se na trhu nevyskytoval žádný program, který by plnil požadovanou funkčnost dané aplikace. Avšak v průběhu vytváření programu pro účely bakalářské práce bylo vydáno několik aplikací, které téměř plnily účel vytvářeného progamu.

Jmenovitě AutoMath Photo Calculator a PhotoMath. Tyto aplikace byly většinou vytvořeny za účelem zviditelnění komerčního OCR frameworku dané firmy.[1]

1 Aplikace PhotoMath je vytvořena společností microblink, která se zabývá počítačovým viděním a rozpoznáváním obrazu https://microblink.com/

2 Optické rozpoznávání znaků

OCR je soubor metod, které realizují převod obrazu tištěného textu do digitální podoby. Optické rozpoznávání znaků se jako většina rozpoznávání dělí na:

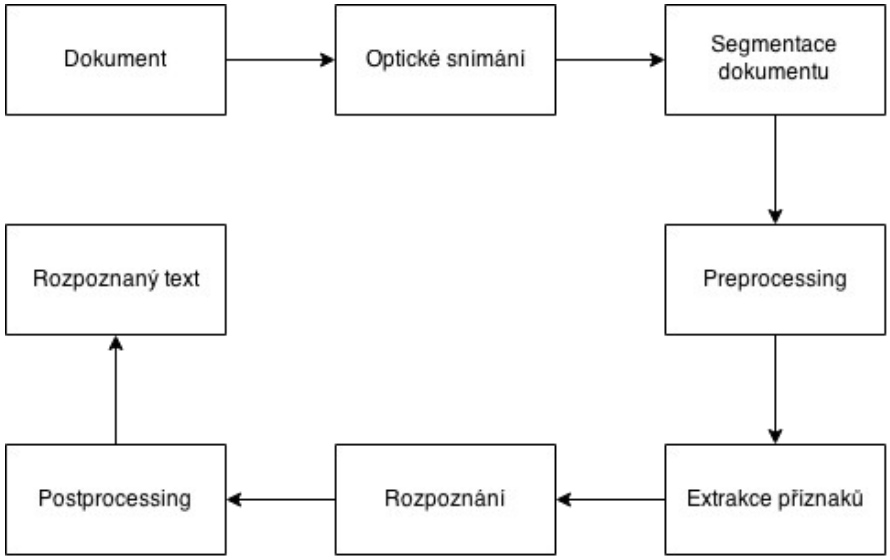
- On-line rozpoznávání – tzn. rozpoznávání dat již během zadávání, např. rozpoznávání znaku již při psaní. Dnes můžeme tuto technologii nalézt například v různých kontrolních aplikacích pro chytrý telefon.
- Off-line rozpoznávání – tzn. rozpoznávání dat poté, co již byla dokončena jejich úprava (např. vytisknutí). Off-line rozpoznávání na trhu stále převládá díky značnému množství již vytvořených aplikací.

Oba případy jsou dnes již zaběhlé, ale kvalita výstupu z těchto aplikací závisí na kvalitě vstupu od uživatele. Digitalizaci textu pomocí OCR dnes můžeme najít např. ve zpracování faktur a jiných účetních dokumentů, při převodu tištěného textu na řeč, nebo v automatickém rozpoznávání státních poznávacích značek aj.

2.1 Postup rozpoznávání metodou OCR

Metody OCR je možné zařadit do oboru Umělé inteligence (Strojové učení – Machine Learning), jelikož snahou OCR je naučit počítač rozpoznávat text jako člověk a tuto činnost tím pádem zrychlit. Hlavním principem automatického rozpoznávání je tedy naučit počítač třídám vzorů a jejich výskytu. OCR software je obvykle učen pomocí učení s učitelem (supervised learning). V OCR jsou jednotlivými třídami vzorů písmena, čísla a speciální symboly. Učení probíhá tím, že se OCR programu předkládají sady obrázků všech tříd vzorů – tedy všech znaků, které chceme rozpoznat, a textu, který je na obrázcích. Z těchto příkladů si OCR program různými algoritmy vyrobí prototyp – popis pro každou třídu. Znak, který chceme rozpoznat, se poté během procesu rozpoznávání porovná s tímto prototypem a na základě nejbližší shody program rozpozná s danou pravděpodobností třídu vzoru. Ve většině komerčních softwarů učení probíhá dříve než se produkt prodá zákazníkovi, ovšem bývá u nich možnost dodatečného učení pro specifické potřeby.

2.2 Části OCR



Ilustrace 1: Části OCR

Proces OCR se skládá z několika kroků viz Ilustrace 1. První nutnou částí je optické snímání, neboli digitalizace původního obrazu, která se může skládat z fotoaparátu či skeneru. Tato část zahrnuje také úpravu celého dokumentu na menší množství dat a ošetření nepřesností. Další aplikovanou metodou je Document segmentation (Location segmentation), tzn. segmentace obrazu na části obsahující text se zachováním kontextové informace. Třetím krokem je Preprocessing, kde se nalezené části dále ošetřují. Poté je potřeba získat informace pro klasifikování, což shrnuje název Extrakce příznaků – popisů tříd. Dále již lze klasifikovat symbol na základě příznaků do tříd. Z pravděpodobnostního ohodnocení tříd získáme nejvhodnější znak. Nakonec se z kontextové informace skládají znaky dohromady, aby tvořily shluky/slova, což pokrývá Postprocessing.

2.2.1 Optické snímání

Pojem optické snímání zahrnuje získání digitálního obrazu, který se dále zpracovává. Jelikož se originální dokument obvykle skládá z černého písma na bílém papíře, je standardem převést případnou barevnou hloubku digitálního obrazu do dvouhodnotové logiky – černá, bílá. Převod probíhá prahováním, které je jednou z nejdůležitějších částí při zpracování, protože celý další proces závisí na kvalitě prahováním převedených dat. Většina dnešního softwaru používá prahování, které rozděluje digitální obraz na segmenty. Poté se snaží v segmentech nalézt optimální práh, a převede segment obrazu

na binarní hodnoty. Na závěr se již všechny binární segmenty spojí do celého binárního obrazu.

2.2.2 Document segmentation

Segmentace probíhá nad binárním obrazem, ve kterém se snaží nalézt písmena či slova. Nalezená slova se poté dělí na písmena, která se rozpoznávají zvlášť. Obvykle je segmentace prováděna izolováním spojených černých částí. Tato technika má ovšem problém, pokud se znaky na původním dokumentu překrývají. Problémy při segmentaci můžeme rozdělit do 3 skupin:

- Extrakce částí, které se překrývají nebo dotýkají.
- Rozeznání šumu od textu.
- Záměna grafiky a textu.

2.2.3 Předzpracování (Preprocessing)

Binární data získaná optickým snímáním vždy obsahují šum. Písmena (slova) mohou být kvůli rozlišení kamery či skeneru, nepřesnostem při zaostřování, či úspěšnosti prahování rozmázána či přerušena. Některé defekty, které by mohly způsobit špatné rozpoznání, mohou být eliminovány uhlazením či normalizováním. Uhlazení značí vyplnění mezer a také zjemnění tlustých čar. Normalizuje se na jednotnou velikost písma, otočení a sklon.

2.2.4 Extrakce příznaků – Klasifikace (Rozpoznání)

Hlavní úlohou extrakce příznaku je získat charakteristiku symbolů, tedy vektor příznaků. Následně je úkolem klasifikace zařadit symbol dle vektoru příznaků do správné třídy. Extrakce s Klasifikací se dělí na dva hlavní přístupy:

- Template matching – porovnávání rastrového obrázku s naučenou databází znaků.
- Feature extraction – snaží se nalézt prototyp, či popis daného symbolu a poté jej porovnat s etalony (reprezentanty) z naučené databáze.

2.2.5 Postprocessing

Postprocessing se snaží slučovat vazby mezi znaky, přičemž využívá kontextovou informaci získanou z předešlého rozpoznávání. Většinou je možné si pod postprocessingem představit vyhledávání ve slovníku. Pokud skupina znaků tvoří slovo

ve slovníku, je považována za správnou. Nicméně pokud není, hledá se nejbližší shoda s dalším slovníkovým slovem, případně vrátí hodnotu, která reprezentuje nerozpoznaná data.

2.3 Systémy pro OCR rozpoznávání textu

Framework je souhrnné řešení obsahující aplikační rozhraní pro snadný vývoj aplikací postavených na tomto řešení a sadu knihoven pro řešení dané problematiky frameworku. Pro vytvoření aplikace bylo nutné získat OCR software, který by dokázal rozpoznat matematické rovnice s určitou přesností. Některé OCR frameworky (OCR software) fungují na online principu zasílání pořízených fotografií na server, kde proběhne rozpoznávání, a výsledek se po zpracování vrátí uživateli. Online OCR jsou výhodné, jelikož snižují zátěž procesoru a stejně tak paměti. Online rozpoznání ovšem vyžaduje stálé připojení a zasílání velkého objemu dat, obzvlášť pokud je účelem kvalitní rozpoznávání. Zatímco při Offline rozpoznávání je celá logika rozpoznávání přenesena na lokální počítač, což vyžaduje větší nároky na procesor a paměť. Výhodou je, že není potřeba odesílat žádná data. Pro potřeby mobilní aplikace bylo zvoleno Offline rozpoznávání, jelikož Online rozpoznávání může být nepříznivé pro uživatele mobilních zařízení (FUP, špatné pokrytí), a to hlavně z důvodu nutnosti odesílat velké množství dat. Dnes již nemusíme vytvářet vlastní OCR, který by splňoval daná specifika, protože se na internetu vyskytují spousty kvalitních frameworků, a některé i open-source.

Mezi nejlepší OCR, které lze nalézt, patří:

- ABBYY FineReader – proprietární software vyvíjen společností ABBYY. Umožňuje převádět naskenované dokumenty, soubory PDF a digitální fotografie na editovatelné dokumenty s možností vyhledávání. Nepřekonatelná přesnost rozpoznání a možnosti převodu fakticky vylučují přepisování na klávesnici a změnu formátu. Intuitivní použití a automatická zadání jedním kliknutím myši umožňuje udělat více v méně krocích[5].
- OmniPage[6] – proprietární software od společnosti Nuance Communications. Podporuje vygenerování do celé řady výsledných formátů od PDF, přes pptx do HTML. Umožňuje připojení do cloudu. Podporuje mobilní platformy.

- Tesseract[7][1] – open-source software od společnosti Google, který vykazuje nejlepší přesnost ve své kategorii.
- SimpleOCR[8] – placený software, který ovšem po zaplacení licence umožňuje zhlédnutí zdrojových kódů pro lepší zapojení aplikace. Jedna z možných verzí obsahuje také freeware verzi OCR softwaru.

Tabulka 1: Porovnání OCR softwaru[12]

	SimpleOCR Freeware	**ABBYY Fine Reader**	**OmniPage**	**Tesseract**
Proprietární	Freeware	Ano	Ano	Ne
Rozpoznání tabulek	Ne	Ano	Ano	Ne
Rozpoznání svislého textu	Ne	Ano	Ano	Ano
Rozpoznání čárového kódu	Ne	Ano	Ano	Ne
Předzpracování obrazu	Ne	Ano	Ano	Ano
Dávkové zpracování	Ne	Ano	Ano	Ano
Počet podporovaných jazyků	3	189	137	> 60

Pro vytvoření funkční aplikace bylo snahou nalézt otevřený software/framework, ve kterém by rozpoznávání fungovalo s dostatečnou přesností. Nejlepším nekomerčním produktem, který je k dispozici, je Tesseract.

3 Tesseract

Tesseract je open source OCR framework, tudíž do něj může každý uživatel přispět. Dnes je dostupný pod záštitou společnosti Google a je napsán převážně v jazyce C a C++. Je možné ho získat pro většinu dnešních platforem, a to konkrétně Linux, Windows, Mac OS X a také pro mobilní platformy Android a iPhone. Tesseract je vybudován primárně pro rozpoznání horizontálně psaných jazyků, mezi které patří např. angličtina, francouzština apod. Framework byl ve svých prvních fázích naprogramován k rozpoznávání pouze anglického jazyka, ale v průběhu let (a za přispění komunity) se počet rozpoznatelných jazyků v poslední verzi přehoupl přes 60. Navíc byla na požadavek uživatelů vytvořena rozpoznávací sada pro matematické výrazy, avšak tato sada se spíše zaměřuje na rozpoznání predikátové logiky a logiky obecně. Proto bylo nutné vytvořit vlastní sadu viz Vytvoření rozpoznávací množiny.

3.1 Vlastnosti a omezení Tesseractu

Tesseract byl primárně navržen pouze pro formát Tiff bez podpory alfa kanálu. Toto omezení bylo zrušeno s přechodem Tesseractu k Leptonica Image Processing Library (dále jen Leptonica), která umožňuje práci s téměř jakýmkoliv obrazovým formátem. Leptonica je nyní implementována uvnitř frameworku a je jeho vstupním i výstupním bodem. Jedním z důležitých faktorů rozpoznávání jsou okraje rozpoznávaného dokumentu. Pokud mají okraje jinou barvu než zbytek obsahu, pak na výstupu z frameworku najdeme neexistující text rozpoznaný z okrajů. Textový výstup je také velmi nepřesný, pokud je původní dokument vychýlený o větší úhel. Uživatel programu by v tomto případě měl dodat jen minimálně šikmý dokument. Tesseract funguje nejlépe s textem, který má alespoň 300 DPI, což je možné si představit jako minimum výšky písma o velikost 10 pixelů. Při nižší velikosti písma či rozlišení přesnost drasticky klesá viz Přílohy-Porovnání vlivu velikosti písma na rozpoznávání. Pro správný chod aplikace by bylo vhodné text menší než 10 pixelů roztáhnout, jelikož rozpoznaný text poté dosahuje vyšší kvality.

V Tesseractu ještě před začátkem rozpoznávání dojde k úpravě vstupních dat. Provede se binarizace obrazu konkrétně Otsuovou metodu, která nalezne optimální práh a poté rozdělí obraz na dvě množiny – popředí a pozadí. Může být vhodné provést před binarizací úpravu původních dat, např. ekvalizací histogramu, nebo jemnějším

prahováním. V aplikaci byly použity obě předešlé úpravy, ale kvalita výstupu z mobilního fotoaparátu nebyla správně rozpoznávána a po dalších úpravách se kvalita dále zhoršovala. Proto byly tyto úpravy během testování aplikace odstraněny.

Framework Tesseract umožňuje získání výstupu buď v UTF-8, nebo ve formátu hOCR (Open standard formát ve formě XML, který definuje rozpoznanou strukturu, pozici a text z rozpoznaného originálu). V nejnovější verzi přibyla možnost výstupu do PDF. Bohužel chybí podpora pro výstup ve formátu Latex.

Faktor, který mění způsob rozpoznávání, je segmentace stránky, neboli jak se rozpoznané kusy textu dále dělí při zpracování. Bez parametru se na vstupu programu očekává soubor s obrazem obsahujícím souvislý text (odstavec). Pokud je v originálním dokumentu text rozptýlen, nebo je jinak formátován, je nutné provést změnu nastavení segmentace obrazu. V případě, že se neprovede změna v nastavení segmentace, je výstup z aplikace velmi nekvalitní.

Tesseract umožňuje několik různých segmentací obrazu, zde je uveden necelý výčet:

0. Automatická segmentace stránky s rozpoznáním orientace a fontu (nefunguje vždy).

1. Předpokládá jeden sloupec s různou šířkou.

2. Předpokládá jeden stejně široký sloupec s vertikálně zarovnaným textem.

3. Předpokládá jeden stejně široký sloupec.

4. Chová se k obrazu jako k jednomu řádku textu.

5. Chová se k obrazu jako k jednomu slovu.

6. Chová se k obrazu jako k jednomu slovu v kruhu.

7. Chová se k obrazu jako k jednomu písmenu.

3.2 Vytvoření rozpoznávací množiny

Pro základní funkčnost aplikace bylo zapotřebí vytvořit kvalitní rozpoznávací množinu. Jelikož standardní rozpoznávací množina nabízená na stránkách Tesseractu nedosahovala potřebných kvalit při rozpoznávání matematických symbolů (jako např. sčítání, odečítání, dělení, matice, integrálu atd.), bylo nutné vytvořit rozpoznávací množinu vlastní. Pro vytváření vlastní rozpoznávací množiny je zapotřebí API

Tesseractu, které je možné užívat i z příkazové řádky[2]. Pro vygenerování rozpoznávací množiny je příkazová řádka naprosto dostačující. Nejprve je nutné získat či vytvořit několik obrázků s požadovaným textem, aby byl Tesseract schopen naučit se rozpoznávat font a znaky obsažené v obrázcích. Pro rozpoznávací množinu bylo vytvořeno 15 obrazů o rozměrech 640x480 s různou velikostí písma a různým fontem, kde jednotlivé obrazy obsahují více příkladů (kolem 15). Tato fáze může být o dost zjednodušena v nové verzi Tesseractu díky funkci text2image, která přetransformuje text do obrázku se zadaným open-source fontem. Na oficiální webové stránce je možné nalézt kód pro vytvoření buildu nové verze. Binární verze pro platformu Windows existují ve starší verzi bez podpory příkazu text2image. Tato funkcionalita zůstala ovšem nevyužita, jelikož bylo zapotřebí vypracovat i rozpoznávání matic, které je nepříjemné takto vytvářet. Pro korektní vytvoření rozpoznávací množiny je nutné, aby byly soubory s obrázky pojmenovány dle standardu (k nalezení na oficiální webové stránce), a to konkrétně [lang].[fontname].exp[num].tif

Lang reprezentuje název pro vytvářený jazyk, ovšem název nemusí být speciální, musí být jednotný v průběhu vytváření rozpoznávací množiny. Jedná se o standard, např. sada pro rozpoznávání angličtiny je pojmenována eng apod. Fontname, obdobně jako lang, reprezentuje název pro font, který je zapotřebí Tesseract naučit, ale pokud má vytvářená sada rozpoznávat více fontů, je pojmenování čistě technické. Num by měl reprezentovat pořadí obrázků. Celé pojmenování pak může vypadat například takto: mat.sans.exp1.tif.

Na správně pojmenované obrazové soubory se aplikuje rozpoznávání obdélníku – boxů příkazem

```
tesseract [lang].[fontname].exp[num].tif [lang].
[fontname].exp[num] batch.nochop makebox,
```

který vytvoří soubor, do nějž se zapíší rozpoznané znaky a koordináty boxů z původního obrázku. Tento soubor je potřeba upravit tak, aby rozpoznané znaky odpovídaly originálu, stejně jako koordináty obdélníků. Pro snadnější manipulaci s tímto souborem je vytvořeno několik programů v různých jazycích, jedním

2 Verzi pro příkazovou řádku je možné si stáhnout z adresy https://code.google.com/p/tesseract-ocr/downloads/list pod jménem tesseract-ocr-setup-3.02.02.exe.

z nejlepších je jTessBoxEditor[3], který byl využit při práci na vytvoření rozpoznávací množiny. Po dokončení validace rozpoznaných znaků je nutné Tesseract naučit rozpoznat dané znaky v obrazu. Toho lze dosáhnout příkazem

```
tesseract [lang].[fontname].exp[num].tif [lang].
[fontname].exp[num] box.train
```

Důležité je dát si pozor na chybové hlášení v podobě APPLY_BOX, tato chyba ohlašuje, že Tesseract správně nerozpoznal obdélník, který byl buď pozměněn, nebo byl špatně rozpoznán už při vytváření boxů. Po dokončení tohoto příkazu pro všechny obrázky by měl být k nalezení dvojnásobný počet nových souborů, které slouží k uložení vnitřních hodnot pro rozpoznání. Dále je nutné vytvořit znakovou sadu z vygenerovaných souborů, což ve skutečnosti zaznamená všechny obsažené znaky v prvním souboru a dále jen rozšiřuje znakovou sadu.

```
unicharset_extractor lang.fontname.exp0.box
lang.fontname.exp1.box...
```

Nyní se musí vytvořit nový soubor s vlastnostmi rozpoznávaných fontů, které byly použity. Ve formátu

```
<fontname> <italic> <bold> <fixed> <serif> <fraktur>,
```

kde každý řádek zastupuje nový font. Tento soubor musí být uložen se znakovou sadou UTF-8 a bez BOM.

Dalším příkazem, který je nutno spustit, je program shapeclustering, jenž slouží pro vytvoření tabulky tvarů, která se vypočítá z dat extrahovaných v předchozích krocích

```
shapeclustering -F font_properties -U unicharset
lang.fontname.exp0.tr...
```

Poté je nutné provést další shlukování tvarů

```
mftraining -F font_properties -U unicharset -O
lang.unicharset lang.fontname.exp0.tr...
```

Předposledním příkazem, který je nutné využít, je normalizace znaků do prototypů.

```
cntraining lang.fontname.exp0.tr...
```

3 Ke stažení na http://vietocr.sourceforge.net/training.html

Nyní je nutné dát vygenerovaným souborům (jmenovitě shapetable, normproto, inttemp, pffmtable) prefix v podobě názvu vytvářeného jazyka, z důvodu pozdějšího zkombinování těchto souborů.

Dále se již zkombinují tyto soubory příkazem

```
combine_tessdata lang.,
```

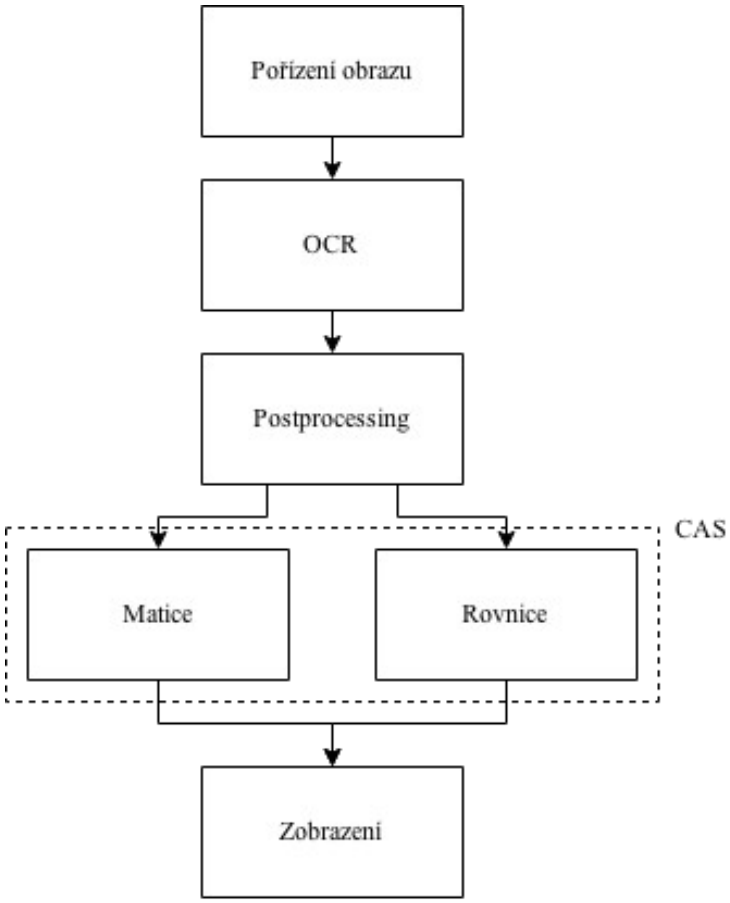
který vytvoří finální soubor připravený k použití. Parametr lang se musí shodovat s prefixem, který zastupuje vytvářený jazyk. Za parametrem lang musí následovat tečka. Pro využívání vygenerované množiny stačí zadat příkaz

```
tesseract input_file.tif output -l lang,
```

kde input_file představuje požadovaný soubor, output požadovaný výstupní soubor a lang název vygenerované množiny.

4 Vytváření aplikace

Aplikaci je možné rozdělit na tři navazující části. První část souvisí s rozpoznáním textu z fotografie či otevřeného souboru fotografie. Rozpoznávání probíhá skrze framework Tesseract přeneseným na platformu Android. Druhá část je vytvoření vyhodnocovacího systému, který zjistí zdali se jedná o text obsahující matematickou rovnici, nebo matici a vrátí vypočtenou informaci na základě uživatelova vstupu. Poslední částí aplikace je navázání Tesseractu na vyhodnocovací systém a poskytnout uživateli grafické prostředí pro komunikaci s aplikací.



Ilustrace 2: Schéma aplikace

4.1 Tesseract na Androidu

Tesseract se bohužel nedá z oficiálních stránek stáhnout ve verzi pro systém Android, ale je možné nalézt aplikaci na Githubu, která poskytuje základní funkcionalitu

frameworku pro systém Android skrze JNI. V případě potřeby lze přidat další funkce frameworku do aplikace využítím nativního rozhraní Javy.

Pro urychlení vývoje byla celá aplikace postavena na experimentálním open source programu https://github.com/rmtheis/android-ocr, který rozpoznává text z kamery telefonu. Android-ocr vzniklo sloučením několika projektů, přičemž většina kódu byla adaptována z ZXing Barcode Scanner[9]. Aplikace podporuje dva režimy pro rozpoznávání. Odložený režim, kdy se aktivuje rozpoznávání až po stisknutí tlačítka a reálný režim, kdy jsou data z fotoaparátu rozpoznávána automaticky již během náhledu. Tyto režimy byly v bakalářské práci adaptovány pro nové účely při rozpoznávání matematických rovnic. Android-ocr je licencováno pod Apache License verze 2.0.

4.2 CAS

Po získání rozpoznaných dat z Tesseractu bylo nutné vyhodnotit text na základě matematického jazyka. Jelikož aplikace byla budována přírůstkově, byl vyhodnocovací systém vystavěn postupně, aby bylo možné zkoušet aplikaci na reálných datech. Pro vytvoření správného vyhodnocovacího algoritmu byl vybudován systém podobný dnešním CAS[3][4]. Pod zkratkou CAS se skrývá název Computer Algebra System, což by se dalo v češtině přeložit jako systém pro symbolické počítání. K vytvoření CAS je nutné definovat si základní strukturu v podobě povolených operandů a daných operací. Jelikož vyhodnocování CAS v aplikaci plně závisí na rozpoznávání, nebylo nutné při implementaci vytvářet veškeré matematické operace, pouze podmnožinu matematického aparátu. Veškeré v CAS povolené matematické výrazy se musejí skládat ze základních objektů:

- Čísel – reprezentuje počítačem představitelnou množinu reálných čísel.
- Proměnných – mají stejný význam jako v matematice či programování.
- Algebraické operátory – +, -, *, /, ^ (je nutné obvykle do rozpoznávaného textu dodat znak násobení, jelikož se při počítání v matematice často vynechává).
- Funkcí – v realizovaném CAS jsou povoleny pouze základní funkce, a to sinus, cosinus, arcus sinus, arcus cosinus, tangens, logaritmus, exponenciela.
- Operací – integrál, derivace.

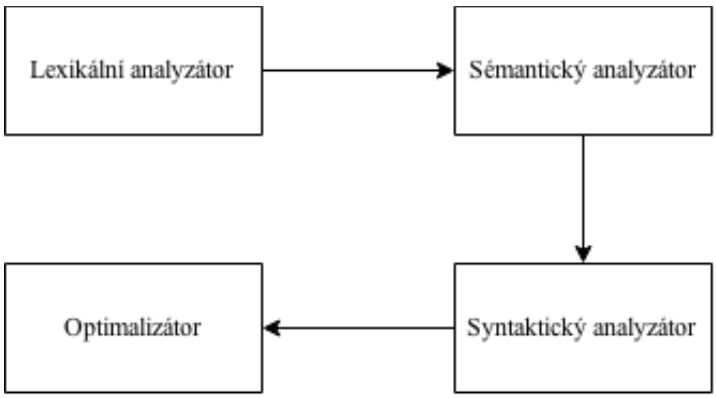
- Relačního operátoru – v systému je pro jednoduchost implementován pouze operátor =.

Pro CAS je nejdůležitější součástí tzv. automatické zjednodušování, které přispívá k vyřešení daného problému postupným aplikováním jednoduchých zjednodušovacích pravidel. Každý ze základních objektů je definován určitou relací k dalším objektům, proto je nutné vytvořit výrazovou strukturu. Výrazovou strukturou je vyhodnocovací strom (Abstract syntax tree, Syntax tree, Evaluation tree). Každý základní objekt rozpoznaný z textu musí být zastoupen ve vyhodnocovacím stromu. Všechny CAS dnes obsahují základní převod textu do vyhodnocovacího stromu. CAS část aplikace tedy musí provádět převod textu do stromové struktury. V případě převodu se aplikace rozděluje na dvě oddělené části, a to parsing matic a parsing ostatních matematických rovnic.

4.3 Parsing matematických rovnic

Převod textu na základě daných kritérií do správné struktury se nazývá parsing. Princip převodu textu matematické rovnice do vyhodnocovacího stromu se obvykle skládá z:

- Lexikálního analyzátoru – zkoumá vstup a na základě pravidel a tabulky povolených symbolů rozdělí vstup na tzv. Tokeny (Lexémy).
- Sémantického analyzátoru – spojuje Tokeny pomocí vazeb a vytváří tzv. syntaktický strom.
- Syntaktického analyzátoru – kontroluje správnost syntaktického stromu.
- Optimalizátor/Simplifikátor – zjednodušuje syntaktický strom.



Ilustrace 3: Struktura parsingu matematických rovnic

4.3.1 Lexikální analyzátor

Lexikální analyzátor v případě aplikace analyzuje text získaný z rozpoznávacího softwaru. Podporovanou podmnožinu jazyka matematiky je nutné si jasně definovat, abychom zabránili nesprávným symbolům. V tuto chvíli aplikace podporuje základní sadu matematických symbolů.

$$a+b,\ a-b,\ a\cdot b,\ a/b,\ a^b,\ \sin(a),\ \cos(a),\ \ln(a),\ \exp(a),\ \int(a),\ (a)',\ldots$$

Pro lepší přehled podporovaných operandů, funkcí a symbolů viz Tabulka tokenů. Token (Lexém) reprezentuje jeden symbol v tabulce symbolů a obsahuje textový úsek daného symbolu a jednoznačný identifikátor. Pro rozdělení vstupního textu na jednotlivé tokeny slouží regulární výrazy, tzn. pro každý symbol v tabulce existuje regulární výraz, který popisuje jak symbol vypadá v textu. Na základě regulárních výrazů se cyklicky rozpoznávají v textu symboly, ze kterých se vytvářejí Tokeny. Vytvářené Tokeny se vkládají do seznamu, který poté již obsahuje pouze povolené symboly. Pokud se na vstupu objeví symbol, který není povolen, je celý vstup zahozen a aplikace zobrazí zprávu s popisem chyby na vstupu. Jednotlivé části Lexikálního analyzátoru je možné v aplikaci nalézt ve třídách *cas.parser.Token* a *cas.parser.Tokenizer*.

Tabulka 2: Tabulka tokenů

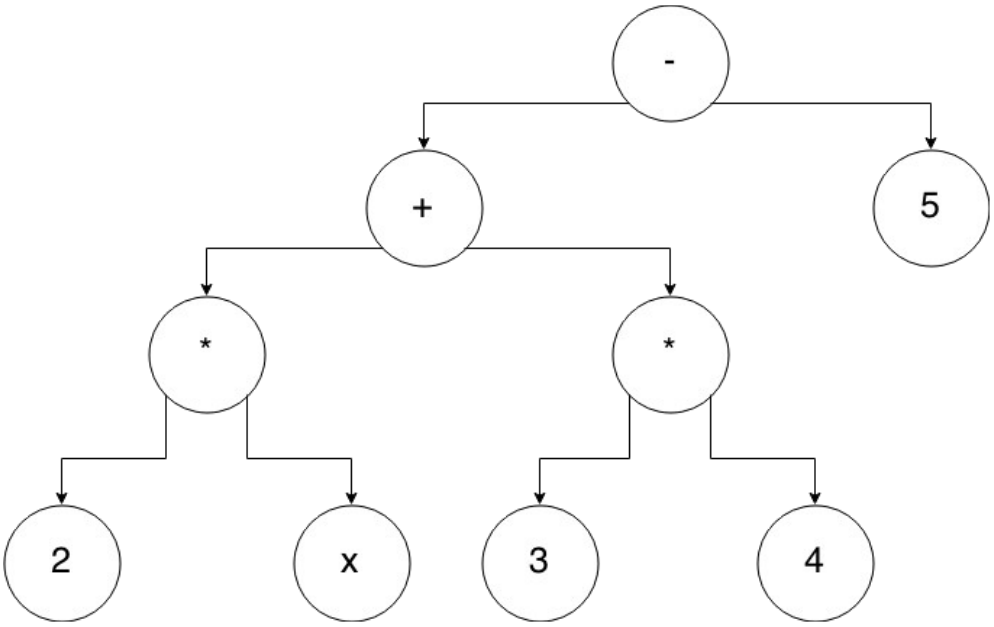
Token	**Regulární výraz**	**Popis**
PLUSMINUS	^([+-])	Token je symbolem pro operátor sčítání a odčítání
MULTDIV	^([*/])	Token představuje operátor pro násobení a dělení

Token	Regulární výraz	Popis
RAISED	^(\^)	Token značící mocninu
FUNCTION	^(cos|sin|...apod.)	Token reprezenující všechny funkce
OPEN_BRACKET	^(\()	Token představuje otevřenou závorku
CLOSE_BRACKET	^(\))	Token je symbolem pro zavřenou závorku
NUMBER	^((?:\d+\.?|\.\d)\d*(?:[Ee][-+]?\d+)?)	Token představuje jakékoliv číslo (celé i desetinné)
VARIABLE	^(d[a-zA-Z|π]\w*)	Token reprezentující proměnnou, která podobně jako v programovacích jazycích musí začínat písmenem a může dále pokračovat písmenem či podtržítkem
EQUAL	^(=)	Token je symbolem pro operátor porovnání
DERIVATE	^('|d/d[a-zA-Z|π]\w*)	Token představuje operátor derivace a to buď výraz ohraničený v závorkách s dodatečným symbolem ', či dodaným textem d/dx, pokud se má derivovat podle proměnné x
INTEGRAL	^(∫)	Token, který značí symbol ∫
INTEGRATION_BY	^(d[a-zA-Z|π]\w*)	Token reprezentuje dle čeho se bude integrovat obdobně jako u derivace
UNIVERSAL	^(_\w+)	Token, který představuje jakoukoliv z předcházejících hodnot. Existuje pro účely rozšiřitelnosti aplikace
EPSILON		Token značící chybu při parsingu

4.3.2 Sémantický analyzátor

Jediným cílem sémantického analyzátoru je vytvořit syntaktický strom, na základě kterého analyzátor vyhodnotí správnost zapsaných symbolů a operací. Struktura syntaktického stromu představuje relace mezi operandy a operátory. Operandy představují konečné uzly (listy, terminal nodes) ve stromu. Operátory jsou naopak n-ární uzly dle relace operátoru. Syntaktický strom se vytváří na základě vstupní sady tokenů, která je výsledkem Lexikálního analyzátoru a souhrnu pravidel (gramatiky).

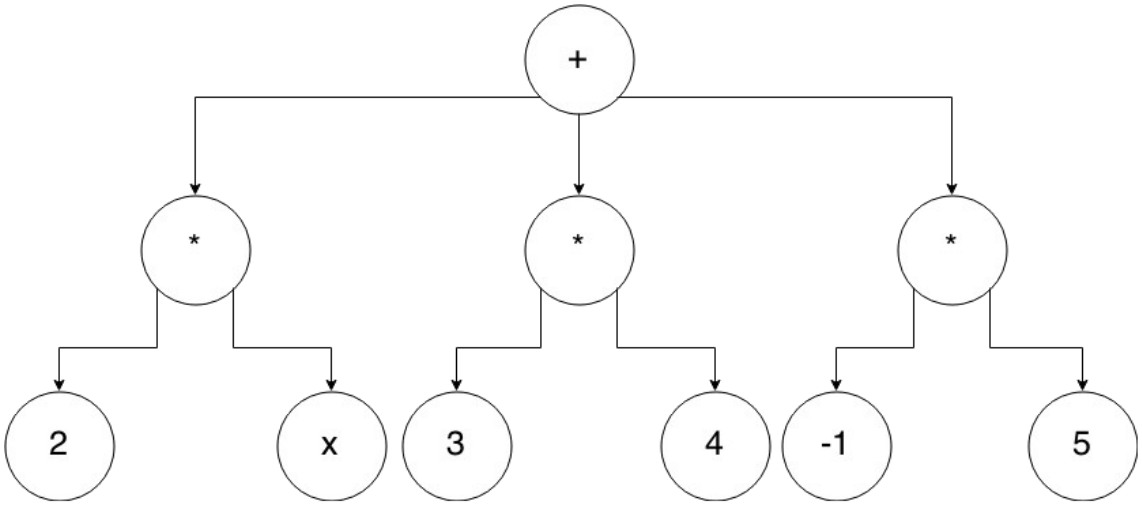
V tomto stromu je operátor s nejnižší prioritou kořenem stromu. I přes tuto definici je možné převést text do syntaktického stromu několika různými způsoby. Příklad jednoho z možných převodů příkladu $2x+4*3-5$ do vyhodnocovacího stromu je možné nalézt na Ilustrace 4.



*Ilustrace 4: Možná reprezentace vyhodnocovacího stromu na příkladu 2x+4*3-5*

Proto je nutné vyjádřit přesnější popis pravidel operátorů. Pro zjednodušení sestavování stromu a poté vyhodnocování je zavedeno automatické zjednodušování. Operátor plus je brán jako n-ární, namísto unárních či binárních, a neměl by se vyskytovat jako předek dalšího operátoru plus. Podobně je operátor násobení brán jako n-ární, namísto obvyklých binárních, a stejně jako operátor plus se násobení nesmí vyskytovat ve svých přímých podstromech. Dále se v syntaktickém stromu již nemůže vyskytnout symbol dělení, a namísto něj existuje pouze násobení, kde původního

dělitele umocníme na mínus první. Nahrazuje se také operátor rozdílu, který je dále realizován součtem, kde je původní odečítaná hodnota reprezentována násobením této hodnoty mínus jednou. Po tomto automatickém zjednodušení je původní příklad $2x+4*3-5$ převeden do normalizovaného tvaru, který je ztvárněn na Ilustrace 5.



*Ilustrace 5: Normalizovaný vyhodnocovací strom příkladu 2x+4*3-5*

I přestože se může zdát, že normalizovaný tvar vypadá oproti původním tvarům složitější, je normalizovaný tvar zásadní pro správné zpracování, protože již neobsahuje redundantní informace, a je možné jej lépe zpracovat při vyhodnocování. Kromě základních operátorů představujících základní matematické operace, musely být přidány další operátory a operandy pro jednodušší zpracování dat a schopnost aplikace zpracovávat složitější příklady. Přidanými operátory jsou integrál, derivace a přidanými operandy – matice a systém lineárních rovnic.

Tabulka 3: Tabulka povolených uzlů syntaktického stromu

Název uzlu	**Popis**
VARIABLE_NODE	Uzel, který představuje proměnnou.
CONSTANT_NODE	Uzel reprezentující číslo, a to celá čísla i desetinná.
ADDITION_NODE	Uzel zastupující součet.
MULTIPLICATION_NODE	Uzel značící násobení.
EXPONENTIATION_NODE	Uzel, který představuje mocninu.
FUNCTION_NODE	Uzel, který zastupuje všechny funkce (jednotlivé funkce jsou odlišeny parametrem uvnitř uzlu).
INTEGRATION_NODE	Uzel reprezentuje integrál s parametrem,

Název uzlu	Popis
	který udává dle jaké proměnné se bude integrovat, pokud uzel neobsahuje tento parametr, je uživatel dále v aplikaci dotázán na doplnění.
DERIVATIVE_NODE	Uzel značící derivace. Funguje obdobně jako uzel představující integraci.
MATRIX_NODE	Uzel zastupuje matici čísel.
EQUATION_SYSTEM_NODE	Uzel, který představuje systém lineárních rovnic. (V budoucnu možno doplnit o nelineární rovnice.)
UNIVERSAL_NODE	Univerzální uzel, který zastupuje všechny předchozí, a tím usnadňuje vyhledávání.

Kvůli zjednodušení aplikace počítá s tím, že systém rovnic bude vždy lineární. Pokud ne, bude uživateli zobrazena zpráva s informací, že aplikace tento systém neumí řešit.

Také pokud Sémantický analyzátor nalezne nekorektní vazbu, ukončí pak svou aktivitu a vyhodnotí chybu. Existují různé způsoby jak implementovat Sémantický analyzátor. Pro potřeby aplikace byl zvolen recursive descent parser, což znamenalo vytvořit na sobě závislé metody, které se volají rekurzivně a představují vyhodnocování souboru pravidel dle gramatiky. Funkčnost Sémantického analyzátoru lze nalézt ve třídě *cas.parser.Parser*

4.3.3 Syntaktický analyzátor

Syntaktický analyzátor by měl plnit funkci kontrolní, a to konkrétně ověřit správnost syntaktického stromu a ověřit možnou posloupnost operátorů. V aplikaci je Syntaktický analyzátor v podstatě již implementován v Sémantickém analyzátoru, který zachycuje nevalidní posloupnost operátorů.

4.3.4 Optimalizátor

Pro potřeby výpočtu jsou implementovány základní matematické operace, které zjednodušují počet operandů a operací v syntaktickém stromu. Každý syntaktický strom projde přes fázi, která jej nejprve normalizuje. Normalizace probíhá na základě seřazení operandů v operátorech. Řadí se dle identifikátoru uzlu, pokud se identifikátory rovnají, řadí se lexikálně. Po normalizaci se přistupuje cyklicky ke zjednodušování. Při

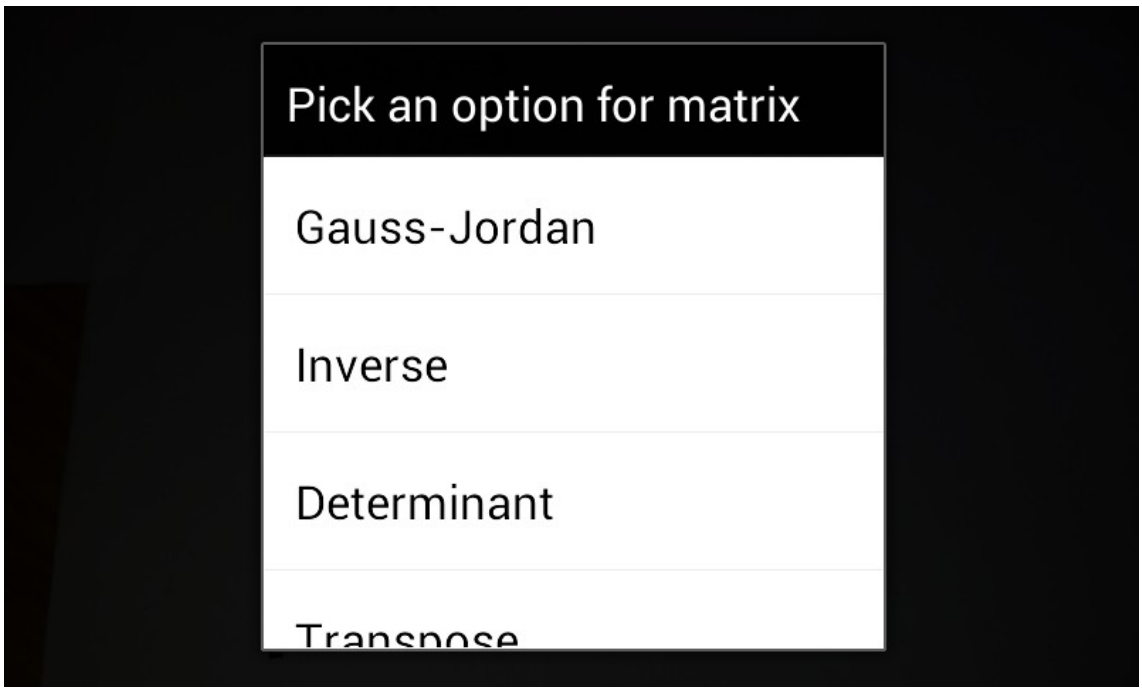
zjednodušování se vyhledají všechny uzly daného typu a proběhne zjednodušení. Poté se vyhledají všechny uzly dalšího typu a aplikují se zjednodušovací transformace atd. Jako příklad se nejprve provádí transformace nad násobením, dále nad součtem, funkcemi, mocninami atd. Příkladem zjednodušovací transformace pro násobení je: $x \cdot 1 = x$, $x \cdot 0 = x$, $x^m \cdot x^n = x^{(m+n)}$, a další.

4.4 Parsing matic

Pro parsing matic je nutné změnit výchozí nastavení aplikace. Parsing matic funguje nejlépe s matematickou rozpoznávací sadou, jelikož při použití ostatních sad rozpoznávání nedosahuje uspokojivých výsledků. Vzhledem k tomu, že Tesseract byl naučen při rozpoznávání s matematickou sadou rozpoznávat matice jako sled znaků ve sloupci oddělených mezerami, bylo již dále jednoduché tyto bloky znaků rozdělit na základě mezery. Pro parsing matic nebyl implementován podobný systém jako u parsingu matematických rovnic, z tohoto důvodu se každý znak v matici považuje za číslo. Po získání všech čísel jsou tato čísla postupně dosazena do dvourozměrného pole reprezentujícího matici. Kvůli zjednodušení struktury aplikace jsou matice zastoupeny operandem ve vyhodnocovacím stromu, který je po parsingu navrácen.

4.5 Vyhodnocování syntaktického stromu

Vyhodnocení probíhá nad kořenovým objektem vyhodnocovacího stromu a nad případnými získanými daty. Další data kromě vyhodnocovacího stromu představují například uživatelovu volbu při vyhodnocování maticového počtu (volba výpočtu transpozice, determinantu apod. viz Ilustrace 6: Zobrazení výběru při vyhodnocování). Na základě kořenového uzlu syntaktického stromu se dále dělí tok aplikace. Pokud je kořenovým uzlem vyhodnocovacího stromu integrál, či derivace, vypočítá se a je vrácen výsledek. Když se v kořenu syntaktického stromu nachází uzel systému matematických rovnic, je na základě systému vypočtena výstupní hodnota, či pokud je kořenovým uzlem stromu matice, je potřebné předat informaci o uživatelově výběru, na základě kterého se vypočítá potřebný výsledek. Aplikace v tuto chvíli podporuje vyhodnocování základního maticového počtu (transpozice, násobení, sčítání, inverze, Gauss-Jordánovu eliminaci), lineárního systému rovnic, kvadratických rovnic a rovnic nižšího řádu, integrace a derivace. Tento výčet by se měl v budoucnu zvyšovat.



Ilustrace 6: Zobrazení výběru při vyhodnocování

5 Vývoj pro systém Android

Aplikace pro systém Android jsou psány v programovacím jazyce Java. Android SDK tools zkompilují kód – s jakýmikoliv daty a prostředky – do souboru APK: *Android package*, což je archivační soubor s příponou *.apk*. APK soubor obsahuje veškerý obsah aplikace vytvářený pro Android, a využívá se pro instalaci na zařízeních běžících pod systémem Android.

Jakmile je aplikace nainstalována na zařízení, je každá Android aplikace v oblasti zabezpečení (security sandbox):

- Operační systém android je víceuživatelským Linuxovým systémem, kde každá aplikace představuje jiného uživatele.
- Ve výchozím nastavení systém přiřadí každé aplikaci unikátní Linuxové uživatelské ID (ID je používáno pouze v systému a aplikace jej nezná). Systém nastavuje povolení pro všechny soubory v aplikaci tak, aby k nim měla přístup pouze aplikace s daným ID.
- Každý proces běží ve svém vlastním virtuálním stroji (virtual machine, VM) tak, aby kód aplikace běžel v izolaci od ostatních aplikací.
- Ve výchozím nastavení každá aplikace běží ve svém vlastním Linuxovém procesu. Android startuje proces, když by měla být jakákoliv z komponent aplikace provedena, a ukončí jej, když už jej není déle třeba, nebo pokud musí systém ušetřit paměť pro ostatní aplikace.

Systém android implementuje princip nejmenších oprávnění. Což znamená, že každá aplikace ve výchozím nastavení má přístup pouze ke komponentám, které vyžaduje pro své vykonávání. To vytváří velmi bezpečné prostředí, ve kterém aplikace nemá přístup k těm částem systému, ke kterým nemá povolení.

Nicméně existují způsoby pro aplikaci jak sdílet data s ostatními aplikacemi, či získat přístup systémové služby:

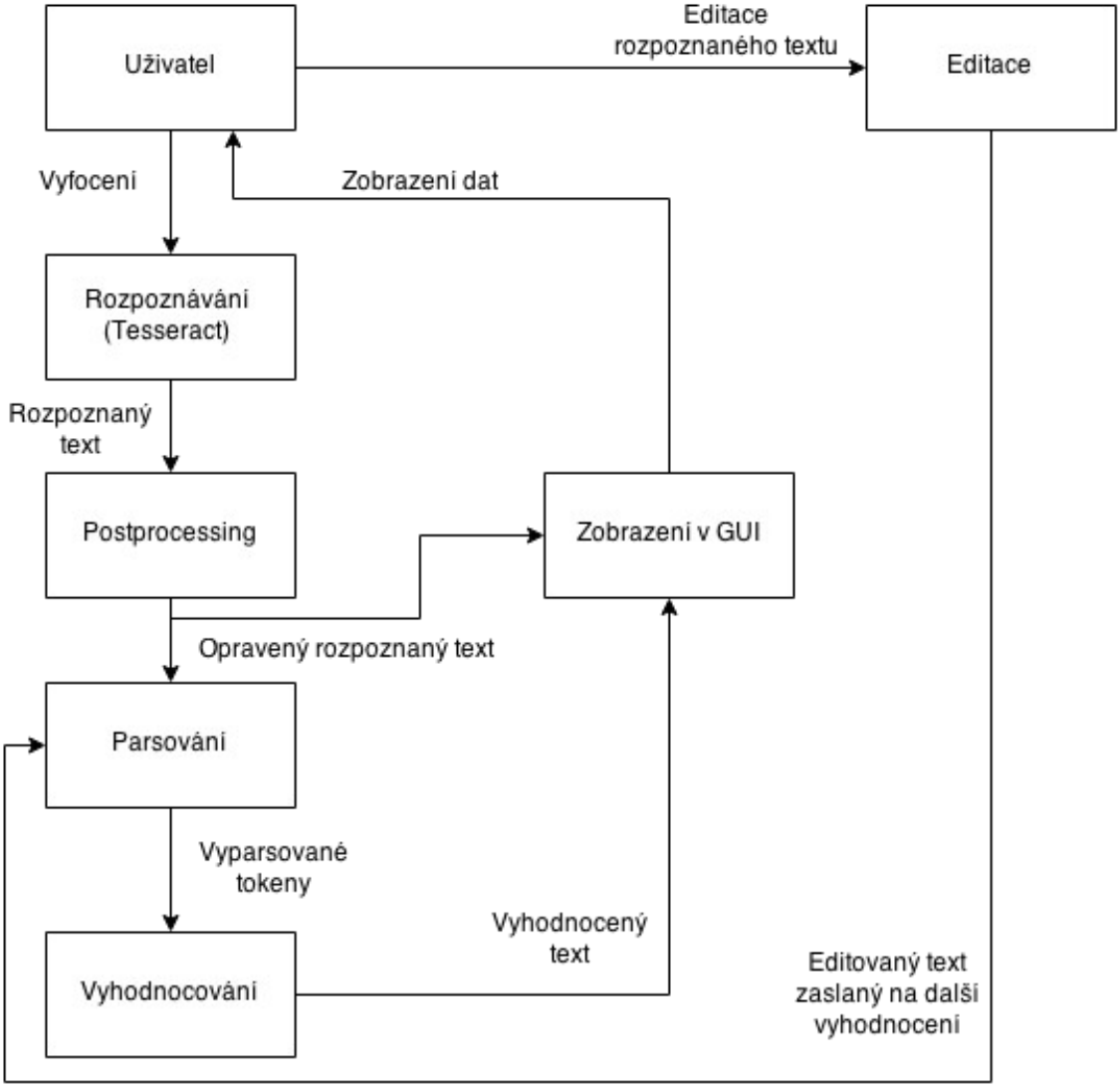
- Je možné, aby dvě aplikace sdílely stejné Linuxové přihlašovací ID, čímž získají přístup k výměně dat. Pro zachování systémových prostředků mohou aplikace se stejným přihlašovacím ID běžet ve stejném Linuxovém procesu a sdílet stejný VM (aplikace musí být podepsány stejným certifikátem).

- Aplikace může požádat o povolení přístupu k datům daného zařízení, jako například fotoaparát, Bluetooth, SD karta a další. Všechny povolení musí být schváleny uživatelem během instalace. [2][10][11]

I přestože Google uvolnil nový nástroj Android Studio, který slouží pro vytváření mobilních aplikací, je tato aplikace stále psána v IDE Eclipse s ADT pluginem, jelikož Android Studio ještě nepodporuje psaní kódu v JNI, což zabraňuje snadnějšímu vývoji v novém IDE. Aplikace je primárně vyvíjena pro Android 4.4, ale je zpětně kompatibilní s nižšími verzemi až do 2.2.x. Pro nainstalování aplikace je nutné povolit tyto povolení (permission):

- android.permission.CAMERA
- android.permission.ACCESS_NETWORK_STATE
- android.permission.INTERNET
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.WRITE_EXTERNAL_STORAGE

5.1 Schéma aplikace na systému Android



Ilustrace 7: Blokový diagram aplikace

Na Ilustrace 7: Blokový diagram aplikace lze vidět schéma realizované aplikace. Startovním bodem diagramu je uživatel, kterému je neustále zobrazován náhled z fotoaparátu. Uživatel specifikuje část náhledu, která by měla být zpracovávaná. Po stisknutí tlačítka pro vyfocení je vyslán požadavek do rozpoznávače, tedy Tesseractu. Rozpoznávač je implementován jako samostatné vlákno, aby nedošlo ke zpomalení odezvy v grafickém vlákně. Následně po dokončení rozpoznání je rozpoznaný text vrácen do hlavního vlákna, kde dojde k postprocessingu. Postprocessing zde zastupuje opravu častých chyb, které Tesseract dělá při rozpoznávání. Z výsledku postprocessingu se uživateli ukáže výsledek rozpoznávání, zatímco výsledek také dále postupuje do fáze parsování, kde dojde k rozdělení rozpoznaného a opraveného textu do jednotlivých

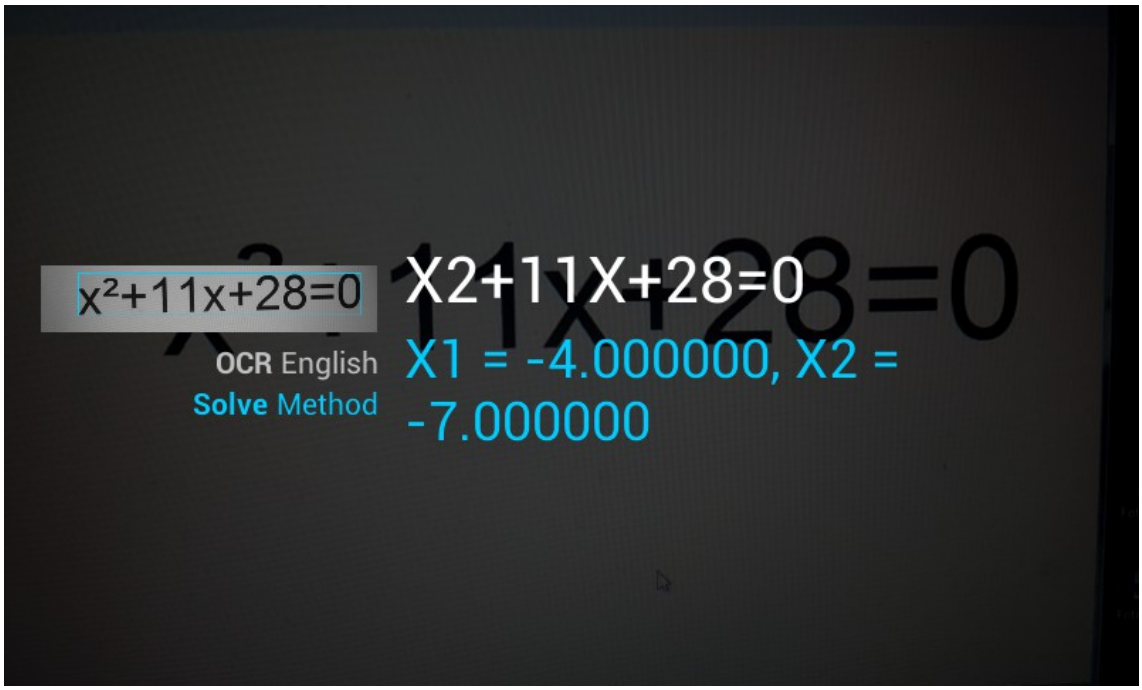
tokenů. Parsování i vyhodnocování je opět tvořeno pomocí vláken ze stejného důvodu jako rozpoznávač. Po získání seznamu tokenů obsažených v původním rozpoznaném textu jsou tato data zaslána do vyhodnocovací části. Z tokenů se zde složí vyhodnocovací strom, který se následně co nejvíce zjednoduší a vyřeší. Poté se vrátí uživateli do grafického zobrazení již vyřešený příklad.

Aplikace také umožňuje rozpoznaný text editovat. Editovaný text je následně opět zaslán do fáze parsování, poté na vyhodnocení a po dokončení vyhodnocení je uživateli graficky zobrazen výsledek upraveného textu.

5.2 Využití Tesseractu v aplikaci

Jelikož Tesseract umožňuje lepší rozpoznávání textu v různých nestandardních případech, bylo vytvořeno menu, kde si zkušenější uživatel může přepnout segmentaci textu, díky které se může zvýšit korektnost rozpoznaného textu. Rozpoznaný text se v aplikaci z frameworku získává ve formátu UTF-8. Jistého zlepšení by dosahovala aplikace pokud by se z frameworku získávala rozpoznaná data ve formátu hOCR, kde by se brala v úvahu pozice v původním obrázku. Ovšem pro vytvoření mechanismu, který by byl schopen zpracovat pozici s rozpoznaným textem, je zapotřebí vytvoření algoritmu, který by formátoval text na základě pozice v původním obrázku. Na druhou stranu by tento algoritmus musel počítat i s nepřesnostmi při rozpoznávání, například nerozpoznání zlomkové čáry, a zároveň by musel brát v úvahu víceřádkové zadání příkladu. V aplikaci tento přístup nebyl implementován z důvodu nejednoznačnosti rozpoznávání a složitého vypracování.

Jelikož je výsledek rozpoznávání získáván ve formátu UTF-8, nastávají problémy při rozpoznávání horních indexů. Po získání UTF-8 textu již není možné rozeznat původní horní index od standardně zapsaného textu. Tento problém je v aplikaci řešen pomocí formátování, které je možné zapnout v nastavení. Formátování probíhá po rozpoznávání a je zabudováno do CAS. Formátování změní všechna čísla nacházející se přímo za proměnnou na horní indexy, tím pádem se čísla za proměnnou stávají mocniteli dané proměnné. Například z x2 se stane x^2.



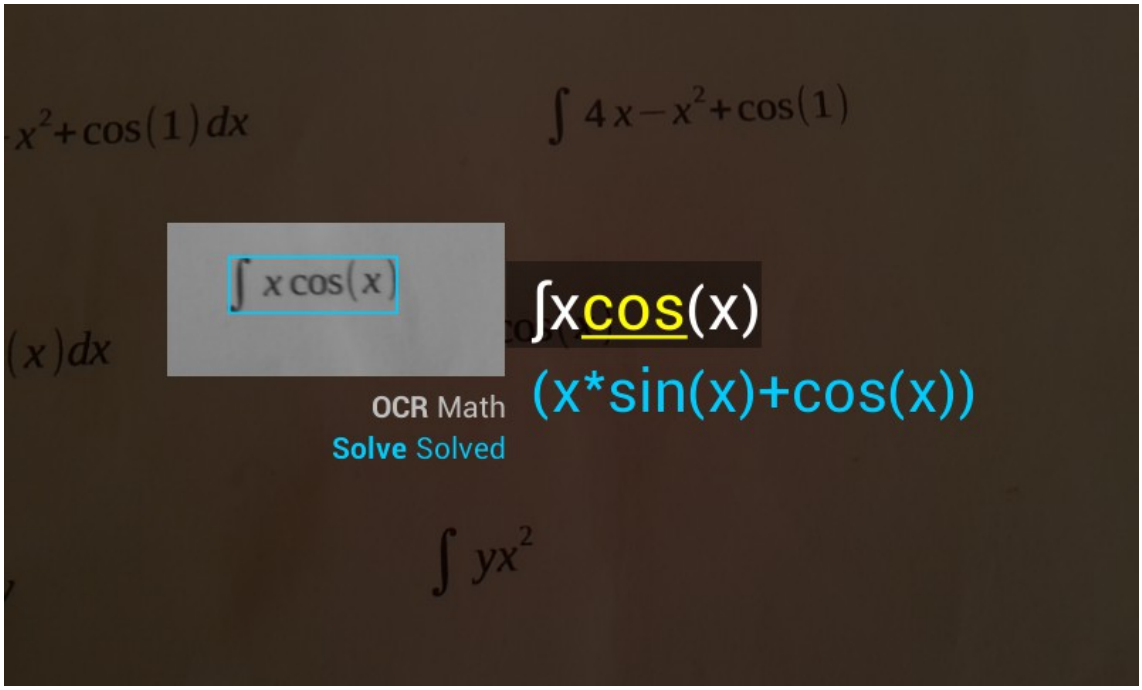
Ilustrace 8: Projev špatně rozpoznaného indexu

Z důvodu, že se v matematice obvykle nepíše znaménko násobení, je nutné jej i přes správné rozpoznání celého obrazu Tesseractem do rozpoznaného textu doplnit. Proto je stejně jako u umocňování vytvořeno formátování textu, které doplňuje tato znaménka na obvyklá místa, např. pro 4xcos(2) bude výstup z formátování 4*x*cos(2).

Dalším pomocným nastavitelným formátováním je separace proměnných majících název delší než jeden znak do více proměnných. Formátování je v aplikaci zabudované v CAS. Možnost zapnutí či vypnutí Separativního formátování víceznakových proměnných je opět možné nalézt v nastavení aplikace. Tedy například pro xyz, které by bylo považováno za jednu proměnnou, bude výsledkem (i podle předchozího pravidla) x*y*z.

Tesseract umožňuje při vytváření rozpoznávací množiny zakomponovat do množiny slovník obvyklých symbolů rozpoznávaného jazyka. Tesseract poté na základě tohoto slovníku mění výstup do pravděpodobnější varianty. Do matematické rozpoznávací množiny ovšem tento slovník přidán nebyl, jelikož při prvních pokusech Tesseract měnil většinu textu právě podle slovníku. Z tohoto důvodu byl do aplikace přidán vlastní slovník (Postprocessing), který formátuje rozpoznaný text z Tesseractu. Postprocessing ošetřuje některé časté případy špatného rozpoznávání a nahrazuje tyto případy obvyklejší variantou. Mezi často špatně rozpoznáváné části textu spadají slova

cos a sin, která se zaměňují např. za c05, 5in apod. Na Ilustrace 9: Oprava špatně rozpoznané části textu lze vidět výsledek opravy častých chyb. Z původního textu ∫xc05(x) se stane ∫xcos(x).



Ilustrace 9: Oprava špatně rozpoznané části textu

5.3 Realizace aplikace

Vstupním bodem aplikace je aktivita CaptureActivity, která představuje UI vlákno a stará se o vykreslování grafických prvků. Zakladní úlohou CaptureActivity je vykreslování náhledu z fotoaparátu mobilního zařízení. Dalšími úlohami jsou vypsání výsledného rozpoznaného textu, zobrazení řešení daného problému a poskytnutí uživateli jednoduchého rozhraní pro orientaci. Pokud dojde v programu k chybě, je uživateli zobrazen Toast (oznamovací pruh obsahující text) s chybou.

Aplikace podporuje dvě možnosti rozpoznávání, a to odložené a reálné (stálé). Odložené rozpoznávání reaguje na stisknutí tlačítka pro rozpoznávání, kdežto reálné rozpoznávání probíhá neustále již při zobrazování náhledu. Při odloženém rozpoznávání aplikace funguje na principu vytvoření vlákna, které zašle data Tesseractu a počká na jeho vyhodnocení, které může při velkém množství dat nebo při kvalitním fotoaparátu na mobilním zařízení chvíli trvat. Výsledek odloženého rozpoznávání je následně

zobrazen v editovatelném boxu aplikace. Pokud je uživatel nespokojen s rozpoznaným textem, nebo by chtěl doplnit některé informace k příkladu, může použít editovatelný box a již znovu nemusí fotografovat požadovaný obrázek. Při reálném rozpoznávání jsou do vlákna neustále zasílána data. Vlákno poté pomocí přepínače (Latch) pošle data do rozpoznávače a ostatní data zahodí, dokud rozpoznávač nevrátí rozpoznaný výsledek. Jakmile je vrácen výsledek rozpoznávání, jsou do Tesseractu zaslána další data. Této funkcionality je dosaženo skrze uchovávání stavů (enum State), handler CaptureActivityHandler, vlákna starající se o vyhodnocení DecodeThread a jeho handleru DecodeHandler, které spolu navzájem komunikují posíláním zpráv. Kvůli možnému delšímu trvání při odloženém rozpoznávání je zobrazen průběh rozpoznávání.

Po získání rozpoznaného textu při odloženém rozpoznávání je text zaslán zpět do CaptureActivity, odkud je dále přeposlán na zpracování parserem. Parser se rozlišuje na základě uživatelovy volby v nastavení, následně je rozpoznaný text zaslán do správného parseru, který po zpracování již vrátí syntaktický strom (ExpressionNode) obsahující veškerá data. Po získání syntaktického stromu se provede kontrola, zda je pro vyhodnocování dostatek informací. Pokud je ve stromu málo informací, znamená to, že rozpoznaný text byl například neúplný, nebo byl špatně rozpoznán. Uživateli by se mělo zobrazit dialogové okno, které jej vyzve k doplnění informací. Z uživatelova doplnění jsou získány vitální informace pro vyřešení příkladu, jako např. volba řešení determinantu, požadavek na Gaussovu eliminaci, či integrál obsahuje více proměnných a v původním obrazu není rozpoznaná nebo zadaná proměnná, dle které by se mělo integrovat apod.

Po získání veškerých nutných informací je vytvořen AsyncTask, reprezentující vyhodnocování s parametrem představujícím syntaktický strom. Vyhodnocování poté pomocí funkcí CAS vyhodnotí syntaktický strom. Výsledek z vyhodnocení je zobrazen uživateli v textovém boxu (Textview). Podobně jako při rozpoznávání je i v případě vyhodnocování zobrazena informace o čekání na vyhodnocení příkladu.

5.4 Struktura aplikace

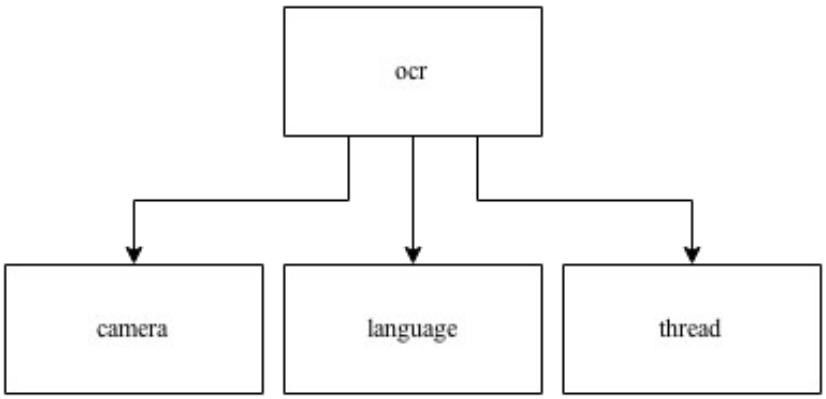
Pro větší přehlednost vytvářené aplikace je potřeba během vytváření programu strukturovat aplikaci do spolu souvisejících funkčních bloků. V Androidu (Javě) je toto zařízeno formou balíčku, který je poté možné nalézt jako složku v adresáři aplikace.

Aplikace je z tohoto důvodu rozdělena do struktury balíčků, které představují hierarchii programu. Mezi dva základní, a zároveň nejdůležitější balíčky patří:

- ocr – zastupuje mobilní aplikaci, která se stará o vykreslování a komunikaci s uživatelem, a zároveň posílá a získává data z CAS.
- cas – reprezentuje vytvářený CAS. Systém sloužící pro vyhodnocování daných matematických příkladů.

5.4.1 Balíček ocr

V balíčku ocr lze nalézt upravenou původní aplikaci, ze které se vycházelo. Navíc obsahuje vlastní metody starající se o získávání, formátování a předávání dat do CAS systému a následné vrácení výsledných dat pro zobrazení.



Ilustrace 10: Balíček ocr

Balíček ocr obsahuje několik dalších podbalíčků:

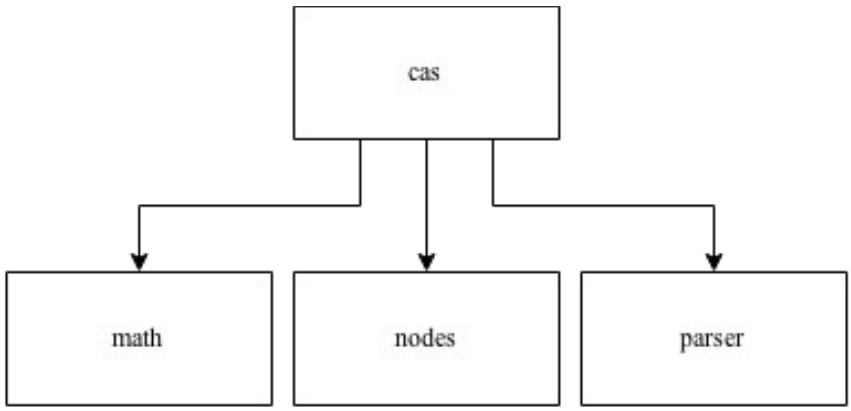
- camera – obsahuje třídy, které využívají a získavají data z fotoaparátu mobilního zařízení.
- language – v balíčku lze nalézt pomocnou třídu, která slouží pro snadnější a přehlednější pojmenování rozpoznávacích množin, kde nové pojmenování je namapováno na původní název. Přejmenování rozpoznávacích množin je vytvářeno z důvodu obvyklého zkratkového názvu rozpoznávacích množin. Třída zaručuje, že je možné získat z nového přehlednějšího názvu originální název rozpoznávací množiny a obráceně. Např. English → eng a eng → English.
- thread – reprezentuje vlákenné metody pro přístup do CAS, které zpřístupňují další funkcionalitu, jako např. uživatelův výběr.

Podbalíček camera je v aplikaci využíván prakticky neustále, jelikož je využíván při zobrazování náhledu fotoaparátu. Kromě náhledu je hlavním účelem balíčku camera získání vyfocených dat a jejich převod do zpracovatelné podoby. V podbalíčku je možné také nalézt třídu zaštiťující nastavení fotoaparátu, což pokrývá zapnutí svítilny, automatické zaostření, či přehrání zvuku při vyfocení.

Dalším podbalíčkem je thread, který obsahuje další balíčky, které slouží pro rozdělení parsovacího procesu od vyhodnocovacího procesu (zaslání dat do CAS). Jelikož je parsování rovnic a matic od sebe odděleno, existují dvě třídy, implementující rozhraní Callable a vracející vyhodnocovací strom (ExpressionNode), které reprezentují parsovací proces viz Parsing matematických rovnic a Parsing matic. Parsery jsou vytvořeny pomocí rozhraní Callable vracející ExpressionNode (reprezentace vyhodnocovacího stromu) a poté z hlavního vlákna volány pomocí ExecutionService. Parsování rozpoznaného textu v těchto třídách dále probíhá pomocí tříd uvnitř CAS v balíčku cas.parser. Vyhodnocovací proces probíhá v podobě třídy implementující AsyncTask, která dostane do parametru syntaktický strom, tedy rozparsovaný text, a na základě kořenového elementu vyhodnotí strom (viz Vyhodnocování syntaktického stromu) a výsledek zobrazí do textview, pokud v průběhu vyhodnocování dojde k chybě, je uživateli zobrazen Toast s chybou.

5.4.2 Balíček cas

Balíček cas je vnitřním modelem CAS. V každém CAS by měl existovat zjednodušovací nástroj. Pro aplikaci byla tato funkcionalita také vytvořena v podobě třídy Simplifier, kterou je nutné zavolat vždy, pokud je nutné strom zjednodušit.



Ilustrace 11: Balíček cas

Balíček je podobně jako předchozí balíček rozdělen do několika podbalíčků:

- math – poskytuje základní matematickou funkcionalitu pro početní úkony nad syntaktickým stromem reprezentující rozparsovaný text. Například třída PolyFunctions pro práci s polynomickými rovnicemi, či třída Calculus, která zpřístupňuje operace integrování a derivování apod.
- nodes – představuje balíček obsahující veškeré operace a operandy povolené v CAS.
- parser – balíček, který slouží pro vytváření syntaktického stromu z daného rozpoznaného textu.

Balíček math obsahuje několik tříd poskytující základní funkcionalitu aplikace:

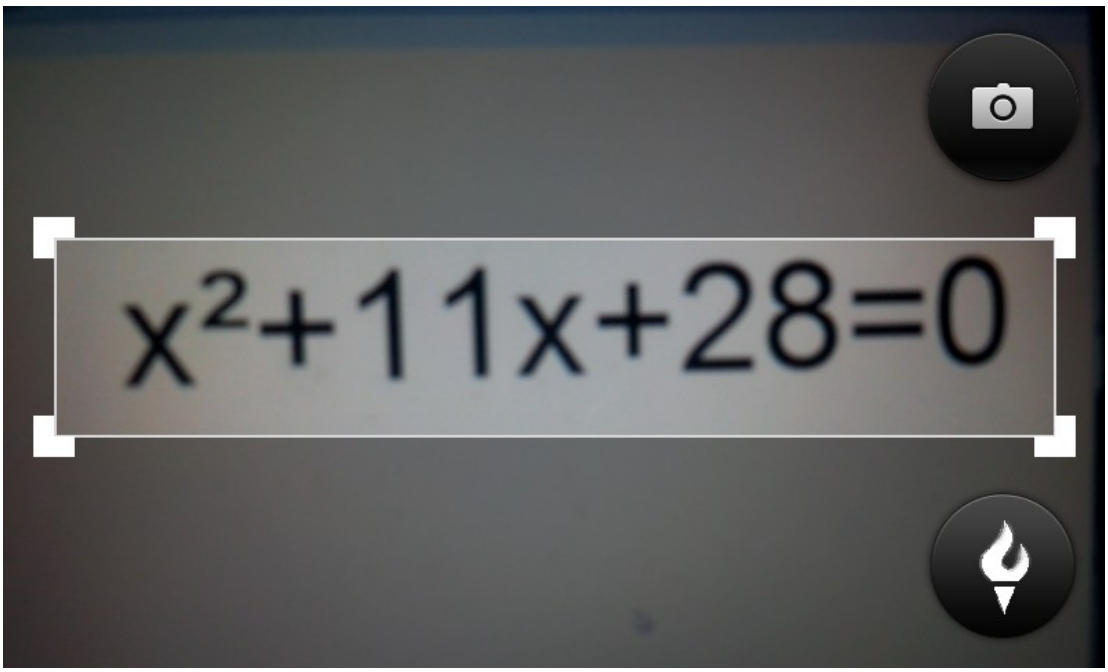
- OrdinaryFuncitons – zahrnuje funkce, které zprostředkovávají snadné sčítání, odečítání, násobení, dělení, mocnění dvou stromů.
- PolyFunctions – představuje práci se stromem jako s polynomickou rovnicí, kde dovoluje následné získání jednotlivých členů polynomu a případně i jeho vypočítání do druhého řádu.
- GPEFunctions – realizuje práci s polynomem obsahujícím více proměnných. Rozhraní pro práci je obdobné jako u PolyFunctions.
- Substitute – dovoluje uvnitř vyhodnocovacího stromu nalézt a zaměnit jeden výraz za druhý.
- Calculus – slouží k vypočtení derivace syntaktického stromu, či vypočítání integrálu (uvnitř třídy se nejprve zkouší různé postupy vyřešení integrálu).

Dalším balíčkem je balíček nodes, ve kterém lze nalézt stromovou strukturu realizující vyhodnocovací strom, a další podpůrné funkce. Z důvodu snadnější rozšiřitelnosti aplikace bylo vytvořeno rozhraní pro jednotlivé operandy a operátory, které toto rozhraní implementovaly pro rozlišení jednotlivých druhů operátorů a operandů viz Přílohy-Ilustrace 13. Bylo vytvořeno několik abstraktních tříd reprezentující obvyklé matematické n-ární operátory SeriesExpressionNode, BinaryExpressionNode, UnaryExpressionNode a abstraktní třída představující operand TerminalExpressionNode.

Aby byl CAS schopen vytvořit z rozpoznaného textu vyhodnocovací strom, bylo nutné vytvořit parser viz Parsing matematických rovnic. Pro zohlednění vytvořených tříd je k dispozici Přílohy-Ilustrace 14, na které lze vidět propojení jednotlivých tříd, kde třída Token realizuje CAS token uchovávající jeden daný lexém určitého typu. Třída TokenInfo v sobě uchovává regulární výraz, dle kterého se poté řetězec dále dělí. Tokenizer na základě jednotlivých tříd TokenInfo separuje vstupní řetězec a vygeneruje spojový seznam tokenů (LinkedList<Token>). Spojový seznam nadále zpracuje třída Parser, která poté pomocí rekurzivně sestupného parseru (recursive descent parser) celý seznam zpracuje a vygeneruje z něj vyhodnocovací strom.

5.5 Funkce aplikace

Aplikace při prvním spuštění vyžaduje připojení k internetu, aby se stáhl balíček se základní rozpoznávací sadou, která slouží pro rozpoznání anglického textu. Po stažení balíčku dojde k rozbalení, uložení a inicializaci OCR rozpoznávání pro anglický jazyk. K aplikaci je přibalena také sada pro rozpoznávání matematického textu. Sada pro rozpoznání matematického textu je v experimentální fázi, jelikož na vytvoření lepšího rozpoznávání by bylo zapotřebí většího množství trénovacích dat. Aplikace poté již při každém spuštění vždy zkontroluje, zdali může v datové paměti mobilního telefonu nalézt poslední vybranou rozpoznávací sadu. Jakékoliv další spuštění nevyžaduje připojení k internetu, jelikož kontrola probíhá buď nad sadou rozpoznávající anglický jazyk, která se stáhla již při prvním spuštění, či sadou pro rozpoznání matematického textu, která je přibalena k aplikaci v části Assets a nemusí být stažena. Po dokončení kontroly rozpoznávací sady je v aplikaci zobrazen náhled kamery s dvěma tlačítky a obdélníkem pro přesnější výběr.



Ilustrace 12: Zobrazení aplikace při spuštění

Aplikace je nastavena pouze pro mód landscape, tedy horizontální položení displeje a rozmístění prvků, které je výhodnější pro přesnější zaměření rozpoznávaného textu. Pro zvýšení přesnosti rozpoznávání v tmavých prostorách je možné zapnout svítilnu fotoaparátu tlačítkem, na kterém je vyobrazena pochodeň. Druhým tlačítkem se již zobrazený náhled vyfotí, uloží a pošle do Tesseractu na rozpoznávání. Po rozpoznání se zobrazí rozpoznaný text a již se začne zpracovávat text pro vyhodnocování. Po skončení vyhodnocování je výsledek zobrazen pod rozpoznaný text. Při stisknutí kontextového menu se vyobrazí nabídka s položkami:

- nastavení,
- načtení obrázku,
- o aplikaci.

Skrze položku nastavení se lze dostat do menu preferencí. Menu preferencí je vyobrazeno jako samostatná aktivita, jak to systém android vyžaduje. Jednotlivé preference jsou důležitou součástí rozpoznávání, jelikož mění způsob průběhu rozpoznávání. Konkrétně můžeme v nastavení preferencí nalézt:

- Změnu rozpoznávací sady,
- přepnutí aplikace do algebraického režimu, či maticového režimu,

- zapnutí dodatečného přeformátování čísel za proměnnými na mocniny,
- povolení pouze jednoznakových názvů proměnných,
- změnu rozpoznávání na reálné rozpoznávání,
- zapnutí zvuku při stisknutí tlačítka pro rozpoznávání,
- zvolení způsobu segmentace obrazu,
- zapnutí režimu automatického zaostření,
- přepnutí na přední fotoaparát (v případě, že ho zařízení obsahuje),
- další možnosti nastavení pro zkušenější uživatele.

V případě rozpoznání textu z obrazu je rozpoznaný text zobrazen v editovatelném boxu, kde si uživatel v případě nesprávného rozpoznání může změnit rozpoznaný text. Po změně stiskne tlačítko hotovo pro dokončení úprav, poté je upravený text znovu zaslán do vyhodnocovací části aplikace. Následně po vyhodnocení je vrácen nový výsledek upraveného textu. Uživatel si může zkopírovat výsledný i rozpoznaný text do clipboradu. Při zaměřování příkladu v náhledu je důležité, aby uživatel, pokud je to možné, co nejvíce zmenšil oblast, ohraničenou obdélníkem, určenou pro rozpoznávání. Zmenšení ohraničené oblasti nejenže zrychlí průběh rozpoznávání z důvodu zmenšení dat vstupujících do rozpoznávače, ale také zpřesní, jelikož Tesseract často chybuje při horších světelných podmínkách.

6 Závěr

Výsledkem této bakalářské práce je funkční aplikace pro mobilní telefon Android, která umožňuje výpočet daných druhů matematických příkladů při vyfocení fotoaparátem. Aplikace zahrnuje rozpoznání textu z obrazu pomocí frameworku Tesseract, který je v aplikaci využit s určitými podmínkami. Tesseract je v podstatě jediným open-source frameworkem, který je schopný rozpoznat text s určitou přesností. Mezi nevýhody Tesseractu patří prakticky nemožné správné rozpoznání horních indexů od normálního textu, pokud je Tesseract nastaven pro výstup do rozpoznaného textu ve formátu UTF-8. Z tohoto důvodu bylo vytvořeno nastavení, které transformuje čísla za proměnnými na horní indexy, pokud si uživatel přeje. Při použití Tesseractu není možné rozpoznat matematické symboly, proto je vytvořena vlastní rozpoznávací sada, která je schopna rozpoznat dané symboly. Během vytváření rozpoznávací množiny bylo vytvořeno několik obrazů s matematickými příklady, které sloužily jak pro vyzkoušení aplikace, tak pro naučení Tesseractu rozpoznávat matematické symboly.

Program pro vyhodnocování, který je součástí aplikace, je po vzoru CAS schopen vypočítat některé matematické příklady. Před vyhodnocením je zapotřebí získat z textu data, nad kterými by bylo možné vyhodnocovat, toho je docíleno rozdělením rozpoznaného textu do tokenů. Vyhodnocení probíhá nad stromovou strukturou, která je vytvořena pomocí sestupného rekurzivního analyzátoru (recursive descent parser) a získanými tokeny. Aplikace je v tuto chvíli schopna řešit základní maticové výpočty (součet, násobení, inverze, eliminace, atd.), systém lineárních rovnic, kvadratické rovnice a rovnice nižších řádů, integrál a derivaci.

Aplikace by v budoucnu mohla být schopna vypočítat větší škálu příkladů. Možnými přidanými druhy pro vyhodnocení mohou být například goniometrické rovnice či systém nelineárních rovnic. Do aplikace by bylo také možno zakomponovat zobrazení mezivýpočtů (průběhu výpočtu) či zobrazení výsledku do grafu.

7 Použitá literatura

[1] S.V. Rice, F.R. Jenkins, T.A. Nartker, The Fourth Annual Test of OCR Accuracy, Technical Report 95-03, Information Science Research Institute, University of Nevada, Las Vegas, July 1995.

[2] *Android Developers* [online]. [cit. 2015-04-05]. Dostupné z: http://developer.android.com/guide/components/fundamentals.html

[3] COHEN, Joel S. *Computer algebra and symbolic computation: elementary algorithms*. Natick, Mass.: A K Peters, 2002, xvii, 323 p. ISBN 15-688-1158-6.

[4] COHEN, Joel S. *Computer alegebra and symbolic computation: mathematical methods*. Natick, Mass.: AK Peters, c2003, xvii, 448 p. ISBN 15-688-1159-4.

[5] ABBYY. *ABBYY FineReader* [online]. [cit. 2015-04-08]. Dostupné z: http://www.abbyy.cz/products/personal/finereader/

[6] Nuance. NUANCE COMMUNICATION. *Nuance* [online]. [cit. 2015-04-08]. Dostupné z: http://www.nuance.com/for-individuals/by-product/omnipage/index.htm

[7] GOOGLE INC. *Tesseract* [online]. [cit. 2015-04-08]. Dostupné z: https://code.google.com/p/tesseract-ocr/

[8] *Simple OCR* [online]. [cit. 2015-04-08]. Dostupné z: http://www.simpleocr.com/

[9] ZXing. *GitHub* [online]. ©2015 [cit. 2015-04-28]. Dostupné z: https://github.com/zxing/zxing/

[10] Mark L. Murphy: Android 2 – Průvodce programováním mobilních aplikací, COMPUTER PRESS, EAN:9788025131947

[11] Steele, J. Nelson, T.: The Android developer's cookbook: building applications with the Android SDK. Upper

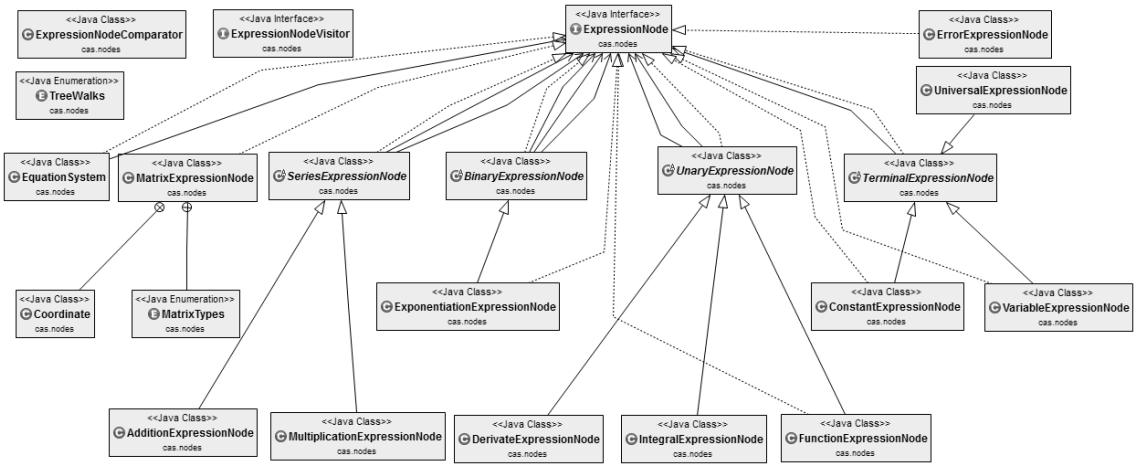
[12] OCR Software Guide: Compare OCR Software. *Simple OCR* [online]. [cit. 2015-05-06]. Dostupné z: http://www.simpleocr.com/OCR_Software_Guide.asp

8 Přílohy vložené na CD

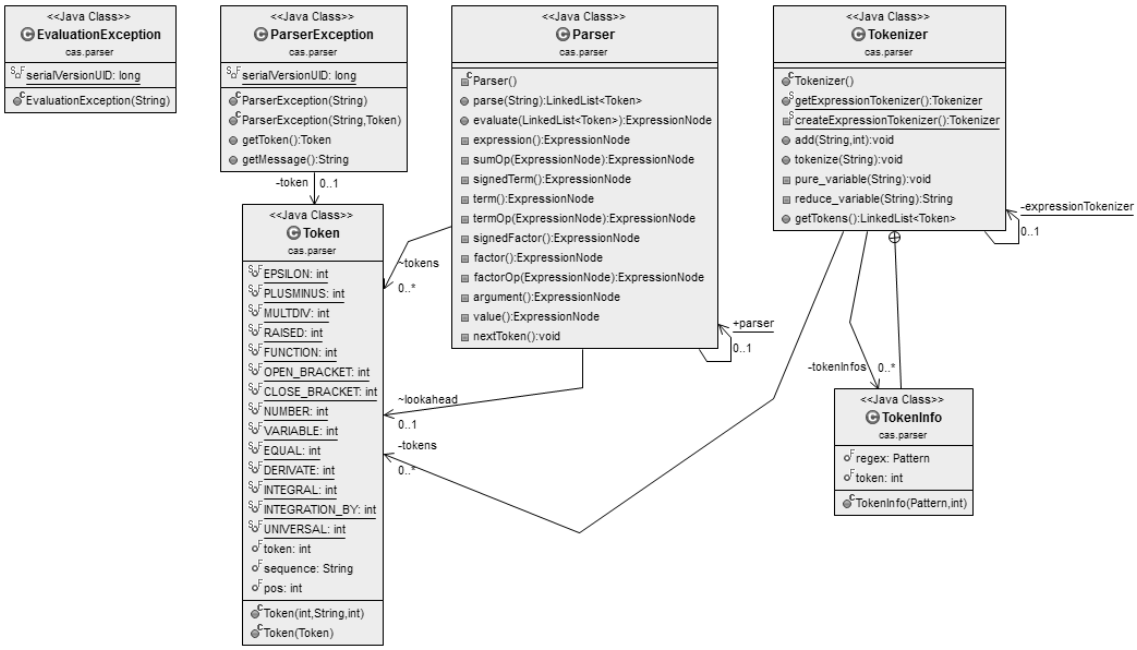
- Text bakalářské práce ve formátu PDF.
- Aplikace pro mobilní telefon se systémem Android.
- Testovací sada obrázků obsahujících matematický text.
- Vytvořená matematická sada pro Tesseract rozpoznávající matematické symboly.
- Verze Tesseractu pro Windows, kterou je možno ovládat skrz příkazovou řádku.

9 Přílohy

9.1 Diagramy tříd



Ilustrace 13: UML diagram tříd balíčku cas.nodes



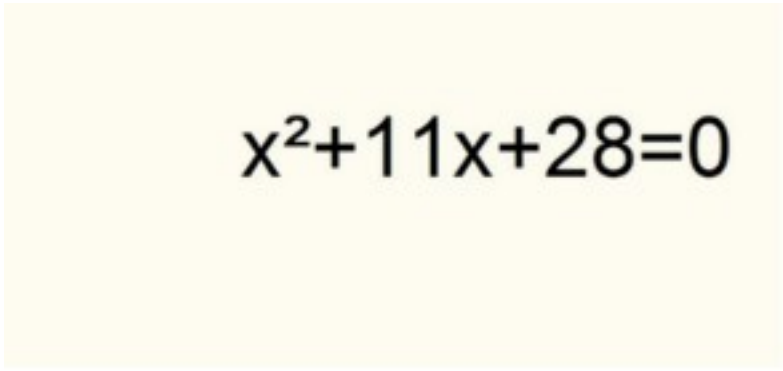
Ilustrace 14: UML diagram tříd balíčku cas.parser

9.2 Porovnání vlivu velikosti písma na rozpoznávání

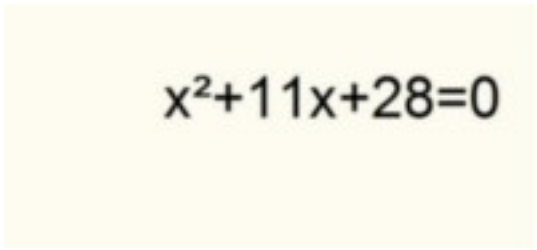
Tabulka 4: Vliv velikosti písma na rozpoznání

Ilustrace číslo	Rozměry	Rozpoznaný text
9	401x186 pixelů	x2+11X+28=O
10	292x137 pixelů	x2+11X+28=O

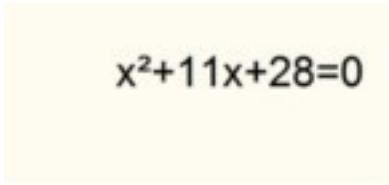
Ilustrace číslo	Rozměry	Rozpoznaný text
11	200x91 pixelů	x'+11X+28=O
12	144x68 pixelů	x=~11X+26=O



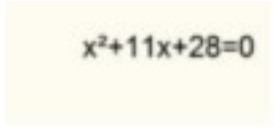
Ilustrace 15: x2+11X+28=O



Ilustrace 16: x2+11X+28=O



Ilustrace 17: x'+11x+28=O



Ilustrace 18: x=~11x+26=0

9.3 Cvičné příklady

x^2+2x+1 $\quad x=-1$

$x+2y=1$
$3+3y=7x$ $\quad x=\frac{4}{17}$, $y=\frac{63}{119}$

$3x+2y=-3z-1$
$2x+2z=-3z-1$
$4x+1=-1y-7z$ $\quad x=2, y=-2, z=-1$

$14+21-3+2=11$ $\quad 34$

$\begin{bmatrix} 3 & 2 & 3 \\ 2 & 3 & 2 \\ 4 & 1 & 7 \end{bmatrix}$ $\quad det=15$

$\int x^2+7x+1\,dx$ $\quad \frac{x^3}{3}+7\frac{x^2}{2}+x$

$(x^2+7x+1)'$ $\quad x+7$

$\int x\cos(x)\,y\,dx$ $\quad y\,x\sin(x)+y\cos(x)$

$\begin{bmatrix} 3 & 2 & 3 & 7 \\ 2 & 3 & 2 & 4 \\ 4 & 1 & 7 & 3 \\ 8 & 3 & 1 & 9 \end{bmatrix}$ $\quad det=-404$