VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

REALIZACE AUTOMATIZOVANÉHO KONVERTORU TABULEK DO LATEX
IMPLEMENTATION OF AN AUTOMATED CONVERTER OF TABLES TO LATEX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jakub Hobža**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Kamil Staněk**
SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Jakub Hobža**
Studijní program:	Strojírenství
Studijní obor:	Základy strojního inženýrství
Vedoucí práce:	**Ing. Kamil Staněk**
Akademický rok:	2021/22

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Realizace automatizovaného konvertoru tabulek do LATEX

Stručná charakteristika problematiky úkolu:

Využívání systému LATEX v odborné sféře charakterizuje nutnost implementovat do textu nejen samotný text a grafiku, ale především tabulky s údaji. Tyto tabulky však nejsou jednoduše implementovatelné, pokud se jedná o složitější struktury tabulek. Proto se mnoho autorů raději uchýlí k převodu do obrázku místo do kopírovatelné tabulky. Vyrobit kvalitní konvertor, který umožní co nejvíce automatizovaný převod tabulek do LATEX, je tedy velmi žádoucí mnohými uživateli. Příprava takového automatizovaného konvertoru tabulek do LATEX je tématem této bakalářské práce včetně ukázek převodů a postupů.

Cíle bakalářské práce:

- Analýza stávajících konvertorů a průvodců tabulek LATEX.
- Zvolení vhodného LATEX editoru pro široké využití a zdrojových tabulek.
- Zvolení vhodného programového nástroje pro tvorbu konvertoru.
- Vytvoření konvertoru tabulek LATEX s GUI rozhraním.
- Provést implementaci do vědeckého dokumentu v LATEX s ukázkami importu.
- V práci mít popis používaných LATEX příkazů s příklady použití.
- V LATEX kódu i programu používat podrobné a srozumitelné české popisy.
- Podrobná dokumentace řešení.

Seznam doporučené literatury:

LAMPORT, Leslie a Duane BIBBY. LATEX: a document preparation system: user's guide and reference manual. 2nd ed. Boston: Addison-Wesley Pub., 1994. ISBN 978-0201529838.

RYBIČKA, Jiří. LATEX pro začátečníky. 3. vyd. Brno: Konvoj, 2003. ISBN 80-7302-049-1.

KOPKA, Helmut a Patrick W. DALY. A Guide to LATEX. 4rd ed. Boston: Addison-Wesley, c2004. ISBN 978-0321173850.

MITTELBACH, Frank.l a Michel GOOSSENS. The Latex Companion. 2nd ed. Boston: Addison-Wesley, 2004. ISBN 978-0201362992.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Bakalářská práce se zabývá návrhem a implementací nástroje, který usnadní sazbu LaTeX tabulek. Na základě získaných informací je naprogramován konvertor s grafickým rozhraním, zároveň je vytvořena podrobná dokumentace řešení. Konvertor pro zadaní vstupních údajů využívá grafické rozhraní tabulkového editoru Microsoft Excel. Zadaná data extrahuje pomocí Python knihovny `openpyxl` a získaná data automaticky převede na LaTeX výstup. Do výstupu lze mj. zahrnout zarovnání textu, barvy textu i pozadí a různé typy okrajů, vše individuálně pro každou buňku tabulky. Podporovány jsou také sloučené buňky, nestandardní délky a různé orientace výstupních tabulek. Konvertor tvoří přehledný, kompaktní a komentovaný textový výstup, který je využitelný pro širokou veřejnost.

ABSTRACT

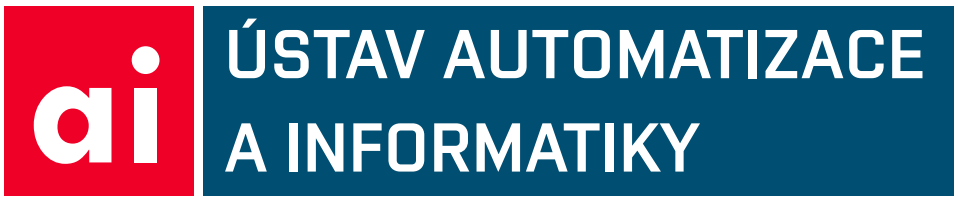
This thesis aims to create a tool to ease the typesetting of LaTeX tables. In this thesis, the tool is designed, implemented, and documented based on the gathered knowledge. The tool reads spreadsheet data from a Microsoft Excel file with the `openpyxl` library. This input data is processed into a compact and readable LaTeX output that is easy to work with for the average user. Moreover, the tool can set text alignment, foreground and background color, and borders, all individually for each cell in the output table. Merged cells and long and rotated tables are also supported.

KLÍČOVÁ SLOVA

tabulka, Excel, LaTeX, tabularray, openpyxl, GUI

KEYWORDS

table, Excel, LaTeX, tabularray, openpyxl, GUI



2022

BIBLIOGRAFICKÁ CITACE

HOBŽA, Jakub. *Realizace automatizovaného konvertoru tabulek do LaTeX*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, 111 s. Bakalářská práce. Vedoucí práce: Ing. Kamil Staněk

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato bakalářská práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 20. 5. 2021 .....

Jakub Hobža

OBSAH

1 ÚVOD

Sazba tabulek v běžných, tzv. WYSIWYG, editorech dokumentů je snadná, jelikož je v každém okamžiku k dispozici náhled výsledné tabulky. Systém LaTeX je však v tomto ohledu fundamentálně odlišný, dokumenty se zde tvoří formou textových souborů, jejichž následným překladem vznikne výsledný dokument ve formátu `.dvi` nebo `.pdf`. Uživatel běžně nemá k dispozici okamžitý náhled, ten vzniká až následující kompilací textového dokumentu. U většiny prvků LaTeX dokumentu není absence okamžitého náhledu překážkou, jelikož mají snadný zápis a dá se jednoduše představit, jak ovlivní výsledný dokument při dalším překladu.

LaTeX tabulky jsou případem prvku, který už má složitější zápis. S rostoucí komplexností struktury tabulky náročnost textového zápisu zpravidla prudce stoupá. Pro složitější struktury už se zpravidla nedá snadno představit, jak bude výstup vypadat. Řada autorů se tak uchyluje ke tvorbě tabulek pomocí jiných prostředků, zpravidla takových, které využívají grafické rozhraní. Vytvoření tabulky v grafickém rozhraní je výrazně jednodušší, ale obvykle nastává problém se zpětným importem do LaTeX dokumentu. Řada autorů tabulku importuje formou obrázku, což s sebou nese několik nevýhod. Užití tabulky formou obrázku vnáší do dokumentu nekonzistentní velikost a řez písma, nemožnost obsah tabulky kopírovat a také složitou ruční úpravu obsahu a formátu tabulky.

Cílem práce je vytvořit automatizovaný nástroj, který eliminuje tyto nevýhody. Nástroj by měl spojit výhody nativních LaTeX tabulek s jednoduchostí vstupu WYSIWYG editorů. Zároveň by nástroj měl umožnit snadné zadání vstupních údajů a široké možnosti doplňujícího formátování, v ideálním případě by také měl produkovat přehledný a srozumitelný výstup s komentáři.

V první části práce jsou zkoumány současné možnosti sazby tabulek. Na základě získaných poznatků je vytvořen výsledný konvertor tabulek.

Jádrem práce je podrobná dokumentace vytvořeného řešení. Text je psán s ohledem na budoucí řešitele podobného problému. Dokumentace obsahuje popis toho, odkud číst vstupní data, jak lze získaná data zpracovat a jak z nich lze vytvořit vhodný výstup. Závěr kapitoly obsahuje krátký návod k použití konvertoru, také obsahuje ukázku typových vstupních tabulek a jim odpovídajících výstupů konvertoru. Na základě uvedených příkladů lze velmi rychle získat přehled o možnostech využití konvertoru i pro běžného uživatele.

2 PŘEHLED SOUČASNÉHO STAVU POZNÁNÍ

V této kapitole budou rozebrány současné možnosti pro ulehčení tvorby tabulek v LaTeX, ať už pomocí nestandardních prostředí tabulek, nebo s využitím různých pomocných programů. Samozřejmostí je i definice základních pojmů.

2.1 Úvod do TeX a LaTeX

TeXje počítačový program určený pro sazbu dokumentů vysoké typografické úrovně. Při tvorbě TeX dokumentů se pracuje s čistě textovým dokumentem, který kromě samotného textu obsahuje i velmi precizní příkazy určující, jak má daný text být vysázen. Při práci s TeXem uživatel nevidí jak bude text ve výsledku vypadat, k zobrazení výsledného dokumentu je nutný překlad vstupního dokumentu. Překladem je vytvořen dokument ve formátu `.dvi`, který lze zobrazit a který je nezávislý na výstupním zařízení (tiskárně). [1]

LaTeX slouží ke zpřístupnění možností TeXu běžným uživatelům, kteří mají povědomí o tom, *co* chtějí ve svém dokumentu vysázet, ale nejsou typografičtí profesionálové, aby tyto prvky uměli vysázet tak precizně, jak TeX požaduje [2]. LaTeX jako takový je soubor maker do TeX umožňující relativně snadnou tvorbu běžně používaných prvků formátování programu TeX [1].

Zápis dokumentů v LaTeX připomíná popisný jazyk (podobný např. HTML, XML), ve kterém se vyskytují

- aktivní znaky, např. `&`, `$`, `%`,
- escape aktivních znaků, `\&`, `\$`, `\%`,
- příkazy, např. `\textit{`*text kurzívou*`}`, `\alpha` (zobrazí α),
- prostředí, např. `\begin{`*equation*`}` text `\end{`*equation*`}` vytvoří prostředí pro sazbu matematických rovnic. Text umístěný v (jakémkoli) prostředí je TeXem vnímán jinak. [2]

2.2 Sazba tabulek v LaTeX

Tabulky se v LaTeX běžně tvoří v prostředí `tabular`[1]. Zápis tabulek se řídí obecným pravidlem, že jednotlivé řádky tabulky jsou odděleny pomocí dvou zpětných lomítek (`\\`) a buňky se tabulky se v rámci řádku oddělují znakem `&`, viz tab. 1. [1]

Základní prostředí `tabular` také umožňuje například slučovat buňky (jen horizontálně), měnit velikost řádkování nebo vložit pouze segment vodorovného okraje, což ukazuje tab. 2.

[1] Existuje také prostředí `table`, které tabulku umístí v rámci strany. Samotná sazba tabulky ale probíhá uvnitř prostředí `tabular` (nebo jemu podobných), které se do `table` vnořuje.

Tab. 1: Základní tabulka v prostředí `tabular`. Pomocí příkazu `\hline` je vložen vodorovný okraj, v parametru bezprostředně za `\begin{tabular}` je definováno zarovnání a svislé okraje všech sloupců [vlastní]

```
\begin{tabular}{|l||c|c|c|}
    \hline
    a  & b  & c  & d \\
    \hline\hline
    11 & 12 & 13 & 14 \\
    21 & 22 &    & 24 \\
    text 31 & 32 & 33 & 34 \\
    \hline
\end{tabular}
```

a	b	c	d
11	12	13	14
21	22		24
text 31	32	33	34

Tab. 2: Obtížnější tabulka prostředí `tabular`. [vlastní]

```
% změna řádkování (1.2-násobek)
\renewcommand{\arraystretch}{1.2}
\begin{tabular}{|l||c|c|c|}
    \hline
    a  & b  & c  & d \\
    \hline\hline
        % sloučená buňka.
    11 & \multicolumn{3}{c|}{1212121} \\
    \cline{2-4} % částečný okraj
    21 & 22 & & 24 \\
    text 31 & 32 & 33 & 34 \\
    \hline
\end{tabular}
```

a	b	c	d
11	1212121		
21	22		24
text 31	32	33	34

Pro zadávání složitějších prvků tabulek už jsou potřebné přídavné balíčky, které si lze představit jako rozšíření počtu dostupných TEX maker. Každý balíček zpravidla přidává jednu možnost formátování (nejen) tabulky. Významné LATEX balíčky spojené s tabulkami vyjmenovává tab. 3, kde balíčkem přidaná funkcionalita je zřejmá z jeho popisu.

S postupně rostoucí nepravidelnosti tabulky neúměrně narůstá délka LATEX kódu potřebná k jejímu zobrazení. Stačí si představit, že na většinu nepravidelností se musí vytvořit sloučená buňka o velikosti 1×1 (pomocí `multicolumn`). Ja-

Tab. 3: Významné balíčky pro rozšíření tabulek prostředí `tabular`

`multirow`	víceřádkové buňky tabulek
`xcolor`	barevný text i pozadí
`rotating`	otočené tabulky, obrázky, nebo otočený text
`longtable`	dlouhé tabulky (delší než jedna strana)

kékoli sloučené oblasti lze pomocí `multicolumn` nastavit vlastní zarovnání i oba svislé okraje. Ke snížení množství kódu pro sazbu dlouhých nebo složitých tabulek vzniklo několik přídavných balíčků nabízejících alternativní prostředí pro tvorbu tabulek – `pgfplotstable`, `nicematrix` a `tabularray`.

2.2.1 Baliček pgfplotstable

Balíček `pgfplotstable` slouží ke snadné konstrukci tabulek z datového souboru na bázi `.csv` (comma-separated values). Je vhodný pro dlouhé datové soubory nebo řadu souborů které mají stejnou strukturu, ale jiný obsah. Balíček umožňuje čtení pouze vybraných sloupců vstupního souboru, třídění čtených hodnot a změnu *číselného* formátu čtených hodnot. Celé formátování *vzhledu* tabulky probíhá uvnitř nepovinného parametru příkazu `\pgfplotstabletypeset`, který umožňuje formátování pouze celých řádků nebo celých sloupců. U obou prvků lze přidat okraje, barvu pozadí a lze definovat zarovnání textu (včetně zarovnání podle desetinného oddělovače). [3]

2.2.2 Baliček nicematrix

Balíček `nicematrix` rozšiřuje klasické prostředí `tabular` o sadu užitečných syntaktických konstrukcí. Definuje prostředí `NiceTabular`, které obsahuje prvky pro zápis opakujících se barev pozadí, textu a elegantní zápis „nepravidelných“ či sloučených buněk pomocí příkazu `\Block`. Balíček umí také plošně nastavit existenci, barvu a typ okrajů (plné, tečkované, vlastní tvar pomocí balíčku `tikz`, ...) a nabízí řadu možností formátování celých sloupců tabulky (sloupce typu `X`[2], použití LaTeX příkazu na obsah celého sloupce, velikost mezer mezi sloupci, ...). [4]

2.2.3 Balíček tabularray

Balíček `tabularray` nabízí zcela nový způsob sazby tabulek – možnost kompletního oddělení formátování tabulky od jejího obsahu a zkrácení zápisu opakujících se prvků formátování – což ukazuje tab. 5. Pro sazbu tabulky definuje nové prostředí `tblr`, které je tvořeno *hlavičkou* obsahující pouze formátování a *tělem* obsahující zpravidla

[2] sloupce typu `X` nejsou standardně součástí LaTeX, přizpůsobují svou šířku tak, aby sloupce výstupu byly v zadaném poměru (1:2:1, apod.) a současně vyplnily zbývající šířku strany

Tab. 4: Syntaxe tabulky prostředí `NiceTabular` [4]

```
\begin{NiceTabular}{lr}[hvlines]
\CodeBefore    % "hlavička"
    \rowcolors{1}{blue!10}{}
        [respect-blocks]
\Body
% velikost sloučené oblasti (2x1)
\Block{2-1}{John}  & 12 \\
                   & 13 \\
Steph              & 8~\\
\Block{3-1}{Sarah} & 18 \\
                   & 17 \\
                   & 15 \\
\end{NiceTabular}
```

John	12
	13
Steph	8
Sarah	18
	17
	15

pouze data. Hlavičku lze tvořit velmi obecně, balíček nabízí možnost její zápis uložit jako motiv pro použití v dalších tabulkách. Balíček `tabularray` nabízí i prostředí `longtblr` pro dlouhé tabulky, které rozšiřuje možnosti prostředí `tblr` o poznámky pod čarou, možnost zobrazit n řádků záhlaví na každé straně, apod. [5]

Tab. 5: Obtížná tabulka v prostředí `tblr` [vlastní]

```
\begin{tblr}{
    % hlavička
    colspec = {lrr},
    column{1,3} = {bg=azure8},
    row{1} = {fg=white, bg=azure3,
        font=\bfseries},
    hline{2} = {dashed, azure3},
    vline{1,4} = {},
    cell{2}{2} = {r=2}{c, m},
}
    % tělo
    Alfa   & Beta     & Gama \\
    Delta  & Epsilon  & Zeta \\
    Eta    &          & Theta \\
\end{tblr}
```

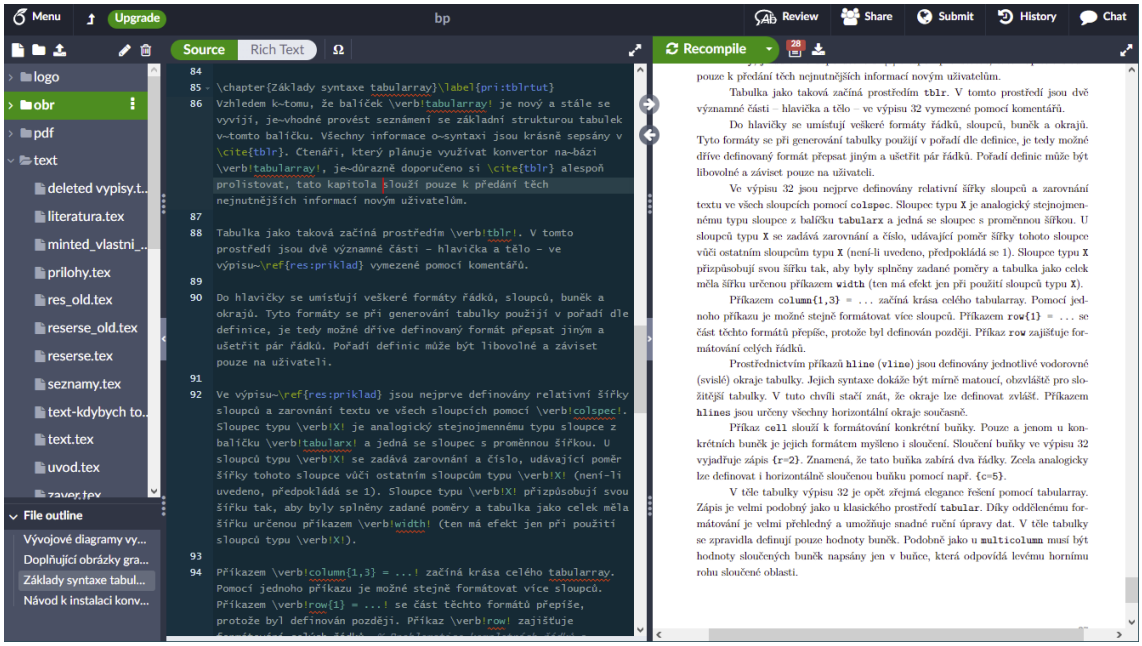
Alfa	Beta	Gama
Delta	Epsilon	Zeta
Eta		Theta

2.3 Editory LaTeX

Čistý TeX i LaTeX dokument lze tvořit v jakémkoli textovém editoru, což ale není příliš praktické, zejména v okamžiku kdy je nutné dokument přeložit. K ulehčení tvorby LaTeX dokumentů vznikla řada různých editorů, které se (se dvěma výjimkami) pro běžného uživatele příliš neliší.

Příkladem typického LaTeX editoru je *TeXstudio*, které nabízí zobrazovač `.pdf` výstupu, našeptávání syntaxe, různé asistenty pro vkládání rovnic, obrázků, případně jednoduchých tabulek, možnosti organizace použité literatury a spoustu dalších užitečných funkcí. [11]. Mezi podobné editory lze zařadit Emacs (v kombinaci s AUCTeX), Vim (spolu s LaTeX-suite), TeXlipse, TeXworks, ...

Pro začínající uživatele je velmi vhodný editor Overleaf. Jedná se o webový editor, který umožňuje tvorbu LaTeX dokumentů bez nutnosti jakékoli instalace nebo nastavení, spolupráci více uživatelů na jednom dokumentu v reálném čase. Tím, že se jedná o webový editor je možné dokumenty v Overleaf upravovat prakticky kdykoli a kdekoli, samozřejmě za předpokladu funkčního (a stabilního) internetového připojení. [22]



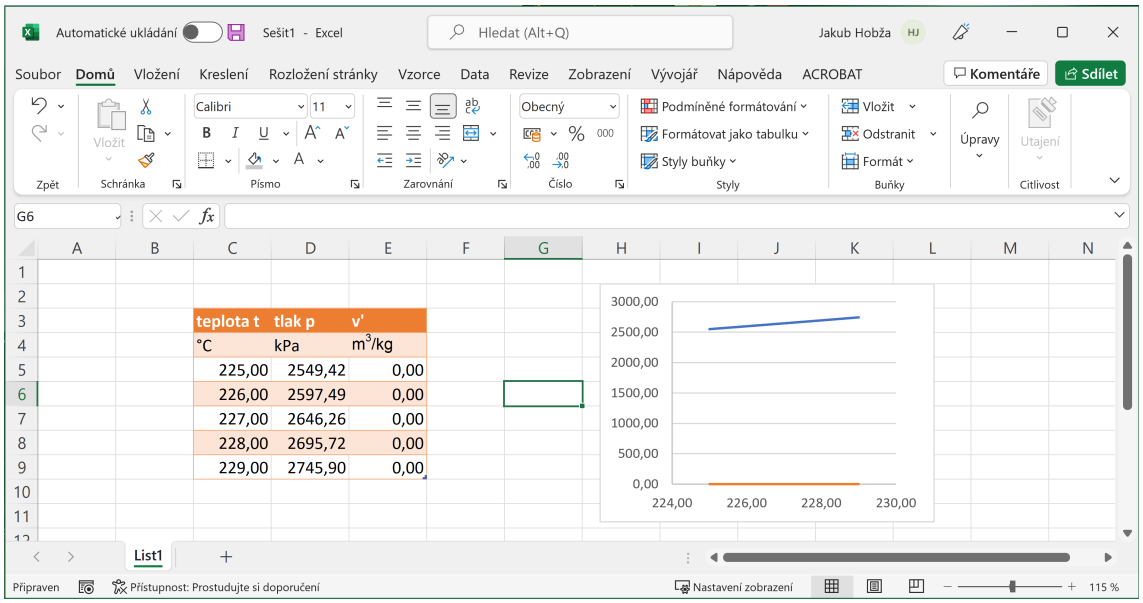
Obr. 1: Editor Overleaf [vlastní]

Dalším netypickým editorem je LyX. Jedná se o mezikrok mezi čistým LaTeX a klasickým textovým procesorem (typicky MS Word). Editor LyX zobrazuje strukturu a zjednodušený náhled výsledného dokumentu v reálném čase. [23]

2.4 Tabulkové editory

Tabulkový editor je program s grafickým rozhraním, umožňující zadat jak tabulková data, tak i jejich formát. Kromě manuálního zadávání dat tabulkové editory zpravidla umí spočítat další hodnoty z hodnot zadaných, zobrazená data třídit, filtrovat nebo zvýrazňovat na základě mnoha kritérií.

Nejpoužívanějším tabulkovým editorem je Microsoft Excel (dále *MS Excel* nebo pouze *Excel*), alternativně se lze setkat s LibreOffice Calc, OpenOffice Calc a Tabulkami Google. Zmíněné alternativy se ale vyskytují spíše minoritně, další pozornost bude tedy zaměřena pouze na MS Excel [6].



Obr. 2: Tabulkový editor MS Excel [vlastní]

2.4.1 Možnosti extrakce dat z MS Excel

Ke zpracování dat do formy LaTeX tabulky je nutné hodnoty z Excelu nějakým způsobem získat. Data lze z Excelu získat prací s datovým souborem typu `.xlsx`, použitím maker, API nebo tvorbou doplňků. V dalším textu tedy bude provedena podrobnější analýza způsobů získání dat z tabulkového editoru MS Excel, pozornost bude zaměřena na jednoduchost použití a kvalitu dokumentace.

Použití VBA maker

Makra pracují přímo v tabulkovém editoru, umožňují čtení, zápis i zpracování hodnot v dokumentu. V MS Excel je možné tvořit makra v jazyce VBA (Visual Basic for Applications), který není příliš rozšířený [20]. Oproti tomu, dokumentace maker je velmi rozsáhlá a plná příkladů použití. Makro je také možné krokovat, sledovat hodnoty proměnných, apod. [7]

Office PIAs (primary interop assemblies)
Jedná se o způsob, kterým lze propojit .NET program přímo s programem MS Excel. Při je často nutná i těchto PIA na zařízení uživatele. Většina extrahovaných hodnot je datového typu `dynamic`, o kterých při krokování .NET programu nejsou dostupné prakticky žádné informace. Situaci nezlepšuje ani dokumentace, která je z velké části generována automaticky a poskytuje málo informací na mnoha místech. [8]

Excel JavaScript API
Pomocí této API je vytvořen Excel doplněk (add-in), který při běhu Excelu může získávat data z některých buněk nebo z celého otevřeného dokumentu. Pro uživatele je vyžadována pouze instalace výsledného doplňku. API používá analogii datového typu `dynamic` zvanou `promise`, ale pro debugování doplňku lze použít log. K dispozici je rozsáhlá dokumentaci s řadou návodů pro začátečníky a kromě samotných dat lze získávat i většinu informací o formátování buněk. [9]

Openpyxl
Openpyxl je knihovna v programovacím jazyce Python. Umožňuje snadné čtení i zápis `.xlsx` souborů (soubory typu `.xls` nejsou podporovány). Knihovna čteným datům automaticky přiřazuje správný datový typ. Při použití openpyxl je možné program krokovat. Knihovna obsahuje rozsáhlou dokumentaci s několika návody pro začátečníky. Instalace knihovny je snadná, k použití `openpyxl` není vůbec potřeba mít na zařízení Excel nebo jakýkoli jiný tabulkový editor. [10]

2.5 Současné nástroje pro ulehčení sazby LaTeX tabulek

V současnosti existuje již několik nástrojů ulehčující tvorbu tabulek v LaTeX. Využívají jak možnosti tabulkových editorů, tak vlastních grafických rozhraní.

2.5.1 Průvodce tabulek v TeXstudio

Velmi jednoduchým příkladem nástroje vytvořeného přímo pro sazbu LaTeX tabulek je průvodce tabulek v TeXstudio. Jedná se o nástroj, který podporuje pouze základní prvky prostředí `tabular`, tedy nejsložitější struktura tabulky, kterou umí vytvořit, jsou tabulky s prvky tab. 2. [11]

2.5.2 Excel2LaTeX

Excel2LaTeX je makro do MS Excel, kromě standardních prvků `tabular` do výstupní tabulky umí také zahrnout:

- Barvu textu a barvu pozadí.
- Individuální okraje buněk.
- Okraje z balíčku `booktabs`.
- Natočení textu.
- Oblasti formátované jako tabulka.
- Formát textu (počet desetinných míst, oddělení tisíců, koncové nuly, apod.).

Využívá přídavných balíčků `xcolor`, `booktabs` a `rotating`.

2.5.3 LaTeX Tables Editor

LaTeX Tables editor je webová aplikace, která umožňuje snadnou tvorbu LaTeX tabulek v prostředí podobném MS Excel. Nabízí širokou škálu možností zadání vstupních dat, z nichž nejzajímavější je prosté vložení dat z jiného tabulkového editoru. Data se vloží včetně veškerého formátování textu i buněk tabulky a je zachována původní struktura celé tabulky. [12]

Po zadání dat kteroukoli z dostupných metod s nimi lze dále pracovat a provádět dodatečné úpravy. V každém okamžiku je zobrazeno jak bude výsledná LaTeX tabulka vypadat. LaTeX Tables Editor do výstupních tabulek umí zakomponovat:

- Diagonálně rozdělené buňky.
- Poznámky pod čarou.
- Sloučené buňky.
- Barvu textu a barvu pozadí.
- Individuální okraje buněk.
- Okraje z balíčku `booktabs`.
- Natočení textu.
- Zarovnání textu *individuálních* buněk (vč. zarovnání podle desetinného oddělovače).
- Escape speciálních LaTeX symbolů (`% # ~ } { ] [`).
- Odrážkové seznamy uvnitř buněk. [12]

Výstupní tabulky mohou být vytvořeny pro prakticky jakékoli prostředí tabulek, které LaTeX nabízí, včetně prostředí z balíčku `tabularray` a `nicematrix`. [12]

3 VLASTNÍ ŘEŠENÍ

V kapitole je provedeno seznámení se s již hotovým konvertorem, jak z hlediska použitých prostředků, tak i pomocí prohlídky grafického rozhraní. Jádrem kapitoly je popis vybraných částí algoritmu v rámci celého procesu převodu tabulky, pozornost je věnována získávání, základům zpracování a tisku potřebných dat.

Samotný text kapitoly je v obtížněji pochopitelných částech doplněn o výpisy kódu. Výpisy primárně slouží k ucelení výkladu, případně jako rychlá reference. Je-li výklad pochopen, je bez problémů možné výpisy přeskakovat.

Často bude proveden i odkaz na přílohu A, která má podobný účel. Tato příloha obsahuje vývojové diagramy vybraných částí algoritmu.

3.1 Volba způsobu řešení

Pro vstupní tabulku byl zvolen přístup s využitím již existujícího tabulkového editoru, jedná se o nejjednodušší řešení. Tabulkové editory jsou mezi uživateli rozšířené a dá se realisticky očekávat, že uživatelé s nimi umí nejen pracovat, ale jako podporovaný vstup jej vyžadují. Vzhledem ke svému dominantnímu postavení byl tedy zvolen tabulkový editor MS Excel.

Extrakce dat ze vstupního souboru je prováděna pomocí knihovny `openpyxl`. Autor textu v minulosti pracoval s .NET, bohužel zde dostupné prostředky jsou obtížně použitelné, viz kap. 2.4.1. Knihovna `openpyxl` má nejmenší vstupní náročnost, tj. s touto knihovnou se dá naučit pracovat snadno a rychle, totéž lze říci i o jazyce Python. Dalším důvodem pro volbu `openpyxl` je možnost celý konvertor krokovat a během krokování mít *detailní* informace o okamžitém stavu celého programu.

Pro tvorbu grafického rozhraní byla zvolena knihovny `pyImGui`. Autor textu má kladnou zkušenost s jejím C++ vzorem, `ImGui`. Obě knihovny jsou prostředkem ke snadné a intuitivní tvorbě vysoce funkčních grafických rozhraní.

3.2 Uživatelské rozhraní

Uživatelské rozhraní je vytvořeno tak, aby umožnilo zadání všech potřebných údajů co možná nejpohodlněji. Rozhraní je rozděleno do sedmi významných sekcí, které lze vidět na obr. 3 a 4 – *Profil nastavení, Vstup, Převádět, LaTeX, Legenda, Výstup* a *Log*.

Sekce *Profil* nastavení slouží k ukládání všech nastavení do souborů `.ini` na pevném disku. Umožní uživateli snadno a rychle celý konvertor nastavit tak, jak potřebuje.

Vstup slouží k získání dat nutných k zahájení převodu. Vždy je nutno vybrat `.xlsx` soubor, který bude sloužit jako zdroj dat[1]. Uživateli je dáno několik možností jak zvolit oblast k převedení dat. Podle zvolené možnosti se rozhraní mírně změní a zobrazí údaje, které se musí zadat. Tuto situaci zachycuje obr. 5.

Převádět obsahuje řadu nastavení, která určují, co vše ze vstupního souboru se má zahrnout do výstupní tabulky. Umožňuje zcela vynechat některé typy formátování uložených ve vstupním souboru.

LaTeX obsahuje dodatečná nastavení výstupní tabulky. Tato nastavení pouze mírně pozmění některé části výstupu.

Legenda umožňuje do výstupní tabulky zahrnout krátké vysvětlení jednotlivých částí výstupu pomocí LaTeX komentářů. Tato vysvětlení slouží primárně jako reference při budoucích úpravách výstupní tabulky bez nutnosti cokoli dohledávat.

Výstup pouze zobrazuje výslednou tabulku a umožňuje ji snadno celou zkopírovat.

Log slouží primárně k upozornění uživatele na nesprávný vstup. Objevují se zde informace typu „nemáte vybraný vstupní soubor“, „nezadali jste oblast k převodu“, ale také například „změna velikosti písma se projeví až po restartu programu“. V případě, že nastane výjimka, program se pokusí ji zachytit a danou chybovou hlášku zobrazit také v tomto okně.

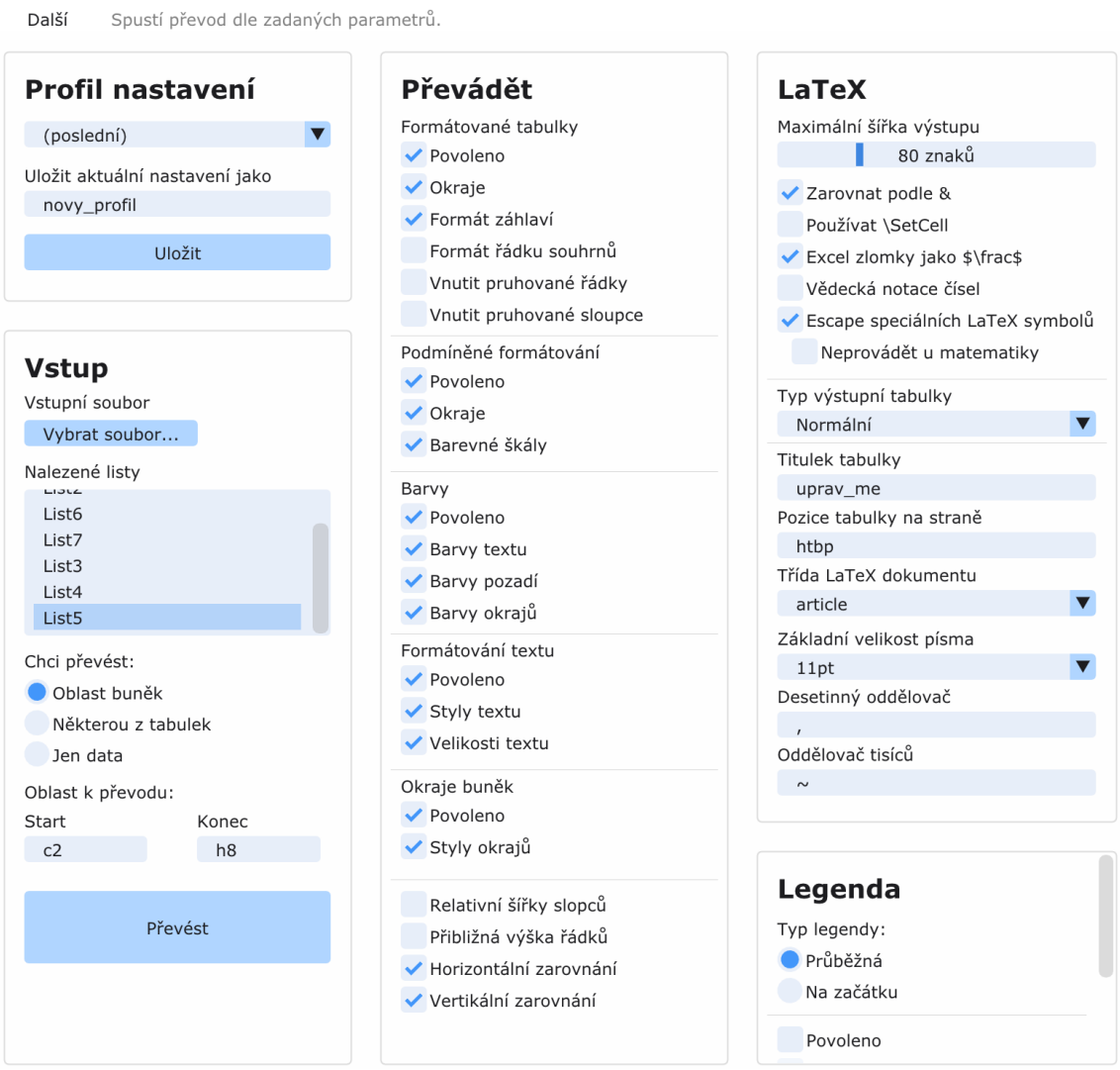
3.2.1 Další prvky a modifikace grafického rozhraní

Grafické rozhraní také obsahuje další prvky, které přímo neslouží k převádění tabulek, ale především jako nápověda pro uživatele. Obrázky k většině těchto prvků se nachází v příloze B. Nejčastěji je využito kontextové nápovědy uvnitř stavového řádku v horní části rozhraní. Zde se po najetí myší na daný prvek rozhraní ukáže popis toho, co daný prvek ovládá, k čemu je, případně jak ovlivní výstup.

Ve stavovém řádku je také menu *Další* obsahující především informace, které by uživatel mohl během práce v konvertoru potřebovat – jak výstupní tabulky zprovoznit v LaTeX dokumentu, jak postupovat když převodník hlásí chybu nebo vysvětlení některých výstupů z Logu. V menu *Další* se dále nachází *Cheat sheet* obsahující syntaxi a popis všech částí výstupu včetně příkladů použití.

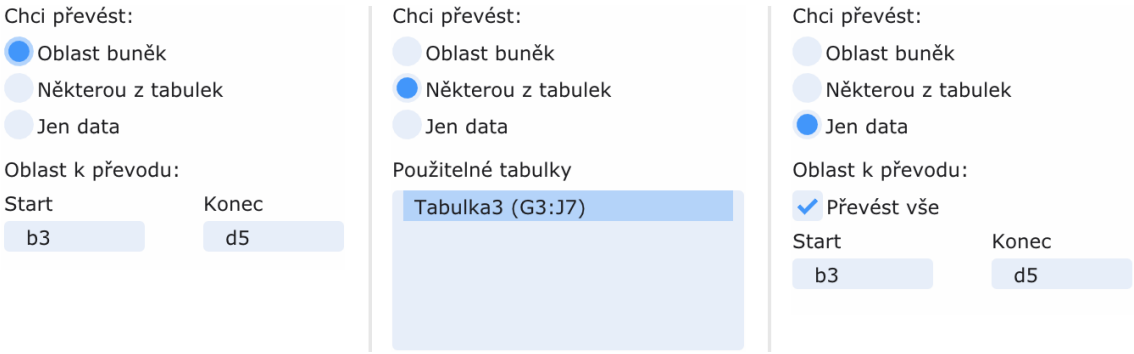
V případě, že je zvolen typ převodu *Jen data*, se celé rozhraní výrazně změní viz obr. 31 v příloze B. Sekce Převádět je nahrazena textovým polem, které umožňuje *ruční* zadání hlavičky výstupní tabulky.

[1] kromě souborů `.xlsx` jsou podporovány i soubory `.xlsm`. Ostatní formáty Excel souborů je nutné před použitím převést na některý z podporovaných formátů.



Obr. 3: Sekce *Profil nastavení*, *Vstup*, *Převádět*, *LaTeX* a *Legenda*. Nahoře se nachází stavový řádek s kontextovou nápovědou. [vlastní]

Obr. 4: Sekce *Výstup* a *Log* [vlastní]



Obr. 5: Možnosti zadání oblasti se vstupními daty [vlastní]

3.2.2 Ukládání nastavení

Jelikož konvertor obsahuje velký počet nastavení, je nezbytné nastavení ukládat, ideálně do více *profilů*. Veškerá nastavení jsou uložena v objektech vlastních datových typů a sjednocena v souboru `settings.py`, ke kterému mají všechny zbylé části kódu přístup.

Ukládání a načítání nastavení je řešeno s pomocí standardní Python knihovny `configparser`, která umožňuje čtení a zápis hodnot `.ini` souborů. Celá logika ukládání a načítání nastavení vychází z dokumentace `configparser` [25], jeden profil nastavení odpovídá jednomu `.ini` souboru. Nastavení je po úspěšném dokončení každého převodu automaticky uloženo do profilu „(poslední)“, který je také vždy při startu programu načten.

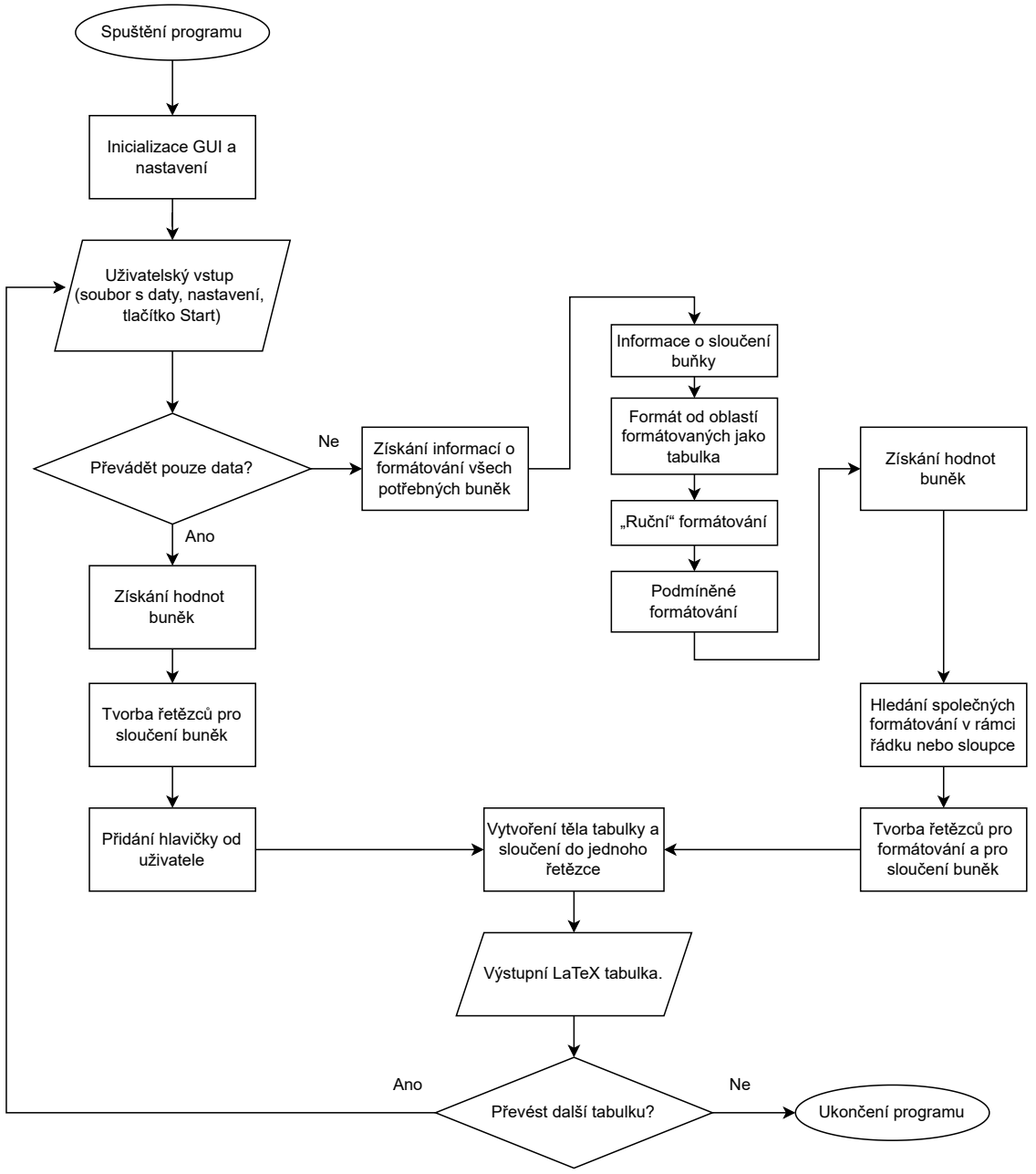
Při čtení hodnot z uložených souborů bylo nezbytné ošetřit případy, kdy bylo se souborem nastavení manipulováno. Konkrétně se jedná se o případy, kdy:

- Soubor nastavení neexistuje .
- V souboru chybí některá nastavení.
- Soubor obsahuje nesprávné datové typy.
- Datový typ je správný, ale hodnota přesahuje nějaké dovolené meze.

3.3 Algoritmus funkce konvertoru

Na obr. 6 je velmi zjednodušeně naznačen algoritmus, pomocí kterého konvertor pracuje. Algoritmus je stavěn dle syntaxe tabulek v balíčku `tabularray`, tedy kromě samotného získání potřebných údajů se z nich snaží vytvořit hlavičku obsahující formátování buněk. Je žádoucí hlavičku vytvořit tak, aby byla co možno nejkompaktnější, tedy místo rozsáhlého seznamu s formátováním každé buňky vytvořit výstup, který využívá možnosti formátovat celé sloupce nebo řádky pomocí jediného řádku textu v hlavičce výstupu.

Získání potřebných údajů buněk probíhá na první pohled zvláštním způsobem. Potřebné údaje jsou v `openpyxl` (resp. v `.xlsx` souborech) uložené na více místech a vzájemně se nevylučují. V buňkách tedy mohou být uložena až tří různá formátování současně a rozhodnutí o tom, které z nich se má použít, je na programátorovi. Ukázalo se, že existuje jistá hierarchie, na jejímž základě se formátování zobrazují a přesně podle ní pracuje i algoritmus konvertoru. Konvertor postupuje od formátování s nižší prioritou (oblasti formátované jako tabulka) k formátování s prioritou vyšší (podmíněné formátování) a získané údaje se postupně přepisují.



Obr. 6: Zjednodušený algoritmus převodu dat do LaTeX [vlastní]

3.4 Použité datové typy a struktury

V textu je velmi často odkazováno na „objekt datového typu `X`“. Tímto spojením je myšlen konkrétní objekt v kódu programu, obsahující potřebná data. Protože se jedná o objekty, které si lze pojmenovat libovolně, jsou v textu použity právě i odkazy na datový typ, aby bylo zřejmé, kde potřebné údaje hledat. V textu je ale také nutno tyto objekty konkrétně pojmenovat a zvolené názvy jsou v seznamu níže uvedeny v závorce za datovým typem.

Jelikož jsou potřebné údaje v `openpyxl` často uloženy na různých místech, bylo založeno i několik vlastních datových typů, do kterých se ukládají postupně získávané údaje z datových typů v `openpyxl`, aby bylo možné snáze vytvářet výstupní řetězce.

1. Datové typy v `openpyxl`:
   - `Workbook` (objekt `wb`): celý Excel dokument.
     - `Worksheet` (objekt `ws`): list Excel dokumentu.
       - `Cell` (objekt `cell`): buňka v daném listu dokumentu.
2. Vlastní datové typy:
   - `CCol` (objekt `my_col`): ukládá společná formátování buněk jednoho sloupce tabulky a reference na buňky, které sám obsahuje (typu `CCell`).
   - `CRow` (objekt `my_row`): ukládá společná formátování buněk jednoho řádku tabulky a reference na buňky, které sám obsahuje (typu `CCell`).
   - `CCell` (objekt `my_cell`): ukládá informace o formátování, o obsahu a o sloučení buněk výstupní tabulky.
     - `CDxf` (objekt `dxf`): obsahuje informace o většině formátování buňky.
   - `CTableStyle`: sjednocuje získané údaje o formátování oblastí tabulek.

Jakékoli datové typy začínající velkým E jsou také vlastní. Jedná se o tzv. `Enum`, konkrétně potomky potomky třídy `Enum` a lze si je představit jako množinu několika možných hodnot. Důvodem k používání datových typů na bázi `Enum` je snížení paměťové náročnosti ukládaných dat. Datové typy na bázi `Enum` jsou hojně používány v objektech typu `CCell`, `CCol`, `CRow` a `CDxf`.

V některých pasážích textu jsou použity odrážkové seznamy k naznačení struktury nějakého datového typu.

- Úrovně v seznamu označují hierarchii probírané datové struktury.
- Zápis „. . . “ znamená že daná struktura pokračuje, ale buďto je způsob pokračování zřejmý, nebo z hlediska výkladu nepodstatný.
- Poslední zápis vyskytující se v těchto seznamech je „(`datový_typ`)“, představující datový typ objektu, vedle kterého je napsán (např. „`(int)`“). Tento zápis je uveden pouze v případě, je-li datový typ daného objektu známý a slouží především k možnosti si vyhledat další informace v patřičné dokumentaci.

3.4.1 Datový typ CDxf

Na tento datový typ je v textu odkazováno na několika mezi sebou vzdálených místech. Pro snadnou referenci je umístěn tam, kde je nejlogičtější jej hledat.

V tomto *vlastním* datovém typu jsou ukládány informace o většině formátování všech tří typů – ručního, podmíněného i oblastí tabulek. Často je použit jako *prostředník* k uložení jednoho typu formátování a v další části algoritmu se z něj potřebné informace uloží do objektu `CDxf` konkrétní buňky. Struktura typu `CDxf` je:

- `CDxf`
  - `font_flags` (`EFontFlags`): informace o řezu písma a podtržení písma.
  - `font_size` (`EFontSize`): LaTeX velikost písma.
  - `bg` (`EColor`): barva pozadí.
  - `fg` (`EColor`): barva textu.
  - `halign` a `halign` (`EHAlign` resp. `EVAlign`): horiontální a vertikální zarovnání textu.
  - `borders` (`EBorder`): informace o tom, které okraje buňka má.
  - `border_styles` (`tuple[EBorderStyle]`): styl okrajů (tučný, tenký, dvojitý, ...).
  - `border_colors` (`tuple[EColor]`): barva okrajů.
  - `border_priorities` (`tuple[int]`): priorita okrajů.

3.5 Získání informací o formátování buněk vstupní tabulky

Celý proces získávání informací o buňkách je jeden velký cyklus, ve kterém se po řádcích iteruje celou tabulkou a v rámci řádku je prováděna iterace jednotlivými buňkami (zleva-doprava). Celý proces zachycuje obr. 26 v příloze A. Náplní této kapitoly je vysvětlení jednotlivých kroků procesu s výjimkou společných vlastností a textového formátu (těmi se zabývají kapitoly 3.6 a 3.7). U jednotlivých kroků je řečeno kde hledat potřebné údaje, jaká je struktura objektů, ze kterých se údaje získávají, a jak se získanými údaji dále nakládat.

3.5.1 Sloučení buněk

Sloučené buňky jsou všechny hromadně uloženy v objektu typu `Worksheet`. V poli `ws.merged_cells.ranges` je k nalezení seznam všech oblastí, které sloučené buňky zabírají. Ke zjištění informace o sloučení buňky stačí zjistit, zda zkoumaná buňka není v tomto seznamu.

Veškeré formátování a data jsou vždy uložena v buňce odpovídající levému hornímu rohu sloučené oblasti. Zbylé buňky ze sloučené oblasti neobsahují žádné údaje a jejich hodnota je `None`. Úplně stejné údaje obsahují i nijak neformátované prázdné buňky. Prakticky jediným způsobem jak odlišit prázdné buňky od části sloučené oblasti je u každé buňky v úplně prvním kroku zkoumat, zda je sloučená.

K zachování formátování v celé oblasti sloučených buněk je vhodné všechna potřebná data ukládat do vlastního datového typu tak, že do každé buňky sloučené oblasti se uloží informace následujícím způsobem:

- Najde se levý horní roh sloučené oblasti buněk (`start_cell`).
- Veškeré údaje o formátování se zjistí ze `start_cell` a uloží do buňky vlastního datového typu `CCell`.

- Do buňky vlastního datového typu se uloží informace o tom, zda se jedná o tu část sloučené oblasti obsahující data či nikoli.
- V případě že se jedná o část obsahující data, uloží se informace o velikosti sloučené oblasti a hodnota buňky. Jinak se jako hodnota buňky nastaví prázdný řetězec.

Důvodem k takovému zacházení se sloučenými buňkami je způsob, kterým jsou řešeny společné vlastnosti buněk (viz kap. 3.7).

3.5.2 Velikosti buněk

Změnou rozměrů řádku (sloupce) ve vstupním souboru je automaticky provedena změna rozměru všech buněk daného řádku (sloupce). Ke zjištění velikosti konkrétní buňky tedy stačí zjistit šířku daného sloupce a výšku daného řádku.

Informace o výškách všech řádků jsou uloženy v poli `row_dimensions` objektu typu `Worksheet`. Jednotlivými prvky pole jsou objekty typu `RowDimension`, které obsahují právě informace o výšce každého řádku uvnitř pole `height`. Výšky jsou uloženy v bodech (pt) a získanou hodnotu stačí jednoduše uložit (výpis 1).

Udaje o šířkách sloupců se nachází v poli `column_dimensions` objektu typu `Worksheet`. Jednotlivé prvky tohoto pole obsahují informaci o šířce sloupce v poli `width`. Uložené údaje mají zvláštní jednotku, která závisí na zvoleném *základním* písmu dokumentu (včetně jeho velikosti) [17]. Řešení použité v konvertoru použije hodnotu v Excel jednotkách jako *relativní* šířku sloupce, tedy tato hodnota udává jednu část poměru, ve kterém budou v další fázi převodu šířky sloupců rozděleny.

V případě, že je nastaven výchozí rozměr řádku (15) nebo sloupce (8,43), příslušný rozměr je ve vstupním souboru uložen jako `None`. V konvertoru je tento případ ošetřen tak, že se danému výstupnímu řádku, resp. sloupci, přiřadí výchozí hodnota.

```
# excel_row (int) - pořadí řádku ve vstupním souboru (indexováno od 1)
my_row.vsize = ws.row_dimensions[excel_row].height
if my_row.vsize is None:
    my_row.vsize = 15

# excel_col (int) - pořadí sloupce ve vstupním souboru (indexov. od 1)
# do column_dimensions je potřeba přistupovat TEXTOVÝM označením
    sloupce (A, B, XD, ...)
col_letter_idx = utils.num_to_col(excel_col)
my_col.hsize = ws.column_dimensions[col_letter_idx].width
if my_col.hsize is None:
    my_col.hsize = 8.43
```

Výpis 1: Získání rozměrů řádku a sloupce [vlastní]

3.5.3 Ruční formáty buněk

Ručními formáty jsou myšlena formátování ze sekce *Písmo* a *Zarovnání* v prostředí Excel (viz obr. 7). Patří sem také formátování, která vzniknou při vkládání mezi Excel dokumenty v případě, že je zvolen typ vložení *Hodnoty a formátování zdroje*.

Všechny informace o formátování buněk těmito způsoby jsou v `openpyxl` uloženy prakticky na jednom místě – přímo v objektu typu `Cell`. Jejich zpracování je relativně snadné a prakticky jedinou velkou překážkou jsou barvy textu i pozadí buněk.

Obr. 7: Prvky pro nastavení „ručních“ formátování [vlastní]

Styl písma a zarovnání textu

Všechny informace o řezu písma jsou jednoduše přístupné v objektu typu `Cell`, konkrétně v poli `cell.font`. Informace o řezu písma lze naleznout v polích `cell.font.b` (tučný text) a `cell.font.i` (kurzíva), informace o podtržení textu se nachází v poli `cell.font.u`.

Informace o zarovnání textu lze také získat snadno, všechny informace jsou v objektu typu `Cell` resp. jeho poli `cell.alignment`. Rozlišují se dva typy zarovnání – horizontální a vertikální – informace o každém typu jsou uloženy zvlášť v polích `cell.alignment.horizontal` a `cell.alignment.vertical` prostřednictvím slovního popisu (typ `str`). Ke zpracování informací o zarovnání stačí uvažovat první znak tohoto popisu.

V současné verzi `openpyxl` (3.0.9) je možné zjistit pouze informace o formátování textu jako celku a informace o formátování pouze části textu buňky nejsou dostupné žádným způsobem.

Velikost písma

Velikost písma je uložena v poli `cell.font.sz`. V LaTeX není za běžných okolností možné zadávat absolutní velikost písma a je nezbytné ji ve výsledku převést na řetězec typu `\Large`, `\small` apod. Celkem je v LaTeX k dispozici až 12 předdefinovaných velikostí písma – `\scriptsize`, `\footnotesize`, `\small`, `\large`, `\Large`, `\LARGE` a `\huge`, případně (pouze pro některé třídy dokumentů) `\Huge`, `\HUGE` a `\miniscule`.

Problémem je určit, jak jsou tyto příkazy, tzv. přepínače, svázány s velikostí písma v bodech. Skutečná velikost písma při použití daného přepínače se odvíjí od základní velikosti písma v LaTeX dokumentu a třídě tohoto dokumentu. Základní velikost písma je velikost, kterou má text v případě, že není aktivní žádný z přepínačů

pro změnu velikosti textu (tj. `\small` apod.). Existují desítky tříd LaTeX dokumentů, které se mohou z pohledu počtu dostupných základních velikostí písma výrazně lišit. Mezi nejzákladnější třídy LaTeX dokumentů se řadí `article`, `report`, `book`, `letter`, `beamer` a `memoir`.

Z těchto vyjmenovaných jsou ve všech kromě třídy dokumentu `memoir` dostupné tři základní velikosti písma – 10 pt, 11 pt a 12 pt – k jedné základní velikosti písma mají všechny tyto třídy přiřazeny stejné i velikosti písma v bodech.

Tab. 6: Přiřazení konkrétních velikostí písma v bodech k LaTeX přepínačům ve třídách dokumentu `article, report, book, letter` a `beamer`

	Základní velikost		
Přepínač	10pt	11pt	12pt
`\tiny`	5	6	6
`\scriptsize`	7	8	8
`\footnotesize`	8	9	10
`\small`	9	10	10.95
`\large`	12	12	14.4
`\Large`	14.4	14.4	17.28
`\LARGE`	17.28	17.28	20.74
`\huge`	20.74	20.74	24.88
`\Huge`	24.88	24.88	24.88

Třída dokumentu `memoir` je z pohledu velikosti písma odlišná. Definuje celkem 11 základních velikostí písma a také další příkazy pro velikost textu v LaTeX. Pro každou základní velikost textu je u třídy `memoir` k dispozici celkem 11 přepínačů písma. Ukázalo se, že v tomto rozsáhlém množství kombinací existuje řád. Velikosti písem (v bodech) od přepínačů vždy ve výsledku odpovídají souvislému úseku 12 prvků z řady

4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 10.95, 12.0, 14.4, 17.28, 20.74, 24.88, 30.0, 36.0, 48.0, 60.0, 72.0, 84.0, 96.0, 108.0, 120.0, 132.0

který pro nejmenší *základní* velikost textu začíná na levém konci zmíněné řady a pro rostoucí základní velikost textu se začátek úseku postupně v rámci řady posunuje směrem vpravo. Pro jinou základní velikost úsek začíná na druhém prvků zleva a výsledná velikost se opět postupně v rámci řady posouvá směrem vpravo, atd.

Samotný převod velikostí písma probíhá tak, že na základě uživatelem zadané třídy dokumentu a základní velikosti písma je vybrán jeden z ručně definovanách `dictionary`[2], viz výpis 2.

```
# 6.0, 8.0, 9.0, ... klíče
# EFontSize.tiny, EFontSize.scriptsize, ... hodnoty
# EFontSize je vlastní datový typ. Lze použít klasicky '\Large' apod.
article11 = {
       6.0: EFontSize.tiny, 8.0: EFontSize.scriptsize,
       9.0: EFontSize.footnotesize, 10.0: EFontSize.small,
       10.95: EFontSize.NORMAL, 12.0: EFontSize.large,
       14.4: EFontSize.Large, 17.28: EFontSize.LARGE,
       20.74: EFontSize.huge, 24.88: EFontSize.Huge
   }
```

Výpis 2: Příklad `dictionary` k určení velikosti písma [vlastní]

U každého `dictionary` jsou použity možné velikosti písma v bodech jako klíče a k nim jsou svázány LaTeX přepínače, kterými pro danou třídu a základní velikost bude této velikosti v bodech dosáhnuto. Postupnou iterací přes všechny klíče vybraného `dictionary` je nalezen ten klíč, který je nejblíže velikosti písma buňky. Hodnota odpovídající tomuto klíči je převedenou velikostí písma.

Barva textu a barva pozadí

Informace o ručně nastavené barvě textu je vždy uložena v poli `cell.font.color`. Barva ručně nastaveného pozadí je uložena v poli `cell.fill.fgColor`. Obě tyto barvy mají stejný datový typ a lze je převést stejně. Struktura tohoto datového typu, `color`, je:

- `index` (`int`): Index barvy v případě, že je indexovaná.
- `rgb` (`str`): Barva v `#AARRGGBB` formátu.
- `theme` (`int`): Pořadí barvy v rámci aktuálního motivu.
- `tint` (`float`): Číslo udávající zesvětlení nebo ztmavení barvy uložené v motivu. Pohybuje se od -1 do 1.
- `type` (`str`): Typ uložené barvy, zpravidla `'theme'`, `'rgb'` nebo `'indexed'`.

[2] `dictionary` (v doslovném překladu *slovník*) je speciální datový typ umožnující k jedné hodnotě svázat druhou hodnotu. Podobně jako slovník ke slovu sváže jeho slovní definici, `dictionary` sváže tzv. *klíč* s *hodnotou*. Datový typ `dictionary` se také dá představit jako pole (`array`), ve kterém se k prvkům nepřistupuje pomocí číselného indexu, ale pomocí obecného datového typu (klíče). `dictionary` nemusí být konstantní ani definovaný předem, při běhu programu lze ukládat nové dvojice klíč-hodnota. Jako hodnotu lze použít prakticky cokoli, klíč musí být *hašovatelného* datového typu. [24]

Během převádění barev mohou nastat následující situace:

1. Barva je nastavena jako RGB (red, green, blue) a je možné ji naleznout v poli `color.rgb`. Tato barva je uložena ve formátu `#AARRGGBB`. Zbylé údaje jsou nastaveny na `None`.
2. Barva je nastavena pomocí motivu (`theme`) a odstínu (`tint`). Zbylé údaje jsou nastaveny na `None`.
3. Barva je indexována (není uvažováno).
4. Barva není ručně nastavena, používá se automatická barva, všude je nastaveno `None`.

Převod barvy motivu do RGB

O buňkách, které mají svou barvu nastavenu jako barvu motivu je v `color` uložen jen index (pořadí) této barvy v rámci motivu a odstín, který udává o kolik se má získaná barva zesvětlit nebo ztmavit. K získání jakékoli barvy z těchto dvou údajů jsou nutné dva další kroky.

V první fázi je potřeba z indexu barvy v rámci motivů získat RGB barvu. Základních 10 barev motivu je uloženo v poli `wb.loaded_theme` objektu typu `Workbook` a lze je získat pomocí nástrojů pro čtení `.xml` souborů. Kompletní algoritmus takové extrakce je k nalezení v [16].

Struktura uložených barev je vidět ve výpise 3. Barvy motivu jsou vždy uloženy pod názvy `dk1`, `dk2`, `lt1`, `lt2`, `accent1`, , `accent2`, . . . , `accent6`. Ke každému z těchto názvů je v `.xml` definována barva v RGB, kterou je nutné u všech názvů vypreparovat.

```
<a:clrScheme name="\xc5\xbdlut\xc3\xa1">
    <a:dk1>
        <a:sysClr val="windowText" lastClr="000000"/>
    </a:dk1>
    ...
    <a:accent1>
        <a:srgbClr val="39302A"/>
    </a:accent1>
    ...
    <a:accent6>
        <a:srgbClr val="9C6A6A"/>
    </a:accent6>
...
</a:clrScheme>
```

Výpis 3: Struktura uložených barev motivu (kráceno) [vlastní]

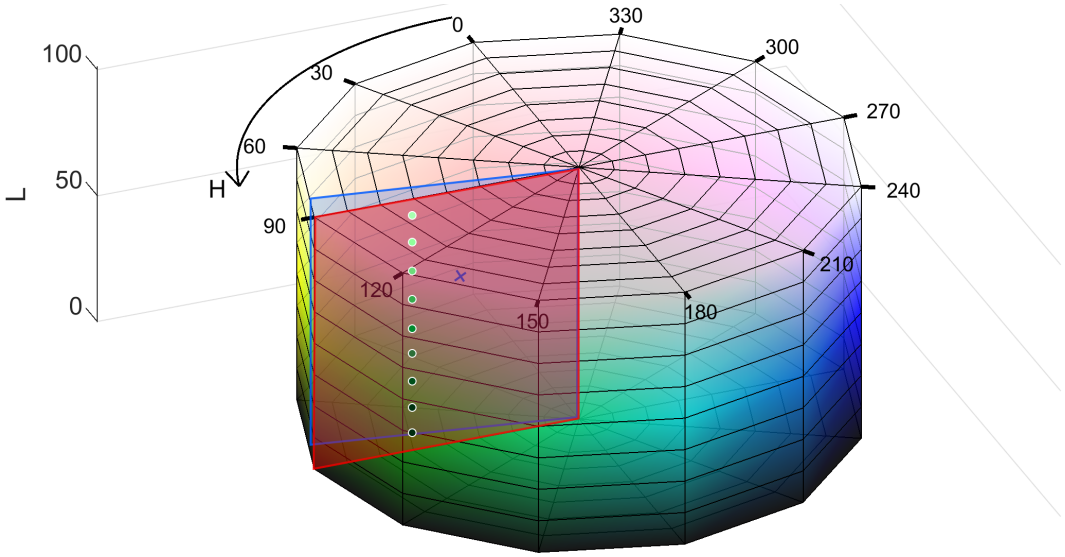
Pomocí indexu uloženého v `color.theme` lze určit, kterou z barev motivu má buňka nastavenou. Tento index udává, o kterou barvu ze sekvence `dk1`, `dk2`, `lt1`, `lt2`, `accent1`, ..., `accent6` se jedná.

Druhým krokem je zesvětlení nebo ztmavení právě získané barvy pomocí odstínu (`color.tint`). V MS Excel je změna světlosti řešena tak, že se barva motivu převede z RGB do HSL a k hodnotě L (luminance) se přičte nebo odečte odstín. Získaná barva je buď převedena do RGB, v případě, že se týká barevné škály, v ostatních případech je ponechána v HSL.

Získání nejbližší zobrazitelné barvy

Při použití balíčku `tabularray` není k dispozici celé barevné spektrum. Počet barev je omezený na 117 předem vybraných HSB barev definovaných v balíčku `ninecolors`. Aby bylo možné barvu ze vstupním souboru ve výsledném dokumentu zobrazit je potřeba k získané barvě buňky najít nejbližší zobrazitelnou barvu z `ninecolors`.

Nejbližší zobrazitelná barva se získává tak, že v prvním kroku je vstupní barva převedena do HSL. Barevný prostor HSL si lze představit jako válcový s úhlovou souřadnicí H (odstín), radiální souřadnicí S (sytost) a axiální souřadnicí L (světlost). Zobrazitelné barvy existují jako body v HSL prostoru, kdy jedna sada zobrazitelných barev má stejnou hodnotu odstínu H a tím pádem leží v jedné polorovině. Celkem lze sestrojit dvanáct polorovin zobrazitelných barev, viz obr. 8, které prochází v ose H prochází násobky 30 stupňů. Zobrazitelné barvy jsou v rámci poloroviny rovnoměrně rozmístěny prakticky na jedné svislé přímce.



Obr. 8: Prostorové uspořádání jedné sady zobrazitelných barev v rámci příslušné poloroviny. Všechny poloroviny prochází hranami tohoto dvanáctistěnu. Barva buňky leží v obecné polorovině a je znázorněna modře, červeně je zobrazena nejbližší z polorovin zobrazitelných barev. [15]

Druhým krokem je nalézt u barvy buňky nejbližší polorovinu zobrazitelných barev. Stačí hodnotu souřadnice H barvy buňky zaokrouhlit na nejbližší násobek 30. Je-li souřadnice H buňky přibližně stejně daleko od dvou polorovin (např. 14 stupňů od jedné a 16 stupňů od druhé), pak je vždy zaokrouhleno nahoru, tedy, je vybrána polorovina s vyšší souřadnicí H.

V posledním kroku je k barvě buňky hledána nejbližší barva ze zvolené poloroviny. Spočítá se vzdálenost barvy buňky od každého z bodů poloroviny pomocí kosinové věty (barva buňky nemusí ležet v dané polorovině) a uloží se spolu s porovnávanou barvou do seznamu vypočítaných vzdáleností. Po srovnání všech barev poloroviny s barvou buňky je v tomto seznamu nalezeno minimum a jelikož je ke každé spočítané vzdálenosti svázána i barva, nalezením minima vzdálenosti je nalezena i nejbližší zobrazitelná barva.

3.5.4 Podmíněné formátování

Informace o podmíněném formátování bohužel nejsou uloženy přímo v objektech typu `Cell`. Konkrétní podmíněná formátování jednotlivých buněk vlastně nejsou uložena nikde a k dispozici je pouze seznam pravidel podmíněného formátování v celém listu. Zmíněný seznam se nachází v objektu typu `Worksheet` v poli `ws.conditional_formatting._cf_rules` jehož struktura je:

- `ws.conditional_formatting._cf_rules`
  - Oblast `A1:D5` (`ConditionalFormatting`)
    - `cells` (`MultiCellRange`): Oblast buněk pro které pravidlo může platit.
    - `cfRule` (`list[Rule]`): Seznam všech pravidel v oblasti.
      - Pravidlo `0` (`Rule`)
        - `colorScale` (`ColorScale`): Informace nutné pro konstrukci barevných škál (typ podmíněného formátování).
        - `dxf` (`DifferentialStyle`): Údaje o formátování buňky, které se mají použít v případě, že hodnota buňky vyhovuje dané podmínce.
        - `formula`: Uživatelem zadaná hodnota, na základě které se určuje platnost pravidla, typicky pro pravidla *menší než*, *rovná se*, apod.
        - `priority` (`int`): Informace o prioritě tohoto pravidla.
        - `type` (`str`): Typ podmíněného formátování (větší než, mezi, duplicitní hodnoty, ...).
        - ...
      - Pravidlo `1` (`Rule`).
      - ...
    - `rules` (`list[Rule]`): Identický s `cfRule`.
  - ...

Přímá informace o tom, která formátování platí pro danou buňku, uloženy nejsou a je také nutné ji manuálně zjistit. Celý postup je ilustrován na příkladu.

Příklad určení podmíněného formátování

K dispozici je konkrétní buňka datového typu `Cell`, u které je snahou získat informace o tom, zda je podmíněně formátovaná. Také je k dispozici objekt typu `Worksheet`, ze kterého buňka pochází.

Prvním krokem je ověření existence pole, ve kterém má být k dispozici seznam podmíněných formátování, tedy

```
if not ws.conditional_formatting._cf_rules:
    return None
```

Výpis 4: Příklad určení podmíněného formátování buňky [vlastní]

V dalším kroku se získá seznam všech oblastí podmíněného formátování, do kterých buňka patří. Je vysoce pravděpodobné, že buňka patří do více takových oblastí.

```
cell_cf_ranges = []
for c in ws.conditional_formatting._cf_rules:
    for cell_range in c.cells.ranges:
        if cell.coordinate in cell_range:
            cell_cf_ranges.append(c)
```

Výpis 5: Příklad určení podmíněného formátování buňky (pokračování) [vlastní]

V posledním kroku se prochází pravidla z právě získaného seznamu a probíhá ověření toho, zda jsou pro zkoumanou buňku platná. Ověření musí proběhnout pro všechna pravidla, jelikož se nemusí vylučovat – například jedno pravidlo může buňce nastavit tučný text, druhé text obarvit zeleně a třetí buňce nastavit modré pozadí. Jedna buňka může být součástí několika pravidel a všechna mohou být platná současně. K zajištění správného výsledného formátování je tedy nezbytné kromě platných pravidel ukládat i informace o jejich prioritě, které jsou uloženy v `cf.priority` (vyšší číslo znamená vyšší prioritu).

V tomto kroku je také nutná *vlastní* implementace všech funkcí, které kontrolují platnost získaných pravidel, tj. například funkce pro výpočet mediánu, nalezení minima, apod. Vlastní implementace je nezbytná proto, že oblasti podmíněného formátování se občas z neznámých důvodů samovolně rozpadnou na několik menších celků – například z původní oblasti `A1:D5` se stane seznam oblastí `A1:D1`, `A2:D2`, . . . , `A5:D5`, který je nutné uvažovat jako jeden celek.

```
styles = {}
# iterace přes oblasti, do kterých buňka patří
for rng in cell_cf_ranges:
    # iterace přes pravidla v~dané oblasti
    for cf in rng.rules:
        ...
        # jeden z mnoha typů podmíněného formátování
        if cf.type == 'aboveAverage':
            # vlastní funkce pro výpočet průměru v oblasti buněk
            # rng.cells může být 1 oblast (A1:D5) nebo seznam podoblastí
                ([A1:D1, A2:D2, ..., A5:D5]) a nelze předem určit, která
                varianta nastane
            avg = calc_average(ws, rng.cells)
            # kontrola platnosti tohoto pravidla pro zkoumanou buňku
            if cell.value > avg:
                # uložení stylu (== dxf), který odpovídá tomuto
                    podmíněnému formátování
                styles[cf.priority] = cf.dxf
        ...
```

Výpis 6: Příklad určení podmíněného formátování buňky (pokračování)[vlastní]

Po získání všech stylů včetně priority se postupně iteruje seznamem, ve které jsou styly uloženy. Při každé iteraci se provede přiřazení jednotlivých vlastností stylu (tj. objektu `dxf`[3]) ke zkoumané buňce. Během celého cyklu je nutné zajistit aby nedošlo k přepsání již přiřazeného formátování buňky stejným prvkem podmíněného formátování s nižší prioritou.

```
my_cell = CCell()
# zajištění toho, že se formátování použijí ve správném pořadí (od
    nejnižší priority)
styles = dict(sorted(styles.items(), reverse=True))

for priority, style in styles.items():
    # přiřazení stylu podmíněného formátování ke zkoumané buňce
    if style.font.b:
        my_cell.dxf.b = style.font.b
    ...
```

Výpis 7: Příklad určení podmíněného formátování buňky (pokračování) [vlastní]

[3] informace uložené v objektu `dxf` (`DifferentialFormat`) jsou popsány později, viz kap. 3.5.5, konec sekce *Vlastní styly*

Barevné škály

Speciálním typem podmíněného formátování jsou barevné škály, které mění barvu pozadí buněk na základě jejich hodnot. Změna barvy pozadí se provede pro všechny buňky v zadané oblasti, které obsahují pouze čísla. O barevných škálách jsou také dostupné pouze informace nutné pro jejich konstrukci a barvu pozadí každé buňky je nutné doslova dopočítat. Konkrétně jsou dostupné údaje o dvou (třech) barvách uložené v poli `cf.colorScale.color`, které odpovídají významným bodům v oblasti. Rozlišují se dvoustupňové a třístupňové barevné škály — dvoustupňová barevná škála je plně určena dvěma barvami, zatímco k tvorbě třístupňové barevné škály jsou nutné barvy tři.

Dvoustupňové škály jsou konstruovány tak, že vždy první známá barva odpovídá buňkám s nejnižšími číselnými hodnotami v dané oblasti a poslední známá barva odpovídá buňkám s nejvyššími číselnými hodnotami v dané oblasti. K přiřazení barvy pozadí buňce zkoumané je zapotřebí zjistit, v jakém vztahu je její hodnota vůči minimální a maximální hodnotě v dané oblasti. Barvy nejnižší a nejvyšší hodnoty si lze představit jako body v trojrozměrném RGB (kartézském) prostoru. Každou složku výsledné barvy jde získat pohybem po průmětu úsečky, která body zadaných barev spojuje, do příslušné souřadnicové osy. K pohybu po průmětu úsečky je užita *lineární interpolace*, kterou lze matematicky zapsat takto:

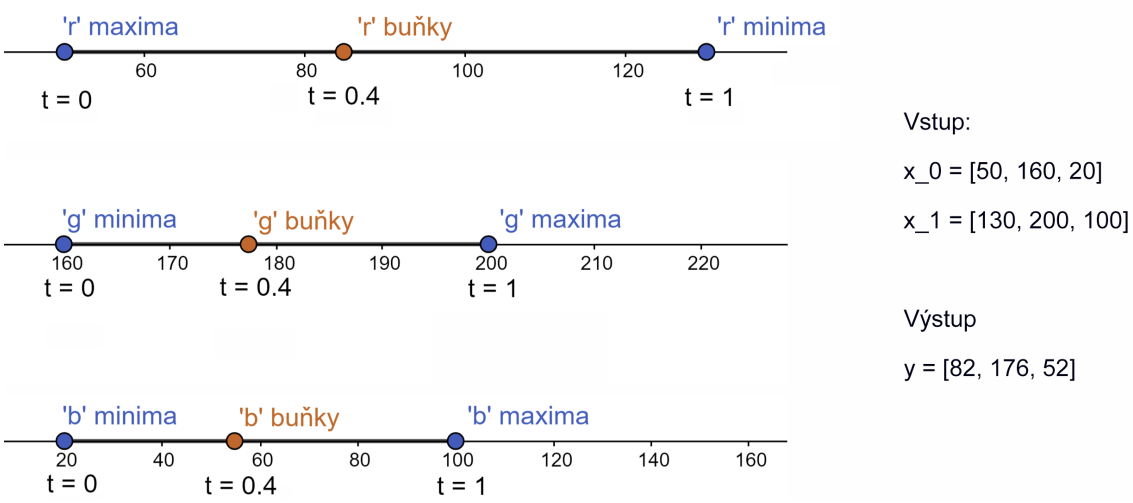
$$y_i = x_{0,i} + t \cdot (x_{1,i} - x_{0,i}) \qquad i \in \{0, 1, 2\} \tag{1}$$

$$t = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad t \in \langle 0, 1 \rangle \tag{2}$$

kde

x_{min}	...	nejnižší hodnota v dané oblasti
x_{max}	...	nejvyšší hodnota v dané oblasti
x	...	hodnota zkoumané buňky
t	...	parametr, vyjadřující polohu v rámci úsečky
y_i	...	složka hledané barvy (neznámá)
$x_{0,i}$	...	složka počáteční barvy
$x_{1,i}$	...	složka koncové barvy

Ve vztahu (1) lze změnou hodnoty parametru t měnit výslednou barvu. Interpolace probíhá ve třech složkách – r, g, a b. Mezi jednotlivými složkami barev je hodnota t vždy stejná, liší se pouze hodnoty x_0 a x_1, viz obr. 9. Hledaná barva se získá interpolací v každé složce barvy pomocí rovnice (1) a složením mezivýsledků do jednoho seznamu. Proces je ukončen převedením barvy do HSL a získáním na nejbližší zobrazitelné barvy.

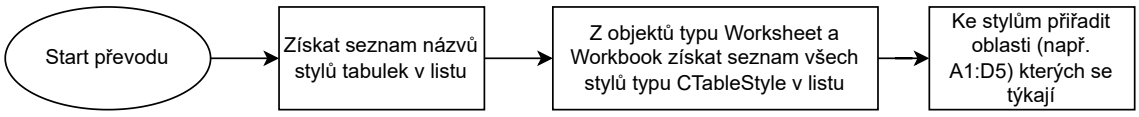


Obr. 9: Náznak interpolace v jednotlivých barevných složkách [vlastní]

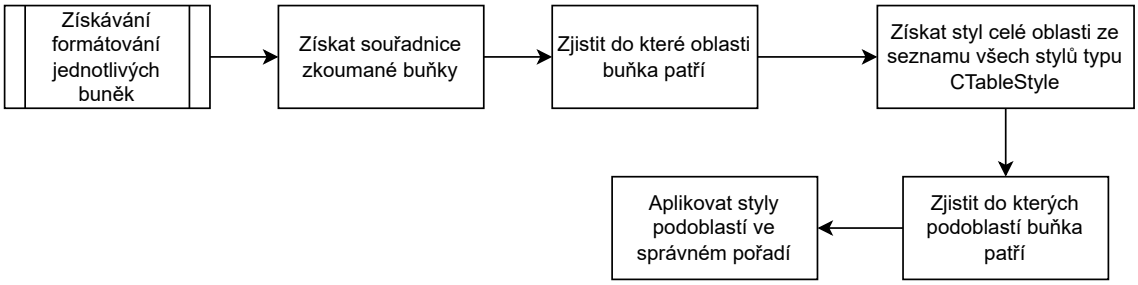
Třístupňová barevná škála je de facto tvořena dvěma plynule navazujícími dvoustupňovými škálami. Barva, ve které dochází k rozdělení škál zpravidla odpovídá hodnotě mediánu (nebo p-kvantilu, apod) a je nutné ji dopočítat s využitím údaje v poli `cf.colorScale.cfvo`. Získání barvy pozadí zkoumané buňky se dá vyřešit převodem na dvoustupňovou barevnou škálu – stačí zjistit, zda je hodnota zkoumané buňky větší než hodnota odpovídající hodnotě barvy rozdělení škál a pokud ano, pak k určení barvy pozadí zkoumané buňky použít první část škály. V opačném případě se musí použít část druhá.

3.5.5 Formátované tabulky

Pod formátovanými tabulkami jsou myšleny oblasti buněk, u kterých bylo v Excelu nastaveno *Formátovat jako tabulku*. Informace o tomto formátování buněk opět nejsou přímo v objektu typu `Cell` a je nutné je získat ručně. Potřebné informace se nachází v objektech typu `Workbook` a `Worksheet`. Celý proces určování stylů buněk z takových oblastí je velmi složitý a propletený, na obr. 10 a 11 je uveden algoritmus, kterým je zisk potřebných informací nutno provést.



Obr. 10: Algoritmus k získání stylů všech oblastí formátovaných jako tabulka [vlastní]



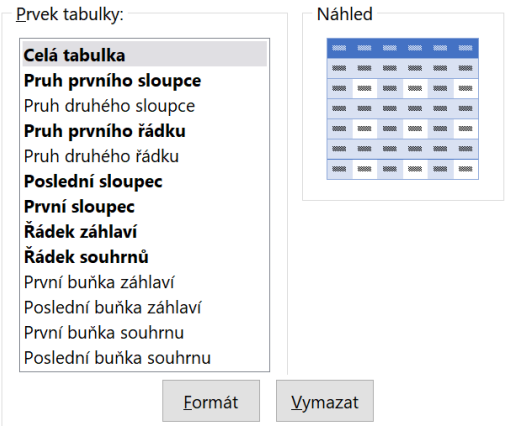
Obr. 11: Algoritmus k získání údajů buněk v oblastech formátovaných jako tabulka [vlastní]

Struktura formátovaných tabulek

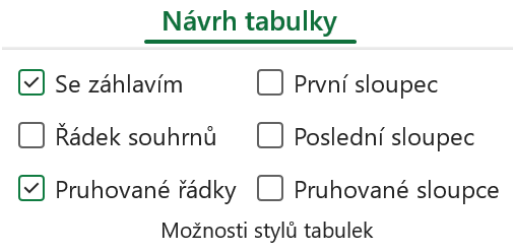
Předtím, než bude možné vyslovit, kde se nachází všechny potřebné údaje, je nutné vysvětlit, jaká je vůbec vnitřní struktura oblastí formátovaných jako tabulka. Tabulky jako takové jsou rozděleny na řadu podoblastí, které lze vidět na obr. 12.

Každá podoblast má úplně vlastní soubor pravidel pro formátování a může obsahovat vše, co je na obr. 14. V rámci jedné tabulky je tak celá řada pravidel pro formátování jedné buňky, podobně jako u podmíněného formátování. I zde je určitá hierarchie a všechna formátování mají mezi sebou nějakou prioritu, která je tentokrát pevně daná. Styly se vždy aplikují v pořadí na na obr. 12 (shora-dolů) a k vytvoření výsledného formátování buňky se postupně složí styly všech podoblastí, do kterých buňka patří.

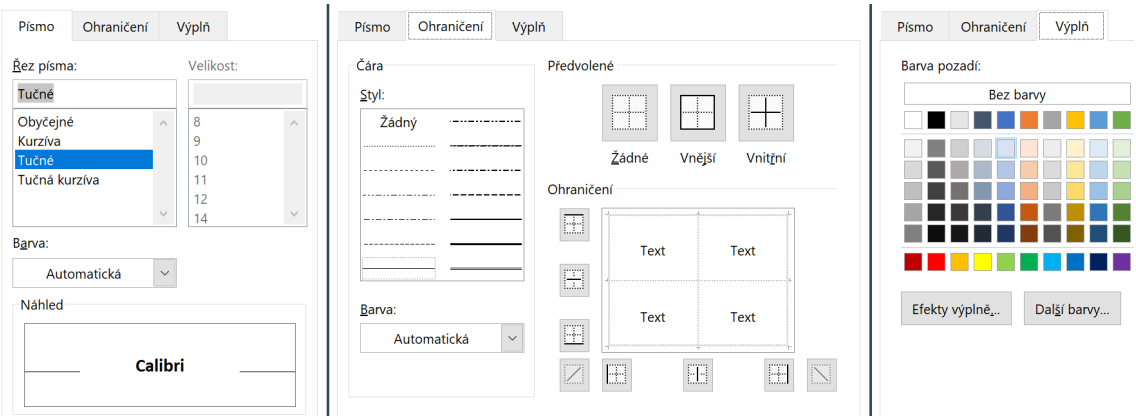
Tabulky jako celek obsahují několik dodatečných nastavení (viz obr. 13), která modifikují vzhled tabulky právě tak, že nepřímo mění seznam podoblastí, do kterých buňky patří.



Obr. 12: Seznam všech podoblastí tabulek [vlastní]



Obr. 13: Dodatečná nastavení celé tabulky [vlastní]



Obr. 14: Možná formátování každé z podoblastí tabulky [vlastní]

Získání seznamu oblastí formátovaných jako tabulka

V každém objektu typu `Worksheet` je k dispozici seznam všech oblastí formátovaných jako tabulka v poli `ws.tables`. Informace v celém seznamu přímo neudávají nic o požadovaném vzhledu tabulky, prakticky pouze sdělují *co* se musí najít. Důležité části struktury seznamu jsou:

- `ws.tables: dict[str, Table]`
  - `key` (`str`): jméno tabulky.
  - `value`: objekt typu `Table`.
    - `ref` (`str`): oblast buněk, kterou tabulka pokrývá.
    - `headerRowCount` (`int`): počet řádků záhlaví (0 nebo 1).
    - `totalsRowCount` (`int`): počet řádků souhrnu (0 nebo 1).
    - `tableStyleInfo`: informace o stylu tabulky.
      - `name` (`str`): jméno stylu.
      - `showColumnStripes` (`bool`): nastavení pruhovaných sloupců.
      - `showFirstColumn` (`bool`): zvýraznění prvního sloupce.
      - `showLastColumn` (`bool`): zvýraznění posledního sloupce.
      - `showRowStripes` (`bool`): nastavení pruhovaných sloupců.
      - ...
    - ...

Pomocí údajů `ref` a `name` se dále získávají dva další seznamy, které obsahují již konkrétní údaje o stylu tabulky a oblasti ke které se styl váže. Z pole `name` je získán styl postupem popsaným v následující kapitole, údaje z `ref` se k takto získanému stylu následně přiřadí.

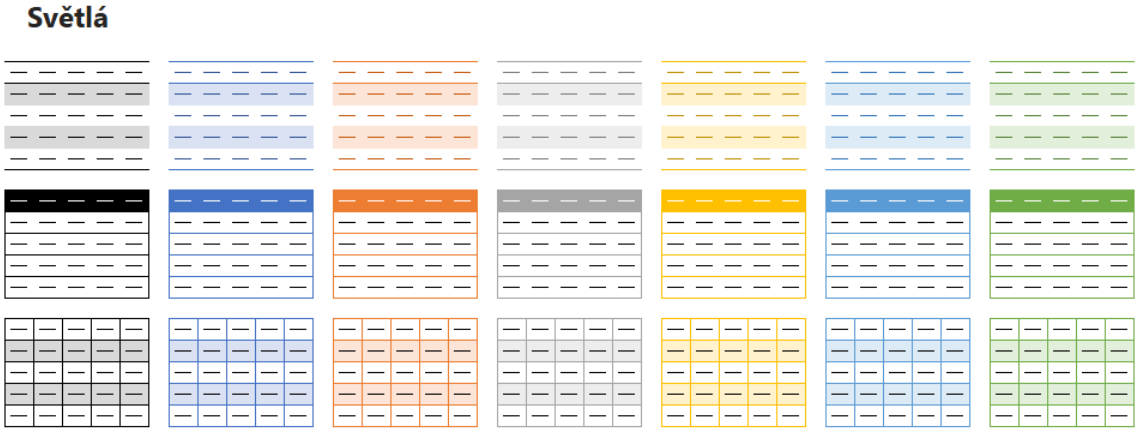
Získání všech stylů tabulek

V prvním kroku (viz obr. 10) se z *názvu* stylu tabulky tvoří objekt vlastního typu `CTableStyle`. Objekt obsahuje konkrétní údaje o formátování jednotlivých podoblastí tabulky spolu s dodatečnými nastaveními tabulky a jeho struktura je:

- `CTableStyle`
  - `whole_table` (`CDxf`): formát podoblasti „Celá tabulka“.
  - `header_row` (`CDxf`): formát podoblasti „Řádek záhlaví“.
  - `total_row` (`CDxf`): formát podoblasti „Řádek souhrnů“.
  - ...
  - `options` (`ETableOptions`): dodatečná nastavení tabulky.
  - ...

Formátování jednotlivých podoblastí se získává dvěma způsoby – záleží, zda se jedná o předdefinovaný nebo vlastní styl. Cílem je v objektu typu `CTableStyle` vyplnit formát všech podoblastí objekty typu `CDxf` a následně hotový objekt typu `CTableStyle` uložit do seznamu všech stylů tabulek v rámci listu.

Předdefinované styly O předdefinovaných stylech tabulek v `openpyxl` nejsou dostupné žádné informace a veškeré údaje o předdefinovaných stylech vychází z normy ECMA-376 (svazek 1, příloha G) [14]. Je předdefinováno celkem 60 stylů tabulky (viz obr. 15), které se dále dělí na tři skupiny – Světlá, Středně sytá a Tmavá, resp. `TableStyleLight`, `TableStyleMedium` a `TableStyleDark`. V rámci každé této skupiny jsou definovány dva až čtyři vzhledy v sedmi barevných provedeních. Do barev předdefinovaných stylů se vždy používají barvy motivu.



Obr. 15: Tři sady (z devíti celkem) předdefinovaných stylů tabulek. Barvy tabulek se mohou lišit, závisí na použitém motivu [vlastní]

Všechny potřebné informace o předdefinovaných stylech formátovaných tabulek je nutné při tvorbě konvertoru *ručně* zjistit a patřičné definice každé podoblasti mít ve svém kódu. Získání definic jednotlivých podoblastí je v Excelu možné například duplikováním předdefinovaného stylu, jelikož se následně v okně programu zobrazí přesný formát jednotlivých podoblastí – ten poté stačí zapsat do objektu `CTableStyle` pro každou podoblast.

Při psaní formátu jednotlivých podoblastí je nutné si uvědomit, že některé předdefinované styly se liší jen barvou. Na obr. 15 je zřetelně vidět, že barvy se periodicky opakují a jejich pořadí odpovídá pořadí barev motivu, které lze vidět na obr. 14. Styl je tedy vhodné psát obecně, v závislosti na jeho pořadí v rámci dostupných barevných provedení. V konvertoru jsou předdefinované styly (typu `CTableStyle`) zapsány dle výpisu 8.

```
# získání barev stylu tabulky pomocí vlastní funkce z:
#   - motivu
#   - pořadí barevného provedení (color_index)
#   - tint (světlost barvy, vždy se používají inkrementy +-0.2)
odd_bg = theme_and_tint_to_ninecolors(wb, color_index, tint=0.8,
    allow_black=True)

# styly podoblastí, definují se zde jen některé prvky formátování
table_style.odd_row = CDxf(bg=odd_bg)
table_style.first_col = CDxf(font_flags=EFontFlags.BOLD)
table_style.odd_col = CDxf(bg=odd_bg)
table_style.last_col = CDxf(font_flags=EFontFlags.BOLD)

# podoblast obsahující okraje.
table_style.total_row = \
    CDxf(border_styles=utils.BS(EBorderStyle.THIN, [2]),
         border_colors=utils.BC(fg_nine, [2]),
         border_priorities=(-11, -11, -9, -11),
         borders=EBorder.TOP,
         font_flags=EFontFlags.BOLD)
...
return table_style
```

Výpis 8: Příklad implementace jednoho předdefinovaného stylu tabulky [vlastní]

Vlastní styly Informace o formátování vlastních stylů formátovaných tabulek se nachází v objektu typu `Workbook` v poli `wb._table_styles` – zde je k nalezení seznam všech vlastních stylů, jehož struktura je:

`wb._table_styles`:

- „`Styl tabulky 1`“
  - `pivot` (`bool`): jedná-li se o kontingenční tabulku.
  - `tableStyleElement`: seznam formátování všech podoblastí.
    - podoblast `00`.
      - `dxfId` (`int`): ID formátování, které má tato podoblast.

            - `size` (`int`): počet sloupců nebo řádků, které daný styl zabírá, viz rozměry pruhovaných řádků a sloupců.
            - `type` (`str`): název podoblasti, o kterou se jedná (celá tabulka, řádek záhlaví, apod.).
            - ...
        * podoblast `01`.
        * zbylé podoblasti z obr. 12.
    - ...
- „`Styl tabulky 2`“.
- ...

V prvním kroku se opět vytvoří objekt typu `CTableStyle` a každé podoblasti je přiřazen prázdný `CDxf`.

Postupně se probíhá všemi podoblastmi vlastního stylu a zjišťuje se jeho `dxfId`. Na základě tohoto ID lze zjistit konkrétní formátování dané podoblasti s využitím seznamu všech `DifferentialFormat` uloženého v objektu typu `Workbook`, resp. v poli `wb._differential_styles`, jehož struktura je:

- `wb._differential_styles`
    - `count` (`int`): počet vlastních formátů.
    - `dxf` (`DifferentialStyleList`): identický se `styles`.
    - `styles` (`DifferentialStyleList`): seznam všech vlastních formátů v celém souboru.
        * `00` (`DifferentialStyle`): objekt obsahující konkrétní informace o formátování.
            - `alignment` (`Alignment`): zarovnání textu.
            - `border` (`Border`): okraje.
            - `fill` (`PatternFill`): barva pozadí.
            - `font` (`Font`): velikost, barva a řez textu.
            - `numFmt` (`NumberFormat`): textový formát číselných hodnot.
            - ...
        * `01` (`DifferentialStyle`): objekt obsahující konkrétní informace o formátování.
        * ...
    - ...

Jednotlivé prvky pole `styles` odpovídají konkrétním `dxfId` v přesném pořadí. Tedy například `styles[12]` odpovídá `dxfId == 12`.

Získaný formát se převede na vlastní typ `CDxf` analogicky jako u ručního formátování buněk, s tím rozdílem, že se údaje nezískávají z objektu typu `Cell`, ale z objektu typu `DifferentialFormat`, a také, že se neukládají informace o formá-

tování buňky, nýbrž o formátování jedné podoblasti tabulky. Nakonec se získaný objekt typu `CDxf` uloží do příslušné podoblasti.

Získání formátování buňky ze stylu tabulky

Celý proces je ilustrován na obr. 11. Nejprve se srovnáním souřadnice buňky a seznamem souřadnic všech oblastí formátovaných jako tabulka určí, zda-li do některé z těchto oblastí buňka patří a pokud ano, tak do které z nich.

Na základě takto získané oblasti souřadnic je získán objekt typu `CTableStyle` z dříve vytvořeného seznamu a zjišťuje se, do kterých podoblastí tabulky buňka patří. Buňka patří do těch podoblastí, které se již z názvů podoblastí zdají logické. Například čtvrtou buňku prvního řádku lze očekávat v podoblastech „První sloupec“, „Pruh prvního sloupce“ a „Pruh druhého řádku“.

S využitím údajů o nastavení tabulky se některé podoblasti mohou vynechat, například je-li vypnut řádek záhlaví, pak formátování od podoblastí obsahující slovo „záhlaví“ nemusí být vůbec uvažováno. K získání seznamu podoblastí je také nezbytné vědět:

- Vypnutím záhlaví se celý řádek záhlaví skryje. Z toho plyne, že pruhované řádky v případě vypnutého záhlaví začínají prvním řádkem oblasti, zatímco u zapnutého záhlaví začínají na řádku druhém.
- Pruhované řádky mohou mít výšku větší než 1. Ke střídání pruhování nemusí docházet pouze v sudých nebo lichých řádcích, ale lze pruhovat různě, například 2 řádky jedním stylem, 5 řádků stylem druhým, 2 řádky prvním, 5 druhým, atd. Z tohoto důvodu se i v Excelu formáty pro oba typy pruhování nejmenují „Pruh sudého řádku“, ale „Pruh prvního řádku, apod. Pruh prvního řádku se vždy použije jako první, po něm se použije pruh druhého řádku, atd.
- Analogická věc platí i pro pruhované sloupce, místo rozdílných výšek jsou ale rozdílné šířky.
- Údaje o rozměrech pruhovaných sloupců i pruhovaných řádků lze získat v poli `wb._table_styles`.

V poslední fázi je postupně na základě priorit aplikováno formátování z objektů typu `CDxf` jednotlivých podoblastí do objektu typu `CDxf` zkoumané buňky, velmi podobně jako u podmíněného formátování.

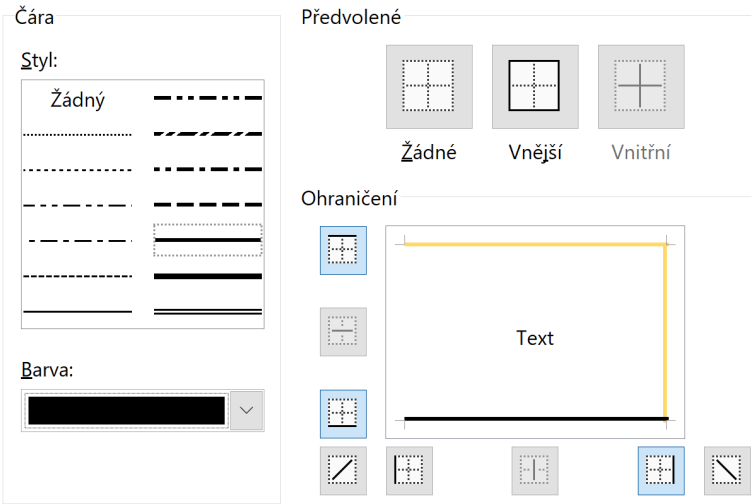
3.5.6 Okraje

Každá buňka má čtyři okraje – horní, spodní, levý a pravý. Ve výstupní tabulce není možné definovat okraje každé buňky zvlášť, naopak, okraje zde musí být definovány jako jedna souvislá čára nebo několik čar známé polohy. Ke správnému zobrazení okrajů je tedy nutné s nimi pracovat jako s čarou ve výstupu.

Oproti ostatním prvkům formátování, která se buďto změní celá, nebo se k formátování zkoumané buňky něco přidá, formátování okrajů buněk se může měnit

jen zčásti. Informace o okrajích buněk se dají získat postupem popsaným v kapitolách 3.5.3, 3.5.4 a 3.5.5. Informace o okrajích se mohou vyskytovat u všech tří typů formátování a jakmile je k dispozici objekt obsahující konkrétní údaje o formátování buňky, pak údaje o okrajích se nachází v poli `border` objektu s informacemi o formátování. Struktura pole `border` je:

- `border` (`StyleProxy`).
  - `bottom` (`Side`): informace o spodním okraji.
    - `border_style` (`str`): typ okraje (tenký, čárkovaný, ...).
    - `color` (`Color`): barva okraje (stejná jako barva textu, pozadí, ...).
    - ...
  - `left` (`Side`): informace o levém okraji.
  - `right` (`Side`): informace o pravém okraji.
  - `top` (`Side`): informace o horním okraji.
  - `horizontal` (`Side`): *vnitřní* vodorovné okraje u oblastí tabulky.
  - `vertical` (`Side`): *vnitřní* svislé okraje u oblastí tabulky.
  - ...



Obr. 16: Vizualizace údajů uložených v `border`

Problém je v tom, že u podmíněného formátování je objektem obsahující konkrétní údaje každé *pravidlo* podmíněného formátování, v oblastech tabulky se jedná o každou *podoblast* (předdefinovanou i vlastní), u ručního formátování se jedná o objekt typu `Cell`. Další překážkou je, že jeden okraj, jakožto čára ve výstupu, může být uložen například jako spodní okraj jedné buňky nebo horní okraj buňky o řádek níže, apod. Celé řešení okrajů spočívá v postupném výběru správného okraje z dostupných dat, a to ve dvou buňkách současně.

Vzájemná priorita okrajů od různých typů formátování

Možné konflikty okrajů od dvou různých typů formátování (např. ručně nastavený okraj versus okraj podmíněně formátované buňky) jsou opět řešeny pomocí jisté priority. K zajištění správného výstupu lze buď u každého okraje při jeho převodu ukládat (vlastní) informace o prioritě nebo provádět převod okrajů od jednotlivých typů formátování ve vhodném pořadí. V rámci typů formátování mají nejmenší prioritu okraje oblastí formátovaných jako tabulka, následují okraje podmíněně formátovaných buněk a nejvyšší priorita je udělena ručně formátovaným buňkám.

Priorita okrajů v rámci jednoho typu formátování

V rámci jednoho typu formátování platí pro *podmíněně formátované okraje* stejná pravidla jako pro ostatní podmíněná formátování (kap. 3.5.4). Priority od jednotlivých pravidel (objektů typu `Rule`) jsou uloženy a dostupné v poli `rank`. Priority jsou zde kladná čísla a vyšší číslo znamená vyšší prioritu.

Ručně formátované okraje se uloží přesně tak, jak jsou ve vstupní tabulce nastaveny a zobrazeny. Opět mají nejvyšší prioritu a musí být řešeny jako poslední.

Okraje v oblastech formátovaných jako tabulka se také řídí prioritou dle obr. 12, informace o této prioritě ale nejsou dostupné přímo a je nutné tyto informace ukládat ručně. K zachování priority mezi podoblastmi je vhodné k prioritám okrajů jednotlivých podoblastí přiřazovat záporná čísla, podoblasti *Celá tabulka* přiřadit co nejmenší číslo (např. -15) a postupně priority zvyšovat až do -1. Tím bude zachována priorita mezi jednotlivými podoblastmi a zároveň zajištěno, že okraje v oblastech tabulek budou mít nižší prioritu než podmíněně formátované okraje.

Algoritmus získání údajů o okrajích

Balíček `tabularray` umožňuje formátovat vodorovné a svislé okraje odděleně. Konvertor této možnosti využívá a do každého objektu `CCell` ukládá informace o vodorovných a svislých okrajích zvlášť. K lepšímu pochopení celého procesu je nutné zopakovat, že buňky v Excel souboru mohou mít až čtyři okraje, zatímco v LaTeX tabulce je nutné pracovat s okrajem jakožto čarou ve výstupu. Situace je řešena tak, že okraje se zjišťují od dvou buněk současně, tedy buď zkoumané buňky a buňky o řádek níže v případě vodorovného okraje nebo zkoumané buňky a buňky o sloupec vpravo v případě okraje svislého.

Získávány jsou informace od všech tří typů formátování a to v daném pořadí. Následně je provedeno vyhodnocení na základě priorit získaných okrajů a poté se uloží vyhodnocený spodní (resp. pravý) okraj do buňky typu `CCell`. Celý proces zachycuje obr. 24 v příloze A.

Tímto přístupem je možné získat informace o okrajích uvnitř celé tabulky s výjimkou vnějších okrajů prvního řádku a prvního sloupce. Ty musí být řešeny zvlášť. Nabízí se jednoduše provést stejnou operaci jako ve zbytku tabulky akorát

o řádek výše (nebo sloupec vlevo). Jedná se o zcela validní přístup v případě, že řádek (sloupec) nad prvním řádkem (sloupcem) zadané oblasti k převodu existuje. K určení existence řádku nad zadanou oblasti stačí pouze ověřit, zda souřadnice první buňky zadané oblasti obsahuje „`1`“. Podobně lze určit existenci sloupce vlevo od zadané oblasti hledáním „`A`“ v souřadnicích zadané oblasti. Je nutné oba případy řešit odděleně. Pokud řádek nebo sloupec nad zadanou oblastí neexistuje, pak příslušné okraje nebudou ve vstupním souboru zobrazeny, tedy, není nezbytné je převádět. Uloženy ale jsou a je zřejmé, kde potřebné údaje nalézt. Kompletní proces získávání okrajů celé tabulky je znázorněn na obr. 25 v příloze A.

Technické informace k okrajům

Problematiku řešení okrajů (získávání dat a zpracování) lze s využitím balíčku `tabularray` kompletně vynechat za předpokladu, že v tabulce budou možné pouze plošně zapnuté nebo plošně vypnuté okraje. Lze zvlášť zapnout nebo vypnout jak horizontální tak i vertikální okraje a dokonce je všechny i formátovat, tj. nastavit tloušťku, typ nebo barvu v každém směru zvlášť. Toto řešení funguje i pro sloučené buňky a je velmi snadné. Potřebné informace se nachází v [5].

Excel nabízí celkem 14 stylů okraje, viz obr. 16, z nichž je možno pomocí `tabularray` v době psaní této práce zobrazit jen některé. Okraje jsou dle syntaxe `tabularray` (kap. 3.8.4) určeny *typem* a *tloušťkou*. Jednotlivé Excel styly okraje jsou tedy definovány ve vlastním datovém typu `EBorderStyle` zachycující typ i tloušťku okraje řetězcem vhodným pro budoucí zápis v `tabularray`.

```
class EBorderStyle(Enum):
    NONE = ''
    DASHED = 'dashed'                # jen typ (čárkovaný)
    MEDIUMDASHED = 'dashed, 0.85pt'  # typ i tloušťka
    # v syntaxi se nesmí nic objevit, ale je nutno typy rozlišovat
    THIN = 't\b'
    # nelze zobrazit v tabularray -> uloženo jako plný okraj
    SLANTDASHDOT = THIN
    ...
```

Výpis 9: Datový typ `EBorderStyle` [vlastní]

3.6 Získání dat uvnitř buněk

Data uložená v buňkách nemusí nutně odpovídat datům zobrazeným. Uložená (zadaná) data lze zobrazit různě, ať už s odlišným počtem desetinných míst, jako zlomek, vědeckou notací, apod. Zobrazená hodnota je daná tzv. *Formátovacím řetězcem*. Kapitola se zabývá strukturou a hledáním těchto formátovacích řetězců.

Obr. 17: Části formátovacího řetězce a jejich význam. [vlastní]

3.6.1 Formátovací řetězce

Formátovací řetězec je vždy uložen v objektu typu `Cell` v poli `cell.number_format` a skládá se z jedné až čtyř částí, oddělených pomocí středníků. Každá část definuje formátování určité skupiny údajů (viz obr. 17). To, kterou skupinu hodnot řídí, závisí na počtu částí celého řetězce a platí:

(a) Pokud má celý řetězec 4 části, pak první část řídí formát kladných čísel nebo datumů, druhá část řídí formát záporných čísel, třetí část definuje formát nulových hodnot a poslední část se týká hodnot obsahujících text.

(b) Pokud má celý řetězec 2 části, pak první část řídí formát nezáporných hodnot (kladné a nula) nebo datumů, druhá část určuje formát záporných hodnot.

(c) Pokud má celý řetězec jen jednu část, pak řídí formát všech typů číselných hodnot.

Uvnitř jednotlivých částí řetězce se nachází zástupné znaky nebo jejich sekvence, pomocí kterých je právě definován formát příslušné části.

Některé znaky se chovají různě v závislosti a tom, zda se nachází před nebo za desetinným oddělovačem nebo podle lokalizace (jazyka) instalace MS Excel. Uvnitř `cell.number_format` se vždy nachází řetězec pro americkou lokalizaci. Seznam všech zástupných znaků je uveden v [18] a v práci jsou přiloženy pouze ty nejdůležitější pro americkou lokalizaci (viz příloha C).

Získání potřebných údajů z formátovacího řetězce

V prvním kroku při formátování obsahu buňky je nutné zjistit, kterou část celého formátovacího řetězce použít. Jelikož jsou vždy odděleny středníky, stačí použít `cell.number_format.split(';')`. Hodnoty buněk jsou v `openpyxl` automaticky ukládány ve správném datovém typu, stačí tedy ověřit, zda je hodnota buňky typu `int` nebo `float` (nebo `datetime`) a poté zkoumat vztah hodnoty buňky vůči nule. V kombinaci s počtem získaných oblastí formátovacího řetězce lze získat jeho správ-

nou část, dále označovanou jako `fmt_to_search`. V případě, že hodnota buňky není číselná stačí ověřit, zda má formátovací řetězec čtyři části. Obsahuje-li formátovací řetězec znak `@`, pak se může zobrazit (spolu s přidanými textovými částmi), v opačném případě musí být hodnota buňky skryta.

Se správnou částí řetězce je vhodné pracovat pomocí tzv. *Regulárních výrazů* (regex). Jedná se o způsob vyhledávání v textu, který nutně nehledá přesnou sekvenci znaků, ale shody hledá na základě obecného předpisu. Použití regulárních výrazů je nezbytné v případě, kdy není známý přesný vstup, ale pouze některé jeho prvky[4], což je přesně tento případ.

V dané části řetězce se musí provést hledání všech „znaků“ nebo sekvencí, které formátují obsah buňky. Z příkladů v tabulkách 19 až 18 v příloze C je zřejmé, že se jednotlivé „znaky“ mohou mezi sebou kombinovat. Při práci s částí formátovacího řetězce lze například sestavit regulární výrazy pro každý možný „znak“ formátovacího řetězce, a zkoumat, které z nich nalezly shodu. Hledáním shod a aplikováním textového formátování ve vhodném pořadí lze docílit správného výsledného formátování textu buňky. Konvertor hledá a aplikuje jednotlivé znaky v pořadí shodném s výpisem 10.

```
# zde se zjistí, které "Znaky" aplikovat
mtc_percent = re.search(r'[#0](\.[0#]+)?\s?%', fmt_to_search)
mtc_fraction = re.search(r'((\?/\?)|(\?\?/\?\?)|(\?{3}/\?{3}))|'
                         r'(\?{3}/[1-9]\d{3})|'
                         r'(\?\?/[1-9]\d)|(\?/[1-9])', fmt_to_search)
mtc_sci = re.search(r'E[+-]0+', fmt_to_search)
mtc_accuracy = re.search(r'[0#]\.((0*#+})|(0+#*))', fmt_to_search)
mtc_sep_thousands = re.search(r'^#,##[0#]', fmt_to_search)
mtc_leading_digits = re.search(r'^\d+\.', fmt_to_search)
mtc_trailing_digits = re.search(r'^[0#]+[1-9]+\.', fmt_to_search)
mtc_currency = re.search(r'\"(\w{1,3})?([\x{20A0}-\x{20B9}])?\"',
    fmt_to_search, re.U)
# a poté se "Znaky" kde byla nalezena shoda aplikují na obsah buňky
```

Výpis 10: Výrazy pro hledání jednotlivých „znaků“ [vlastní]

V dalších fázích algoritmu jsou aplikovány textové formáty od jednotlivých „znaků“ kde byla nalezena shoda, tedy probíhá postupná úprava hodnoty buňky. Výsledná hodnota po aplikování všech „znaků“ je nakonec uložena do pole `disp_value` objektu typu `CCell`.

[4] více o regulárních výrazech lze najít například v [19]

3.6.2 Data (datumy)

Kromě číselných a textových hodnot mohou buňky obsahovat i data (dále nesprávně „datumy“, pro lepší přehlednost). Ty jsou rovnou uloženy jako datový typ `datetime`. Datumům vždy odpovídá první část formátovacího řetězce, dle konvence jsou uloženy jako kladná čísla. Hodnota 1 odpovídá datumu 1. ledna 1900. Ostatní datumy jsou reprezentovány počtem dní od tohoto *základního* data, jeden den odpovídá nárůstu o hodnotu jedna. Tedy např. hodnota 20 odpovídá 20. lednu 1900. Desetinné hodnoty dále upřesňují část dne, např. hodnota 1,5 odpovídá 1. lednu 1900 12:00:00. Datumy starší 1. ledna 1900 se ukládají jako obyčejný řetězec, a nepodporují další formátování.

Rozlišuje se několik zástupných znaků, které umožní datum formátovat – `h` (hodiny), `m` (měsíc nebo minuta, dle kontextu), `d` (den), `yy` (rok), `h` (hodina), `s` (sekunda). Jejich význam je zcela intuitivní a podrobné informace lze najít v [18]. Za zmínku stojí pouze fakt, že opakováním jednoho zástupného znaku lze postupně získávat delší výstupní řetězec, například `m` → `1`, `mm` → `01`, `mmm` → `led` a `mmmm` → `leden`

Samotné řešení formátování datumů je provedeno podobně jako v řešení znaků v předchozí kapitole. Snahou je zjistit, které zástupné znaky se v části formátovacího řetězce vyskytují, v jakém pořadí a jak se opakují. Na základě toho lze vybrat zástupné znaky z knihovny `datetime`, které naformátují hodnotu buňky dle části formátovacího řetězce.

Je zřejmé, že jednotlivé zástupné znaky se budou moci ve formátovacím řetězci vyskytovat v různém pořadí, které je nutné zachovat. Datumy musí být vždy něčím odděleny a v prvním kroku se provede rozdělení části formátovacího řetězce v místech, kde se vyskytuje jeden nebo více netextových znaků (závorky, lomítka, interpunkce, . . .), pomocí regulárního výrazu, viz výpis 11.

```
# vstup: fmt_to_search = 'dddd\\ d\\.\\ mmmm\\ yyyy'
fmt_to_search = fmt_to_search.replace('\\', '')

# rozdělení v místě těchto znaků. dělicí znaky se také uloží
parts = re.split(r'([-_/\\+.,*\s!?\'\"|<>]+)', fmt_to_search)
```

Výpis 11: Rozdělení formátovacího řetězce datumu [vlastní]

Následuje iterace všemi prvky seznamu `parts`. Během každé iterace se hledá shoda prvku z `parts` s některým Excel zástupným znakem. V případě, že je shoda nalezena, vybere se vhodný zástupný znak z knihovny `datetime`, který se přidá do seznamu vně cyklu. V případě, že shoda není nalezena, se jedná o netextový znak, který slouží jako oddělovač a který je do seznamu `parts` uložen beze změn. Po do-

končení iterace je tedy k dispozici seznam ve kterém byly převedeny zástupné znaky z Excelu na zástupné znaky z knihovny `datetime` a ve kterém je zachováno oddělení částí datumu. Prvky seznamu se převedou na řetězec, který slouží jako vstup do funkce `strftime` z knihovny `datetime`. Aplikováním této funkce na uloženou hodnotu buňky je získán potřebný formát datumu.

```
# sem se ukládají zástupné znaky z knihovny 'datetime'
fmt_string_parts = []

for part in parts:
    # 'm' může být měsíc i minuta dle kontextu.
    # je-li kolem 'hh' nebo 'ss', pak je to minuta, jinak měsíc.
    mtc_month = re.search(r'm{1,4}', part)
    maybe_minute = re.search(r'((hh?\W{0,2}mm?\W{0,2}ss?)|'
                             r'(hh?\W{0,2}mm?)|(mm?\W{0,2}ss?))',
                             fmt_to_search)
    mtc_day = re.search(r'd{1,4}', part)
    mtc_year = re.search('yy(yy)?', part)
    ...
    # oddelovač, jedná se o prvek v 'parts' s indexem o 1 větší.
    # nemělo by se stát, že v něm bude nalezena shoda s některým znakem
    sep = ''
    if i < len(parts) - 2 and re.search(r'\w+', parts[i + 1]) is None:
        sep = parts[i + 1]
    else:
        sep = ' '

    if mtc_day is not None:
        d_len = len(mtc_day.group())
            if d_len == 2:
                # uložení. Excel znak -> znak pro knihovnu datetime
                fmt_string_parts.append('%d' + sep)
            ...
            elif d_len == 4: ...
    elif mtc_year is not None: ...
    ...
# tvorba výsledného řetězce pro datetime
dt_fmt_str = ''.join(fmt_string_parts)
# formátování hodnoty buňky
return cell.value.strftime(dt_fmt_str)
```

Výpis 12: Aplikování formátu datumů [vlastní]

V řadě případů se stává, že v knihovně `datetime` není potřebný zástupný znak. Vždy se jedná o případ, ve kterém se zástupný znak opakuje pouze jednou, což odpovídá číselným hodnotám bez počátečních nul. Daný problém lze řešit tak, že se do seznamu ukládaných zástupných znaků před problematický zástupný znak přidá písmeno, které se ve výstupu nemůže objevit (například `%d` → `X%d`). Tento znak knihovna `datetime` vnímá jako „oddělovač“ a ve výstupním řetězci jej ponechá. Poté stačí nahradit všechny `'X0'` prázdnými řetězci.

3.7 Společné vlastnosti

Získání vlastností jednotlivých buněk je pouze zlomek celého procesu převodu tabulky. S využitím balíčku `tabularray` lze zapsat stejné prvky formátování buněk v rámci celého řádku nebo celého sloupce jedním příkazem. Nemusí se přitom jednat o všechny prvky formátování, buňky nemusí být identické. Stačí, aby měly alespoň něco společného. V této kapitole je popsáno, jak jsou získané údaje zpracovány tak, aby z nich bylo možno vytvořit kompaktnější zápis.

Zpracování společných vlastností zčásti probíhá současně se získávám formátování jednotlivých buněk. Důvodem je minimalizace počtu cyklů, které konvertor během převodu vykoná. K ukládání společných vlastností byly vytvořeny dva vlastní datové typy `CRow` a `CCol`, které reprezentují řádek a sloupec výstupní tabulky. Oba datové typy mají velmi podobnou strukturu:

- `CRow`
  - `index` (`int`): pořádí v rámci výstupní tabulky.
  - `members` (`list[CCell]`): seznam s odkazy na jednotlivé buňky, které by měl obsahovat.
  - `shared_bgcolor` (`EColor`): společná barva pozadí buněk řádku.
  - `shared_fgcolor` (`EColor`): společná barva textu buněk řádku.
  - `shared_font_size` (`EFontSize`): společná LaTeX velikost písma.
  - `shared_font_flags` (`EFontFlags`): společné vlastnosti textu.
  - `shared_text_halign` (`EHAlign`): společné horizontální zarovnání textu.
  - `shared_text_valig` (`EVAlign`): společné vertikální zarovnání textu.
  - `vsize` (`float`): výška řádku v Excel jednotkách (u `CCol` je zde `hsize` a jedná se o šířku sloupce v Excel jednotkách).
  - `is_converted` (`EConvState`): informace o tom, zda se řádek shoduje s jiným (kap. 3.8.2).
  - `whole_borders_clr` (`EColor`): barva společného spodního okraje buněk řádku.
  - `whole_borders` (`EBorderStyle`) = styl společného spodního okraje buněk řádku.

Po získání formátování každé buňky a uložení získaných dat do objektu typu `CCell` je tento objekt uložen do seznamu `members` buněk daného řádku (sloupce) typu `CRow` (`CCol`).

```
# seznam všech sloupců CCol, řádků CRow
all_cols = []
all_rows = []

# iterace po řádcích vstupní tabulky, iter_rows je funkce z openpyxl,
    minx, ..., maxy jsou XY souřadnice převáděné oblasti buněk
for row in ws.iter_rows(max_row=maxy, max_col=maxx, min_row=miny,
    min_col=minx):
    # zavedení předpokladu, že všechny typy formátování jsou společné
    what_is_shared_row = {'shared_fgcolor': True, 'shared_bgcolor':
        True, 'shared_font_size': True, 'shared_font_flags': True,
        'shared_text_halign': True, 'shared_text_valign': True,
        'whole_borders': True}
    my_row = CRow()
    # pořadí aktuálního sloupce
    cur_col = 1

    # iterace mezi buňkami v řádku
    for cell in row:
        # získání informací o formátování, náplň kapitol 3.5 a 3.6
        my_cell = get_formatting(cell)
        # uložení do řádku
        my_row.members.append(my_cell)
        # uložení do sloupce (chybný, ale lépe pochopitelný zápis)
        all_cols[cur_col - 1].append(my_cell)
        cur_col += 1
    # zde začíná další výpis
    cell_to_compare = my_row.members[0]
```

Výpis 13: Ukládání buněk do vlastních řádků a sloupců [vlastní]

Během získávání dat o formátování buněk je buňkami vstupního souboru iterováno po řádcích a po dokončení iterace v rámci jednoho řádku nastává proces, ve kterém jsou vyhodnocovány společné vlastnosti buněk v rámci tohoto *řádku*. Tento proces je zachycen ve výpise. 14.

Všechny buňky daného řádku jsou porovnávány s první buňkou tohoto řádku, dále označovanou jako *porovnávací buňka*. Nejprve je zaveden předpoklad, že všechny prvky formátování jsou v daném řádku totožné, pomocí seznamu společných vlast-

ností, který obsahuje pouze logické hodnoty. U každé buňky se postupně zjišťuje, které formátovací prvky má stejné jako buňka porovnávací a v případě, že se některý prvek lišil, je příslušný prvek v seznamu logických hodnot nastaven na `False` a u dalších buněk se tento formátovací prvek dále neporovnává.

Jakmile jsou porovnány všechny buňky daného řádku s buňkou porovnávací, lze s využitím seznamu společných formátování nastavit tyto prvky objektu typu `CRow`, který je následně uložen do seznamu všech řádků výstupní tabulky.

```
# stále probíhá uvnitř cyklu 'ws.iter_rows' z předchozího výpisu
# porovnávací buňka
cell_to_compare = my_row.members[0]

# porovnávání jednotlivých typů formátování v rámci řádku.
for my_cell in my_row.members:
    if what_is_shared_row['shared_fgcolor'] \
        and my_cell.dxf.fgcolor != cell_to_compare.dxf.fgcolor:
        what_is_shared_row['shared_fgcolor'] = False
    ...

# nastavení společných formátování do objektu CRow
if what_is_shared_row['shared_fgcolor']:
    my_row.shared_fgcolor = cell_to_compare.dxf.fgcolor
...

# uložení do seznamu všech sloupců
all_rows.append((my_row, what_is_shared_row))
```

Výpis 14: Společné vlastnosti řádků [vlastní]

Objekty typu `CCol` jsou do podobného seznamu ukládány v okamžiku, kdy je během získávání formátování jednotlivých buněk započat nový sloupec, viz výpis 13. Při přidání do takového seznamu obsahují pouze odkaz na příslušnou buňku typu `CCell` a během iterace postupně přibývají odkazy na další buňky.

V okamžiku, kdy je dokončena iterace v celé vstupní tabulce po řádcích teprve následuje velmi podobný cyklus porovnávání buněk, jako v případě společných vlastností řádků. Zde dochází k porovnávání buněk jednotlivých sloupců s první buňkou daného sloupce. Získané společné vlastnosti jednoho sloupce jsou následně zapsány do objektu typu `CCol` a z buněk, na které objekt odkazuje jsou tyto vlastnosti vymazány.

```
for my_col in all_cols:
    ... # získání společných vlastností a uložení do objektů CCol
    # smazání společných vlastností buňkám
    for my_cell in my_col.members:
        if what_is_shared_col['shared_fgcolor']:
            # EColor je vlastní datový typ na bázi Enum,
            # jeho hodnota NOT_SET je analogií None
            my_cell.dxf.fgcolor = EColor.NOT_SET
            ...
```

Výpis 15: Smazání společných vlastností v jednotlivých buňkách sloupce [vlastní]

V posledním kroku se zpětně iteruje seznamem všech řádků a z buněk, na které řádky odkazují, jsou postupně vymazány společné vlastnosti v rámci řádků, podobně jako při mazání společných vlastností z buněk v rámci sloupce. Tento krok je nezbytné provést až na závěr, jinak by nebylo možné předvídatelně získávat společné vlastnosti v rámci sloupce.

Z podstaty celého procesu získávání společných vlastností by mělo být zřejmé, že údaje, které buňky nemají společné v rámci řádku nebo sloupce v nich zůstanou nadále uloženy a není nutné je nijak ošetřovat. Buňky nemusí mít společný kompletní formát, ale pouze některé prvky, a právě ty jsou uloženy v objektech typu `CRow` resp. `CCol`.

Získávání společných okrajů probíhá úplně stejně jako u zbylých typů formátování a dokonce zároveň s nimi. Zjišťuje se pouze, zda je daný okraj po celé délce (výšce) tabulky stejný a pokud ano, jsou informace o stylu okraje uloženy do objektu typu `CRow` (horizontální okraj) resp. `CCol` (svislý okraj).

3.8 Tvorba hlavičky výstupu

Následující trojice kapitol se zabývá jednotlivými fázemi tvorby výstupního řetězce, od tvorby jednotlivých prvků hlavičky, přes vytvoření těla až po umístění celého výstupu do vhodného LaTeX prostředí. Celý proces tvorby výstupního řetězce zachycuje obr. 27 v příloze A a přestože se zdá jednoduchý, jedná se o náročné finále celého převodu.

Tato kapitola je věnována pouze tvorbě hlavičky výstupu, tedy té části výstupní tabulky, obsahující veškeré údaje o formátování buněk. Uvnitř hlavičky se vyskytují řetězce definující formát jednotlivých „prvků“ výstupní tabulky – řádků, sloupců, buněk i okrajů. V kapitole bude zvlášť popsán postup, kterým je tvořen každý typ prvku, lze předem prozradit, že postupy si budou podobné, v případě řádků a sloupců jsou dokonce totožné. V hlavičce se také může vyskytovat „legenda“, tj. popis a syntaxe jednotlivých části hlavičky. Tvorba legendy je podrobně

popsána v závěru kapitoly a je nutné hned zmínit, že probíhá současně s tvorbou ostatních prvků hlavičky viz obr. 27 v příloze A.

Po získání společných vlastností buněk v rámci řádků nebo sloupců jsou všechny objekty reprezentující řádky i sloupce výstupní tabulky uloženy ve dvou seznamech, na které je během kapitoly odkazováno slovy „*seznam všech řádků*“ resp. „*seznam všech sloupců*“. Jedná se o seznamy, které jsou výstupem kapitoly 3.7, ve výpisech (i kódu konvertoru) označované jako `all_rows` a `all_cols`. Z objektů v těchto dvou seznamech, tj. celých řádků nebo celých sloupců, jsou postupně převáděny uložené informace do formy řetězce vhodného pro `tabularray`. Aby byla hlavička ve výsledku co možno nejkratší, je během její tvorby snaha zápis totožných prvků zjednodušit, proces zjednodušení je však zcela dobrovolný a jeho vynechání na funkčnosti výstupu nic nemění.

3.8.1 Parametry colspec a rowspec

Parametry `colspec` a `rowspec` slouží ke kompaktnímu zápisu zarovnání a velikosti celých sloupců resp. řádků, viz výpis 16.

```
% zarovnání ve vodorov. směru, relativní šířka sloupců (poměry šířek)
colspec = {X[5, r] X[3, c] X[4, l]},
% zarovnání ve vodorovném směru, bez zadání šířky
colspec = {llcrlc},
% zarovnání ve svislém směru, absolutní výška řádků
rowspec = {hhm{15.3pt}ffm{8pt}}
```

Výpis 16: Použité varianty `colspec` a `rowspec` [vlastní]

Parametr `colspec` lze chápat jako rozšíření hlavičky klasického prostředí `tabular` – je možné zadávat stejné parametry, ale s o něco vyššími možnostmi nastavení [5]. Parametr `rowspec` vyjadřuje totéž jako `colspec`, akorát ve směru řádků.

Konvertor využívá pouze malou část z řady dostupných možností `colspec` a `rowspec`, konkrétně jeden směr zarovnání textu, sloupce „typu `X`“ a výšku řádků viz tab. 7 a 8.

Ze syntaxe obou parametrů je zřejmé, že musí být uvedeno zarovnání *všech* řádků resp. všech sloupců. V případě, že je nutné některý sloupec (řádek) vynechat, použije se obecný typ sloupce (řádku) „`Q`“, který neprovádí změnu zarovnání a udává pouze existenci daného sloupce (řádku). U *sloupce* „typu `Q`“ lze nastavit jak

[5] Vše je vidět v [5]. Paradoxně část, která je v [5] `colspec` a `rowspec` věnována, obsahuje pouze minimum informací a většinu možných parametrů lze vidět na příkladech uvnitř druhé poloviny úvodní kapitoly. Jelikož v těchto příkladech `colspec` bývá jediným parametrem, nebývá explicitně uvedeno, že se jedná o něj.

Tab. 7: Údaje objevující se v `colspec` u výstupu z konvertoru [vlastní]

`l`	zarovnání vlevo
`r`	zarovnání vpravo
`c`	zarovnání na střed (horizontálně)
`Q`	obecný typ sloupce („placeholder“)
`X[rw, halign]`	sloupec s relativní šířkou `rw` a zarovnáním `halign` (tj. zarovnání `l`, `r` nebo `c`)

Tab. 8: Údaje objevující se v `rowspec` u výstupu z konvertoru [vlastní]

`h{vk}`	zarovnání nahoru, výška řádku `vk`
`f{vk}`	zarovnání dolů, výška řádku `vk`
`m{vk}`	zarovnání na střed (vertikálně), výška řádku `vk`
`Q[vk]`	obecný typ řádku („bez zarovnání“), výška řádku `vk`

zarovnání, tak absolutní rozměr, konvertor ani jedné možnosti nevyužívá. S *řádkem* „typu `Q`“ lze zacházet stejně, konvertor zde přidává i informaci o výšce.

Samotná tvorba obou parametrů probíhá tak, že je iterováno seznamem všech sloupců (`colspec`) resp. řádků (`rowspec`) a na základě hodnot uložených v objektech typu `CCol` (`CRow`) daného seznamu je postupně skládán jeden velký řetězec obsahující prvky z tab. 7 a 8.

Nemá-li sloupec (řádek) stejně zarovnané buňky, pak je hledáno nejčastější zarovnání v rámci tohoto sloupce (řádku) pomocí objektu knihovny `Counter` a jeho funkce `most_common`. Rozměry řádků i sloupců se použijí přesně tak, jak byly zjištěny v kap. 3.5.2. U sloupců se uložená šířka přímo vkládá jako parametr `rw` sloupce „typu `X`“, u řádků se uložená výška vkládá jako parametr `vk` u všech typů zarovnání (pokud se nejedná o výchozí výšku 15 bodů).

Údaje uložené v parametrech `colspec` i `rowspec` lze přepsat. V případě, kdy je vytvořen např. „`colspec = {rrr}`“, není výstupní tabulka omezena pouze na text zarovnaný vpravo. Umístěním `colspec` do horní části hlavičky jej bude moci jakékoli další horizontální zarovnání částečně přepsat. Parametry `colspec` a `rowspec` tedy slouží jako takové celkové zarovnání, které se aplikuje v případě, že není nastaveno žádné jiné.

Formátovací řetězce dalších prvků hlavičky, zejména samostatných buněk, je vhodné tvořit tak, aby neopakovaly údaje vyskytující se v `colspec` a `rowspec`. To je možné buďto mazáním údajů o zarovnání příslušných buněk při tvorbě `colspec` (`rowspec`) nebo kontrolou v pozdější fázi převodu.

3.8.2 Řetězce celých řádků

Shodné řádky se v `tabularray` zapíší dle komentáře ve výpise 17.

```
% formát vybraných řádků
% row{zjednodušený_index} = {formátovací řetězec}
% formátovací řetězec = 'valign, výška, fg=barva_textu,
    bg=barva_pozadí, font=\font_cmds, cmd={\latex_cmd}'
row{1, 4-5} = {m, fg=red4, bg=red8, font=\small\bfseries\itshape},

% formát všech řádků
% rows = {formátovací řetězec}
```

Výpis 17: Syntaxe celých řádků uvnitř hlavičky (`valign` je svislé zarovnání textu, viz tab. 8) [vlastní]

Během tvorby řádků uvnitř hlavičky jsou nejprve získány indexy opakujících se řádků, které jsou následně zjednodušeny na *zjednodušený index*, ke kterému je vytvořen *formátovací řetězec* udávající samotný formát prvku zápisem vhodným pro `tabularray`.

Samozřejmě může nastat i situace, že řádek nemá s jiným nic společného, nebo není žádoucí tvořit zjednodušené indexy. V tom případě stačí místo zjednodušeného indexu použít skutečný index řádku a pouze vytvořit formátovací řetězec.

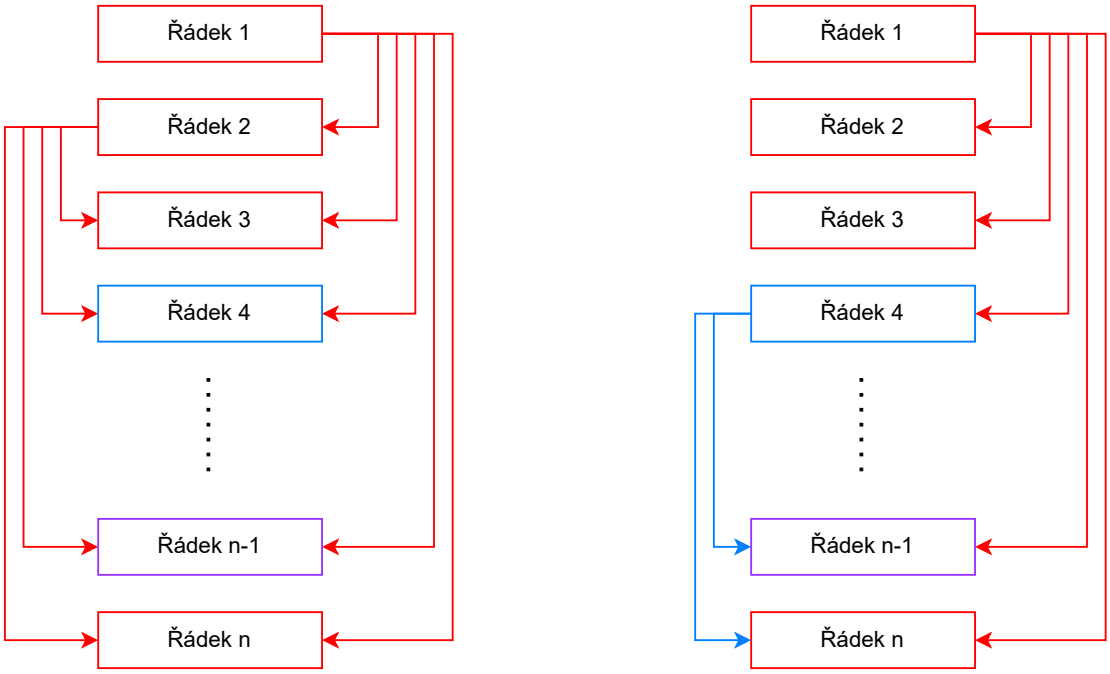
Získání indexů opakujících se řádků

Informace o společném formátování celých řádků jsou uloženy v seznamu všech řádků `all_rows`. Cílem je v tomto seznamu najít objekty, které jsou shodné, ale liší se svým indexem (pořadím ve výstupní tabulce).

Hledání shodných řádků je prováděno způsobem znázorněným ve výpise 18 a na obr. 18. Postupně se prochází seznam všech řádků a během jedné iterace je porovnán i-tý řádek s řádky následujícími ($i+1$ až n). Je uložen seznam těch řádků, které se shodují s i-tým řádkem a do objektů shodných řádků je uložena informace o tom, že se s nějakým řádkem shodují.

V nejhorším případě dochází k postupnému porovnání prvního řádku se druhým, prvního se třetím, až prvního s n; druhého řádku se třetím, druhého se čtvrtým, až druhého s n, atd. až do porovnání řádku $n-1$ s n.

S využitím informace o již nalezených shodných řádcích lze některé iterace i až n zcela vynechat, což ukazuje obr. 18 na konkrétním příkladu. Zde nejprve musela být provedena počáteční iterace vnějšího cyklu, tj. porovnání prvního řádku s řádky 2 až n. V této iteraci se zjistilo, že řádky 2, 3 a n jsou shodné s prvním řádkem. V další iteraci vnějšího cyklu tak šlo řádky 2 a 3 přeskočit a porovnávat až řádek 4 s řádky 5 až n.



Obr. 18: Princip hledání shodných řádků. Vlevo je průběh bez ukládání informací o shodných řádcích, vpravo průběh včetně ukládání těchto informací. V obou případech jsou znázorněny pouze dvě iterace vnějšího cyklu (proměnná i). [vlastní]

```
# cyklus pro získání indexů shodných řádků (u sloupců je totožný)
# all_rows je seznam všech řádků výstupní tabulky (list[CRow])
for i in all_rows:
    # už se s nějakým shodoval, přeskakujeme
    if i.is_converted:
        continue

    # zde se ukládají indexy řádků, se kterými se i-tý řadek shoduje.
    # určitě se bude shodovat sám se sebou.
    matches = [i]

    # zde dochází k porovnání i-tého řádku s řádky i+1 až n
    for j in all_rows[i.index:]:
        if i == j:
            matches.append(j)
            j.is_converted = True # informace o shodnosti

    # shodné objekty se setřídí a vytáhne se index
    matches = sorted(matches, key=lambda x: x.index)
    mtc_indexes = sorted([j.index for j in matches])
```

Výpis 18: Získání indexů totožných řádků [vlastní]

Tvorba zjednodušeného indexu řádku

Balíček `tabularray` umožňuje indexy shodných objektů zadat hned několika způsoby:

1. vyjmenované: 1, 2, 5, 9, 11.
2. rozsah: 1-8.
3. sudé: even.
4. liché: odd.
5. kombinace: 1, 2, 5-9; even, 3, 9.
6. všechny: pomocí (mírně) odlišného příkazu.

Seznam řádků, které se s aktuálním (i-tým) řádkem shodují je tedy v dalším kroku zjednodušen na *zjednodušený řetězec*.

Dopředu není známé, jak se řádky opakují a musí se rozhodnout mezi třemi způsoby. Buď jsou všechny řádky shodné, nebo se zde vyskytují všechny liché resp. všechny sudé řádky, a poslední možností je obecná posloupnost.

V případě, že se shodují všechny řádky, je provedeno zjednodušení na řetězec „`all`“, protože požadují mírně zvláštní zacházení při tvorbě finálního výstupu.

Zbylé dva případy jsou řešeny současně. V prvním kroku je se shodnými indexy zacházeno jako s obecným případem a je provedeno zjednodušení na zápis typu „`1-2, 4, 6, 8-10`“, apod. V dalším kroku se provede zjednodušení na zápis využívající sudá a lichá čísla, „`even, 1, 9`“, a jsou porovnány délky obou přístupů – zvolen je kratší výstup.

Konkrétní algoritmus zjednodušení obecné posloupnosti je převzat z [21].

U zjednodušení pomocí sudých a lichých čísel je situace snadná. Vysvětlení proběhne pouze pro zjednodušení na sudá čísla, u lichých čísel je postup analogický. Stačí seznam shodných indexů rozdělit na sudá čísla a „ostatní“. Je-li množina (tzv. `set`) získaných sudých čísel shodná s množinou všech sudých od 1 do n, pak je vrácen řetězec spojující „`even`“ a ostatní čísla. Hodnota n je v tomto případě celkový počet řádků výstupní tabulky.

Tvorba formátovacího řetězce řádku

Tvorba formátovacího řetězce řádku probíhá bezprostředně po vytvoření zjednodušeného indexu. Údaje ze seznamu všech řádků se pouze naformátují pomocí f-string tak, aby splňovaly syntaxi `tabularray`, tj. dle výpisu 17.

U daného objektu ze seznamu všech řádků se postupně prochází jednotlivá pole objektu a kontroluje se, zda obsahuje hodnotu, která není výchozí. Pokud obsahuje, je k hodnotě přidána nutná syntaxe (např. „`fg=`“), poté je připojena k formátovacímu řetězci a oddělena čárkou. Jelikož je vše uloženo v datovém typu na bázi `Enum`, je skutečná uložená hodnota získána pomocí funkce `value`. V případě použití obyčejných datových typů (`str`, apod.) se podobný zápis neprovádí.

```
# zjednodušený index
# mtc_indexes (list[int]) -> simpl_index (str)
...
# formátovací řetězec
for my_row in all_rows
    # kontrola nastavení
    if settings.conv.color.enable \
            and settings.conv.color.bg \
            # kontrola nevýchozí hodnoty
            and my_row.shared_bgcolor not in [EColor.NOT_SET,
                EColor.WHITE]:
        # přidání do formátovacího řetězce
        fmt_str += f'bg={my_row.shared_bgcolor.value}, '
        # zapnutí prvku legendy
        settings.legend._bg = True
    ...
```

Výpis 19: Tvorba formátovacího řetězce řádku hlavičky [vlastní]

Uložení řetězců řádků hlavičky

Formátovací řetězec je spolu se zjednodušeným indexem uložen do `dictionary` tak, že zjednodušený index je klíč a formátovací řetězec hodnota. V kódu konvertoru je označen názvem `all_rows_ltx` a ve finále zde budou uloženy dvojice zjednodušených indexů jim příslušných výsledných řetězců.

Výsledný řetězec obsahuje celý zápis pro formátování jednoho řádku, tedy jak klíčové slovo `row`, tak zjednodušený index a formátovací řetězec včetně všech potřebných složených závorek. Tvorba výsledného řetězce probíhá pomocí f-string ve výpise 20 kde znak pro složenou závorku ve výstupu je zapsán pomocí dvojice složených závorek uvnitř f-string. Tedy pro zobrazení řetězce „`{a`“ je nutný f-string obsahující zápis „`{{a`“.

```
# simpl_index != 'all' (některé řádky)
f'\t\t row{{{simpl_index}}} = {{{fmt_str}}}'

# simpl_index == 'all' (všechny řádky)
f'\t\t rows = {{{fmt_str}}}'
```

Výpis 20: f-string pro tvorbu výsledného řetězce řádku hlavičky [vlastní]

3.8.3 Řetězce celých sloupců

Shodné řádky se v `tabularray` zapíší dle komentáře ve výpise 17.

```
% formát vybraných sloupců
% column{zjednodušený_index} = {formátovací řetězec}
% formátovací řetězec = 'halign, výška, fg=barva_textu,
    bg=barva_pozadí, font=\font_cmds, cmd={\latex_cmd}'
column{odd, 2} = {r, fg=red4, bg=brown6, cmd={\underline}},
```

Výpis 21: Syntaxe celých sloupců uvnitř hlavičky (`halign` je vodorovné zarovnání textu, viz tab. 7) [vlastní]

Tvorba formátovacího řetězce i zjednodušeného indexu celých sloupců, je skoro totožná s procesem tvorby formátovacího řetězce celých řádků.

Celý proces tvorby řádků proto umožňuje práci jak s objekty typu `CRow` tak i s objekty typu `CCol`. Prakticky jediným rozdílem mezi oběma procesy je práce s jiným vstupním seznamem (`all_cols` místo `all_rows`) a klíčové slovo použité při tvorbě výsledného řetězce (`column` místo `row`).

3.8.4 Řetězce okrajů

Okraje mají v tabulkách `tabularray` specifický zápis, viz výpis 22.

```
% hline{zjednodušený_index} = {indexy_segmentu}{formát_segmentu}
% formát_segmentu = barva, tloušťka, typ
% horizontální okraj, vyskytuje se pouze ve sloupcích 1 až 3
hline{2-9} = {1-3}{0.5pt, azure8, dotted},

% vertikální okraj, po celé výšce tabulky stejný
vline{3, 6} = {red8}

% '{indexy_segmentu}{formát_segmentu}' == řetězec segmentu
% 'hline{zjednodušený_index}' + řetězec segmentu == výstupní řetězec
```

Výpis 22: Syntaxe okrajů v hlavičce [vlastní]

Syntaxe okrajů umožňuje zkrácení zápisu ve dvou osách současně, tedy lze zkrátit zápis jednotlivých segmentů jednoho okraje, a poté zapsat, kde v rámci celé tabulky se takto segmentovaný okraj opakuje. Pro lepší pochopení je uveden výpis 23 a jemu odpovídající tab. 9.

Podobně jako lze sloupce řešit naprosto identicky jako řádky a na konci pouze změnit klíčové slovo, tak i tvorba *výsledných řetězců* horizontálního a vertikálního okraje se liší pouze použitým klíčovým slovem a vstupním seznamem dat.

```
\begin{table}[ht]
    \centering
    \caption{Ukázka segmentovaných
        okrajů}
    \begin{tblr}{
        % azure == modrá,
        % magenta == růžová
        % opakují se segmenty
        hline{1-2} = {1-2}{1pt,
            azure6},
        hline{1-2} = {3-4}{1pt,
            magenta6},
        % opakuje se celý okraj
        hline{3-4} = {1pt, azure6},
    }
         11 & 12 & 13 & 14 \\
         21 & 22 & 23 & 24 \\
         31 & 32 & 33 & 34 \\
    \end{tblr}
\end{table}
```

Výpis 23: Zjednodušený zápis okrajů [vlastní]

Tab. 9: Ukázka segmentovaných okrajů

11	12	13	14
21	22	23	24
31	32	33	34

Celý princip bude tedy pouze vysvětlen na okrajích horizontálních. Při tvorbě vertikálních okrajů stačí veškeré indexy `0` zaměnit za `1`, všechny řetězce „`hline`“ nahradit řetězcem „`vline`“ a místo seznamu `all_rows`, použít seznam `all_cols`.

Tvorba *výsledného řetězce* okraje probíhá ve dvou fázích – v první fázi se zjistí indexy segmentů a segmentům příslušné formátovací řetězce, ve druhé fázi jsou hledány opakující se segmenty v rámci celé tabulky.

Proces tvorby výsledného řetězce okraje má krásný řád, který lze vidět na konkrétním příkladu v obr. 29 přílohy A. Během čtení zbytku kapitoly je doporučeno v případě nejasností na tento příklad nahlédnout.

Tvorba segmentů v rámci jednoho okraje

Vstupem pro tuto fázi je objekt typu `CRow` obsahující jak informace o celých (nesegmentovaných) okrajích jednoho řádku tak i seznam buněk tohoto řádku, kde mohou být uloženy okraje jednotlivých buněk. Je nutné připomenout, že v objektech typu `CRow` jsou ukládány pouze společné spodní okraje buněk (nesegmentované) a ze seznamu buněk daného řádku je při tvorbě horizontálního okraje podstatný pouze spodní okraj (uložen by měl být pouze spodní a pravý).

Je-li ve vstupním objektu typu `CRow` uložen společný okraj, pak se jedná o nesegmentovaný okraj a je možné rovnou vytvořit formátovací řetězec.

V opačném případě je nutné provést *segmentaci okraje*. Segmentace probíhá tak, že nejprve je vytvořen `dictionary` do kterého budou ukládány jednotlivé styly jako klíče (dvojice barva-typ) a k danému stylu (klíči) bude postupně tvořen číselný seznam segmentů, ve kterých se daný styl opakuje.

```
# klíč: border_style, hodnota: list[int] pořadí segmentů
border_styles_in_row = {}
cur_col = 1   # počítadlo (pořadí daného segmentu)

for my_cell in my_row.members:
    # i-tý segment jednoho okraje. index 0 == spodní okraj
    border_style = (my_cell.dxf.border_styles[0],
                    my_cell.dxf.border_colors[0])

    # je zde okraj?
    if border_style[0] != EBorderStyle.NONE:
        # opakuje se? ne
        if border_style not in border_styles_in_row:
            border_styles_in_row[border_style] = [cur_col]
        # opakuje se? ano
        else:
            border_styles_in_row[border_style].append(cur_col)
    cur_col += 1

# zde začíná následující výpis
single_hline = []
```

Výpis 24: Získání segmentů jednoho okraje [vlastní]

Jednotlivé seznamy indexů uvnitř výsledného `dictionary` jsou následně zjednodušeny podobně jako u tvorby zjednodušených indexů opakujících se řádků. Zjednodušení je zde o něco snazší, jelikož v době tvorby konvertoru nebylo možné indexy segmentů zjednodušit na „`odd`“ a něco dalšího, fungovaly pouze zjednodušení obecným zápisem, nebo čistě pomocí „`odd`“ a „`even`“ bez dalších indexů.

Ke každému prvku `dictionary` všech segmentů jednoho řádku je poté vytvořen formátovací řetězec, opět, analogicky s tvorbou formátovacího řetězce celých řádků. K formátovacímu řetězci je rovnou připojen i zjednodušený index segmentů a výsledek je uložen do výstupního seznamu, který je po dokončení všech prvků `dictionary` všech segmentů jednoho řádku vrácen.

```
# výstupní seznam segmentů
single_hline = []

# cyklus probíhá na stejné úrovni jako cyklus z předchozího výpisu
for style in border_styles_in_row:
    # segment obsahuje barvu i styl
    # border_styles_in_row[style] == index segmentu
    # style[0].value == typ okraje (str)
    # style[1].value == barva okraje (str)
    if style[1] != EColor.NOT_SET and style[0] != EBorderStyle.THIN:
        single_hline.append(f'{{{{border_styles_in_row[style]}}}}'
                            f'{{{style[0].value}, {style[1].value}}}')
    ...
    # jen barva
    else:
        single_hline.append(f'{{{{border_styles_in_row[style]}}}}'
                            f'{{{style[1].value}}}')
return single_hline
```

Výpis 25: Převedení jednotlivých segmentů na seznam řetězců [vlastní]

Dvojité okraje Dvojité okraje vyžadují mírně speciální přístup. Obecně se dvojité okraje v `tabularray` zadávají způsobem znázorněným na výpise 26 a umožňují široké možnosti vzhledu. Jelikož v Excelu lze jako dvojité zadat jen dva totožné okraje, navíc vždy tenkou souvislou čarou, jsou v konvertoru dvojité okraje řešeny velmi jednoduše.

Prakticky se s nimi celou dobu pracuje jako s každým jiným okrajem a rozdíl nastává až při tvorbě řetězce segmentu. Do řetězce segmentu jsou za sebe složeny dvě sady stejných údajů (indexy segmentů a jim příslušný formátovací řetězec) tak, že je dodržena syntaxe `tabularray` pro dvojité okraje. Aby bylo možné dvojité okraje správně naformátovat uvnitř hlavičky, tedy přidat k oběma kopiím zjednodušený index, je mezi kopie přidána sekvence znaků, která se zde nemůže objevit a která slouží jako místo kde bude formátovací řetězec na konci druhé fáze tvorby okrajů rozdělen. Každé části pak bude přiřazen stejný zjednodušený index.

```
% hline{zjednodušený_index} =
    {pořadí}{indexy_segmentu}{formátovací_řetězec}
hline{1-3} = {1}{1-2}{1pt},
hline{1-3} = {2}{2-3}{red6, 1pt}
```

Výpis 26: Syntaxe dvojitých okrajů [vlastní]

Tab. 10: Příklad dvojitých okrajů (hlavička tvořena výpisem 26)

11	12	13	14
21	22	23	24
31	32	33	34

Zjednodušení opakujících se segmentů

Vstupem pro tuto fázi je seznam všech řádků tabulky `all_rows`. Tento proces probíhá „nad“ získáním segmentů, tedy probíhá iterace všemi řádky `all_rows` a během každé z nich je získán seznam řetězců segmentů pro okraje uložené v daném řádku. Každý prvek (tj. segment) získaného seznamu je uložen do `dictionary` jako klíč a hodnotou je pořadí okraje ve výstupní tabulce, v tomto případě se jedná o pořadí aktuálního řádku plus 1. Postupnou iterací všemi řádky je `dictionary` všech segmentů postupně rozšiřován o další výskyty stejných segmentů v jiných oblastech tabulky nebo o nové segmenty a o místo kde se vyskytovaly.

Jakmile je iterace všemi řádky `all_rows` dokončena proběhne zjednodušení uložených výskytů pro každý segment, naprosto stejným způsobem jako tvorba zjednodušeného indexu segmentů okraje. V poslední fázi jsou poskládány hodnoty jednotlivých segmentů do výsledného řetězce spojením řetězce segmentu, zjednodušeného indexu a klíčového slova.

3.8.5 Řetězce jednotlivých buněk

Formát jednotlivých buněk se v `tabularray` píše dle komentáře ve výpise 27.

```
% cell{řádek}{sloupec} =
%   {rozměry_sloučených_buněk}{formátovací řetězec}
% formátovací řetězec = 'halign, valign, fg=barva_textu,
    bg=barva_pozadí, font=\font_cmds, cmd={\latex_cmd}'

% sloučená a formátovaná
cell{1}{3} = {r=2, c=3}{m, c, fg=white, bg=azure3, font=\slshape},
% sloučená
cell{3}{8} = {r=3}{},
% formátovaná
cell{11}{2} = {l, bg=azure8},
```

Výpis 27: Syntaxe formátu buněk uvnitř hlavičky (`valign` je svislé zarovnání textu, `halign` je vodorovné zarovnání textu) [vlastní]

Je nutné připomenout, že jednotlivé objekty buněk po získání společných vlastností v sobě mají uloženy pouze ty údaje, které nejsou společné v rámci celého řádku nebo sloupce. Proces slouží k zakomponování těchto „zbylých“ údajů do výstupní tabulky.

Tvorba výsledného řetězce je velmi jednoduchá a funguje na stejném principu jako v případě tvorby *formátovacího řetězce* celých řádků. Oproti postupu u řádků je zde místo zjednodušeného indexu vložen zápis s xy souřadnicemi buňky v rámci výstupní tabulky a v případě sloučených buněk i jedna sada složených závorek udávající velikost sloučené oblasti. Výsledné řetězce od jednotlivých buněk jsou spojovány po řádcích do jednoho řetězce, v případě že by řádek byl příliš dlouhý, je ručně zalomen.

3.8.6 Legenda

Legenda slouží jako rychlá reference pro uživatele při práci s již hotovým výstupem. Účelem legendy je poskytnout uživateli informace o syntaxi, významu nebo možných hodnotách použitých prvků hlavičky. Vzhled legendy je velmi podobný syntaxi prvků hlavičky užitý v předchozích kapitolách, viz výpisy 22 a 21. Užití legendy je zcela dobrovolné a její implementace může být libovolná, není zde žádné omezení.

U zvoleného řešení jsou jednotlivé řetězce legendy hromadně uloženy v několika předdefinovaných `dictionary`, kde každý odpovídá jednomu typu prvku legendy. Jeden obsahuje pouze syntaxe, druhý pouze slovní vysvětlení, třetí seznam možných zkratek a v posledním jsou uloženy možné hodnoty parametrů vyskytujících se v hlavičce.

Během ostatních fází tvorby hlavičky je ukládána informace o použitých prvcích syntaxe do jednoho objektu tvořeného řadou logických hodnot. Informace jsou ukládány zpravidla až při tvorbě formátovacích řetězců, viz výpisy 19 a 27, a po dokončení jednoho nebo všech typů prvků hlavičky nastává vyhodnocení uložených logických hodnot.

K tvorbě jednoho prvku legendy je přistupováno pomocí názvu prvku hlavičky, tj. `row`, `hline`, `columns`, apod. Na základě názvu prvku jsou z `dictionary` syntaxe a slovního vysvětlení získány příslušné řetězce legendy přímo, tedy každému prvku hlavičky odpovídá právě jeden řetězec legendy.

K parametrům použitým uvnitř řetězců syntaxe se dále tvoří seznam jejich možných hodnot. Jelikož se parametry mezi syntaxemi různých prvků opakují, je seznam tvořen jako poslední. Jednotlivé parametry jsou do legendy přidány podle toho, které prvky se v rámci celé syntaxe objeví.

Průběžná a hromadná legenda

Vytvoření *výsledné* legendy probíhá po jednotlivých potřebných prvcích a rozlišují se dva způsoby zakomponování legendy – průběžné a hromadné.

U průběžného zakomponování prvků je vždy po dokončení tvorby jednoho typu prvku hlavičky vytvořen prvek obsahující syntaxi a slovní popis tohoto prvku a ten je přidán před danou část hlavičky. Po dokončení tvorby řetězců všech prvků hlavičky se vytvoří popis parametrů a zkratek.

V případě *hromadného* zakomponování prvků se nejprve vytvoří všechny prvky hlavičky a poté všechny potřebné prvky legendy, včetně popisu parametrů a zkratek. Oba řetězce se spojí tak, aby legenda byla na začátku.

3.9 Tvorba těla výstupu

Tělem výstupu je myšlena ta část výstupní tabulky, která obsahuje data jednotlivých buněk. V kapitole 3.6 byl naznačen způsob získaní zobrazovaných hodnot buněk a v této fázi stačí uložené hodnoty jednotlivých buněk pouze vytisknout resp. uložit do jednoho velkého řetězce.

Ve zjednodušeném případě stačí vytvořit složený cyklus, kde vnější cyklus iteruje řádky v seznamu všech řádků a vnitřní cyklus iteruje buňkami daného řádku.

```
# výsledné tělo výstupu
body_str = ''
for row in all_rows:
    # řádek těla výstupu
    converted_row = '\t\t'
    for my_cell in row.members:
        # sloučené buňky bez hodnot musí být ve výstupu prázdné
        if my_cell.is_merged == EMergedCell.MERGED_EMPTY:
            converted_row += ' & '
        else:
            # tisk uložené hodnoty buňky (vč. formátování textu)
            converted_row += f'{my_cell.disp_value} &'

    # odstranění přebytečného '&' na konci a přídání '\\'
    body_str += f'{coverted_row[:-2]}\\\\\n'
```

Výpis 28: Základní princip tvorby těla výstupu [vlastní]

U konvertoru je ke zvýšení přehlednosti výstupu celý postup z výpisu 28 mírně odlišný. Kromě samotného tisku se provádí i zarovnání celého výstupu podle znaku „&“ tak, aby odpovídající buňky byly i v textovém výstupu nad sebou. Dále jsou manuálně zalamovány příliš dlouhé řádky, aby jejich automatickým zalamováním v textovém editoru nedošlo ke ztrátě přehlednosti.

Zarovnání buněk podle „`&`“ je řešeno tak, že se před tvorbou těla výstupu určí délky (ve znacích) všech hodnot uložených v buňkách. Pro každý sloupec výstupní tabulky je postupně nalezena hodnota s nejvyšší délkou ve znacích, která je pro každý sloupec uložena do `dictionary` na základě pořadí sloupce ve výstupní tabulce (pole `.index`). Samotné zarovnání proběhne zápisem ve výpise 29

```
# cell_str ... hodnota zobrazená v těle výstupu
# max_col_char_widths ... dictionary šířek obsahu sloupců
# cur_col ... index aktuálního sloupce (inkrementuje se uvnitř
    vnitřního cyklu)
cell_str = f'{my_cell.disp_value:<{max_col_char_widths[cur_col]}} & '
```

Výpis 29: Zarovnání jednoho prvku těla výstupu [vlastní]

který zobrazené hodnotě zprava přidá takový počet mezer, aby měla ve výsledku stejný počet znaků stejnou jako hodnota `max_col_char_widths[cur_col]`, tedy jako nejvyšší počet znaků výstupu těla v rámci sloupce. Stejným počtem znaků všech buněk v rámci jednoho sloupce těla bude zaručeno i zarovnání podle „`&`“.

3.10 Tvorba finálního LaTeX výstupu

Po vytvoření hlavičky a těla výstupní tabulky už jen zbývá vše umístit do příslušného LaTeX prostředí. Použitá kombinace prostředí záleží na uživatelem zvoleném typu tabulky a konvertor rozlišuje čtyři typy prostředí tabulek, které se liší délkou a orientací.

Tab. 11: Přehled typů výstupní tabulky

Typ tabulky	Použitá prostředí (v pořadí)	Úroveň hlavičky a těla	Poznámka
Normální	`table`, `tblr`	2	
Sidewaystable	`sidewaystable`, `tblr`	2	Otočená, vždy zabírá celou stranu
Dlouhá	`longtblr`	1	Přes více stran, nesmí být uvnitř jiného prostředí
Rotatebox	`table`, `rotatebox`, `tblr`	3	Otočená

Přidání řetězců prostředí je řešeno tak, že je nejprve na základě zvoleného typu tabulky vytvořen začátek celého výstupu, tedy `\begin` od potřebných LaTeX

prostředí v tab. 11 a poté probíhá tvorba hlavičky a těla. Po vytvoření každého řádku hlavičky i těla je k řádku přidán počet tabulátorů daný úrovní hlavičky (těla) a proces pokračuje. Jakmile je tělo dokončeno, nastává tvorba konce celého výstupu, tj. přidání popisku pomocí `\caption` a ukončení všech použitých LaTeX prostředí pomocí `\end`. Řetězce pro tvorbu potřebných `\begin` a `\end` jsou ručně definované přímo v kódu konvertoru pro každý typ tabulky zvlášť a jejich struktura je naznačena ve výpise 30.

```
\begin{table}[ht]
    \centering
    \begin{tblr}{
        % hlavička
    }
        % tělo
    \end{tblr}
    \caption{Typ "Normální"}
\end{table}

\begin{table}[ht]
    \centering
    \rotatebox{90}{
        \begin{tblr}{
            % hlavička
        }
            % tělo
        \end{tblr}
    }
    \caption{Typ "rotatebox"}
\end{table}

\begin{sidewaystable}[ht]
    \centering
    \begin{tblr}{
        % hlavička
    }
        % tělo
    \end{tblr}
    \caption{Typ "sidewaystable"}
\end{sidewaystable}

\begin{longtblr}[
    caption = {Typ "Dlouhá"},
    label = {tab:dlouha},
]{
    % hlavička
}
    % tělo
\end{longtblr}
```

Výpis 30: Syntaxe dostupných typů tabulky [vlastní]

3.11 Převod „jen dat“

Při převádění dat z více a více tabulek může často nastat situace, že hodnoty jsou ve výsledku formátovány stejně. Dá se očekávat, že uživatel bude ve svém dokumentu požadovat, aby tabulky měly stejné „téma“ a lišily se pouze obsahem. Konvertor nabízí možnost ze vstupního souboru převádět pouze uložené (resp. zobrazené) hodnoty a přidat jim uživatelem definovanou hlavičku. Tím odpadá nutnost vstupní data jakkoli formátovat, uživatel šetří čas. Zápis hlavičky pomocí syntaxe `tabularray` může

být velmi obecný a umožňuje vytvořit jeden zápis pro libovolně dlouhý vstup, tedy za předpokladu, že se v zápisu nevyskytují chyby.

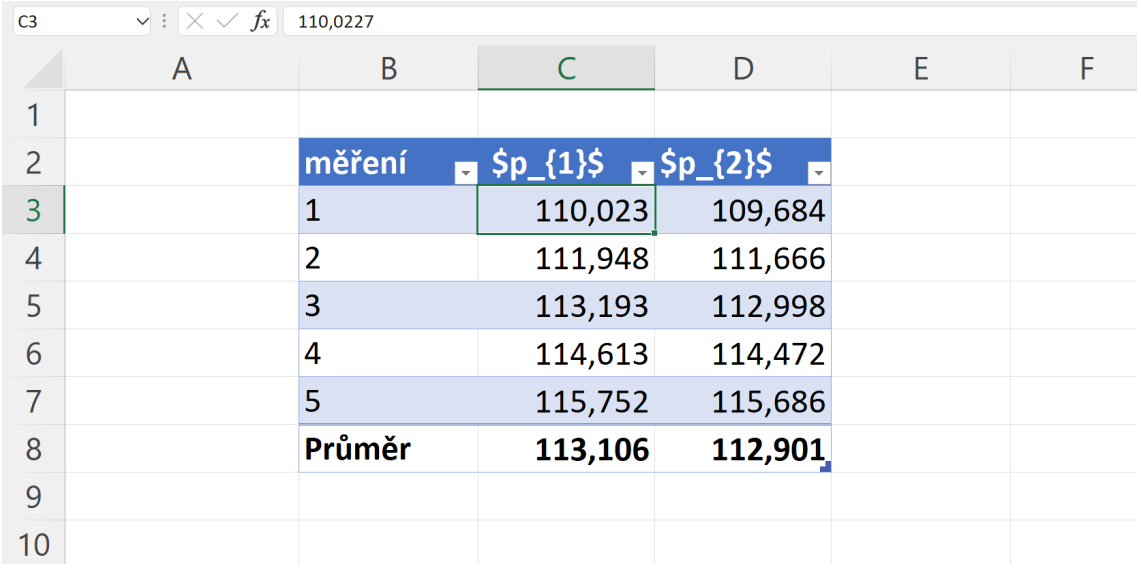
K ulehčení tvorby vlastní hlavičky nebo možnosti ručních úprav hlaviček z konvertoru je v programu vytvořena krátká nápověda popisující vybrané prvky formátování z hlediska syntaxe a významu. U každého prvku jsou uvedeny i okomentované příklady použití, ukázku této nápovědy lze vidět na obr. 32 v příloze B.

Celý převod probíhá jako složený cyklus v rámci uživatelem zadané oblasti, z každé buňky je převedena pouze informace o sloučení (kap. 3.5.1) a zobrazené hodnotě (kap. 3.6). Oba údaje se uloží do vlastního datového typu `CCellDO`. Po dokončení celého cyklu nastává proces tvorby výstupního řetězce, který se oproti klasickému převodu liší pouze v tvorbě hlavičky. Tvorba hlavičky probíhá tak, že se vytvoří pro řetězce formát jednotlivých buněk (kap. 3.8) a připojí za uživatelem zadanou hlavičku.

3.12 Ukázkový převod

Tato kapitola je věnována popisu kompletního procesu převodu tabulky od samého začátku až po konečný import do LaTeX dokumentu, krok za krokem.

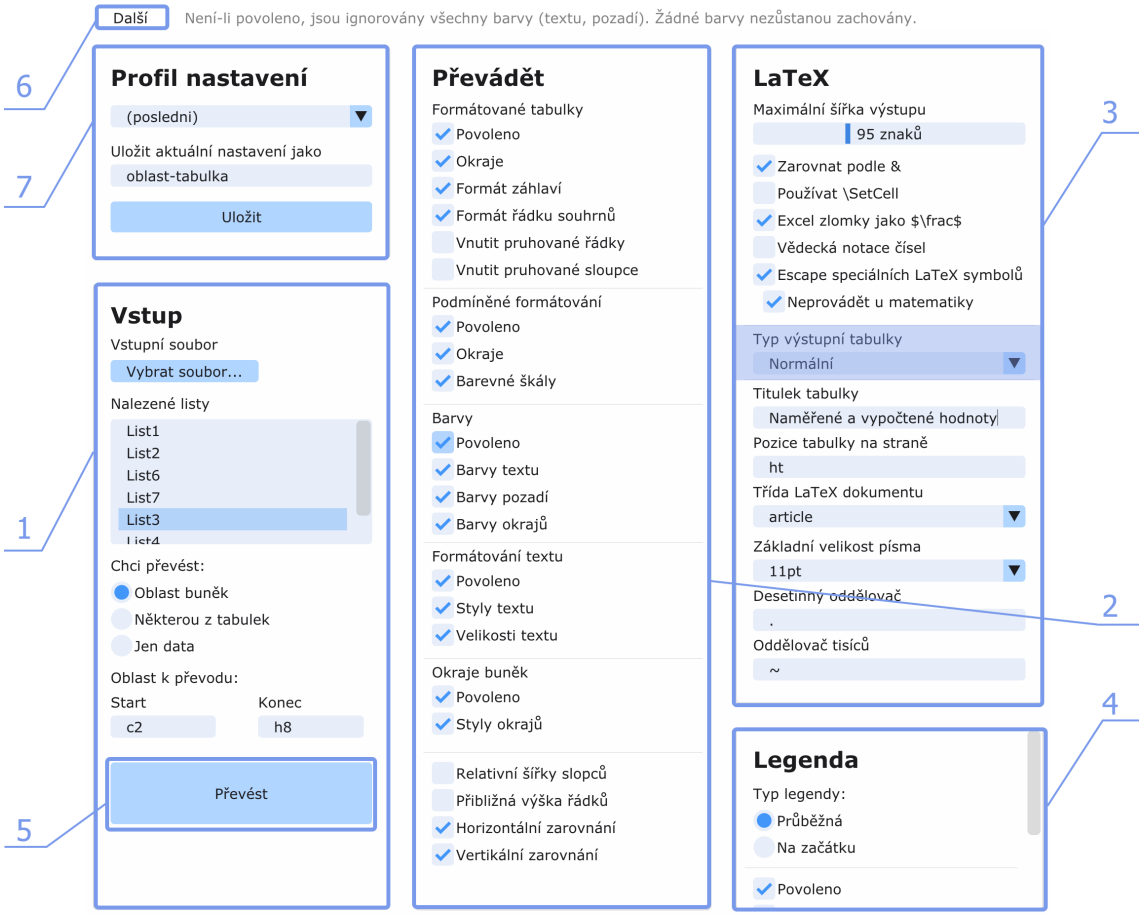
Každý převod začíná tvorbou vstupní tabulky v MS Excel. Prvním krokem je vytvoření vstupní tabulky včetně formátování, které bude ve výsledném LaTeX dokumentu požadováno. Na obr. 19 je vytvořen vstup formou oblasti formátované jako tabulka.



Obr. 19: Příklad vstupní tabulky [vlastní]

Dalším krokem je spuštění konvertoru a následný výběr souboru, ve kterém se tato tabulka nachází pomocí tlačítka *Vybrat soubor...* (oblast 1 na obr. 20).

Následuje výběr té části souboru, která obsahuje data k převedení, v příkladu na obr. 19 se jedná o list *tabulka* a oblast dat `B2` až `D8`. Jelikož se jedná o oblast dat formátovanou jako tabulka, lze oblast dat urychlit zvolením možnosti *Některou z tabulek* a výběrem odpovídající tabulky ze seznamu.



Obr. 20: Kompletní nastavení konvertoru, použité u tohoto a dalších příkladů. Čísla oblastí odpovídají pořadí jednotlivých kroků při práci s konvertorem. [vlastní]

Poté se v konvertoru zadají typy formátování, které se mají během převodu uvažovat, prostřednictvím části *Převádět* v grafickém rozhraní konvertoru (oblast 2 na obr. 20). V tomto případě se jedná o formátovanou tabulku se všemi typy formátování kromě pruhovaných sloupců. Aby se celý vstup převedl tak, jak je formátován, musí také být povoleny okraje, barvy a formátování textu, přestože nejsou v části *Formátované tabulky*. Pomocí zmíněných nastavení konvertoru jsou daná formátování ovládána plošně, tedy v celém vstupu.

Následně se v části *LaTeX* (oblast 3 na obr. 20) zadají dodatečná nastavení týkající se výstupní tabulky. Tabulka bude vkládána do dokumentu třídy `book` se základní velikostí písma 12 pt a jelikož není příliš dlouhá ani široká půjde o typ tabulky *Normální*. Ve vstupu se objevoval text uvnitř „$$“ obsahující speciální LaTeX symboly, které je potřebné zachovat – je vhodné zapnout *Neprovádět u matematiky*.

Jelikož se v dokumentu vyskytují desetinná čísla, která se obvykle oddělují desetinnou čárkou, tak je také patřičně přenastaven *Desetinný oddělovač*. Ze sekce *Legenda* (oblast 4 na obr. 20) bude ponechán jen slovní popis použitých LaTeX příkazů.

Nyní lze spustit převod stisknutím tlačítka *Převést* (oblast 5 na obr. 20). V okně *Výstup* je zobrazena výstupní tabulka, kterou lze zkopírovat.

Před vložením tabulky do LaTeX dokumentu je nezbytné mít nainstalovány potřebné balíčky. Návod k jejich instalaci je naznačen v sekci *Další* → *Potřebné packages* (oblast 6 na obr. 20) a v příloze D.

Jakmile jsou potřebné balíčky nainstalovány, lze do dokumentu vložit obsah okna *Výstup*, čímž celý proces převodu končí. Výsledný textový i zkompilovaný výstup je ve výpise 31 a tab. 12.

```
\begin{table}
    \centering
    \begin{tblr}{
        % row = Nastaví zadaný formát konkrétního řádku.
        row{2,4,6} = {bg=azure9},
        row{1} = {fg=white, bg=azure5, font=\bfseries},
        row{7} = {font=\bfseries},
        % hline = Vytvoří horizontální okraj.
        % Tento okraj bude pouze v sloupcích dle parametru
            'indexy_sloupců'.
        hlines = {azure5},
        % colspec = Řídí šířku a horizontální zarovnání textu všech
            sloupců.
        colspec = {lcc},
    }
        měření & $p_{1}$ & $p_{2}$ \\
        1      & 110,023 & 109,684 \\
        2      & 111,948 & 111,666 \\
        3      & 113,193 & 112,998 \\
        4      & 114,613 & 114,472 \\
        5      & 115,751 & 115,686 \\
        Průměr & 113,106 & 112,901 \\
    \end{tblr}
    \caption{Naměřené a~vypočtené hodnoty}
\end{table}
```

Výpis 31: LaTeX kód k vytvoření tabulky [vlastní]

Po úspěšném provedení každého převodu se použité nastavení a oblast vstupních dat uloží do profilu nastavení „*(poslední)*“. Zadaná nastavení je možné uložit do

měření	p_1	p_2
1	110,023	109,684
2	111,948	111,666
3	113,193	112,998
4	114,613	114,472
5	115,751	115,686
Průměr	**113,106**	**112,901**

Tab. 12: Naměřené a vypočtené hodnoty

vlastního profilu nastavení zadáním (libovolného) názvu profilu do pole *Uložit aktuální nastavení jako*. Při dalších převodech pak stačí tento profil vybrat ze seznamu v horní části sekce *Profil nastavení* (oblast 7 na obr. 20). Výběrem profilu nastavení odpadá nutnost pokaždé provádět nastavení v sekcích *Převádět*, *LaTeX* i *Legenda* (oblasti 2, 3 a 4 na obr. 20) a stačí pouze vybrat soubor a oblast vstupních dat.

3.13 Konkrétní ukázky vstupu a výstupu

V kapitole jsou uvedeny tři typy vstupních tabulek – běžná, dlouhá a otočená – a jim odpovídající výstupní tabulky jak ve formě kódu, tak i ve formě vizuální reprezentace. Vzhledem k rozsahu vizuálního materiálu je další výklad uvnitř popisků, vždy je dodrženo pořadí vstup – výstup – kód. Nastavení konvertoru u všech uvedených příkladů vychází z obr. 20, vždy bude provedena pouze změna nastavení *Typ výstupní tabulky*.

Oddělení	Říjen	Listopad	Prosinec
Maso	101 000 Kč	145 000 Kč	115 000 Kč
Pečivo	114 000 Kč	150 000 Kč	154 000 Kč
Plodiny	134 000 Kč	97 000 Kč	137 000 Kč
Plodiny	150 000 Kč	162 000 Kč	127 000 Kč
Lahůdky	150 000 Kč	142 000 Kč	154 000 Kč
Maso	139 000 Kč	93 000 Kč	138 000 Kč
Pečivo	129 000 Kč	111 000 Kč	111 000 Kč
Lahůdky	116 000 Kč	98 000 Kč	104 000 Kč

Obr. 21: Příklad 1, obvyklá vstupní tabulka. Obvyklou tabulkou se rozumí vstup, který je pravidelný, tj. opakující se prvky vstupu by mělo jít zjednodušit. Předpokládá se, že nepůjde o příliš dlouhou nebo příliš širokou tabulku. [vlastní]

Tab. 13: Příklad 1, výstupní tabulka. Konvertor úspěšně vytvořil velmi věrohodný výstup. [vlastní]

Oddělení	Říjen	Listopad	Prosinec
Maso	101 000 Kč	145 000 Kč	115 000 Kč
Pečivo	114 000 Kč	150 000 Kč	154 000 Kč
Plodiny	134 000 Kč	97 000 Kč	137 000 Kč
Plodiny	150 000 Kč	162 000 Kč	127 000 Kč
Lahůdky	150 000 Kč	142 000 Kč	154 000 Kč
Maso	139 000 Kč	93 000 Kč	138 000 Kč
Pečivo	129 000 Kč	111 000 Kč	111 000 Kč
Lahůdky	116 000 Kč	98 000 Kč	104 000 Kč

```
\begin{table}[htbp]
    \centering
    \caption{Příklad 1, výstupní tabulka. Konvertor ...}
    \begin{tblr}{
        colspec = {llll},
        row{even} = {bg=violet9, l},
        row{3,5,7,9} = {l},
        row{1} = {fg=white, bg=violet5, l, font=\bfseries},
        column{1} = {font=\bfseries},
        hlines = {violet5},
    }
        Oddělení & Říjen    & Listopad  & Prosinec  \\
        Maso     & 101~000 Kč & 145~000 Kč & 115~000 Kč \\
        Pečivo   & 114~000 Kč & 150~000 Kč & 154~000 Kč \\
        ...
        Pečivo   & 129~000 Kč & 111~000 Kč & 111~000 Kč \\
        Lahůdky & 116~000 Kč & 98~000 Kč & 104~000 Kč \\
    \end{tblr}
\end{table}
```

Výpis 32: Příklad 1, výstup (kráceno). Lze vidět že výstup je maximálně zjednodušen, kód je přehledný a vhodný pro ruční úpravy. [vlastní]

Vstup	Formátovací řetězec				
	0.000	#.0#	#" "??/16	# ###.0	"ttt "@
0	0,000	,0	0	0,0	0
0,456	0,456	,46	7/16	0,5	0,456
3,1415926	3,142	3,14	3 2/16	3,1	3,1415926
abcd	abcd	abcd	abcd	abcd	ttt abcd
12345,67	12345,670	12345,67	12345 11/16	12 345,7	12345,67

Obr. 22: Příklad 2, vstupní tabulka. Jedná se o relativně široký vstup a může nastat, že se nevejde na šířku strany. Bude záměrně otočen o 90 stupňů. Tento příklad vstupu demonstruje sloučené buňky, dvojité okraje i použití formátovacích řetězců (kap. 3.6). [vlastní]

Tab. 14: Příklad 2, výstupní tabulka. Tabulka byla záměrně otočená jako ukázka možností konvertoru. Formátovací řetězce jsou aplikovány v souladu s kap. 3.6, také došlo k escape speciálních LaTeX symbolů. [vlastní]

Vstup	Formátovací řetězec				
	0.000	#.0#	#""??/16	# ###.0	"ttt "@
0,456	0,000	,0	0	0,0	0
3.1415926	0,456	,46	$\frac{7}{16}$	0,5	0,456
abcd	3,142	3,14	$3\ \frac{2}{16}$	3,1	3,1415926
12345,67	abcd	abcd	abcd	abcd	ttt abcd
	12345,670	12345,67	12345 $\frac{11}{16}$	12 345,7	12345.67

```
\begin{table}[htbp]
\centering
\caption{Příklad 2, výstupní tabulka. Tabulka byla...}
\rotatebox{90}{%
    \begin{tblr}{
        colspec = {lccccc},
        row{1-2} = {fg=white, bg=azure7, font=\bfseries},
        row{3,5,7} = {bg=azure9},
        hline{1,8} = {},
        hline{2} = {2-6}{},
        hline{3} = {1}{-}{},
        hline{3} = {2}{-}{},
        vline{1-2,7} = {},
        vline{3-6} = {2-7}{},
        cell{1}{1} = {r=2}{m}, cell{1}{2} = {c=5}{},
    }
        Vstup    & Formátovací řetězec &       &                    &
                   &          \\
                 & 0.000              & \#.0\#  & \#" "??/16         &
        \# \#\#\#.0 & "ttt "@ \\
                 & 0,000              & ,0      & 0                  &
        0,0        & 0        \\
        0,456    & 0,456              & ,46     & $\frac{7}{16}$     &
        0,5        & 0,456    \\
        3.1415926 & 3,142             & 3,14    & 3 $\frac{2}{16}$   &
        3,1        & 3,1415926 \\
        abcd     & abcd               & abcd    & abcd               &
        abcd       & ttt abcd \\
        12345,67 & 12345,670          & 12345,67 & 12345
           $\frac{11}{16}$ &
        12~345,7   & 12345.67 \\
    \end{tblr}
}
\end{table}
```

Výpis 33: Příklad 2, výstup. Výstup je zjednodušený a vhodný pro ruční úpravy. Orientace v těle tabulky je mírně náročnější. [vlastní]

i	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9
1	0,229	0,244	0,250	0,252	0,254	0,254	0,254	0,254	0,254
2	0,271	0,286	0,291	0,294	0,295	0,295	0,295	0,295	0,295
3	0,288	0,303	0,308	0,311	0,312	0,312	0,312	0,312	0,312
4	0,297	0,313	0,317	0,320	0,321	0,321	0,321	0,321	0,321
5	0,303	0,318	0,323	0,326	0,327	0,327	0,327	0,327	0,327
6	0,307	0,322	0,327	0,329	0,331	0,331	0,331	0,331	0,331
7	0,310	0,325	0,330	0,332	0,334	0,334	0,334	0,334	0,334
8	0,312	0,327	0,332	0,334	0,336	0,336	0,336	0,336	0,336
9	0,314	0,329	0,334	0,336	0,337	0,337	0,337	0,337	0,337
10	0,315	0,330	0,335	0,337	0,339	0,339	0,339	0,339	0,339
11	0,316	0,331	0,336	0,339	0,340	0,340	0,340	0,340	0,340
12	0,317	0,332	0,337	0,339	0,341	0,341	0,341	0,341	0,341
13	0,318	0,333	0,338	0,340	0,342	0,342	0,342	0,342	0,342
14	0,319	0,334	0,339	0,341	0,342	0,342	0,342	0,342	0,342
15	0,319	0,334	0,339	0,341	0,343	0,343	0,343	0,343	0,343
16	0,320	0,335	0,340	0,342	0,343	0,343	0,343	0,343	0,343
17	0,320	0,335	0,340	0,342	0,344	0,344	0,344	0,344	0,344
18	0,321	0,336	0,341	0,343	0,344	0,344	0,344	0,344	0,344
19	0,321	0,336	0,341	0,343	0,345	0,345	0,345	0,345	0,345
20	0,322	0,337	0,341	0,344	0,345	0,345	0,345	0,345	0,345
21	0,322	0,337	0,342	0,344	0,345	0,345	0,345	0,345	0,345
22	0,322	0,337	0,342	0,344	0,345	0,345	0,345	0,345	0,345
23	0,322	0,337	0,342	0,344	0,346	0,346	0,346	0,346	0,346
24	0,323	0,338	0,342	0,345	0,346	0,346	0,346	0,346	0,346
25	0,323	0,338	0,343	0,345	0,346	0,346	0,346	0,346	0,346
26	0,323	0,338	0,343	0,345	0,346	0,346	0,346	0,346	0,346
27	0,323	0,338	0,343	0,345	0,347	0,347	0,347	0,347	0,347
28	0,323	0,338	0,343	0,345	0,347	0,347	0,347	0,347	0,347
29	0,324	0,339	0,343	0,345	0,347	0,347	0,347	0,347	0,347
30	0,324	0,339	0,343	0,346	0,347	0,347	0,347	0,347	0,347
31	0,324	0,339	0,343	0,346	0,347	0,347	0,347	0,347	0,347
32	0,324	0,339	0,344	0,346	0,347	0,347	0,347	0,347	0,347
33	0,324	0,339	0,344	0,346	0,347	0,347	0,347	0,347	0,347

Obr. 23: Příklad 3, vstupní tabulka. Vstupní tabulka je relativně dlouhá a bude tedy převedena na stejnojmenný typ výstupní tabulky. Bude záměrně provedena typografická chyba, aby došlo k rozdělení tabulky na více stran. Vstupní tabulka obsahuje barevnou škálu, všechny hodnoty jsou vypočteny a zaokrouhleny na tři desetinná místa. [vlastní]

Tab. 15: Příklad 3, výstupní tabulka. [vlastní]

i	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9
1	0,229	0,244	0,250	0,252	0,254	0,254	0,254	0,254	0,254
2	0,271	0,286	0,291	0,294	0,295	0,295	0,295	0,295	0,295

Pokračuje na další straně

Tab. 15: Příklad 3, výstupní tabulka. [vlastní] (pokračování)

i	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9
3	0,288	0,303	0,308	0,311	0,312	0,312	0,312	0,312	0,312
4	0,297	0,313	0,317	0,320	0,321	0,321	0,321	0,321	0,321
5	0,303	0,318	0,323	0,326	0,327	0,327	0,327	0,327	0,327
6	0,307	0,322	0,327	0,329	0,331	0,331	0,331	0,331	0,331
7	0,310	0,325	0,330	0,332	0,334	0,334	0,334	0,334	0,334
8	0,312	0,327	0,332	0,334	0,336	0,336	0,336	0,336	0,336
9	0,314	0,329	0,334	0,336	0,337	0,337	0,337	0,337	0,337
10	0,315	0,330	0,335	0,337	0,339	0,339	0,339	0,339	0,339
11	0,316	0,331	0,336	0,339	0,340	0,340	0,340	0,340	0,340
12	0,317	0,332	0,337	0,339	0,341	0,341	0,341	0,341	0,341
13	0,318	0,333	0,338	0,340	0,342	0,342	0,342	0,342	0,342
14	0,319	0,334	0,339	0,341	0,342	0,342	0,342	0,342	0,342
15	0,319	0,334	0,339	0,341	0,343	0,343	0,343	0,343	0,343
16	0,320	0,335	0,340	0,342	0,343	0,343	0,343	0,343	0,343
17	0,320	0,335	0,340	0,342	0,344	0,344	0,344	0,344	0,344
18	0,321	0,336	0,341	0,343	0,344	0,344	0,344	0,344	0,344
19	0,321	0,336	0,341	0,343	0,345	0,345	0,345	0,345	0,345
20	0,322	0,337	0,341	0,344	0,345	0,345	0,345	0,345	0,345
21	0,322	0,337	0,342	0,344	0,345	0,345	0,345	0,345	0,345
32	0,324	0,339	0,344	0,346	0,347	0,347	0,347	0,347	0,347
33	0,324	0,339	0,344	0,346	0,347	0,347	0,347	0,347	0,347

U dlouhých tabulek je šířka titulku přizpůsobena šířce tabulky bez možnosti toto chování jednoduše změnit. Titulek se navíc opakuje ve všech rozdělených částech a tento výklad je tedy nutné umístit jinak. Dlouhé tabulky mohou zabírat více stran, v místech rozdělení je automaticky vložen text, informující čtenáře o rozdělení tabulky. Řádek záhlaví se v obou částech opakuje, počet opakujících se řádků záhlaví i zápatí lze v konvertoru nastavit.

Výstup v tab. 15 připomíná vstupní tabulku, je zde viditelná nepřesnost v barvách pozadí (viz kap. 3.5.3). Konvertor správně poznal, že v záhlaví tabulky se nachází matematický zápis a všechny speciální LaTeX symboly ponechal tak, jak byly zadány na vstupu.

```
\begin{longtblr}[
    caption = {Příklad 3, výstupní tabulka. \vlastni},
    entry = {Příklad 3, výstupní tabulka.},
    label = {tab:o3},
]{
    colspec = {cccccccccc},
    rows = {c, m},  % jediné zjednodušení v celém výstupu
    cell{2}{2} = {bg=red4}, cell{2}{3} = {bg=red4}, cell{2}{4} =
    {bg=red5}, cell{2}{5} = {bg=red5}, cell{2}{6} = {bg=red5},
    cell{2}{7} = {bg=red5}, cell{2}{8} = {bg=red5}, cell{2}{9} =
    {bg=red5}, cell{2}{10} = {bg=red5},
    % vynecháno 128 (32x4) velmi podobných řádků
    % je zřejmé, že se nejedná o přehledný výstup
    cell{34}{2} = {bg=red9}, cell{34}{4} = {bg=azure8}, cell{34}{5} =
    {bg=azure5}, cell{34}{6} = {bg=azure4}, cell{34}{7} = {bg=azure4},
    cell{34}{8} = {bg=azure4}, cell{34}{9} = {bg=azure4}, cell{34}{10} =
    {bg=azure4},
    rowhead = 1,
}
    $i$ & $\mu_1$ & $\mu_2$ & $\mu_3$ & $\mu_4$ & $\mu_5$ & $\mu_6$ &
    $\mu_7$ & $\mu_8$ & $\mu_9$ \\
    1   & 0,229  & 0,244  & 0,250  & 0,252  & 0,254  & 0,254  &
    0,254  & 0,254  & 0,254  \\
    2   & 0,271  & 0,286  & 0,291  & 0,294  & 0,295  & 0,295  &
    0,295  & 0,295  & 0,295  \\
    % vynecháno 58 podobných řádků
    32  & 0,324  & 0,339  & 0,344  & 0,346  & 0,347  & 0,347  &
    0,347  & 0,347  & 0,347  \\
    33  & 0,324  & 0,339  & 0,344  & 0,346  & 0,347  & 0,347  &
    0,347  & 0,347  & 0,347  \\
\end{longtblr}
```

Výpis 34: Příklad 3, výstup (kráceno). Vzhledem k tomu, že se jedná o barevnou škálu je výstupní kód dlouhý, v současné implementaci konvertoru zadaný vstup nelze více zjednodušit. Při zanedbání přesnosti barev lze výstup považovat za přijatelný v případě, že při nutnosti úprav bude opět vytvořen pomocí konvertoru. Ruční úpravy výstupu se v tomto případě realizují těžce, jelikož s hodnotou buňky by měla být svázána i její barva pozadí. Buňky s bílou barvou pozadí se v hlavičce nevyskytují. [vlastní]

4 ZHODNOCENÍ A DISKUZE

Tato kapitola je zaměřena na shrnutí poznatků o celém řešení konvertoru a popisu dosažených výsledků. Dále uvádí několik omezení konvertoru, které dosud nemohly být popsány.

Konvertor výrazně usnadňuje tvorbu tabulek v LaTeX. Oproti klasickému způsobu tvorby LaTeX tabulek, kdy se tabulka sází pomocí *textových* příkazů, je vstupní tabulka tvořena v grafickém rozhraní tabulkového editoru MS Excel. Díky využití MS Excel uživatel v každém okamžiku vidí, jak bude výstup vypadat, s jistými omezeními může také svá data filtrovat, třídit, provádět dodatečné výpočty, apod.

Konvertor pouze zpracuje výstupní soubor z editoru MS Excel (typu `.xlsx`, případně na tento typ převedený), ke čtení dat ze souboru využívá Python knihovnu `openpyxl`. Knihovna je míněna především pro zápis údajů, a způsob *čtení* některých typů údajů v době psaní textu často nebyl autorem `openpyxl` řádně zdokumentován. Struktury, ve kterých byly uložené potřebné údaje, bylo nutné často prozkoumat pomocí debugovacích nástrojů. Z tohoto důvodu jsou v textu práce ty nejvýznamnější struktury detailněji popsány.

Často také nebylo zřejmé, *co* nebo *kde* ve čteném `.xlsx` souboru hledat. Ve většině případů byly údaje v `openpyxl` uloženy formou předpisu, pomocí kterého se určilo výsledné formátování. Zpracování takovýchto předpisů už bylo v rukou programátora. Z tohoto důvodu konvertor nezpracovává řazení, filtry a další pokročilé funkce MS Excel. Tyto funkce MS Excel by vyžadovaly velmi složitou implementaci, konvertor tedy lze minimálně v tomto ohledu v budoucnu rozšířit.

Konvertor umí individuálně u každé buňky výstupní tabulky nastavit zarovnání textu, barvu a styl okraje, barvu pozadí, barvu textu a velikost buňky. S výjimkou velikosti buňky lze tyto údaje nastavit jedním ze tří typů formátování, „ručně“, podmíněně anebo oblastí tabulky. Konvertor zpravidla nemá potíže s žádným z vyjmenovaných typů formátování a typy formátování lze i kombinovat.

Přesnost převedených údajů (tj. míra shody vstupu a výstupu) se u vstupů obsahujících pouze „ruční“ formátování buněk blíží sto procentům. Pro vstupy obsahující zbylé typy formátování je přesnost výstupních tabulek různá a zpravidla diktována složitostí okrajů buněk ve vstupní tabulce. Se složitějšími strukturami okrajů samozřejmě přesnost klesá, ale pouze u okrajů, ostatní prvky formátování jsou zpravidla převedeny přesně.

Případy, kdy konvertor nevytváří správný výstup, lze řešit převodem vstupních dat na „ruční“ formátování. Stačí vstupní hodnoty zkopírovat a vložit do jiné části vstupního souboru. Při vložení je nutné zvolit možnost *Hodnoty a formátování zdroje*. Tímto přístupem je také možné řešit chybějící řazení a filtry.

LaTeX výstup z konvertoru je postaven na balíčku `tabularray`, tedy je možné výstup použít *pouze* v kombinaci s tímto LaTeX balíčkem. Použití `tabularray` s sebou nese řadu výhod, zejména možnost zkrácení zápisu opakujících se prvků formátování a možnost oddělení formátovaní tabulky od obsahu tabulky. Proto byl také zvolen, při použití balíčku `tabularray` je tvorba výstupu mnohonásobně snazší než u jiných prostředí LaTeX tabulek.

Konvertor pro pravidelné vstupy tvoří kompaktní, vysoce přehledný LaTeX výstup, který lze snadno i ručně upravit. Pro nepravidelný vstup konvertor zpravidla vytvoří méně kompaktní zápis, jehož délka závisí na složitosti vstupu. Čím méně pravidelný vstup, tím méně kompaktní zápis. Obsah (tělo) tabulky je přehledný vždy. Konvertor také nabízí možnost si hlavičku výstupu vytvořit ručně a ze vstupního souboru převádět pouze data. Tím je zaručena vysoká konzistence výstupních tabulek, jelikož hlavičku lze vytvořit velmi obecně tak, aby fungovala pro libovolně dlouhé tělo tabulky.

Použití balíčku `tabularray` s sebou nese i dvě zásadní nevýhody – horní limit prvků formátování jedné tabulky (při užití editoru Overleaf) a omezený počet barev.

Na horní limit prvků formátování lze narazit u velmi dlouhých anebo velmi složitých vstupů. Jedná se o limit *formátování*, tj. není problém s množstvím dat zobrazených v tabulce, ale s množstvím příkazů v hlavičce tabulky. Limitu bylo v Overleaf dosaženo tabulkou s přibližně 4000 prvků hlavičky.

Mimo užití jiného editoru lze tento limit řešit dalším zjednodušením opakujících se prvků formátování. Kromě implementovaných zjednodušení (v rámci řádku, sloupce, okraje) lze zkrátit i zápis pro jednotlivé buňky a to ve dvou osách současně. Tato funkcionalita však byla do `tabularray` přidána až ve verzi 2022A, kdy konvertor byl ve velmi pokročilém stádiu vývoje. Implementace by tak vyžadovala radikální zásah do celé logiky převodu, proto zatím nebyla provedena. Zjednodušení opakujících se buněk může velmi výrazně zkrátit celou řadu výstupů a je hlavním cílem budoucího vývoje konvertoru.

V tabulkách lze v základu použít pouze 117 barev, což je pro většinu aplikací dostačující. Nedostatky se projeví zpravidla až v tabulkách s mnoha odstíny barev nebo při užití barevné škály.

Počet použitelných barev lze v současnosti rozšířit, možná řešení jsou však značně nepraktická. Jednou z možností je zvýšení počtu definovaných barev. Všechny použitelné barvy jsou definovány v balíčku `ninecolors`, stačilo by mezi uživatele uvést upravenou verzi tohoto balíčku a konvertor náležitě upravit. Řešení má ale jednu nevýhodu, k barvám na vstupu se musí (vždy) algoritmicky najít nejbližší zobrazitelná barva. S rostoucím počtem zobrazitelných barev výrazně roste výpočetní náročnost této operace.

5 ZÁVĚR

Výstupem této bakalářské práce je nástroj, který usnadní proces tvorby LaTeX tabulek, podporuje širokou škálu doplňujícího formátování a vytváří přehledný a srozumitelný výstup.

Celá práce je členěna do několika částí. V první části, která je rešeršního charakteru, jsou definovány základní pojmy týkající se systému LaTeX a sazby tabulek v LaTeX. Dále je uveden přehled současných nástrojů, které také usnadňují proces tvorby LaTeX tabulek.

Druhá část práce detailně popisuje princip funkce výsledného konvertoru. Jde o stěžejní část práce pro budoucí řešitele podobného problému. Nejprve je provedena volba vhodných programových nástrojů pro tvorbu konvertoru. Poté je podrobně vysvětleno odkud jsou získávána vstupní data, jakým způsobem jsou data ukládána, zpracována a poskládána do výstupního řetězce. V závěru kapitoly je ukázka práce s konvertorem. Následuje několik příkladů vstupních tabulek a jim odpovídajících výstupů konvertoru.

V poslední části je provedeno shrnutí poznatků o konvertoru, v rámci kterého je možné se dozvědět to nejdůležitější o výsledném řešení. Součástí kapitoly je popis výsledků práce a úskalí spojených s tvorbou konvertoru. Také je zde nastíněn další možný vývoj tohoto nástroje.

6 SEZNAM POUŽITÉ LITERATURY

[1] KOPKA, Helmut a Patrick W. DALY. *Guide to LATEX*. 4th ed. Boston: Addison-Wesley, c2004. ISBN 978-0321173850.

[2] RYBIČKA, Jiří. *LATEX pro začátečníky*. 3. vyd. Brno: Konvoj, 2003. ISBN 80-7302-049-1.

[3] FEUERSÄNGER, Christian. Manual for Package PgfplotsTable. *Pgfplotstable – Loads, rounds, formats and postprocesses numerical tables* [online]. Heidelberg: CTAN team, c2007 [cit. 2022-05-04]. Dostupné z: `https://mirrors.nic.cz/tex-archive/graphics/pgf/contrib/pgfplots/doc/pgfplotstable.pdf`

[4] PANTIGNY, François. The package nicematrix. *Nicematrix – Improve the typesetting of mathematical matrices with PGF* [online]. Heidelberg: CTAN team, c2018 [cit. 2022-05-04]. Dostupné z: `https://mirrors.nic.cz/tex-archive/macros/latex/contrib/nicematrix/nicematrix.pdf`

[5] LYU, Jianrui. Tabularray - Typeset Tabulars and Arrays with LaTeX3. *Tabularray – Typeset tabulars and arrays with LaTeX3* [online]. Heidelberg: CTAN team, c2021-c2022 [cit. 2022-05-04]. Dostupné z: `https://mirrors.nic.cz/tex-archive/graphics/pgf/contrib/pgfplots/doc/pgfplotstable.pdf`

[6] PONGPATTRACHAI, Dichapong, Paul CRAGG a Richard FISHER. IT infusion within the audit process: Spreadsheet use in small audit firms. *Omega* [online]. 2014, 37(1), 26-46 [cit. 2022-05-02]. ISSN 1467-0895. Dostupné z: `doi:https://doi.org/10.1016/j.accinf.2013.03.001`

[7] Excel VBA reference. *Microsoft technical documentation* [online]. Redmond: Microsoft, 2016 [cit. 2022-05-04]. Dostupné z: `https://docs.microsoft.com/en-us/office/vba/api/overview/excel`

[8] Microsoft. Office primary interop assemblies. *Microsoft technical documentation* [online]. Redmond: Microsoft, 2016 [cit. 2022-05-04]. Dostupné z: `https://docs.microsoft.com/en-us/visualstudio/vsto/office-primary-interop-assemblies`

[9] Microsoft. Office Add-ins documentation. *Microsoft technical documentation* [online]. Redmond: Microsoft, 2016 [cit. 2022-05-04]. Dostupné z: `https://docs.microsoft.com/cs-cz/office/dev/add-ins/`

[10] GAZONI, Eric a Charlie CLARK. Openpyxl. *Openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files* [online]. Portland: Read the Docs,

2010 [cit. 2022-05-04]. Dostupné z: https://openpyxl.readthedocs.io/en/stable/index.html

[11] ZANDER, Benito van der, Jan SUNDERMEYER, Daniel BRAUN a Tim HOFFMANN. TeXstudio. *TeXstudio: A LaTeX editor* [online]. Lübeck, 2009 [cit. 2022-05-04]. Dostupné z: https://www.texstudio.org

[12] JDMCreator. LaTeX Tables Editor. *Help: LaTeX Tables Editor* [online]. 2018 [cit. 2022-05-04]. Dostupné z: https://www.latex-tables.com/help.html

[13] HUGHES, Chelsea, Kirill MÜLLER a Ivan KOKAN. Excel2LaTeX. *Excel2LaTeX: Convert Excel spreadsheets to LATEX tables* [online]. Heidelberg: CTAN team, c1996-c2016 [cit. 2022-05-04]. Dostupné z: https://ctan.org/tex-archive/support/excel2latex

[14] ECMA-376. *Office Open XML file formats*. 5. Ženeva: Ecma International, 2021. Dostupné také z: https://www.ecma-international.org/publications-and-standards/standards/ecma-376/

[15] CIACCI, Massimo. 3D color spaces. *MATLAB Central File Exchange* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://www.mathworks.com/matlabcentral/fileexchange/60939-3d-color-spaces

[16] tuck1s. Getting Excel cell background themed color as hex with openpyxl. *Stack Overflow* [online]. New York: Stack Exchange, 2020 [cit. 2022-05-04]. Dostupné z: https://stackoverflow.com/a/65426130

[17] Microsoft. Description of how column widths are determined in Excel. *Microsoft technical documentation* [online]. Redmond: Microsoft, 2019 [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/office/troubleshoot/excel/determine-column-widths

[18] Number format codes. *Microsoft Support* [online]. Redmond: Microsoft, c2022 [cit. 2022-05-04]. Dostupné z: https://support.microsoft.com/en-us/office/number-format-codes-5026bbd6-04bc-48cd-bf33-80f18b4eae68

[19] FRIEDL, Jeffrey E. F.*Mastering regular expressions*. 3rd ed. Sebastopol: O'Reilly, 2006. ISBN 9780596528126.

[20] ZAPPONI, Carlo. Github Language Stats. *GitHut 2.0* [online]. 2016 [cit. 2022-05-04]. Dostupné z: https://madnight.github.io/githut/

[21] MERCADO, Jeff. Grouping consecutive numbers into ranges in Python 3.2. *Stack Exchange* [online]. New York: Stack Exchange, 2011 [cit. 2022-05-04]. Dostupné z: https://codereview.stackexchange.com/a/5202

[22] Creating a document in Overleaf. *Overleaf, Online LaTeX Editor* [online]. Londýn: Digital Science UK Limited, 2012 [cit. 2022-05-04]. Dostupné z: `https://www.overleaf.com/learn/how-to/Creating_a_document_in_Overleaf`

[23] ETTRICH, Matthias. LyX. *LyX – The Document Processor* [online]. LyX Team, 1995 [cit. 2022-05-04]. Dostupné z: `https://www.lyx.org/Home`

[24] RAMALHO, Luciano. *Fluent Python*. Sebastopol: O'Reilly Media, 2015. ISBN 9781491946008.

[25] Configparser — Configuration file parser. *Python 3.10.4 documentation* [online]. Beaverton: Python Software Foundation, 2022 [cit. 2022-05-07]. Dostupné z: `https://docs.python.org/3/library/configparser.html`

7 SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK

.NET	plaforma „dot net“
API	application programming interface rozhraní pro programování aplikací
AARRGGBB	reprezentace průhledných RGB barev zápisem v šestnáctkové soustavě
bg	barva pozadí
bool	logická hodnota
cf	objekt s informacemi o jednom pravidle podmíněného formátování
cmd	příkaz
col	sloupec
colspec	parametr udávající šířku a zarovnání textu sloupců výstupní tabulky
csv	comma-separated values hodnoty oddělené čárkou
dict	datový typ dictionary
dvi	device independent format formát nezávislý na výstupním médiu
dxf	objekt s informacemi o formátování (všech tří typů)
enum	výčtový typ
fg	barva popředí
float	číslo s plovoucí čárkou
GUI	graphical user interface grafické uživatelské rozhraní
halign	horizontání zarovnání textu
hline	horizontální okraj
HSL	hue, saturation, lightness barevný prostor se souřadnicemi odstín, sytost, světlost
HTML	hypertext markup language hypertextový značkovací jazyk

ID	identifikátor
ini	inicializační soubor
int	celé číslo
LaTeX	zdrojový soubor obsahující předpis pro sazbu dokumentu
longtblr	prostředí pro sazbu dlouhé tabulky balíčku `tabularray`
MS	microsoft
pdf	portable document format
PIA	primary interop assembly
pt	body (jednotka)
regex	regulární výraz
RGB	red, green, blue barevný prostor se souřadnicemi červená, zelená, modrá
rowspec	parametr udávající výšku a zarovnání textu řádků výstupní tabulky
rw	relativní šířka
str	řetězec
tblr	prostředí pro sazbu tabulky balíčku `tabularray`
TeX	systém, který provede samotnou sazbu dokumentu
valign	vertikální zarovnání textu
VBA	visual basic for applications
venv	virtuální prostředí Python
vk	výška v bodech
vline	vertikální okraj
wb	instance objetku typu `Workbook`
ws	instance objetku typu `Worksheet`
WYSIWYG	what you see is what you get volně přeloženo „co vidíš, to dostaneš“
xls	starý (binární) typ Excel souborů
xlsm	moderní typ Excel souborů s podporou maker
xlsx	moderní typ Excel souborů
XML	extensible markup language rozšiřitelný značkovací jazyk

Seznam obrázků

Seznam tabulek

Seznam výpisů

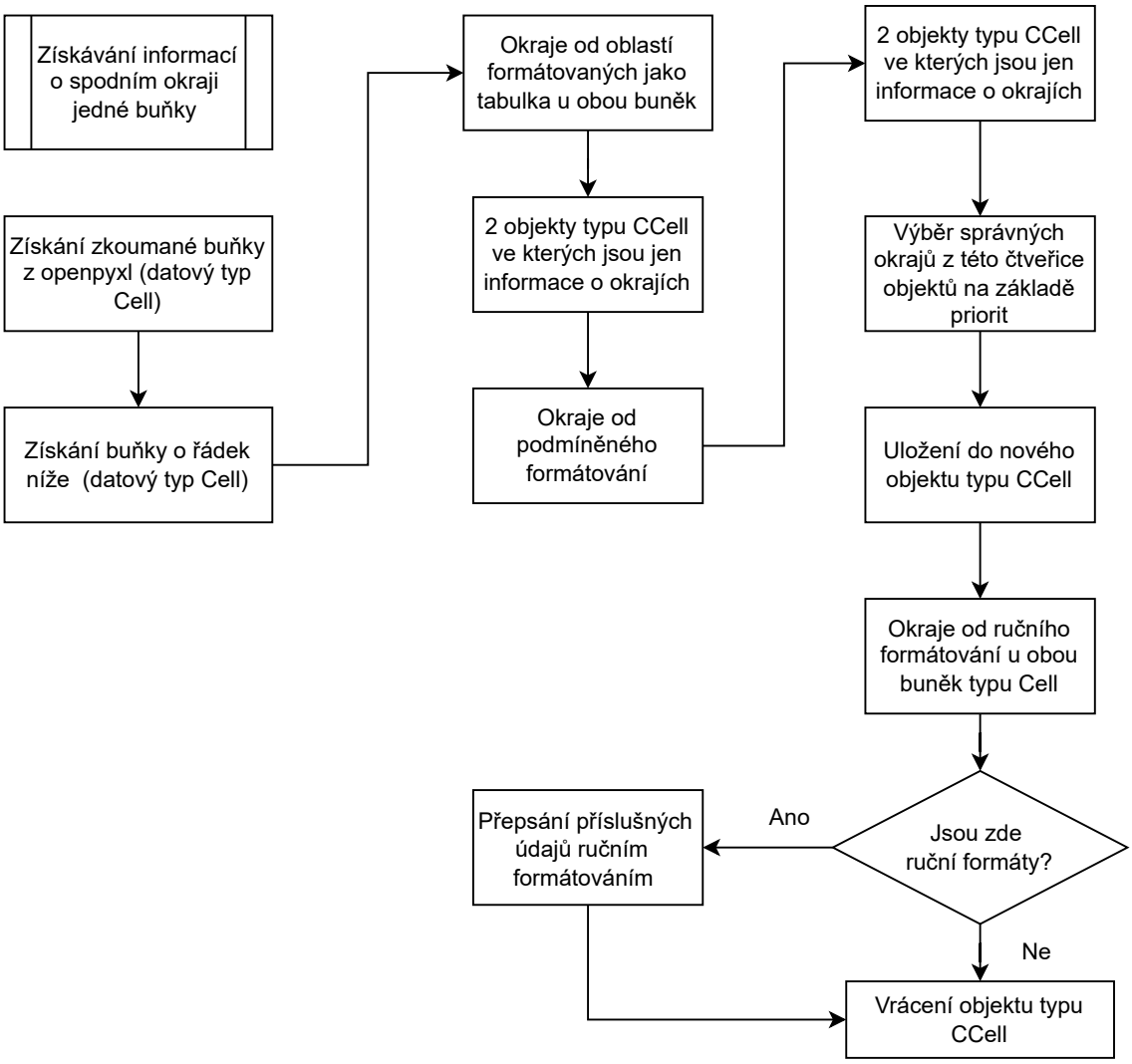
8 SEZNAM PŘÍLOH

Obsah přiloženého zip archivu

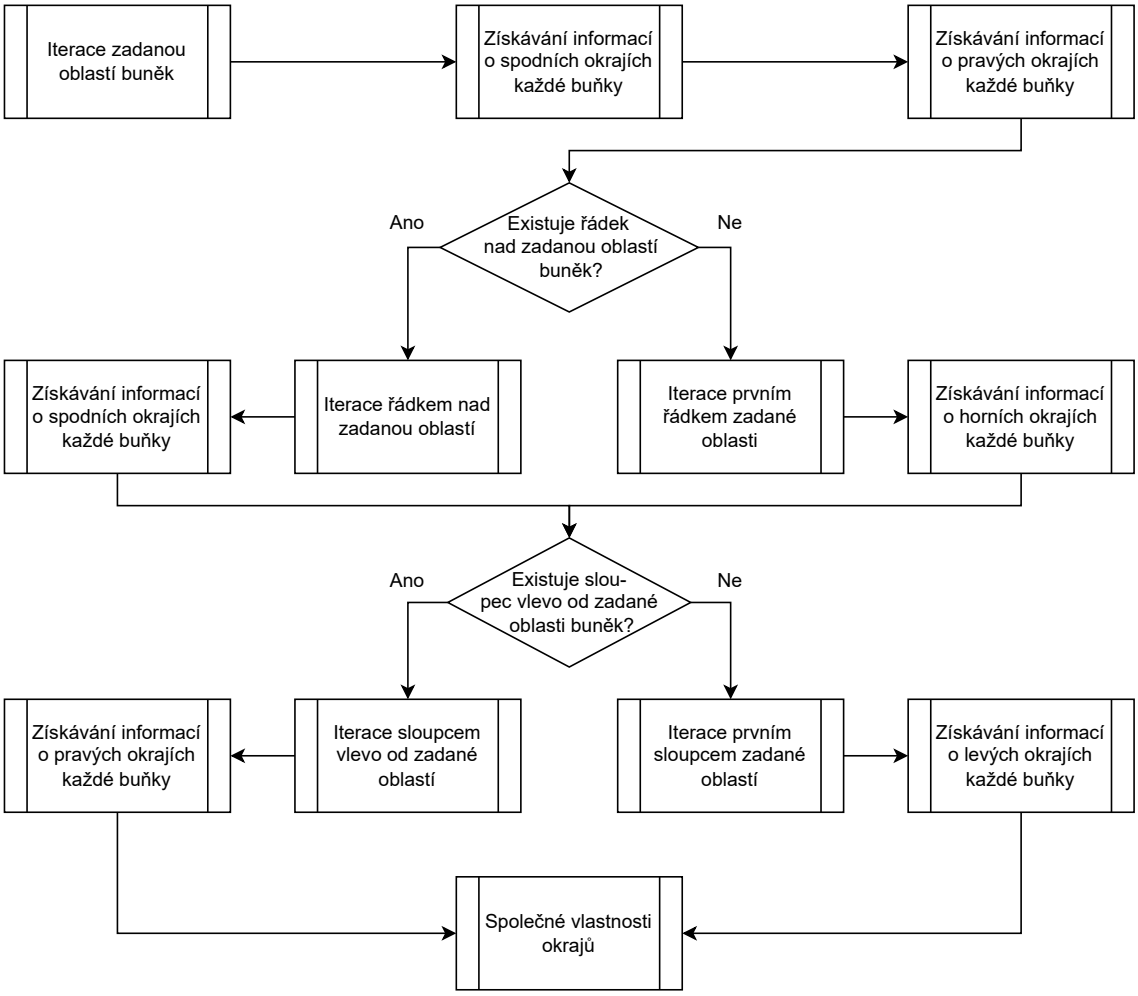
`2022_BP_Hobza_Jakub_217208_prilohy.zip`

- `konvertor.exe` – samotný konvertor.
- `priklady_io.xlsx` – sada testovacích vstupních tabulek (vstupy užité v textu práce a několik dalších příkladů).
- `readme.txt` – návod k použití.
- `zdrojový kód` – složka obsahující zdrojový kód konvertoru.
  - `classes.py` – obsahuje definice vlastních datových typů.
  - `colors.py` – obsahuje kód pro získání barev a převod na nejbližší zobrazitelnou barvu.
  - `condi.py` – kód pro zjištění podmíněného formátování (funkce pro určení platnosti pravidel, aplikování formátování od pravidel).
  - `dataGet.py` – zde se vyhodnocují data získaná od všech tří typů formátování.
  - `enums.py` – obsahuje definice datových typů na bázi Enum.
  - `fontsize.py` – kód k převedení velikosti písma.
  - `legend_entries.py` – obsahuje prvky legendy a funkce pro tvorbu legendy.
  - `main.py` – celý kód spojuje dohromady. Dále obsahuje logiku grafického rozhraní a kód pro vyhodnocení společných vlastností buněk.
  - `settings.py` – obsahuje kód pro načítání, ukládání a kontrolu nastavení.
  - `table.py` – získává formátování z oblastí formátovaných jako tabulka.
  - `toLatex.py` – obsahuje kód pro tvorbu výstupních řetězců.
  - `utils.py` – soubor pomocných funkcí.

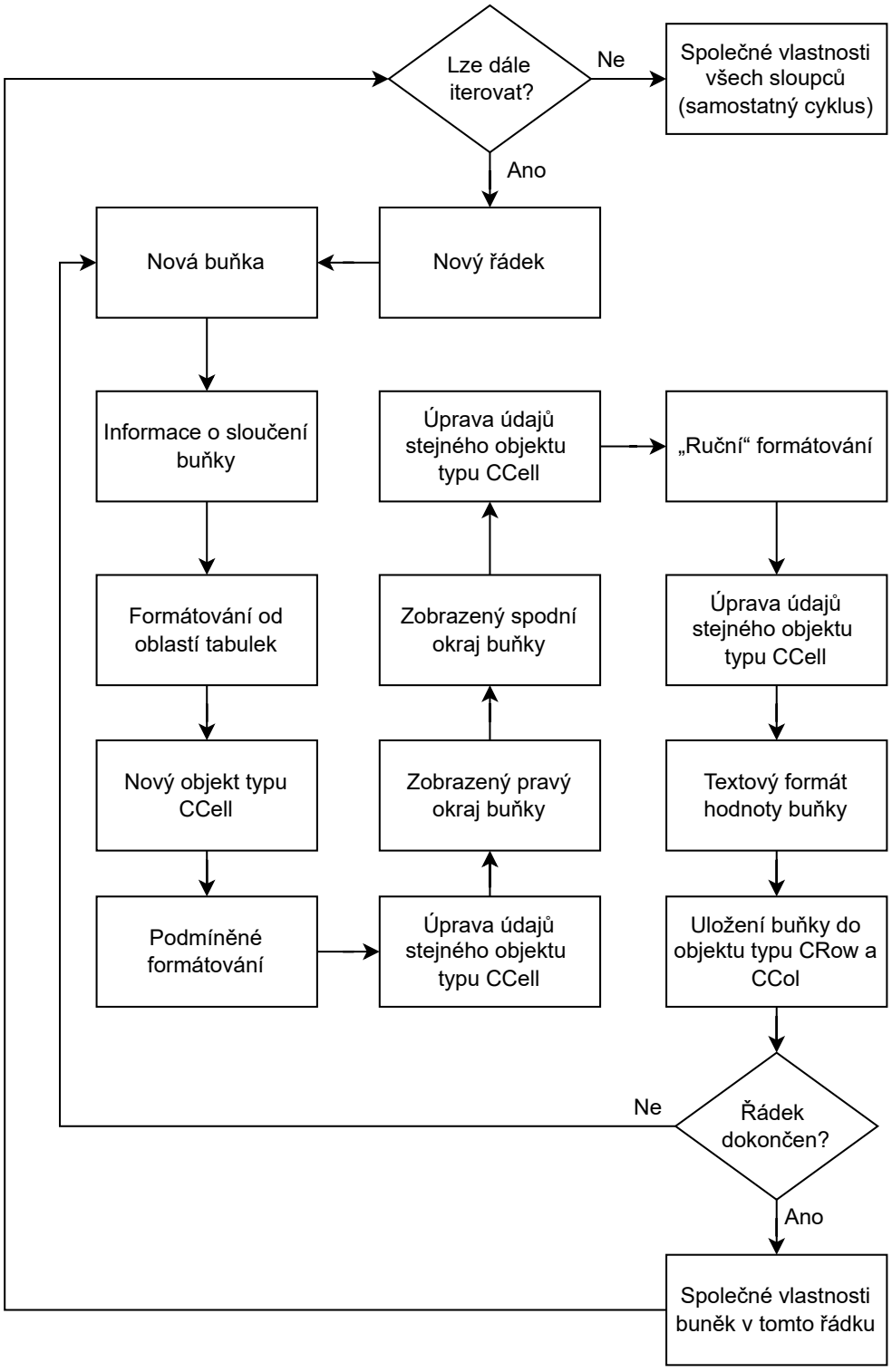
A Vývojové diagramy částí převodu



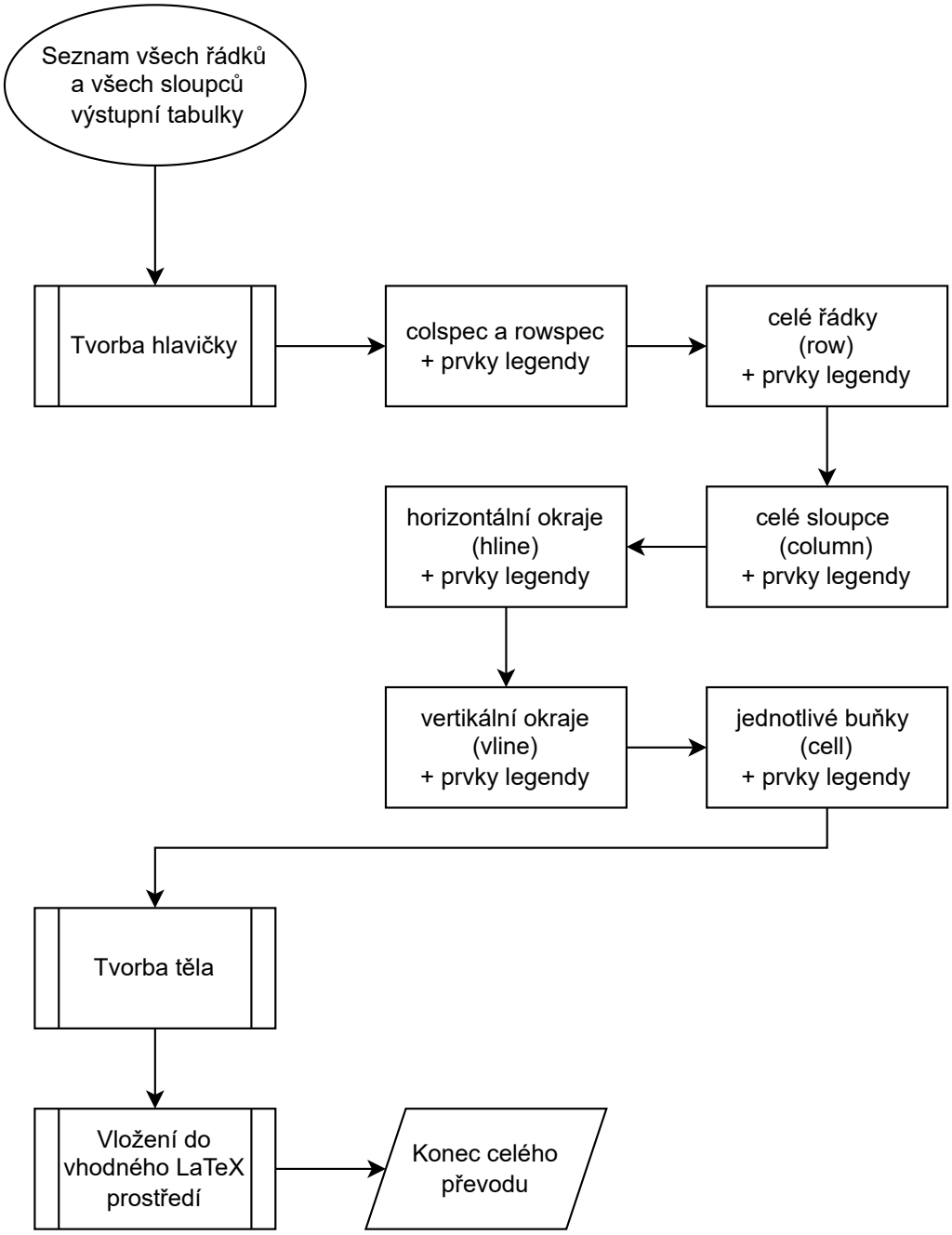
Obr. 24: Jeden „krok“ celého procesu získání okrajů. Jednotlivé „kroky“ se příliš neliší, prakticky jediný rozdíl je ve srovnávaných okrajích – místo horního a dolního jsou srovnávány pravý a levý u dvou sousedních buněk jednoho řádku. Je nutné zdůraznit, že jednotlivé porovnávací operace se provádí pro každý okraj zvlášť a každý okraj může být srovnáván s naprosto jinými daty, z čehož plyne možnost úplně různých výsledných okrajů [vlastní]



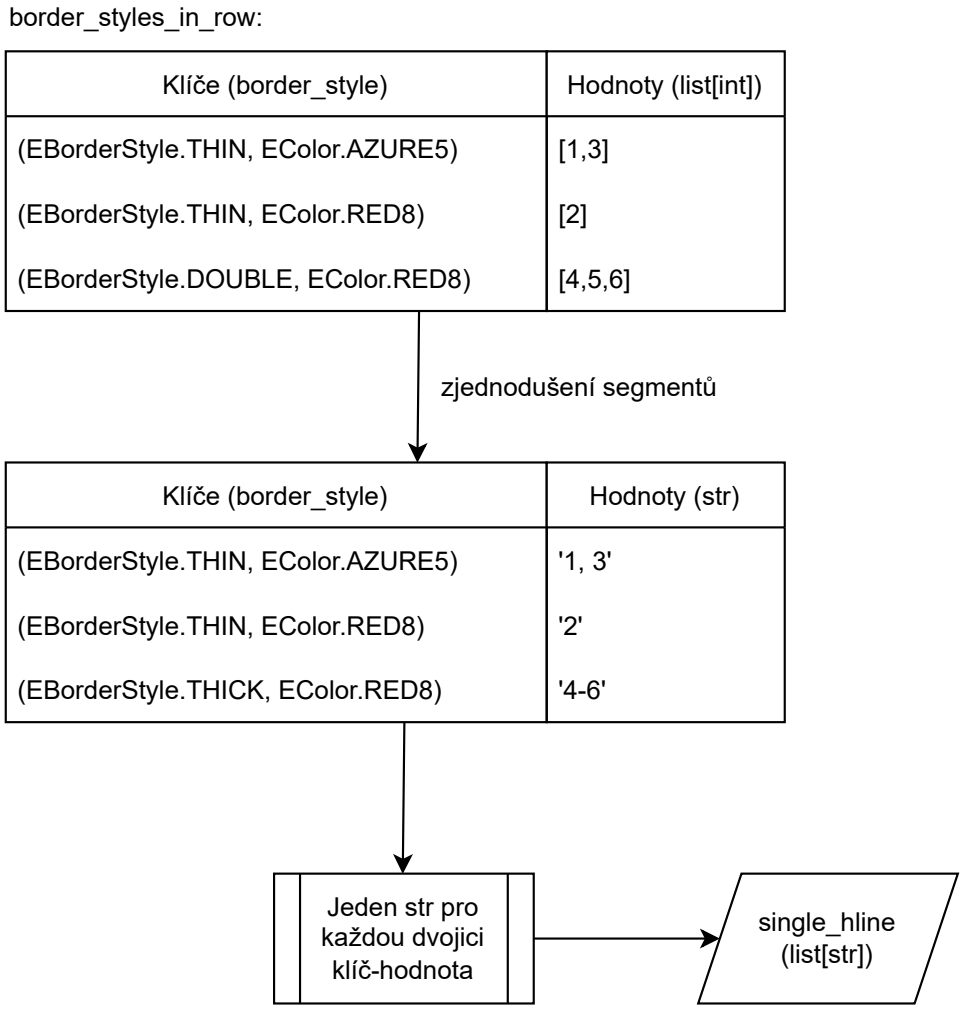
Obr. 25: Proces získání okrajů. Každé pole tohoto procesu představuje celý soubor dalších operací, kde jednu z nich lze vidět na obr. 24 [vlastní]



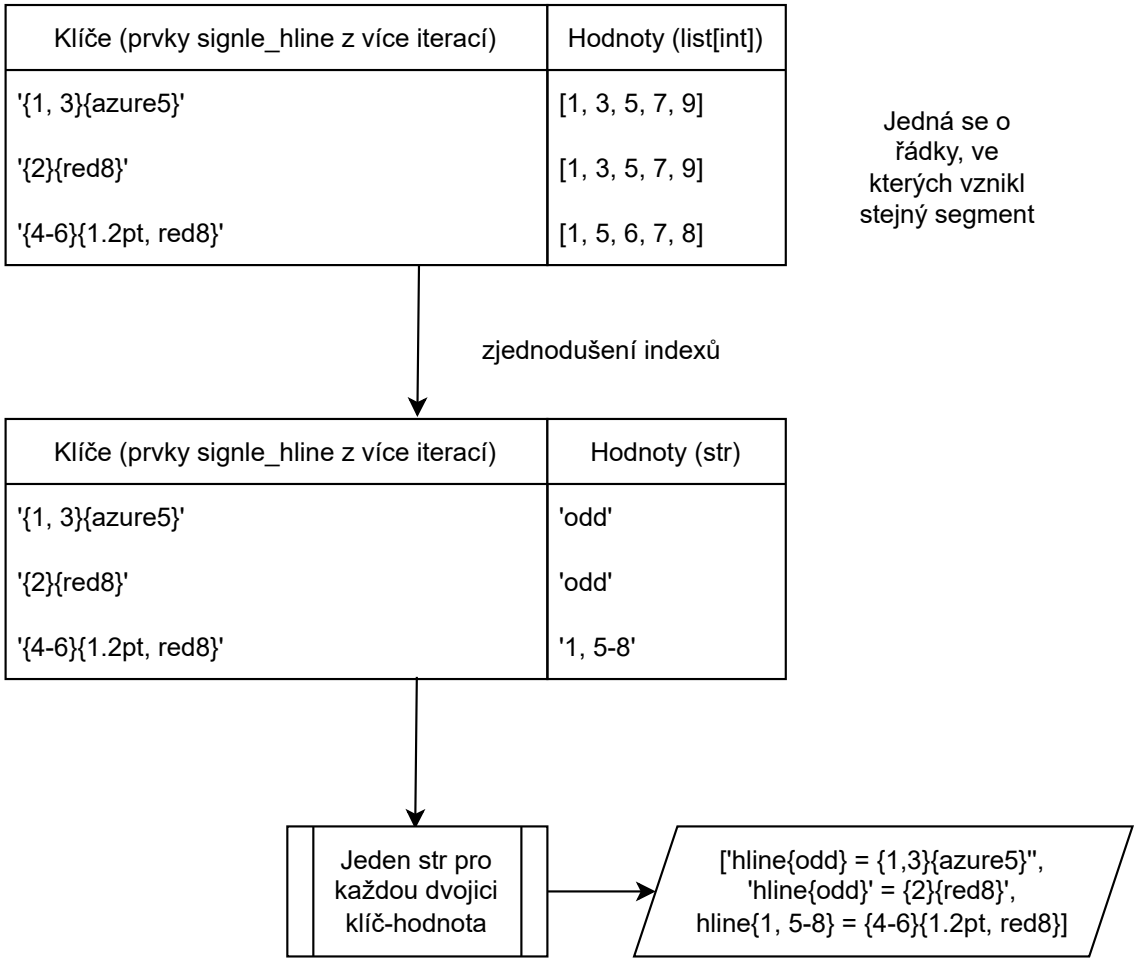
Obr. 26: Proces získávání dat z buněk vstupní tabulky. Algoritmus začíná v místě „Lze dále iterovat?“ v horní části. Získaná data se po dokončení některých kroků ukládají do objektu vlastního datového typu `CCell` resp. jeho části typu `CDxf`, aby byla na jednom místě. [vlastní]



Obr. 27: Tvorba výstupního LaTeX řetězce [vlastní]

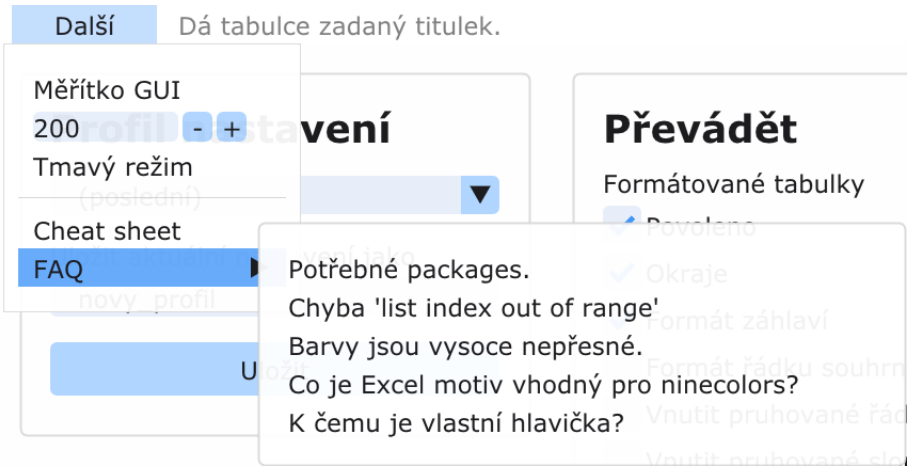


Obr. 28: Konkrétní příklad tvorby segmentů jednoho okraje [vlastní]

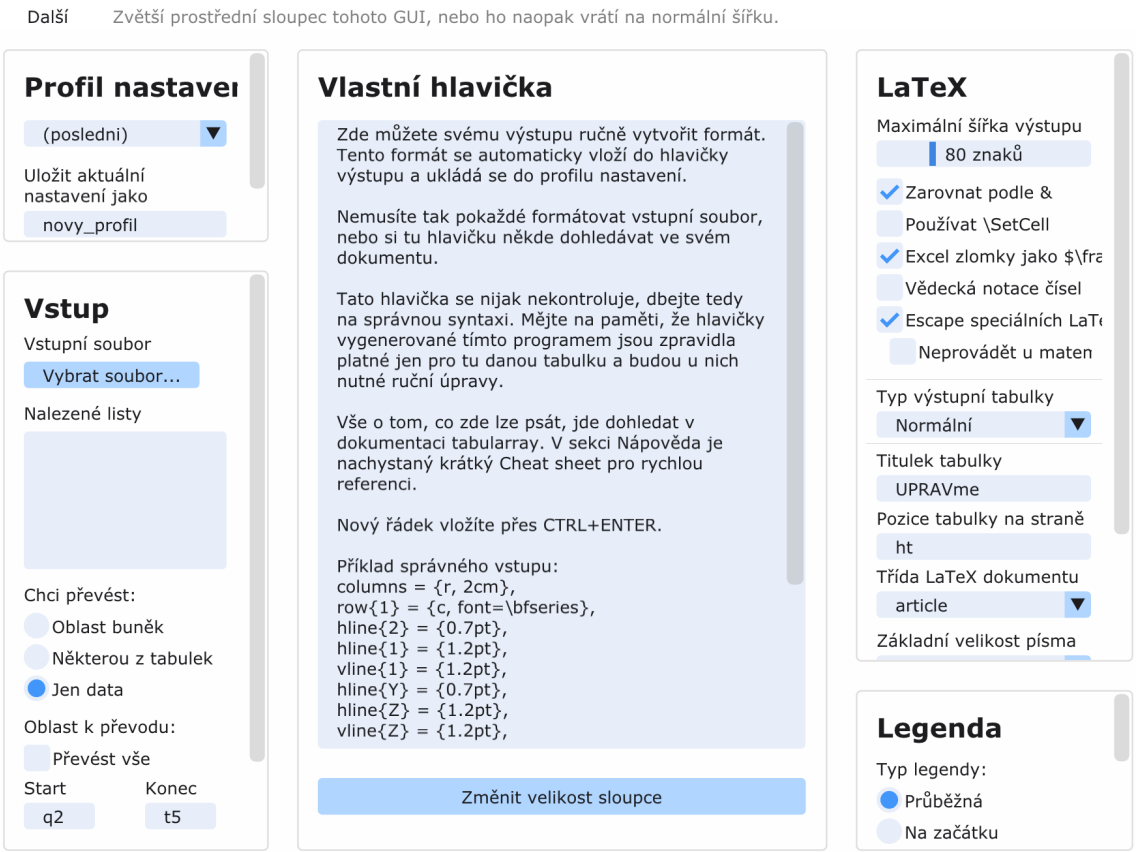


Obr. 29: Konkrétní příklad hledání totožných segmentů v rámci celé tabulky a tvorby výstupních řetězců okraje [vlastní]

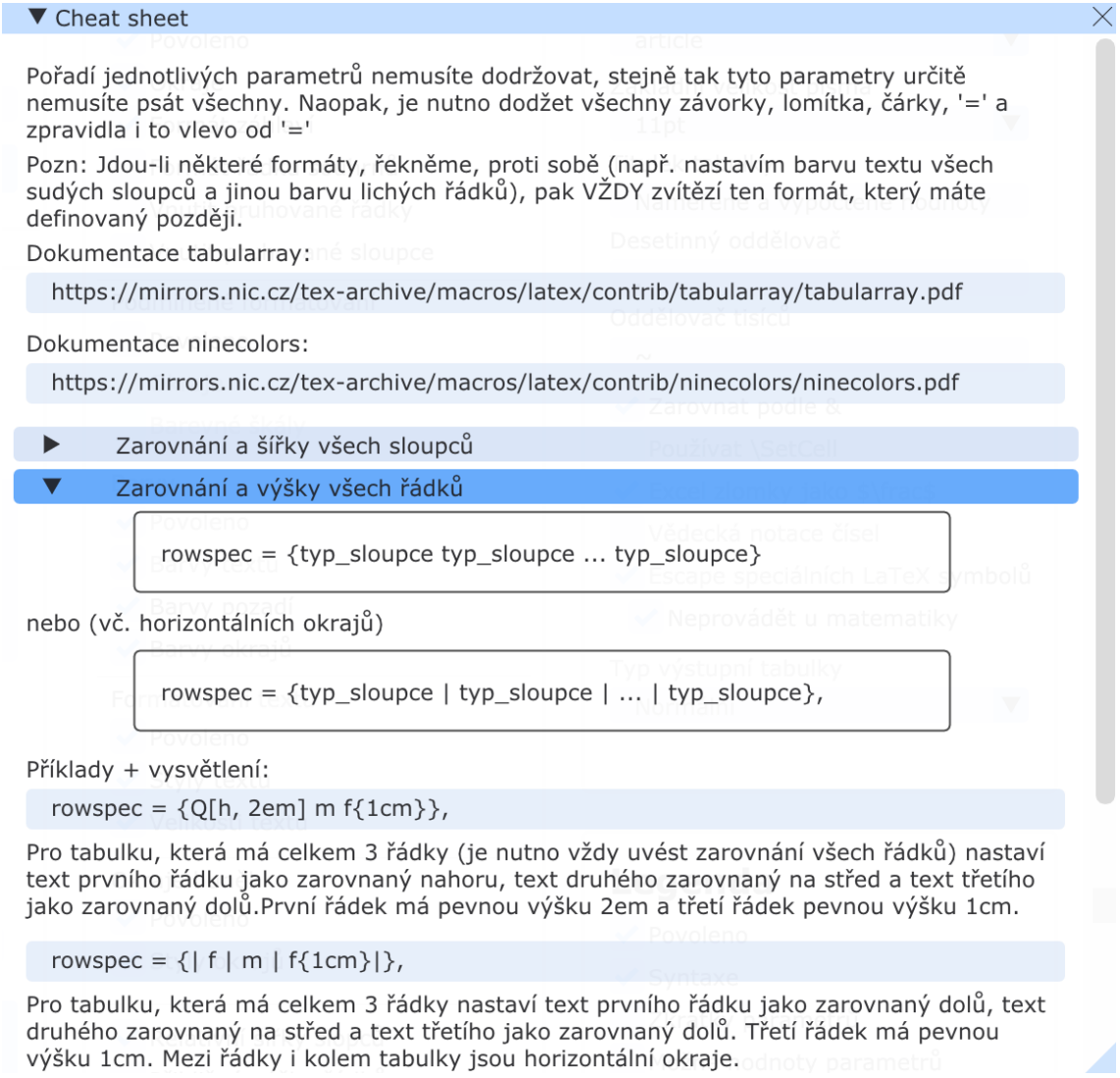
B Doplňující obrázky grafického rozhraní



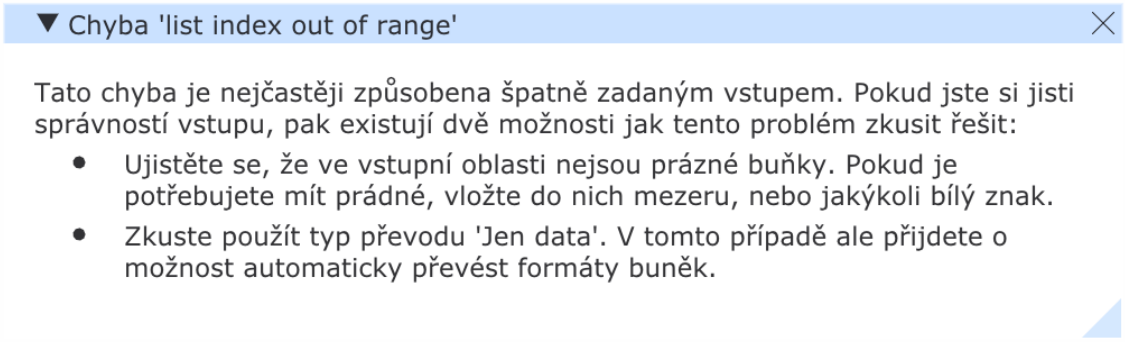
Obr. 30: Stavový řádek a menu *Další* [vlastní]



Obr. 31: Změna rozhraní při volbě převodu *Jen data* [vlastní]



Obr. 32: Ukázka *Cheat sheet* [vlastní]



Obr. 33: Ukázka jedné z položek nápovědy [vlastní]

C Prvky formátovacích řetězců

Tab. 17: Sekvence pro práci s textovými hodnotami [18]

Znak	Význam	Řetězec	Vstup	Výstup
`"text"`	Slouží k přidání textu mezi "" na konkrétní místo	`"abc"#.##`	0.5	abc0.5
`@`	Odkazuje na celou hodnotu uvnitř buňky.	`"\item "@`	xyz	\item xyz

Tab. 18: Sekvence pro obarvení textu [18]

Znak	Význam	Řetězec	Vstup	Výstup
`[Red]`	Obarví text červeně.	`[Red]#.0#`	0.05	.05
`[Red] [>100]`	Obarví text červeně při splnění podmínky.	`[Red] [>100]0.0`	99.21	99.2

Tab. 19: Znaky pro formát čísel za desetinným oddělovačem [18]

Znak	Význam	Řetězec	Vstup	Výstup
`0`	Počet nul udává počet desetinných míst, které se vždy zobrazí. V případě, že vstupní hodnota nemá požadovanou přesnost, přidá se nula.	`0.000`	8.35494	8.355
		`0.0`	1	1.0
`#`	Počet `#` udává počet desetinných míst, které se zobrazí v případě, že vstupní hodnota má dostatečnou přesnost. Pokud ji nemá, nic se nepřidává. Všechny `#` by měly být za `0`	`0.0##`	6.0035	6.004
		`0.###`	12.0055	12.006
		`0.##0`	12	12.0

Tab. 20: Znaky a sekvence pro formát čísel před desetinným oddělovačem [18]

Znak	Význam	Řetězec	Vstup	Výstup
`#,###`	Oddělí tisíce pomocí čárky.	`#,###.0#`	1122334	1 122 334.0
`,`	Každá čárka v této sekvenci vstupní hodnotu vydělí 1000	`0.0#,`	1122334	11 223.34
`0`	Počet nul udává délku (v číslicích) celočíselné části hodnoty. V případě, že je počet číslic menší, jsou doplněny počáteční nuly.	`0000`	16	0016
		`0000`	545	0545
`#`	Umožňuje vynechat hodnotu před desetinným oddělovačem v případě, že se jedná o číslo s absolutní hodnotou menší než 1.	`#.00#`	0.1	.10
		`0.00#`	0.1	0.10
		`#.0#`	12.524	12.52

Tab. 21: Znaky a sekvence pro speciální zápis čísel [18]

Znak	Význam	Řetězec	Vstup	Výstup
`E+00`	Vědecký zápis čísel.	`0.##E+00`	2000.655	2E+03
`?/?`	Zápis čísla co možno nepřesnějším zlomkem s jednociferným jmenovatelem	`?/?`	15.517	31/2
`??/13`	Zápis desetinné části pomocí třináctin (lze použít jakékoli jednociferné až trojciferné číslo). Pomocí `#` je zapsána celočíselná část, mezi `""` vložena je mezera aby byla i ve výstupu.	`#" "??/13`	6.841	6 11/13
`%`	Hodnota v procentech (vynásobena 100 a přidán znak %)	`0.0#%`	0.8546	85.46%

D Návod k instalaci konvertoru

Konvertor je dostupný pouze pro platformu Windows 10[1]. Je primárně distribuován formou předkompilovaného `.exe` souboru, který se nachází v příloze práce. Tento soubor stačí stáhnout, po spuštění by se mělo objevit grafické rozhraní konvertoru.

Po prvním spuštění bude s velkou pravděpodobností program zablokován operačním systémem kvůli neznámému vydavateli. Stačí zvolit *Další informace* a následně *Přesto spustit*.

V případě, že je po spuštění `.exe` souboru hlášena chyba, může pomoci program spustit jako správce.

Hlásí-li program chybějící knihovnu „`api-ms-...`“, pak je nutné doinstalovat Microsoft Visual C++ Redistributable z `https://docs.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist`.

Práce se zdrojovým kódem konvertoru pro pokročilejší uživatele
V příloze k textu práce je také umístěn zdrojový kód konvertoru. Chcete-li pracovat se zdrojovým kódem, nainstalujte si knihovny `imgui` a `openpyxl` pomocí

```
pip install imgui[glfw]
pip install openpyxl
```

Po instalaci těchto knihoven stačí spustit soubor `main.py`. Konvertor vyžaduje Python 3.9 (nebo novější), byl testován pro následující verze knihoven:

`glfw`	`3.3.7`
`imgui`	`1.4.1`
`openpyxl`	`3.0.9`
`pyglfw`	`2.5.3`
`pyopengl`	`3.1.6`
`python`	`3.9.12`

V případě, že je hlášena chyba „`Failed to load shared GLFW3 library`“, pomocí vyhledávání v kořenové složce instalace Pythonu, prostředí Anacondy nebo použitého venv najděte soubory `glfw3.dll` a `msvcr110.dll`. Oba soubory zkopírujte do složky

```
~\Lib\site-packages\glfw
```

kde „`~`“ označuje cestu ke kořenové složce.

[1] Starší verze Windows nejsou v době psaní tohoto textu podporovány. S Windows 11 by neměly být potíže, ale nebylo možné jej otestovat.