Zpravodaj Československého sdružení uživatelů TeXu

Vít Novotný

Sazba textu označkovaného v jazyce Markdown uvnitř TeXových dokumentů

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 26 (2016), No. 1-4, 78–93

Persistent URL: http://dml.cz/dmlcz/150247

Sazba textu označkovaného v jazyce Markdown uvnitř TEXových dokumentů

Vít Novotný

Článek pojednává o novém makrobalíku pro formáty odvozené od plain TEXu, který umožňuje do sázených dokumentů přímo vkládat pasáže v odlehčeném značkovacím jazyce Markdown. Autor popisuje motivaci pro vznik balíku a způsob, jakým balík pracuje. Použití je ilustrováno na příkladech.

Klíčová slova: Markdown, odlehčené značkování, Lua, plain TEX, LATEX, ConTEXt, Pandoc

Úvod

TEX a přidružené programy jsou vhodným nástrojem pro sazbu mnoha druhů dokumentu, nemusí však nutně být vhodným jazykem pro přípravu jejich obsahu. Značkovací jazyky založené na SGML a XML umožňují zachytit strukturu dokumentu, aniž by hrozilo riziko vnesení chybného příkazu, který znemožní sazbu, jako je tomu u TEXu. Podstatnou výhodou je i fakt, že dokumenty lze dále publikovat a zpracovávat i mimo TEXový svět. Při přípravě hladšího materiálu lze pak využít i tzv. odlehčené značkovací jazyky. Heslem dne je zde vizuální čistota, snadný zápis a malý poměr značkování vůči textu. Jedním z odlehčených značkovacích jazyků, původně určeným pro přípravu HTML dokumentů, je Markdown (Gruber, 2013). Typickým nástrojem pro převod Markdownu (a jeho nejrůznějších dialektů) do formátů TEXu je Pandoc (MacFarlane, 2016).

Pandoc je víceúčelový nástroj, který umožňuje v první řadě převod Markdownu na obecnější jazyky (jako je LATEX, ConTEXt[1], HTML nebo XML Docbook) a do řady výstupních formátů (jako je ODF, OOXML nebo PDF). Konfigurovat parametry převodu a přidávat dodatečná metadata lze skrz parametry zadávané na příkazové řádce a skrz metabloky v jazyce YAML zapsané v záhlaví dokumentu. Při převodu na obecnější jazyky je Pandoc schopný generovat buďto pouze fragmenty, které uživatel následně vloží do těla svého dokumentu, nebo ucelené dokumenty. V druhém případě dochází k využití šablon, uvnitř kterých lze používat i jednoduchý makrojazyk; ten má přístup k metadatům zadaným na příkazové řádce a v YAML metablocích. Detailní rozbor schopností Pandocu ve vztahu k TEXu lze nalézt ve starším článku z TUGboatu (Dominici, 2014).

[1]Program Pandoc podporuje pouze převod do TEXových formátů ConTEXtu a LATEXu. Jazyk Markdown je však natolik oblíbený, že existují i programy pro jeho převod na makra alternativních makrobalíků. Takovéto podpoře se těší např. balík OPmac (Horáček, 2016).

 DOI: 10.5300/2016-1-4/78

Pandoc je bezesporu silný a užitečný nástroj při přípravě dokumentů s řadou výstupních formátů. Z pohledu uživatele, pro kterého je hlavním výstupním formátem TEX, má však i množství slabin. Při převodu například nelze jednoduše ovlivnit výstupní makra. Pro následující text v Markdownu:

```
# Úvod {#uvod}
[...]
Po zkušenostech s~nástrojem *Pandoc* jsem se rozhodl připravit
\TeX ový makrobalík, který by netrpěl neduhy zmíněnými
v~[úvodu](#uvod).
```

vygeneruje Pandoc 1.17.0.3 následující LATEXový výstup:

```
\hypertarget{uvod}{\section{Úvod}\label{uvod}}
[...]
Po zkušenostech s~nástrojem \emph{Pandoc} jsem se rozhodl
připravit \TeX ový makrobalík, který by netrpěl neduhy zmíněnými
v~\protect\hyperlink{uvod}{úvodu}.
```

Uživatel může makra `\hypertarget`, `\section`, `\label`, `\protect` a `\hyperlink` lokálně zadefinovat tak, aby jednotlivé markdownové značky odpovídaly zamýšlenému užití; nemá však garantováno, že tento výstup zůstane zachován i v budoucích verzích nástroje.

TEXové příkazy a jejich parametry jsou ve výstupním dokumentu zachovány beze změny, což autorovi umožňuje vnést do textu dokumentu chybné příkazy, které znemožní sazbu. Při použití stroje LuaTEXu bez volby `-safer`, nebo TEXového stroje s povoleným přístupem k příkazové řádce operačního systému se pak jedná i o bezpečnostní riziko. Pro detekci TEXových příkazů a jejich parametrů je použita heuristika; v textu, který není vyhodnocen jako TEXový příkaz, jsou veškeré speciální znaky plain TEXu (včetně vlnek) nahrazeny kódem, který je v příslušném TEXovém formátu vysází. Toto komplikuje vkládání TEXových příkazů, které nejsou ve formátu `\příkaz{`⟨*parametr*⟩`}`, a díky využití heuristiky není garantováno, že stejný výstup zůstane zachován i v budoucích verzích nástroje. Stejný problém sužuje i matematiku ve vstupním dokumentu.

Knuthův TEX je od roku 1989 pouze udržován. Dokumenty psané ve stabilním formátu, jako je plain TEX, a nezávislé na externích makrobalících proto při překladu TEXem dávají stejné výsledky nezávisle na verzi TEXové distribuce. Pokud uživatel využije aktivně vyvíjené TEXové formáty, makrobalíky a stroje, vyžadují již dokumenty údržbu a s novými verzemi TEXové distribuce se může měnit jejich výstup. Pokud však uživatel ve svých dokumentech nevyužívá systémové fonty ani makrobalíky spouštějící programy mimo TEXovou distribuci, měl by obdržet při překladu se stejnou verzí TEXové distribuce stejné výsledky. Pandoc je externí nástroj, který není obsažen v TEXových distribucích. Při jeho využití tedy nedává ani verze TEXové distribuce garanci stabilního výstupu překládaného dokumentu.

Dalším z nepříjemných důsledků odtržení Pandocu od TEXových distribucí je jeho absence na platformách vybudovaných nad TEXovými distribucemi. Služby jako `http://overleaf.com/` a `http://www.sharelatex.com/` umožňují spolupráci několika autorů na jednom LATEXovém dokumentu v reálném čase. Podobně jako na wiki sítích, i zde by často dávalo smysl využít okleštěný, ale snadno přístupný formát Markdownu.

Vznik makrobalíku Markdown

Po zkušenostech s nástrojem Pandoc jsem se v listopadu roku 2015 rozhodl připravit TEXový makrobalík, který by netrpěl neduhy zmíněnými v úvodu. Balík měl umožnit volně prokládat TEXový zdrojový kód textem naznačkovaným v Markdownu. Vykreslování jednotlivých markdownových značek pak mělo být řízeno TEXovými makry. Ty by sice obsahovaly rozumnou výchozí definici pro příslušný TEXový formát, ale byly by snadno předefinovatelné uživatelem.

Prvotní otázkou bylo, jakou technologii pro vývoj makrobalíku použít. V raných fázích návrhu jsem vytvořil prototyp pro plain TEX, který pomocí aktivních znaků rozpoznával redukovanou variantu jazyka Markdown. Podobné balíky již existují (Lücк, 2015), ale typicky rozpoznávají pouze jednoduché regulární a LL(k) gramatiky garantující časovou složitost $\mathcal{O}(n)$ a prostorovou složitost $\mathcal{O}(k)$. Jazyk Markdown však obsahuje ostře kontextové prvky. Existující parsery reprezentují jazyk PEG gramatikami, nebo provádí rekurzivní sestup, a na vybraných vstupech dosahují díky backtrackingu časové složitosti $\mathcal{O}(2^n)$, nebo díky memoizaci prostorové složitosti $\mathcal{O}(n)$ (Ford, 2004, sekce 6).

Alternativou bylo napsat parser odděleně v jiném programovacím jazyce, případně použít již existující parser. Inspirací mi byl LATEXový makrobalík minted (Poore, 2016), který slouží k zvýrazňování syntaxe zdrojových kódů. Minted předává zvýrazňovaný kód pythonové knihovně Pygments, která jej rozparsuje, zvýrazní a výstup v LATEXu navrátí zpět makrobalíku minted. Komunikace probíhá skrz pomocné soubory a výstupní proud 18 (příkaz `\write18`), přes který moderní TEXové stroje zpřístupňují příkazovou řádku operačního systému. Namísto Pythonu jsem však chtěl použít jazyk Lua, jehož interpret `texlua` se již nachází v TEXových distribucích. V případě použití LuaTEXu by navíc bylo možné kód spouštět přímo, aniž by bylo třeba využívat `\write18`, jehož povolení je bezpečnostním rizikem.

Začal jsem se pídit po existujících parserech Markdownu psaných v jazyce Lua. Podstatné pro mě bylo, aby parser závisel výhradně na knihovnách, které jsou staticky přilinkované k stroji LuaTEXu, aby jej bylo možné upravit a sublicencovat pod LPPL 1.3 a aby parser vyhovoval specifikaci jazyka Markdown. Jako vyhovující se ukázal balík Lunamark (MacFarlane, 2012), který je shodou okolností dílem autora nástroje Pandoc. Lunamark závisel na knihovnách LPeg, Cosmo, Selene

Unicode a Alt-getopt, z nichž klíčovými pro fungování parseru byly LPeg a Selene Unicode. Obě tyto knihovny byly staticky přilinkovány k stroji LuaTEXu (LuaTEX dev. team, 2016, sekce 3.3). Kód balíku byl zároveň uvolněn pod permisivní licencí MIT a měl k sobě připojenou sadu kvalitních regresních testů.

Lunamark sestával z 44 souborů v jazyce Lua. Jednalo se o moduly pro výstupní formáty, spustitelné soubory tvořící rozhraní pro příkazovou řádku a soubory s pomocnými definicemi. Z neaktivity na repozitářích Lunamarku a komunikace s autorem bylo zřejmé, že balík je již pouze udržován; mohl jsem tedy provést výrazné změny, aniž bych si zavíral cestu k začleňování budoucích aktualizací. Balík jsem zredukoval na jeden Lua soubor vhodný pro zanesení do TEXových distribucí. Parser bylo třeba dále upravit tak, aby ve vstupu nerozpoznával HTML kód a aby na výstupu místo materiálu k sazbě navracel syntaktický strom vstupního dokumentu představovaný TEXovými makry. Na vývoj byly vyhrazeny prostředky v rámci programu pro podporu studentských výzkumných a vývojových projektů na Fakultě informatiky Masarykovy univerzity v Brně. Kolem přepracovávaného balíku jsem začal budovat makrobalík pro TEXový formát plain TEXu a v zájmu uživatelské přívětivosti i pro formáty LATEXu a ConTEXtu ve verzích Mark II a Mark IV. Makrobalík jsem nazval Markdown. První veřejnou verzi jsem na archiv `http://ctan.org/` umístil v červnu roku 2016 (Novotný, 2016).

Architektura a použití makrobalíku Markdown

Rozhraní pro jazyk Lua

Na nejnižší úrovni je možné balík využívat z jazyka Lua. Rozhraní poskytuje metody pro konverzi textu v kódování UTF-8 a v jazyce Markdown do zdrojového textu plain TEXu a je implementováno v souboru `markdown.lua`. Pokud vytvoříme soubor `skript.lua` s následujícím obsahem:

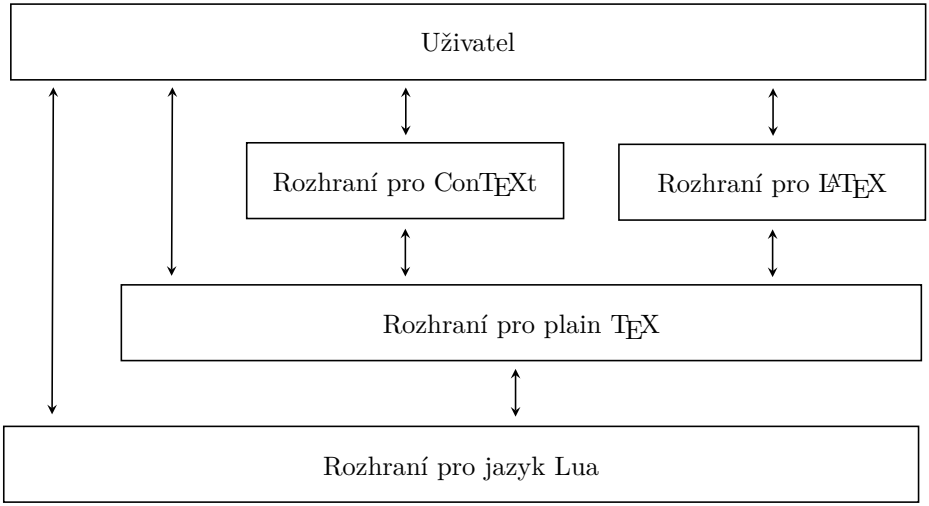
```
#/usr/bin/env texlua
local kpse = require"kpse"
kpse.set_program_name"kpsewhich"
local markdown = require"markdown"
local convert = markdown.new()
print(convert"Makrobalík jsem nazval *Markdown*.")
```

a přeložíme jej příkazem `texlua skript.lua`, měli bychom obdržet výstup v následujícím tvaru:

```
\input"./1cc0428cde58007078df6dfa2818ef75.md.tex"\relax
```

Soubor vkládaný příkazem `\input` pak obsahuje následující kód:

```
Makrobalík jsem nazval \markdownRendererEmphasis{Markdown}.\relax
```



Obrázek 1: Blokový diagram znázorňující architekturu balíku Markdown

Následující kód pak ilustruje použití rozhraní pro jazyk Lua v rámci stroje LuaTEXu při použití TEXového formátu LATEXu:

```
\documentclass{minimal}
\usepackage{fontspec}
\begin{document}
  \let\markdownRendererEmphasis=\emph
  \directlua{
    local markdown = require"markdown"
    local convert = markdown.new()
    tex.sprint(convert[[
      Makrobalík jsem nazval *Markdown*.
    ]]) }
\end{document}
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `lualatex priklad` obdržíme na výstupu dokument obsahující text: „Makrobalík jsem nazval *Markdown*.“

Metodě `markdown.new()` lze jako první argument předat tabulku obsahující parametry překladu z Markdownu. Důležitým parametrem je volba `hybrid`, která umožňuje konfigurovat, jakým způsobem se parser chová k speciálním znakům plain TEXu ve vstupním textu. Ve výchozím nastavení parser veškeré speciální znaky ve vstupu nahrazuje makry `\markdownRenderer`⟨*Značka*⟩, což je ideální

pro zpracování externích dokumentů, kterým nechceme povolit spouštět TEXové příkazy. Naopak při ruční přípravě dokumentů nám může přijít vhod hybridní režim, ve kterém parser speciální znaky plain TEXu ponechává beze změny. Následující kód v jazyce Lua:

```
#/usr/bin/env texlua
local kpse = require"kpse"
kpse.set_program_name"kpsewhich"

local markdown = require"markdown"
local convert_safe   = markdown.new()
local convert_unsafe = markdown.new({ hybrid = true })
local input  = [[
  Víme, že $1+2=3$, ale \TeX{} nám to umožní i~nádherně vysázet.
]]

print(convert_safe(input))
print(convert_unsafe(input))
```

nám dává při překladu interpretem `texlua` na výstupu dvojici dokumentů. V prvním jsou veškeré výskyty speciálních znaků plain TEXu nahrazeny makry:

```
Víme, že \markdownRendererDollarSign{}1+2=3%
\markdownRendererDollarSign{}, ale \markdownRendererBackslash{}%
TeX\markdownRendererLeftBrace{}\markdownRendererRightBrace{}
nám to umožní i\markdownRendererTilde{}nádherně vysázet.\relax
```

Druhý je pak až na závěrečný `\relax` totožný se vstupem.

Nabídka značek v základním Markdownu je značně omezená. Skrze parametry překladu lze proto aktivovat i různorodá rozšíření syntaxe. V době přípravy článku balík podporuje rozšíření pro citace, poznámky pod čarou, zdrojové kódy s vyznačením názvu programovacího jazyka a definiční seznamy:

```
#/usr/bin/env texlua
local kpse = require"kpse"
kpse.set_program_name"kpsewhich"

local markdown = require"markdown"
local convert = markdown.new({
  citations       = true, -- Rozšíření pro citace
  footnotes       = true, -- Rozšíření pro poznámky pod čarou
  fencedCode      = true, -- Rozšíření pro zdrojové kódy
  definitionLists = true, -- Rozšíření pro definiční seznamy
})
```

```
local input  = [[
  @doe09 tvrdí, že časová složitost následujícího řadicího
  algoritmu je lineárně logaritmická[^sleepsort]:

  ``` sh
  #!/bin/sh
  for N; do
    (sleep $N; echo $N) &
  done
  wait
  ``````

  [^sleepsort]: Kvadratická, pokud uvážíme práci plánovače jádra.

  žába
  :   slizká
  :   zelená [viz -@smith12, s. 123]
  :   kvákající
]]

print(convert(input))
```

Při překladu interpretem `texlua` pak obdržíme následující dokument:

```
\markdownRendererTextCite{1}+{}{}{doe09} tvrdí, že časová
složitost následujícího řadicího algoritmu je lineárně
logaritmická\markdownRendererFootnote{Kvadratická, pokud uvážíme
práci plánovače jádra.}:\markdownRendererInterblockSeparator
{}\markdownRendererInputFencedCode{./04602c8f5c223967cfe7aa8282d%
6f82f.verbatim}{sh}\markdownRendererInterblockSeparator
{}\markdownRendererDlBeginTight\markdownRendererDlItem{žába}%
\markdownRendererDlDefinitionBegin slizká%
\markdownRendererDlDefinitionEnd
\markdownRendererDlDefinitionBegin zelená \markdownRendererCite
{1}-{viz}{s.\markdownRendererNbsp{}123}{smith12}%
\markdownRendererDlDefinitionEnd
\markdownRendererDlDefinitionBegin kvákající%
\markdownRendererDlDefinitionEnd\markdownRendererDlItemEnd
\markdownRendererDlEndTight\relax
```

Výčet známých parametrů nalezneme v dokumentaci (Novotný, 2016, sekce 2.1.2).

Rozhraní pro plain TEX

Pokud zamýšlíme výstup parseru bezprostředně sázet, můžeme s výhodou využít plainTEXové rozhraní, které nás odstíní od komunikace s interpretem jazyka Lua. To oceníme především při práci s TEXovými stroji, které skriptování v jazyce Lua přímo neumožňují (pdfTEX, XƎTEX). PlainTEXové rozhraní je implementováno v souboru `markdown.tex`, který zavedeme pomocí příkazu `\input markdown`.

Rozhraní definuje makra `\markdownRenderer`⟨*Značka*⟩, která se mohou nacházet ve výstupu metod luového rozhraní. Makra `\markdownRenderer`⟨*Značka*⟩ implicitně expandují na makra `\markdownRenderer`⟨*Značka*⟩`Prototype`, která pak obsahují výchozí definici pro jednotlivé markdownové značky. Toto rozdělení má následující význam: makra `\markdownRenderer`⟨*Značka*⟩ jsou určena k předefinování uživatelem, zatímco makra `\markdownRenderer`⟨*Značka*⟩`Prototype` jsou určena pro tvůrce makrobalíků. Ti jim mohou nastavit výchozí hodnoty, které považují za rozumné, aniž by mohlo dojít k přepsání uživatelovy konfigurace:

```
% Tento makrobalík slouží pro sazbu kuchařských receptů.
\def\nadpis#1{Recept: #1}
% [...]
\ifx\markdownVersion\undefined\else
  \let\markdownRendererHeadingOnePrototype=\nadpis
\fi
```

Úplný výčet značek lze nalézt v dokumentaci (Novotný, 2016, sekce 2.2.3).

PlainTEXové rozhraní rovněž poskytuje přístup k parametrům překladu z luového rozhraní. Konkrétně platí, že pro libovolný ⟨*parametr*⟩ je dostupné makro `\markdownOption`⟨*Parametr*⟩, které může uživatel předefinovat a ovlivnit překlad:

```
\def\markdownOptionHybrid{true}%
\def\markdownOptionFencedCode{true}%
```

Kromě parametrů překladu umožňuje plainTEXové rozhraní nastavovat i další parametry, které souvisí s komunikací s interpretem jazyka Lua. Úplný výčet rozpoznávaných maker lze nalézt v dokumentaci (Novotný, 2016, sekce 2.2).

Text v Markdownu lze zapsat mezi příkazy `\markdownBegin` a `\markdownEnd`:

```
\input markdown
\def\markdownRendererEmphasis#1{\it#1}%
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\bye
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `pdfcsplain -shell-escape priklad` obdržíme na výstupu dokument obsahující text „Makrobalík jsem nazval *Markdown*.“

S použitím příkazů `\markdownBegin` a `\markdownEnd` se pojí několik nedostatků. Markdown například umožňuje do výstupu vložit řádkový zlom tak, že uživatel na konec řádku ve vstupu přidá dvě a více mezer. V TEXu jsou však veškeré mezery na konci řádku zahozeny ještě před tím, než se text objeví ve vstupním bufferu (Knuth, 1986, strana 46). Při použití příkazů `\markdownBegin` a `\markdownEnd` je vstupní text v Markdownu načítán TEXem a řádkové zlomy tedy není možné detekovat. Další drobné nedostatky příkazů `\markdownBegin` a `\markdownEnd` jsou popsány v dokumentaci (Novotný, 2016, sekce 2.2.1).

Pokud máme markdownový text uložený v externím souboru, můžeme jej do TEXového dokumentu vložit pomocí příkazu `\markdownInput`:

```
\input markdown
\markdownInput{priklad.md}%
\bye
```

Nedostatky spojené s použitím příkazů `\markdownBegin` a `\markdownEnd` se na příkaz `\markdownInput` nevztahují.

Rozhraní pro LaTeX

TEXový formát LaTeXu implementuje většinu maker plain TEXu (Braams et al., 2016, sekce 9). V rámci LaTeXu však nelze používat přímo plainTEXové rozhraní:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\input markdown
\begin{document}
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\end{document}
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `lualatex priklad` obdržíme následující chybu:

```
Module luatexbase Error: Attempt to use callback.register()
(luatexbase)             directly on input line 6
```

PlainTEXová implementace pro LuaTEX využívá při načítání vstupního textu háček `process_input_buffer` (LuaTEX dev. team, 2016, sekce 8.3.1). LaTeX však pro načítání textu poskytuje vlastní rozhraní (Braams et al., 2016, sekce 73.4).

Pokud dokument přeložíme příkazem `pdflatex -shell-escape priklad`, obdržíme následující chybu:

```
! Package inputenc Error: Unicode char ?kj (U+E2)
(inputenc)                not set up for use with LaTeX.
```

```
l.1 Makrobal?k j
                sem nazval \markdownRendererEmphasis{Markdown}...
```

PlainTEXová implementace pro stroj pdfTEXu je schopna si poradit se speciálními znaky plain TEXu ve vstupu, ale nepočítá s použitím LATEXového balíku inputenc, který počáteční bajty znaků mimo ASCII učiní TEXovými aktivními znaky.

Z těchto důvodů je vhodné použít přímo LATEXové rozhraní. To je implementováno v souboru `markdown.sty`, který zavedeme pomocí příkazu `\usepackage[`⟨*parametry*⟩`]{markdown}`. Jak znázorňuje obrázek 1 ze str. 82, zavádí LATEXová implementace zároveň plainTEXovou implementaci. Můžeme tedy přímo používat makra z plainTEXového rozhraní:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{markdown}
\let\markdownRendererEmphasis=\emph
\begin{document}
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\end{document}
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazy `lualatex priklad` nebo `pdflatex -shell-escape priklad` obdržíme na výstupu dokument obsahující text: „Makrobalík jsem nazval *Markdown*.“

Nad úroveň plainTEXového rozhraní zavádí LATEXové rozhraní navíc příkaz `\markdownSetup{`⟨*parametry*⟩`}`, LATEXová prostředí `\begin{markdown}`...`\end{markdown}` a `\begin{markdown*}{`⟨*parametry*⟩`}`...`\end{markdown*}` a zakrývá plainTEXový příkaz `\markdownInput{`⟨*vstupní soubor*⟩`}` svým příkazem `\markdownInput[`⟨*parametry*⟩`]{`⟨*vstupní soubor*⟩`}`.

Pomocí příkazů `\usepackage[`⟨*parametry*⟩`]{markdown}` a `\markdownSetup{`⟨*parametry*⟩`}` můžeme nastavovat parametry překladu z luového rozhraní:

```
\usepackage[
  citations,        %% Rozšíření pro citace
  footnotes,        %% Rozšíření pro poznámky pod čarou
]{markdown}
\markdownSetup{
  fencedCode,       %% Rozšíření pro zdrojové kódy
  definitionLists, %% Rozšíření pro definiční seznamy
}
```

Stejně tak můžeme pomocí parametrů `renderers` a `rendererPrototypes` nastavovat makra z plainTEXového rozhraní ve tvaru `\markdownRenderer`⟨*Značka*⟩ a `\markdownRenderer`⟨*Značka*⟩`Prototype`:

```
\markdownSetup{
  renderers = {
    emphasis = {\emph{#1}},
  }, rendererPrototypes = {
    link = {\href{#1}{#3}},
  }
}
```

Vzhledem ke způsobu, jakým LaTeX 2$_\varepsilon$ čte parametry příkazu `\usepackage[`⟨*parametry*⟩`]{markdown}`, nemohou ⟨*parametry*⟩ obsahovat víceodstavcový text ani argumenty maker (například `#1`). V praxi toto činí parametry `renderers` a `rendererPrototypes` nepoužitelnými, a proto jsou v rámci příkazu `\usepackage` zakázány.

LaTeXové prostředí `\begin{markdown}`...`\end{markdown}` je svou funkcí totožné s příkazy `\markdownBegin` a `\markdownEnd` z plain TEXového rozhraní. Zajímavějšími pro nás budou LaTeXové prostředí `\begin{markdown*}{`⟨*parametry*⟩`}`...`\end{markdown*}` a příkaz `\markdownInput[`⟨*parametry*⟩`]{`⟨*vstupní soubor*⟩`}`, které nám podobně jako příkaz `\markdownSetup{`⟨*parametry*⟩`}` umožňují nastavovat parametry překladu z luového rozhraní a makra `\markdownRenderer`⟨*Značka*⟩ a `\markdownRenderer`⟨*Značka*⟩`Prototype` z plainTEXového rozhraní:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{markdown}
\markdownSetup{renderers = {
  emphasis = {\textbf{#1}},
}}
\begin{document}
\begin{markdown*}{renderers = {
  emphasis = {\emph{#1}}
}}
  Makrobalík jsem nazval *Markdown*.
\end{markdown*}
\begin{markdown*}{hybrid}
  Víme, že $1+2=3$, ale \TeX{} nám to
  umožní i~nádherně *vysázet*.
\end{markdown*}
\end{document}
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `pdflatex -shell-escape priklad` obdržíme na výstupu dokument obsahující text „Makrobalík jsem nazval *Markdown*. Víme, že $1 + 2 = 3$, ale TEX nám to umožní i nádherně **vysázet**.“ Vidíme, že parametry LATEXového prostředí `\begin{markdown*}{`⟨*parametry*⟩`}`...`\end{markdown*}` jsou zpracovány způsobem, který odpovídá naší intuici, a mají pouze lokální platnost. Příkaz `\markdownInput[`⟨*parametry*⟩`]{`⟨*vstupní soubor*⟩`}` se chová analogicky.

LATEXové rozhraní se snaží nastavit rozumné výchozí hodnoty pro makra `\markdownRenderer`⟨*Značka*⟩`Prototype` z plainTEXového rozhraní. Při běžném použití proto můžeme konfiguraci balíku přeskočit a rovnou se pustit do psaní. Výchozí hodnoty maker jsou popsány v dokumentaci (Novotný, 2016, sekce 3.3.4).

Rozhraní pro ConTEXt

TEXové formáty ConTEXt Mark II a Mark IV implementují většinu maker plain TEXu. Podobně jako u LATEXu však nelze používat přímo plainTEXové rozhraní:

```
\input markdown
\starttext
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\stoptext
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `texexec --passon=-shell-escape priklad` obdržíme následující chybu:

```
! Undefined control sequence.
\markdownReadAndConvert ...de `##1=12}\dospecials
                                                  \catcode `\ ...
l.3 \markdownBegin
```

TEXový formát ConTEXt Mark II nedefinuje plainTEXový příkaz `\dospecials`. Pokud definici doplníme:

```
\input markdown
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\_\do\%\do\~}%
\starttext
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\stoptext
```

a překlad opakujeme, podaří se nám již překlad dokončit. Pokud však dokument přeložíme příkazem `context priklad`, obdržíme v terminálu následující varování:

```
system> callbacks > not registering frozen 'process_input_buffer'
```

a překlad nikdy neskončí. PlainTEXová implementace využívá při načítání vstupního textu v stroji LuaTEXu háček `process_input_buffer` (LuaTEX dev. team, 2016, sekce 8.3.1). Podobně jako LATEX však formát ConTEXt Mark IV háčky LuaTEXu využívá interně a pro načítání textu poskytuje vlastní rozhraní.

Z těchto důvodů je vhodné použít přímo ConTEXtové rozhraní. To je implementováno v souboru `t-markdown.tex`, který zavedeme příkazem `\usemodule [t][markdown]`. Jak znázorňuje obrázek 1 ze str. 82, zavádí ConTEXtová implementace zároveň plainTEXovou implementaci. Můžeme tedy přímo používat makra z plainTEXového rozhraní:

```
\usemodule[t][markdown]
\let\markdownRendererEmphasis=\emph
\starttext
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\stoptext
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `texexec --passon=-shell-escape priklad` nebo `context priklad` obdržíme na výstupu dokument obsahující text: „Makrobalík jsem nazval *Markdown*.“

Nad úroveň plainTEXového rozhraní zavádí rozhraní pro ConTEXt příkazy `\startmarkdown` a `\stopmarkdown`, které jsou svou funkcí totožné s příkazy `\markdownBegin` a `\markdownEnd` z plainTEXového rozhraní. Kromě těchto maker není pro ConTEXt definováno žádné další rozšiřující rozhraní.

Podobně jako LATEXové rozhraní se i rozhraní pro ConTEXt snaží nastavit rozumné výchozí hodnoty pro makra `\markdownRenderer`⟨*Značka*⟩`Prototype` z plainTEXového rozhraní. Při běžném použití proto můžeme konfiguraci balíku přeskočit a rovnou se pustit do psaní. Výchozí hodnoty maker jsou popsány v dokumentaci (Novotný, 2016, sekce 3.4.3).

Specifika československé sazby

V češtině a slovenštině je typografickou chybou ponechat neslabičné předložky (k, s, v, z) na koncích řádků. V TEXu tento problém řešíme ručním vložením nezlomitelné mezery mezi předložku a následující slovo, použitím programu vlna (Olšák, 2010), nebo použitím LATEXových makrobalíků XƎVlna (Wagner, 2013) a encxvlna (Olšák; Wagner, 2014).

Pokud pro překlad dokumentu využívajícího makrobalík Markdown použijeme stroj XeTeX, nebo encTeX, stačí nám zavést makrobalíky XeVlna, nebo encxvlna a nezlomitelné mezery budou automaticky doplněny na příslušná místa. Stejně tak můžeme, pokud makrobalík Markdown používáme v hybridním režimu, vložit do dokumentu nezlomitelné mezery ručně nebo pomocí programu vlna.

Pokud však makrobalík Markdown používáme mimo hybridní režim a zároveň pro sazbu nevyužíváme stroje XeTeX a encTeX, stává se vkládání nezlomitelných mezer mírně komplikovanějším. Při překladu z Markdownu se totiž veškeré znaky vlnky (v plain TeXu představující nezlomitelné mezery) na vstupu přeloží na makro `\markdownRendererTilde`, které ve výchozím nastavení znak vlnky vysází. Nejidiomatičtějším řešením je změnit definici makra `\markdownRendererTilde` tak, aby expandovalo na nezlomitelnou mezeru:

```
\input markdown
\let\markdownRendererTilde=~
\markdownBegin
  Nyní v~textu mohu zadávat nezlomitelné mezery.
\markdownEnd
\bye
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `pdfcsplain -shell-escape priklad` obdržíme na výstupu dokument obsahující text „Nyní v textu mohu zadávat nezlomitelné mezery.“ Nevýhodou tohoto řešení je, že autorovi dokumentu upíráme možnost vysázet znak vlnky.

Pokud nás to trápí, můžeme se uchýlit k alternativnímu řešení. Makrobalík Markdown si přeložené soubory odkládá do adresáře zadaného makrem `\markdownOptionCacheDir` (Novotný, 2016, sekce 2.2.2.1), které implicitně expanduje na `_markdown_\jobname`. Toho využijeme v následujícím dokumentu:

```
\input markdown
\markdownBegin
  Nyní v textu mohu používat vlnky: ~.
\markdownEnd
\bye
```

Pokud dokument uložíme do souboru `priklad.tex` a přeložíme jej příkazem `pdfcsplain -shell-escape priklad`, pak nám v adresáři `_markdown_priklad` vznikne soubor s názvem ve tvaru ⟨*haš vstupu*⟩`.md.tex` a s následujícím obsahem:

```
Nyní v textu mohu používat vlnky: \markdownRendererTilde{}.\relax
```

Pokud tento soubor zpracujeme programem vlna a opět dokument přeložíme příkazem `pdfcsplain -shell-escape priklad`, pak makrobalík Markdown použije náš upravený soubor `_markdown_priklad/`⟨*haš vstupu*⟩`.md.tex` a na výstupu obdržíme dokument obsahující text: „Nyní v textu mohu používat vlnky: ~.“

Reference

Johannes Braams, David Carliste, Alan Jeffrey a kol. *The LaTeX 2ε Sources* [on-line]. 2016. [cit. 2016-06-02]. Dostupné na: `http://mirrors.ctan.org/macros/latex/base/source2e.pdf`.

Dominici, Massmiliano. An overview of Pandoc [on-line]. *TUGboat*, 2014, **35**(1), s. 44–50. [cit. 2016-08-15]. (ISSN 0896-3207.) Dostupné na: `http://tug.org/TUGboat/tb35-1/tb109dominici.pdf`.

Ford, Brian. Parsing expression grammars: A recognition-based syntactic foundation. *ACM SIGPLAN Notices*, New York, NY, USA : ACM, 2014, **39**(1), s. 111–122. (ISSN 0362-1340.). Dostupné z DOI: `10.1145/982962.964011`.

Gruber, John. *Markdown* [on-line]. 2013. [cit. 2016-08-15]. Dostupné na: `http://daringfireball.net/projects/markdown/`.

Horáček, Michal. *Markdown to OPmac converter* [on-line]. *Ver. 2dd262d 2016-06-24*. 2016. Dostupné na: `https://bitbucket.org/horacmi/md2opmac`.

Knuth, Donald Ervin. *The TeXbook*. 3. vyd. Boston : Addison-Westley, 1986. ix + 479 s. ISBN 0-201-13447-0.

LuaTeX dev. team. *LuaTeX reference manual* [on-line]. 2016. [cit. 2016-12-23]. 222 s. Dostupné na: `http://www.luatex.org/svn/trunk/manual/luatex.pdf`.

Lück, Uwe. *nicetext: Minimal markup for simple text (Wikipedia style) and documentation* [on-line]. *Ver. r0.67*. 2015. Dostupné na: `https://www.ctan.org/pkg/nicetext`.

MacFarlane, John. *lunamark* [on-line]. *Ver. 0.4.0*. 2012. Dostupné na: `http://jgm.github.io/lunamark`.

MacFarlane, John. *Pandoc: A universal document converter* [on-line]. *Ver. 1.17.2*. 2016. Dostupné na: `http://pandoc.org`.

Novotný, Vít. *A Markdown Interpreter for TeX* [on-line]. 2016. [cit. 2016-08-17]. Dostupné na: `http://mirrors.ctan.org/macros/generic/markdown/markdown.pdf`.

Olšák, Petr. *Program vlna* [on-line]. *Ver. 1.5*. 2010. Dostupné na: `http://petr.olsak.net/ftp/olsak/vlna/vlna-1.5.tar.gz`.

Wagner, Zdeněk, Olšák, Petr. *encxvlna: Vlna implemented in encTeX* [on-line]. *Ver. 1.1*. 2014. Dostupné na: `https://www.ctan.org/pkg/encxvlna`.

Poore, Geoffrey M. *The minted package* [on-line]. *Ver. 2.4*. 2016. Dostupné na: `http://ctan.org/pkg/minted`.

Wagner, Zdeněk. *XeVlna: Vlna implemented in XeTeX* [on-line]. *Ver. 1.0*. 2013. Dostupné na: `https://www.ctan.org/pkg/xevlna`.

Summary: Rendering Markdown inside TEX Documents

The article describes a new package for plain TEX derivatives that enables the direct inclusion of Markdown-formatted text into TEX documents. The author describes their motivation for the creation of the package and its inner workings. The usage of the package is explained through example.

Keywords: Markdown, lightweight markup, Lua, plain TEX, LATEX, ConTEXt, Pandoc

Vít Novotný, witiko@mail.muni.cz