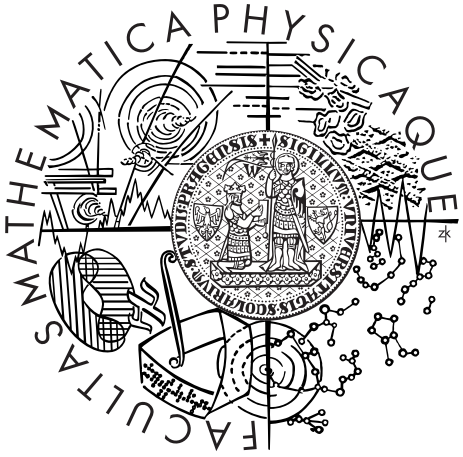
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jiří Marek

Editor rovnic pro Android

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Petr Kučera, Ph.D.
Studijní program: Informatika
Studijní obor: Programování

Praha 2014

Děkuji RNDr. Petru Kučerovi, Ph.D., za rady při vývoji aplikace a psaní této práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 31. července 2014 Podpis autora

Název práce: Editor rovnic pro Android

Autor: Jiří Marek

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Petr Kučera, Ph.D.

Abstrakt: Práce se zabývá implementací aplikace pro mobilní operační systém Android, která umožňuje interaktivní vytváření matematických výrazů a jejich vysázení pomocí programu LaTeX. Následně je schopna tento výsledek uložit a sdílet v různých formátech nebo ho přímo začlenit do rozsáhlejšího LaTeXového dokumentu, který je otevřen v editoru LaTeXu, který je rovněž součástí aplikace. Po začlenění matematického výrazu do LaTeXového kódu je možné tento výraz zpětně editovat. Aplikace je navržena takovým způsobem, aby i uživatel neznalý LaTeXové syntaxe, byl schopen tvořit složité matematické výrazy. Přitom je možné plynule přecházet mezi interaktivním zadáváním matematického výrazu a editací LaTeXového kódu.

Klíčová slova: Rovnice, Editor, LaTeX, Android

Title: Equation editor for Android

Author: Jiří Marek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Petr Kučera, Ph.D.

Abstract: The goal of this project is to implement an application for the mobile operating system Android, which allows user to interactively create mathematical expressions and typeset them in LaTeX. The application is also capable of saving and sharing its output in various file formats. The application contains a general editor and compiler of LaTeX documents. It is possible to use the equation editor to create a formula, place it into a full LaTeX document and to edit it later in the equation editor again. The application is designed in such a way that even a user, who doesn't know LaTeX, is capable of creating complex mathematical expressions. It is also possible to seamlessly switch between an interactive type of input and direct editing of a LaTeX code.

Keywords: Equation, Editor, LaTeX, Android

Obsah

[1] User Interface

1. Úvod

Existuje čím dál více různých mobilních zařízení, ať již jde o telefony, tablety, phablety, ultrabooky či jiné. Tato zařízení jsou povětšinou ovládána dotykově a jejich výkon je často srovnatelný i se stolními počítači. I přesto, že většina uživatelů je využívá především pro zábavu, jejich původní účel je především usnadnit práci na cestách.

Většina těchto zařízení má odlišný operační systém i architekturu procesorů a dalších hardwarových prvků než počítače používající Windows (ne RT edice), Linux (nemobilní distribuce) či Mac OS. Tyto změny především reflektují potřebu nižšího energetického odběru. Mezi další vlastnosti typické pro mobilní zařízení patří dotykové ovládání a větší omezení kladená ze strany operačního systému na uživatelské programy (Tato omezení jsou zde zejména proto, aby aplikace třetích stran nenarušily chod základních funkcí jako volání, sms zprávy,...). Tyto vlastnosti omezují snadnou konverzi softwaru pro tyto zařízení.

Jeden z nejvíce používaných mobilních operačních systémů je Android. Android je otevřený systém, který může použít jakýkoliv výrobce na svém zařízení.

Jedna z aplikací, která ještě nebyla kvalitně převedena na Android, je program pro sestavování studentských či vědeckých prací. Nejpoužívanější software pro tuto činnost (hlavně v oborech jako matematika, fyzika či informatika) jsou odvozené verze sázecího programu TEX. Jedna z těchto odvozených verzí je LATEX.

LATEX umožňuje psaní krásně vysázených dokumentů. Využívá sázecí program TEX, který byl v roce 1978 vydán Donaldem Knuthem, následně vydal i knihu The TEXbook [6], která obsahuje popis jazyka. LATEX i TEX jsou oba značkovací jazyky (markup languages), to znamená, že obsahují struktury sloužící k obohacení textu o dodatečné informace, která typicky značí způsob vykreslení textu. V LATEXu se tyto struktury nazývají makra (macros) nebo příkazy (commands). Makra mají tvar `\macro[nepovinné parametry]{povinný parametr}..{..}`, konkrétně třeba makro pro **tučný text** je `\textbf{tučný text}`. Existují tu i specifická makra nazvaná prostředí (environments), které předpokládají, že jejich parametr má být delší text, většinou začínají `\begin{název prostředí}` a končí `\end{název prostředí}`. Jakýkoliv uživatel si může dopsat svoje vlastní makra. Tato makra se většinou, pro distribuci, sdružují do balíků (packages). LATEX je jen balík maker pro TEX, původně napsaných Lesliem Lamportem a stále je rozšiřován. Leslie Lamport popisuje LATEX ve své knize [7]. Psaní TEXového nebo LATEXového kódu se někdy také nazývá sázení textu a kompilace se nazývá vysázení.

Pro Android zatím není k dispozici žádná spolehlivá aplikace, která je zdarma, dokáže kompilovat LATEX přímo na zařízení, podporuje automatické stahování balíků a zároveň funguje i jako editor. Přitom na nemobilních operačních systémech (Mac OS X, Linux, Windows,...), jsou takové programy běžně dostupné.

Jedna ze složitějších činností v LATEXu je sestavování matematických výrazů, obzvlášť pro lidi, kteří nepoužívají LATEX denně. Existuje totiž spousta maker pro sázení různých výrazů, různými způsoby. Pro zjednodušení této práce existují různé editory rovnic. Pro Android existují také, ale žádný z nich není dostatečně provázán s editorem LATEXu.

Moje aplikace toto mění a poskytuje Editor rovnic, který je pevně integrován v editoru LATEXu.

1.1 Cíl práce

Cílem práce je naprogramovat Editor rovnic pro operační systém Android. I přesto, že v názvu aplikace je rovnice neznamená to, že tento editor bude schopný vytvářet/editovat pouze rovnice, bude možné v něm sestavovat skoro jakýkoliv matematický výraz. V této práci budou občas tyto termíny zaměněny.

Editor rovnic by měl mít několik základních vlastností:

Kvalitně vysázený výstup - Pro výstup z Editoru rovnic bude použitý program LaTeX.

Intuitivní ovládání - Přestože se budou rovnice vykreslovat pomocí LaTeXu, bylo by dobré, aby byl editor rovnic použitelný i pro uživatele, kteří LaTeX neovládají. To znamená i těmto uživatelům umožnit, bez jakýchkoliv potíží sestavit si rovnici. Na druhou stranu je důležité, aby na rovnici byly možné i pokročilé úpravy, které samotný Editor rovnic neumožňuje. Editor by měl tedy poskytovat i přístup k LaTeXovému kódu a povolit jeho editaci uživatelem.

Náhled výsledku - Editor by měl být schopný rychle zobrazit náhled rovnice, vysázené pomocí LaTeXu, bez nutnosti otevírání externích aplikací.

Výstupní formáty - Uživatel bude chtít svoji rovnici distribuovat. To znamená, že chce mít na výběr z různých obrázkových formátů ať už rastrových (JPEG[1], PNG[2], WEBP[3]) či vektorových (PDF[4], SVG[5]).

Vkládání rovnice do LaTeXového dokumentu - Rovnice by měla jít vložit dokumentu napsaného v LaTeXu a to nejen jako obrázek, ale i jako zdrojový kód. Aby bylo možné vkládat rovnice přímo do dokumentu, je vhodné aby tento dokument bylo možné otevřít a editovat. Proto další z cílů této práce bude vytvoření editoru LaTeXu pro Android.

1.1.1 Editor LaTeXu

Kromě možnosti vkládání rovnic do dokumentů, je zde několik dalších požadavků, které jsou očekávány od Editoru LaTeXu:

Kompilace - Bez možnosti kompilovat LaTeX, by editor byl jen editor textu. Tuto schopnost bude již aplikace obsahovat, díky sázení výstupu pro Editor rovnic.

Stahování LaTeXových balíků - Mobilní zařízení většinou nemají dostatečnou kapacitu pro instalaci všech balíků LaTeXu. Proto je dobré stahovat jen ty, které uživatel potřebuje, a tento proces co nejvíce zjednodušit.

[1] Joint Photographic Experts Group
[2] Portable Network Graphics
[3] Web Photo
[4] Portable Document Format
[5] Scalable Vector Graphics

Zvýrazňování syntaxe - Editace textu se může stát nepřehlednou, proto by každý editor komplikovanějších textových struktur, měl být schopný je zpřehlednit a rozlišit funkční obsah (makra) od ostatního obsahu. Editor LaTeXu by proto měl být schopen zvýrazňovat syntaxi během editování textu.

1.1.2 Distribuce aplikace a vyhodnocení

Aplikace bude veřejně distribuována. Je potřeba najít vhodný distribuční kanál a vyhodnotit úspěšnost aplikace na základě odezvy uživatelů.

1.2 Struktura práce

V **kapitole Analýza** je popsáno, jak jsou řešeny jednotlivé problémy, které se při návrhu aplikace objevily, a jejich alternativní řešení, která byla zamítnuta. Součástí analýzy je i porovnání s jinými programy.

V **kapitole Uživatelská dokumentace** je rozebráno ovládání aplikace.

Kapitola Programátorská dokumentace se zabývá implementací včetně vztahů mezi jednotlivými moduly aplikace.

V **závěrečné kapitole** je zhodnocení práce. Shrnutí toho co bylo v práci vyřešeno, co zůstalo nedořešeno a co mohlo být vyřešeno lépe.

2. Analýza

Tato kapitola rozebírá všechny závažnější problémy, které byly řešeny při implementaci této práce. U většiny problémů jsou rozebrána alternativní řešení a důvod pro použití zvoleného řešení.

2.1 Kompilace LaTeXového kódu na Androidu

Editor rovnic má vykreslovat rovnice pomocí LaTeXu. Zároveň využití kompilování LaTeXového kódu bude i v Editoru LaTeXu, který tato aplikace obsahuje. Pro kompilaci je potřeba kompilátor a ten pro Android neexistuje, alespoň ne žádná oficiální verze. Narozdíl od operačních systémů jako Windows, Linux nebo Mac OS pro které existují distribuce jako MikTeX, TeX Live, MacTeX a další (MacTeX je ve skutečnosti vývojová větev TeX Live). Tyto programy kromě kompilování LaTeXu zajišťují i správu balíčků a další služby.

2.1.1 Možná řešení

Online kompilace

Toto řešení znamená, že kompilace LaTeXového kódu nebude probíhat přímo na zařízení Android, ale bude probíhat v „cloudu“ na nějakém vzdáleném serveru. To vyžaduje mít vlastní nebo využít nějaký stávající server pro kompilaci LaTeXového kódu. Výhody tohoto řešení jsou, že nevyžaduje příliš mnoho místa v datovém úložišti mobilního zařízení, na rychlém datovém připojení může být vytváření dokumentů rychlé. Nevýhoda je, že při nedostupnosti datového připojení či serveru aplikace nefunguje.

Offline kompilace

Distribuce TeX Live i MikTeX jsou open-source. Obě by tedy mohly jít skompilovat pro Android. Je možné je tedy využít pro kompilaci LaTeXového kódu přímo na zařízení. Výhoda tohoto řešení oproti online řešení je, že pokud máme všechny balíky, můžeme vytvářet dokumenty bez potřeby datového připojení. Hlavní nevýhoda je, že aplikace musí být schopna zpracovávat výstup z těchto distribucí a na jeho základě nabídnout uživateli automatické stažení balíků. To proto, že mít nainstalované všechny dostupné balíky najednou, by vyžadovalo spoustu místa v datovém úložišti, kterého většinou není moc na mobilním zařízení. Navíc by se mělo zajistit, aby uživatelé měli ve svém zařízení vždy aktuální verzi balíků.

2.1.2 Zvolené řešení

Nakonec bylo zvoleno offline řešení. Ke kompilaci LaTeXového kódu je využita knihovna LAHTeX, která řeší i automatické stahování nových balíků. Tato knihovna je postavena nad distribucí TeX Live, kterou autor knihovny skompiloval pro Android. Licence pro využití této knihovny je umístěna v příloze A.

2.2 Uživatelský vstup pro sestavování rovnic

Jeden ze základních cílů této práce je možnost sestavit rovnice i bez znalosti LaTeXu. To znamená, že je nutné mít v aplikaci způsob pro interaktivní zadávání rovnic.

2.2.1 Možná řešení

Zjednodušená syntaxe

Princip spočívá v tom, že pro zadávání rovnice by byl použit kód se zjednodušenou syntaxí (oproti LaTeXu), specificky navrženou pro psaní matematických výrazů. Šlo by o podobný princip, který je použitý v aplikaci Equation Editor (com.vic.equationeditor) (více o této aplikaci je v sekci 2.13.3). V tomto editoru je například tato rovnice $f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$ vyjádřená tímto zjednodušeným kódem `f'(x) = \lim↙ {h→ 0}{ f(x+h) - f(x) }/h`, zatímco v LaTeXu by byla vyjádřena takto `f'(x) = \lim \limits_{h\to 0} \frac{ f(x+h) - f(x) }{h}`. Toto řešení je možno vylepšit tím, že se zkonstruuje strukturované menu, které bude pomáhat při vkládání jednotlivých příkazů.

Výhoda tohoto řešení je jednoduchá implementace. Nevýhoda je, že si skoro vůbec nepomůžeme oproti zápisu LaTeXu. Také uživatele budeme učit syntaxi, kterou nikde jinde nebude schopný použít. Navíc toto řešení se nedá hodnotit ani jako interaktivní.

Rozpoznávání prstem psaných rovnic na displeji

Podstata tohoto řešení je využití dotykových displejů, pro které je operační systém Android navržen, takže se dá spolehnout na to, že převážná většina zařízeních, která používají tento systém, mají také dotykový displej.

Na rozdíl od rozpoznávání ručně psaných rovnic z fotografie, máme výhodu v tom, že již známe trasu a směr jakým se uživatel snaží znaky napsat. Nevíme, ale kolika tahy jeden znak napíše nebo jestli dokonce spojí více znaků do jednoho tahu, toto řešení má tedy dvě fáze.

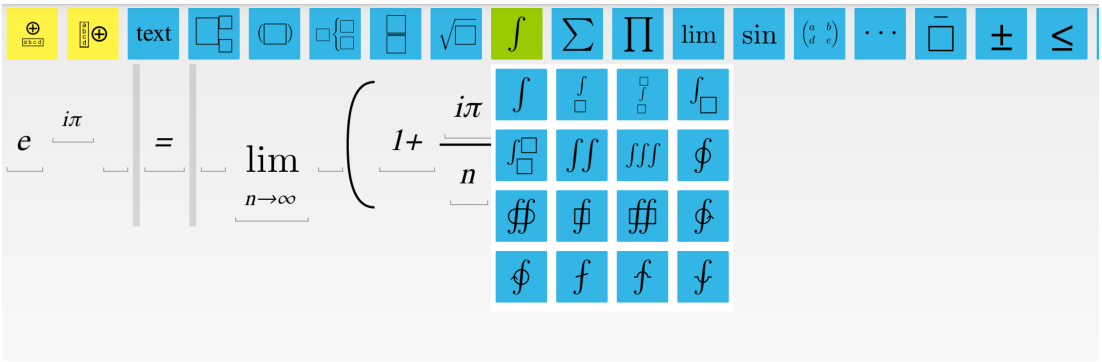
První fáze tohoto řešení je správně rozdělit nakreslený výraz do jednotlivých znaků. Druhá fáze je správně rozeznat jednotlivé znaky. Tyto fáze musí být mezi sebou propojené, abychom věděli jak a které tahy můžeme rozdělit na jednotlivé znaky. Typicky se nám zde bude nabízet více variant pro rozdělení a my musíme zvolit tu optimální. Více variant se nám bude nabízet ze dvou důvodů. První je ten, že jeden znak může obsahovat jiný znak. Druhý důvod je, že při rozpoznávání psaného textu musíme zvážit různé styly psaní uživatelů. Toto řešení je tedy značně komplikované a výsledek není jistý. Nedají se ani spolehlivě pokrýt všechny znaky, především kvůli jejich vzájemné podobnosti. Výhoda tohoto řešení je, že kdokoliv je schopný napsat daný matematický výraz rukou, by měl být schopný ho vytvořit i tímto způsobem. To znamená vysokou intuitivnost ovládání.

WYSIWYG[1] a sestavování rovnice vybíráním matematických elementů z menu

Toto je řešení, které využívá většina editorů rovnic, které jsem testoval. Uživatel jednoduše vybírá jednotlivé matematické elementy ze strukturovaného menu a jednotlivé znaky (písmena a číslice) vkládá pomocí klávesnice. Matematický výraz se poté vykresluje v „editačním módu“, to znamená, že vzhled rovnice je uzpůsobený tomu, aby se dal jednoduše editovat. Při editování na dotykovém displeji je nutné, aby všechny plochy, které mají být interaktivní, byly dostatečně veliké tak, aby se na ně uživatel byl schopný trefit prstem. To ale nebrání, tomu aby se matematický výraz vykreslený v tomto „editačním módu“, co nejvíce podobal výslednému výstupu.

2.2.2 Zvolené řešení

Bylo zvoleno řešení, které je popsané v sekci „WYSIWYG a sestavování rovnice vybíráním matematických elementů z menu“. Prostředí pro zadávání rovnic interaktivním způsobem je naznačeno na obrázku 2.1.



Obrázek 2.1: Náhled na prostředí při sestavování rovnice.

2.3 Definování matematických elementů

V sekci 2.2 je popsán způsob jakým bude uživatel vybírat jednotlivé elementy. Další důležitá část je, jak tyto elementy budou definovány. Matematické výrazy mají spousty různých symbolů a znaků, z nichž většina se chová stejně jako písmena, to znamená, že zabírají jasně daný prostor. Pak jsou zde však i takové symboly, jako třeba odmocnina nebo závorka, které by se měly přizpůsobovat svému obsahu.

Definice elementů musí popisovat tři vlastnosti, způsob vykreslení daného elementu, způsob zobrazení elementu v menu a také jak daný symbol převést do a z LaTeXu.

[1] What you see is what you get

2.3.1 Možná řešení

Rozšiřování objektového návrhu

Vytvořit kvalitní objektový návrh a ten rozšiřovat. Jednotlivé třídy by definovaly jednotlivé matematické elementy, popřípadě skupiny elementů. Jejichž jednotlivé instance by rozlišovaly jednotlivé elementy.

Externí soubor

Načítání správně naformátovaného externího souboru, který definuje jednotlivé matematické elementy.

2.3.2 Zvolené řešení

V aplikaci je použit externí soubor pro definování symbolů. Externí soubor je sice méně variabilní, ale jednodušeji se edituje a při přidání/odebrání/změně elementu není nutné znovu kompilovat celou aplikaci.

Výběr formátu souboru

Při volbě vhodného formátu souboru pro popis elementů byly uvažovány dvě možnosti. Objektový datový formát JSON[2] a značkovací jazyk XML[3]. Byl zvolen formát XML, zejména proto, že parsování XML je již implementováno v Android SDK[4] (Nejedná se o stejné XML soubory, ve kterých jsou definovány různé resources při využívání Android SDK, jde o vlastní formát). Dalším důvodem je to, že XML soubor se oproti JSON snadněji upravuje běžným textovým editorem.

Menu

Jednotlivé matematické elementy se v aplikaci vkládají pomocí menu (viz obrázek 2.1). Externí soubor, který definuje jednotlivé výrazy, také definuje strukturu tohoto menu.

Definování jednotlivých symbolů

Pro definici zobrazení elementu při zadávání rovnice jsou použity tři způsoby. Jeden je statický, druhý je dynamický a třetí je textový. Každý definovaný symbol většinou odpovídá jednomu LaTeXovému makru. Většina maker, která jsou definovaná v této aplikaci, jsou přebrána z „LaTeX Wikibook“ [13] z kapitol „Mathematics“ a „Advanced Mathematics“.

Jednotlivé způsoby se dají mezi sebou kombinovat pro dosažení požadovaného výsledku. S tím, že pokud má dané makro nějaké povinné či nepovinné parametry, je to v XML také zaznamenáno. Pro definování vzhledu symbolů v menu se využívá pouze statický a textový způsob.

[2] JavaScript Object Notation
[3] Extensible Markup Language
[4] Software Development Kit

Statický způsob - Definuje se pomocí LaTeXu. Výsledkem jsou statické obrázky. Aplikace se distribuuje s již vygenerovanými PDF soubory (pokud by v aplikaci chyběly, vygenerují se automaticky ve chvíli prvního použití / spuštění, což by ale nemělo nastat). Pro každé zařízení se při prvním spuštění z těchto PDF souborů automaticky vygenerují PNG soubory s rozlišením závisejícím na dpi[5] displeje daného přístroje. Tohle se provádí, protože v PDF jsou obrázky elementů definovány vektorově, takže z nich je vždy možné vygenerovat PNG tak, aby obrázek byl na všech displejích stejně velký a v dostatečné kvalitě. Pro samotné vykreslování není použito přímo PDF nebo jiný vektorový formát, protože je výpočetně náročné je načítat.

Dynamický způsob - Rozdíl mezi dynamickým a statickým způsobem je, že dynamické elementy se dokáží přizpůsobit svému okolí na základě jejich parametrů. Například, když se symbol odmocniny zvětšuje podle toho, jaký je v něm obsah.

Dynamický způsob se definuje pomocí syntaxe, která je podobná syntaxi pro definici „SVG Paths“ [12] elementů. Tento způsob vždy definuje nějaké křivky, které jsou definovány jako posloupnost příkazů s různým počtem parametrů, tyto parametry reprezentují reálná čísla. Parametry mohou být i složitější výrazy s proměnnými. Například definice pravé závorky vypadá takto: `M 0 2*dip Q w 2*dip w h*0.5 Q w h-2*dip 0 h-2*dip`. Kde `Q` a `M` jsou příkazy a `dip`, `w`, `h` jsou proměnné. `Q` je příkaz pro vykreslení Bézierovy křivky, startovní bod Bézierovy křivky je tam, kde skončil předchozí příkaz a dva zbylé body jsou definovány čtyřmi parametry tohoto příkazu. Druhý příkaz je `M`, který definuje posunutí aktuální pozice (bez vykreslení), tedy může definovat začátek nové křivky, přijímá dva parametry. Proměnná `dip` představuje 1 dip[6]. Proměnné `w` a `h` definují aktuální rozměry daného elementu.

Možná by šlo v některých případech využít i tzv. nine-patch obrázky, u kterých se dá definovat, která část se má při roztahování opakovat na dané ose, to ale stejně není univerzální řešení (nelze tímto způsobem řešit opakování více částí, při roztahování na jedné ose).

Textový způsob - Reprezentuje symboly, které jsou zobrazeny pomocí fontu, stejně jako třeba čísla nebo písmena z latinky. Znaky definované tímto způsobem, musejí být takto zapsány z toho důvodu, že je LaTeX nedokáže vykreslit pomocí jejich unicode kódu, ale pro každý z nich existuje nějaké specifické makro. Tento způsob vlastně definuje konverzi těchto znaků z LaTeXu do unicode a zpátky. Zmiňme například znak zpětného lomítka (\), který je v LaTeXu použit jako uvozující znak pro makra, a nelze jej tedy použít přímo. V matematickém prostředí ho lze ale zapsat pomocí makra `\backslash`.

2.4 Vykreslování rovnic v „editačním módu“

Výstup aplikace je zprostředkován pomocí LaTeXu. Pro vykreslování rovnic v „editačním módu“ (je zmíněn v sekci 2.2.1), je potřeba zajistit dvě vlastnosti. Vzorec se musí rychle překreslovat po změně a musí vypadat co nejpodobněji výstupu

[5] dots per inch

[6] Density-independent Pixels

z LaTeXu. To znamená, že když uživatel něco změní, například přidá číslo nebo přidá nový symbol, aktualizace se musí provést okamžitě. Narozdíl od výstupu z LaTeXu, kde čekáme na jeho kompilaci.

2.4.1 Jak Android vykresluje svoje UI

Hlavní prvek při sestavování uživatelského rozhraní v operačním systému Android je objekt *Activity*, který se stará o vytvoření okna (Okno si můžeme představit, stejně jako na systémech Windows či Linux, tady ve většině případech zabírá celý displej tzv. fullscreen). *Activity* obsahuje objekty *View*. Objekty *View* tvoří stromovou hierarchii uvnitř *Activity*. Každý objekt *View* má daný obdelník do kterého může vykreslovat svůj obsah. Velikost a umístění tohoto obdelníku mu určí jeho rodič ve stromové hierarchii v rámci svého prostoru. Kořenu určí velikost vykreslovacího obdelníku objekt *Activity*. Ne každý objekt *View* může mít potomky pouze ty, které jsou odvozené od třídy *ViewGroup*.

Když je od *Activity* vyžádáno vykreslení, tak předá objekt *View*, který představuje kořen stromu, Android frameworku, který se postará o vykreslení. Vykreslení poté probíhá dvoufázově. První fáze je tzv. měřící a druhá fáze je tzv. rozvrhovací. Obě fáze jsou založeny na průchodu hierarchické stuktury do hloubky systémem „post-order“ (nejdříve se provede daná operace na potomcích a až potom na jejich rodiči).

První fáze má za úkol zjistit rozměry každého *View*. Když rodič přeměřuje svého potomka může se stát, že ho bude přeměřovat více než jednou. *View* má nastaveno nějaké omezení velikosti při přeměřování a když chce přeměřit své potomky musí je také omezit. Toto omezení, ale může být různé, záleží na tom o jaký typ *View* jde a jaké má nastavené parametry pro rozměry. Typický průběh měření je takový, že nejdříve se přeměří potomek bez jakýkoliv omezení a potom na základě jeho rozměrů a rozměrů ostatních potomků téhož rodiče se určí omezení a pokud je to potřeba, dané uzly se přeměří znovu s nastaveným omezením. Při přeměřování potomka se vždy přeměří celý podstrom, který reprezentuje. To znamená, že v nejhorším případě může mít přeměřování až exponenciální složitost.

Ve druhé fázi je každý rodič zodpovědný za určení pozice svých potomků s použitím rozměrů zjištěných v první fázi. Více informací lze najít v Android Guide [11] v článku „How Android Draws Views“.

2.4.2 Možná řešení

Vykreslování pomocí Android UI

Využít stávajícího vykreslování uživatelského rozhraní v Androidu. To znamená, sestavit si stromovou strukturu pomocí objektů *View* zanořených do sebe a kde je třeba, vykreslovat jednotlivé elementy do jejich objektu *Canvas*. Více informací o vykreslování lze najít v Android Guide [11] v článku „Custom Drawing“.

Vykreslování pomocí OpenGL

OpenGL[7] je cross-platform API[8] pro vykreslování 3D (i 2D) grafiky. Jednotlivé funkce OpenGL API jsou implementovány hardwarově (Mohou být i softwarově, pokud je hardware nepodporuje) na grafické kartě, tím lze získat podstatně vyšší výkon při vykreslování grafiky. Na Androidu lze použít OpenGL ES[9], což je odnož OpenGL pro jednoúčelová (embedded) zařízení. Použití OpenGL znamená, že je nutné připravit většinu komponent, které Android poskytuje, od znovu. Jednalo by se především o implementaci textového pole a vstupů z displeje.

2.4.3 Zvolené řešení

Bylo zvoleno vykreslování pomocí Android UI. Hlavně kvůli složitosti implementace OpenGL řešení.

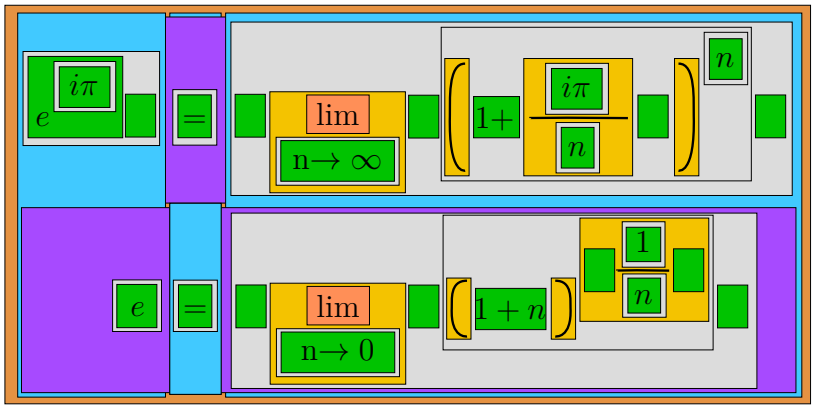
Vnitřní reprezentace rovnice pro vykreslování

Matematický výraz je v aplikaci reprezentován pomocí hierarchické struktury. Například tyto dvě zarovnané rovnice:

$$\begin{aligned} e^{i\pi} &= \lim_{n\to\infty}\left(1+\tfrac{i\pi}{n}\right)^n \\ e &= \lim_{n\to 0}(1+n)^{\frac{1}{n}} \end{aligned}$$

Reprezentuje struktura, která je znázorněna na obrázku 2.2.

[7] Open Graphics Library
[8] Application Programming Interface
[9] OpenGL for Embedded Systems



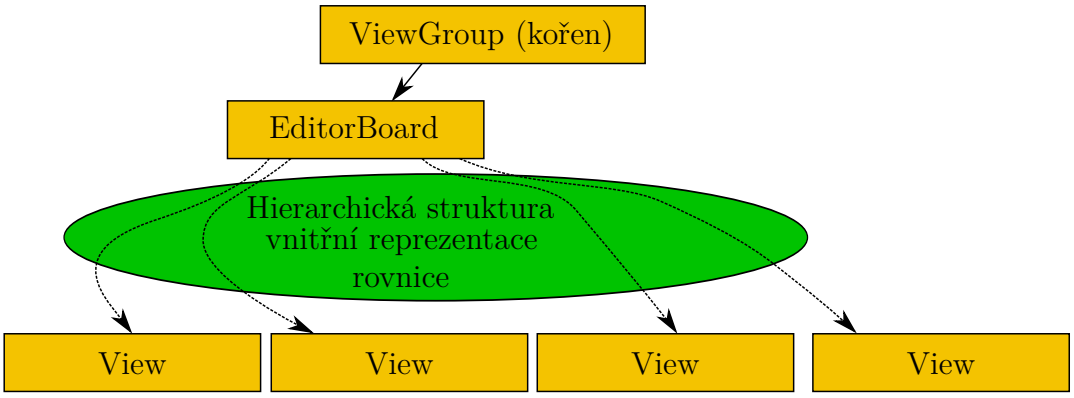
Obrázek 2.2: Na obrázku je zobrazena vnitřní reprezentace rovnice pro její vykreslování a editaci. Každý obdélník představuje jeden objekt a všechny obdélníky, které v sobě obsahuje jsou jeho potomci. Šedě na obrázku jsou zobrazeny logické řádky. Logický řádek je abstraktní pojem pro element, ve kterém se řadí vedle sebe elementy horizontálně. Zelené jsou textové objekty, to jsou objekty, do kterých se dá psát pomocí klávesnice a jsou vykreslovány pomocí fontu. Je zde i několik prázdných zelených objektů, ty jsou umísťovány vždy tam, kde by se jinak nedalo psát, ale uživatel by tam mohl chtít psát. Žluté obdélníky jsou kreslící objekty. Do těchto objektů lze kreslit pomocí dynamického způsobu zmíněného v sekci 2.3.2 a lze do nich vkládat textové objekty i ty vykreslené pomocí statického způsobu (červené obdélníky). Světle modře jsou zakresleny sloupce a fialově řádky tabulky, jejich společný předek je tabulka, která je zakreslena oranžově.

Vykreslování rovnic pomocí hierarchické struktury objektů *View*

První nápad vedl přímo k použití hierarchické stromové struktury, kterou využívá Android UI takovým způsobem, že každý objekt z vnitřní reprezentace popsané výše je děděný od třídy *View*. Ukázalo se, že takto strukturované zanořování nebude fungovat pro situace, kdy se dostaneme do větší hloubky zanoření. To proto, že v hloubce 10-15 (přesná hodnota je závislá na zařízení), dochází k přetečení zásobníku. Druhý důvod je, že při větší hloubce trvá přeměřování jednotlivých *View* dlouhou dobu, obzvláště na méně výkonných zařízeních (pravděpodobně z důvodu popsaného v sekci 2.4.1).

Vykreslování rovnic pomocí hierarchické struktury objektů *View* (Vylepšení)

Z předchozí diskuze je jasné, že je nutné zamezit zanořování objektů *View* do velké hloubky. To je možné docílit tím, že struktura, která se používá pro reprezentaci rovnice, bude zprostředkovávat obě fáze vykreslování Android UI samostatně, přičemž nebude tak výpočetně a paměťově náročná jako při zanořování objektů *View*. Objekty *View* budou všechny reálně napojeny na jeden objekt zděděný od třídy *ViewGroup*. O tyto objekty se ale bude starat (tzn. měřit a rozvrhovat) vnitřní reprezentace rovnice, respektive ty její části, které potřebují něco vykreslit. Toto řešení si lze představit tak, jak je znázorněno na obrázku 2.3.



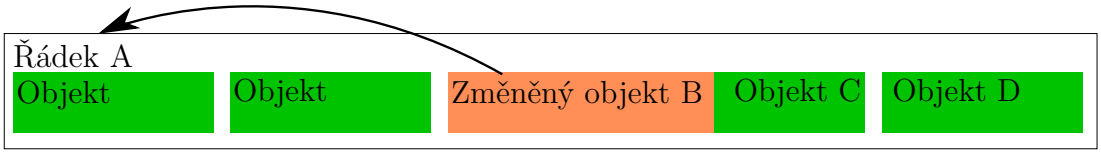
Obrázek 2.3: Na tomto obrázku je naznačeno, jak vypadá zmíněná hierarchie, když se zredukuje její hloubka. Žlutě jsou označeny potomci třídy *View*. *EditorBoard* je objekt dědící od objektu *ViewGroup*. Hierarchická struktura je zobrazena detailně na obrázku 2.2

Aktualizace struktury rovnice probíhá dvěma způsoby. Tím že aktualizuji strom hierarchie shora dolů nebo zdola nahoru (podle situace).

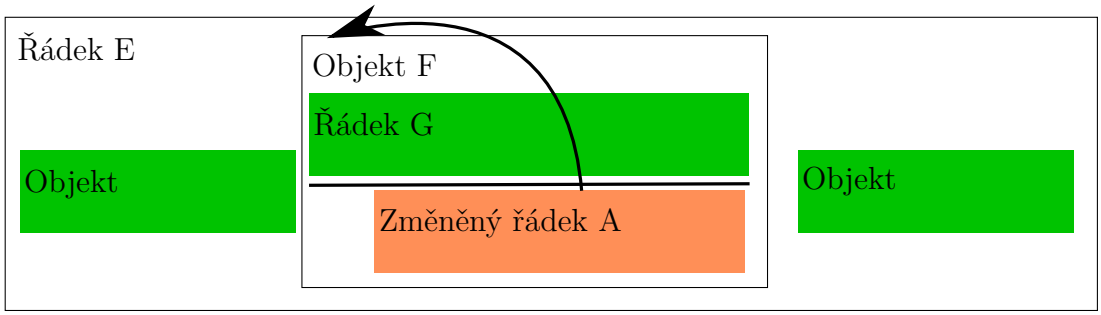
Shora dolů - Vykreslování shora dolů je použito pro situace, kdy je potřeba aktualizovat buď celou rovnici nebo určitou jasně danou část (podstrom). Výpočet probíhá podobně jako vykreslování Android UI. Také má dvě fáze měřící a rozvrhovací. V těchto fázích se aktualizuje strom od daného uzlu (aktualizuje se podstrom, který tento uzel reprezentuje). V případě, že se aktualizuje od kořene, tak to znamená, že se aktualizuje celá rovnice. Aktualizace shora dolů se používá při prvním načtení rovnice.

Zdola nahoru - Tento způsob slouží pro případ, že se změní element nebo elementy rovnice a je potřeba aktualizovat část, která dopředu není známá. Princip je takový, že uzel, který se změnil, se přeměří. Pokud se zjistí, že jeho velikost se změnila takovým způsobem, který by mohl ovlivnit jeho předka, dá mu vědět, aby se také změnil. Požadavky na přepočítání se takto předávají vzhůru. Postup končí buď v kořenu nebo v uzlu jehož rozměry ani pozici není třeba měnit. Na těchto obrázcích 2.4, 2.5, 2.6 a 2.7 je popsán příklad aktualizace elementů, při změně šířky jednoho z elementů. Řádek na obrázcích představuje logický řádek.

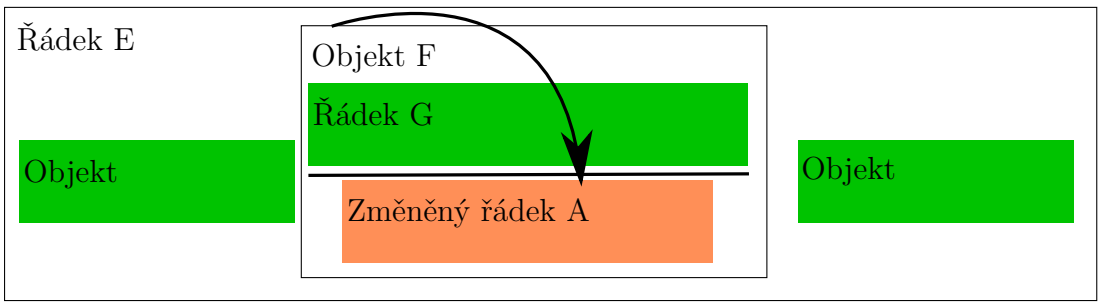
Na obrázcích je popsaný pouze jeden z případů aktualizace. V jiných případech se může zvýšit či snížit výška elementu. Nebo v případě, že využíváme i více sloupců či řádků tabulky, tak aktualizace může vypadat trochu jinak, než na obrázcích, ale princip je vždy zachován.



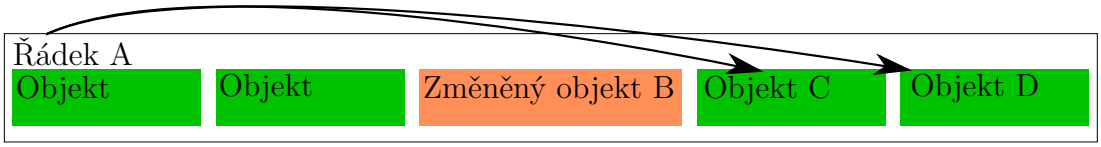
Obrázek 2.4: Objekt B se změnil a nyní by začal překrývat objekt C. Začne tedy probíhát 1. fáze (měřící). Objekt B vyšle zprávu svému rodiči (řádek A) se změnou velikosti a ten se v tu chvíli může přeměřit, protože ví, o kolik se změnila velikost objektu B.



Obrázek 2.5: Přeměřování pokračuje, řádek A vysílá zprávu o změně objektu F (což je objekt reprezentující zlomek), ten zjistí že už se sám nezvětšil, takže už nic dál nepošle (jinak by změnu ohlásil řádku E)



Obrázek 2.6: Objekt F začíná 2. fázi (rozvrhovací), s tím že určí novou polohu řádku A (zarovná ho na střed).



Obrázek 2.7: V tomto pokračování rozvrhovací fáze mohou nastat dva případy. První je rychlejší a nastává v případě, že by se nezměnila pozice řádku A. Je upravena pozice objektů C a D, protože je ovlivnila změna objektu B. V tomto příkladu, ale nastává druhý případ, protože se změnila pozice řádku A, je nutné znovu rozvrhnout všechny objekty a jejich podobjekty v tomto řádku (kromě změněného objektu B, ten si rozvrhne podobjekty sám).

2.5 Analýza LaTeXového kódu

Jednou ze základních vlastností v tomto Editoru rovnic je možnost plynule přecházet mezi interaktivním ovládáním a ručním zadávání LaTeXového kódu. K tomu je nutné, aby byl program schopný tento kód LaTeXu analyzovat a vytvářet. Vytvářet LaTeXový kód z objektově definované struktury je triviální, správně ho analyzovat po uživatelském vstupu a vytvořit z něho objektovou strukturu již není tak jednoduché.

Algoritmus pro analýzu LaTeXového kódu matematického výrazu vytváří strom (AST[10]), tento strom se poté použije pro vytvoření vnitřní reprezentace rovnice tak, jak je popsána v sekci 2.4. Strom obsahuje 4 typy objektů. První je rodič, to je takový objekt, který má potomky. Potomek může být buď textový objekt nebo příkazový objekt. Jsou dva druhy textového objektu, první reprezentuje matematický text a druhý obyčejný text vložený do matematického výrazu. Příkazový objekt je takový objekt, který může mít parametry. Ať už povinné, které jsou značené pomocí `{..}`, či nepovinné, které jsou značené pomocí `[..]`. Příkazový i textový objekt může mít přiřazen dolní index (značený `_{..}`) a horní index (značenou `^{..}`). Povinné parametry, nepovinné parametry, horní a dolní index jsou všechno objekty typu rodič, tedy mohou mít zase potomky. Posledním typem objektu jsou přepínače, ty jsou značené `\limits`, `\nolimits` a `\hline` a měly by být vždy přiřazeny k příkazovým objektům. Například pomocí rovnice:

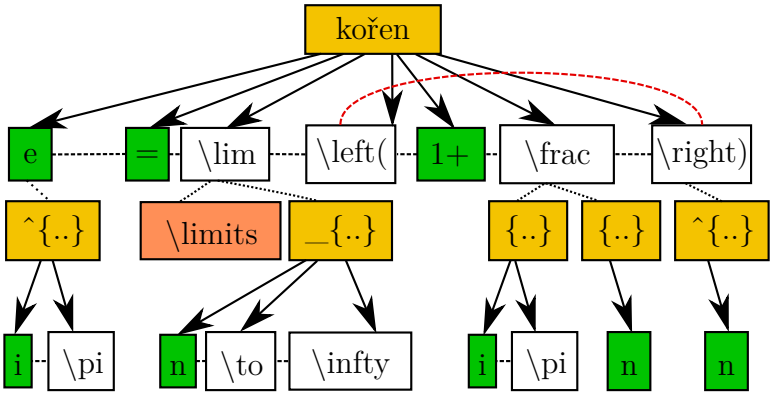
$$e^{i\pi} = \lim_{n\to\infty}\left(1+\frac{i\pi}{n}\right)^{n}$$

Zapsané kódem:

```
e^{i\pi }=\lim \limits_{n\to \infty }\left( 1+\frac{i\pi }{n}
\right) ^{n}
```

Se sestaví strom zobrazený na obrázku 2.8.

[10] Abstract syntax tree



Obrázek 2.8: Zobrazení stromu vytvořeného po analýze LaTeXového kódu. Žlutě jsou označeny objekty typu rodič. Zelené jsou textové objekty, bílé jsou příkazové objekty a červené jsou přepínače. Červená přerušovaná čára naznačuje propojení mezi dvěma příkazovými objekty. Ta znamená, že tyto objekty budou ve vnitřní reprezentaci rovnice tvořit jeden objekt.

V algoritmu při sestavování stromu platí, že v každém rodičovském objektu je definovaný jeho poslední potomek a uzavírací znak, tento znak značí, na jaký znak rodič čeká, aby se uzavřel. Kořen i přesto, že je rodič, nemá žádný uzavírací znak. Pod pojmem „aktuální objekt“ se v následujícím textu myslí poslední potomek aktuálního rodičovského objektu.

Strom se vytváří tak, že se nejdříve vytvoří prázdný rodič a nastaví se jako aktuální rodičovský objekt, tento objekt je kořen stromu. Algoritmus poté začne procházet text a v případě, že narazí na znak:

1. `\` - Začnou se prohledávat LaTeXová makra definovaná v externím souboru popsaném v sekci 2.3, tato makra jsou nyní načtená v datové struktuře *Trie* a po nalezení nejdelší shody s aktuálním textem se makro přidá do aktuálního rodičovského objektu. Dále se za tímto znakem testuje na shodu s předdefinovanými makry `\mathcolor`, `\text` a `\\` a všemi přepínači.

2. `{` nebo `[` - Pokud poslední definovaný objekt byl příkazový, tak se vytvoří nový objekt typu rodič a nastaví se jako aktuální. Tento objekt se přidá do posledního objektu jako povinný příkaz v případě, že znak je `{`, jinak jako nepovinný. Pro `{` se nastaví uzavírací znak `}` a pro `[` se nastaví uzavírací znak `]`.

3. `^` nebo `_` - Vytvoří se nový rodičovský objekt přiřazený k aktuálnímu objektu. Pokud příští znak je `{`, tak se nový rodičovský objekt nastaví jako aktuální a uzavírací znak se nastaví na `}`. Pokud je příští znak jiný, tak se do nového rodičovského objektu vloží tento jiný znak a poté se hned tento rodičovský objekt uzavře.

4. Uzavírací znak aktuálního rodičovského objektu - Přenastaví se aktuální rodičovský objekt na rodiče objektu, ke kterému je aktuální rodičovský objekt připojen.

5. `&` - Tento znak značí začátek nového sloupce. V algoritmu se s ním zachází jako s příkazem.

6. Bílý znak - Ignoruje se
7. Jiný nebílý znak - Pokud aktuální objekt není textový, tak se vytvoří nový textový objekt a přidá se do rodičovského objektu. Poté se nový znak přidá do aktuálního objektu.

Nyní se tento strom projde systémem pre-order a sestaví se z něj interní reprezentace rovnice tak, jak je popsáno v sekci 2.4.

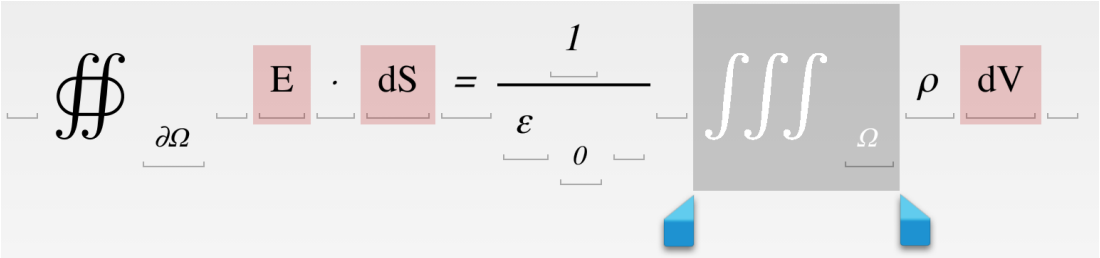
2.6 Editace matematického výrazu

V sekci 2.4 se probírá jakým způsobem se rovnice vykresluje a jakým způsobem se aktualizuje při změně. V této sekci se řeší, jakým způsobem může uživatel způsobit změnu.

2.6.1 Označování objektů

Pokud uživatel chce označit nějaký objekt, tak dlouho podrží prst na daném objektu a on se označí. Druhé možné řešení bylo použít dvojklik na daný objekt. Tato řešení jsou prakticky totožná. Obě se používají i pro označování textu, takže uživatelé jsou zvyklí obě dvě metody používat.

Algoritmus pro určení označeného objektu probíhá tímto způsobem. Známe pouze souřadnice bodu, kde má uživatel položený prst, což nám stačí pro rozpoznání objektu, který máme označit. Pro nalezení správného objektu, musíme prohledat strukturu, která reprezentuje aktuální rovnici, tato struktura je popsána v sekci 2.4. Procházení stromu se začíná od kořene a vždy se prochází všichni potomci daného uzlu a pokud některý z nich obsahuje daný bod, pokračuje hledání uvnitř daného uzlu, pokud uzel už nemá potomky, tzn. je to list, označí se. Pokud se zjistí, že daný bod není obsažen v žádném listu, tak se nic neoznačí. Toto řešení, by bylo možné zefektivnit (například binárním prohledáváním logického řádku), ale není to potřeba, protože tato operace se neprovádí často.



Obrázek 2.9: Náhled označení elementu. Modrá táhla slouží k rozšiřování výběru.

Po označení objektu, se označený objekt zvýrazní. Uživatel má poté možnost výběr rozšířit. Rozšíření výběru je možné pomocí modrých táhel, která jsou zobrazeny na obrázku 2.9. Rozšiřování lze vždy provádět jen na jednom logickém řádku. Stačí vždy proto, zjistit na jakou stranu uživatel pohnul s táhlem a na tu stranu se výběr rozšíří nebo zredukuje. Například s pravým táhlem se doprava rozšíří a doleva zredukuje. Díky tomu nám stačí testovat jen objekty na jedné straně místo celého logického řádku. Táhlo se vždy musí dostat přes půlku objektu, aby rozšířilo nebo zredukovalo výběr.

Rozšiřování výběru je omezeno pouze na jeden logický řádek, to může být v některých případech velmi omezené rozšíření (na jednom logickém řádku může být třeba jen jeden objekt). Z tohoto důvodu byla přidána možnost označení rodičovského objektu (tak jak jsou definovány v sekci 2.4). Ne každý rodičovský objekt má, ale smysl označovat. Jeden z příkladů může být, že máme označený jeden z objektů buňky matice a ten by měl nadřízený objekt buňku, ale kdyby byla označená buňka, tak to uživateli nějak nepomůže. U buňky matice nejde rozšířit výběr, mazání jedné buňky nedává smysl (to by zničilo strukturu matice). Proto se označuje jen takový rodičovský objekt, který se sám nachází v logickém řádku, pokud takový není, tak se pokračuje s prarodičovským objektem atd... V případě uvedeném výše by se označila celá matice.

2.6.2 Vstup z klávesnice

Vstup z klávesnice (softwarové nebo hardwarové) umožňuje přidávat a odebírat znaky z textových polí, která jsou vložena v rovnici. Tím se mění velikost textového pole, což způsobuje aktualizaci struktury rovnice.

2.6.3 Vstup z menu

Z menu se vkládají objekty a znaky, podle toho jak jsou definovány v sekci 2.2.1. Znaky se vkládají do stejných textových polí jako vstup z klávesnice. Objekt se vkládá tam, kde je aktuálně umístěn kurzor nebo tam, kde jsou označené objekty. Některé objekty umožňují, aby byl do nich vložen obsah. Jde například o odmocninu či různé závorky. Při vložení takového objektu, se poté aktuálně označené objekty vloží do nově vloženého objektu. V případě, kdy se ale vloží nějaký jiný objekt (například symbol integrálu), který toto nepodporuje, tak se aktuálně označené objekty přemažou tímto novým objektem.

2.6.4 Kopírování z a do schránky

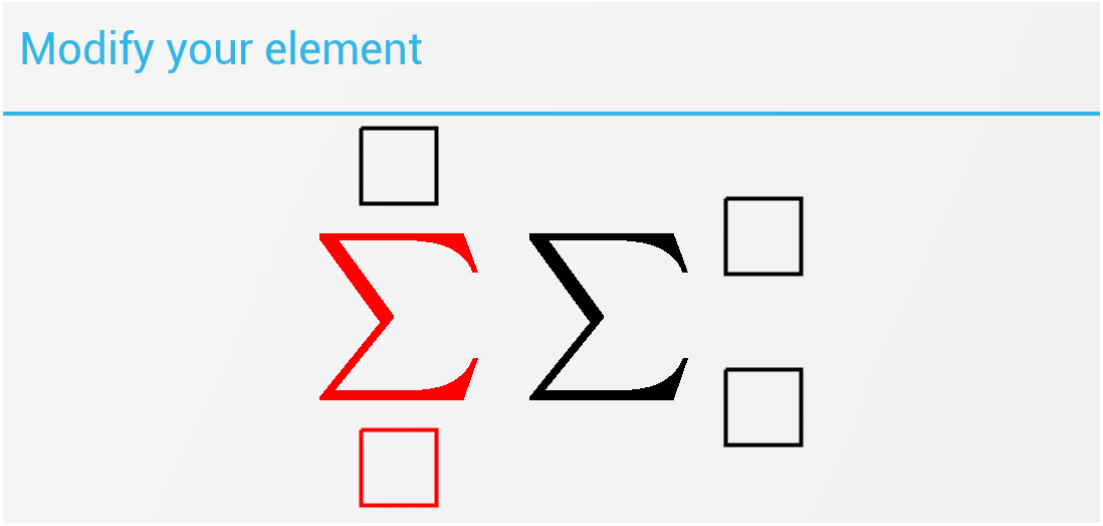
Když se elementy označí, tak je možnost je kopírovat do systémové schránky. Ve skutečnosti se do schránky zkopíruje LaTeXový kód aktuálního výběru. Při vložení se použije stejný algoritmus jako je popsaný v sekci 2.5, jen se vygenerované objekty vloží na místo dané aktuálním výběrem.

2.6.5 Změna barvy elementu

Ve chvíli, kdy se elementy označí, dostane uživatel možnost změnit barvu označených elementů. Pro výběr barvy se uživateli zobrazí paleta barev.

2.6.6 Změna limit, mocniny, indexu či nepovinných parametrů na matematickém elementu

Pro modifikaci již vloženého symbolu lze na něj kliknout a zobrazí se dialog s možnostmi změny daného symbolu, tak jak je znázorněno na obrázku 2.10.



Obrázek 2.10: Modifikace sumy. Červené položky jsou vybrané. Aktuálně je vybraná suma se spodní limitou.

2.7 Výstupní formáty

Pro vykreslení rovnic je použit program LaTeX, který sice rovnice pěkně naformátuje, ale jeho výstup ve formátu DVI[11] či výstup pdfLaTeX ve formátu PDF, nemusí být pro všechny uživatele ideální. Je tedy potřeba vyřešit konverzi z těchto formátů a dát uživateli více možností výstupu. Android nemá podporu ve svém SDK pro načtení nebo konverzi PDF ani DVI. Je zde podpora pro vytváření PDF a to až v SDK verze 19 a v budoucím SDK verze 20 (Android 5.0), by již měla být podpora pro vykreslování PDF. Zbylo tedy pouze jediné univerzální řešení, hledat externí knihovnu s požadovanou funkčností, pomineme-li možnost parsování PDF vlastním algoritmem.

2.7.1 Řešení

Hledání knihovny

Hledaná knihovna by měla podporovat konverzi z PDF do PNG a SVG a měla by mít možnost nastavit barvu pozadí dokumentu (včetně průhledné). Po vyloučení všech různých knihoven buď pro špatnou funkčnost nebo jejich licenční podmínky, mi zbyly pouze dvě: MuPDF a PlugPDF. Obě tyto knihovny jsou napsané v C.

PlugPDF - Má již vytvořené rozhraní pro Android v Javě a je možné její využití bez poplatku pro nezávislé vývojáře (Indie[12]). V této knihovně není možné nastavit vyexportovaným obrázkům průhledné pozadí a nepodporuje export do SVG. Nejedná se o open-source knihovnu, takže nelze tyto nedostatky ani odstranit.

MuPDF - Je to open-source knihovna. Nemá žádné rozhraní pro Android, ale to se dá dodělat. Nevýhoda této knihovny je její licence, která je AGPLv3 [3], jinak

[11] Device Independent File Format
[12] Independent

splňuje všechny požadavky.

Problém s licencí GNU AGPLv3

Pokud bych ve své práci použil knihovnu MuPDF uvolněnou pod licencí AGPLv3, musel bych podle pravidel této licence uvolnit i zdrojový kód celé mé aplikace pod touž licencí AGPLv3 (jedná se o tzv. copyleft licenci). Tomu na druhou stranu brání licence, s níž mi bylo umožněno použití knihovny LAHTeX, kterou používám pro práci s LaTeXem (více o této licenci v sekci 2.1 a příloze A).

Zvolená knihovna

Nakonec jsem nalezl způsob, jak se s konfliktem licence knihovny LAHTeX s licencí AGPLv3 vypořádat, a proto jsem použil knihovnu MuPDF. Problém s licencí AGPLv3 jsem vyřešil následujícím způsobem, který mi doporučili autoři LAHpdf [5]. MuPDF je společně s jejím novým rozhraním skompilovaná pomocí Android NDK[13] toolchain do binárního souboru. Při interakci s tímto binárním souborem je vždy vytvořen nový proces (pomocí *ProcessBuilder*), ve kterém běží jako nezávislý program. Tedy nevznikne žádné přímé spojení do mé aplikace a zbytek aplikace nemusí být pod AGPLv3 licencí.

Konverze

Pomocí MuPDF zkonvertuji PDF do PNG (využívá bezztrátovou kompresi). Potom načtu PNG do aplikace jako bitmapu a do ostatních rastrových formátů (JPEG, WEBP) již konvertuji pomocí Android SDK knihoven. Do SVG konvertuji přímo pomocí MuPDF.

2.8 Hledání, načítání a kompilace pluginů

Aplikace má dva hlavní moduly Editor LaTeXu a Editor rovnic. Musí se definovat propojení mezi těmito dvěma moduly. Aplikace se dělí na hlavní část a tyto dva moduly. Hlavní část se stará o načtení modulů jejich spouštění, interakci mezi nimi, kompilování LaTeXového kódu (popsáno v sekci 2.1) a konverzi PDF do dalších formátů (popsáno v sekci 2.7). Pro demonstraci oddělenosti modulů od zbytku aplikace a tedy snadnou rozšiřitelnost aplikace, je aplikace rozdělena na dvě. První hlavní aplikace obsahuje hlavní část a Editor LaTeXu. Druhá aplikace obsahuje Editor rovnic. Tato sekce se zabývá tím, jak budou mezi sebou tyto dvě aplikace komunikovat.

2.8.1 Možná řešení

Pomocí zpráv (*Intent*)

Nejprimitivnější způsob je používání objektů *Intent* jako zpráv mezi různými aplikacemi či službami. Tyto objekty slouží k nastartování služby či aplikace a potom k přijmutí výsledku při jejich ukončení. Podporují jen primitivní data nebo objekty implementující rozhraní *Parcelable* nebo *Serializable*. Toto řešení je

[13] Native Development Kit

nevýhodné pro častější komunikaci, protože služba či aplikace se musí vždy znovu nastartovat při novém požadavku, a také není vhodné pro složitější objekty.

Komunikace mezi procesy pomocí AIDL[14]

V obou aplikacích musí být definováno rozhraní pomocí AIDL. Aplikace, která reprezentuje plugin, by měla mít vytvořený objekt *Service*, reprezentující server. Potom se na ní hlavní aplikace připojí a už mohou komunikovat pomocí metod definovaných v rozhraní (v souborech typu AIDL). Nevýhodou této metody je použití jen primitivních typů a objektů (*String*, *CharSequence*) a generických kolekcí *List<T>* a *Map<K,V>* (ve skutečnosti se, ale přenáší *ArrayList* a *HashMap*) nebo objektů implementujících rozhraní *Parcelable* nebo *Serializable*. Více informací o AIDL, lze najít v Android Guide [11] v článku „Android Interface Definition Language (AIDL)“.

Sdílená paměť mezi procesy (ashmem[15], binder)

Toto je přímé využití některých vlastností jádra Androidu. Lze je využívat pouze pomocí nativního kódu v C, C++. Pomocí knihovny ashmem se dá naalokovat kus paměti, který lze sdílet mezi procesy. Po naalokování získáme fd[16], pomocí něho lze přistoupit k tomuto kusu paměti v jiném procesu. Pro předávání fd mezi procesy lze využít binder. Binder je vlastně nízkoúrovňový AIDL, které je samo pomocí binderu implementováno. Toto řešení má tedy podobné možnosti jako to pomocí AIDL, je uvedeno jen pro úplnost.

Spouštění bytecodu cizí aplikace

Aplikace pro Android je doporučeno psát v Javě, jednotlivé třídy se poté kompilují do bytecode, které se potom „zadexují“ do DEX[17] souborů a ty jsou potom interpretované pomocí DVM[18] (obdoba JVM[19]), jednotlivé třídy se kompilují do strojového kódu za běhu aplikace tzv. JIT[20] kompilátorem. Google testuje nový volitelný způsob běhu Android aplikací na zařízeních se systémem Android 4.x (od Androidu 5.x bude tento způsob výchozí), který využívá kompilátor AOT[21] to znamená, že při instalaci aplikace se DEX soubor skompiluje (pomocí nástroje dex2oat) a vytvoří se z něj OAT soubor, který je již vytvořený specificky pro dané zařízení. Poté aplikace běží pomocí ART[22] místo DVM. Více informací o ART a DVM lze najít v „The Android Source Code“ [9] v článku „Introducing ART“.

Android používá pro distribuci aplikací balíky APK[23] (ve skutečnosti je to .zip archiv), které jsou dostupné i po nainstalování aplikace. V tomto balíku se nachází mimo jiné i soubor „classes.dex“.

[14] Android Interface Definition Language
[15] Android shared memory
[16] File descriptor
[17] Dalvik EXecutable
[18] Dalvik Virtual Machine
[19] Java Virtual Machine
[20] Just-in-time
[21] Ahead-of-time
[22] Android Runtime
[23] Android application package file

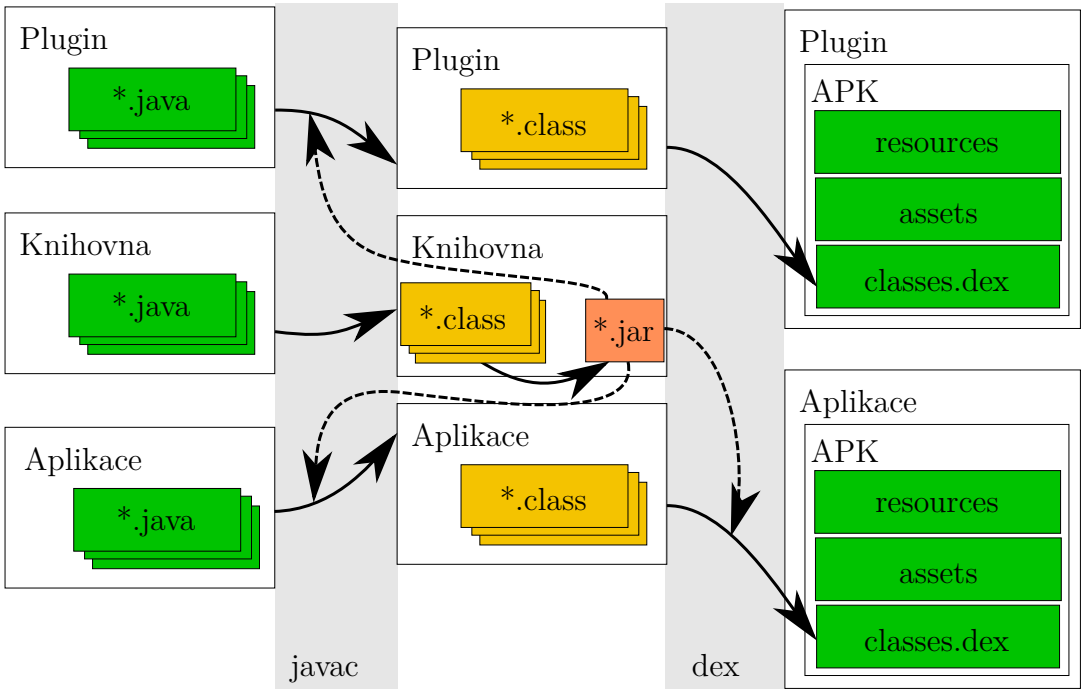
Do již spuštěné aplikace je možné načíst zmíněný DEX soubor z jiné aplikace. Z tohoto souboru pak lze načíst, jednotlivé třídy. Jen je nutné zajistit, aby některá z načtených tříd používala stejné rozhraní, jako je použito v aplikaci, aby bylo možné tu třídu ovládat.

Toto řešení lze používat i když aplikace běží pomocí ART.

2.8.2 Zvolené řešení

Zvolil jsem řešení popsané v sekci „Spouštění bytecodu cizí aplikace“. Pro správnou identifikaci všech nainstalovaných pluginů je zneužit *BroadcastReceiver*. Zneužit proto, že jeho využití je většinou trochu jiné. Je to služba, kterou poskytuje operační systém Android. Princip je takový, že všechny aplikace, které mají zaregistrovaný *BroadcastReceiver* s daným identifikátorem, obdrží data vyslaná jinou aplikací s týmž identifikátorem. Je možné najít informace o všech aplikacích, které mají zaregistrovaný daný *BroadcastReceiver* i přesto, že přes něj nedokážou přijmout žádná data. Mimo jiné je možno z těchto informací zjistit umístění APK souboru dané aplikace a plné jméno třídy, ke které je zaregistrovaný daný *BroadcastReceiver*. V této aplikace je to plné jméno třídy, která implementuje rozhraní pluginu.

Pro načtení této třídy, je třeba vytvořit *ClassLoader* z DEX souboru s jehož pomocí je možné danou třídu načíst. Problém je v tom, že daný *ClassLoader* znovu načte rozhraní, které je již v hlavní aplikaci přítomno a pomocí kterého se přistupuje k objektu v pluginu. V tom případě budou tato dvě rozhraní považována za různá, ačkoli ve skutečnosti jsou totožná. Pro překonání tohoto problému je nutné z načítaného pluginu odstranit toto rozhraní a poté přinutit daný *ClassLoader*, aby použil rozhraní umístěné v hlavní aplikaci. Je tedy nutné upravit proces kompilace a balení pluginu. Tento proces je znázorněn na obrázku 2.11.



Obrázek 2.11: Znázornění průběhu kompilace pluginu a hlavní aplikace. Při kompilaci pluginu se nejdříve skompiluje knihovna, která obsahuje rozhraní pro komunikaci mezi pluginem a aplikací, a skompilované třídy se zabalí do JAR[24] souboru. Tento JAR se poté využije pro kompilaci tříd pluginu, ale při zabalení tříd do DEX souboru se třídy z JAR souboru knihovny už nepřidají. Kompilace aplikace probíhá podobně, až na to, že třídy z knihovny se přidají do koncového DEX souboru.

2.9 Zvýrazňování syntaxe LaTeXu

V obou modulech aplikace v Editoru LaTeXu i v Editoru rovnic se přímo edituje LaTeXový kód. Editace většího množství textu se může stát velmi nepřehlednou. Proto je potřeba připravit systém zvýrazňování syntaxe. V sekci 2.5 se již jednou probírala analýza LaTeXu. Analýza LaTeXu popsaná v této sekci se liší ve dvou ohledech. Tato implementace nemusí rozumět zanořování jednotlivých příkazů do sebe, na druhou stranu však musí být univerzální, to znamená, že musí rozeznávat i příkazy, které nezná.

2.9.1 Zvýrazňování textu v Androidu

Každému objektu, který implementuje rozhraní *Spannable*, je možno nastavit zvýraznění. Dva hlavní zástupci objektů, které implementují toto rozhraní, jsou *SpannableStringBuilder* a *SpannableString*. Rozdíl mezi nimi je ten, že pro *SpannableString* se jednou nastaví text a už nejde měnit, zatímco *SpannableStringBuilder* změny textu podporuje (implementuje rozhraní *Editable* a *Appendable*).

Android má dva hlavní druhy textového pole: *TextView* a jeho potomka *EditText*. Pro tuto aplikaci je důležitý *EditText*, protože to je pole, které může uživatel měnit, zatímco *TextView* je pro statický text.

EditText si uchovává instanci *SpannableStringBuilder*, od které si zjišťuje, jestli je nastaveno označení na určitém úseku textu. Sledování tohoto objektu probíhá neustále na právě viditelném textu (Z mého pozorování jsem zjistil, že toto sledování vždy probíhá po řádcích.). Pokud jsou v tomto úseku nějaká zvýraznění, tak je dostane *EditText* zpátky jako pole objektů, které reprezentují tato označení.

Označování lze nastavovat pouze po jednom. Lze nastavit počátek, konec a vlastnosti označení (barva textu, barva pozadí,...). Tyto parametry lze měnit i později, to znamená i potom, co jsme předali naše označení do objektu, který implementuje *Spannable*.

2.9.2 Naivní implementace

První testovaný způsob implementace byl vyzkoušen pouze pro testovací účely. Analýza textu je v tomto pokusu triviální. Analyzovaný úsek se prochází a když narazí na znak:

1. `\` Zapamatuje se začátek označení a pokračuje se v procházení, do té doby dokud se nenarazí na nějaký neabecední znak, v tu chvíli nastane konec označení. (`\\[a-zA-Z]*`)

2. `%` Zapamatuje se začátek označení a pokračuje se na konec řádku (nebo textu) a tam je konec označení. (`%[^\n]*`)

3. `{`, `}`, `$`, `#`, `^`, `_`, `&`, `[` nebo `]` Označí se začátek a příští znak jako konec.

Po nalezení konce označení, se vždy úsek textu přidá do objektu *SpannableStringBuilder*, jako označení. Při změně textu se smažou všechna označení a proběhne celá analýza znovu. Toto řešení se při změnách v delších textech stává velmi pomalým. A čím je text delší, tím je to horší.

Malé vylepšení tohoto řešení je analýzu odložit (v řádu sekund) a provést ji v jiném vlákně a až za určitý čas, když uživatel přestane psát. Tím se vyřeší to nejhorší zasekávání, ale při delších textech se i přesto čekání na označení textu může stát neúnosné.

2.9.3 Lepší implementace

Efektivnější než pokaždé provést analýzu na celém textu, je analyzovat pouze změněnou část. To už ale není tak jednoduché. Dejme tomu, že máme text `\sectin{name}` a uživatel se rozhodne přidat `o` a vznikne nám `\section{name}`. Kdybychom aktualizovali text tam, kde jsme udělali změnu, tak nám algoritmus neoznačí korektně příkaz `\section`, ale místo toho označí jen `\sectio`, protože si bude pamatovat předešlé označení, které začínalo na pozici 0 a mělo 7 znaků.

Místo toho musíme správně aktualizovat i okolí změny. Zde máme dvě možnosti, buď najít začátek okolí, tak že najdeme začátek označení, do kterého zasahujeme (v případě, že jich je víc, tak to, které je nejdál). Konec okolí najdeme opačně jako konec označení do kterého zasahujeme, k němuž přičteme délku změny textu (může být i záporná, v případě mazání). Nebo použít jednodušší řešení a vždy aktualizovat celé řádky, protože víme, že žádné označení nezasahuje na více než jeden řádek. V aplikaci je použito řešení, které vždy aktualizuje celé řádky.

Za předpokladu, že se již na textu provedla prvotní kompletní analýza, tak algoritmus pro aktualizaci zvýraznění vypadá takto:

1. Zjistí se, kde je začátek řádku, na který zasahuje začátek změny a také se zjistí, kde je konec řádku, na který zasahuje konec změny.

2. Smažou se všechna zvýraznění, které jsou mezi začátkem a koncem řádků zmíněných v bodě 1.

3. Zanalyzuje se znovu úsek textu, na kterém bylo smazáno zvýraznění v bodě 2.

4. Přidá se zvýraznění nalezené při analýze v bodě 3.

Toto řešení je již efektivní, ale stále není úplně ideální. Zvlášť proto, že implementace mazání a přidávání zvýrazňování v Androidu není optimální, a přitom se maže a přidává poměrně často. Pro každé odebrání zvýraznění je složitost O(n), kde n je počet zvýraznění (prochází se pole se všemi zvýrazněnými a vymaže se poté, co se najde to požadované). Přidání typicky trvá O(1) (může trvat déle v případě, že je nutno rozšířit pole pozic), zvýraznění se přidá na konec pole. Tyto informace jsou ze zdrojových kódů, které jsou přiloženy v Android SDK verze 19.

Problém je tedy s mazáním. Řešení tohoto problému je použití vlastní datové struktury. V této datové struktuře je důležité, aby bylo možno rychle přidávat prvky na stejné místo (uživatel typicky píše déle na stejné místo, přechod na jiné místo trvá déle). To znamená, když se přemazává stejný řádek, musí to být rychlé.

Pro předělání této implementace je potřeba si vytvořit vlastní třídu, která implementuje *Spannable*. V aplikaci je použitá třída, která dědí od *SpannableStringBuilder*. Je v ní použitá nová struktura pro ukládání označení. Tato struktura je setříděné pole pozic (počátků označení), které je rozděleno na dva segmenty. První segment má v jednotlivých prvcích uložené skutečné pozice a v druhém segmentu jsou uložené prvky s posunutou pozicí. To umožňuje vkládání na konec prvního segmentu a na začátek druhého bez toho, aby bylo nutné měnit pozice všech prvků v druhém segmentu. Změní se pouze hodnota posunutí.

Při vytvoření datové struktury první segment začíná na nule a druhý na konci pole. Udržuje se pozice konce prvního segmentu, začátku druhého a hodnota posunutí.

Nápad pro tuto strukturu jsem dostal na základě podobné struktury použité v projektu TeXPert z doby, kdy byl open-source.

Hledání správné pozice

Poté, co algoritmus dostane od systému upozornění, že se text změnil, obdrží pozici začátku a konce změny. Je potřeba zjistit, na kterém řádku se začátek a konec nachází.

Proto kromě pozic označení, se v témže typu struktury udržují i pozice řádků a aktualizují se zároveň s pozicemi označení. Pro rychlé nalezení správných indexů mohu využít toho, že je struktura setříděná. Nejprve zjistím v jakém segmentu se pozice hledaného řádku nachází. A potom daný segment prohledám binárně. Tím se dostaneme na logaritmickou složitost.

Mazání

Mazání probíhá vždy v nějakém rozsahu. Jsou známy indexy počátku a konce tohoto rozsahu. Mazání pak může probíhat třemi různými způsoby.

1. Když každý počátek i konec indexu mazání je v jiném segmentu, tak v tomto případě je mazání nejjednodušší a stačí posunout počátek druhého segmentu a konec prvního segmentu, změnit velikost a upravit hodnotu posunutí. Tento případ má složitost O(1).

2. Když je celý rozsah mazání v prvním segmentu, musí se vše za tímto rozsahem přesunout do druhého segmentu. Složitost je O(n), kde n je počet označení.

3. Když je celý rozsah mazání v druhém segmentu, musí se vše, co je před tímto rozsahem, přesunout do prvního segmentu. Složitost je O(n).

Dá se očekávat, že bude nejčastěji nastávat první případ. Hlavně proto, že uživatel bude psát na jedno místo. Po jakémkoliv mazání nastává, že konec prvního segmentu je na stejném indexu, jako byl počátek rozsahu mazání. To je důležité pro přidávání nových pozic.

Přidávání

Přidávání se provádí pouze do jednoho místa, a to do místa, kde končí první segment, což znamená, že přidávání je výhodné provádět právě po mazání. Šlo by to i jinam, ale to není potřeba pro tento případ, protože přidávání nastává vždy po mazání.

Přidávání je jednoduché, stačí dosadit hodnotu na správný index a posunout konec prvního segmentu o jedna. Pokud je pole objektů s pozicemi plné, musí se zvětšit, to ale nastává jen málokdy.

2.9.4 Označování bloků textu

Do teď jsme řešili jen označování jednotlivých maker a znaků, které nepřesahují jeden řádek. Ale v LaTeXu jsou i prostředí, která typicky přesahují několik řádků a navíc mohou být v sobě různě zanořená. To znamená, že potřebují znát okolí změny a potřebuji ho znát až na začátek celého textu.

Dejme tomu, že máme text na jednom řádku: `text $rovnice$ text`. Potom při aktualizaci tohoto textu bychom označili text `rovnice`. Mohlo by se však stát, že nějaký předchozí řádek obsahuje znak `$` a nějaký následující řádek tento znak obsahuje také. To by znamenalo, že text `rovnice` by označený být neměl, což bychom nezjistili, kdybychom procházeli jen řádky obsahující změnu.

Abychom se vyhnuli procházení celého textu, tak se musí ke každému objektu, který mění prostředí (začíná nebo ukončuje), přidat kontext. To znamená, že se k němu přidá zásobník ve kterém jsou všechna prostředí, ve kterých je právě zanořený. Kromě toho jsou hranice prostředí udržované zvlášť, v témže typu struktury jako jsou pozice označení. To umožňuje rychle zkontrolovat předchozí změnu prostředí a zjistit jak hluboko je následující blok zanořený. Také díky tomu je možné rychle aktualizovat všechny následující změny prostředí, pokud je to potřeba, protože když dojde na nějakém řádku ke změně prostředí, tak to může ovlivnit i ty následující.

2.9.5 Vylepšená analýza textu

Později bylo potřeba rozeznávat více různých příkazů, znaků a slov v textu. Proto místo jednoduchého procházení textu a porovnávání těchto slov, byl implementován algoritmus Aho-Corasickové pro prohledávání textu.

2.10 Vložení rovnic do LaTeXového dokumentu

V této sekci se věnujeme tomu, jak mezi sebou propojit Editor LaTeXu a Editor rovnic pro vkládání nových a editaci stávajících rovnic.

2.10.1 Rozpoznání matematického prostředí a jeho otevření

Jak je popsáno v sekci 2.9.4, Editor LaTeXu je schopný rozlišit prostředí. Kromě toho je schopný načíst druhy prostředí ze všech pluginů, které aplikace má načtené. Například Editor rovnic definuje několik prostředí.

1. Vykreslovány na řádku s ostatním text - `\(..\)`, `$..$` a `\begin{math}..\end{math}`
2. Vykreslovány na zvláštním řádku - `\[..\]`, `$$..$$`, `\begin{displaymath}..\end{displaymath}` a `\begin{equation*}..\end{equation*}`
3. Vykreslovány na zvláštním číslovaném řádku - `\begin{equation}..\end{equation}`

Všechna tato prostředí načte Editor LaTeXu, začne je zvýrazňovat a označí je jako klikatelné.

Pro editaci prostředí, ale zvýraznění textu nestačí. Musíme zajistit, aby klikatelné bloky textu registrovaly správně doteky uživatele. To se řeší pomocí odchytávání všech doteků na textovém poli. Z každého takového doteku lze získat jeho souřadnice. Poté se využije metod objektu *EditText*, které podle souřadnic doteku získají pozici v textu. Je-li známa pozice v textu, stačí jen binárním prohledáváním nalézt příslušné označení (jejich struktura je popsána v sekci 2.9.3). Pokud je nalezené klikatelné prostředí, tak uživatel dostane nabídku pro otevření Editoru rovnic pro editaci výrazu. Po uzavření Editoru rovnic, pokud si to uživatel bude přát, se nový obsah vloží do otevřeného prostředí.

2.10.2 Rozpoznání příkazů \input a \include

Editor automaticky zvýrazňuje všechny makra `\input{..}` a `\include{..}`. Poté detekuje dlouhé kliknutí na tato makra podobným způsobem jako v sekci 2.10.1. Po kliknutí se pokusí otevřít soubor, na který tato makra odkazují a pokud existuje, tak podle jeho koncovky pro něj otevře správný modul.

2.11 Kompatibilita

Android je otevřený systém, který může jakýkoliv výrobce používat na svých zařízeních. Na jednu stranu je to skvělé, protože uživatel si může vybírat z většího počtu přístrojů, na druhou stranu každý výrobce si musí připravit specifické ovladače a další úpravy systému tak, aby systém na jeho přístrojích dobře běžel. Ne každý výrobce chce aktualizovat svoje staré přístroje na nejnovější verzi operačního systému. (Obzvlášť když už z toho nebude mít žádné profit a stálo by ho to znovu spoustu práce.) To způsobuje, že je stále velký počet přístrojů, které používají starší verze operačního systému Android. Starší verze systému používají starší verze SDK. Starší verze SDK nepodporují všechny funkce těch novějších a na druhou stranu novější se zbavují některých překonaných funkcí těch starších. S tím vším musí vývojář pro systém Android počítat, když se snaží aby jeho aplikace byla kompatibilní jak s novými, tak se staršími přístroji.

Google se snaží tyto problémy řešit a usnadňovat výrobcům aktualizace systému. Například jeden z posledních projektů, který toto řeší se jmenuje Android Silver, což by měl být standard pro výrobce přístrojů. Když se tohoto standardu budou výrobci držet, tak jim Google pomůže s aktualizacemi systému, či dokonce je bude řešit sám.

Další potíže mohou vzniknout při použití rozdílných architektur procesoru. Android nyní podporuje procesory typu: ARM[25], x86 a MIPS[26]. Do budoucna by měl podporovat i 64-bitové procesory. Android řeší tento problém použitím JIT kompilátoru (stejně jako třeba .NET), popřípadě AOT kompilátoru, jak je popsáno v sekci 2.8.1. To ale nepomůže v případě, když se používá C nebo C++ kód skompilovaný pomocí Android NDK.

2.11.1 Jednotný vzhled aplikace

Nové verze Android SDK nepřináší pouze nové funkce a komponenty. Také přináší nové styly (témata). Styl je kolekce vlastností, které definují vzhled a formát pro objekty *View* a okno aplikace. Styl může definovat vlastnosti jako je výška, rámec (vzhled ohraničení), barva textu, velikost textu a mnoho dalšího pro různé typy objektů. Styly jsou definovány pomocí XML souborů. Téma je styl aplikovaný na celý objekt *Activity* nebo na celou aplikaci. Více o tématech a stylech lze najít v Android Guide [11] v článku „Styles and Themes“.

Android poprvé zavedl téma Holo ve verzi 3.0, od té doby ho v každé verzi vylepšoval až do současné verze 4.4.4. Holo je první sjednocené a vyspělé téma, které vydrželo v Androidu po více verzí. Nižší verze Androidu než 3.0 měly více roztříštěná témata.

Jedna z klíčových nových UI komponent, která byla představena společně s tématem Holo, je Action Bar. Action Bar je prvek UI, který představuje horní lištu aplikace, která může obsahovat akční tlačítka (včetně menu) a může také poskytovat ovládání navigačních prvků UI (jako třeba vyhledávání nebo přepínání záložek). Více o Action Baru lze najít v Android Guide [11] v článku „Action Bar“.

Action Bar je jeden ze základních prvků také této aplikace. Když začal vývoj této aplikace (léto 2013), neexistovala oficiální metoda, jak zprovoznit Action

[25] Advanced RISC Machine

[26] Microprocessor without Interlocked Pipeline Stages

Bar na zařízeních se systémem starším než Android 3.0. Pro získání Action Baru byla dostupná neoficiální knihovna ActionBarSherlock. Tato knihovna byla použita v prvních verzích této aplikace. Tato knihovna, ale obsahovala téma Holo jen pro zmíněný Action Bar, zbytek aplikace byl ve výchozím tématu podle verze systému. Později Google vydal knihovnu pro zajištění kompatibility, která nahradila vše, co dělá knihovna ActionBarSherlock. Tuto knihovnu kompatibility začala využívat neoficiální knihovna HoloEverywhere, která dokáže přidat téma Holo do všech objektů aplikace Androidu verze 2.1 a vyšší. HoloEverywhere společně s knihovnou kompatibility od Googlu, tedy více než nahradily ActionSherlockBar v aktuální verzi aplikace.

Toto řešení není finální, protože vzhled Androidu se neustále vyvíjí a s ním i oficiální doporučení pro vzhled aplikací. Na konferenci I/O 2014 Google oznámil novou verzi (5.0) systému Android, která bude obsahovat nové téma (Material). Toto téma vychází z tématu Holo, ale i přesto bude tato aplikace vypadat zastarale a bude nutné ji aktualizovat.

2.11.2 Nativní kód

V této aplikaci jsou využity dva nativní programy (skompilované pomocí Android NDK), první je TeX Live (více v sekci 2.1), druhý je program, který využívá knihovnu MuPDF (více v sekci 2.7).

Android NDK oficiálně podporuje kompilaci pro čtyři CPU architektury: ARM, ARMv7, x86 a MIPS. Přičemž skompilované programy pro ARM podporují i všechny novější 32-bitové ARM procesory (včetně ARMv7), naopak to nefunguje.

Kompilace distribuce TeX Live není moje práce, binární soubory pro tento program byly převzaty společně s knihovnou LAHTeX, tak jak je popsáno v sekci 2.1. Tento program je skompilovaný pro architekturu ARM. Program založený na knihovně MuPDF, by šel skompilovat i pro ostatní architektury, ale nemělo by to moc velký smysl, pokud by nefungovala kompilace LaTeXového kódu pro tyto architektury.

To znamená, že tato aplikace běží pouze na zařízeních se systémem Android, která používají ARM architekturu.

2.11.3 Různé dpi a rozdílné rozměry displeje

U mobilních aplikací je důležité, aby jednotlivé ovládací prvky byly, pokud je to možné, stejně velké napříč všemi přístroji. Při dotykovém ovládání musí být totiž uživatel schopný se správně trefit na tlačítko a nesmí se stát, aby v případě, kdy má displej jinou hustotu pixelů, bylo tlačítko tak malé, že se na něj nedá kliknout, nebo naopak zbytečně velké. Android tento problém řeší zavedením jednotky dip. Další informace o hustotě displejů lze najít v „Android Guide“ [11] v článku „Supporting Multiple Screens“.

Android umožňuje zvolit různé rozvržení prvků (layout) podle velikosti displeje. V aplikaci je toho využito v několika případech (např. při šířce alespoň 900 dip displeje zařízení se hlavní menu objeví statické a při menší šířce se objeví jako vysouvací.).

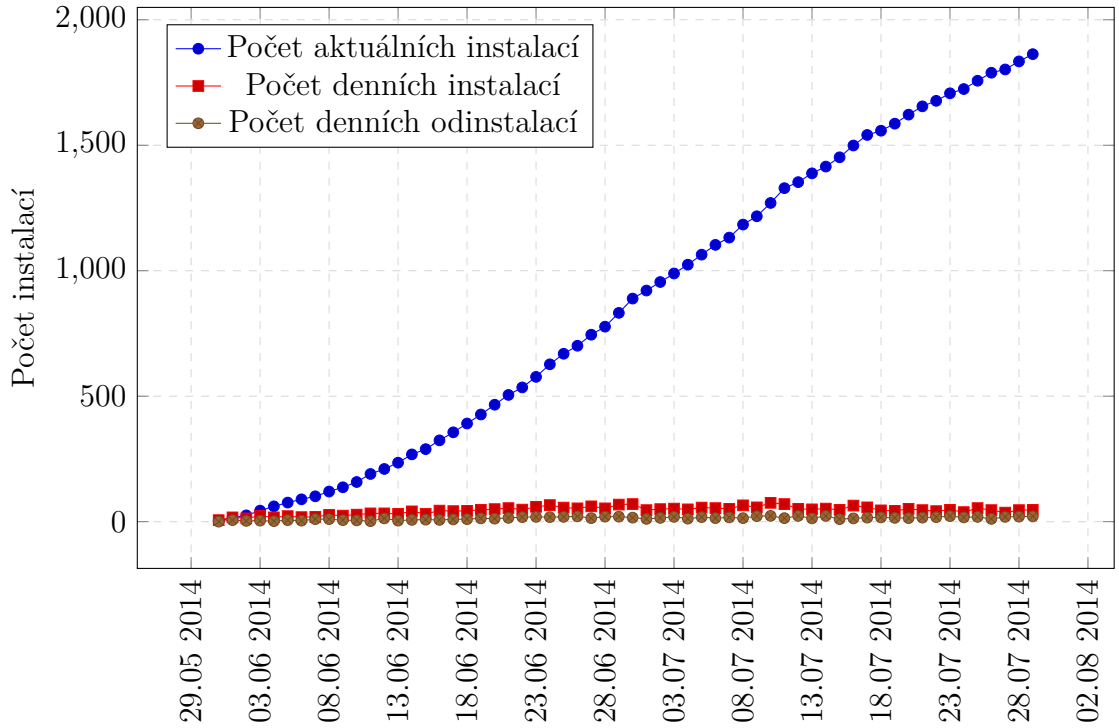
2.12 Distribuce

Aby měl softwarový projekt smysl, měl by ho někdo používat, aby ho někdo mohl používat, musí se k němu projekt dostat. Existuje několik možností, jak distribuovat Android aplikaci mezi uživatele. Nejpřímější způsob distribuce je publikovat aplikaci na Google Play, což je oficiální obchod s aplikacemi od společnosti Google. Kromě této možnosti je však k dispozici až překvapivé množství alternativních obchodů. Druhý největší obchod pro platformu Android je Amazon Appstore for Android, původně vznikl hlavně jako obchod pro operační systém Fire OS, což je větev Androidu, která je dodávána na tabletech a telefonech od Amazonu. Další výrobce, který vlastní svůj obchod s aplikacemi, je Samsung, obchod se jmenuje Samsung Apps. Narozdíl od Amazonu, na zařízeních od Samsungu lze využívat i Google Play. Je zde i několik dalších obchodů jako třeba GetJar, Slide ME nebo F-Droid. Také je tu možnost publikovat svoji aplikaci na vlastních webových stránkách nebo jakýmkoliv jiným fyzickým nebo síťovým přenosem.

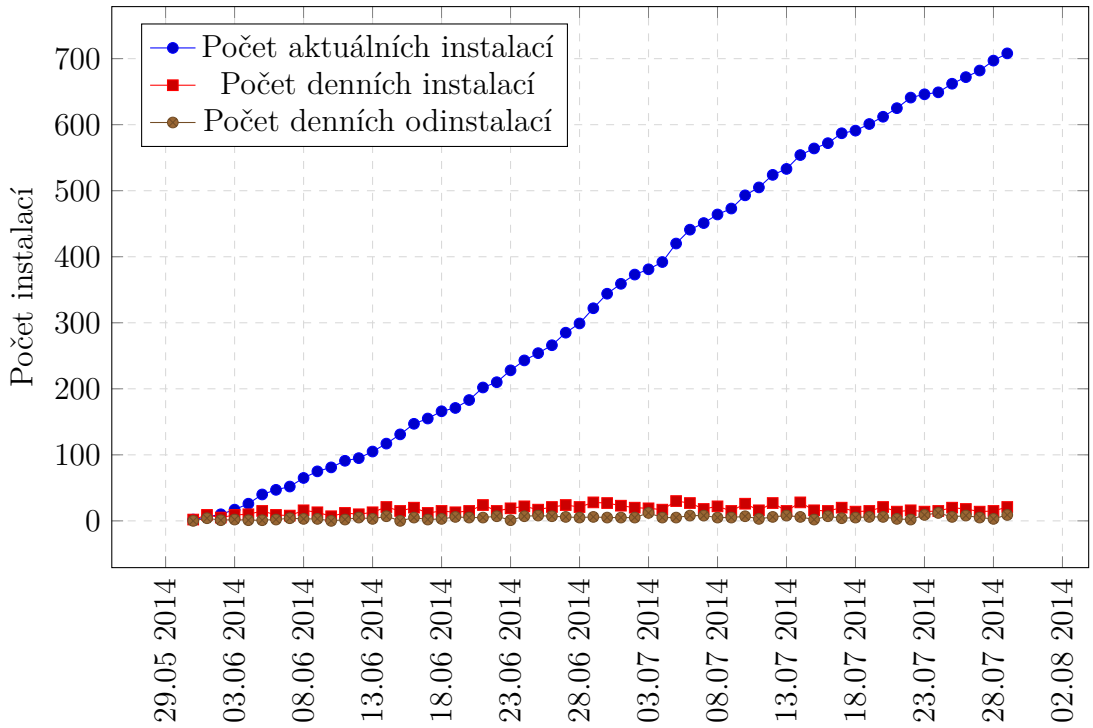
Aplikace je nakonec publikována pouze na Google Play. Byla to nejjednodušší možnost, jak získat co nejvíce uživatelů a zároveň získávat jednoduše jejich odezvu.

2.12.1 Statistiky

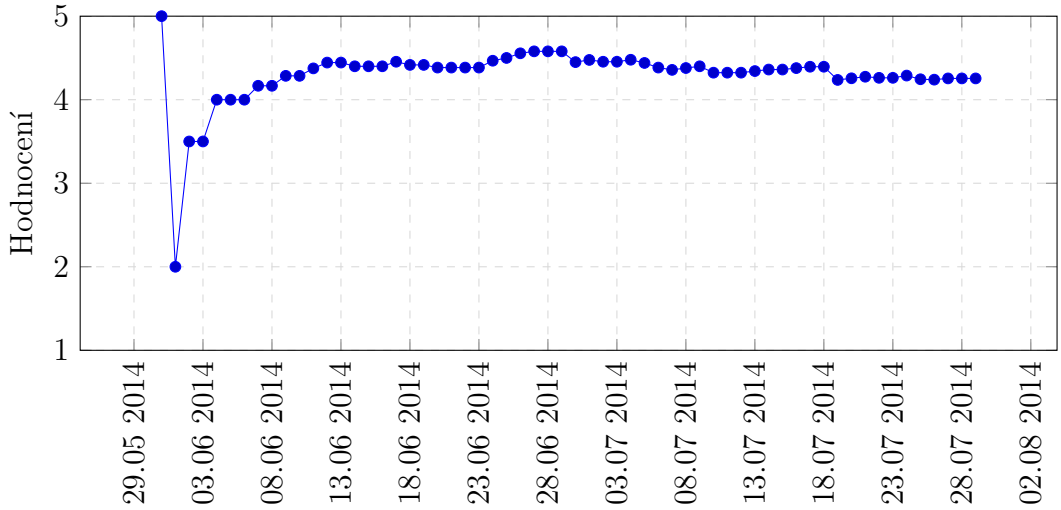
Play Store mimo jiné poskytuje vývojářům různé statistiky. Pro měření úspěšnosti aplikace, jsem vyexportoval některé z nich. Lze si je prohlédnout na obrázcích 2.12, 2.13, 2.14. Aplikace byla přijata převážně kladně, jak je vidět z hodnocení uživatelů, které se pohybuje mezi čtyřmi a pěti (přičemž pět je nejlepší).



Obrázek 2.12: Vývoj počtu instalací na unikátních zařízeních, pro editor LaTeXových dokumentů



Obrázek 2.13: Vývoj počtu instalací na unikátních zařízeních, pro Editor rovnic



Obrázek 2.14: Průměrné hodnocení od uživatelů (1 je nejhorší a 5 je nejlepší), pro Editor LaTeXu

2.13 Podobný software

Software, se zcela shodnými vlastnostmi neexistuje, ale je tu několik, které mají vždy některé z těchto vlastností.

2.13.1 TeXPortal (lah.texportal.donate) a TeXPert (lah-.texpert)

TeXPortal a TeXPert jsou Android aplikace, které mají spolu symbiotický vztah. TeXPortal je kompilátor (nejen LaTeXového kódu, ale podporuje i odvozené verze na TeXu) a TeXPert je IDE pro psaní LaTeXových dokumentů, které pro překlad využívá aplikaci TeXPortal. Obě aplikace bývaly zdarma a open-source. Z doby, kdy bývaly open-source jsem od nich převzal některé z knihoven, hlavně se jedná o knihovnu LAHTeX, která umí zpracovávat výstup z LaTeXu a automaticky stahovat chybějící balíčky, bohužel již teď není open-source. Dále jsem z ní převzal nápad na strukturu popsanou v sekci o zvýrazňování syntaxe 2.9.3. Tyto aplikace dříve mezi sebou komunikovaly pomocí zpráv Intent popsaných v sekci 2.8.1. Jak mají teď řešenou komunikaci mezi sebou, nevím.

2.13.2 MathMagic Lite (com.infologic.mathmagiclite)

MathMagic Lite je Android aplikace pro sestavování rovnic a jejich následný export v několika formátech (PNG, LaTeX). Tato aplikace se asi nejvíce blíží k zadání mé práce. Proto jsem se u ní inspiroval, zejména ve stavbě menu pro výběr matematických symbolů. Umí vyexportovat i výstup pro LaTeX, ale nedodává použité balíčky. Během mého testování LaTeXový kód nešel načíst zpátky do aplikace.

2.13.3 Equation Editor (com.vic.equationeditor)

Další Android aplikace pro sestavování rovnic je Equation Editor. Tato aplikace využívá vlastního pseudo kódu pro jejich vytváření. Jde tedy o řešení, které jsem popisoval v sekci 2.2.1. Výhoda této aplikace je velmi rychlé zobrazování náhledu. Nevýhodou je složité zadávání syntaxe a nedostatečný export (nedává uživateli vybrat, kam chce výsledek uložit a je zde na výběr pouze JPEG).

2.13.4 VerbTeX LaTeX Editor (verbosus.verbtex)

VerbTeX LaTeX Editor je editor a kompilátor LaTeXového kódu pro Android. Zkoušel jsem používat pouze neplacenou verzi. Nevýhodou této aplikace je, že kompilace LaTeXového kódu probíhá na vzdáleném serveru (popsáno v sekci 2.1.1). Což může být v případě pomalého datového připojení (či žádného) překážkou. Také označování syntaxe textu na pomalejších zařízení mi funguje zpomaleně.

2.13.5 TeXPad

TeXPad je aplikace pro iOS a OS X. Je to velmi propracované LaTeX IDE[27] s rozšířenými možnostmi psaní LaTeXu a správy projektu.

2.13.6 MyScript MathPad

MyScript MathPad je aplikace pro iOS s velmi kvalitním rozpoznáváním ručně psaných matematických výrazů a jejich následnou konverzí do LaTeXu. Nejdou

[27] Integrated Development Environment

tam tak dobře zadávat komplikovanější výrazy jako v moji aplikaci, ale zadávání je rychlejší.

2.13.7 Další

Kromě Android aplikací jsou tu i desktopové aplikace, které umožňují sestavovat rovnice. Například Microsoft Office obsahuje Microsoft Equation. Svoji obdobu má i OpenOffice a LibreOffice. LaTeXových IDE na systémech Windows, Linux či OS X je pak celá řada.

3. Uživatelská dokumentace

Tato sekce obsahuje popis použití aplikace Editor LaTeXu a jeho pluginu Editoru rovnic na mobilním operačním systému Android. Samotný Editor LaTeXu lze využít pro kompilování LaTeXových dokumentů s PDF výstupem. V případě, že vám chybí LaTeXový balík, aplikace ho automaticky stáhne. Editor rovnic nabízí vytváření a editaci matematických výrazů s přecházením mezi interaktivním ovládáním a přímou modifikací LaTeXového kódu. Editor rovnic je tedy vhodný jak pro pokročilé uživatele, tak pro začátečníky. Je také napojen na Editor LaTeXu tak, aby Editor LaTeXu umožňoval editaci pomocí Editoru rovnic, těch částí dokumentu které obsahují matematické výrazy. Alespoň pro správné použití Editoru LaTeXu je nezbytná znalost LaTeXové syntaxe, proto je dobré v případě, že neovládáte LaTeX si pročíst některou z knih o LaTeXu. Pěkný průvodce je například „The Not So Short Introduction To LATEX 2ε“[8].

3.1 Předpoklady

Aplikaci je možné spustit na jakémkoliv zařízení s Androidem, které má verzi systému alespoň 2.3 a nebo vyšší a používá architekturu procesoru ARM.

Veškeré ukázky v této dokumentaci jsou pro zařízení s větším displejem (tablety). Pro ostatní zařízení (telefony) jsou principy stejné, pouze rozmístění jednotlivých prvků na displeji se může lišit.

3.2 Instalace

V případě, že jsou aplikace instalovány na zařízení, které obsahuje „Google Play“, lze aplikace nainstalovat pomocí něj. Může jít, ale o novější verze, než jsou popsané v tomto projektu. Pro získání jednoznačného výsledku skrz vyhledávač v obchodu „Google Play“, lze zadat frázi „cz.talos.latexeditor“ pro nalezení Editoru LaTeXu a pro Editor rovnic lze použít frázi „cz.talos.latexeditor.equationeditor“.

Pro instalaci pomocí přiložených APK souborů musíte nejdříve povolit v nastavení zařízení instalace aplikací z cizích zdrojů. Poté stačí nahrát APK soubory do zařízení a spustit je. O zbytek se již postará instalační průvodce.

Pokud nainstalujete nejdříve Editor LaTeXu a až poté Editor rovnic je nutné Editor LaTeXu restartovat pro správné načtení jeho pluginu.

3.3 Nápověda v aplikaci

Při prvním spuštění Editoru rovnic či Editoru LaTeXu, dostanete nabídku pro spuštění nápovědy přímo v aplikaci. Pokud budete chtít zobrazit nápovědu později, tak je umístěná v menu.

Obrázek 3.1: Hlavní obrazovka aplikace s nainstalovaným Editorem rovnic

3.4 Editor LaTeXu

3.4.1 První spuštění

Při prvním spuštění si můžete vybrat, kde bude umístěn váš texmf[1] adresář. Poté se provede instalace několika základních LaTeXových balíků pro sázení jednoduchých dokumentů. Tato instalace se provede pokaždé, když se aplikace zapne s nově nainstalovaným pluginem, který vyžaduje balíky, které ještě nebyly nainstalovány.

3.4.2 Vytvoření dokumentu

Jsou dvě možnosti jak vytvořit nový dokument, buď úplně nový nebo ze šablony.

Nový dokument

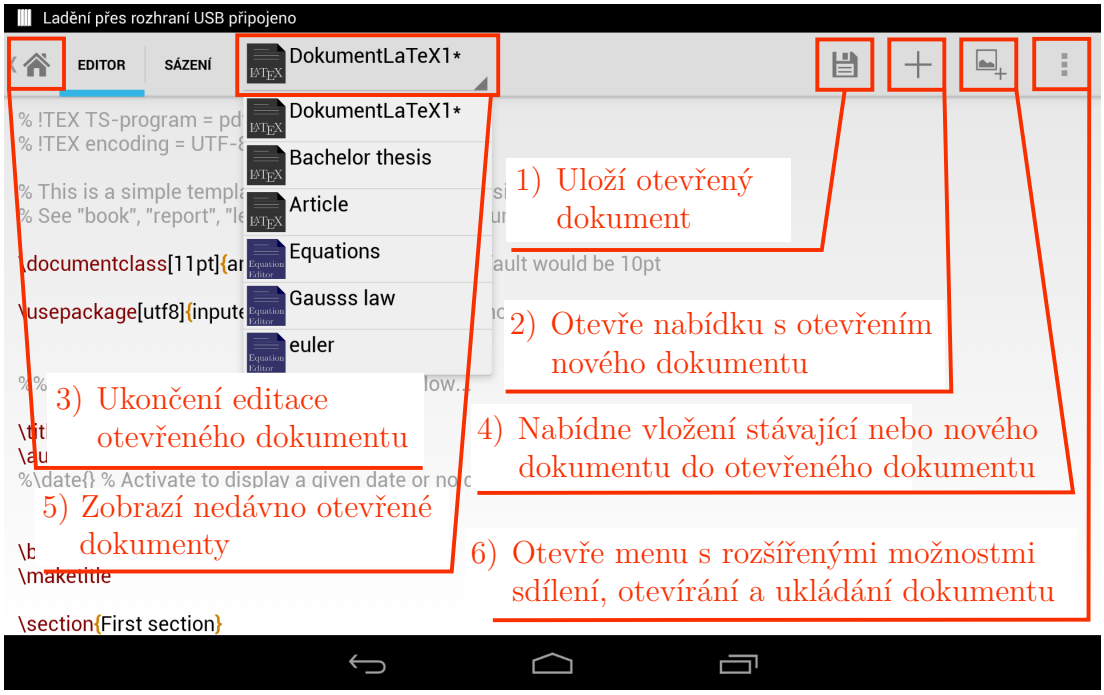
Vytvoření nového dokumentu lze provést z hlavní obrazovky, která je zobrazena na obrázku 3.1. Kliknutím na ikonu s nápisem „Editor LaTeXu“ se otevře nový dokument.

Nový dokument lze také otevřít během editování jiného dokumentu. Stačí kliknout na ikonu zobrazenou na obrázku 3.2 v bodě 2).

Nový dokument ze šablony

Pro otevření dokumentu ze šablony klikněte na položku v menu označenou „Šablony“ na hlavní obrazovce, znázorněné na obrázku 3.1 a poté zvolte šablonu

[1] TeX and METAFONT



Obrázek 3.2: Obrazovka s ukázkou editace dokumentu a popisem tlačítek.

kliknutím na její ikonu.

3.4.3 Ukládání dokumentu

V aplikaci se rozlišují tři druhy uložení:

1. **Dokument** - Soubory uložené jako dokumenty, se ukládají do výchozího adresáře pro dokumenty, který se dá změnit v nastavení. Každý dokument je uložen do zvláštní složky do které se ukládají i jeho pomocné soubory. Zvláštní složka do které se dokument uloží, má stejný název jako je název dokumentu.

2. **Šablona** - Šablony se ukládají stejným způsobem jako dokumenty, jen jsou uloženy ve skrytém adresáři.

3. **Soubor** - Soubor se dá uložit kamkoliv.

Při vytvoření nového dokumentu se dokument vždy ukládá jako dokument (tím se myslí použití tlačítka z obrázku 3.2 v bodě 1)). Pro uložení jiným způsobem se musí otevřít menu zobrazené na obrázku 3.2 v bodě 6), v menu vybrat položku „Uložit jako“, v podmenu vybrat položku „LaTeX“ a v otevřeném dialogovém okně zvolit způsob uložení.

V této dokumentaci se nazývá editovaný soubor LaTeXu dokument i soubor, to znamená, že se nerozlišuje mezi těmito pojmy tak, jak jsou uvedeny v této sekci. Názvosloví uvedené v této sekci, platí pouze když se mluví o ukládání dokumentu/souboru a jasně se to zdůrazní.

3.4.4 Otevírání dokumentu

Otevření dokumentu obsahujíc LaTeXový kód lze provést několika způsoby.

1. **Nedávno otevřený dokument** - Nedávno otevřené dokumenty lze procházet buď na hlavní obrazovce pomocí položky v menu „Nedávno otevřené dokumenty“, jak je zobrazeno na obrázku 3.1. Nebo v případě, že již editujeme nějaký dokument, je možné si zobrazit poslední otevřené dokumenty, pomocí tlačítka zobrazeného na obrázku 3.2 v bodě 5).

2. **Otevřít soubory uložené jako dokumenty** - Soubory uložené jako dokumenty lze procházet, pomocí položky „Moje dokumenty“ na hlavní obrazovce, která je zobrazena na obrázku 3.1.

3. **Otevření z úložiště zařízení** - Pro otevření souboru z úložiště zařízení je možné buď využít položku „Úložiště“ v menu na hlavní obrazovce zobrazenou na obrázku 3.1 nebo jakoukoliv externí aplikaci, která dokáže otevřít soubor v jiné aplikaci.

3.4.5 Výstup

Pro vysázení LaTeXového dokumentu je nutné mít dokument se správným kódem. Poté ho můžeme vysázet a vyexportovat do několika různých formátů. Formáty, které jsou k dispozici jsou PDF, JPEG, PNG, WEBP a SVG. Aplikace nabízí několik akcí, které s těmito výstupy lze provést:

1. **Sdílet** - Nabídne všechny aplikace, které podporují sdílení daného formátu. To můžou být různé e-mailové aplikace, či sociální sítě.

2. **Otevřít** - Nabídne všechny aplikace, které podporují otevření daného formátu. To může být například galerie.

3. **Uložit** - Nabídne uložení obrázku v úložišti zařízení.

Všechny tyto akce se provádí pomocí menu na obrázku 3.2 v bodě 6). Pokud chcete sledovat průběh sázení dokumentu, přepněte z výchozí záložky „EDITOR“ na záložku „SAZBA“.

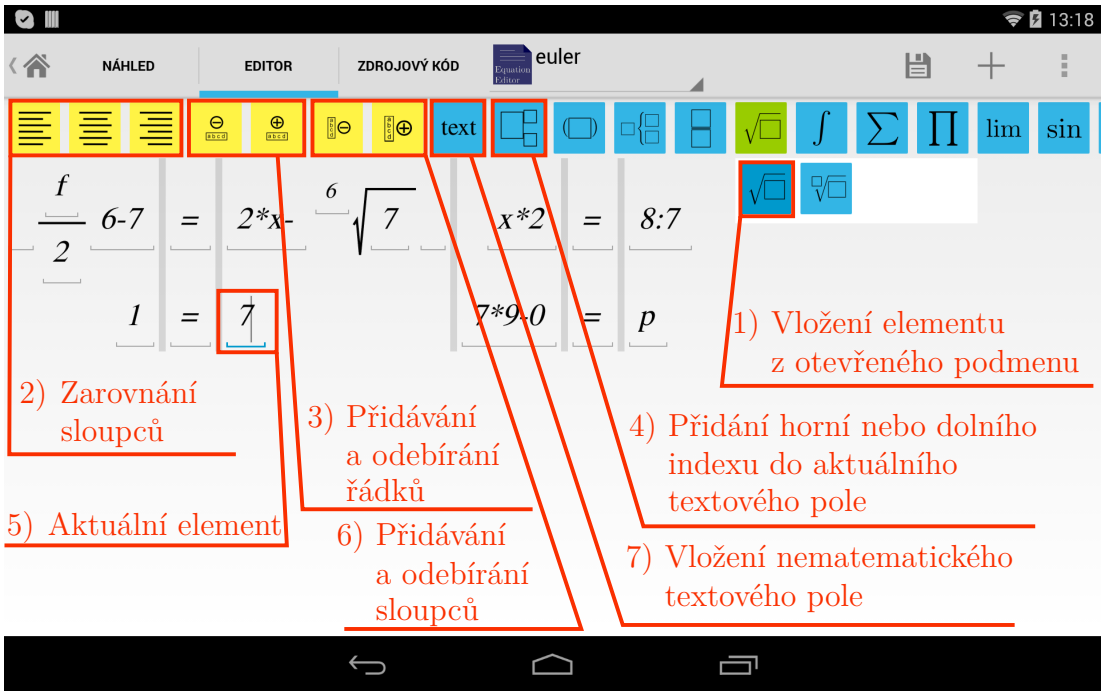
3.4.6 Vložení souboru do aktuálního dokumentu

Do aktuálně otevřeného LaTeXového dokumentu lze vložit jiný dokument. To lze provést pomocí tlačítka zobrazeného na obrázku 3.2 v bodě 4). To nám nabídne výběr, buď vložení souboru uloženého jako dokument nebo jakéhokoliv souboru. Po tomto výběru se nám ukáže seznam souborů a tři možnosti vložení. Buď můžeme vložit celý obsah vybraného souboru pomocí možnosti „text“ nebo ho můžeme vložit jako odkaz, pomocí LaTeXových maker `\input{..}` nebo `\include{..}`.

Pokud použijeme makra, která vkládají jiný soubor, ať už ručně nebo pomocí procedury popsáné výše, tak tyto makra se vždy stávají klikatelné. To znamená, že pokud na ně dlouze kliknete, otevře se dokument, na který tato makra odkazují.

3.5 Editor rovnic

Editor rovnic podporuje vytváření, ukládání, otevírání a exportování do týchž výstupních formátů jako Editor LaTeXu, který je popsaný v sekci 3.4. V případě výstupních formátů máme na výběr o jednu možnost více oproti Editoru LaTeXu. Nabízí se zde nový výstup „Rovnice“. Tento výstup uloží přímo kód, který se dá znovu otevřít v Editoru rovnic.



Obrázek 3.3: Obrazovka s ukázkou editace rovnice.

3.5.1 Interaktivní vytváření matematických výrazů

Přidávání nových elementů

Elementy se přidávají pomocí menu. Ne všechny elementy v menu jsou pro přidávání nových elementů. Pouze modře označené položky přidávají nové elementy (nebo otvírají podmenu s položkami, které přidávají nové elementy). Příklad položky z menu, která přidává nový element, je zobrazen na obrázku 3.3 v bodě 1), tento element bude vložen do aktuálně označeného textového pole na místo kurzoru, na obrázku je toto textové pole označené bodem 5).

Horní a dolní indexy, limity a nepovinné parametry

Po kliknutí na většinu vložených symbolů se otevře dialogové okno, ve kterém je možné u daného symbolu nastavit, jestli má mít indexy nebo limity a také jestli mají být horní nebo dolní. V případě, že symbol má nepovinné parametry, jako třeba odmocnina, lze tento parametr také přidat nebo odebrat v tomto dialogovém okně.

Pro přidání indexů k textovému poli se používá tlačítko na obrázku 3.3 v bodě 4), které otevře podmenu s výběrem horního a dolního indexu. Po kliknutí na položky v podmenu se zvolený index přidá do aktuálně vybraného textového pole na místo kurzoru.

Přidávání a odebírání řádků a sloupců

Přidávat či odebírat řádky nebo sloupce lze pouze v případě, že se aktuálně označený element nachází v elementu, který podporuje více-řádkové nebo více-sloupcové prostředí. Elementy, které toto podporují, jsou buď matice, kořenový element a nebo specifická LaTeXová prostředí. Na obrázku 3.3 v bodě 3) jsou označená tlačítka pro přidávání a odebírání řádků a v bodě 6) pro přidávání a odebírání řádků.

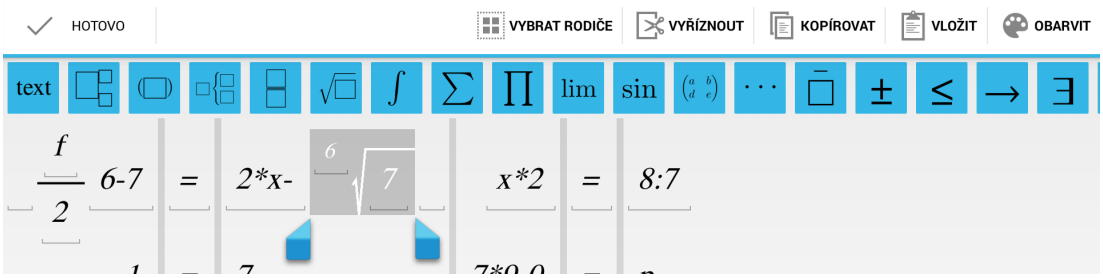
Pro vizuální rozlišení sloupců při vytváření matematických výrazů se mezi každý sloupec vkládají tlusté šedé čáry.

Zarovnávání sloupců

Pro jednotlivé sloupce lze nastavit zarovnání jejich elementů. Na obrázku 3.3 jsou tlačítka, která se používají pro tato zarovnání, označena bodem 2). Proto, aby bylo možné zarovnávat sloupce, je nutné mít alespoň dva řádky.

V maticích lze zarovnávat pouze všechny sloupce najednou. Toto omezení vyplývá z možností LaTeXu. Vložte LaTeXové prostředí „array“ proto, aby bylo možné zarovnávat každý sloupec zvlášť.

Výběr více elementů, jejich kopírování, vkládání, mazání a obarvování



Obrázek 3.4: Ukázka kontextového menu při označení elementu.

Pro výběr více elementů zvolte první element a proveďte na něm dlouhý klik. Element se poté označí a vedle něj se objeví modrá táhla, pomocí kterých lze výběr elementů rozšiřovat. Rozšiřovat pomocí táhel lze vždy jen na sousedy na stejné úrovni. Modrá táhla jsou zobrazena na obrázku 3.4. Na tomto obrázku je také ukázané kontextové menu, které se zobrazí po označení jednoho nebo více elementů. Kontextové menu se objeví v horní části displeje a nabízí nám několik nových tlačítek, která ovlivňují aktuální výběr.

1. **Vybrat rodiče** - Toto tlačítko se používá pro vybrání elementu na vyšší úrovni.
2. **Vyříznout** - Vymaže aktuální výběr a jeho LaTeXový kód vloží do systémové schránky.

3. **Kopírovat** - LaTeXový kód aktuálního výběru vloží do systémové schránky.

4. **Vložit** - Vezme LaTeXový kód ze systémové schránky a sestaví z něho elementy, kterými přepíše aktuální výběr.

5. **Obarvit** - Otevře dialogové okno s paletou barev. Po potvrzení zvolené barvy se aktuální výběr danou barvou obarví.

Pro vybrání jen části textu se používá dvojklik na daný text. V tomto módu výběru, ale nejsou dostupné stejné možnosti jako při vybrání celého elementu.

Vložení nematematického textu

Matematický text se vysází vždy bez mezer a pomocí jiného fontu, než zbytek LaTeXového dokumentu. Pokud chcete vložit do rovnice kus normálního textu s mezerami, je možné vložit tento speciální element pomocí tlačítka na obrázku 3.3 v bodě 7).

3.5.2 Zobrazení rychlého náhledu matematického výrazu

Při editování matematického výrazu přepněte na záložku „NÁHLED“, kde se po chvíli čekání zobrazí náhled výstupu editovaného matematického výrazu.

3.5.3 Vytváření rovnice pomocí LaTeXového kódu

Pro práci s LaTeXovým kódem aktuálně vytvářeného matematického výrazu stačí změnit záložku na „ZDROJOVÝ KÓD“. Poté se zobrazí LaTeXový kód v případě úpravy LaTeXového kódu změna se projeví i v interaktivním režimu. V kódu nelze používat nepodporovaná makra. Seznam podporovaných maker najdete v příloze B.

3.5.4 Propojení s Editorem LaTeXu

Po nainstalování pluginu Editor rovnic získá Editor LaTeXu získá možnosti editace matematických výrazů.

Při vkládání jiných souborů do Editoru LaTeXu, které je popsané v sekci 3.4.6 získáte v otevřené nabídce novou možnost vložení nové rovnice. Tato možnost přímo otevře Editor rovnic, kde můžete sestavit novou rovnici a po ukončení editace dokumentu pomocí tlačítka zpět nebo tlačítka zobrazeného na obrázku 3.2 v bodě 3) se vloží editované rovnice do původně otevřeného dokumentu. Rovnice lze také vložit ze stávajících dokumentů. Při každém vkládání rovnice jsou na výběr matematická LaTeXová prostředí, do kterých se rovnice vloží.

V případě, že se v otevřeném LaTeXovém dokumentu nachází matematické prostředí, tak pozadí jeho obsahu je barevně zvýrazněno. To znamená, že na tuto část kódu lze použít dlouhý klik pro jeho editování v Editoru rovnic.

4. Programátorská dokumentace

Tento projekt obsahuje dvě aplikace. Jedna aplikace je hlavní a druhá je plugin (Editor rovnic) pro hlavní aplikaci. Technicky tento projekt obsahuje dva pluginy, ale jeden z nich (Editor LaTeXu) je obsažen již v hlavní aplikaci.

I přesto, že druhá aplikace je pouze plugin, z pohledu operačního systému Android se jedná o aplikaci. Obě aplikace jsou napsány v Javě, což je oficiální jazyk pro programování Android aplikací, s tím že využívají některé programy napsané v C.

4.1 Android

Před vysvětlením architektury tohoto projektu je důležité si objasnit několik pojmů a základních faktů o tom, jak fungují jednotlivé komponenty systému Android. Nejsou zde popsány všechny nejdůležitější komponenty a funkce v systému, ale jsou tu popsány ty, které jsou použity v tomto projektu.

Android podporuje multi-tasking. Běžící aplikace se rozdělují na dva druhy. Ty, které běží v popředí (foreground) a ty, které jsou minimalizované (background). Systém může kdykoliv minimalizovanou aplikaci ukončit, stejně tak může ukončit i aplikaci v popředí v případě, že jiná aplikace s vyšší prioritou bude potřebovat více systémových prostředků. Před ukončením umožní systém aplikaci uložit data tak, aby byla schopná se znovu nastartovat ve stavu, ve kterém byla ukončena. O správné uložení a interpretování dat z aplikace se již musí postarat vývojář aplikace.

Jednotlivé komponenty mají tzv. životní cyklus (lifecycle). Ten rozhoduje, mezi jakými stavy tato komponenta přechází a také kdy mezi nimi přechází. Občas tento cyklus může ovlivnit vývojář, jindy je daný systémem.

4.1.1 Android manifest

Android manifest je XML soubor, který předává nezbytné informace o aplikaci systému Android. Tento soubor se musí vždy nacházet v kořenové složce aplikace. Bez něj není možné aplikaci skompilovat a zabalit do balíku APK. Tento soubor by měl vždy obsahovat alespoň tyto informace:

1. **Jméno Java balíku** - To slouží jako identifikátor aplikace. Mělo by být unikátní. Je také dobrý zvyk používat jméno tohoto balíku pro ostatní třídy aplikace. Je to jediný povinný parametr pro kompilaci aplikace v Android manifestu, ostatní vyjmenované jsou nutné pro správnou funkčnost aplikace.

2. **Oprávnění** - Všechna oprávnění, která naše aplikace potřebuje, je nutné správně zadefinovat. Mezi tato oprávnění patří například přístup k Internetu nebo přístup k datovému úložišti zařízení.

3. ***Activity*** - Pokud chceme, aby naše aplikace byla spustitelná, musíme v manifestu definovat, alespoň jednu *Activity*. Každé *Activity* můžeme nastavit jeden nebo více tzv. Intent filterů. V každém Intent filteru můžeme definovat, při jaké příležitosti má aplikace spouštět tuto *Activity*.

4. **Verze aplikace** - Textový a číselný identifikátor verze aplikace, s jehož pomocí systém pozná, zda je nainstalovaná verze aplikace aktuální. Aplikace musí tuto informaci obsahovat, pokud je distribuována v různých obchodech jako třeba Google Play.

V tomto souboru lze definovat spoustu dalších parametrů, jako třeba defaultní téma aplikace, či předefinovat hlavní aplikační třídu.

4.1.2 Application

Application je objekt, který je svázán s procesem, ve kterém běží daná aplikace. Lifecycle tohoto objektu je tedy pevně daný tím, kdy uživatel nebo systém ukončí, zastaví nebo spustí aplikaci. Třídu tohoto objektu lze přepsat a tuto modifikovanou třídu vnutit aplikaci tím, že ji nastavíme jako výchozí v Android manifestu.

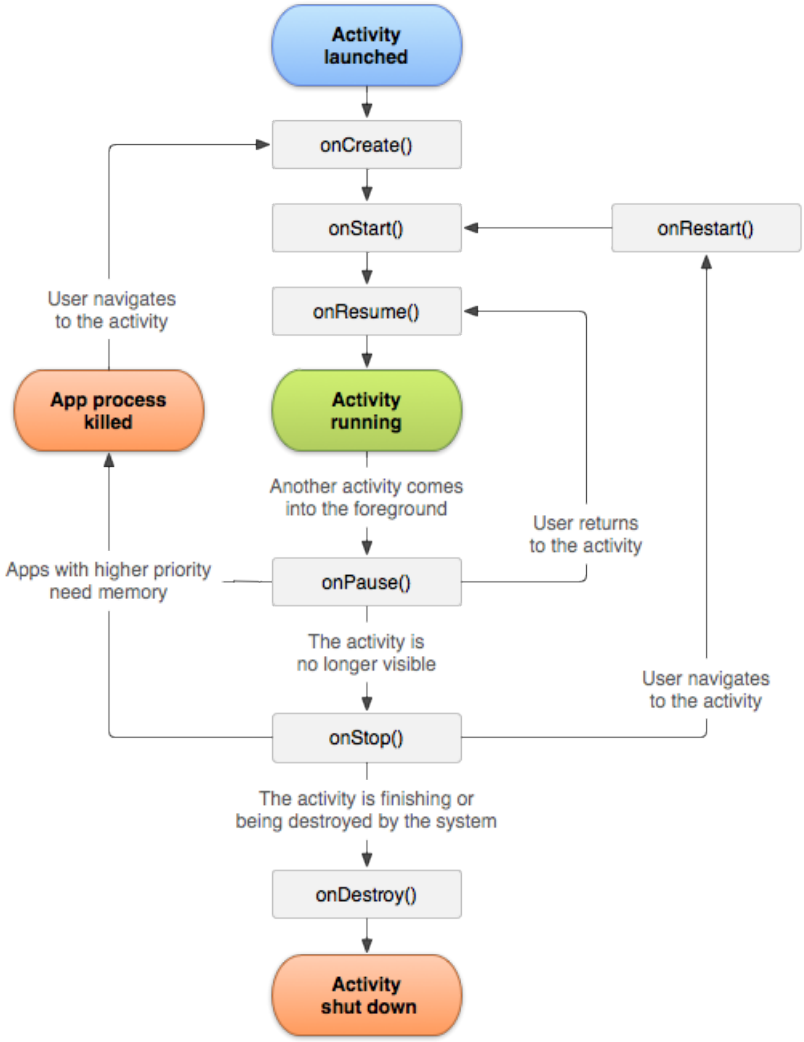
Tím, že použijeme modifikovanou třídu *Application*, můžeme docílit toho, aby se některé akce provedly ve chvíli, kdy startuje aplikace, bez toho, aby záleželo na tom jaká *Activity* nebo *Service* bude použita. Toho docílíme přepsáním metody *onCreate()*. Také je možné třídu tohoto objektu využívat jako singleton.

4.1.3 Activity

Objekt *Activity* by měl provádět vždy nějakou jasně danou akci nebo činnost. Například v aplikaci pro prohlížení obrázků bychom mohli mít dvě *Activity*. Jedna by reprezentovala zobrazení obrázku a druhá by zobrazovala všechna alba (ta by byla hlavní). Potom jiná aplikace může poslat požadavek, že chce zobrazit obrázek, protože to sama neumí. *Activity* pro zobrazení obrázků má zaregistrovaný intent filter v android manifestu, který říká, že dokáže zobrazit obrázek. Systém tedy může uživateli nabídnout otevřít tuto aplikaci s *Activity* pro zobrazení obrázku. Přitom pokud by si tuto aplikaci chtěl uživatel přímo spustit, tak by se mu otevřela hlavní *Activity* pro prohlížení alb.

Aplikace může tedy obsahovat více než jednu *Activity* a každá *Activity* může být spuštěná v několika instancích. V okamžiku, kdy jedna *Activity* spustí druhou pro nějakou novou akci, umístí se stará do zásobníku. Poté když uživatel naviguje zpátky, ať už pomocí tlačítka zpět, které by mělo být na všech zařízeních s Androidem v softwarové nebo hardwarové podobě, či pomocí jiného navigačního elementu, tak se ukončí aktuální *Activity* a vytáhne se ze zásobníku ta předchozí. V případě, že byla mezitím ukončena systémem, tak se znovu nastartuje. Jeden takový zásobník je nazývaný úkol (Task). Pokud již nejsou žádné další *Activity* v zásobníku, tak se úkol ukončí. Aplikace nemusí startovat jenom svoje *Activity*, ale může využít i těch z jiných aplikací.

Předtím, než je *Activity* zničena, je zavolána metoda *onSaveInstanceState(Bundle)*, která dává možnost uložit aktuální stav *Activity* ve formě primitivních objektů, které jsou uloženy v objektu *Bundle*. *Bundle* je objekt, který představuje asociativní pole. Akceptuje pouze primitivní data a objekty těch tříd, které implementují *Parcelable* (to je obdoba *Serializable* z Javy). Tento objekt je potom předán při obnovení *Activity* ve volání metody *onCreate(Bundle)*, pokud jde o novou *Activity* (tzn. neobnovenou), tak je objekt typu *Bundle null*.



Obrázek 4.1: Na obrázku je zobrazen životní cyklus *Activity*. Obrázek je převzat z Android Guide [11].

Životní cyklus *Activity* tak, jak je zobrazený na obrázku 4.1, obsahuje tři základní cykly:

1. **Celý životní cyklus** začíná při prvním zavolání metody *onCreate(Bundle)* a končí při zavolání metody *onDestroy()*

2. **Viditelný cyklus** začíná při zavolání metody *onStart()* a končí při zavolání metody *onStop()*. Během tohoto cyklu může uživatel *Activity* vidět na svém displeji, ale ta nemusí být v popředí a interagovat s uživatelem.

3. **Cyklus v popředí** začíná při zavolání metody *onResume()* a končí při zavolání metody *onPause()*. Během tohoto cyklu je *Activity* v popředí a uživatel s ní může interagovat.

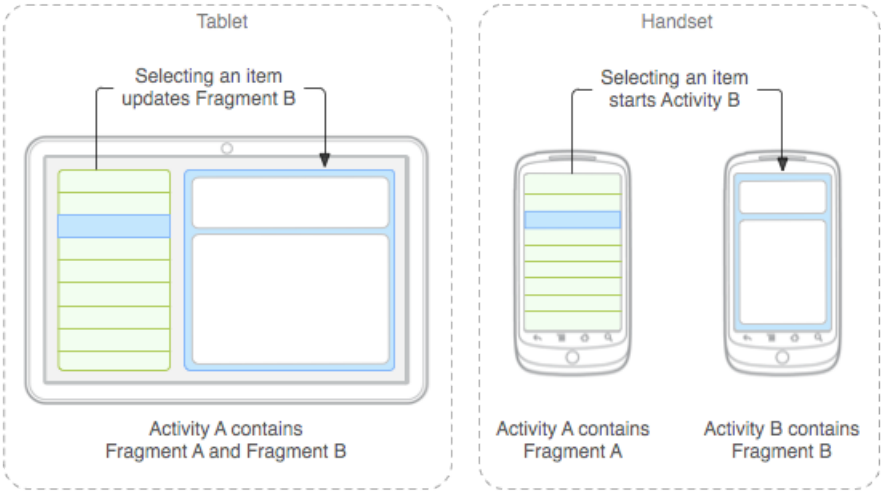
Pokud chceme *Activity* bez UI, můžeme využít komponentu *Service*.

Jakákoliv v popředí běžící *Activity*, může nastartovat novou *Activity* nebo *Service*. Pro upřesnění jakou *Activity* chceme nastartovat, je použit objekt *Intent*. Objekt *Intent* může být buďto explicitní nebo implicitní. Explicitní specifikuje

přesně, kterou *Activity* má nastartovat jejím celým jménem. Většinou se explicitní *Intent* využívá hlavně při interním spouštění *Activity*. V implicitním se oznámí jen záměr, například že chceme sdílet obrázek, potom nám systém nabídne všechny aplikace, které mají nastavený intent filter pro tento záměr, v tomto případě půjde například o různé e-mailové aplikace, sdílení přes bluetooth nebo různé sociální sítě.

4.1.4 Fragment

Komponenta *Fragment* se dá popsat jako *Activity*, jejíž vlastnosti byli omezeny jen pro zobrazování UI. Jejich podstatnou vlastností je, že je možné vložit neomezený počet *Fragment* objektů do *Activity* nebo do jiného *Fragment* objektu. Hlavní motivace pro vytvoření fragmentů je jejich znovupoužití v různých rozvrženích (layoutech). Jeden z příkladů je znázorněn na obrázku 4.2.



Obrázek 4.2: Ukázka, jak mohou být dva rozlišné návrhy UI pomocí fragmentů kombinovány do jedné *Activity* pro tablet a do dvou pro telefon. Obrázek byl převzat z Android Guide [11].

Každý *Fragment*, musí mít právě jednu hostitelskou *Activity*, která se stará o jeho životní cyklus. To znamená, že když je zastavena *Activity*, tak jsou zastaveny i její fragmenty. Pokud je spuštěná, je možné, aby její fragmenty byly spouštěny i zastavovány. UI fragmentu žije uvnitř objektu *ViewGroup*, který patří hostitelské *Activity*.

Načítání a obnovení stavu funguje podobně jako v případě *Activity*. Životní cyklus fragmentu lze změnit voláním jeho metody *setRetainInstance(boolean)* s parametrem *true*. V tom případě se *Fragment* nezničí spolu s *Activity*, ale pouze se od ní odpojí a poté se připojí k nově vytvořené *Activity*. V tomto případě není třeba obnovovat jeho stav, protože ten zůstane zachován.

Speciální potomek třídy *Fragment* je *DialogFragment*. Ten si uchovává objekt *Dialog*, který využívá k zobrazení svého obsahu. *Dialog* je další z komponent UI, která má za úkol zobrazit svůj obsah v plovoucím okně. Výhoda použití *DialogFragment* místo pouhého dialogu je, že *DialogFragment* si dokáže uložit svůj stav, tedy při rotaci displeje nebo při znovuvytvoření *Activity* se správně

obnoví i se svým dialogem. Další výhoda je, že *DialogFragment* lze využít i jako normální *Fragment*, to znamená, že jeho obsah lze vložit i jinam než do plovoucího okna, což lze využít pro sestavování různých rozvržení (layoutů). Později, když v textu budu napsáno, že se otevře nějaká podtřída *DialogFragment*, znamená to, že se otevře objekt *Dialog* pomocí této podtřídy.

4.1.5 View

Objekt *View* je základní stavební kámen UI. Každé *View* obsadí obdélníkovou oblast a poté je zodpovědné za její vykreslení a za předávání událostí (jako třeba doteků nebo kliků). Důležitá podtřída *View* je *ViewGroup*, to je kontejner, do kterého lze vnořit další *View*. *ViewGroup* se stará o vykreslení obdélníku, který zahrnuje obdélníky všech *View*, které *ViewGroup* obsahuje.

Při vytvoření vlastního poděděného *View*, je nutné přepsat metodu *onDraw(android.graphics.Canvas)*, která se volá pokaždé, když by mělo být *View* překresleno.

V každém okně, které vytváří *Activity*, jsou *View* sestaveny do stromu, takovýto strom je v každé *Activity* právě jeden.

4.2 Použité knihovny

V hlavní aplikaci jsou použity tři druhy knihoven.

4.2.1 Android knihovny

Android knihovny obsahují kromě tříd napsaných v jazyce Java také další zdroje, jako třeba rozvržení rozhraní (layouts) nebo různé obrázky a další vykreslitelné objekty (drawables).

HoloEverywhere

Knihovna, která umožňuje zachování zpětné kompatibility tématu HOLO a některých funkcí systému se staršími verzemi systému Android. Tato knihovna je pod licencí LGPL [4].

SlidingMenu

Pomocí této knihovny, lze vytvořit jednoduše konfigurovatelné vysouvací menu. Tato knihovna je pod Apache licencí [2].

LAHDownloader

Tato knihovna umožňuje stahování souborů z internetu se zobrazením stavu stahování v systémové notifikační liště. Na tuto knihovnu se vztahuje licence uvedená v příloze A.

ColorPickerView

Tato knihovna vytváří UI komponentu s paletou barev. Knihovna je pod Apache licencí [2].

4.2.2 Java knihovny

Toto jsou knihovny, které obsahují pouze třídy napsané v jazyce Java.

LAHTeX a LAHSpectre

Tyto knihovny se starají o správné zpracování výstupu z TeX Live a automatické vyřešení chyb z tohoto výstupu (například stažením dodatečného LaTeXového balíku). Tyto knihovny také obsahují správce úkolů, který se stará o asynchronní operace těchto knihoven a o jejich spouštění ve správném pořadí. Na tyto knihovny se vztahuje licence uvedená v příloze A.

4.2.3 C knihovny

V tomto projektu jsou použité i některé knihovny/programy napsané v jazyce C.

MuPDF

Tato knihovna slouží k vykreslování jednotlivých stránek PDF do různých rastrových a vektorových formátů ve vysoké kvalitě. Více informací o licenci MuPDF je v sekci 2.7.1.

TeX Live

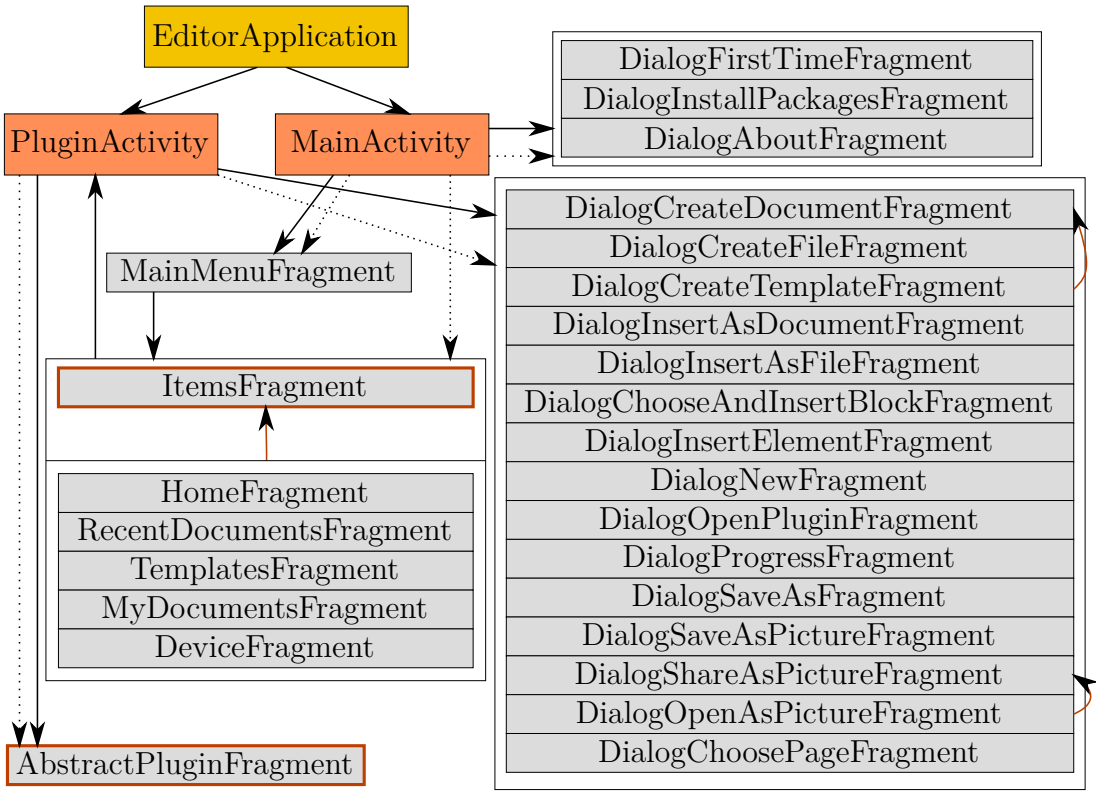
Knihovna LAHTeX využívá různé programy z distribuce TeX Live.

4.3 Hlavní aplikace

Hlavní aplikace obsahuje kompilátor LaTeXu, konvertor obrázků, prohlížeč souborů, textové pole se zvýrazňováním syntaxe, rozhraní pro načítání a spouštění pluginů a plugin Editor LaTeXu.

4.3.1 Propojení UI komponent

Na obrázku 4.3 je zobrazeno, jak jsou jednotlivé hlavní komponenty UI na sebe napojené a jak se mezi sebou přepínají.



Obrázek 4.3: Popis UI komponent hlavní aplikace. Každý obdélník s jménem představuje jednu třídu, která představuje jednu UI komponentu. Žlutě je označena komponenta *Application*. Červené jsou *Activity* a šedé jsou fragmenty, všechny fragmenty, které začínají slovem Dialog, dědí od třídy *DialogFragment*. Objekty, které mají červené rámce, jsou abstraktní třídy.
Plné černé šipky ukazují z komponenty, která otevírá komponentu, do které šipka ukazuje. Přerušované šipky ukazují vždy do fragmentu z jeho hostitelské *Activity* a červené šipky znázorňují dědičnost. Když je více tříd sdruženo v jednom velkém rámci, tak se všechny šipky mířící do tohoto rámce považují za šipky vedoucí do každého objektu zvlášť.

Následuje popis vlastností a funkcí jednotlivých UI komponent. Nejsou zde popsány vlastnosti a funkce jejich předků, ty už byly popsány výše v sekci 4.1.

1. *EditorApplication*

   (a) **Inicializace** - Při startu aplikace se vytvoří všechny nutné složky (jde o složky, kde jsou uchovávány dokumenty, šablony a binární soubory). Správně nastaví konvertor obrázků (MuPDF) a stahovač LaTeXových balíků (LAHDownloader)

   (b) **Načítání pluginů** - Díky tomu, že tento objekt je singleton a lze k němu přistoupit skoro z jakékoliv komponenty, tak při spuštění aplikace stačí pluginy jednou načíst a uložit. Poté je možné k nim přistupovat z různých částí aplikace.

   (c) **Rozhraní pro kompilátor LaTeXu** - Knihovna LAHTeX, zmíněná v sekci 4.2, potřebuje pro svou práci třídu, která implementuje roz-

hraní *IEnvironment*. Toto rozhraní definuje například metodu *getPackage(String)*, která má na starost stáhnout LaTeXový balík. Také vrací složku s uloženými binárními soubory distribuce TeX Live a složku se systémovými fonty. Toto rozhraní má ještě několik dalších vlastností, které ale již nejsou tak důležité. Třída *EditorApplication* implementuje toto rozhraní.

2. *MainActivity*

   (a) **Spouštění při startu aplikace** - V Android manifestu má tato *Activity* nastavené ve svém intent filteru spouštění při normálním (například ze systémového menu) startu aplikace.

   (b) **Prvotní nastavení** - Při prvním spuštění *MainActivity* otevře *DialogFirstTimeFragment* pro nastavení umístění LaTeXových balíků.

   (c) **Nainstalování nutných balíků** - Při každém spuštění této *Activity* se otevře *DialogInstallPackagesFragment* v případě, že byl přidán nový plugin, který vyžaduje nějaké LaTeXové balíky, které ještě nejsou nainstalované. Tyto balíky se rozpoznají podle toho, že je daný plugin má uložené v assets.

   (d) **Zobrazení menu a obsahu jeho položek** - Menu je zobrazeno pomocí *MainMenuFragment*. Jeho zobrazení je závislé na šířce displeje. Pokud má 900dip a více, zobrazí se menu staticky. Pokud má méně, zobrazí se jako zasouvatelné. Pro zobrazení zasouvatelného menu je použitá knihovna SlidingMenu. Zbytek obsahové části (když nepočítáme Action bar) je vyhrazen pro obsah, který dodávají položky menu.

   (e) **Zobrazení nastavení a informací o aplikaci** - Ve standardním Android menu zobrazí položky pro otevření nastavení a informací o aplikaci.

   (f) **Otevření neuložených dokumentů** - Při každém spuštění se kontroluje, jestli uživatel nemá nějaké neuložené dokumenty. Pokud má, tak se otevře dialog (*RecoveryListDialog*) se seznamem neuložených dokumentů, kde si je uživatel může otevřít nebo smazat. Neuložené dokumenty se ukládají do složky „.recovery“.

   (g) **Uložení šablon z jednotlivých pluginů** - Při každém startu se kontrolují všechny pluginy jestli nemají ve svých „assetech“ ve složce „templates“ soubory, které jsou interpretované jako šablony. Takovéto soubory jsou uloženy do složky s ostatními šablonami do složky „.templates“.

3. *PluginActivity* - Třída tohoto objektu implementuje rozhraní *FragmentPluginListener* podrobněji popsané v sekci 4.4.1.

   (a) **Otevření UI pluginu** - Podle toho, jaký identifikátor pluginu dostane *PluginActivity*, použije příslušný objekt implementující *EditorPlugin* z daného pluginu. Z tohoto objektu, pomocí metody *getFragmentClass()*, potom získá fragment, který dědí od *AbstractPluginFragment* a vytváří UI pro otevíraný plugin v *PluginActivity*.

(b) **Ukládání obsahu** - *PluginActivity* poskytuje ukládání obsahu, který dodává aktuálně otevřený plugin. Obsah je uložen přímo, pokud je obsah otevřený ze zapisovatelného souboru. Pokud ne, tak se uživateli zobrazí dialog *DialogCreateDocumentFragment*, kde si vybere jméno dokumentu a uloží ho jako dokument. Uživatel také může využít možnost uložit obsah jako, poté má několik možnosti, jak uložit svůj obsah.

   i. **Jako dokument** - To znamená, že se obsah uloží do definované složky s dokumenty. A pro přehlednost se pro každý dokument vytvoří v této složce další složka s týmž názvem jako má vytvářený dokument. Tato činnost je provedena v *DialogCreateDocumentFragment*.

   ii. **Jako soubor** - To znamená, že se soubor vytvoří v jakékoliv složce, kterou si uživatel vybere. Provádí se v *DialogCreateFileFragment*.

   iii. **Jako šablonu** - Uloží se stejným způsobem, jako by to byl dokument, jen do jiné složky. Definováno v *DialogCreateTemplateFragment*.

(c) **Vysázení LaTeXového dokumentu** - Plugin si kdykoliv může vyžádat vysázení LaTeXového kódu a nazpátek dostane PDF.

(d) **Konverze PDF do bitmapy nebo SVG** - *PluginActivity* poskytuje konvertor, který je schopný převést do objektu *android.graphics.Bitmap* nebo získat textový výstup ve formátu SVG.

(e) **Sdílení výstupu** - Pokud plugin chce mít výstup ve formátu PDF, musí vždy poskytnout kromě svého obsahu také validní LaTeXový kód. Pomocí tohoto kódu se vytvoří PDF a z něho se vytvoří SVG nebo bitmapa. Bitmapa se převede na PNG, JPEG nebo WEBP. A tyto všechny výstupy (včetně samotného LaTeXového kódu a zdrojového souboru aplikace) se sdílí v rámci systému Android s ostatními aplikacemi, pomocí správně nastaveného *Intent*. Pro nastavení různých vlastností (jako třeba velikost) obrázků před sdílením se využívá *DialogShareAsPictureFragment*. Pokud má dokument více stránek zobrazí se ještě dialog *DialogChoosePageFragment* s výběrem stránek dokumentu pro výstup.

(f) **Otevírání vytvořených souborů** - Použijí se stejné výstupy a UI komponenty jako při sdílení výstupu, jen se jinak nastaví *Intent*.

(g) **Ukládání vytvořených souborů** - Použijí se stejné výstupy jako při sdílení výstupu. Ale pro uložení je nutné nastavit také cestu a ta se nastavuje v *DialogSaveAsFragment* a pro obrázky (společně s velikostí obrázku) v *DialogSaveAsPictureFragment*.

(h) **Otevření pluginu s omezenými možnostmi** - Aby jeden otevřený plugin mohl využít jiný, tak ho může otevřít v nové instanci *PluginActivity* s omezenými možnostmi. Hlavní rozdíl je, že v tomto režimu není možné mít soubor, do kterého se bude ukládat obsah. Je pouze možné uložit obsah do jiného souboru. Dále v tomto režimu není možné vytvářet nové dokumenty.

(i) **Vkládání obsahu jiného souboru nebo odkaz na jiný soubor** - Tato funkce narozdíl od předcházejících není inicializovaná v *PluginActivity*, ale v objektu dědícím od *AbstractPluginFragment*. Ten vyšle požadavek, že by chtěl vložit obsah, a *PluginActivity* na to otevře *DialogInsertElementFragment*. Ten nabídne tři možnosti. První možností je z aktuálně nainstalovaných pluginů vytvořit novou instanci *PluginActivity* s omezenými možnostmi a jí vytvořený obsah posléze vložit do aktuálně editovaného dokumentu. Dalšími dvěma možnostmi jsou vložení dokumentu (pomocí *DialogInsertAsDocumentFragment*) nebo souboru (pomocí *DialogInsertAsFileFragment*). Tyto dvě možnosti jsou skoro identické ve funkčnosti s tím rozdílem, že vložení jako dokument zobrazí pouze výběr pouze ze souborů, které byly uloženy jako dokumenty. Pro vložení souboru jako odkaz se využívají LaTeXová makra `\include{..}` a `\input{..}`. Při vkládání obsahu souboru/dokumentu tak v případě, že obsah byl vytvořen pluginem, jehož obsah by měl být uložen v LaTeXovém prostředí, tak uživatel dostane nabídku těchto prostředí pomocí *DialogChooseAndInsertBlockFragment*.

(j) **Otevírání jiného pluginu pro editaci části obsahu** - Po vložení obsah vytvořeného v jednom pluginu do jiného je možné editovat takový obsah opět v pluginu, který jej vytvořil. Uživatel musí tuto akci potvrdit v *DialogOpenPluginFragment*.

(k) **Otevření nového dokumentu** - Rychlé otevření nového dokumentu z nabídky všech nainstalovaných pluginů, pomocí *DialogNewFragment*.

(l) **Obnovení souborů** - Při každém minimalizování aplikace je vytvořena záloha aktuálně otevřeného dokumentu. Tato záloha se vytvoří ve stejném adresáři jako je původní soubor i se stejným jménem s příponou „.RECOVERY“, tento soubor je po uložení uživatelem smazán. Pokud takový soubor existuje při znovu otevření dokumentu, je uživateli nabídnuto jeho otevření. V případě, že daný dokument ještě nebyl nikdy uložen, tak se jeho záloha ukládá do speciální složky a o jeho znovuotevření se stará *MainActivity* při startu aplikace.

(m) **Ukládání historie otevřených dokumentů** - V případě, že se otevře nějaký uložený soubor, tak se uloží čas jeho otevření. Tento seznam souborů s jejich časy se ukládá v databázi, kterou spravuje třída *RecentDocumentsDatabase*.

(n) **Otevírání nedávno otevřených dokumentů** - V Action Baru zobrazuje *PluginActivity* komponentu *Spinner*, která slouží jako seznam položek, který se rozbaluje po kliknutí na aktuálně vybranou položku. V takovéto komponentě se zobrazuje seznam naposledy otevřených dokumentů z databáze spravované třídou *RecentDocumentsDatabase*.

4. *MainMenuFragment* - Zobrazí položky v menu pro fragmenty, na které vede šipka z tohoto fragmentu na obrázku obrázku 4.3. Potom zobrazí tyto fragmenty v *MainActivity*. Každý z těchto fragmentů zobrazuje svoje položky buď v *ListView* nebo *GridView*, to jsou komponenty pro zobrazení seznamu.

5. *HomeFragment* - Zobrazí ikonu pro každý z načtených pluginů. Každá z

těchto ikon po kliknutí otevře *PluginActivity* s identifikátorem svého pluginu.

6. *RecentDocumentsFragment* - Z databáze, kterou spravuje třída *RecentDocumentsDatabase*, vybere určitý počet nedávno otevřených dokumentů a zobrazí je uživateli, aby mohl rychle pokračovat v práci.

7. *MyDocumentsFragment* - Ze složky pro dokumenty jsou načteny soubory a poté zobrazeny v tomto fragmentu.

8. *TemplatesFragment* - Ze složky pro šablony jsou načteny soubory a poté zobrazeny v tomto fragmentu.

9. *DeviceFragment* - V tomto fragmentu je prohlížeč veřejného úložiště zařízení. Uživatel si zde může najít soubor s příponou, kterou definuje některý z jeho pluginů, a otevřít ho.

4.3.2 Kompilátor LaTeXového kódu

O kompilování LaTeXového kódu se stará knihovna LAHTeX, popsaná v sekci 4.2. Rozhraní k této knihovně je umístěno v objektu *ExtendedCompileDocument*, který využívá správce úkolů *ExtendedTeXMF*, aby mohl běžet asynchronně. Dále je zde objekt *ExtendedInstallPackage*, který je použit pro instalaci LaTeXových balíků. Pro správnou konfiguraci vyžaduje tato knihovna třídu implementující rozhraní *IEnvironment*, kzde se jedná o třídu *EditorApplication*. Všechny třídy začínající na *Extended* zmíněné v tomto odstavci dědí od odpovídajících tříd z knihovny LAHTeX, popsané v sekci 4.2.

4.3.3 Konvertor z PDF do výstupních formátů

Pro konverzi PDF do bitmapy a SVG je použitý program napsaný v jazyku C, který používá knihovnu MuPDF popsanou v sekci 4.2. Tento program je spouštěný jako samostatný proces pomocí objektu *MuPDFCore* a vždy otevírá právě jedno PDF. Proto je nutné po použití tento objekt uzavřít pomocí metody *close()*. Další vybrané funkce metod, které obsahuje, jsou:

1. **Zjištění rozměrů první stránky PDF** - *getDimensionsOfCurrentPage()*

2. **Zjištění počtu stránek** - *getPageCount()*

3. **Vygenerování *Bitmap* objektu** - *getBitmap(int, int, int, boolean)*

4. **Získání SVG kódu** - *getSVG(int, int, int, boolean)*

Stejné funkce tento objekt nabízí pomocí rozhraní *IImageConverter*, které implementuje. Tento objekt je použitý například v objektu *ExtendedMuPDF*, který využívá správce úkolů *ExtendedTeXMF*, to mu umožňuje běžet asynchronně.

4.3.4 Textové pole se zvýrazňováním syntaxe LaTeXového kódu

Textové pole je reprezentované objektem *CodeText*, který rozšiřuje *EditText*. Využívá objekt *LaTeXSpannableStringBuilder*, který rozšiřuje *SpannableStringBuilder* a je použitý tak, jak je popsáno v sekci 2.9. *CodeText* implementuje rozhraní *ICodeText*, které zpřístupňuje několik metod:

1. **Nastavení textu** - *setContent(CharSequence)*
2. **Získání textu** - *getText()*
3. **Začátek aktuálního výběru** - *getSelectionStart()*
4. **Konec aktuálního výběru** - *getSelectionEnd()*
5. **Odchytávání změn textu** - Pokud plugin potřebuje zachytávat změny uživatele v textu, zavolá metodu *setOnTextChangedListener(OnTextChangedListener)*. Pomocí objektu implementujícího *OnTextChangedListener* bude získávat informace nejen o změnách textu, ale i o případech, kdy uživatel změní některý z LaTeXových kontrolních maker, to jsou taková, která vytváří sekce v dokumentu. Nebo v případě, když uživatel klikne na některé z definovaných LaTeXových prostředí.
6. **Přidání LaTeXových balíků a maker do textu** - Pomocí metody *addPackages(String[])*, lze správně umístit (v případě, že tam ještě nejsou) do preambule LaTeXového dokumentu makra pro vložení LaTeXových balíků. Lze také stejně správně umístit i uživatelem definované LaTeXové makra, ty se přidávají metodou *addNewCommands(String[])*
7. **Informace o kontrolních makrech** - *ICodeText* rozšiřuje rozhraní *SectionsAdapter*, které dokáže zjistit počet kontrolních maker pomocí metody *getSectionCount()*, zjistit informace o daném kontrolním makru pomocí *getSectionAt(int)* a přejít na určitou pozici v objektu, který toto rozhraní implementuje, pomocí metody *gotoPosition(int)*.

Další objekt, který implementuje rozhraní *ICodeText*, je *ScrollCodeText*. Tento objekt rozšiřuje *ScrollView* a obsahuje *CodeText*. Důvod pro použití tohoto objektu místo přímého použití *CodeText* je vylepšené posouvání textu.

4.3.5 Editor LaTeXu

Editor LaTeXu i přesto, že je obsažený v hlavní aplikaci, využívá skoro výhradně pro svoji činnost pouze rozhraní dostupných pro implementaci pluginu tak, jak jsou popsány v sekci 4.4. Rozhraní *EditorPlugin* je implementováno v třídě *LaTeXEditorPlugin* a UI má definováno ve fragmentu *DocumentFragment*. Na hlavní fragment *DocumentFragment* jsou napojené dva další podfragmenty pomocí objektu *ViewPager*:

1. *DocumentEditorFragment* - Jediné *View* v tomto objektu je objekt *ScrollCodeText*, reprezentovaný rozhraním *ICodeText*.

2. *DocumentTypesetterFragment* - Tento fragment se stará o zobrazení výpisů z TeX Live, zobrazuje stav kompilace aktuálního dokumentu a informace o tom, jestli se instaluje nějaký dodatečný LaTeXový balík.

Další část tohoto pluginu, kterou vytváří *DocumentFragment*, je menu, které vyjíždí z levého boku obrazovky, menu je vytvořené pomocí knihovny SlidingMenu, více o této knihovně je v sekci 4.2. V tomto menu jsou uchovávány informace o kontrolních makrech nalezených v objektu, které implementuje *ICodeText* ve fragmentu *DocumentEditorFragment*. Pomocí metody *gotoPosition(int)* v *ICodeText* se po kliknutí na dané kontrolní makro přesune kurzor na správné místo v textu.

4.4 Implementace pluginu

Způsob načítání pluginu byl již popsán v sekci 2.8. Jediný plugin, který není třeba načítat, je Editor LaTeXu, protože ten je již součástí hlavní aplikace. V této sekci popíšeme, jak správně implementovat plugin.

4.4.1 Použití Android knihovny LaTeXEditorLibrary

Tato knihovna, obsahuje všechna rozhraní, pomocí kterých mezi sebou komunikuje plugin a hlavní aplikace. Jsou tu dva druhy rozhraní: Ta která by měl implementovat plugin a ta která by měla implementovat hlavní aplikace. Popišme nejprve rozhraní, která by měl implementovat plugin:

1. *EditorPlugin* - Toto rozhraní musí být implementováno vždy, pokud plugin chce aby ho hlavní aplikace rozeznala. Při použití tohoto rozhraní by měl vývojář implementovat tyto vlastnosti:

   (a) **Jméno pluginu** - Jméno, které bude zobrazeno společně s ikonou tohoto pluginu, je získáno z metody *getName()*.

   (b) **Ikony pluginu** - Jsou dva druhy ikon, které by měl vývojář přidat. První je ikona samotného pluginu, nastavená pomocí metody *getIcon()*. Druhá je ikona, která bude zobrazena pro dokument vytvořený pomocí tohoto pluginu, nastavená pomocí metody *getDocumentIcon()*.

   (c) **Přípony souborů** - Seznam přípon souborů. Plugin by měl být schopný otevřít obsah souborů s takovými příponami, které definuje. První přípona v seznamu bude použita pro vytváření nových dokumentů. Přípony souborů se definují v *getExtensions()*.

   (d) **Verze použitého rozhraní** - Pokud to bude jiná verze, než používá hlavní aplikace, plugin nebude použit.

   (e) **Fragment reprezentující UI pluginu** - Každý plugin se musí starat o svoje vlastní UI, a to pomocí fragmentu, který dědí od *AbstractPluginFragment*. Pro správné předání fragmentu je nutné přepsat metodu *getFragmentClass()*.

   (f) **Seznam prostředí** - Pokud je plugin schopný editovat obsah různých LaTeXových prostředí, měl by tato prostředí definovat v seznamu,

který je dostupný pomocí metody *getPluginBlocks()*. Každé prostředí je reprezentováno pomocí objektu, který implementuje prostředí *PluginBlock*.

(g) **Další** - Několik dalších nepovinných vlastností, které lze definovat, jsou výchozí jméno nově vytvořeného dokumentu, mime-type vytvořeného souboru a stránky nápovědy. Další povinné vlastnosti jsou již definované v abstraktní třídě *EditorPluginDefault* a týkají se hlavně načítání grafických a jiných zdrojů pluginu.

2. *AbstractPluginFragment* - Toto je ve skutečnosti abstraktní třída, která dědí od třídy *Fragment* a obsahuje několik abstraktních metod, jež je potřeba implementovat. Je nutné, aby hostitelská *Activity* tohoto fragmentu implementovala rozhraní *FragmentPluginListener*, pomocí kterého bude fragmentu poskytovat některé nezbytné služby. Třída, která rozšiřuje *AbstractPluginFragment*, by měla zvládat alespoň tyto činnosti:

   (a) **Přijmout a odeslat svůj obsah** - Tento fragment má za úkol editovat obsah, a tedy ho musí nějak načíst. To se děje pomocí metody *setContent(String)*. Také musí počítat s tím, že kdykoliv si od něj může hlavní aplikace vyžádat aktuální obsah pomocí metody *getContent()*.

   (b) **Mít obsah, který kompilovatelný jako LaTeXový kód** - Zatímco normální obsah může být úplně cokoliv, tak v případě, že chce tato aplikace sdílet svůj obsah v různých výstupech (například v PDF), musí také implementovat metodu *getLaTeXContent()*. Pokud ale je její normální obsah stejný jako LaTeX výstup z *getLaTeXContent()*, měla by metoda *isContentAsLaTeX()* vracet *true*.

   (c) **Použité LaTeXové balíky a makra** - Je-li vyžádán obsah aktuálního dokumentu, mohou být také vyžádány balíky a uživatelsky definovaná makra, které jsou nutné ke správnému skompilování dokumentu. Balíky jsou vyžadovány pomocí metody *getRequiredPackages()* a makra pomocí *getRequiredNewCommands()*.

   (d) **Další** - V případě, že si plugin vyžádá vložení dat, měl by také implementovat metody *insertBlock(PluginBlock)*, *insertData(String, InsertType)*, *insertDataAtSelection(String,InsertType)*, *replaceLastData(String)*, *addRequiredNewCommands(String[])*, *addRequiredPackages(String[])*. Tyto metody jsou implementovány v Editoru LaTeXu pro vkládání dat z jiných pluginů.

3. *PluginBlock* - Objekt třídy implementující toto rozhraní představuje jedno LaTeXové prostředí.

Hlavní aplikace implementuje tato rozhraní, která zpřístupní její funkce pluginu.

1. *PluginFragmentListener* - Toto rozhraní implementuje *Activity*, která je hostitelská pro fragment pluginu, který rozšiřuje *AbstractPluginFragment*. Služby, které poskytuje pluginu, jsou následující:

   (a) **Vytvoření textového pole s automatickým zvýrazňováním syntaxe LaTeXu** - Textové pole je pokaždé získáno jako rozhraní *ICo-*

deText. Jsou tu, ale dvě varianty textového pole s automatickým zvýrazňováním syntaxe LaTeXu. První je pouze *EditText*, získává se pomocí *getCodeText(Context, CharSequence, List<PluginBlock>)* a druhý je *EditText* zabalený ve *ScrollView*, získává se pomocí *getScrollCodeText(Context, CharSequence, List<PluginBlock>)*. Druhá varianta má díky *ScrollView* vylepšené posouvání textu.

(b) **Konvertor pro PDF do různých obrázkových formátů** - Zde existují dvě různé varianty. Buď se využije metody *getTask(FragmentPlugin, ImageType, int, int, ImageResize, int, boolean, String)*, která vrací objekt implementující rozhraní *Task*, který provádí konverzi asynchronně. Nebo se použije metoda *getImageConverter(String path)*, která vrací objekt implementující *IImageConverter*.

(c) **Kompilace LaTeXového kódu s PDF výstupem** - V případě, že plugin chce vytvořit PDF z LaTeXového kódu, tak to provede voláním metody *createTexTask(FragmentPlugin, String,String)*, kde třetí parametr by měl udávat absolutní cestu k souboru s LaTeXovým kódem.

(d) **Otevření jiného dokumentu** - V případě, že plugin chce otevřít nový dokument, měl by zavolat metodu *openDocument(String)*. Podle přípony souboru se rozhodne, jaký plugin bude použit pro zpracování či úpravu tohoto dokumentu. Před otevřením nového dokumentu bude uživatel vyzván k uložení toho starého. Dokument se otevírá v téže instanci *PluginActivity*, v níž byl otevřen i ten předchozí.

(e) **Editování obsahu v jiné instanci *PluginActivity*** - Pokud plugin chce editovat část svého obsahu v jiném pluginu, má možnost zavolat metodu *openData(EditorPlugin, String)*. Jakmile uživatel potvrdí otevření, tak se otevře nová omezená instance *PluginActivity*.

(f) **Vyžádání vložení jiného souboru** - Pokud by chtěl plugin vložit do sebe obsah z jiného souboru (ať už napřímo nebo odkazem), tak by měl použít metodu *requestInsert()* o všechno ostatní už se postará *PluginActivity*. Toto využívá Editor LaTeXu.

2. *ICodeText* - Objekty využívající toto rozhraní jsou popsány v sekci 4.3.4.

3. *SectionsAdapter* - Tímto rozhraním je rozšířen *ICodeText*. Jeho účel je předávat informace o LaTeXových makrech, které vytváří nové sekce v dokumentu. To jsou například `\chapter{..}` nebo `\section{..}`.

4. *IImageConverter* - Objekt implementující toto rozhraní je popsán v sekci 4.3.3.

5. *Task* - Reprezentuje operace, které lze provést synchronně (blokovaně) metodou *syncStart()*, nebo asynchronně metodou *start()*. Objekt implementující toto rozhraní je například objekt vrácený metodou *PluginFragmentListener#createTexTask(FragmentPlugin, String, String)*.

4.4.2 Úprava Android manifestu v pluginu

Pokud již máme vytvořenou třídu implementující *EditorPlugin*. Je nyní nutné zajistit, aby hlavní aplikace byla schopná aplikaci pluginu nalézt. Nalezení pro-

bíhá pomocí služby *BroadcastReceiver*, to je popsáno v sekci 2.8. Pro použití *BroadcastReceiver* musíme do Android manifestu přidat záznam, jehož ukázku najdete 4.1.

```
<manifest ..>
        ..
        <application ..>
                ..
                <receiver
                        android:name="full_class_name"
                        android:enabled="true">
                        <intent-filter>
                                <action android:name="cz.talos.latexeditor.
                                    ↪ library. interfaces .EditorPlugin" />
                        </intent-filter>
                </receiver>
                ..
        </application>
        ..
</manifest>
```

Kód 4.1: V kódu je zobrazeno, kam se má umístit uzel receiver v Android manifestu. Namísto řetězce „full_class_name“ se umístí plné jméno třídy, která implementuje *EditorPlugin*.

4.4.3 Kompilace pluginu

Jak je naznačeno v sekci 2.8, tak je nutné zařídit, aby výsledný DEX soubor neobsahoval žádné třídy reprezentující rozhraní. To lze provést pomocí Gradle, což je automatizační nástroj, který je použit pro sestavení aplikace. Tento nástroj se konfiguruje pomocí souboru build.gradle. Ukázka je umístěna v kódu 4.2. Pro automatickou instalaci a spuštění aplikace je nutné mít nainstalovaný program adb[1] ve svém počítači, ten je součástí Android SDK.

```
1  apply plugin: 'android'
2
3  android {
4      compileSdkVersion 20
5      buildToolsVersion "19.1.0"
6      defaultConfig {
7          applicationId "cz.talos.latexeditor.equationeditor"
8          minSdkVersion 9
9          targetSdkVersion 20
10     }
11     signingConfigs {
12         release {
13             storeFile   file ('../latex_editor.keystore')
14             storePassword '***'
```

[1] Android Debug Bridge

```
15              keyAlias 'latexeditor'
16              keyPassword '***'
17          }
18      }
19      buildTypes {
20          release {
21              runProguard false
22              proguardFiles getDefaultProguardFile('proguard-android.txt'), '
                    ↪ proguard-rules.txt'
23              signingConfig signingConfigs.release
24          }
25          applicationVariants.all { variant ->
26              variant.install.doLast {
27                  def resultStop = exec {
28                      executable = 'adb'
29                      args = ['shell', 'am', 'force-stop', 'cz.talos.
                            ↪ latexeditor']
30                  }
31                  def result = exec {
32                      executable = 'adb'
33                      args = ['shell', 'am', 'start', '-c', 'android.intent.
                            ↪ category.LAUNCHER', '-n', 'cz.talos.latexeditor
                            ↪ /cz.talos.latexeditor.activities.MainActivity']
34                  }
35              }
36              def libraryFiles = new ArrayList<File>()
37              variant.dex.libraries.each {
38                  File file ->
39                      if (file.absolutePath.contains("colorpickerview"))
40                          libraryFiles.add(file);
41              }
42              variant.dex.libraries = libraryFiles
43          }
44      }
45
46
47  }
48  repositories {
49      flatDir {
50          dirs 'libs'
51      }
52  }
53  dependencies {
54      compile files('colorpickerview.jar')
55      compile(name:'holoEverywhereLibrary', ext:'aar')
56      compile project(':LaTeXEditorLibrary')
57  }
```

Kód 4.2: Ukázka Gradle souboru pro sestavení Editoru rovnic, lze ji použít jako podklad pro jakýkoliv jiný plugin hlavní aplikace. Odstranění všech konfliktních knihoven probíhá na řádku 42, kde se předává nový profiltrovaný seznam knihoven. Protože samotný plugin nelze pustit, tak pro automatické spuštění hlavní aplikace jsou umístěné příkazy na řádcích 27-30 (pro zastavení aplikace) a 31-34 (spuštění aplikace). Řádek 54 není důležitý pro všechny pluginy, je to pouze knihovna, která je specifická pro Editor rovnic.

Poté již stačí otevřít složku se souborem build.gradle a provést příkaz `gradle zipAlignDebug` pro vytvoření APK ve složce build/outputs/apk. Popřípadě lze využít příkaz `gradle installDebug`, který sestaví nové APK a rovnou ho nainstaluje v ladící verzi do zařízení připojeného k počítači nebo do emulátoru. Pro případ neladící verze aplikace aplikace nahradíme `Debug` pomocí `Release`. Proto, aby se `release` verze skompilovala, je nutné plugin správně podepsat certifikátem. Ukázka nastavení podpisu certifikátem je v kódu 4.2 na řádcích 11-18. Jak vytvořit certifikát, je popsáno v Android Tools [10] v článku „Signing Your Applications“.

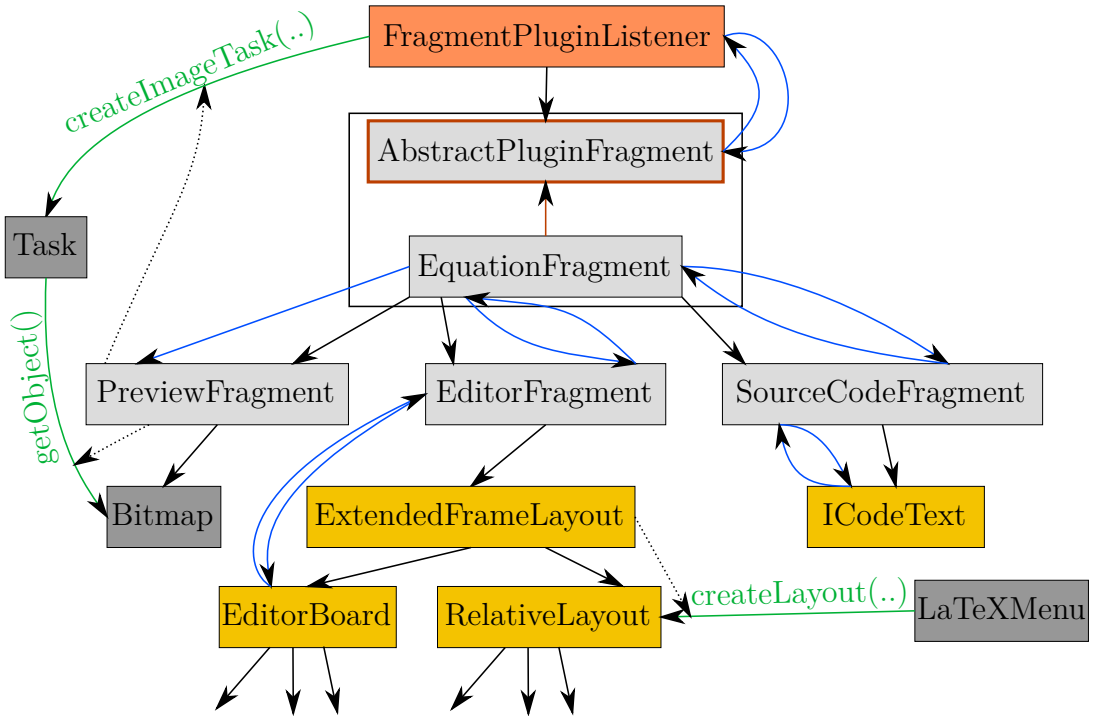
4.5 Editor rovnic

Jako plugin, Editor rovnic implementuje rozhraní *EditorPlugin* ve třídě *EquationEditorPlugin*. UI je definováno ve fragmentu *EquationFragment*. Na obrázku 4.4 jsou znázorněné závislosti jednotlivých objektů. Na hlavní fragment *EquationFragment* jsou napojené tři další. Tyto tři jsou napojené na *EquationFragment* pomocí objektu *ViewPager*, která se stará o správné zobrazení těchto podfragmentů a o přepínání jejich zobrazení. Každý z těchto fragmentů také potřebuje získávat obsah aktuálního dokumentu. O správné předávání obsahu se stará *EquationFragment*. Detekuje změny v obsahu a pokud se obsah nezměnil, tak ho neaktualizuje u objektů u kterých to není nutné. Textový obsah v Editoru rovnic představuje všechen kód uvnitř LaTeXového prostředí pro matematiku.

1. *PreviewFragment* - Tento fragment pouze přijímá obsah a zobrazuje jeho náhled. Vždy, když je zobrazen a detekuje změněný obsah, tak zadá požadavek na získání nového objektu *Bitmap*, pomocí něhož zobrazí náhled rovnice vysázené pomocí LaTeXu.

2. *EditorFragment* - Tento fragment přijímá i odesílá změněný obsah. Zobrazuje rovnici v „editačním módu“, který byl zmíněn v sekci 2.4. Je to nejvíce komplikovaná část Editoru rovnic a proto je rozebrána podrobněji v sekci 4.5.1.

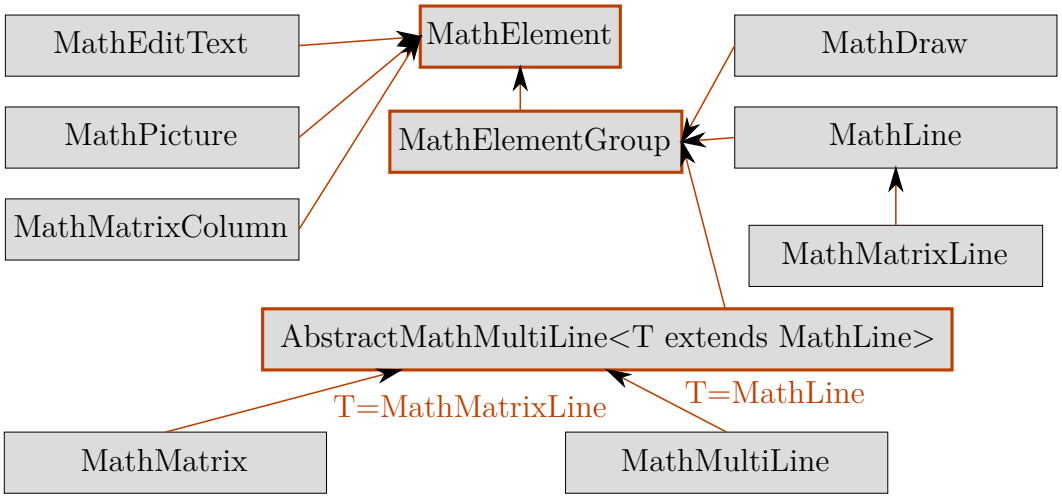
3. *SourceCodeFragment* - Tento fragment přijímá obsah a odesílá změněný obsah. Obsah edituje pomocí textového pole s LaTeXovým kódem. Výhodou je, že tento fragment nemusí nějak konvertovat získaný kód, edituje přímo obsah. Pouze zvýrazňuje syntaxi, o což se stará objekt, který implementuje *ICodeText*.



Obrázek 4.4: Obrázek ukazuje zjednodušenou strukturu objektů Editoru rovnic. Červený objekt je *Activity*, světle šedé objekty jsou fragmenty. Žluté objekty jsou vždy děděné od třídy *View* a tmavě šedé jsou blíže nespecifikované objekty. Černé šipky znázorňují zobrazení jedné UI komponenty uvnitř druhé. Modré šipky značí přenos obsahu dokumentu, zelené zavolání metody z objektu a ukazují na objekt, který byl vytvořen, přerušované šipky vždy ukazují na zelené šipky z objektů, který volal danou metodu, a červené šipky znázorňují dědičnost.

4.5.1 „Editační mód“ editoru rovnic

Jak již bylo zmíněno na začátku sekce 4.5, interaktivní editace rovnice probíhá ve fragmentu *EditorFragment*. Jednotlivé elementy rovnice jsou vkládány do objektu *EditorBoard*, který dědí od *ViewGroup* a implementuje rozhraní *IBoard*. V sekci 2.4, je ukázáno, že Editor rovnic má hierarchickou strukturu, která se stará o to, aby jednotlivé objekty *View* byly správně vykresleny. Tato hierarchická struktura má několik typů objektů, jež jsou zobrazeny na obrázku 4.5.



Obrázek 4.5: Dědičnost tříd, které jsou použity pro vnitřní reprezentaci rovnice.

Každý objekt, pomocí kterého se sestavují rovnice, má trochu jiné vlastnosti. Skoro všechny objekty lze rozdělit na dvě kategorie. V první kategorii jsou ty objekty, které v sobě mohou obsahovat další objekty, to jsou kontejnery a dědí od *MathElementGroup*. Do druhé kategorie patří objekty, které v sobě nemohou obsahovat další objekty a pouze vykreslují obsah. Objekt, který se nehodí ani do jedné kategorie je *MathMatrixColumn*.

1. *MathElement* - Je abstraktní třída, od které dědí všechny objekty, které představují matematické elementy. Kromě definování rozhraní pro aktualizaci elementu, vyexportování elementu jako LaTeXového kódu a dalších funkcí, se tato třída stará o správné umístění horního a dolního indexu a limity. Jejich umístění je buď klasicky za elementem, nebo pokud se nastaví, aby se zobrazovaly jako limity, tak se tyto indexy zobrazí nad a pod elementem. Další důležitá vlastnost je udržování souřadnic a velikosti elementu. Velikost elementu se udržuje jako hodnota v pixelech udávající, o kolik element přesahuje nad řádkem, tuto velikost vrací metoda *getOverTop()*, a o kolik element přesahuje pod řádkem, to vrací metoda *getOverBottom()*. Jsou zde ještě velikosti definované nad těmito velikostmi, to jsou *getOuterTop()* a *getOuterBottom()*, ty jsou použity v případě, když element obsahuje indexy nebo limity, které přesahují řádek a samotný vnitřní objekt už také řádek přesahuje, tak tyto hodnoty udávají o kolik přesahují indexy nebo limity. Každý *MathElement* si také udržuje svoje nejbližší sousedy, má-li nějaké (to znamená, že k sousedům lze takto přistupovat jako ke spojovému seznamu) a každý element (kromě kořenového) také zná svého rodiče. Každá třída dědící od *MathElement* má předefinovanou metodu *toString()*, pomocí které vrací obsah ve formě LaTeXového kódu.

2. *MathElementGroup* - Toto je abstraktní třída, která rozšiřuje třídu *MathElement*. Tato třída rozšiřuje vlastnosti třídy *MathElement* o možnost obsahovat v sobě další objekty. Umožňuje tedy objektu chovat se jako kontejner.

3. *MathEditText* - Objekty této třídy jsou využívány jako editovatelné textové pole. Obsahují v sobě objekt *EditText*, který poté jen správně umísťují.

Existují dva druhy objektu *MathEditText*, první vykresluje matematický text a druhý normální. Pokud jde o normální text metoda *isNormal()* vrátí *true*.

4. *MathPicture* - Objekty této třídy jsou využívány pro zobrazení *Bitmap* objektů, které byly vygenerovány z LaTeXového kódu. Objekty *Bitmap* jsou vykreslovány pomocí upraveného objektu *View*. Objekt *MathPicture* je vždy uvnitř objektu *MathDraw*.

5. *MathLine* - Toto je kontejner, který řadí svoje objekty horizontálně vedle sebe. Je to velmi důležitý objekt, protože každý objekt, který vykresluje nějaký matematický element, musí být umístěn na nějakém logickém řádku tohoto typu.

6. *MathDraw* - Objekt této třídy je kontejner, jehož objekty jsou konfigurovatelné (poloha, velikost,...). Také dokáže vykreslovat křivky ve speciálně upraveném objektu *View*.

7. *AbstractMathMultiLine<T>* - Toto je generická abstraktní třída, která tvoří kontejner, který řadí svoje objekty pod sebe. Třídu objektů tohoto kontejneru určuje generický parametr *T*, který je vždy potomkem třídy *MathLine*.

8. *MathMultiLine* - Toto je výchozí rozšíření třídy *AbstractMathMultiLine<MathLine>*. Objekty této třídy tedy tvoří více-řádkové prostředí. Přidává možnost zarovnávat řádky.

9. *MathMatrixLine* - Rozšíření třídy *MathLine*, do nějž je možné vkládat jen prvky typu *MathMatrixColumn*. Tyto prvky, ale ve skutečnosti spravuje rodič *MathMatrixLine* a to je vždy *MathMatrix*. Každý objekt *MathMatrixLine* v jednom objektu *MathMatrix* má vždy stejné prvky, jinými slovy je sdílí. Jediné rozlišení je, že si každý *MathMatrixLine* pamatuje číslo svého řádku a s ním ke svým prvkům přistupuje.

10. *MathMatrixColumn* - Objekt této třídy sice není kontejner, přestože spravuje více objektů. Tyto objekty se nachází v logickém sloupci (vždy jde o objekty *MathLine*, představující buňky matice), ale zároveň se nehlásí jako jejich rodič, místo toho vytváří každému umělého rodiče tak, aby se příslušný objekt odkázal do správného řádku. Dále objekt třídy *MathMatrixColumn* předává nadřazenému objektu *MathMatrixLine* šířku sloupce, ale místo výšky sloupce předává výšku buňky, na kterou odkazuje daný řádek *MathMatrixLine*.

11. *MathMatrix* - Toto je rozšíření třídy *AbstractMathMultiLine<MathMatrixLine>*. Tato třída tvoří maticové prostředí. Potomci objektu této třídy jsou objekty (řádky) *MathMatrixLine*, ale objekt *MathMatrix* také udržuje další sadu potomků a to jsou objekty *MathMatrixColumn*, které představují sloupce. To znamená, že je přidává a maže, protože řádky (*MathMatrixLine*) nemohou mazat sloupce, i přestože to jsou jejich potomci, protože je sdílí mezi sebou a nebylo by jasné, kdo a kdy je má mazat.

Pro vytvoření struktury těchto objektů z LaTeXového kódu. Je nutné provést jeho analýzu. Ta se provádí v objektu *LaTeXParser* pomocí metody *parseLaTeXIntoLineBehindObject(..)*. Nejdříve se provede první analýza, tak jak je popsána v sekci 2.5 pomocí metody *parseLaTeXIntoObject(String)* a na výsledek z této metody se provede metoda *prepareUI(..)*, která přímo přidá objekty dědící od *MathElement* do aktuální rovnice.

Označování elementů

O označování elementů se stará objekt *ObjectSelector*. Ten si vytváří dvě táhla (jako objekty *View*) jak je ukázáno v sekci 2.6.1 a ta umístí do *EditorBoard*. Poté při zavolání metody *startSelection(MathElement)*, vytvoří označení pro daný element. Pro vykreslení označení v *EditorBoard* by se měla volat metoda *draw(Canvas)*. Pro obnovení aktuální pozice táhel v případě posunutí rovnice či změně velikosti obrazovky, by se měla zavolat metoda *layout()*.

ObjectSelector také implementuje rozhraní *ActionMode.Callback*, které konfiguruje kontextové menu pro Action Bar. Toto menu se zobrazí pokaždé, když uživatel označí alespoň jeden element. V tomto menu jsou možnosti kopírovat výběr do schránky, vystřihnout výběr, vložit obsah ze schránky, označit rodičovský element a obarvit výběr. Operace kopírování, vkládání a vystřihnutí probíhají pomocí stejných funkcí, jako když se obsah přesouvá mezi fragmenty *EditorFragment* a *SourceCodeFragment*. Pro výběr barvy při obarvování elementů se používá *ColorDialogFragment*, který využívá knihovny ColorPickerView (více o této knihovně v sekci 4.2) pro zobrazení barevné palety.

Vykreslování rovnice

Tato sekce se zabývá tím, jakým způsobem se vnitřní reprezentace rovnice aktualizuje při změně rovnice tak, aby správně vykreslila objekty *View*, o které se stará. Jak už bylo zmíněno v sekci 2.4, jsou dva způsoby aktualizace. U obou způsobů jsou dvě fáze měřící a rozvrhovací. Měřící fáze má za úkol změřit velikost elementu a rozvrhovací fáze má za úkol určit správně pozici svých potomků a objektů *View*, pokud nějaké obsahuje. Každý element by měl být nejdříve změřen než bude rozvržen.

Shora dolů - Tato aktualizace se používá, pokud chceme aktualizovat element a všechny jeho potomky (přímé i nepřímé), dá se tedy říci, že to je aktualizace celého podstromu, kterého je daný element kořenem. Měřící fáze se spouští pomocí metody *measure(boolean recursive, MathElement child)*. Pokud už víme, že všichni potomci mají správnou velikost, nastavíme parametr *recursive* na *false*, aby se dále nepřeměřovali. Pokud jen jeden element má správnou velikost, tak referenci na tento element předáme v parametru *child* a on se nepřeměří, jinak může být *null*. Rozvrhování probíhá pomocí metody *layout(int left, int top, int right, int bottom, boolean recursive, MathElement child)*, kde první čtyři parametry vymezují hraniční obdélník, do kterého se může tento element vykreslovat. Další dva parametry mají obdobný význam jako u měřící fáze. V případě, že se nerozvrhuje celý element, volá se jen metoda *setBounds(int left, int top, int right, int bottom)* pro nastavení hraničního obdelníku. Toto jsou jen doporučení, jak by

se tyto fáze měly řešit, protože každá třída, která rozšiřuje třídu *MathElement* si implementuje tyto fáze po svém.

Zdola nahoru - Tímto způsobem se aktualizace provádí, pokud změna velikosti elementu ovlivní i jeho předky a chceme provést úspornou aktualizaci bez toho, abychom museli aktualizovat celou rovnici. V kódu 4.3, je metoda *invalidateFromOrigin(boolean)*, která se volá v případě, že se nějaký element změnil. Počínaje elementem, na kterém je tato metoda volána, se nejdříve provede měření stejné jako u aktualizace shora dolů a pokud se vyhodnotí, že se velikost změnila, tak se zavolá rodičovský element metodou *invalidateFrom(MathElement)*. Tato metoda, již používá optimalizované měření metodou *measureChild(MathElement)*, které využívá toho, že ví že se změnil pouze jeden element. Například u *MathLine* to znamená, pokud se změnila šířka jednoho z elementů, tak se jen změna šířky přičte k celkové šířce logického řádku bez toho, aby bylo nutno přeměřovat a sčítat šířky všech elementů. Poté se pokračuje stejným typem měření až do elementu, který nezměnil velikost, nebo do kořene. Poté nastává rozvrhovací fáze, pro kterou jsou zde dvě možnosti. V případě, že se změnila pozice elementu, tak je nutné změnit pozice i všem podelementům, takže se použije stejná metoda jako u aktualizace shora dolů, jen se jí nastaví, aby přeskakovala zdrojový element, jinak bychom zbytečně rozvrhovali stejný element několikrát. V druhém případě, když se nezměnila pozice, lze využít optimalizované metody *layoutChild(MathElement, boolean)*, kterou má každý element implementovanou podle svých potřeb. Metoda *layoutChild(..)* by neměla znovu volat rekurzivně rozvržení na zdrojový element a měla by mu pouze mu nastavit nový hraniční obdélník.

```
1  public void invalidateFromOrigin(boolean notDirty) {
2      if(measure(true)){
3          ParentInterface parent = getParent();
4          if (parent != null) {
5              parent.invalidateFrom(this);
6          } else {
7              requestParentLayout();
8          }
9      }
10     layout(true, null);
11     if (!notDirty)
12         makeDirty();
13 }
14 public void invalidateFrom(MathElement mathElement) {
15     int left = getLeft();
16     int top = getTop();
17     if (measureChild(mathElement)) {
18         ParentInterface parent = getParent();
19         if (parent != null) {
20             parent.invalidateFrom(this);
21         } else {
22             requestParentLayout();
23         }
24     }
```

```
25      if( left !=getLeft()  ||  top!=getTop())
26          layout(true, mathElement);
27      else
28          layoutChild(mathElement, true);
29  }
```

Kód 4.3: Ukázka metod, která se používají pro aktualizaci zdola nahoru.

Definice matematických elementů a menu

V sekci 2.3 je popsán princip definování matematických elementů, v této sekci si popíšeme formát souboru, v němž je uložena jejich definice. Elementy jsou definovány ve stejném XML souboru jako je definovaná struktura menu.

Základní struktura bez definic matematických elementů je zobrazena v kódu 4.4, jedná se tedy především o definici struktury menu, kde uzly `item` značí novou položku v menu a uzly `subitem` vytváří podmenu a jsou vždy umístěné v nějakém uzlu `item`. Pokud nechci podmenu, použiji uzel `direct`, který přiřadí vytvoření elementu přímo hlavní položce v menu. V každé položce menu a podmenu, lze nastavit vzhled pomocí různých atributů.

1. `source` - Do tohoto atributu se vkládá LaTeXový kód z matematického prostředí. Například pro vygenerování odmocniny lze vložit `source="\sqrt"`. Pokud je tento atribut nastavený, je zbytečné definovat atribut `unicode`.

2. `name` - Tento atribut definuje jméno souboru, do kterého se ukládá vygenerovaný obrázek pomocí atributu `source`.

3. `unicode` - Do tohoto atributu lze vložit libovolné unicodové znaky pomocí jejich kódu. Například pro vložení slova „text“, se musí nastavit `unicode="\u0074\u0065\u0078\u0074"` (atribut `unicode` je využit především pro speciální znaky nebo pro popis položek v menu). Pokud je již definovaný atribut `source`, tento atribut nebude použitý.

4. `size` - Tento atribut se vkládá pro nastavení velikosti textu, definovaného pomocí atributu `unicode`. Například pro 60% velikost textu vložím `size="0.6"`.

5. `template` - V tomto atributu může být umístěn identifikátor šablony. V tom případě, ale nesmí být v této položce už žádný další matematický element definován.

6. `normal` - Toto je asistence pro definici elementu, pokud některá položka, pomocí které se vkládá element má nastaveno `normal="1"`, tak oznamuje, že se bude vkládat textové pole s normálním fontem. Hlavní rozdíl mezi matematickým a normálním textovým polem spočívá v tom, že zatímco matematické textové pole je vkládáno mezi každé dva elementy tak, aby měl uživatel možnost vložit kamkoli matematický text, normální textové pole se chová jako element v tom smyslu, že i mezi dvěma normálními textovými poli je vloženo pole matematické pro případné vložení matematického textu.

Dále je vhodné si v kódu všimnout uzlů `template`, v nichž lze definovat matematický element jako šablonu. Pokud element v šabloně chci využít v menu, musím mu nastavit identifikátor pomocí atributu `id`.

Pokud některé položce v menu nastavím atribut `template` s identifikátorem šablony, bude poté tato položka vytvářet daný element. Pro konfiguraci šablon v menu lze použít uzel `addon`. V němž lze nastavit atribut `limits`, který má dvě možné hodnoty `0` a `1`. Při hodnotě `0` se zobrazí horní a dolní indexy, při hodnotě `1` se zobrazí horní a dolní limity. V tomto uzlu může být nastaveno také několik potomků.

1. `upper` - Ve výchozím zobrazení bude zobrazen horní index nebo limita.
2. `lower` - Ve výchozím zobrazení bude zobrazen dolní index nebo limita.
3. `optional` - Ve výchozím zobrazení bude zobrazen nepovinný parametr s identifikátorem nastaveným pomocí atributu `id`. To může být například u symbolu odmocniny zobrazení toho, kolikátá odmocnina to je.

```
1  <menu>
2          <template id="..">
3                  ! definice  elementu!
4          </template>
5          <item name=".." source=".." unicode="..">
6                  <subitem name=".." source=".." size=".." unicode=".."
                       ↪ normal="0|1">
7                          ! definice  elementu!
8                  </subitem>
9                  <subitem name=".." source=".." size=".." unicode=".."
                       ↪ normal="0|1" template=".."/>
10                 <subitem name=".." source=".." size=".." unicode=".."
                       ↪ normal="0|1" template=".."
11                         <addon limits="0|1">
12                                 <upper/>
13                                 <lower/>
14                                 <optional id=".." />
15                         </addon>
16                 </subitem>
17         </item>
18         <item name=".." source=".."/>
19                 <direct normal="1|0">
20                         ! definice  elementu!
21                 </direct>
22         </item>
23
24         <item name=".." source=".."/>
25                 <direct name=".." source=".." unicode=".." normal="1|0"
                       ↪ template=".."/>
26         </item>
27         <escape latex=".." unicode=".."/>
```

```
28          <item name=".." unicode="..">
29                  <subitem latex=".." unicode=".." />
30
31  <menu>
```

Kód 4.4: Ukázka struktury XML souboru pro definování menu

Definice textových elementů lze vytvořit dvěma způsoby. První je v kódu 4.4 na řádku 27 pomocí uzlu `escape`, který se nezobrazí v menu. A pro zobrazení v menu je ukázána definice na řádku 29.

Definice složitějších elementů lze také vložit na více míst, jak bylo ukázáno v kódu 4.4, na každém z těchto míst vypadá definice vždy stejně. V kódu 4.5 je ukázka definic několika různých elementů pomocí XML. Každý uzel, který představuje objekt děděný od objektu *MathElement*, může mít tyto atributy:

1. `extra` - Možné hodnoty tohoto atributu jsou `0` nebo `1`. Výchozí je `0`. Pokud je nastaveno `1`, znamená to, že je povoleno přidávat k tomuto elementu horní a dolní index nebo limitu.

2. `excludeLimits` - Možné hodnoty tohoto atributu jsou `0` nebo `1`. Výchozí není ani jedna. Pokud je nastaveno `0`, znamená to, že nebude povoleno zobrazení horního a dolního indexu. V případě, že je nastaveno `1`, tak není povoleno nastavení horní a dolní limity. Aby tento atribut měl smysl, musí být nastaveno `extra="1"`.

3. `size` - Možné hodnoty tohoto atributu jsou `fillParent` a `wrapContent`. Výchozí hodnota je `wrapContent`. Pokud je nastaveno `fillParent`, tak se použije aktuální výška logického řádku, ve kterém se tento element nachází pro nastavení výšky elementu. Pokud je na druhou stranu nastaveno `wrapContent`, tak se nastaví velikost podle velikosti potomků tohoto elementu.

4. `essential` - Tato hodnota se nenastavuje na kořenový uzel elementu, ale pouze na jeho potomky. Možné hodnoty tohoto atributu jsou `0` nebo `1`. Výchozí je `0`. Pokud je nastaveno `1`, znamená to, že tohoto potomka, nelze odstranit od hlavního elementu.

5. `outer` - Možné hodnoty tohoto atributu jsou `0` nebo `1`. Výchozí je `0`. Pokud je nastaveno `1`, znamená to, že horní a dolní index a limity budou mimo zbytek objektu. To znamená, že když se do objektu umístí jiný objekt na vertikální pozici 0, bude to tam kde by měl správně objekt začínat, jinak by to bylo tam, kde začíná horní index nebo limita.

Uzly, které vytváří objekty děděné od objektu *MathElement*, jsou tyto:

1. `canvas` - Tento uzel vytváří objekt *MathDraw*. Můžou v něm být zanořené jakékoliv jiné uzly. Pro správné určení pozice může mít každý zanořený uzel tyto atributy:

   (a) `x` - Určuje pozici na horizontální ose.

   (b) `y` - Stejné jako `x`, jen na vertikální ose.

(c) `id` - Identifikátor pro použití elementu v matematických výrazech. Pokud je nastaveno `id="base"`, bude spodek tohoto podelementu zarovnán s řádkem, na kterém je celý element umístěn.

(d) `required` - Unikátní index povinného parametru LaTeXu, který by měl být na uzlu, který ho podporuje. Typicky na logickém řádku. V kódu 4.5 je umístěn na symbolu odmocniny. Indexy by měly začínat od nuly.

(e) `optional` - Unikátní index nepovinného parametru LaTeXu, který by měl být na uzlu, který ho podporuje. Typicky na logickém řádku. V kódu 4.5 je umístěn na symbolu odmocniny. Indexy by měly začínat od nuly.

Uzel `canvas` může obsahovat uzel `path` a pomocí něho lze vykreslovat různé křivky. Uzel `path` by měl obsahovat atribut `d`. V tomto atributu je posloupnost příkazů pro vykreslení křivek. Jakýkoliv tvar vykreslený pomocí uzlu `path` nikdy nemění velikost elementu. O samotné vykreslení se stará objekt *PathDrawer*, což je zanořený objekt v objektu *MathDraw*. Každý příkaz vždy vyžaduje přesně daný počet parametrů. Každý příkaz má také přístup k bodu, kde skončilo vykreslování naposledy, přičemž se začíná v bodě [0,0]. Souřadný systém začíná vlevo nahoře. Příkazy, které jsou podporovány jsou vypsány zde:

(a) `M` - Posune aktuální bod na nové místo určené parametry. Potřebuje 2 parametry.

(b) `L` - Z aktuálního bodu vykreslí přímku do nového bodu, určeného parametry. Potřebuje 2 parametry.

(c) `Q` - Vykreslí kvadratickou Bézierovu křivku pomocí jednoho kontrolního bodu. Potřebuje 4 parametry, z toho první dva jsou pro určení kontrolního bodu a druhé dva značí konec křivky.

(d) `C` - Vykreslí kubickou Bézierovu křivku pomocí dvou kontrolních bodů. Potřebuje 6 parametrů, z toho první 4 určují kontrolní body a zbylé dva značí konec křivky.

(e) `CIRCLE` - Vykreslí kružnici zadanou jedním bodem a poloměrem. První dva parametry určují bod a třetí určuje poloměr.

Uzel `canvas`, podporuje několik atributů:

(a) `plusWidth` - K aktuálně spočítané šířce elementu se přidá tato nově nastavená hodnota.

(b) `plusHeight` - K aktuálně spočítané výšce elementu se přidá tato nově nastavená hodnota.

(c) `weight` - V případě, že element má nastavenou výšku pomocí *size = "fillParent"* a tento parametr je nastavený na více jak *0*. Vynásobí se hodnota tohoto parametru aktuální výškou pro výpočet šířky elementu.

(d) `maxWidth` - V případě, že šířka elementu je vyšší než tato hodnota, tak se šířka elementu sníží na tuto hodnotu.

(e) `base` - Určuje vertikální souřadnici, ke které bude zarovnán řádek. V případě, že je nastaven některý podelement s identifikátorem `base`, tak se zarovnání posune o hodnotu v tomto atributu.

(f) `optional` - Index jednoho z nepovinných parametrů, který má být zobrazen.

Atributům `x` a `y` v podelementech, všem parametrům pro příkazy v uzlu `path` a atributům `plusWidth`, `plusHeight` a `maxWidth` lze nastavit hodnota pomocí matematických výrazů. Jsou podporovány operátory +, -, * a / se standardním významem a prioritou, kterou lze ovlivnit pomocí závorek (a), také lze využít funkce max(_,_) pro získání vyššího čísla a min(_,_) pro získání nižšího čísla. Také lze v těchto výrazech použít tyto proměnné:

(a) `w` - Aktuální šířka elementu, nemělo by se používat při určování souřadnic podelementů (v té době není známá).

(b) `h` - Aktuální výška elementu, nemělo by se používat při určování souřadnic podelementů (v té době není známá). Pro nastavení souřadnic podelementu, lze využít pouze v případě, když používáme `size = fillParent`.

(c) `inside` - Aktuální výška řádku.

(d) `dip` - Představuje 1 dip, určený podle parametrů uživatelova displeje.

(e) `{id}.w` - Šířka sousedního podelementu s daným identifikátorem.

(f) `{id}.h` - Výška sousedního podelementu s daným identifikátorem.

O správné vyhodnocení těchto matematických výrazů se stará objekt *Calculator*, který vždy při vybrání objektu sestaví AST pomocí Shunting-yard algoritmu.

2. `line` - Vytvoří nový logický řádek pomocí objektu *MathLine*. Tento uzel může mít atribut `depth`, který má dvě možné hodnoty `0` a `1`. Výchozí hodnota je `0`. Pokud je nastaveno `1`, tak všem podřádkům tohoto řádku se nastaví vyšší hloubka, což je parametr, který ovlivňuje výšku řádku. Čím je vyšší hloubka, tím je nižší výška řádku.

3. `textbox` - Vytvoří editovatelné textové pole pomocí objektu *MathEditText*. Tento uzel může mít tyto atributy:

   (a) `face` - Mění font textového pole. Jsou možné dvě hodnoty `math` a `normal`. Výchozí hodnota je `math`.

   (b) `text` - Pokud má tento atribut hodnotu `selected`, tak se do něj vkládají aktuálně označené elementy při vytvoření tohoto elementu z menu.

4. `picture` - Vytvoří obrázek vykreslený pomocí LaTeXu a zobrazený pomocí objektu *MathPicture*. Tento uzel může mít tyto atributy:

   (a) `size` - Výška elementu bude vynásobena tímto číslem.

   (b) `under` - Část obrázku, která bude pod řádkem.

(c) `valign` - V případě, že je element menší než řádek, tak ho zarovná buď na střed hodnotou `center`, nahoru hodnotou `top`, nebo dolů hodnotou `bottom`.

5. `multiline` - Vytvoří víceřádkové prostředí pomocí objektu *MathMultiLine*. Tento uzel může mít tyto atributy:
   (a) `height` - Určuje výchozí počet řádků.
   (b) `align` - Tento atribut určuje způsob zarovnání řádků a může nabývat těchto hodnot:
      i. `left` - Pro zarovnání vlevo
      ii. `center` - Pro zarovnání na střed
      iii. `right` - Pro zarovnání vpravo
   (c) `latex` - Určuje začátek LaTeXového makra, tohoto prostředí. Bude použit před obsahem tohoto prostředí. Pokud se nevyplní atribut `latexEnd`, bude tento atribut použit jako makro. V tomto případě je nutné ještě zadefinovat atribut `required`, klidně jen prázdným řetězcem, aby se při analýze kódu rozeznalo, že makro které tento uzel definuje má mít nějaký povinný parametr.
   (d) `latexEnd` - Určuje konec LaTeXového makra, tohoto prostředí. Bude použit za obsahem tohoto maticového prostředí.
   (e) `depth` - Funguje stejně jako u uzlu `line`.

6. `matrix` - Vytvoří maticové prostředí pomocí objektu *MathMatrix*. Tento uzel může mít tyto atributy:
   (a) `width` - Určuje výchozí počet sloupců.
   (b) `height` - Určuje výchozí počet řádků.
   (c) `align` - Tento atribut určuje způsob zarovnání sloupců a může nabývat těchto hodnot:
      i. `left` - Ve výchozím režimu jsou všechny sloupce zarovnány doleva.
      ii. `right` - Ve výchozím režimu jsou všechny sloupce zarovnány doprava.
      iii. `center` - Ve výchozím režimu jsou všechny sloupce zarovnány na střed.
      iv. `align` - Ve výchozím režimu je pravé a levé zarovnání sloupců střídáno. Začíná se s pravým zarovnáním.
      v. `array` - Ve výchozím režimu je pravé, na střed a levé zarovnání sloupců střídáno. Začíná se s pravým zarovnáním.
   (d) `latex` - Stejné použití jako v uzlu `multiline`.
   (e) `latexEnd` - Stejné použití jako v uzlu `multiline`.
   (f) `optional` - Jediná možná hodnota tohoto atributu je `columnsGlobal`. Ta změní chování tak, že uživatel bude moct zarovnávat všechny sloupce najednou na jeden typ zarovnání. A zvolené zarovnání bude vypsáno jako volitelný parametr za začátek LaTeXového makra. Ve formě `l` pro zarovnání vlevo, `c` pro zarovnání na střed a `r` pro zarovnání vpravo.

(g) `required` - Jediná možná hodnota tohoto atributu je `columns`. Ta změní chování tak, že uživatel bude moct zarovnávat každý sloupec, jak bude chtít. A zvolená zarovnání budou vypsána jako povinný parametr za začátek LaTeXového makra. Například pro zarovnání prvního sloupce vpravo, druhého vlevo a třetího na střed bude použit parametr `rlc`.

Všechny elementy, definované ať jako šablona (i nepoužitá v menu) nebo přímo v menu, budou použity při sestavování rovnice z LaTeXového kódu.

```
<!-- index horní/dolní -->
<attribute type="upper|lower" />

<!-- Symbol integrálu -->
<canvas extra="1" latex="\int" outter="1" weight="0">
        <picture name="integral" id="base" size="1.3" source="\int" under
            ↪ ="0.3"/>
</canvas>

<!-- závorky přizpůsobující se obsahu -->
<line excludeLimits="1" extra="1">
        <canvas essential="1" latex="\left{" maxWidth="25*dip" size="
            ↪ fillParent" weight="0.21">
                <path d="M w 2*dip Q w*0.5 2*dip w*0.5 h*0.25 Q w
                    ↪ *0.5 h*0.5 0 h*0.5 Q w*0.5 h*0.5 w*0.5 h*0.75 Q w
                    ↪ *0.5 h-2*dip w h-2*dip" />
        </canvas>
        <textbox text="selected" />
        <canvas essential="1" latex="\right}" maxWidth="25*dip" size="
            ↪ fillParent" weight="0.21">
                <path d="M 0 2*dip Q w*0.5 2*dip w*0.5 h*0.25 Q w*0.5
                    ↪  h*0.5 w h*0.5 Q w*0.5 h*0.5 w*0.5 h*0.75 Q w*0.5
                    ↪  h-2*dip 0 h-2*dip" />
        </canvas>
</line>

<!-- více řádkové a více sloupcové LaTeXové prostředí array -->
<matrix align="array" latex="\begin{array}" latexEnd="\end{array}"
    ↪ required="columns" />

<!-- více řádkové LaTeXové prostředí substack ->
<multiline align="center" depth="1" height="2" latex="\substack" required
    ↪ ="" />

<!-- symbol odmocniny ->
<canvas extra="1" latex="\sqrt" outter="1" plusWidth="2*dip">
        <attribute excludeLimits="1" />
        <line depth="1" id="square" optional="0" y="base.h*0.5-square.h"
            ↪ >
```

```
            <textbox />
        </line>
        <path d="M 0 base.h*0.6 L base.h*0.05 base.h*0.52 L base.h*0.1
            ↪ (base.h+4*dip) L base.h*0.2 2*dip L w-square.w+2*dip 2*
            ↪ dip" x="square.w" />
        <line id="base" required="0" x="square.w+base.h*0.2" y="4*dip"
            ↪ >
            <textbox text="selected" />
        </line>
</canvas>
```

Kód 4.5: Ukázka definic několika matematických elementů.

Sestavení menu

Menu se sestavuje pomocí objektu *LaTeXMenu*, ten pomocí metody *loadData(Context,Reader)* načte XML soubor popsaný v předchozí sekci a vytvoří strukturu pomocí tříd *LaTeXMenuButton*, který vytváří podmenu, a *LaTeXButton*, který vytváří tlačítka v menu a v podmenu. Ke každému tlačítku je přiřazena definici elementu, který vytváří. Pro samotné vytvoření menu se zavolá na objekt *LaTeXMenu* metoda *createLayout(..)*.

Editování elementu

Po kliknutí na element se zobrazí jeho možnosti, podle toho jeho definice. To se týká atributů `extra` a `excludeLimits` a elementů, které jsou nastavené jako nepovinné. Všechny tyto parametry jsou popsané v sekci 4.5.1. Pomocí fragmentu *ObjectDialogFragment* se zobrazí právě všechny tyto parametry aktuálního elementu vizuálně. Použije se stejné vykreslování jako pro vykreslení rovnice jen logické řádky, které tento element vykresluje se vykreslí jako čtverce. To je proto, že narozdíl od objektu *EditorBoard*, v tomto fragmentu jsou zobrazeny elementy pomocí vlastního objektu, který implementuje rozhraní *IBoard*.

Závěr

Práce se trochu odchýlila od původního zadání tím, že byly vytvořeny dvě aplikace. Editor LaTeXu a jeho plugin Editor rovnic se instalují zvlášť, ale samostatně funguje pouze Editor LaTeXu. Už v cílech práce bylo napsáno, že jako vedlejší produkt této práce vznikne i Editor LaTeXu. Později při návrhu aplikace bylo logické řešení umístit Editor LaTeXu jako hlavní aplikaci a Editor rovnic k němu dodávat jako doplněk, který dokáže fungovat jako samostatná komponenta uvnitř hlavní aplikace i jako pomocný nástroj pro editaci matematických výrazů v Editoru LaTeXu. Hlavní aplikace byla navržena tak, aby bylo možné doplňků jako je Editor rovnic vytvořit více. I přes tyto změny práce stále splňuje zadání.

Největší nedostatky práce jsou, že oba editory postrádají funkci historie změn a tedy i jakýkoliv příkaz zpět. Dále Editor rovnic nezachovává ručně vložená neznámá LaTeXová makra a mazání elementů není příliš intuitivní. Poslední zásadní nedostatek je, že aplikace jsou podporovány pouze na zařízeních s architekturou procesoru ARM. Tyto nedostatky nebyly odstraněny pro nedostatek času.

Možná rozšíření pro Editor LaTeXu jsou napovídání známých maker při psaní kódu, tlačítka pro rychlé vložení často používaných LaTeXových znaků a správa větších projektů, ne pouze jednotlivých dokumentů. Jedním z možných rozšíření Editoru rovnic by mohlo být například rozpoznávání matematických symbolů psaných na displej zařízení. Bylo by možné přidávat i další pluginy do celé aplikace, mohlo by se například jednat o editor obrázků, tabulek, grafů nebo třeba bibliografie.

Seznam použité literatury

[1] Creative Commons. Creative commons attribution 2.5. http://creativecommons.org/licenses/by/2.5/legalcode.

[2] Apache Software Foundation. Apache license. http://www.apache.org/licenses/LICENSE-2.0.html, January 2004.

[3] Free Software Foundation. Affero general public license 3.0. http://www.gnu.org/licenses/agpl-3.0.html, November 2007.

[4] Free Software Foundation. Lesser general public license 3.0. https://www.gnu.org/licenses/lgpl.html, June 2007.

[5] Lame Android Hero. LAHpdf. http://lameandroidhero.bitbucket.org/lahpdf.html, 2014. [Online; accessed 19-June-2014].

[6] Donald Knuth. *The TeXbook*. Addison-Wesley, Reading, Mass, 1986.

[7] Leslie Lamport. *LATEX : a document preparation system : user's guide and reference manual*. Addison-Wesley Pub. Co, Reading, Mass, 1994.

[8] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. The Not So Short Introduction to LaTeX 2ε. http://tobi.oetiker.ch/lshort/lshort.pdf, 2011. [Online; accessed 19-June-2014].

[9] The Android Open Source Project. The Android Source Code | Android Developers. https://source.android.com/source/index.html, 2014. [Online; accessed 19-June-2014].

[10] The Android Open Source Project. Developer Tools | Android Developers. https://developer.android.com/tools/index.html, 2014. [Online; accessed 19-June-2014].

[11] The Android Open Source Project. Introduction to Android | Android Developers. https://developer.android.com/guide/index.html, 2014. [Online; accessed 19-June-2014].

[12] W3. Paths - svg 1.1 (second edition). http://www.w3.org/TR/SVG/paths.html, 2014. [Online; accessed 19-June-2014].

[13] Wikibooks. Latex — Wikibooks, The Free Textbook Project. http://en.wikibooks.org/w/index.php?title=LaTeX&oldid=2669245, 2014. [Online; accessed 19-June-2014].

Seznam obrázků

Seznam použitých zkratek

adb Android Debug Bridge - *Program pro komunikaci s připojenými zařízeními Android anebo běžícími emulátory systému Android*

AIDL Android Interface Definition Language - *Jazyk pro definici rozhraní mezi službou a aplikací v systému Android*

AOT Ahead-of-time - *Kompilace kódu jako C nebo různých „bytecodů“ do strojového kódu*

API Application Programming Interface - *Přístupné rozhraní (například různých knihoven) pro komunikaci se zbytkem kódu*

APK Android application package file - *Balík souborů, který se používá pro instalaci aplikací na systému Android*

ARM Advanced RISC Machine - *Architektura procesorů s nízkou spotřebou energie*

ART Android Runtime - *Způsob spouštění a instalace aplikací s využitím AOT*

ashmem Android shared memory - *Knihovna v C integrovaná v systému Android, která umožňuje sdílení paměti mezi procesy*

AST Abstract syntax tree - *Vyhodnocovací stromová struktura , která reprezentuje nějaký zdrojový kód*

DEX Dalvik EXecutable - *Kompilovaný balík kódu v systému Android*

dip Density-independent Pixels - *Abstraktní jednotka, která je definována jako 1px na displeji s hustotou 160 dpi*

dpi dots per inch - *Jednotka, která udává kolik pixelů je v jednom palci*

DVI Device Independent File Format - *Formát souboru, který je výstup programu TEX*

DVM Dalvik Virtual Machine - *Způsob spouštění a instalace aplikací s využitím JIT*

fd File descriptor - *Identifikátor, který je použit pro označení části paměti v knihovně ashmem*

IDE Integrated Development Environment - *Prostředí pro vývoj softwaru*

Indie Independent - *Označení pro nezávislé vývojáře*

JAR Java ARchive - *Souborový formát, který se využívá pro distribuci knihoven a programů zkompilovaných do „java bytecodu“*

JIT Just-in-time - *Kompilátor, který kompiluje „bytecode“ do strojového kódu až ve chvíli, kdy je to nutné*

JPEG Joint Photographic Experts Group - *Rastrový ztrátový obrázkový formát, především využívaný pro fotografie.*

JSON JavaScript Object Notation - *Univerzální formát určený pro přenos dat, která reprezentují pole a objekty.*

JVM Java Virtual Machine - *Virtuální stroj, který spouští programy napsané v „java bytecodu“*

MIPS Microprocessor without Interlocked Pipeline Stages - *Architektura procesorů bez automaticky organizované „pipeline“*

NDK Native Development Kit - *Balík nástrojů pro sestavování programů a knihoven pro systém Android ve strojovém kódu.*

OpenGL Open Graphics Library - *Multiplatformní API pro tvorbu počítačové grafiky.*

OpenGL ES OpenGL for Embedded Systems - *Verze OpenGL pro jednoúčelová a mobilní zařízení.*

PDF Portable Document Format - *Multiplatformní souborový formát pro ukládání dokumentů*

PNG Portable Network Graphics - *Bezztrátový rastrový obrázkový formát*

SDK Software Development Kit - *Balík nástrojů pro vytváření určitého softwaru.*

SVG Scalable Vector Graphics - *Vektorový formát založený na XML*

texmf TEX and METAFONT - *Adresář, kam se ukládají data z LATEXových balíků*

UI User Interface

WEBP Web Photo - *Bezztrátový i ztrátový rastrový obrázkový formát vyvinutý Googlem pro použití na webu.*

WYSIWYG What you see is what you get - *Druh editoru, který zobrazuje při editaci svoje data stejným způsobem, jako se budou zobrazovat na výstupu*

XML Extensible Markup Language

Přílohy

Příloha A

Toto je licence, kterou jsem obdržel od autorů knihoven LAHTeX, LAHSpectre a LAHDownloader pro využívání těchto knihoven.

On behalf of the author, we grant you (Jiří Marek) and only you license to use LAHTeX and LAHDownloader and other library projects LAH* of the GitHub user anhoavu for personal purpose (i.e. no distribution in source/binary form) and re-distribution of your derived work such as Android apps subject to the conditions:

[The term "our work"refers to all files originating, derived or copied in part/full from anhoavu GitHub repository.]

0) You do NOT change the authorship of our work.

1) You do NOT release any part of our work even if you make modifications to include your own code.

That said, if you have to distribute source code of your derived work, you need to exclude all files defined as our work from that distribution. Also, this implies you should extend our classes whenever possible instead of modifying them directly if code ownership is a concern to you.

2) You give proper credit to us, for e.g. in the description on Google Play or in some Help/About screen within your app. Also, please indicate that you are using an old version of our work and whether you modify it in your distribution.

3) You inform your recipient that we [LAH] are not to take any warranty, liability or responsibility for our work used in your derived product.

4) We reserve the right to decide on uncovered circumstances should they arise.

5) The usage and distribution should be limited to your existing products (LaTeX Editor and Equation Editor plug-in). Please contact us when you want to use our work in any software you distributed in the future.

6) If you intend to charge for your derived product, please do so in perfect numbers (see `http://en.wikipedia.org/wiki/Perfect_number`). [This is what he demands us, pretty weird.]

That said, the numerical representation of the charged amount without separators should spell perfect number. For example, you could charge $0.6/user as it is equivalent to $.6 notation, leaving 6 when dropping the ”.” separator. So are $.28, $2.8 and $28. You are free to choose the base currency such as Euro, GBP, etc. instead of USD.

Příloha B

Seznam podporovaných LaTeXových maker v Editoru rovnic:

1. `\acute`
2. `\arccos`
3. `\arcsin`
4. `\arctan`
5. `\bar`
6. `\begin{Bmatrix*}`
7. `\begin{Bmatrix}`
8. `\begin{Vmatrix*}`
9. `\begin{Vmatrix}`

10. \begin{aligned}
11. \begin{array}
12. \begin{bmatrix*}
13. \begin{bmatrix}
14. \begin{dcases}
15. \begin{drcases}
16. \begin{matrix*}
17. \begin{matrix}
18. \begin{pmatrix*}
19. \begin{pmatrix}
20. \begin{vmatrix*}
21. \begin{vmatrix}
22. \cdots
23. \check
24. \cos
25. \cosh
26. \cot
27. \coth
28. \csc
29. \dbinom
30. \ddddot
31. \dddot
32. \ddot
33. \ddots
34. \dot
35. \dotsb+
36. \dotsc,
37. \dotso
38. \end{Bmatrix*}
39. \end{Bmatrix}
40. \end{Vmatrix*}
41. \end{Vmatrix}
42. \end{aligned}
43. \end{array}
44. \end{bmatrix*}
45. \end{bmatrix}
46. \end{dcases}
47. \end{drcases}
48. \end{matrix*}
49. \end{matrix}
50. \end{pmatrix*}
51. \end{pmatrix}
52. \end{vmatrix*}
53. \end{vmatrix}
54. \fint
55. \frac
56. \grave
57. \hat
58. \iiint
59. \iint
60. \int
61. \landdownint
62. \landupint
63. \left(
64. \left[
65. \left\langle
66. \left\lceil
67. \left\lfloor
68. \left\{
69. \left|
70. \left|\left|
71. \lim
72. \mathring
73. \not
74. \oiint
75. \oint
76. \ointclockwise
77. \ointctrclockwise
78. \overbrace
79. \overleftarrow
80. \overline
81. \overrightarrow
82. \prod
83. \right)
84. \right\rangle
85. \right\rceil
86. \right\rfloor
87. \right]
88. \right|
89. \right|\right|
90. \right\}
91. \sec
92. \sfrac
93. \sin
94. \sinh
95. \sqiint
96. \sqint
97. \sqrt
98. \substack

99. \sum
100. \tan
101. \tanh
102. \tilde
103. \underbrace
104. \underline
105. \vdots
106. \vec
107. \widehat
108. \widetilde
109. \xLeftarrow
110. \xLeftrightarrow
111. \xRightarrow
112. \xhookleftarrow
113. \xhookrightarrow
114. \xleftarrow
115. \xleftrightarrow
116. \xmapsto
117. \xrightarrow
118. \backslash
119. \&
120. _
121. \hat{}
122. \%
123. \$
124. \{
125. \}
126. \#
127. \sim
128. \infty
129. \pm
130. \mp
131. \times
132. \div
133. \oplus
134. \ominus
135. \otimes
136. \oslash
137. \odot
138. \circ
139. \ast
140. \star
141. \dagger
142. \ddagger
143. \cdot
144. \diamond
145. \bigtriangleup
146. \bigtriangledown
147. \triangleleft
148. \triangleright
149. \bigcirc
150. \bullet
151. \wr
152. \amalg
153. \leq
154. \geq
155. \neq
156. \doteq
157. \equiv
158. \approx
159. \approxeq
160. \cong
161. \simeq
162. \propto
163. \parallel
164. \nparallel
165. \models
166. \perp
167. \vdash
168. \dashv
169. \ll
170. \gg
171. \preceq
172. \succeq
173. \prec
174. \succ
175. \smile
176. \frown
177. \gets
178. \to
179. \leftrightarrow
180. \mapsto
181. \impliedby
182. \implies
183. \iff
184. \Leftarrow
185. \Rightarrow
186. \Leftrightarrow
187. \hookleftarrow

188. \hookrightarrow
189. \uparrow
190. \Uparrow
191. \downarrow
192. \Downarrow
193. \exists
194. \nexists
195. \forall
196. \neg
197. \wedge
198. \vee
199. \in
200. \ni
201. \notin
202. \subset
203. \supset
204. \not\subset
205. \not\supset
206. \subseteq
207. \supseteq
208. \not\subseteq
209. \not\supseteq
210. \subsetneq
211. \supsetneq
212. \emptyset
213. \varnothing
214. \cap
215. \cup
216. \setminus
217. \uplus
218. \sqcap
219. \sqcup
220. \alpha
221. \beta
222. \gamma
223. \delta
224. \epsilon
225. \zeta
226. \eta
227. \theta
228. \iota
229. \kappa
230. \lambda
231. \mu
232. \nu
233. \xi
234. \omicron
235. \pi
236. \rho
237. \sigma
238. \tau
239. \upsilon
240. \phi
241. \chi
242. \psi
243. \omega
244. \varepsilon
245. \vartheta
246. \varkappa
247. \varpi
248. \varrho
249. \varsigma
250. \varphi
251. \Gamma
252. \Delta
253. \Theta
254. Lambda
255. \Xi
256. \Pi
257. \Sigma
258. \Upsilon
259. \Phi
260. \Psi
261. \Omega
262. \partial
263. \eth
264. \hbar
265. \imath
266. \jmath
267. \ell
268. \Re
269. \Im
270. \wp
271. \nabla
272. \Box
273. \aleph
274. \beth
275. \gimel
276. \mathbb{C}
277. \mathbb{N}
278. \mathbb{Q}
279. \mathbb{P}
280. \mathbb{Q}
281. \mathbb{R}
282. \mathbb{Z}

Příloha C

Obsah přiloženého DVD-ROM je následovný:

1. **Dokumentace** - Dokumentace vygenerovaná pomocí programu javadoc se nachází ve složce „documentation“.
2. **Zdrojové kódy** - Zdrojové kódy jsou umístěny v následujících složkách:
   (a) **Hlavní aplikace** - „project/LaTeXEditorApp/src/main/java“
   (b) **Editor rovnic** - „project/EquationEditor/src/main/java“
   (c) **Program používající knihovny MuPDF** - „project/mupdf/main.c“
3. **Skompilované instalační balíky**
   (a) **Hlavní aplikace** - „bin/LaTeXEditorApp-release.apk“
   (b) **Editor rovnic** - „bin/EquationEditor-release.apk“