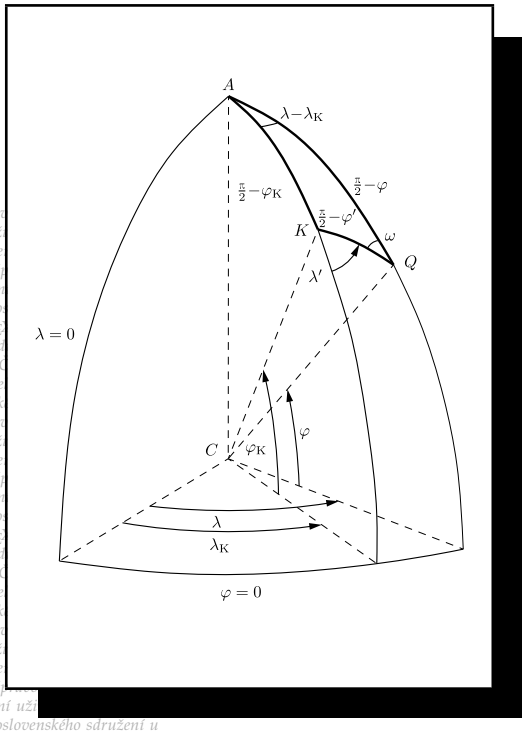
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

3–4

2020

ISSN 1211-6661 Tištěná verze

ISSN 1213-8185 Online verze

Ročník 30

OBSAH

Zpravodaj Československého sdružení uživatelů TEXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Vydaná čísla Zpravodaje v elektronické podobě (PDF) jsou bezodkladně veřejně vystavena na webové adrese `https://www.cstug.cz/`.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`), na e-mailovou adresu `bulletin@cstug.cz`. Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí TEX Live).

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

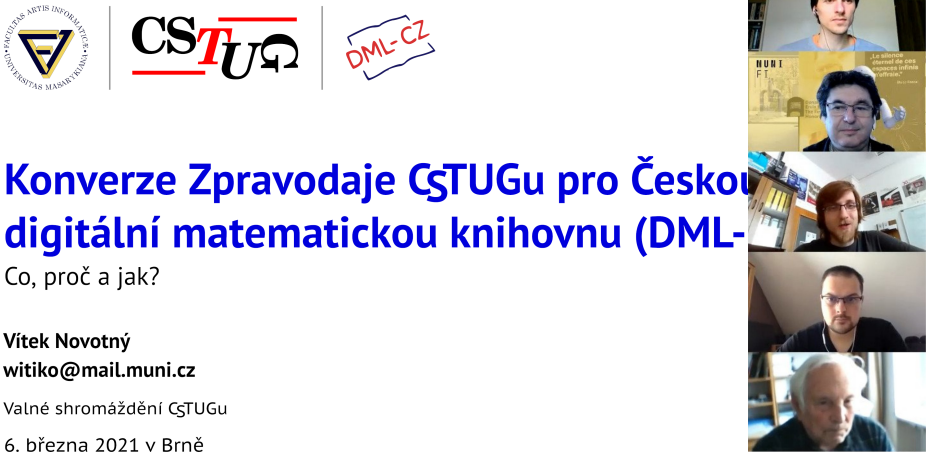
Úvodník

Petr Sojka

Milé čtenářky a čtenáři, TEXistky a TEXisté,

i v dnešní svízelné době se lze radovat z drobností, jakou je toto druhé dvojčíslo Zpravodaje roku 2020. Vzhledem k epidemickým opatřením jsme se nemohli v roce 2020 sejít na tradiční předvánoční přednášce a na valném shromáždění. Výbor na vzniklou situaci reagoval návrhem změn stanov, které v budoucnu umožní korektně realizovat valné shromáždění i v online nebo hybridní podobě. Podle starých stanov svolané valné shromáždění již navrženou změnu schválilo. Volební valné shromáždění na podzim 2021 tak bude uspořádáno již podle nové verze stanov, kterou najdete na webu sdružení. Uvítáme nominace a kandidaturu zejména nových agilních členů do výboru!

Valnému shromáždění předcházela přednáška Vítka Novotného o projektu digitalizace Zpravodaje. V České digitální matematické knihovně (DML-CZ) již Vítkovým přispěním máme první ročník Zpravodaje a zbývajících 30 ročníků bude brzy následovat. Online podoba přednášky zajistila vyšší účast než při kontaktní podobě v předchozích letech. Své články v prvním ročníku Zpravodaje tak mohl vidět i zakládající předseda CSTUGu:



Obrázek 1: Online přednáška o digitalizaci Zpravodaje do DML-CZ s bohatou účastí členů včetně zakládajícího předsedy sdružení Jiřího Veselého (vpravo dole)

DOI: 10.5300/2020-3-4/105

Obrázek 2: Zpravodaj ČSTUGu v digitální matematické knihovně na `dml.cz`

Bohatství článků Zpravodaje tak bude dostupné dalším generacím TEXových nadšenců. Snad se i zvýší motivace uživatelů v něm publikovat.

Na jednoho z prvních předsedů ČSTUGu a jeho využívání TEXu a METAFONTu při sazbě časopisů v Matematickém ústavu AV ČR, pro sazbu učebnic a knih v dalších nakladatelstvích nebo pro organizační zajištění matematických olympiád vzpomíná v prvním článku čísla Pavel Stříž. Karel byl TEXovým perfekcionistou a nadšencem METAFONTu, účastnícím se mnoha TEXových konferencí a projektů.

Jsme stále *Československé* sdružení bez pomlčky, fungujeme společně, naše mateřské jazyky mají velmi podobnou morfologii i pravidla typografie, tak proč nemít i společné dělení slov? Tuto minimalistickou snahu popisuji se synem v druhém článku tohoto čísla. Nové společné vzory mají nejen větší přesnost a generalizační vlastnosti, ale jsou udržitelně generovány z vytvořené otevřené, aktuální a obrovské slovní zásoby obou jazyků zkompilované z veřejně publikovaných českých a slovenských textů na Internetu.

Minimalismus, použití nejnovějších nástrojů a efektivita bez kompromisů jsou nejen znaky designu závodních vozů formule 1, ale i principy, na kterých stojí návrh formátu OpTEX Petra Olšáka. V článku si přečtete výhody, které makrobalík skýtá těm uživatelům TEXu, kteří ho znají pod kapotou do posledního šroubku, a mohou tak OpTEX s výhodou využít pro své publikační projekty místo pohodlného sedanu LATEXu či ConTEXtu. Znáte-li TEX jako Petr, přečtěte si jeho článek: můžete se těšit z minimalismu a rychlosti jeho nového závodního stroje.

Obrázek 3: Karel Horák (vlevo), spolu s dalšími třemi reprezentanty CSTUGu na konferenci BachoTEX 2017 Fotografie: Frans Goddijn

Dobu mezi koronavirovými vlnami využili uživatelé makrobalíku ConTEXt a sešli se v Sibřině u Prahy. O akci referuje včetně fotoreportáže Jano Kula.

Dobré a špatné příklady makrodefinic jazyka METAPOSTu prezentuje ve svém článku Taco Hoekwater.

Číslo tradičně uzavírá další přeložený díl seriálu funkčních ukázek sázecích technik od Petera Wilsona.

Doufám, že karanténa některým z vás umožní se soustředit na napsání článku o svých zkušenostech s TEXem a příští číslo, již v DML-CZ, bude ještě obsáhlejší než toto a my budeme radostnější nejen z porážky viru, ale i ze sdílení svých typografických zkušeností a kvalitní typografie systémem TEX.

Summary: Introductory Word

In addition to the introduction of the content of the Zpravodaj issue, the author refers about the changes of CSTUG bylaws, about the digitization project of Zpravodaj, and about the former CSTUG president Karel Horák.

Masarykova univerzita, Fakulta informatiky, Botanická 68a, 602 00 Brno
sojka@fi.muni.cz

Sbohem, drahý Karle, sbohem!

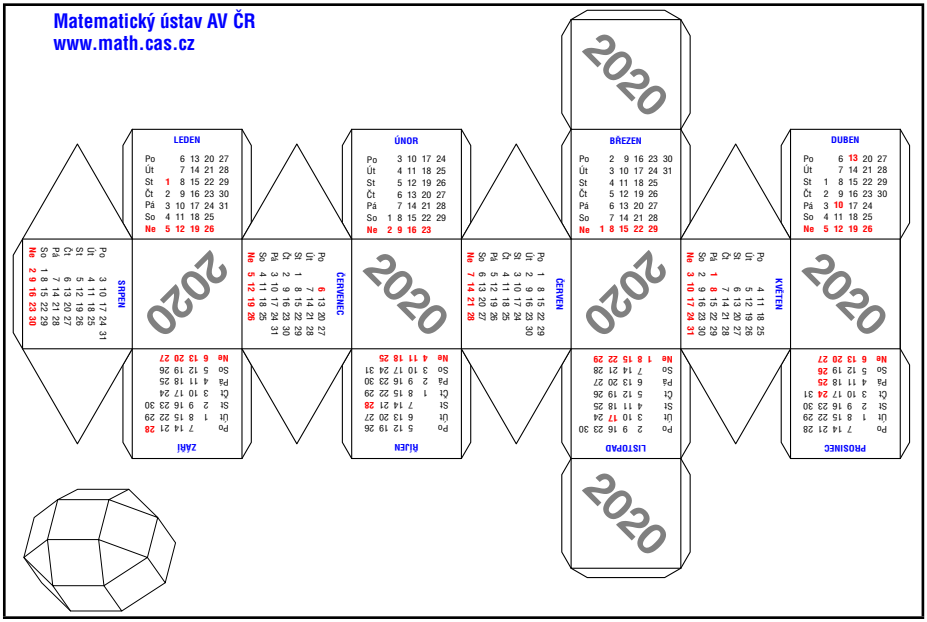
Pavel Stříž

Karel Horák se s námi rozloučil 22. srpna 2020 v jeho raných 66 letech.

Setkávání se s panem doktorem

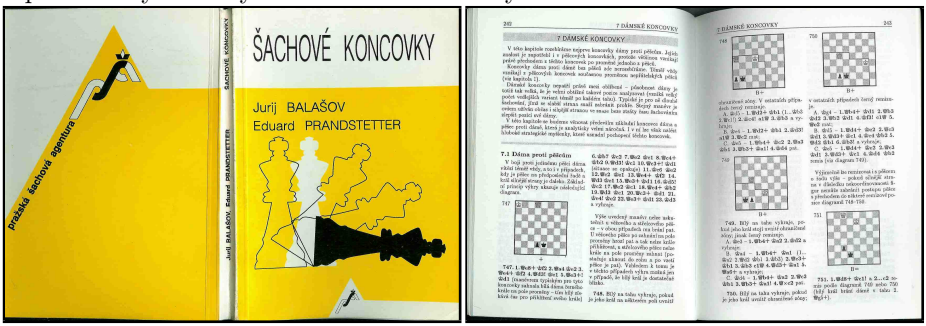
Pravděpodobně každý z vás má v okolí osobu, u které si nevzpomene, kdy a kde se s ní poprvé setkal, ať už osobně či zprostředkovaně. To je můj případ u Karla Horáka, mj. spolužáka Jaromíra Antocha. Ať usilovně přemýšlím, nevím, jestli jsem jeho jméno zahlédl poprvé u některé z tiráží sbírek úloh, nebo jako jednoho z úspěšných soutěžících či členů komise u matematických olympiád, nebo jako jednoho z bývalých předsedů Čs. sdružení uživatelů TEXu, či jako jednoho z účastníků BachoTEXů, EuroTEXů či ConTEXt mítinků.

Řada TEXistů se s jeho prací poprvé setkala prostřednictvím jeho kalendářů. Zde je jedna ukázka. Je to uděláno na koleně METAPOSTem. Začal s nimi na přelomu tisíciletí (tipuji od padesátin) a skončil u 17 stran. Lze změnit rok a vše se přepočítá a překreslí. Napsal o nich příspěvek *Geometric Diversions with TEX, METAFONT and METAPOST* do časopisu TUGBoat, Vol. 24, No. 3, 449–452, 2003, `https://www.tug.org/TUGboat/tb24-3/horak.pdf`.



DOI: 10.5300/2020-3-4/108

Podobně výrazně se zapsal do sazby šachu, kde vytvořil potřebná makra pro Jaroslava Poláška, a bylo možné se proklikávat z PostScriptu do zdrojového souboru, což nyní umí u PDF například TEXworks. Ukázku zmínil v článku *Do šachu s TEXem!* ve Zpravodaji Československého sdružení uživatelů TEXu, Vol. 1, No. 3, 25–26, 1991, `https://www.cstug.cz/bulletin/pdf/bul913.pdf`. Přikládám obálku a ukázku sazby z knihy Jurij Balašov, Eduard Prandstetter: *Šachové koncovky,* Pražská šachová agentura, 1991, neb Karel vedle plakátů, diplomů a výsledkových listin tvořil i obálky.



S jistotou mohu říci, že Karel nevynechal jedinou konferenci TEXperience, kterou jsem organizoval za svého působení ve Zlíně. Při každém setkání se mě ptal, jestli se neplánuje další ročník. Podobně mě Karel popohánět, jestli jsem už nepokročil se sazbou knihy českých erbů měst, o které jsem mu vyprávěl, jak ji plánuji vysázet a jaká data už mám. Těšil se na ni hodně.

Díky mé zálibě v kresbě a překreslování grafů a obrázků mi Karel nabídl práci na Matematickém ústavu AV ČR na pozici technického sazeče. Jako jeho učeň jsem nahlédl víc do PlainTEXu, byť jsem si u úprav šablon spíš rval vlasy. Karel obdivoval práci Donalda E. Knutha, Petra Olšáka, Zdeňka Wagnera, Karla Píšky, Jana Kuly, Honzy Šustka, holandských a polských TEXistů. Jeho záběru u PostScriptu se málokdo vyrovnal. To bylo Karlovo! Jednou mi hrdě prozradil, že přímo ve velké tiskárně sedl ke stroji a v textovém editoru zasáhl do tiskového podkladu, aby se mohlo začít vyrábět. Jednalo se o změnu barevného prostoru v postscriptovém souboru.

A znovu jsem díky Karlovi nahlédl víc do METAPOSTu. Karel byl vždy trochu smutný, když jsem ho přemlouval k LATEXu či Ti*k*Zu. Uznal, že pro méně zkušené uživatele TEXu je to lepší, ale že je potřeba stejně jít do jádra. Co mě však potěšilo, že z učně jsem se stal mistrem pro učně Karla u Lua(TEXu). Osobně i e-mailově jsme řešili různé vychytávky a úpravy textů pomocí Lua. Ať už se jednalo o zásahy do textových či datových souborů.

Jednu ze vzpomínek mám, když jsme Karla ztratili během výletu na konferenci TEXperience. Karel nikde a přitom všichni ostatní už byli z výletu dávno zpátky. Zničehonic mi máma podává mobil, že mám hovor. A hle, Karel, že si odbočil na

prohlídku zříceniny, že je tma, a že neví, jak se dostat zpět na rekreační středisko Jestřabí na Rusavě.

— Pane Horáku, vidíte někde nějaká světla?

— Vidím.

— Tak jděte za tím světlem, buď je to naše chata či začátek vesnice a z tama my vás už vyzvedneme autem. Volejte.

— Dobře, já to tedy zkusím.

Mám ještě jednu vzpomínku, se kterou se rád podělím. Karel byl kavalír, džentlmen a věděl vždy, kdy už stačí. Na mítinku o ConTEXtu na Mlýně Brejlov u Týnce nad Sázavou jsem Karlovi nabízel ořechovku od táty.

— Mám tu vzorek z roku 2008. Dáte si?

— Ochutnám.

— Mám tu ještě vzorek z roku 2007.

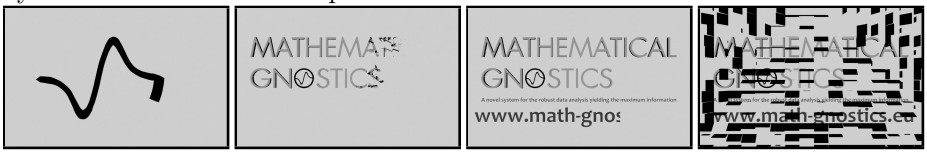
— Neměl bych, ale to ještě ochutnám.

— Nedáte si ještě? Mám tu ještě vzorky z let 2005 a 2006. Táta každý rok trochu experimentoval.

— Ne, ne, opravdu děkuji, ale mně už stačí. Mám akorát. Ale vyřiďte tatínkovi, že je výborná, že se mu podařila.

Trochu mě mrzí, že Karel nedorazil ani jednou do Žiliny na konferenci OSSConf, kde vznikla za víc jak desetiletí zajímavá TEXová sekce s příjemným posluchačstvem. Pro mne to bylo volné pokračování české konference TEXperience. TEXperience byla pro kolegy ze Slovenska vždy daleko. OSSConf má tu výhodu, že člověk může nahlédnout do jiných sekcí: vývoje opensource software, OSS ve vzdělávání, opendatové, openhardwarové, open GISové, ale i do 3D tisku. Říkával, že se obává, že by lidé nerozuměli jemu a on jim. Byl by příjemně překvapený.

Předposlední e-mail od Karla mi dokázal, že je to bojovník. Podrobně mi popsal, co a kde mu z těla doktoři vedou, že je upoután na lůžko doma a čeká mezi chemoterapiemi. Psal, že má prostor díky koronní ráně dokončit své projekty, bez ohledu na to, jak léčba zhoubného nádoru na slinivce, který zablokoval žlučovod, dopadne. Psal jsem mu, že držím palce, a můj poslední e-mail byl náhled na animaci pro Zdeňka Wagnera o matematické gnostice. To by pro METAPOST byla trochu síla. Přikládám pár vzorků z animace.



Odepsal mi: „Pavliku, Tys hracicka :-) Zdravim, Karel“. Tak se se mnou virtuálně rozloučil, to byl od něj poslední e-mail, náhodou vyšel na den mých narozenin.

Karel mi bude chybět, s ním odchází velký kus TEXového umění. Na rozdíl od Knutha, který si od komunity drží odstup, Karel vždy poradil a pomáhal.

Rozloučím se s ním náhledy z jeho tvorby a překreslování, které jsem rozdělil do několika bloků dle místa jeho působení. Je to střípek z jeho obří práce da Vinciho záběru. Vždy si rád vzpomenu na pana doktora z poslední TEXperience, jak nás s křídou v ruce u černé školní tabule, o jejíž instalaci nás poprosil TEXpert Petr Olšák, léčí ze záludností TEXového světa.

Nakladatelství Prometheus a JČMF

`http://prometheus-nakl.cz/` a `https://www.jcmf.cz/`

Karel sázel nejrůznější knihy, učebnice[1] a sbírky úloh. Snažil se vše sázet jen v PlainTEXu a ve svých makrech. LATEX neměl rád a do ConTEXtu se nestihl zamilovat. Na obrázky používal METAPOST s řadou udělátek, pomůcek a konverzních skriptů.

Mne konkrétně zaujal odstavec o proběhlé matematické olympiádě v roce 2006 v *Rozhledech matematicko-fyzikálních*, Vol. 81, No. 2, 44–47. Jedná se o zajímavý obrázek, kdybychom jej chtěli překreslit, umístěný v pozadí odstavce zprávy. Zároveň je to náhled na styl Karlova psaní.

Přípravu a zdárný průběh celé akce zajišťovali organizátoři z řad členů *Mexické matematické společnosti* za podpory mexického ministerstva školství, vlády státu Yucatán, tamních univerzit a desítek sponzorů. Nashromážděné finanční prostředky umožnily ubytovat všechny soutěžící, vedoucí družstev i členy výborů a hodnotících komisí v areálu luxusních hotelů nedaleko centra yucatánské metropole, založené španělskými dobyvateli roku 1542 na místě mayského města *Tihó*. Mexičtí hostitelé připravili výborné podmínky pro vlastní soutěž i zajímavý doprovodný program, jehož vrcholem byl celodenní výlet ke zříceninám mayského města *Chichén Itzá*. Závěr olympiády mírně narušil příchod hurikánu *Emily*, který však nakonec Méridu minul zhruba o 80 km a v samotném městě se projevil jen silnějším větrem.

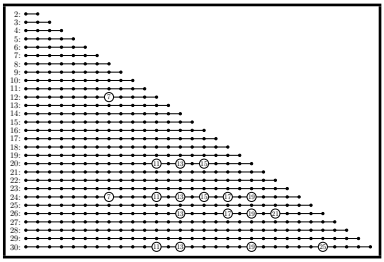
Vedoucím českého družstva byl RNDr. *Karel Horák*, CSc., z Matematického ústavu Akademie věd v Praze. Soutěžní družstvo, které doprovázel pedagogický vedoucí doc. RNDr. *Jaromír Šimša*, CSc., z Přírodovědecké fakulty Masarykovy univerzity v Brně, bylo jmenováno na základě výsledků ústředního kola 54. ročníku MO v Benešově a následného týdenního soustředění v Bílovci. Tvořili je *Jaroslav Hančl* z 3. ročníku

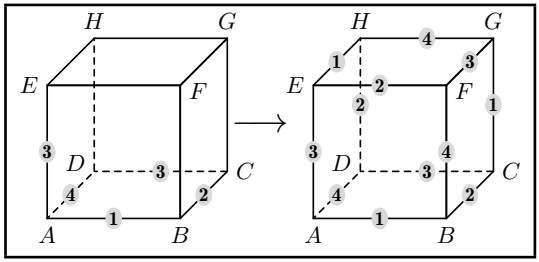
[1] Zmiňme zde alespoň svazek 1 Edice Překlady vysokoškolských učebnic: D. Halliday, R. Resnick, J. Walker – Fyzika. Vysokoškolská učebnice obecné fyziky. ISBN 81-7196-214-7 (Prometheus), 2000 (Typografie a sazba programem TEX RNDr. Karel Horák, CSc.) jako příklad kvalitní a krásné sazby programem TEX. Poznámka redakce.

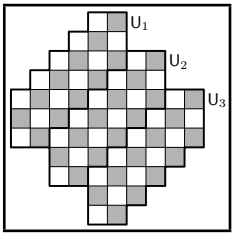
Matematická olympiáda MO

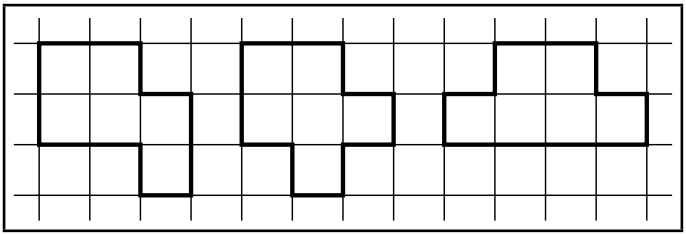
http://www.matematickaolympiada.cz/

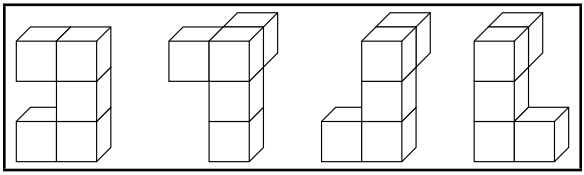
Karel patřil k národním vítězům v letech 1971–1973, později k aktivním organizátorům a tvůrcům úloh. Zde je několik ukázek obrázků z ročníků 65 až 69. Pérovky (černobílé obrázky, ty bez barev a šedé) měl nejraději.

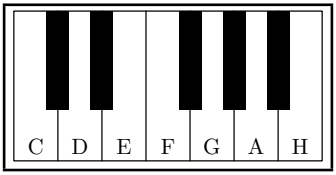


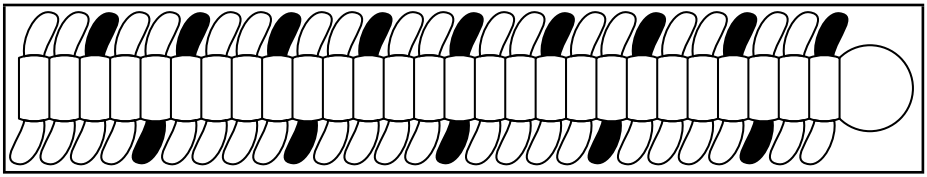


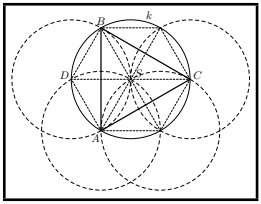


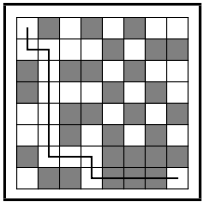


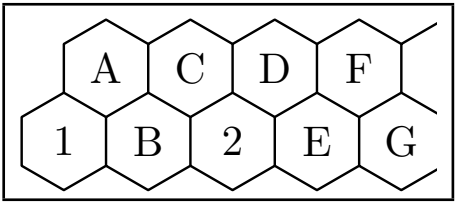


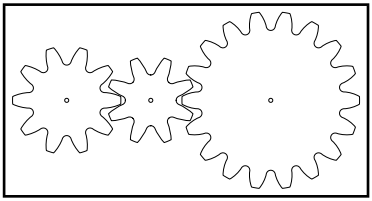
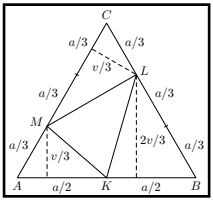


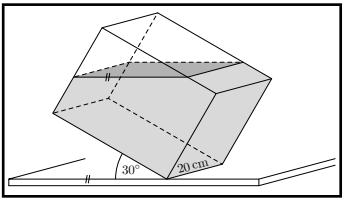




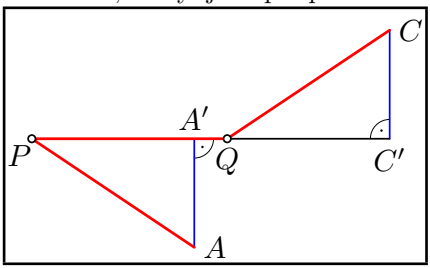


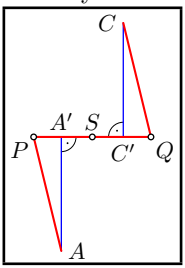


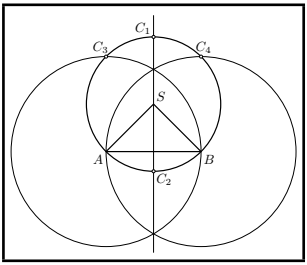





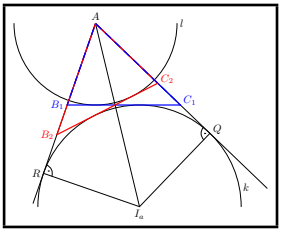
Vždy se pečlivě snažil průniky křivek a důležité body zvýraznit prázdným kroužkem, to byl jeho podpis. Značka kvality.

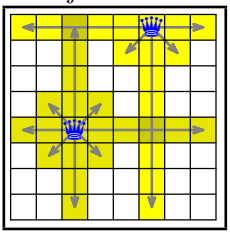
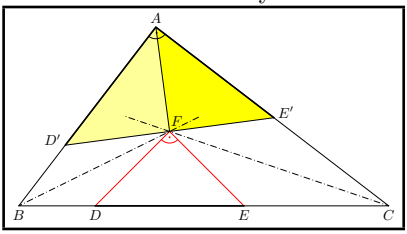


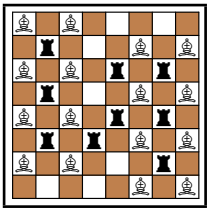
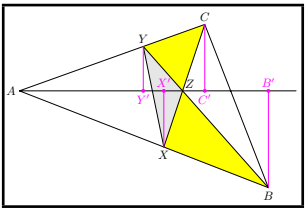


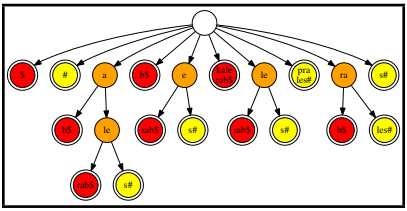


Užití barev se neštítil. Zde je několik dalších obrázků v barvě z týchž ročníků.





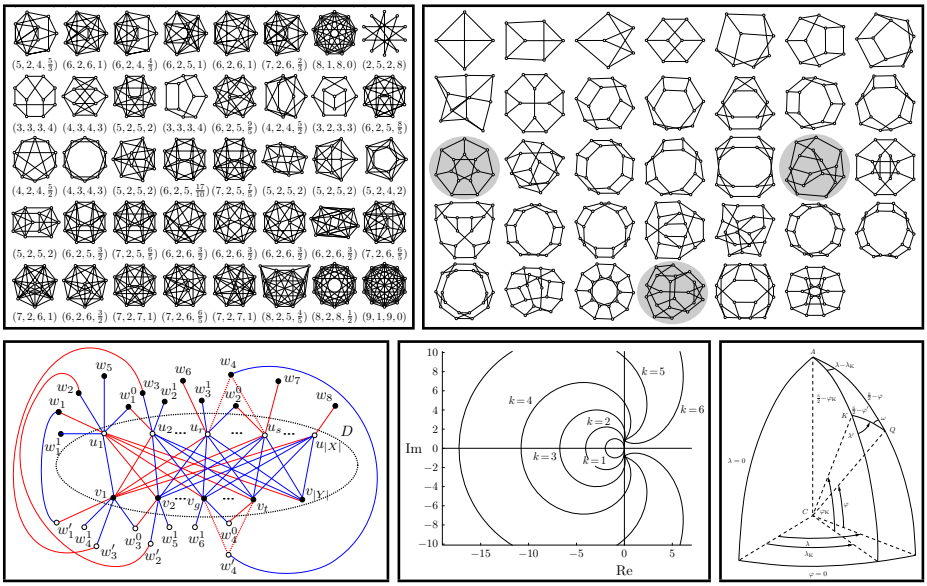
Matematický ústav AV ČR

`http://www.math.cas.cz/`

Karlovou domovinou byl Matematický ústav AV ČR. I zde šířil povědomí o TEXu, METAPOSTu, a jak mi potvrdili jeho kolegové, vždy každému rád poradil. Z první ruky mohu potvrdit, že častokrát i ve dvě ráno.

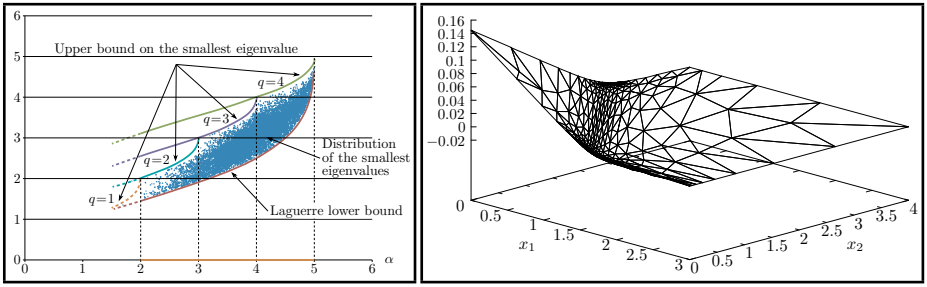
Nejtěžší sazební úkoly vyplynuly z časopisů ústavu: Czechoslovak Mathematical Journal (dále CMJ, založen 1951, `http://cmj.math.cas.cz/`), Applications

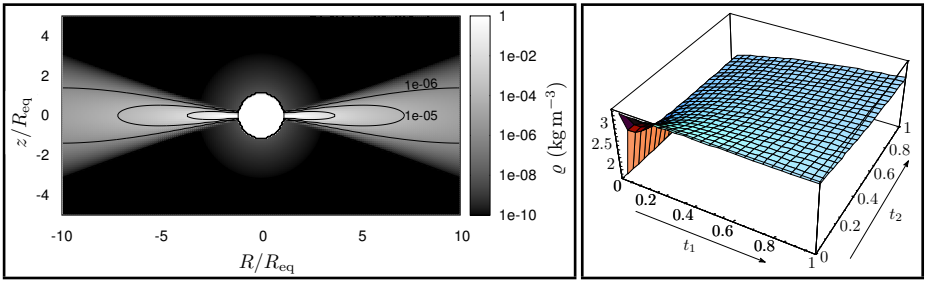
of Mathematics (AM, založen 1956, `http://am.math.cas.cz/`) a Mathematica Bohemica (MB, založen 1872, `http://mb.math.cas.cz/`).



Z článků J. C. Hurajová, T. Madaras: More on betweenness-uniform graphs, CMJ, 68(2), 293–306, 2018 (dva obrázky); J. Yun Yue, S. Meiqin Wei, T. Yan Zhao: Proper connection number of bipartite graphs, CMJ, 68(2), 307–322, 2018; M. Vlasák: Time discretizations for evolution problems, AM, 62(2), 135–169, 2017 a T. Bayer, M. Kočandrlová: Reconstruction of map projection, its inverse and re-projection, AM, 63(4), 455–481, 2018.

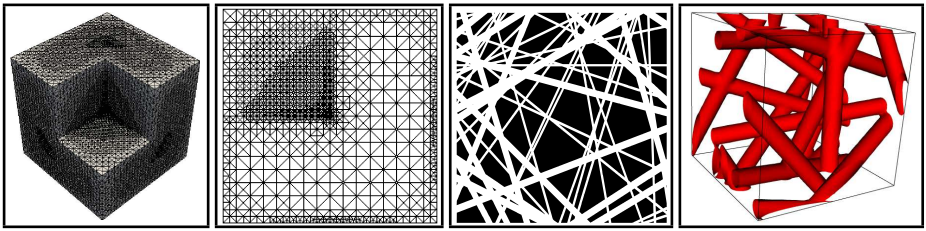
U některých grafů nerad uznal, že jsou příliš těžké na překreslení bez vstupních dat, tak se snažil o sazební jednotu popisků v grafu či alespoň o sazbu popisků jednotlivých os.





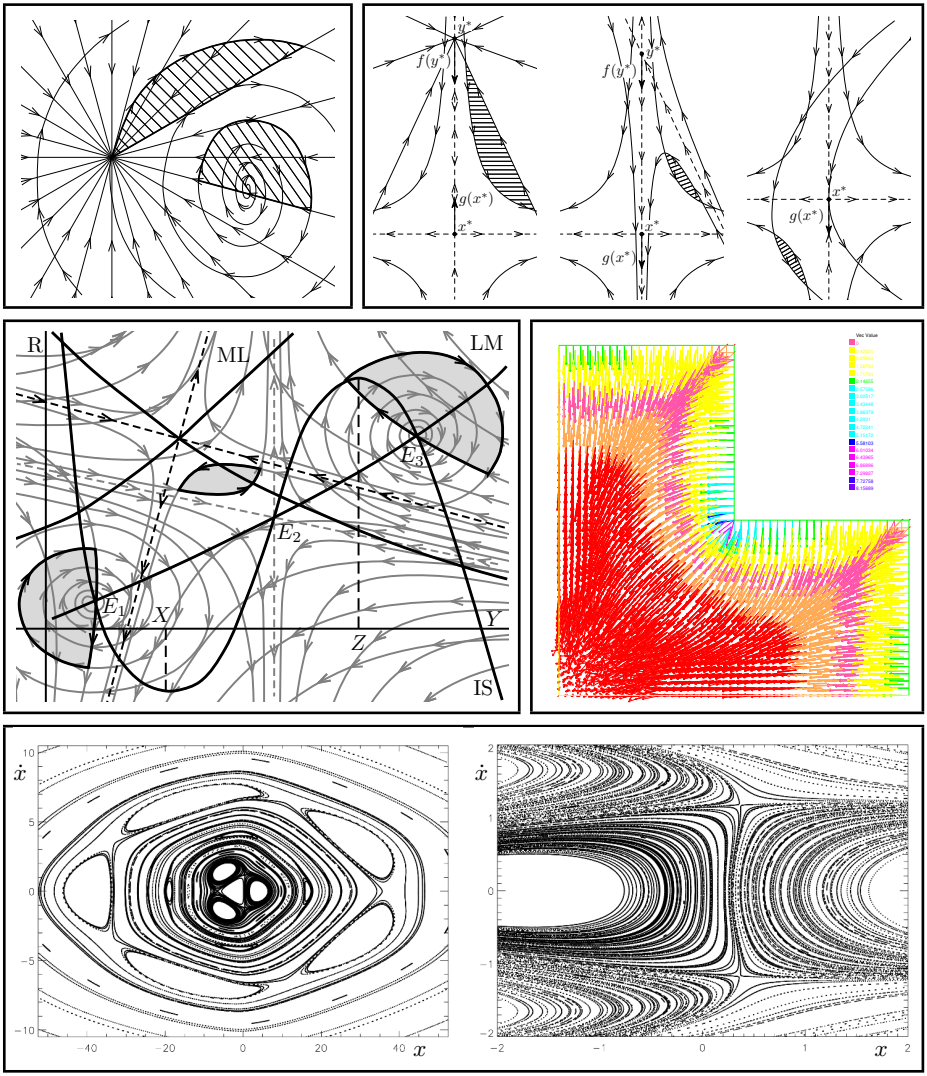
Z článků Y. Yamamoto: On the optimality and sharpness of Laguerre's lower bound on the smallest eigenvalue of a symmetric positive definite matrix, AM, 62(4), 319–331, 2017; J. Hozman, T. Tichý: DG method for numerical pricing of multi-asset Asian options—the case of options with floating strike, AM, 62(2), 171–195, 2017; P. Kurfürst, J. Krtička: Time-dependent numerical modeling of large-scale astrophysical processes: from relatively smooth flows to explosive events with extremely large discontinuities and high Mach numbers, AM, 62(6), 633–659, 2017 a A. Ghost, C. Kundu: On generalized conditional cumulative past inaccuracy measure, AM, 63(2), 167–193, 2018.

Občas se objevily podklady, kde nebylo možné či praktické zasáhnout, tak říkal, že to se nedá nic dělat. Zmínil, že autory oslovoval o lepší verzi, ale kde to šlo, autory dál nezatěžoval. Zde je ukázka ponechaného rastrového obrázku, vektorový obrázek o mnoha linkách by byl neúnosně velký (obrázky vlevo). Podobně to platívá u fotek, skenů a modelů se světly a stíny (obrázky vpravo).



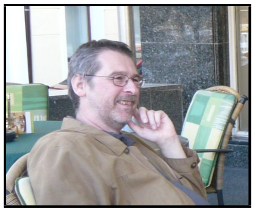
Po dvou obrázcích z článků Fei Xu, Hehu Xie: A full multigrid method for semilinear elliptic equation, AM, 62(3), 225–241, 2017 a D. Jeulin: Iterated Boolean random varieties and application to fracture statistics models, AM, 61(4), 363–386, 2016.

Podklady nám chodily v různém tvaru – od ručních náčrtků přes kresby vytvořené v TEXu až po nejrůznější výstupy z výpočetních a simulačních nástrojů. Jednalo se o celou šíři oboru. Zde je několik extra „vypečených“ ukázek pro potěšení oka čtenáře.



Z článků B. Volná: Chaotic behaviour of continuous dynamical system generated by Euler equation branching and its application in macroeconomic equilibrium model, MB, 140(4), 437–445, 2015 (tři obrázky); X. Hu, P. Huang, X. Feng: A new mixed finite element method based on the Crank-Nicolson scheme for Burgers' equation, AM, 61(1), 27–45, 2016 a J. Málek, K. R. Rajagopal, P. Suková: Response of a class of mechanical oscillators described by a novel system of differential-algebraic equations, AM, 61(1), 79–102, 2016.

Byl jednou jeden pan doktor

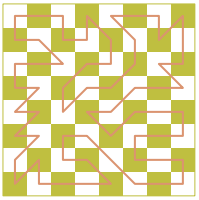


Během naší komunikace jsme řešili nejrůznější zajímavosti, a když Karel něco nevěděl, to už bylo co říct. Jeden z problémů, který mi zmínil, bylo přesázení zdrojových kódů zaniklého ruského časopisu Kvant, `http://www.kvant.info/zkm_main.htm` (rusky Журнал Квант). Byl tam problém s chybějícími písmy, zjištěním kódové stránky, i s tím, jak získat vektorovou podobu z rastrových obrázků. Shrnul jsem své postřehy na stránce `https://tex.stackexchange.com/questions/181153`. Jeho idea byla si ročníky 1970 až 2009 nejprve přesázet do hezké knihy, vytisknout a pak svázat. Karel dobře věděl, kde hledat inspiraci a jak inspirovat ostatní.

Nyní se píše rok 2020 a PDF získaná TEXem stále neobsahují původní zdrojové kódy. Tohoto zlepšováku se Karel nedočkal.

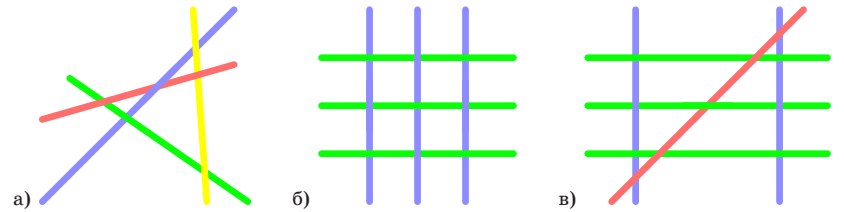
Rozloučím se s vámi, drazí pozůstalí, ukázkami ze stran 31, 143 a 263 souboru `http://www.kvant.info/zkm_tex/zkm_main.pdf` Sbírky matematických úloh časopisu Kvant. Chybí nám všem. Je to rána! Byl to borec.

220. Король обошёл шахматную доску, побывав на каждом поле ровно один раз и вернувшись последним ходом на исходное поле. (Король ходит по обычным правилам: за один ход он может перейти по горизонтали, вертикали или диагонали на любое соседнее поле.) Когда нарисовали его путь, последовательно соединив центры полей, которые он проходил, получилась замкнутая ломаная без самопересечений. Какую наименьшую и какую наибольшую длину может она иметь? (Сторона клетки равна единице.) *А.В.Климов.*



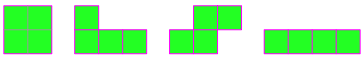
VII Всесоюзная олимпиада. *Решение — в №5–1974. Статья И.Ф.Акулича «Прогулки короля» третьего номера 2000 года. Статья Н.Б.Васильева «Вокруг формулы Пика» двенадцатого номера 1974 года*

1085* Несколько попарно скрещивающихся прямых, расположенных в пространстве, спроецировали на горизонтальную плоскость. Их проекции изображены так, чтобы в точках пересечения было видно, какая точка расположена выше, а какая ниже:



Могла ли получиться проекция, изображённая на рисунке? *С.Л.Табачников.* *Решение — в №5–1988. Комментарий — в статье О.Я.Виро и Ю.В.Дроботухиной «Сплетения скрещивающихся прямых» третьего номера 1988 года, в статье С.Л.Табачникова «Линейные неравенства и задача М1085» шестого номера 1989 года и в статье А. Скопенкова «Ещё раз о сплетающихся прямых» одиннадцатого номера 1989 года*

2128. Вася отметил 10 клеток в клетчатой таблице размером 10×10. Всегда ли Петя может вырезать из этой таблицы по линиям сетки 19 фигурок, каждая из которых — одного из четырёх видов, показанных на рисунке, таким образом, чтобы фигурки не содержали ни одной отмеченной клетки? *И.Богданов и О.Подлипский.* *Решение — в №5–2009*



Towards New Czechoslovak Hyphenation Patterns

Petr Sojka, Ondřej Sojka

Space- and time-effective segmentation and hyphenation of natural languages stay at the core of every document preparation system, web browser, or mobile rendering system. Recently, the unreasonable effectiveness of pattern generation has been shown – it is possible to use hyphenation patterns to solve the dictionary problem for a single language without compromise. In this article, we will show how we applied the marvelous effectiveness of `patgen` for the generation of the new Czechoslovak hyphenation patterns that cover two languages. We show that the development of more universal hyphenation patterns is feasible, allows for significant quality improvements and space savings. We evaluate the new approach and the new Czechoslovak hyphenation patterns.

Keywords: hyphenation, hyphenation patterns, `patgen`, syllabification, syllabic hyphenation, Czech, Slovak, Czechoslovak patterns

"Any respectable word processing package includes a hyphenation facility. Those based on an algorithm, also called logic systems, often break words incorrectly."
Major Keary in [1]

Introduction

Hyphenation is at the core of every document preparation system, be it TEX or any modern web browser. It has been shown [2] that data-driven approaches to hyphenation and syllabification algorithms outperform rule-based ones, reaching accuracy around 95% per a single language. Bartlett et al. [3] developed a machine learning approach for automatic syllabification, motivated by the needs of letter-to-phoneme conversion. Trogkanis et al. [4] used conditional random fields for word hyphenation, and compared the accuracy and other metrics with the original technique of Liang [5]. Their results abstracted from heuristics to optimize generated patterns by `patgen`, diminishing achievable performance.

There are about 5,000 languages supported by Unicode Consortium that are still in use today. In a digital typographic system that supports Unicode and its languages in full, there should be support for algorithms, rules, or language hyphenation patterns. Recently, there were attempts to tackle the word segmentation problem in different languages by Shao et al. [6]. The algorithm is error-prone, but it was developed primarily for speech recognition and language representation tasks, where a small number of errors is tolerated. On the contrary, in a typesetting system like TEX, errors in hyphenation are not tolerated at all—all exceptions have to be covered by the algorithm.

 doi: 10.5300/2020-3-4/118

Current typesetting support in the TEX Live distribution contains [7] hyphenation patterns for about 80 different languages. All these patterns have to be loaded into TEX's memory at the start of every compilation, which slows down compilation.

There are essentially two quite different approaches to hyphenation:

etymology-based The rule is to cut a word on the border of a compound word or the border of the stem and an affix, prefix or negation. A typical example are the British hyphenation rules by the Oxford University Press [8].

phonology-based Hyphenation based on the pronunciation of syllables allows reading text with hyphenated lines almost as if the hyphenation were not there. This pragmatic approach is preferred by the American publishers [9] and the Chicago Manual of Style [10].

In this paper, we evaluate the feasibility of the development of universal phonology-based (syllabic) hyphenation patterns. As a case study, we describe the development of Czechoslovak hyphenation patterns from word lists of Czech [11, 12, 13] and Slovak [14]. We document our reproducible workflow and resources in a public repository.

"Hyphenation does not lend itself to any set of unequivocal rules. Indeed, the many exceptions and disagreements suggest it is all something dreamed up at an anarchists' convention." Major Keary in [1]

Methods

The core idea is to develop common hyphenation patterns for phonology-based languages. In the case that these languages share pronunciation rules, homographs from different languages typically do not cause problems, as they are hyphenated the same. The very rare cases, where hyphenation is dictated by the seam of compound word contrary to phonology (`roz-um` vs. `ro-zum`), could be simply solved by not allowing the hyphenation of this particular word around this particular seam.

Recently, it was shown that the approach to generate hyphenation patterns from word list by program `patgen` is unreasonably effective [15]. One can set the parameters of the generation process so that the patterns cover 100% of hyphenation points, and the size of the patterns remains reasonably small. For the Czech language, hyphenation points from 3,000,000 hyphenated words are squeezed into 30,000 bytes of patterns, as stored in the compressed trie data structure. That means achieving a compression ratio of several orders of magnitude with 100% coverage and nearly zero errors [15]. For a similar language such as Slovak, the pronunciation is very similar, syllable-forming principles are the same, and also compositional rules and prefixes are pretty close, if not identical.

We have decided to verify the approach by developing hyphenation patterns that will hyphenate both Czech and Slovak words without errors, with only a few

missed hyphens. That means that *only* words like `oblít` will not be hyphenated, because the typesetting system currently cannot decide in which meaning the word is used: `o-blít` or `ob-lít`.

To generate these hyphenation patterns, we needed to create lists of correctly hyphenated Czech and Slovak words.

Data Preparation

For our work, word lists with frequencies for Czech and Slovak were donated by Lexical Computing CZ from the TenTen family of corpora [16, 17].

The Czech word list was cleaned up and extended as described by Sojka et Sojka [15, 18], using the Czech morphological analyzer `majka`. Only words that occurred more than ten times were used in further processing. The final word list `cs-all-cstenten.wls` contained 606,494 words.

For Slovak, we obtained 1,048,860 Slovak words with frequency higher than ten from 2011 SkTenTen corpora [16]. We only used words with a frequency higher than thirty that comprised only of ISO Latin 2 characters, obtaining file `sktenten.wls` with 544,609 words.

By joining both language files, we got 967,058 Czech and Slovak words in `cssk-all-join.wls`, of which 106,016 were contained in the intersection of both word lists: `cssk-all-intersect.wls`.

Pattern Development

The workflow of the Czechoslovak pattern development is illustrated in Figure 1 on the facing page. We have used recent accurate Czech patterns [15] for the hyphenation of the joint Czech and Slovak word list. We had to manually fix incorrect hyphenation, typically near the prefix and stem of words when phoneme-based hyphenation point was one character away from the seam of the prefix or compound word: `neja-traktivnější`, `neja-teističtější`, `neje-kologičtější`.

We have then hyphenated words used in both languages also by current Slovak patterns. There were only a few word hyphenations that needed to be corrected—we created the file `sk-corrections.wlh` that contained the fixed hyphenated words. Finally, we used them as an input to `patgen` with a higher weight during generation of the final Czechoslovak hyphenated patterns.

It must be noted that we did not pursue 100% coverage at all costs, because the source data is noisy and we do not want the patterns to learn all the typos and inconsistencies. We expand on this in the Jupyter notebook of Sojka et Sojka [21]. Gentle reader may also find the scripts used there.

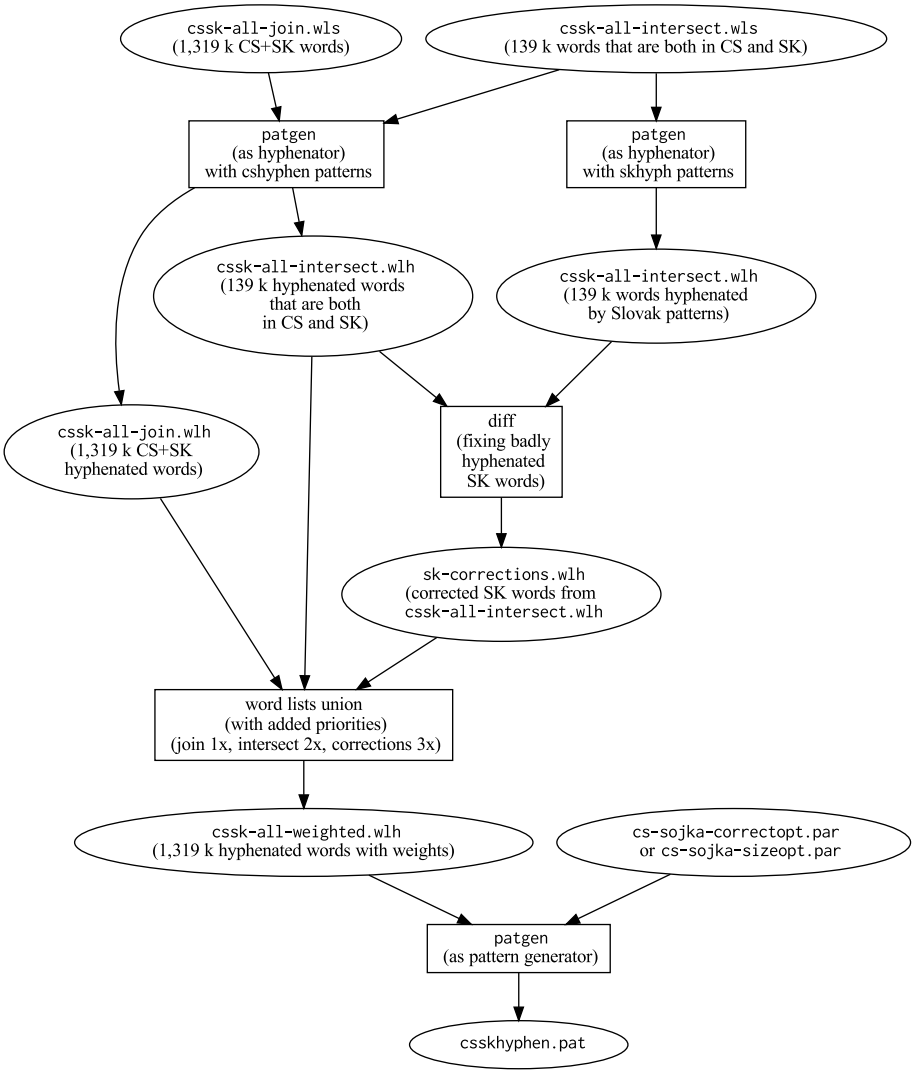


Figure 1: The development process of the new Czechoslovak patterns: Bootstrapping with Czech patterns, checking and fixing with a higher weight Slovak words that are common with Czech ones.

Table 1: Statistics from the generation of Czechoslovak hyphenation patterns with custom parameters.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	830	2,819,833	470,649	35,908	1 3	1 3 12
2	1,590	2,748,581	3,207	107,160	2 4	1 1 5
3	2,766	2,852,334	12,197	3,407	3 6	1 2 4
4	1,285	2,851,931	986	3,810	3 7	1 4 2

Table 2: Statistics from the generation of Czechoslovak hyphenation patterns with correct optimized parameters.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	2,032	2,800,136	242,962	55,605	1 3	1 5 1
2	2,009	2,791,326	10,343	64,415	1 3	1 5 1
3	3,704	2,855,554	11,970	187	2 6	1 3 1
4	1,206	2,854,794	33	947	2 7	1 3 1

Table 3: Comparison of the efficiency of different approaches to hyphenating Czech and Slovak. Note that the Czechoslovak patterns are comparable in size and quality to single-language ones—there is only a negligible difference compared to e.g. purely Czech patterns.

Word list	Parameters	Good	Bad	Missed	Size	Patterns
Slovak	[19, by hand]	N/A	N/A	N/A	20 kB	2,467
Czech	correctopt [15]	99.76%	2.94%	0.24%	30 kB	5,593
Czech	sizeopt [15]	98.95%	2.80%	1.05%	19 kB	3,816
Slovak	[20, Table 1, `patgen`]	99.94%	0.01%	0.06%	56 kB	2,347
Czechoslovak	sizeopt	99.67%	0.00%	0.33%	40 kB	7,417
Czechoslovak	correctopt	99.99%	0.00%	0.01%	45 kB	8,231
Czechoslovak	custom	99.87%	0.03%	0.13%	32 kB	5,907

Table 4: Results of 10-fold cross-validation with evaluated parameters

Parameters	Good	Bad	Missed
correctopt	99.81%	0.15%	0.04%
custom	99.64%	0.22%	0.14%
sizeopt	99.41%	0.18%	0.40%

Evaluation

The evaluation of the quality of developed patterns could be done as an evaluation of *coverage* of hyphenation points in the training wordlist—how many hyphenation points used in training were correctly predicted by the patterns—and of *generalization* properties—how the patterns behave on unseen data, on the words not available in the data used during `patgen` training.

Both coverage and generalization could be viewed as a *classification* task, i.e. how the patterns classify hyphenation points in the training and testing wordlists, respectively.

Classification

For evaluation of classification, there are four numbers in the contingency matrix that compare hyphenation point prediction by patterns with the ground truth expressed in the wordlist: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). In tables 1– 3 on the preceding page, we report: **Good** sum or percentage of found hyphenation points as a sum of TP and TN, **Bad** sum or percentage of badly suggested hyphenation points (FP, type 1 error), **Missed** sum or percentage of missed hyphenation points (FN, type 2 error).

Type 1 errors are clearly more severe than type 2 errors in our hyphenation points setup. Nonzero **Bad** results do not necessarily mean that the patterns performed badly, the opposite is often the case—patterns have found a rule that is not obeyed in the ground truth wordlist. In other words, they found an inconsistency that needs to be fixed in the underlying wordlist, rather than a valid exception. This practice of manually inspecting bad hyphenation points has been used during the development of the wordlist.

Generalization

To assess the generalization properties, we used 10-fold cross-validation, leaving one tenth out of the training set to evaluate the effectiveness of the patterns on unseen words. The results are shown in the Table 4 on the facing page. The evaluation metrics slightly differ with different `patgen` parameters, with best results achieved when coverage of the training set is maximized.

The achieved results show that both evaluation metrics are close to perfection, as we are free to either push for perfect coverage, and reach it (lossless compression of wordlist hyphenation points by the developed pattern), or push to maximize generalization qualities, and miss only less than 1% of valid hyphenation points. Doing that for two languages in parallel seems like a good result.

"Esoteric Nonsense? Hyphenation is neither anarchy nor the sole province of pedants and pedagogues.... Used in moderation it can make a printed page more visually pleasing. If used indiscriminately it can have the opposite effect, either putting the reader off or causing unnecessary distraction. If the intended audience is bound to read the work (a user manual, for example) poor hyphenation practice may not matter. If the author wants to attract and hold an audience, then hyphenation needs just as careful attention as any other aspect of presentation." Major Keary in [1]

Conclusion and Future Works

We have shown that the development of common hyphenation patterns for languages with similar pronunciation is feasible. `Patgen` was able to generalize hyphenation rules common for both languages with a negligible increase in the size of the generated patterns.

The resulting Czechoslovak patterns hyphenate Czech and Slovak much better than the former single-language patterns, see a *Jupyter demo notebook* and all **source code** by Sojka et Sojka [21].

We will offer the new patterns for "the Czechoslovak language" to the TEX Live distribution, creating the first language support package to be shared by multiple languages. With this route, the new patterns will appear in most typesetting systems and browsers including OpenOffice and Chrome quickly, as they use the pattern technology and patterns from the TEX community (the `tex-hyphen` repository).

Acknowledgement
We are indebted to Don Knuth for questioning the common properties of Czech and Slovak hyphenation during our presentation of [15] at TUG 2019, which has led us in this research direction.

References

1. KEARY, Major. On Hyphenation – Anarchy of Pedantry. *PC Update, The magazine of the Melbourne PC User Group*. 2005. Available also from: `https://web.archive.org/web/20050310054738/http://www.melbpc.org.au/pcupdate/9100/9112article4.htm`.
2. MARCHAND, Yannick, ADSETT, Connie R. et DAMPER, Robert I. Automatic Syllabification in English: A Comparison of Different Algorithms. *Language and Speech*. 2009, vol. 52, no. 1, pp. 1–27. Available from DOI: `10.1177/0023830908099881`.
3. BARTLETT, Susan, KONDRAK, Grzegorz et CHERRY, Colin. Automatic Syllabification with Structured SVMs for Letter-to-Phoneme Conversion.

In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, 2008, pp. 568–576. Available also from: https://www.aclweb.org/anthology/P08-1065.

4. TROGKANIS, Nikolaos et ELKAN, Charles. Conditional Random Fields for Word Hyphenation. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, 2010, pp. 366–374. Available also from: https://www.aclweb.org/anthology/P10-1038.
5. LIANG, Franklin M. *Word Hy-phen-a-tion by Com-put-er*. 1983. PhD thesis. Stanford University.
6. SHAO, Yan, HARDMEIER, Christian et NIVRE, Joakim. Universal Word Segmentation: Implementation and Interpretation. *Transactions of the Association for Computational Linguistics*. 2018, vol. 6, pp. 421–435. Available from DOI: 10.1162/tacl_a_00033.
7. REUTENAUER, Arthur et MIKLAVEC, Mojca. *TEX hyphenation patterns*. TUG, [n.d.]. Available also from: https://tug.org/tex-hyphen/. Accessed 2019-11-24.
8. ALLEN, R. E. (ed.). *The Oxford Spelling Dictionary*. Oxford University Press, 1990. The Oxford Library of English Usage.
9. GOVE, Philip Babcock et WEBSTER, Merriam. *Webster's Third New International Dictionary of the English language Unabridged*. Springfield, Massachusetts, U.S.A: Merriam-Webster Inc., 2002.
10. ANONYMOUS. *The Chicago Manual of Style*. 17th ed. Chicago: University of Chicago Press, 2017. ISBN 9780226287058.
11. SOJKA, Petr. Notes on Compound Word Hyphenation in TEX. *TUGboat*. 1995, vol. 16, no. 3, 290–297. Available also from: https://tug.org/TUGboat/tb16-3/tb48soj2.pdf.
12. SOJKA, Petr et ŠEVEČEK, Pavel. Hyphenation in TEX—Quo Vadis? *TUGboat*. 1995, vol. 16, no. 3, 280–289. Available also from: https://tug.org/TUGboat/tb16-3/tb48soj1.pdf.
13. SOJKA, Petr. Hyphenation on Demand. *TUGboat*. 1999, vol. 20, no. 3, 241–247. Available also from: https://tug.org/TUGboat/tb20-3/tb64sojka.pdf.
14. SOJKA, Petr. Slovenské vzory dělení: čas pro změnu? (Slovak Hyphenation Patterns: A Time for Change?) *CSTUG Bulletin*. 2004, vol. 14, no. 3–4, 183–189. Available from DOI: 10.5300/2004-3-4/183.
15. SOJKA, Petr et SOJKA, Ondřej. The unreasonable effectiveness of pattern generation. *TUGboat*. 2019, vol. 40, no. 2, pp. 187–193. Available also from: https://tug.org/TUGboat/tb40-2/tb125sojka-patgen.pdf.
16. JAKUBÍČEK, Miloš, KILGARRIFF, Adam, KOVÁŘ, Vojtěch, RYCHLÝ, Pavel et SUCHOMEL, Vít. The TenTen Corpus Family. In: *Proc. of the*

7th International Corpus Linguistics Conference (CL). Lancaster, 2013, pp. 125–127.
17. KILGARRIFF, Adam, RYCHLÝ, Pavel, SMRŽ, Pavel et TUGWELL, David. The Sketch Engine. In: *Proceedings of the Eleventh EURALEX International Congress*. Lorient, France, 2004, pp. 105–116.
18. SOJKA, Petr et SOJKA, Ondřej. The Unreasonable Effectiveness of Pattern Generation. *Zpravodaj ČSTUG*. 2019, vol. 29, no. 1–4, 73–86. Available from DOI: 10.5300/2019-1-4/73.
19. CHLEBÍKOVÁ, Jana. Ako rozdeliť (slovo) Československo (How to hyphenate the word Czechoslovakia). *Zpravodaj ČSTUG*. 1991, vol. 1, no. 4, 10–13. Available from DOI: 10.5300/1991-4/10.
20. SOJKA, Petr. Slovenské vzory dělení: čas pro změnu? In: *Proceedings of SLT 2004, 4th seminar on Linux and TEX*. Znojmo: Konvoj, 2004, 67–72. Available also from: https://fi.muni.cz/usr/sojka/papers/skhyp.pdf.
21. SOJKA, Ondřej et SOJKA, Petr. *cshyphen repository*. [N.d.]. Available also from: https://github.com/tensojka/cshyphen.

Na cestě k novým československým vzorům dělení

Prostorově a časově efektivní segmentace a dělení slov přirozených jazyků zůstává jádrem každého systému pro přípravu dokumentů, webového prohlížeče nebo zlomu dokumentů na mobilních zařízeních. Nedávno jsme ukázali obrovskou účinnost generování vzorů a bylo prokázáno, že je možné použít vzory dělení slov k vyřešení slovníkového problému (automatické segmentace) pro jeden jazyk bez kompromisů (100% pokrytí). V tomto článku ukazujeme, jak jsme použili úžasnou účinnost patgenu pro generování vzorů dělení slov, které pokrývají dva jazyky zároveň, pro nové, společné vzory československého dělení. Ukazujeme, že je možné vyvinout univerzálnější vzory dělení slov, což umožňuje jak kvalitativní zlepšení, tak i úsporu místa oproti předchozí dvojici vzorů pro jednotlivé jazyky. Hodnotíme nový přístup a nové společné československé vzory dělení.

Klíčová slova: patgen, vzory dělení slov, československé dělení, efektivní segmentace, slabičné dělení pro více jazyků

Petr Sojka, ORCID: 0000-0002-5768-4007
Faculty of Informatics, Masaryk University
Brno, Czech Republic and ČSTUG, sojka@fi.muni.cz

Ondřej Sojka, ORCID: 0000-0003-2048-9977
Faculty of Informatics, Masaryk University
Brno, Czech Republic, 454904@mail.muni.cz

OpTEX – pokračování maker OPmac pro LuaTEX

Petr Olšák

Někteří čtenáři možná znají TEXovou sadu maker OPmac pro plainTEX. Ta umožňuje při zachování jednoduchosti a přímočarosti užití maker plainTEXu využít v podstatě všechny pokročilé vlastnosti při sazbě současných dokumentů. Pokračovatelem této koncepce je sada maker OpTEX [1], která společně s LuaTEXem tvoří ucelený nástroj na přípravu dokumentů. Zůstává zachována přímočarost a věci se navíc zjednodušují, protože nemusíme myslet na rozličné vlastnosti jednotlivých TEXových enginů ani na všemožná starodávná kódování národních abeced. Vše totiž běží výhradně v LuaTEXu a v Unicode.

V únoru 2020 jsem oslovil odběratele TEXového listu `cstex`, že jsem zveřejnil alfa verzi OpTEXu a uvítám případnou diskusi nad koncepcí a požadovanými vlastnostmi. Děkuji všem, kteří se mi v té době ozvali. Tehdy to bylo velmi nehotové dílo, které (ačkoli bylo okamžitě zařazeno do TEXlive i MikTEXu) vlastně vznikalo uživatelům pod rukama. Nyní, po roce intenzivního vývoje, OpTEX dospěl k verzi 1.0 a je tedy ustálen a připraven k běžnému použití. V tomto článku bych chtěl o něm zevrubně seznámit české a slovenské uživatele TEXu.

Klíčová slova: OpTEX, LuaTEX, formát TEXu

Hlavní vlastnost: přímočaré použití TEXu

OpTEX je LuaTEXový formát, podobně jako LATEX nebo ConTEXt. Na rozdíl od těchto dvou jmenovaných ale nevytváří novou vrstvu pro řízení dokumentu nad primitivním TEXem. Předpokládá se, že uživatel použije makra OpTEXu ve výchozí podobě nebo je pozmění k obrazu svému, když třeba řeší nějaký konkrétní typografický požadavek. Makra tedy neřeší vymezení nějaké odlišné syntaxe od syntaxe TEXu ani stovky nově deklarovaných parametrů, kterými je možné nastavovat vlastnosti při zpracování dokumentu. Mezi takovými parametry nakonec pokročilým uživatelům stejně něco bude chybět a budou muset sáhnout po možnostech, které nabízí výhradně jen primitivní úroveň samotného TEXu. Proč tedy tímto přístupem nezačít rovnou?

Tento koncept si uvedeme na příkladu. Chceme-li v OpTEXu nastavit šířku sazby, použijeme primitivní registr `\hsize` a píšeme třeba `\hsize=10cm`. V LATEXu je naproti tomu pro uživatele deklarovaný registr `\textwidth` a v dokumentaci se dočteme, že je možné jej nastavit pomocí `\setlength{\textwidth}{10cm}`, tedy odlišnou syntaxí od primitivní. Je také potřeba pořádně vědět, v jakých případech se hodnota deklarovaného registru `\textwidth` prokopíruje do primitivního `\hsize`. Věci jsou tedy daleko komplikovanější.

doi: 10.5300/2020-3-4/127

ConTEXt v tomto ohledu není o moc přímočařejší, spíše naopak. Je třeba nastavit parametr `width` v jistém kontextu toho ConTEXtu a v key-value syntaxi. A jak to souvisí s primitivním `\hsize` je těžko dohledatelné. Je zde vytvořena zcela nová vrstva nabízející nastavení sazby nad původní TEX primitivní vrstvou.

V OpTEXu můžeme nastavit umístění sazby na stránce pomocí primitivních `\hoffest` a `\voffset`, ovšem můžeme také celkově nastavit okraje sazby velmi pohodlně jedním příkazem `\margins`, což je zkratka za nastavení `\hsize`, `\vsize`, `\hoffset` a `\voffset` v souvislosti se znalostí fyzického rozměru papíru. LATEX potřebuje k takovému nastavení okrajů stránky externí balíček `geometry`.

OpTEX tedy primitivní vrstvu TEXu nezakrývá, naopak předpokládá její časté použití. Pracujeme s TEXem, stačí nám tedy vědět, co a jak dělá TEX, a nezkoumat nějaké další syntaxe, parametry a nové terminologie a algoritmy s tím spojené.

Dokumentace

Na stránce 26 dokumentace k OpTEXu [2] najdete sumární přehled příkazů použitelných pro značkování dokumentů psaných v OpTEXu. Ke každému příkazu vede odkaz na některou z předchozích 25 stránek, kde je příkaz dokumentován z uživatelského pohledu podrobněji. Takže autorovi obsahu dokumentu v zásadě stačí těchto 25 stránek. Ovšem zvídavý autor může v této části dokumentace znovu kliknout myší na jméno příkazu a octne se v místě, kde je technický rozbor implementace příkazu a výpis odpovídající části zdrojového kódu. Tato technická část dokumentace má zhruba zbylých 170 stránek.

Dokumentace je pochopitelně vytvořená OpTEXem. Při jejím zpracování se čtou aktuální zdrojové kódy OpTEXu a k nim připojené průvodní texty a toto je zařazeno do sazby technické části dokumentace. Takže to připomíná literální programování, kde máme ve stejném místě, jako jsou zdrojové texty maker, rovnou napsaný i technický rozbor problému.

Aby mohl čtenář rozumět technické dokumentaci a mohl si přizpůsobit makra dle svého přesvědčení a navíc přesně rozuměl, co se při sazbě jeho dokumentu děje, musí znát TEX. S ním se může seznámit v jiném dokumentu [3], kde jsem se v kostce snažil komprimovat základní informace o TEXu a plainTEXu včetně přehledu všech jeho primitivních kontrolních sekvencí a jejích vlastností do 26 stránek textu. V takto komprimované podobě se informace o kompletních vlastnostech TEXu (včetně jeho běžných rozšíření) asi nikde jinde nevyskytují.

Třetí dokumentací, která doplňuje předchozí dvě, je technický souhrn chování TEXu, maker plainTEXu a Unicode-math v matematické sazbě [4]. Rovněž se jedná o velice zhuštěnou, ale kompletní, informaci na třiceti stránkách.

A to je úplně všechno. Znalosti uvedené v těchto třech dokumentech umožní uživateli získat absolutní kontrolu nad sazbou. Nepotřebuje k tomu tři tisíce LATEXových balíčků včetně dokumentace ke každému z nich (těch balíčků je cca

desetkrát více než je TEXových primitivních příkazů) ani se nepotřebuje vyznat ve 180 PDF souborech monstrózní ConTEXtové dokumentace.

Na cestě k absolutní kontrole nad sazbou může český či slovenský čtenář také využít [5] a ostatní čtenář třeba [6].

Základní vlastnosti

OpTEX jsem již představil v článku [7]. Povědomí o základních vlastnostech OpTEXu můžete také získat prolistováním prvních 26 stránek již zmíněné dokumentace [2]. Zde uvedu jen stručný souhrn a k některým věcem se vrátím v dalších sekcích tohoto článku.

- Systém pro výběr fontů,
- Unicode-math,
- možnost nastavení barev,
- vkládání obrázků včetně Inkscape výstupu s popisky,
- kresba jednoduché grafiky v závislosti na velikosti sazby,
- vkládání objektů na přesně vymezenou pozici,
- externí, interní odkazy včetně hyperlinků,
- automatické generování obsahu,
- automatické generování seznamů referencí přímo z `.bib` souborů,
- automatické generování rejstříků,
- pohodlné nastavení okrajů sazby,
- vzory dělení pro všechny jazyky dostupné v distribuci,
- přepínání automaticky generovaných textů podle zvoleného jazyka,
- poznámky pod čarou,
- poznámky v okraji sazby,
- listingy včetně automatického zvýraznění dle syntaxe jazyka,
- záložky pro PDF prohlížeč v Unicode,
- pohodlná tvorba tabulek,
- tisk textů lorem ipsum dolor sit,
- možnost snadné přípravy prezentací pro zpětný projektor,
- předdefinované styly report a letter,
- nástroje pro programátory maker (cykly, čtení údajů klíč-hodnota, ...),
- oddělené jmenné prostory pro uživatele a pro programátory maker.

Není třeba externích programů

V LATEXu se pro sestavování bibliografických referencí používá BibTEX nebo Biber, pro seřazení rejstříku Makeindex nebo Xindy a pro automatické zvýraznění syntaxe v listingu Python nebo podobný nástroj. Nic z toho OpTEX nepotřebuje, pro plnění těchto úkolů si vystačí sám. Vše probíhá na úrovni maker TEXu.

OpTEX čte přímo BibTEXové soubory `.bib` a sestavuje z nich seznam referencí. Stačí tedy dva průchody dokumentem a máte citace v pořádku. Při prvním průchodu se OpTEX seznámí s použitými referencemi v `\cite` příkazech a v místě seznamu literatury je rovnou použije, tj. vyhledá z `.bib` souboru potřebné údaje, správně je seřadí, přidělí k nim čísla a vytiskne seznam v souladu s použitým bibliografickým stylem. V druhém průchodu pak přidělení čísel v seznamu referencí použije v místech `\cite` příkazů. Je to jistě jednodušší než známá čtveřice `latex`, `bibtex`, `latex`, `latex`. K nastavení vlastností bibliografického stylu se používají přímo makra TEXu.

OpTEX sestaví rejstřík přímo z dat z předchozího průchodu tak, že použije sofistikovaný vícekrokový algoritmus řazení, který se dá konfigurovat pro většinu běžných jazyků. Pro češtinu, slovenštinu a angličtinu funguje bez problémů. Dodatečnými makry se pak dá řídit podoba rejstříku. Je daleko příjemnější vědět, že nemusíme pamatovat na závěrečné spuštění `makeindex` po posledních korekturách, protože celý „Makeindex“ je přímo součástí OpTEXu a projeví se při každém zpracování dokumentu. Navíc funguje relativně rychle díky řadicímu algoritmu mergesort, který využívá makrojazyk TEXu velmi efektivně.

Automatické zvýraznění syntaxe tištěných listingů probíhá rovněž na úrovni makrojazyka TEXu. Jednotlivá pravidla pro zvýraznění se konfigurují v příslušných makro souborech `hisyntax-c`, `hisyntax-html` atd. Není zapotřebí externího programu.

Systém pro výběr fontů

OpTEX nabízí uživateli pohodlnou manipulaci s fonty a fontovými rodinami se zachováním základní koncepce z plainTEXu, že tedy mezi variantami jednou vybrané rodiny fontů přepínáme pomocí `\rm`, `\bf`, `\it` a `\bi`. Mimoto je možné předsunout před takové přepínače „fontové modifikátory“, např. `\caps`, `\cond`, pokud to fontová rodina podporuje. Rodinu fontu uživatel nastaví pomocí `\fontfam`. Modifikátory mohou přepínat na sobě nezávislé vlastnosti a mohou se všelijak shlukovat a kombinovat.

Uživatel také může použít přímo primitivní příkaz `\font` pro výběr fontů a může snadno zakomponovat vytvořený primitivní přepínač fontu do maker tak, že bude tento font zvětšen podle uživatelem stanovené velikosti kdekoli v dokumentu společně s ostatními fonty zařazenými do fontových souborů. Není tedy třeba (jako v plainTEXu) psát celou sadu primitivních příkazů `\font` pro různé velikosti.

OpTEX umožňuje zařazení deklarací rodin fontů do tzv. fontových souborů. Desítky rodin fontů již takto zařazeny jsou. Uživatel se může inspirovat, jak je to uděláno, a zařadit další rodinu. Nebo se o formátu těchto souborů dočte v technické dokumentaci [2].

Uživatel si může OpTEXem vytisknout vzorníček všech rodin fontů, které jsou zařazeny do fontových souborů, jednoduše pomocí

```
\fontfam[catalog]
\bye
```

V tomto vzorníčku jsou přehledně uvedeny všechny přepínače variant a všechny modifikátory dostupné v každé jednotlivé rodině následované ukázkou textu z takto vybraného fontu. Viz také [8].

OpTEX přirozeně nabízí možnost použití „fontfeatures“ z OpenType fontů. Je k tomu vyhrazen fontový modifikátor `\setff` s parametrem, který obsahuje zkratku příslušné vlastnosti. Jedná se o standardní čtyřpísmennou zkratku známou z dokumentace k fontu, dokumentace k OpenType standardu nebo z výpisu příkazu `otfinfo -f`. Nad těmito názvy „fontfeatures“ není vytvořena žádná nová vrstva jiných názvů, jako to dělá třeba `fontspec` z LATEXu. Věci se totiž v OpTEXu dělají přímočaře.

Podporuje-li rodina optické velikosti fontů, podporuje je i OpTEX. Výchozí rodinou fontů je Latin Modern, která optické velikosti umožňuje věrna tradici, kterou v této věci zavedl už Donald Knuth, když představil svou rodinu fontů Computer Modern.

V OpTEXu pracujeme výhradně s fonty v Unicode (OpenType). Tím se vyhneme všelijakým obskurnostem s různými starodávnými kódováními. Matematická sazba také probíhá v Unicode-math, což výrazně zjednodušuje práci s matematickým fonty. Máme jenom jeden font se všemi matematickými abecedami a se všemi symboly pohromadě na jednom místě. Práce s více matematickými rodinami fontů (jako v klasickém plainTEXu) tím není ale omezena. Není ovšem nutné ji moc využívat, ledaže by základní matematický font neobsahoval všechny potřebné symboly a bylo nutno sazbu kombinovat s dalším matematickým fontem.

Křehká makra v názvech sekcí neexistují

Uživatelé LATEXu asi znají pojem „fragile command“ a jaké s tím je trápení. Ve stručnosti: nelze napsat do názvu sekce, kapitoly atd. jednoduše každé makro, protože takové makro se může na cestě k automaticky generovanému obsahu rozsypat. Tato cesta totiž vede expanzí přes pracovní textové soubory. Divím se, že po skoro třiceti letech, kdy byly do eTEXu přidány příkazy `\detokenize` a `\scantokens`, stále LATEXoví uživatelé bojují s křehkými příkazy a snaží se to řešit starodávným a komplikovaným způsobem pomocí `\protect` nebo `\DeclareRobustCommand`, což vkládá tajemnou závěrečnou mezeru do názvu makra. To zase způsobuje méně snadné trasování.

OpTEX čte názvy sekcí ve verbatim módu a tokenizuje je až v okamžiku použití. Tím odpadají veškeré starosti s křehkými makry. Navíc to umožní vkládat do názvů sekcí i verbatim text, například:

```
\sec Titul včetně ‘\takove{‘ záležitosti
```

Všimněte si, že verbatim konstrukce (uzavřená mezi znaky `‘...‘` deklarované v OpTEXu jako `\verbchar`), umožňuje zapsat i nepárové TEXové závorky. To je v LATEXu v podstatě nemožné, protože titul je v LATEXu ohraničen pomocí těchto závorek v páru. V OpTEXu je titul ohraničen koncem řádku. Verbatim konstrukce z titulů sekcí a kapitol správně doputují do obsahu, do plovoucího záhlaví či do PDF záložek. Bez kompromisů, kterým se musel podřídit například LATEXový balíček `cprotect`.

Odlehčené značkování

Značkovací jazyk OpTEXu vychází z předpokladu, že zdrojový text `.tex` je určen především pro člověka. Stroj by se tomu měl podřídit. Člověk tento text vytváří, čte jej, manipuluje s ním. Proto jsem se od počátku vzniku svých maker snažil o co nejstručnější, efektivní a přehledné značkování. Takže jsem se vyvaroval nadbytečnému množství závorek `{...}` a nepřijal jsem vnořující se konstrukce z LATEXu `\begin{...}...\end{...}`. Text dokumentu *nekódujeme*, ale prostě *píšeme*.

Faktum je, že existují ještě přívětivější způsoby značkování, jako je Markdown. Jenže v tomto jazyku zase nemáme absolutní kontrolu nad sazbou, kterou potřebujeme, když vytváříme PDF verzi dokumentu. Je jistě možné, že nějaký autor dokumentu použije Markdown a pak jej nechá zkonvertovat do OpTEXu. Sazeč poté má k dispozici verzi dokumentu `.tex`, ve které doplní další potřebné věci a vytvoří sazbu pomocí OpTEXu. Takže sazeč manipuluje s `.tex` souborem, a nakonec `.tex` soubor je konečná verze dokumentu určená k archivaci. Soubor `.tex` by měl být ideálně středem všeho, z něj proběhne jednak sazba do PDF a jednak konverze do dalších případných formátů a je určen k archivaci pro možné budoucí použití například při novém vydání dokumentu.

Na základě těchto úvah vznikl dokument OMLS [9], který vymezuje standard značkování souborů `.tex` při použití OpTEXu. Tento standard obsahuje stanovení syntaxe a významu všech značek, které umožní v OpTEXu vytvořit dokument s obvyklými vlastnostmi (podobně jako Markdown). Je určen pro případné autory konvertorů z `.tex` do jiných formátů a obráceně. Autoři těchto programů se tedy mohou opřít o přesné vymezení možností, které se v `.tex` souborech mohou vyskytovat. A je-li tam nějaká jiná konstrukce mimo standard OMLS, má být ignorována. Pro případné zpracování matematických vzorečků mezi `$...$` nebo `$$...$$` se očekává interpret analogický systému MathJax.

Dokument OMLS [9] je poměrně nová věc, zatím tedy podle něj skutečné aplikace nejsou vytvořeny.

Odlehčený formát

Nejen značkování, ale i implementace maker v OpTEXu je odlehčená. Makra jsou co možná nejjednodušší. Uvedu dvě srovnání.

Velikost formátového souboru `optex.fmt` je na mé implementaci TEXlive cca 750 kB. Srovnáme-li to s velikostí ostatních běžných formátových souborů pro LuaTEX (viz tabulku 1), shledáme, že je skoro nejmenší.

Tabulka 1: Velikosti formátových souborů `.fmt` v TEXlive

OpTEX	eplain	csplain	LuaLATEX	ConTEXt
750 kB	1,2 MB	380 KB	3,2 MB	11 MB

Přitom ve formátu OpTEXu je skrytá srovnatelná inteligence LATEXového jádra `latex.fmt` včetně doplňkových balíčků `xcolor`, `hyperref`, `url`, `listings`, `biblatex`, `graphicx`, `geometry`, `amsmath`, `amsymb`, `fontspec`, `unicode-math`, `cprotect`, `eqparbox`, `tabularx`, `booktabs`, `keyval` a mnoha dalších. K dispozici je možnost použít všechny dostupné vzory dělení všech jazyků. Vzory dělení ve `.fmt` souboru totiž nepřekážejí, LuaTEX jako jediný engine je dokáže načíst až podle potřeby během zpracování dokumentu.

Druhé srovnání vychází z testovacího souboru:

```
\loggingall
\cslang             % české vzory dělení slov
\fontfam[Adventor]  % fontová rodina Adventor (Unicode)

Ahoj světe!
\bye
```

a jeho srovnatelných variant vytvořených pro ostatní dva formáty. Sledujeme počet řádků v `.log` souboru při `\loggingall`. Dále vypneme `\loggingall` a sledujeme čas. Viz tabulku 2.

Vidíme, že soutěž o nejmenší počet trasovaných řádků LATEX zcela prohrává. Provádí asi dvatisíckrát více interních úkonů než OpTEX. ConTEXt se zase potřebuje spustit za všech okolností třikrát, i když to v tomto příkladě není nutné. A dobíhá při měření rychlosti do cílové rovinky poslední, i kdyby si dal to kolečko

Tabulka 2: Počet řádků v `.log` souboru při plném trasování a čas bez trasování

OpTEX	LuaLATEX	ConTEXt
1,8 k	3,6 M	3×231 k
0,5 s	1,0 s	$3 \times 1{,}1$ s

jen jednou. Asi přeci jen nějakou dobu trvá, než se přečte a dekomprimuje 11MB binární `.fmt` soubor.

Příjemnou drobností je, že OpTEX nevytváří pracovní soubory, když nepotřebuje. Takže výše uvedený příklad vygeneroval jen `.log` a `.pdf`, zatímco LATEX si založil `.aux`, který v tomto případě nepotřebuje, a ConTEXt si založil `.tuc` soubor. Nezakládání zbytečných pracovních souborů možná oceníte, když si budete dělat krátké testovací dokumenty a nebudete kvůli tomu muset mazat z disku desítky `.aux` souborů a jemu podobných.

OpTEX v rámci pravidla „dělat věci jednoduše“ zakládá (když je to nutné) jediný pracovní soubor `.ref`, kde má pohromadě vše potřebné pro obsah, bibliografii, dopředné reference, rejstřík atd. Nemáte tedy na disku sbírku souborů `.toc`, `.aux`, `.tof`, `.idx`, `.ind`, `.bbl` a dalších.

LATEX občas trpí problémem, že se zasekne při opakovaném čtení `.aux` souboru, například když změníte hlavní jazyk dokumentu. To v OpTEXu nenastane. Je navíc vybaven kontrolou verze svého `.ref` souboru, aby nedošlo ke zmatení čtení například po přidání nového rysu do nové verze. Vidí-li OpTEX nekompatibilní verzi `.ref` souboru, ignoruje jej a založí si nový. Makra OPmac také zakládají `.ref` soubor, ale s jiným číslem verze, takže nikdy nedochází ke vzájemnému zmatení.

Balíčky v OpTEXu

Ačkoli OpTEX obsahuje vše potřebné pro tvorbu běžných dokumentů, občas se vyskytne požadavek něco dalšího implementovat pro konkrétní účel. Řešení k velkému množství takových požadavků, z nichž by si asi každý vyžádal samostatný LATEXový balíček, vkládám do souhrnného návodu na stránce OpTEX triků [10]. Zejména, pokud si toto řešení vyžádá jen několik málo řádků makro kódu, což je poměrně obvyklý případ.

Někdy ale je potřeba vyřešit něco obsáhlejšího. OpTEX nabízí možnost tvorby balíčků, které jsou pak do dokumentu zavedeny pomocí `\load`. Balíčky může vytvořit dle pokynů v technické části dokumentace kdokoli, ovšem zatím jsem několik prvních balíčků připravil jako ukázku sám:

- `qrcode` pro výpočet a sazbu QR kódů,

- `vlna` automaticky přidává nezlomitelné mezery za předložkami s využitím `luavlna`,
- `emoji` umožní používat barevné emotikony z rozsáhlé nabídky podle [11].

Jmenné prostory

Uživatel může definovat a používat kontrolní sekvence typu `\slovo`. Některé z nich nesou význam primitivní kontrolní sekvence nebo makra OpTEXu, ale uživatel si je může předefinovat bez ztráty funkčnosti OpTEXu. Má tedy k dispozici jmenný prostor typu `\slovo` a když si něco předefinuje a v původním významu nikdy ve svém dokumentu přímo nepoužije, nemá problém. Proč to funguje? OpTEX interně používá duplikáty primitivních sekvencí tvaru `\_slovo` a interní makra má rovněž v tomto tvaru. Uživatel tedy bez problému může provést třeba `\def\fi{finito}` a žádné interní makro se nerozsype, protože interní makra OpTEXu používají konstrukce `\_if...\_fi`. Když ale uživatel sám používá svá makra obsahující `\if...\fi` a napíše omylem `\def\fi{finito}`, samozřejmě se mu jeho makra rozsypou. Ale to se celé děje uvnitř jeho uživatelského prostoru. Uživatel by měl mít představu o jménech, která sám deklaruje a pak používá.

OpTEX nabízí také izolované jmenné prostory pro každý případný balíček. Kontrolní sekvence primitivní či deklarované v OpTEXu tam mohou být použity v read módu ve tvaru `\_slovo` a dále si balíček může definovat vlastní interní sekvence ve tvaru `\_pkg_slovo`. Přitom tvůrce maker nemusí opakovaně psát `\_pkg_foo`, `\_pkg_bar` (až oči přecházejí), ale deklaruje si jednou `\_namespace{pkg}` a používá daleko čitelnější syntaxi `\.foo`, `\.bar`. Konceptu, kdy se v makro kódu neustále opakují názvy jmenných prostorů (jako to dělá Expl3), jsem se s radostí vyhnul.

Z pravidla o uživatelském jmenném prostoru zatím existuje jedna smutná výjimka. Řídicí sekvence `\par` je zadrátována jako jediná hluboko v TEXu samotném a rodí se například na prázdných řádcích. Její předefinování tedy ovlivní chování TEXu. Už jsem ale připravil záplatu ke zdrojovým textům `pdftex.web`, která umožní deklarovat tuto rodící se sekvenci jakkoli jinak, ne nutně jako `\par`. Tuto záplatu mám kompletně připravenou, zdokumentovanou a testovanou. Bohužel jsem ji nestihl podat včas pro nový TEXlive 2021. Takže se to objeví asi až za rok. Do té doby mám v dokumentaci OpTEXu i pro uživatele informaci o výjimce ve jmenném prostoru: „vyvarujte se `\def\par`, pokud netušíte, co tím můžete způsobit“. Předpokládám, že záplatu z pdfTEXu převezmou v budoucnu i tvůrci LuaTEXu, jako to udělali i s jinými mými návrhy na změnu chování pdfTEXu. Takže to pak s radostí využiji v OpTEXu.

Využití rozšířených vlastností LuaTEXu

LuaTEX umí především interpretovat Lua kódy. To jsem využil v minimální míře, některé věci by se daly v budoucnu ještě přepsat. Lua skripty jsem použil při generování Unicode řetězců pro PDF záložky a při interpretaci znaků `_` a `.` v řídicích sekvencích. Dále jsem je využil na drobnosti jako např. porovnávání stringů. OpTEX načítá vlastní sadu základních Lua funkcí `optex.lua` a ukládá si ji přímo do formátu. Při každém startu pak tuto sadu funkcí inicializuje prostřednictvím `\everyjob`.

Dále je v Lua implementován font-loader pro OTF fonty. Ten se spustí až dle potřeby, při prvním `\fontfam`. Příslušný Lua kód je kompletně čten z LATEXového balíčku `luaotfload`, což bohužel činí OpTEX závislým na tomto balíčku. Tvůrci `luaotfload` přebírají (pravidelně při každé nové verzi) Lua kódy font-loaderu v nezměněné podobě z ConTEXtu a přidávají tam pár svých dalších drobností. Stál jsem před rozhodnutím přebrat font-loader přímo z ConTEXtu (pak by byl ale OpTEX závislý kompletně na ConTEXtu, což je daleko větší bumbrlíček než jen jeden balíček `luaotfload`). Nebo pravidelně přebírat kódy z ConTEXtu a ohýbat je pro své potřeby (to bych ale vlastně dubloval práci tvůrců `luaotfload`). Nebo si font-loader naprogramovat v Lue zcela sám (to ale dosud nikdo jiný nedokázal než Hans Hagen a znamenalo by to vlastně jen vymýšlet jednou objevené kolo a při zveřejnění nového standardu OTF s novými vlastnostmi toto kolo neustále samostatně vylepšovat nezávisle na Hansi Hagenovi). Ponechal jsem tedy kompromis a závislost na `luaotfload`. Netěší mě to, musím neustále sledovat, co tam s těmi kódy v nových verzích jejich tvůrci dělají a zda nedojde k nekompatibilitě se základními Lua funkcemi OpTEXu.

Kromě zmíněného skriptovacího jazyka Lua disponuje LuaTEX dalšími rozšířeními, z nichž některé jsem využil. Například se dají v LuaTEXu velmi pohodlně udělat cykly pracující jen na úrovni expand procesoru, což jsem s radostí využil pro cykly `\fornum` a `\foreach`, které nabízím programátorům maker v kolekci nástrojů rozšířeného API (nad standardními nástroji z plainTEXu).

Velmi praktický je v LuaTEXu například primitivní příkaz `\csstring`, který konečně odstraňuje veškeré tanečky s překážejícím a proměnlivým `\escapechar` při použití klasického `\string`. A takových dobrých drobností by se dalo najít více.

OpTEX tedy využívá na mnoha místech specifické vlastnosti LuaTEXu a není proto snadno přeportovatelný pro jiný TEXový engine (například XƎTEX). Ani to nikdy nebylo záměrem a, domnívám se, není k tomu žádný důvod.

Některé věci jsou udělány jinak

Makra plainTEXu a OPmac jsou do OpTEXu přepsána. Zpětná kompatibilita ale není z dobrých důvodů zcela dodržena. Odlišnosti od plainTEXu jsou přesně popsány v dokumentaci. Uvedu některé příklady.

Výchozí sada fontů je Latin Modern a ne Computer Modern. Když napíšu `\meaning\makro`, pak rovnou vidím výpis makra v sazbě bez obskurních znaků, které se ve fontech Computer Modern odchylují od ASCII. Odlišnost od ASCII a sedmibitovost dělá v dnešní době rodinu Computer Modern nepoužitelnou.

Knuth rozhodl dosti nešťastně dát počátek souřadnic pro rozmístění sazby do bodu (1in, 1in) od levého horního rohu papíru. To je velmi frustrující při výpočtech skutečných okrajů, zejména když v Evropě počítáme v milimetrech. OpTEX má tento počátek jednoduše v levém horním rohu papíru, takže třeba hodnota `\hoffset` přímo udává velikost levého okraje.

V makrech plainTEXu (a bohužel toto je pak převzato i v jiných formátech) se vyskytují interní řídicí sekvence ve tvaru `\p@`, `\f@@t` a další. To je značně nečitelné. Kdo mě zná, jistě ví, že tento způsob zápisu maker bytostně nesnáším. Takže v OpTEXu nic takového nenajdete.

Rozhodl jsem se uživatele přinutit psát přímo akcentované znaky a nikdy je neskládat pomocí `\"a`, `\v c` a podobných. Takže makra pro akcenty nejsou záměrně v OpTEXu definována. Uživatel může tyto jednoznakové řídicí sekvence s výhodou použít k jakémukoli jinému účelu. Pokud by se chtěl konzervativní uživatel vrátit k nostalgickému značkování akcentů, může použít deklaraci `\oldaccents`, ale nedoporučuji to. Naopak, uživatel může deklarovat `\csquotes`, `\frquotes`, `\dequotes` nebo podobné a pak může psát `\"text"` nebo `\'text'` a má dvojité a jednoduché uvozovky dle pravidel zvoleného jazyka.

Makra OPmac jsou zcela nově přepsána a v mnohém doplněna o další vlastnosti. Značkování na uživatelské úrovni zůstalo stejné. Ale staré soubory doplňujících maker vycházejících z OPmac nejsou bohužel snadno přenositelné pro OpTEX, protože typicky se v těchto souborech autor maker opírá o nějaké vnitřní záležitosti OPmac, které vypadají v OpTEXu zcela jinak. Opustil jsem tak kompatibilitu obrovského množství svých vlastních makro souborů, ale už nyní vidím, že se to vyplatilo. Člověk tak nezůstane stát na jednom místě, ale posune se dál, k modernímu způsobu využití TEXu.

Odkazy

1. OLŠÁK, Petr. *OpTEX: LuaTEX format with extended plainTEX macros* [online]. 2021 [cit. 2020-03-18]. Dostupné z: `http://petr.olsak.net/optex/`.

2. OLŠÁK, Petr. *OpTEX: Format Based on Plain TEX and OPmac* [online]. 2021 [cit. 2020-03-18]. Dostupné z: http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf. Verze 1.01.
3. OLŠÁK, Petr. *TEX in a nutshell*. Praha: České vysoké učení technické, 2020. Dostupné také z: http://petr.olsak.net/ftp/olsak/optex/tex-nutshell.pdf.
4. OLŠÁK, Petr. *Typesetting Math with OpTEX* [online]. 2021-01 [cit. 2020-03-18]. Dostupné z: http://petr.olsak.net/ftp/olsak/optex/optex-math.pdf. Verze 03.
5. OLŠÁK, Petr. *TEXbook naruby*. 2. vyd. Konvoj, 2001. ISBN 80-7302-007-6. Dostupné také z: http://petr.olsak.net/tbn.html.
6. EIJKHOUT, V. *TEX by topic* [online]. 2007 [cit. 2020-03-18]. Dostupné z: http://mirrors.ctan.org/info/texbytopic/TeXbyTopic.pdf. Verze 1.1. Viz také texdoc texbytopic.
7. OLŠÁK, Petr. OpTEX – A new generation of Plain TEX. *TUGboat*. 2020, roč. 41, č. 3, s. 901–907. ISSN 0896-3207. Dostupné také z: http://petr.olsak.net/ftp/olsak/bulletin/tb128olsak-optex.pdf.
8. OLŠÁK, Petr. *Font Catalogue generated by OpTEX* [online]. 2021-03-05 [cit. 2020-03-18]. Dostupné z: http://petr.olsak.net/ftp/olsak/optex/op-catalog.pdf.
9. OLŠÁK, Petr. *OpTEX Markup Language Standard* [online]. 2021 [cit. 2020-03-18]. Dostupné z: http://petr.olsak.net/ftp/olsak/optex/omls.pdf. Verze 0.1.
10. OLŠÁK, Petr. *OpTEX: tips, tricks, howto* [online] [cit. 2020-03-18]. Dostupné z: http://petr.olsak.net/optex/optex-tricks.html.
11. ZENG, Xiangdong. *The emoji package* [online]. 2020 [cit. 2020-03-18]. Dostupné z: http://ctan.org/tex-archive/macros/luatex/latex/emoji/emoji-doc.pdf. Viz také texdoc emoji.

Summary: OpTEX – successor of OPmac macros for LuaTEX

The article describes OpTEX [1] – a LuaTEX format based on plain TEX and OPmac. This macro package was introduced in TugBoat [7] and now, it is presented for Czech and Slovak users. The presentation is slightly different than in the cited article [7]. For example, the comparison with LaTEX and ConTEXt is mentioned here including benchmarks in tables 1 and 2.

Keywords: OpTEX, LuaTEX, TEX format

Petr Olšák
http://petr.olsak.net

ConTEXt Meeting 2020

Jano Kula

Článek stručně popisuje konferenci ConTEXt Meeting 2020 v Sibřině.

Klíčová slova: ConTEXt Meeting.

V pandemickém roce 2020 proběhl 14. ConTEXt Meeting v Sibřině u Prahy podle plánu 6.–12. září. Ostatní tradiční konference buď neproběhly (BachoTEX, DANTE Meeting), nebo se odehrály online (TUG 2020). Měli jsme štěstí, že jsme se s dvouletým předstihem trefili do termínu, kdy cestování už (a stále ještě) bylo možné. O týden později by už většina účastníků přijet nemohla. ConTEXt Meeting se tak pravděpodobně stal jediným osobním setkáním příznivců TEXu v roce 2020. I když účastníci z Francie a Itálie museli svou účast na poslední chvíli odvolat, sešla se obvyklá společnost přátel z Nizozemska a Německa. Ani československých účastníků nebylo málo. V celkově nízkém počtu, díky za něj, tvořili zhruba třetinu osazenstva.

Jelikož v LuaTEXu a ConTEXtu Mark IV již k žádnému novému vývoji nedochází, byla většina novinek věnována LuaMetaTEXu [1] v přednáškách Hanse Hagena. Prostor byl i pro náročnější lekce z MetaPostu z dílny Taco Hoekwatera nebo Honzy Šustka. Nejoblíbenější přednáškou se stala prezentace studentek Tomáše Hály Adriany Kašparové a Tamary Kocurové, které v rámci svých bakalářských prací společně vyvinuly ConTEXtový modul pro kreslení grafů. Nejenom, že se za krátkou dobu musely seznámit s mnoha záludnostmi sazby ConTEXtem a přispěly komunitě novým modulem, ale i grafická úprava manuálu [2] převyšuje obvyklou úpravu dokumentace TEXu.

Není ConTEXt Meetingu bez výletu. Díky nízkému počtu účastníků jsme mohli navštívit kapli sv. Kříže na Karlštejně, která by sama o sobě mohla být vrcholem dne. My jsme se však přesunuli do nedalekých lomů Amerika. V nich se nachází Hagenova štola, ve které se po druhé světové válce dle místní legendy [3] ukrýval německý voják Hans Hagen. Místní dobrovolníci, starající se o štolu a opuštěný lom, se sešli v hojném počtu, aby si mohli prohlédnout živého Hanse Hagena.

Kompletní program konference i některé fotografie z ní si můžete prohlédnout na následujících stránkách. Abstrakty přednášek, většinou doplněné o prezentaci, najdete na stránkách konference [4].

DOI: 10.5300/2020-3-4/139

Pondělí 7. 9. 2020	
Hans Hagen	*From MKII to MKIV to LMTX: the fundamental differences*
Hans Hagen	*LuaMetaTEX: where do we stand*
Arthur Rosendahl	*What is new in Unicode 13?*
Hans Hagen	*A conceptual upgrade... examples and discussion*
Arthur Rosendahl	*A ConTEXt user about LATEX development*
Jan Šustek	*Animations from constructive geometry in Metapost*
Harald König	*Zigbee building monitoring*
Úterý 8. 9. 2020	
Hans Hagen	*Tokens*
Hans Hagen	*Additional data types*
Hans Hagen	*Implementors*
Taco Hoekwater	*MetaPost macros and definitions*
Ulrik Vieth	*Math fonts*
Středa 9. 9. 2020	
Hans Hagen	*ECMASCRIPT: for those who love JAVASCRIPT*
Taco Hoekwater	*The new sourcebrowser software*
Čtvrtek 10. 9. 2020	
Willi Egger	*Book production*
Tomáš Hála	*The concept of* `\setupdocument`
Taco Hoekwater	*Wiki news*
Pátek 11. 9. 2020	
Tamara Kocurová, Adriana Kašparová	*Module for drawing statistical charts*
Hans Hagen	*MetaPost: making simple fonts*
Ton Otten	*Dealing with details: how I make finals from xml input, a demo*
Hans Hagen	*SVG: for what it's worth*
Taco Hoekwater	*Work in progress: a CSS parser with LPEG*
Arthur Rosendahl	*ConTEXt for wedding*

Statek U Škodů, kde se ConTEXt Meeting konal již podruhé. Pohled do dvora.

Tělesná teplota účastníků byla kontrolována hromadně.

Snídaně venku.

Přednášky uvnitř.

Návštěva Hagenovy štoly v lomech Amerika.

Dobrovolníci spolku Hagen s Hansem Hagenem.

Pestrá účast: veteráni Hans Hagen a Willi Egger i studentky Mendelovy univerzity Tamara Kocurová (vlevo) a Adriana Kašparová.

Adriana Kašparová a Tamara Kocurová při večerním ladění své přednášky. Poslední rady udělují Hans Hagen a Tomáš Hála.

Účastníci první poloviny konference (zleva, přední řada: Ton Otten, Willi Egger, Tomáš Hála, Jan Šustek, zadní řada: Wolfgang Schuster, Michal Vlasák, Barbara Wilińska, Henning Hraban Ramm, Ulrik Vieth, Hans Hagen, Arthur Rosendahl [Reutenauer], Jano Kula, Harald König).

Účastníci druhé poloviny konference (zleva, přední řada: Henning Hraban Ramm, Arthur Rosendahl [Reutenauer], Tamara Kocurová, Adriana Kašparová, Willi Egger, Barbara Wilińska, zadní řada: Ulrik Vieth, Wolfgang Schuster, Hans Hagen, Jano Kula, Ton Otten, Tomáš Hála, Harald König).

Příští ConTEXt Meeting se uskuteční 20.–25. 9. 2021 v Bassenge (Belgie), v dalším roce pak v Dreifeldenu (Německo) tradičně v některém zářijovém týdnu. Ročník 2023 se přesune do Nizozemska či sousední Belgie. Na další setkání v Čechách se můžete těšit nejdříve v roce 2024.

Odkazy

1. LUATEX DEVELOPMENT TEAM; CONTEXT DEVELOPMENT TEAM. *LuaMetaTEX Reference Manual* [online]. 2021 [cit. 2021-03-02]. Dostupné z: `https://www.pragma-ade.nl/general/manuals/luametatex.pdf`.
2. KOCUROVÁ, Tamara; KAŠPAROVÁ, Adriana; HÁLA, Tomáš. *Drawing statistical charts* [online]. Brno, 2020 [cit. 2021-03-02]. Dostupné z: `https://akela.mendelu.cz/~thala/statcharts/statistical-charts.pdf`.
3. DRAHOTA, Leoš. *Legenda jménem Hans Hagen* [online]. 2013-12-17 [cit. 2021-03-02]. Dostupné z: `https://www.lomy-amerika.cz/legenda-jmenem-hans-hagen/`.
4. *ConTEXt User Meeting 2020: Abstracts* [online] [cit. 2021-03-02]. Dostupné z: `https://meeting.contextgarden.net/2020/abstracts.shtml`.

Summary: ConTEXt Meeting 2020

This is a short report about the ConTEXt Meeting 2020 in Sibřina near Prague.

Keywords: ConTEXt Meeting

MetaPost Definitions

Taco Hoekwater

The paper presents all syntactic rules for definitions in MetaPost. It contains many good and bad examples of definitions.

Keywords: MetaPost, definitions

1. Introduction

Definitions in MetaPost are a fairly complicated subject. This talk and paper tries to cover everything you need to know about writing your own definitions, but it assumes a fair bit of familiarity with MetaPost's data types and general syntax. In particular, I assume you read last year's 'sparks, tags, suffixes and subscripts' article [1].

2. Macro definitions

The `def` command defines a token to be replaced with a replacement text. This is close to how macros work in TEX, and therefore the easiest to explain. So let's start with that.

In its simplest form, it looks like this:

```
def ⟨symbolic token⟩ =
    ⟨replacement text⟩
enddef;
```

Since each `def` is a complete statement in MetaPost, the semicolon after `enddef` is required. There is no need for a semicolon to end the ⟨replacement text⟩, because the expansion of the macro can happen in the middle of an expression, where an extra semicolon would interfere. The ⟨replacement text⟩ itself can be almost anything (with some minor limitations, see Section 6 for the exact rules).

A simple predefined example is the `--` macro that is used to draw straight lines in path definitions. It is defined like this:

```
def -- = {curl 1}..{curl 1} enddef;
```

2.1. Macros with arguments

Macros are much more useful if they take arguments. Here is where MetaPost is quite different from TEX (or any other language, for that matter). There are

DOI: 10.5300/2020-3-4/147

quite a few ways to write the definition such that the defined macro accepts arguments of some form or another. There are options for delimited arguments and an undelimited argument, and arguments can come in various types as well.

Let's discuss the three basic types of arguments first: These are `expr`, `suffix`, and `text`.

- `expr` arguments are for passing expression values.
- `suffix` arguments are for passing (parts of) variable names.
- `text` arguments are for passing a list of arbitrary tokens.

In the most simple form, defining a macro with an argument looks like:

```
def mymac (expr a) =
...
```

where you can replace `expr` with `suffix` or `text`. We will talk about passing multiple arguments later on.

The parameter name `a` in the example is actually a ⟨symbolic token⟩ itself. It has to be a single symbolic token (not a literal number or a string), but it is not required to be alphanumeric. Any symbolic token will do, except for tokens explicitly set as `outer`. You could do:

```
def mymac (expr ,) =
  , = 5
enddef;
```

although that is only useful in obfuscation contests. What you can not do is use numeric suffixes like in many other languages:

```
def mymac (expr a1) =  % Error
  a1 = 5
enddef;
```

The parameter names do not actually exist as variables, they are only there to give you something to use in the ⟨replacement text⟩. Whatever the value of the parameter is, is stored in a temporary value slot that has no name (these things are called `capsules` internally). You can actually see these slots in the terminal or log if you turn on tracing. In the trace output they are represented as the parameter type in uppercase with a sequence number attached, for example

```
tracingall;
def mymac (expr a) =
  a = 5
enddef;

mymac(5);
```

will show:

```
mymac(EXPR0)->(EXPR0)=5
(EXPR0)<-b
{(b)=(5)}
## b=5
```

While there is not really a parameter named `a`, the use of `a` as a placeholder for the `capsules` does make it seemingly impossible to refer to an actual variable or command named `a`. But there is a solution to that.

If you want to access an outside name `a` within a macro that has a parameter named `a`, you can still do so by using the `quote` command:

```
def mymac (expr a) =
  quote a = a
enddef;

mymac(b);
```

The first ('quoted') `a` is now referring to an outside variable. In this case, it will set up an equation `a = b` because `b` is the replacement value of the parameter `a`.

Using `quote` like this works for all three parameter types, and it can also help you if you happen to have a macro parameter name that matches some MetaPost command's name. However, a small warning: it is usually better to *not* write macros that alter outside variables as side effects because you are likely to confuse yourself. And for access to other macros and/or commands it is much better to come up with more unique parameter names.

2.1.1. `expr` arguments

Arguments of type `expr` pass a value to the macro. The argument has to be a valid expression, and that expression is interpreted to produce a value that is then stored in the temporary variable that is used in the replacement text of the macro.

The expression is interpreted as far as possible first. Here is an example of what that means. If the macro `mymac` is defined to have an argument of type `expr`, you can call it like this:

```
mymac(5);
```

or like this:

```
mymac(2+3);
```

and in both cases the replacement text will see the value `5`. But the value does not have to be 'known'. In these two calls:

```
mymac(5b);
mymac((2+1)*b +2b);
```

the replacement text will see `5b` if the variable `b` is not known at that point in time.

Because the replacement text works with a nameless variable's value, it is not assignable. That means that inside the replacement text, the formal names of `expr` parameters cannot be used on the left side of an assignment (`:=`). For example, this is forbidden:

```
def mymac (expr a) =
  a := 5 % Error
enddef;
```

But that does not mean you cannot alter the value itself, because equations (`=`) with that parameter name are still allowed:

```
def mymac (expr a) =
  a = 5
enddef;

mymac(5b);
```

is correct input and resolves the outer variable `b` to the integer value `1` (its value will remain known even after the macro call).

Be aware though that macros that have such hidden side-effects are hard to maintain, so you need to be quite certain of how the macro will be called if you make use of this. For general purpose macros, it is almost always better to receive and/or return a value instead of modifying the parameter. The above definition would be cleaner if written like this:

```
def mymac(expr a) =
  a
enddef;

5b = mymac(5);
```

Well, this is a silly example of course, but the point should be clear, I hope.

2.1.2. `suffix` arguments

Arguments of type `suffix` pass a 'suffix' to the macro. A 'suffix' is the trailing part of a variable name, possibly consisting of multiple segments, and possibly being the whole name. The argument passed to the macro really is the (partial)

name of a variable. Per the normal rules, if you try to pass an undefined suffix (or whole variable), it is initialised to be of type `numeric`.

```
def mymac(suffix a) =
  a = 5
enddef;

mymac(b);
```

assigns the value of `5` to the variable named `b`, assuming it is of type `numeric`. If it not numeric, an error will be raised.

And

```
def mymac(suffix a) =
  pair a
enddef;

mymac(b);
```

converts `b` into a variable of type `pair`.

Since this is all resolved by name, this:

```
def mymac(suffix a) =
  pair c.a
enddef;

mymac(b);
```

sets up `c.b` to be of type `pair`. The variable `c` itself remains untouched, just as if you wrote

```
pair c.b;
```

without any macro definition.

Macros with suffix parameters can sometimes be a little complicated to read because of the 'passing a name as a parameter value'. Just to be clear, inside the macro there is at *no* time a variable named `c.a`. In fact, if there was a variable `c.a` defined before the macro call, then it will remain completely untouched.

Another effect of the `suffix` parameters passing a name instead of a value is that they actually *can* be used on the left side of an assignment:

```
def mymac(suffix a) =
  a := 5
enddef;

mymac(b);
```

does indeed assign the value `5` to the variable `b`.

2.1.3. `text` arguments

Arguments of `text` pass literal input text as a macro argument. That text fragment is not even limited to a single expression or statement.

```
def mymac (text a) =
    a = 5
enddef;

mymac(b);
```

Of all three possible argument types, this is the closest to a 'true' macro replacement. The argument's input text is processed exactly where it is called for in the replacement text:

```
def mymac (text a) =
    c = 5;
    a = c
enddef;

mymac(b);
```

Gives `b` the value of `5` as well as doing the same to `c`. This is another case where it is very important to remember that there is never a variable named `a`. The parameter name is just a placeholder that temporarily shields any preexisting variable named `a`.

Because of the rules for `text` parameters, this is also allowed:

```
def mymac (text a) =
    c = a c
enddef;

mymac(5; a =);
```

But constructions like that are not advised if you want you code to remain understandable.

So how does MetaPost decide that a `text` argument to macro has ended? When it sees an unmatched closing parenthesis.

Matching parentheses in the argument are counted, so

```
def mymac (text a) =
    a + 5
enddef;

mymac(b = (3 + 4));
```

works just fine and equates `b` to `12`. You cannot get an unmatched parentheses into the replacement text without some trickery (by defining a macro with a single parentheses as its replacement text and using that in the macro call instead of a literal `(`).

2.1.4. Multiple delimited arguments

So far, we have only dealt with single arguments, but it is also possible to have multiple arguments.

The formal syntax definition for delimited arguments is as follows:

```
def ⟨symbolic token⟩ ⟨delimited part⟩ =
    ⟨replacement text⟩
enddef;
⟨delimited part⟩ → ⟨empty⟩
     | ⟨delimited part⟩ (⟨parameter type⟩ ⟨parameter tokens⟩)
⟨parameter type⟩ → expr | suffix | text
⟨parameter tokens⟩ → ⟨symbolic token⟩ | ⟨parameter tokens⟩, ⟨symbolic token⟩
```

Reading a formal syntax like the one above takes a bit of practise, but converted to English it says that the delimited part of a macro definition header is a possibly empty sequence of items enclosed in parentheses. Each of these parenthesised sequences start with `expr`, `suffix` or `text` followed by a comma-separated list of at least one symbolic token.

Not expressed in the formal syntax is that whitespace is ignored except as a way to separate tokens, as normal in the MetaPost language.

Starting with some examples to illustrate the above syntax rules will hopefully help you in learning how to apply those rules. Here are some correct ways to start a definition:

```
def mymac =
def mymac(expr a) =
def mymac(expr a,b) =
def mymac(expr a)(expr b) =
def mymac(suffix a)(expr b, c) =
def mymac(suffix a)
         (expr b, c)
         ( suffix d )
         (text e) =
```

When a macro that is defined with multiple delimited arguments is called, specifying the internal delimiters is optional unless the argument is of `text` (this will be explained below). You can even insert delimiters that where not there in the definition or split the groups for `expr` and `suffix` parameters differently.

That last `mymac` macro above with the five delimited arguments in four groups can be called in various ways:

```
mymac (a,b,c,d,e);
mymac (a)(b)(c)(d)(e);
mymac (a,b)(c,d,e);
```

But all five arguments are required, and all of them must be part of delimited group. Here are some attempts that are *not* allowed:

```
mymac (a,b);          % bad 1
mymac a;              % bad 1
mymac (a)(b)()()(e);  % bad 2
mymac (a,b,,,e);      % bad 2
mymac (a,b)c(d)(e);   % bad 3
mymac (a,b)(c,d) e;   % bad 3
```

The ones marked `bad 1` are obviously illegal because some parameters are missing completely.

The ones marked `bad 2` are illegal because parameters cannot be empty. If you want to implement some sort of default behaviour, you will have to pass on a variable of a special type or value and deal with that as a special case in the replacement text. Just skipping the parameter is not allowed.

The ones marked `bad 3` are disallowed because all delimited arguments must be delimited.

But the rules above do not mean that you have to always specify all five arguments explicitly. MetaPost is expanding macros as it searches for the opening delimiters of the arguments of a macro call, so this is legal input:

```
def helper = (d)(e) enddef;
mymac (a,b)(c) helper;
```

You could even put all of the delimited arguments in separate other macro definitions.

Coming back to that last `bad 3` case for a bit, this is allowed

```
def e = (f) enddef;
mymac (a,b)(c,d) e;
```

Here, the macro `e` passes the replacement text of itself as the final argument to `mymac`.

But this is also possible:

```
def e = (f) enddef;
mymac (a,b)(c,d)(e);
```

And here, the macro `e` *itself* is the final argument to `mymac`. That is because once MetaPost has found a symbolic token that will become a macro argument, it will not expand it any further, so the macro itself is passed as the `text` argument instead of its replacement.

You need to remain aware of the fact that the expansion of macros only happens while MetaPost is actively looking for argument delimiters (opening parentheses and commas). You could do this:

```
def e = (f enddef;
mymac (a,b)(c,d) e);
```

or this:

```
def e = ,f enddef;
mymac (a,b)(c,d e);
```

or even this:

```
def c = f) enddef;
mymac (a,b)(c(d,e);
```

but not this:

```
def e = f) enddef;
mymac (a,b)(c,d)(e;
```

because in that last example MetaPost has already stopped looking for more arguments. It knows that there are only five arguments, so it does not bother to scan for a delimiter that would start a sixth argument.

2.1.5. Multiple and `text` arguments

Because of the nature of `text` arguments, they need an extra rule. It is possible to define a delimited macro with multiple text arguments like this:

```
def mymac (text e,f) =
    show e; show f;
enddef;
```

But this macro cannot be called without extra parentheses. With

```
mymac(g,h);
```

the replacement text of the `e` argument becomes `g,h` and MetaPost stops with an error about the missing argument `f`. If there are multiple `text` arguments or other arguments following a `text` argument, extra parentheses groups are required.

This is OK:

```
mymac (g)(h);
```

and this is also OK:

```
mymac (g; i; j; k; l)(h);
```

There is a simple rule to remember: always put text arguments in separate parentheses.

2.1.6. Undelimited arguments

Besides delimited arguments, macros can also have one undelimited argument. There can be only one of those and it has to be the last argument, but all three types are allowed, and there are some extra options as well. The syntax for undelimited arguments is as follows:

```
def ⟨symbolic token⟩ ⟨delimited part⟩ ⟨undelimited part⟩ =
    ⟨replacement text⟩
enddef;
⟨undelimited part⟩ → ⟨empty⟩
      | ⟨parameter type⟩ ⟨parameter⟩
      | ⟨precedence level⟩ ⟨parameter⟩
      | expr ⟨parameter⟩ of ⟨parameter⟩
⟨precedence level⟩ → primary | secondary | tertiary
```

(⟨delimited part⟩, ⟨parameter type⟩ and ⟨parameter⟩ have not changed and are omitted from the listing for brevity).

The three types of argument we have already discussed in the previous paragraph are the familiar cases. They are much like their delimited parts, except without delimiters. But there are a few extra notes:

- An `expr` argument grabs the longest expression it can find. When such a macro is called, MetaPost also allows an `=` or `:=` just before the argument.
- A `suffix` argument takes the longest suffix it can find. MetaPost allows that suffix to be enclosed in parentheses.
- A `text` argument stops at the next semicolon or `endgroup`.

The new options are these:

- `primary`, `secondary` and `tertiary` arguments are just like `expr`, except they grab a 'smaller' argument (a partial expression). This will be explained below.
- `expr` ⟨parameter⟩ `of` ⟨parameter⟩ is useful for creating macros that mimic the primitive operation `point t of p`. It grabs the longest syntactically correct ⟨expression⟩ `of` ⟨primary⟩ (see below for the explanation of ⟨expression⟩ and ⟨primary⟩). It is not possible to fake the `point of` primitive syntax in another way.

3. Operator definitions

3.1. About operators

Quite often, you will want a macro defined with an `expr` argument to take only a part of the following expression instead of the whole of it. That is where the `primary`, `secondary` and `tertiary` keywords come in, as they operate on *part* of an expression instead of on the whole of it.

But for better understanding, we need to back up a bit. Just like there are syntactic rules for macro definitions, there are formal rules for all other bits of a MetaPost program as well.

A MetaPost program is a sequence of statements. Most statements are internal commands, equations, or assignments. Expressions are part of equations and assignments. And expressions can be further subdivided into operators that work on variables or on further subdivisions of expressions.

There are a few other options for statements, and all the expression cases exist for all variable types (booleans, numerics, pairs, etc.). For brevity, I will concentrate on the numeric expressions to explain what is going on, and ignore all those other cases. In the syntax definition below, all ⟨...⟩ are extra rules that I skipped.

Here are the parts that are relevant right now:

⟨equation⟩ → ⟨expression⟩ = ⟨right-hand side⟩
⟨assignment⟩ → ⟨variable⟩ := ⟨right-hand side⟩
⟨right-hand side⟩ → ⟨expression⟩ | ⟨...⟩
⟨expression⟩ → ⟨numeric expression⟩ | ⟨...⟩
⟨numeric expression⟩ → ⟨numeric tertiary⟩
⟨numeric tertiary⟩ → ⟨numeric secondary⟩
| ⟨numeric tertiary⟩ + | - ⟨numeric secondary⟩
| ⟨...⟩
⟨numeric secondary⟩ → ⟨numeric primary⟩
| ⟨numeric secondary⟩ * | / ⟨numeric primary⟩
⟨numeric primary⟩ = ⟨numeric atom⟩
| ⟨numeric atom⟩ [⟨numeric expression⟩ , ⟨numeric expression⟩]
| ⟨...⟩
⟨numeric atom⟩ → ⟨numeric token⟩
| (⟨numeric expression⟩)
| ⟨...⟩

Working top-down, you can split a numeric expression in parts to the left and right of a plus or minus operation. Those left and right sides can each be split further into left and right sides of the multiply and divide operators. These sides can each be split even further into the arguments of the mediation operator.

In case you are wondering: the off by one between the left and the right is what makes operators in MetaPost left-associative.

Following those rules, let us investigate this expression:

```
4*(a+1) - b / 2[4,8]
```

Using the nomenclature from the official syntax, we can say that there are four primaries: `4`, `(a+1)`, `b` and `2[4,8]`. The two secondaries are `4*(a+1)` and `b / 2[4,8]`. The single tertiary is the whole `4*(a+1) - b / 2[4,8]`, which is also the whole expression.

The content of `(a+1)` is itself a nested expression, which can be subdivided using the same rules, but with a few shortcuts: `a` and `1` are the primaries. These are also the numeric secondaries, because there are no multiplication or division operations specified. The tertiary is `a+1`, which is also the expression value.

MetaPost supports four levels of operators: primary, secondary, tertiary, and expression. Not all value types have operators defined for all levels, though. That is why a ⟨numeric expression⟩ is the same as a ⟨numeric tertiary⟩. The rules for ⟨string expression⟩ look quite different:

⟨string expression⟩ → ⟨string tertiary⟩
| ⟨string expression⟩ `&` ⟨string tertiary⟩
⟨string tertiary⟩ → ⟨string secondary⟩
⟨string secondary⟩ → ⟨string primary⟩
⟨string primary⟩ → ⟨string variable⟩
| `char` ⟨numeric primary⟩
| ⟨...⟩

As you can see, strings only have operators on the primary and expression level. The operators for the other types are yet again different, but the expression structure stays the same.

When you are planning on defining operators yourself, it would be helpful to have a list of the current operators and their level. But alas, such a list typically does not exist because the built-in operators that are part of the bare MetaPost binary itself can (and usually will be) augmented by the MetaPost macro package you are using. If you are lucky, the macro package manual contains a concise list somewhere. If not, you will have to do trial and error until your definitions 'work' ...

3.2. Unary operator definitions

Getting back to macro definitions: `expr` grabs an ⟨... expression⟩. `primary` grabs a ⟨... primary⟩, `secondary` grabs a ⟨... secondary⟩ and `tertiary` grabs a ⟨... tertiary⟩.

It should now be clear that in:

```
def mymac primary arg =
 arg
enddef;
res = mymac 4*(a+1) - b / 2[4,8];
```

the argument is the 4.

In this version

```
def mymac secondary arg =
 arg
enddef;
res = mymac 4*(a+1) - b / 2[4,8];
```

the argument is the `4*(a+1)` (well, actually it is `4a+4`, because MetaPost interprets the partial expression before storing it in the parameter capsule).

And

```
def mymac expr arg =
 arg
enddef;
res = mymac 4*(a+1) - b / 2[4,8];
```

and

```
def mymac tertiary arg =
 arg
enddef;
res = mymac 4*(a+1) - b / 2[4,8];
```

both get the full expression as argument (actually `-0.08333b+4a+4`).

The net effect of using an undelimited `expr`, `primary`, `secondary` or `tertiary` is that you have created a new unary operator at that level. See further on for how to define binary operators for the top three levels. Primary operators in MetaPost are always unary operators.

3.3. Binary operator definitions

It is now time to learn about binary operator definitions.

⟨macro definition⟩ → ⟨macro heading⟩ = ⟨replacement text⟩ `enddef`
⟨macro heading⟩ → `primarydef` ⟨parameter⟩ ⟨symbolic token⟩ ⟨parameter⟩
| `secondarydef` ⟨parameter⟩ ⟨symbolic token⟩ ⟨parameter⟩
| `tertiarydef` ⟨parameter⟩ ⟨symbolic token⟩ ⟨parameter⟩

A macro defined using `primarydef` defines a new operator with a ⟨... secondary⟩ on the left and a ⟨.. primary⟩ on the right of its name. For a `secondarydef` that is a ⟨... tertiary⟩ on the left and a ⟨.. secondary⟩ on the right, and for a `tertiarydef` it is an ⟨... expression⟩ on the left and a ⟨... tertiary⟩ on the right.

This definition creates an alias for `*`:

```
primarydef a mult b =
  a * b
enddef;
```

The names of the primitives seem off by one compared to the keywords for undelimited `def` arguments we encountered earlier. But since MetaPost does not support binary primary operators, there would be only three possible levels anyway. You'll just have to get used to that. And at least `tertiarydef` sounds more natural than the fictitious '`exprdef`'. And remember, you can define unary binary operators with an undelimited `def` argument of type `primary`.

4. Variable definitions

Note: much of this section is copied and modified from my earlier paper.

The previous commands `def`, `primarydef`, `secondarydef`, and `tertiarydef` have one thing in common: they produce what MetaPost calls **spark**s. Effectively, you are defining a new 'command' instead of a 'variable'. But sometimes you may want a macro to behave more like a named variable. For that to work, the macro has to be what MetaPost calls a **tag**.

The restriction of `def` always producing a **spark** is why there is a dedicated command for creating macros that are actually **tag**s. That command is `vardef`.

Because `vardef` defines a new **tag** instead of a **spark**, the name that is being defined can still can be used in the middle of an unrelated compound variable name. Occasionally, you may want to define a macro with a name that would also make sense as a suffix to another variable. The MetaFont book highlights the example of `dir`. The variable macro `dir` is defined as a `vardef` precisely because doing it that way means it is still legal to have a pair variable named `p5dir`.

In simple uses, use of `vardef` is very similar to using `def`.

```
def stuff =
  fill unitsquare
enddef;
```

and

```
vardef stuff =
  fill unitsquare
enddef;
```

appear equivalent when they are executed. But there is a difference in execution: the `vardef` version actually expands into:

```
begingroup
  fill unitsquare
endgroup
```

The added grouping makes the macro expansion syntactically equivalent to an expression, which is important because it avoids confusing the MetaPost parser. We will get to the use of grouping later on.

Here is the formal definition of the syntax of `vardef`:

⟨macro definition⟩ → ⟨macro heading⟩ = ⟨replacement text⟩ `enddef`
⟨macro heading⟩ → `vardef` ⟨declared variable⟩ ⟨delimited part⟩ ⟨undelimited part⟩
| `vardef` ⟨declared variable⟩ `@#` ⟨delimited part⟩ ⟨undelimited part⟩

The ⟨delimited part⟩ and ⟨undelimited part⟩ are the same as before and are not repeated.

The use of ⟨declared variable⟩ instead of the ⟨symbolic token⟩ from the earlier definition commands is important: that is what makes this type of definition produce a **tag** instead of a **spark**. The ⟨declared variable⟩ is actually the syntax rule for a single item in a type declaration command (`boolean`, `path`, `picture`, etc.).

You can define segmented variable names, and even use collective subscripts:

```
vardef mymac[]arr =
  4
enddef;
```

defines all variables of the form `mymac[]arr` to be macros that expand into `begingroup 4 endgroup`.

The second option for the ⟨macro heading⟩ of a `vardef` syntax introduces an extra keyword `@#`. This is easiest to explain with an example from `plain.mp`:

```
vardef z@#=
  (x@#,y@#)
enddef;
```

This defines the variable macro `z`. What makes this definition of `z` special is that it now has a built-in parameter of type ⟨suffix⟩ that is named `@#`.

There is a subtle difference between this definition of `z` and the more naïve version:

```
vardef z suffix v =
  (x.v,y.v)
enddef;
```

The special token `@#` only applies to an immediately following suffix; the suffix that becomes the argument may not be enclosed in parentheses, unlike in the definition with an undelimited argument. This makes the special definition exceptionally useful for manipulating sub-variables (like `z` does).

The `@#` somewhat replaces `suffix v`. You can still define a macro like this:

```
vardef mymac @# suffix v =
  (x@#v,y@#v)
enddef;
```

but you always have to call that macro with parentheses around parameter `v`, otherwise the whole argument becomes part of the `@#` suffix:

```
origin = mymac1right;
```

will have `1right` as `@#` and `v` empty. With

```
origin = mymac1(right);
```

that does not happen, but then you could have equivalently defined `mymac` as

```
vardef mymac @# (suffix v) =
  (x@#v,y@#v)
enddef;
```

Finally every `vardef`, with or without the special `@#`, also has two other special implicit arguments that can be used anywhere in the ⟨replacement text⟩. The special argument name `@` returns the last segment of the name of the defined macro itself, and the special argument name `#@` returns the complement: all segments before that last one.

When is this useful? Look at this:

```
vardef p[]dir=
  (#@dx,#@dy)
enddef;
```

After this definition, `p5dir` expands into:

```
(p5dx,p5dy)
```

allowing you to write, for example:

```
p5dir = up;
```

to define the `dx` and `dy` subvariables, and query those values by

```
if p5dir = up: .... fi
```

which looks and feels a lot nicer than manipulating the `dx` and `dy` variables 'manually'.

In definitions like `p[]dir`, the special token `@` which expands into the macro 'name' is not very useful (we already know that it is `dir`), but keep in mind that **subscript**s can also be `vardef` macros themselves. Since `@` expands into the actual subscript in that case, it can then be used to differentiate between macro calls for specific subscripts by using a numerical comparison, like this:

```
vardef a[] =
  if odd @: message("odd")
  else:     message("even")
  fi
enddef;
a1;  % prints "odd"
a20; % prints "even"
end.
```

In cases where the expansion of one of the special tokens (`#@`, `@`, or `@#`) is not known beforehand to be numeric, to test its value you can use the `str` command instead to force an expression with type ⟨string⟩ (this makes most sense with implicit suffixes):

```
vardef a@# =
  if str @# = "o": message("odd")
  else:            message("even")
  fi
enddef;
a.o; % prints "odd"
a.e; % prints "even"
```

A warning about using `vardef`: because the result of the `vardef` is a macro, it only works as the *last* typed segment in a complete variable name. After the definition above, you can **not** now add another suffix:

```
pair p[]dir.target; % WRONG!
```

This is disallowed, because that set of variables would actually be inaccessible.

Because of how the MetaPost parser works, the `target` part of the name would always become a suffix argument to the `p[]dir` macro. In this case, as the macro is defined without a suffix argument, the result would be a syntax error. However, if you really want to write things like `p5dir.target`, you could extend the definition of `p[]dir` to also accept the undelimited suffix `@#`, and then process the `target` within the macro expansion.

In some cases, the implicit extra grouping added by `vardef` is an impediment, and it would be better to use `def`. But sometimes that extra grouping level can

be a bonus as well: it allows trivial macro definitions that need that grouping to be a bit shorter. Still, that is only a very minor advantage, and the MetaPost manual explicitly warns against abusing `vardef` just for grouping.

5. Grouping

The sequence `begingroup`... `endgroup` can be used as a standalone statement. The formal definition of ⟨statement⟩ looks like this:

⟨statement⟩ → ⟨equation⟩ | ⟨assignment⟩ | ⟨declaration⟩
| ⟨definition⟩ | ⟨title⟩ | ⟨command⟩ | ⟨empty⟩
| `begingroup` ⟨statement list⟩ ⟨statement⟩ `endgroup`

That last statement inside the group should be a valid statement on it own, but it can be empty.

Grouping in MetaPost is a bit unusual (yet another way in which MetaPost is unusual!) in that the `begingroup`... `endgroup` block is not only usable as a list of ⟨statement⟩s grouped together. It can also be used as an ⟨expression⟩. And when viewed as an expression (which is usually the case for `vardef` macro expansions, but you can also write explicit group blocks in the middle of an equation, or as the body of any type of macro), all the statements in the group are executed as normal, but the last expression inside the group (which could be empty) is taken as the value to use for the expression outside of the group. And precisely that oddity of grouping is what makes `vardef` definitions syntactically equivalent to variables.

Formally, all of the expression syntaxes also have an extra `begingroup` block. For example, ⟨numeric expression⟩ also has:

⟨numeric atom⟩ → ⟨numeric token⟩
| `(` ⟨numeric expression⟩ `)`
| `begingroup` ⟨statement list⟩ ⟨numeric expression⟩ `endgroup`
| ⟨...⟩

and likewise for all other expression types: at the bottom level, there is an `begingroup` ... `endgroup` that is equivalent with a delimited group like `(` ⟨numeric expression⟩ `)`.

The statements in the ⟨statement list⟩ are executed, but not seen by the expression parser.

The extra grouping usually will not matter, but it means you cannot do things like

```
vardef stuff =
  fill unitsquare
enddef;
stuff withcolor green;
```

which makes sense once you realise that `vardef` is supposed to equate to a variable. If we assume for a moment that there was instead a normal path variable named `stuff`, the statement would look like this:

```
fill stuff withcolor green;
```

and indeed, after adjusting the `vardef` to

```
vardef stuff =
  unitsquare % earlier 'fill' deleted
enddef;
```

that works just fine. This is a silly example of course, but the point to remember is that the last line in a `begingroup` ... `endgroup` should produce a valid expression (which may be empty).

The main point of `begingroup` ... `endgroup` is so that you can save and temporarily redefine variables and internals. But it creates only an implicit grouping, nothing is automatically saved. If you want to save an outside variable or internal, you have to explicitly use `save` or `interim`.

The above rules for the final parts inside of a group block make grouping behave similar to an anonymous function call with one return value. Or as a named function, when using `vardef` or the result of a `def` with a `begingroup` ... `endgroup` block around the whole replacement text.

Additionally, because the expression parser does not 'see' the ⟨statement list⟩, you can do complicated things right in the middle of an equation. The plain MetaPost macro named `hide` makes use of that:

```
def hide(text t) = gobble begingroup t; endgroup enddef;
def gobble primary g = enddef;
```

(this is the example definition from the MetaFont book, the actual definition is trickier).

The `begingroup` ... `endgroup` block inside `hide` always results in an empty expression because of the explicit `;` at the end. But an empty expression is still an expression, so that is why the `gobble` macro is needed to 'eat' that empty expression.

One final note about `vardef`: the addition of `begingroup` ... `endgroup` around the ⟨replacement text⟩ is literal. If you wanted to, you could write a `vardef` including `endgroup` ... `begingroup` to temporarily escape to the outer group. But of course then the expansion would not be a valid ⟨... expression⟩ any more, so you could not use that macro in the middle of an expression any more.

6. Replacement text details

A ⟨replacement text⟩ is stored for later use without any expansion at definition time. In almost all cases, the meaning of the symbolic tokens will be looked up and applied at expansion time. However, some tokens that may occur inside the ⟨replacement text⟩ have to be interpreted by the program right away to avoid internal confusion:

- `def`, `vardef`, `primarydef`, `secondarydef` and `tertiarydef` are the start of an embedded definition.
- `enddef` ends the ⟨replacement text⟩ unless it matches an embedded definition that started in the previous rule.
- Each ⟨symbolic token⟩ that stands for a macro parameter is changed into a placeholder for that parameter, for later substitution at replacement time.
- `quote` prevents any of the previous rules applying to the next token. After afterward, the `quote` token itself is removed from the replacement.
- In all the above cases, the check is made for the meaning of the token, not its literal representation. In other words, prior use of `let` can alter the list of 'keywords'.

The preceding rules mean that

```
def bfour =
  def b = 4 enddef
enddef;
```

is allowed. Any subsequent use of `bfour` in the program expands into a definition that makes `b` be a macro that expands to the value `4`. But

```
def defbfour =
  def b = 4
enddef;
```

will fail, because the definition of `defbfour` does not end. The `enddef` stops the embedded definition of `b`.

To get around that, you can write

```
def defbfour =
  quote def b = 4
enddef;
```

which is a valid definition of `defbfour`. Now you will have to use `defbfour enddef;` when using `defbfour` later, of course. Otherwise the embedded definition of `b` never ends.

The existence of `quote` allows some special syntaxes. With the above definition, you could specify

```
defbfour *4 enddef;
```

which would define `b` to be a macro with replacement text `4*4`. Admittedly, this is not very useful. But I want to document everything related to definitions, and the use of `quote` cannot be omitted.

7. Formal definition syntax

⟨macro definition⟩ → ⟨macro heading⟩ `=` ⟨replacement text⟩ `enddef`
⟨macro heading⟩ → `def` ⟨symbolic token⟩ ⟨delimited part⟩ ⟨undelimited part⟩
| `vardef` ⟨declared variable⟩ ⟨delimited part⟩ ⟨undelimited part⟩
| `vardef` ⟨declared variable⟩ `@#` ⟨delimited part⟩ ⟨undelimited part⟩
| ⟨binary def⟩ ⟨parameter⟩ ⟨symbolic token⟩ ⟨parameter⟩
⟨delimited part⟩ → ⟨empty⟩
| ⟨delimited part⟩ `(`⟨parameter type⟩ ⟨parameter tokens⟩`)`
⟨parameter type⟩ → `expr` | `suffix` | `text`
⟨parameter tokens⟩ → ⟨parameter⟩ | ⟨parameter tokens⟩`,` ⟨parameter⟩
⟨parameter⟩ → ⟨symbolic token⟩
⟨undelimited part⟩ → ⟨empty⟩
| ⟨parameter type⟩ ⟨parameter⟩
| ⟨precedence level⟩ ⟨parameter⟩
| `expr` ⟨parameter⟩ `of` ⟨parameter⟩
⟨precedence level⟩ → `primary` | `secondary` | `tertiary`
⟨binary def⟩ → `primarydef` | `secondarydef` | `tertiarydef`

8. Final words

You now know all about how to define your own MetaPost macros, in theory. But the best way of learning is by doing and making mistakes, and that is definitely the case here as well. When I started using MetaPost in earnest, at first nothing I tried seemed to work. Remembering my own initial frustrations about anything more than trivial use of the MetaPost programming language is what prompted me to write these papers. I hope they will be helpful to you.

Reference

1. HOEKWATER, Taco. Sparks, tags, suffixes and subscripts. In: *Proceedings of the 13th ConTEXt Meeting*. Bassenge-Boirs, Belgium: ConTEXt Group, 2019. Draft available at `https://meeting.contextgarden.net/2019/talks/05-mpost-vars/05-mpost-vars.pdf`.

Definice v MetaPostu

Článek popisuje všechna syntaktická pravidla pro definice v MetaPostu. Obsahuje množství dobrých i špatných příkladů definic.

Klíčová slova: MetaPost, definice

Mělo by to fungovat X

Peter Wilson

Abstrakt

Článek přináší ukázku využití programovacího jazyka Lua ke zjištění počtu jednotlivých znaků v daném textovém souboru. Druhá část článku se zabývá možností změny rozvržení stránky v LaTeXu v souvislosti se sazbou vícestránkového seznamu obrázků.

Klíčová slova: Počet slov, četnost znaků, Lua, textový soubor, rozvržení strany, prostředí `quote`, prostředí `quotation`, balíček changepage, třída memoir, seznam obrázků

> His eye, which scornfully glisters like fire,
> Shows his hot courage and his high desire.
>
> *Venus and Adonis*
> William Shakespeare

Cílem tohoto seriálu je ukázat čtenáři krátké kousky kódu, které mohou vyřešit některé z jeho problémů. Doufám, že situaci ještě více nezkomplikuji v důsledku mých chyb. Opravy, poznámky a návrhy na změny budou vždy vítány.

> Words are wise men's counters, they do but reckon
> with them, but they are the money of fools.
>
> *Leviathan*
> Thomas Hobbes

Počítání slov

Některé druhy publikací mohou být omezeny předepsaným počtem slov. Pak vyvstává otázka, jakým (nejlepším) způsobem můžeme slova spočítat. Toto detailně vysvětlují Otázky a odpovědi [1]. Nicméně je také možné jednoduše spočítat slova na jedné stránce rukopisu a výsledek vynásobit počtem stran. Tuto techniku, nazývanou *casting off*, používají designéři knih a vydavatelé při zpracování knihy, viz například [3, kap. 8]. Vydavatelé ani tak nepotřebují vědět přesný počet slov v knize, ale zajímá je počet stránek knihy po vysázení.

Z anglického originálu *Glisterings* [7] přeložil Jan Šustek. Všechny díly *Glisterings* byly také sepsány do knihy, kterou je možné zakoupit prostřednictvím TUG na stránce `https://tug.org/store/glisterings/` (pozn. redakce)

 doi: 10.5300/2020-3-4/168

Když píšete diplomovou práci, je možné, že jste limitováni předem stanoveným rozsahem počtu slov. Přesto je pravděpodobné, že nikdo nebude kontrolovat přesný počet slov, pokud nebude na první pohled vidět nesoulad zadaného rozsahu a délky vaší diplomové práce. Otázkou ale je, co se počítá jako jedno slovo. Za kolik slov se počítá tabulka nebo obrázek? Když se používají písma v různých velikostech, mají slova stejnou váhu? A co složená slova, poznámky pod čarou, rovnice, webové adresy, verše? Za kolik slov se mají počítat?

V době, kdy čtete tyto řádky, je již LuaTEX na světě a možná jste jej i použili. V době, kdy jsem tyto řádky psal, byl LuaTEX ještě ve vývoji a sám jsem jej nezkoušel. Ale předpokládal jsem, že jazyk Lua [2] bude dostupný na všech platformách, kde je TEX, a proto jsem se jej rozhodl použít pro počítání slov.

Měl jsem to štěstí, že jsem mohl pracovat s ruční sazbou, podobně jako to dělával Gutenberg v 15. století. Na rozdíl od digitální sazby, kde je k dispozici neomezený počet jednotlivých znaků, je v ruční sazbě počet znaků jasně omezen. Pokud ve fontu máme pouze 23 tiskařských liter „e“, pak lze najednou vysázet text, v němž se znak „e“ nevyskytuje více než 23krát. Je proto důležité vědět, kolik jednotlivých znaků je třeba k vysázení jedné stránky textu. Kdysi jsem chtěl vysázet báseň ze 16. století (pouze dva verše na stránku) v konkrétním fontu, ale nemohl jsem, protože bych potřeboval jednu literu „h“ navíc (text obsahoval staroanglická slova thee, thou, thine, slova s příponou -eth apod.).

Pracuji v systému Linux, pro který existují programy na zjištění počtu slov a celkového počtu znaků v daném souboru. Dá se očekávat, že podobné programy existují i pro jiné operační systémy. Já jsem však potřeboval program, který zjistí počty jednotlivých znaků (počet znaků „A“, počet znaků „a“ atd.), proto jsem si jej zkusil vytvořit v jazyce Lua.

```
#!/usr/local/bin/lua5.1
local BUFSIZE = 2^13
local f = io.input(arg[1])  -- otevření vstupního souboru
local cc = 0       -- počet všech znaků
local lc = 0       -- počet řádků
local wc = 0       -- počet slov
local ct = {}      -- počty jednotlivých znaků
for i = 33,126 do  -- inicializace tabulky
  ct[i] = 0
end
local tc = 0       -- počet znaků bez znaků konce řádku
while true do
  -- načtení bloku textu
  local lines, rest = f:read(BUFSIZE, "*line")
  if not lines then break end
  if rest then
```

```
17      lines = lines .. rest  .. "\n" end
18    cc = cc + #lines
19    -- zjištění počtu slov v bloku
20    local _, t = string.gsub(lines, "%S+", "")
21    wc = wc + t
22    -- zjištění počtu konců řádku v bloku
23    _, t = string.gsub(lines, "\n", "\n")
24    lc = lc + t
25    -- tabulka počtu jednotlivých znaků
26    local K
27    for i = 1, string.len(lines) do
28      K = string.byte(lines,i)
29      if K >= 33 and K <= 126 then
30        ct[K] = ct[K] + 1
31      end
32    end
33  end
34  -- odstraní příponu vstupního souboru a nahradí ji příponou .gwc
35  base, ext = string.match(arg[1], "(%w+)%.(%w+)")
36  ofile = base .. ".gwc"
37  tc = cc - lc
38  io.output(ofile)
39  io.write("Informace o souboru ", arg[1], "\n")
40  io.write("", "pocet radku = ", lc, "\n",
41               "pocet slov  = ", wc, "\n",
42               "pocet znaku = ", tc, "\n\n")
43  io.write("Jednotlive znaky\n")
44  for k,v in pairs(ct) do
45    if v > 0 then
46      print(string.char(k),v)
47      io.write("  ", string.char(k), "      ",
48               string.format("%4d",v), "\n")
49    end
50  end
51  print("Vystup ulozen do souboru ", ofile)
```

Pokud program uložíme do souboru gwc.lua, pak počty znaků (například v souboru soubor.txt) získáme jeho spuštěním z příkazového řádku:[1]

```
52 gwc.lua soubor.txt
```

Tento program jako slovo počítá posloupnost tisknutelných znaků ukončenou jednou nebo více mezerami nebo koncem řádku. Program se zajímá pouze o znaky odpovídající literám ve fontu, který jsem měl k dispozici. Naštěstí v angličtině se stačí omezit na množinu tisknutelných znaků v ASCII. V případě jiných jazyků je třeba program rozšířit.[2] Část kódu byla převzata z [2, s. 198], kde je také kód detailně popsán.

> Change is not made without inconvenience,
> even from worse to better.
>
> *A Dictionary of the English Language: Preface*
> Samuel Johnson

Změna rozvržení stránky

Čas od času uživatelé položí otázku „Jak mohu změnit rozvržení konkrétní stránky?“, kde pod rozvržením mají na mysli rozměry nebo umístění sazby, různá záhlaví či zápatí.

Tvar stránky

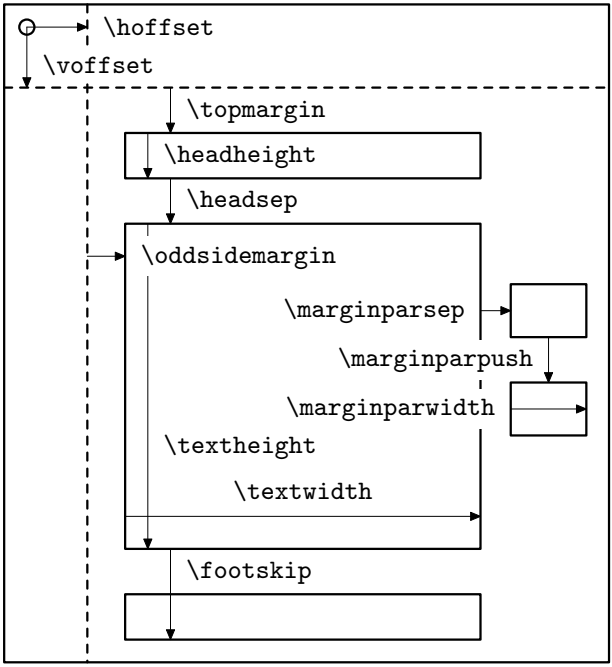
V TEXu můžete udělat mnoho věcí, ale co nemůžete,[3] je změnit šířku textu uvnitř odstavce. Například pokud by měl mít na první stránce text šířku 15 cm, zatímco na druhé stránce šířku 12 cm, pak odstavec, který začíná na první straně, bude pokračovat se šířkou 15 cm i na druhé straně. To je způsobeno tím, že TEX nejdříve interně vysází odstavec podle aktuálně nastavené šířky textu. Poté určí, zda se odstavec vysází na aktuální stranu, nebo zda je třeba stránku zlomit. Pokud ke zlomu stránky dojde, vysází se část odstavce na první stranu a zbytek odstavce na druhou stranu, odstavec však již má řádky vysázené na původní šířku.

Na obrázku 1 jsou uvedeny parametry, které se používají při nastavení rozvržení stránky. Kroužkem je označen Knuthův bod o souřadnicích (1 in, 1 in) od levého horního okraje stránky.

[1] V systému Linux musíme soubor gwc.lua nastavit jako spustitelný. Na řádku 1 pak musíme mít správně uvedenu cestu k interpretu jazyka Lua. Další možností je zavolat přímo interpret jazyka Lua, tj. zavolat `lua gwc.lua soubor.txt` (pozn. překl.)

[2] Pro soubor v češtině nebo slovenštině můžeme program jednoduše upravit, pokud soubor bude v některém osmibitovém kódování. V tom případě je třeba na řádcích 8 a 29 namísto hodnoty `126` psát hodnotu `255`. Kódování výstupního souboru soubor.gwc bude stejné jako kódování vstupního souboru soubor.txt. (pozn. překl.)

[3] Čtenář, který nemá v TEXu rád negativní odpovědi, si může přečíst článek [5], kde je podrobně rozebrán a vyřešen obecnější problém. Pokud se na stránce nevyskytují pružné vertikální mezery, je možné problém vyřešit vhodným použitím primitivu `\parshape`. (pozn. překl.)



Obrázek 1: LaTeXové parametry rozvržení liché (pravé) stránky

Ke změně výšky textového bloku na konkrétní stránce lze použít LaTeXového makra `\enlargethispage`. Toto makro má jeden délkový parametr, jehož hodnota se připočte k registru `\textheight`, a tato změna je platná lokálně pouze pro aktuální stranu – kladná hodnota parametru zvýší hodnotu `\textheight`, záporná ji sníží. Změna se projeví na spodní hraně textového bloku, horní hrana zůstane na stejném místě.

Prostředí `quote` a `quotation` dočasně změní velikost okrajů a šířku textu. Obecněji můžete šířku textu dočasně změnit pomocí prostředí `adjustwidth` z balíčku changepage. [6]

> Prostředí `adjustwidth` se volá se dvěma délkovými parametry. Uvnitř prostředí jsou levý a pravý okraj zvětšeny o hodnoty příslušných parametrů. Například v tomto odstavci bylo použito
> `\begin{adjustwidth}{3cm}{1cm}`
> na začátku odstavce a podobně na konci odstavce bylo použito
> `\end{adjustwidth}`

Aktuální stránka bude vysázena s hodnotami parametrů platnými v okamžiku, kdy je na stránku vložena první položka (například znak nebo box). Parametry, které se změní až poté, budou platné od následující stránky. Je však možné změnit parametry po vysázení jedné stránky před vložením první položky na následující stránku. Toho se využívá, když se v LaTeXu přepíná z jednosloupcové sazby do dvousloupcové a naopak. LaTeXová makra při tom přepočítají parametry rozvržení stránky. Obecný postup je, že se aktuální stránka ukončí a vysází, změní se parametry rozvržení, nastaví se nový počet sloupců sazby a tyto sloupce se již sázejí s novými hodnotami parametrů. Pokud pracujeme v jednosloupcovém dokumentu, postup je tento:

```
53 \clearpage
54 % změna parametrů
55 \onecolumn
```

Při změně parametrů na řádku 54 můžete použít také makra z LaTeXového balíčku changepage.

Aby bylo zřejmé, co makro `\onecolumn` dělá, uvádím zde jeho definici:

```
56 \def\onecolumn{%
57   \clearpage
58   \global\columnwidth\textwidth
59   \global\hsize\columnwidth
60   \global\linewidth\columnwidth
61   \global\@twocolumnfalse
62   \col@number \@ne
63   \@floatplacement}
```

Definice makra `\twocolumn` je složitější. Mimo jiné má toto makro jeden nepovinný parametr[4], který však nemá vliv na rozložení sazby.

Jako příklad si ukážeme, jak je možné zvětšit nebo zmenšit výšku a šířku textového bloku na některé skupině stránek.

```
64 \newcommand*{\addtotextheightwidth}[2]{%
65   \clearpage
66   \addtolength{\textheight}{#1}
67   \addtolength{\textwidth}{#2}
68   \onecolumn}
```

Až budeme chtít změnu provést, například pro stránky s přílohami, zavoláme

```
69 \addtotextheightweight{-1cm}{1cm}
```

[4]Obsah parametru je vysázen v jednosloupcovém režimu před započetím dvousloupcové sazby. (pozn. překl.)

Záhlaví a zápatí

Další typ dotazu na rozvržení stránky, který jsem viděl, se týkal umístění slova „strana“ do záhlaví nad čísla stran ve vícestránkovém obsahu nebo seznamu obrázků. Řekněme, že chceme seznam obrázků vysázet tak, že v záhlaví stránky se seznamem bude slovo „obrázek“ zarovnané doleva a slovo „strana“ zarovnané doprava. Toto by se mělo opakovat na všech stranách seznamu obrázků. Dále chceme, aby stránka se seznamem obrázků měla své číslo zarovnané na střed v zápatí, podobně jako v LaTeXovém stylu stránky `plain`. Podobné požadavky budeme mít pro vysázení obsahu a seznamu tabulek. Zde si však ukážeme pouze sazbu seznamu obrázků.

I když ke změně stylu stránky můžete použít také balíček fancyhdr [4], zde si ukážeme třídu memoir, která nabízí podobné možnosti.

Třída memoir umožňuje definovat libovolné množství stylů. Potřebujeme vytvořit styl stránky pro navazující stránky v seznamu obrázků (a podobné styly pro obsah a pro seznam tabulek). Styl stránky pojmenujeme `lof`, záhlaví nastavíme tak, že vlevo bude slovo „obrázek“ a vpravo slovo „strana“, zatímco v zápatí uprostřed bude číslo stránky.

```
\makepagestyle{lof}
  \makeevenhead{lof}{obrázek}{}{strana}
  \makeoddhead{lof}{obrázek}{}{strana}
  \makeevenfoot{lof}{}{\thepage}{}
  \makeoddfoot{lof}{}{\thepage}{}
```

Musíme zajistit, že se styl `lof` nastaví během sazby seznamu obrázků. Toho docílíme tak, že nastavení stylu přidáme k definici makra `\listoffigures`. Ve třídě memoir je pro tyto účely definováno makro `\addtodef`. Makro má tři parametry – prvním je název upravovaného makra, druhým je TeXový kód, který se přidá na začátek definice, a třetím TeXový kód, který se přidá na konec definice.

```
\addtodef{\listoffigures}{%
  \clearpage\pagestyle{lof}}{}
```

Ve třídě memoir je definováno makro, které se zavolá těsně před vysázením nadpisu seznamu obrázků, a makro, které se zavolá těsně po vysázení. Předefinováním těchto maker nastavíme, že první stránka seznamu obrázků bude mít prázdné záhlaví a že se výše uvedený text záhlaví vysází pod nadpis.

```
\renewcommand*{\afterloftitle}{%
  \thispagestyle{plain}
  \par\nobreak
  {\normalfont\normalsize obrázek\hfill strana}
  \par\nobreak}
```

Nastavení sazby obsahu a seznamu tabulek je prakticky identické, pouze patřičným způsobem změníme názvy použitých maker.

Je třeba si ještě dát pozor na to, že po vysázení seznamu obrázků zůstane nastavený styl `lof`. Proto je nutné po zavolání makra `\listoffigures` nastavit zpět původní styl. Předpokládejme, že základní styl dokumentu se jmenuje `mujstyl`. Pak můžeme psát

```
\documentclass[...]{memoir}
%% nastavení stylu mujstyl
%% úpravy \listoffigures apod. (řádky 70 až 81)
\pagestyle{mujstyl}

\begin{document}
%% úvodní strany, předmluva atd.
\tableofcontents
\clearpage\pagestyle{mujstyl}
\listoffigures
\clearpage\pagestyle{mujstyl}
\listoftables
\clearpage\pagestyle{mujstyl}
...
\end{document}
```

Seznam literatury

[1] Robin Fairbairns. *texfaq. A compilation of Frequently Asked Questions with answers*. Version 3.28. 10. čvn. 2014. URL: `https://www.ctan.org/pkg/texfaq`.

[2] Roberto Ierusalimschy. *Programming in Lua*. 2. vyd. Rio de Janeiro, 2006.

[3] Ruari McLean. *The Thames and Hudson Manual of Typography*. Thames and Hudson, 1980.

[4] Pieter van Oostrum. *fancyhdr. Extensive control of page headers and footers in LaTeX 2ε*. Version 4.0.1. 28. led. 2021. URL: `https://www.ctan.org/pkg/fancyhdr`.

[5] Jan Šustek. „Sazba odstavců do textových oblastí“. In: *Zpravodaj CSTUG* 19.3 (2009), s. 124–137.

[6] Peter Wilson. *changepage. Margin adjustment and detection of odd/even pages*. Version 1.0c. 20. říj. 2009. URL: `https://www.ctan.org/pkg/changepage`.

[7] Peter Wilson. „Glisterings“. In: *TUGboat* 31.1 (2010), s. 90–93.

[8] Peter Wilson. *memoir. Typeset fiction, non-fiction and mathematical books*. Version 3.7n. 4. říj. 2020. URL: `https://www.ctan.org/pkg/memoir`.

Summary: It Might Work X

This paper shows an example how to use Lua program to find the number of particular characters in a given text file. The second part of the paper shows ways to change the page layout in LaTeX, with applications for typesetting a multi-page list of figures.

Keywords: Word counting, character frequency, Lua, text file, page layout, `quote` environment, `quotation` environment, changepage package, memoir class, list of figures

Peter Wilson, herries.press@earthlink.net
18912 8th Ave. SW
Normandy Park, WA 98166 USA

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Ilustrace na obálce:	Karel Horák
Počet výtisků:	275
Uzávěrka:	19. 3. 2021
Odpovědný redaktor:	Jan Šustek
Redakční rada:	Pavel Haluza, Lukáš Novotný, Vít Novotný, Michal Růžička a Jan Šustek (šéfredaktor)
Vědecká rada:	Ján Buša (předseda), Jiří Demel, Jaromír Kuben (zástupce předsedy), Jiří Rybička a Petr Sojka
Technická redakce:	Vít Novotný
Evidenční číslo MK:	E 7629
Adresa:	CSTUG, Nejedlého 373/1, 638 00 Brno
Email:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD a DVD

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz`

www server sdružení:
`https://www.cstug.cz`

CONTENTS