Zpravodaj Československého sdružení uživatelů TeXu

David Antoš; Petr Sojka

Generování vzorů pomocí knihovny PatLib a programu OPatGen

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 12 (2002), No. 1, 3–12

Persistent URL: http://dml.cz/dmlcz/149876

Generování vzorů pomocí knihovny PatLib a programu OPatGen

David Antoš, Petr Sojka

Článek[1] popisuje techniku generování vzorů jako prostředek pro získávání informace z rozsáhlých dat. Typickou aplikací této techniky je vytvoření časově i prostorově velmi efektivního algoritmu dělení slov ze seznamu již rozdělených slov. Doposud chyběl generátor vzorů dělení pro UNICODE (pro systém Ω) a rozšíření dosud užívaného programu PatGen, omezeného osmibitovým ASCII, nebylo již nadále únosné. Proto vyvíjíme knihovnu PatLib pro obecnou manipulaci se vzory a na ní postavený generátor vzorů dělení slov OPatGen. Popíšeme architekturu tohoto systému. Vzory lze použít i pro rozpoznávání hranic složených slov, proto zmíníme návrhy na rozšíření následníků TEXu o klasifikované dělení s více typy dělicích bodů a o automatické potlačování ligatur na švech složených slov.

Klíčová slova: PatLib, OPatGen, soutěživé vzory, rozpoznávání vzorů, generování vzorů, dělení (složených) slov, PatGen, Ω.

Úvod

"Go forth and make masterpieces of hyphenation patterns . . . "
—Yannis Haralambous [5]

Téměř vše lze považovat za symbol a ze symbolů můžeme kombinovat *vzory*. Vzory jsou často zjevením a popisem vyšších zákonitostí. Hofstadter [9] považuje rozpoznávání vzorů za centrální pojem inteligence a v tomto smyslu jsou zaměřeny i mnohé inteligenční testy.

Technika vzorů je efektivním prostředkem pro extrakci informací a rozpoznávání v datech. Na vzorech je v TEXu [12] postaveno elegantní a na jazyku nezávislé řešení pro kvalitní dělení slov. Tento efektivní algoritmus pak našel cestu i do téměř všech následně vznikajících sázecích systémů včetně těch komerčních a dle našeho mínění si zaslouží ještě mnohem větší pozornost pro své široké možnosti použití.

Generování vzorů pro dělení pomocí programu PatGen [14] nedostačuje plně současným potřebám a pro jeho širší použití je třeba po dvaceti letech od

[1]Předběžná verze tohoto článku byla publikována v [3]. Výzkum byl prováděn v rámci výzkumného záměru CEZ:J07/98:143300003.

jeho vzniku provést mnohá zobecnění. Vznikl systém Ω [6] mj. s cílem umožnit přímou práci se slovy všech jazyků v UNICODE (universální vzory dělení, kontextové substituce, překladové procesy Ω – ΩTP). Tyto nové výzvy a v dalším textu naznačené analýzy nás dovedly k závěru, že nejlepším řešením bude úplná reimplementace PatGENu.

V článku popíšeme základní principy techniky vzorů, jakým způsobem jsou vzory generovány a jak je potom TEX používá. Dále se budeme věnovat architektuře systému OPatGen a knihovny PatLib a jejímu použití jako universální knihovny pro manipulaci se vzory. Závěrem zmíníme možné aplikace technologie založené na zpracování vzorů.

Vzory

> *Middle English* patron *'something serving as a model', from Old French. The change in sense is from the idea of a patron giving an example to be copied. Metathesis in the second syllable occured in the 16th century. By 1700* patron *ceased to be used on things, and the two forms became differentiated in sense.*
> — Origin of word pattern: [4]

Vzory slouží pro rozpoznávání „zajímavých míst" v datech. Zajímavým místem může být hranice znaků, kde je v daném slovu povoleno dělit, hranice voda/les na leteckém snímku krajiny a podobně. Zajímavé místo lze rozpoznat pomocí jeho kontextu. Vzory jsou podslova dané množiny slov, mezi jejichž symboly je vyznačena informace o zajímavých místech.

Informace o těchto bodech je v principu dvojího druhu: jedna říká, že dané místo *je* místem zájmu, druhá, že *není*. Typickou reprezentací bývají přirozená čísla, lichá pro ano, sudá pro ne. Tedy vzory máme *pokrývací* a *zabraňující*. Lze také použít další speciální symboly, jako například tečku značící začátek nebo konec slova. Například české dělení obsahuje vzory `i1h`, `i3hl.` a `i2hl.`. Použití vzorů je toto: pro všechna podslova daného slova se najdou všechny odpovídající vzory. Tedy slovu `cihla` odpovídají vzory `i1h`, `i2hl` z naší množiny, takže máme `ci2hla`. Vzory se při použití *přebíjejí*, můžeme říci, že *soutěží*, a výsledkem je maximum z hodnot odpovídajících pozici mezi znaky. Za chvíli si vysvětlíme, jak se vzory generují. To také lépe objasní, proč se používají zrovna takto.

Podrobný popis použití vzorů lze nalézt v [12, příloha H]. Laskavého čtenáře se zájmem o podrobnější a formálnější informace o vzorech odkazujeme na články [19, 10, 1], o nástroje na práci s konečnými automaty pak na [15, 11, 16, 13, 2].

Generování vzorů

"An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil's behaviour."
—Alan Turing, [21]

Požadujeme, aby vzory pokrývaly co nejvíce míst zájmu, tedy aby měly *maximální úplnost*. Zároveň chceme, aby chybovaly co možná nejméně (*maximální přesnost*) a přitom byly *prostorově minimální*, tedy tak malé, jak je to jen možné. Z těchto protichůdných požadavků je nejméně bolestný požadavek minimality, ze stoprocentní přesnosti však slevovat nechceme. Iterativní metodou lze dosáhnout překvapivě dobrých výsledků a komprimovat informace ze vstupních dat do množiny vzorů. Popišme, jak takové generování probíhá.

Potřebujeme rozsáhlou množinu vstupních dat, ve které jsou označena místa zájmu, v našem případě se bude jednat o slova přirozeného jazyka a v nich označená místa, kde je dovoleno dělit.

Nyní budeme opakovaně procházet soubor vstupních dat. Jednotlivé průchody nazvěme *úrovněmi*. V lichých úrovních generujeme pokrývací vzory, v sudých zabraňující.

V každé úrovni zvolíme *pravidlo*, pomocí něhož vybíráme *kandidáty* na vzory. V našem případě pravidlo může mít tvar „k-znakový podřetězec slova obsahující dělicí bod". Vybereme kandidáty na vzory a uložíme je do vhodné datové struktury. Ne každý kandidát je ovšem dobrý vzor, proto potřebujeme *pravidlo pro výběr vzorů*. Obvykle je rozumné projít všechny kandidáty na vzory a vyzkoušet jejich funkci (přitom se používají i vzory z předchozích úrovní, je o práci všech dosud vytvořených vzorů) na slovech ze vstupních dat. Přitom spočteme, kolikrát kandidát pracoval dobře a kolikrát chyboval. Pravidlem pak může být funkce nad těmito hodnotami porovnávaná se zadanou hodnotou, tedy klasické lineární prahování.

Kandidáty, které jsme v předchozím procesu označili za dobré, zařadíme mezi vzory. Tyto vzory ovšem mohou (a většinou také budou) stále chybovat. Takže budeme pokračovat další úrovní, tentokrát sudou, v níž budeme vytvářet zabraňující vzory. Dobrým vzorem určité úrovně je kandidát, který opravuje chyby vzorů z nižších úrovní. Podrobněji řečeno, v pokrývací (liché) úrovni je dobrý kandidát, který najde správný dělící bod. V zabraňující úrovni je dobrý ten, který opraví bod označený chybně předchozí množinou vzorů jako místo dělení.

Tímto způsobem v principu pracuje program PatGen, s poměrně pevně danými pravidly pro výběr vzorů (lineární prahování). Vzory dělení slov pro TeX existují pro několik desítek jazyků, většina generována PatGenem ze slovníku rozdělených slov. Musíme poznamenat, že se často také postupuje tak, že se vzory vytvářejí částečně ručně (buď pro bootstrapping, nebo ručně).

Jaká je úspěšnost této techniky? Ze slovníku velikosti několika MB lze vytvořit vzory velikosti řádově desítek KB pokrývající nad 98 % dělicích bodů a s chybovostí pod 0,1 %. Četné experimenty ukázaly, že se vystačí často se čtyřmi úrovněmi [20]. Pomocí vhodných technik (bootstrapping, stratifikace) a strategií nastavení parametrů pro lineární prahování bylo ukázáno [20, 17, 18], jak se dají generované vzory optimalizovat. Příklady statistik z generování variant českých vzorů dělení jsou v tabulkách 1, 2 a 3. Pro nastavování parametrů nemáme žádné teoretické podklady, generování vzorů je zatím do značné míry závislé na umění a zkušenostech.

Tabulka 1: Standardní generování českých vzorů s parametry Lianga

úroveň	délka	param	% dobré	% špatné	# vzorů	velikost
1	2–3	1 2 20	96,95	14,97	+ 855	
2	3–4	2 1 8	94,33	0,47	+1706	
3	4–5	1 4 7	98,28	0,56	+1033	
4	5–6	3 2 1	98,22	0,01	+2028	32 kB

Tabulka 2: Standardní generování českých vzorů s optimalizací na velikost vzorů

úroveň	délka	param	% dobré	% špatné	# vzorů	velikost
1	1–3	1 2 20	97,41	23,23	+ 605	
2	2–4	2 1 8	85,98	0,31	+ 904	
3	3–5	1 4 7	98,40	0,78	+1267	
4	4–6	3 2 1	98,26	0,01	+1665	23 kB

Tabulka 3: Standardní generování českých vzorů s optimalizací pokrytí dělicích bodů

úroveň	délka	param	% dobré	% špatné	# vzorů	velikost
1	1–3	1 5 1	95,43	6,84	+2261	
2	1–3	1 5 1	95,84	1,17	+1051	
3	2–5	1 3 1	99,69	1,24	+3255	
4	2–5	1 3 1	99,63	0,09	+1672	40 kB

Proč PatGen nestačí?

"The road to wisdom?
Well it's plain and simple to express:
Err and err and err again
but less and less and less."
—Piet Hein [8]

Program PatGen má několik vážných omezení. Je to monolit strukturovaného kódu, který, ačkoli je velmi dobře dokumentován (WEB, tj. dokumentovaný Pascal), není snadné upravovat. Obsahuje radikální optimalizace, které umožnily, že se proces generování vzorů dělení vešel do paměti PDP-10. Tyto optimalizace značně snižují srozumitelnost kódu, ztěžujíce tak možnosti úprav.

Datové struktury PatGenu jsou postaveny na osmibitový ASCII kód, přitom rozšíření na UNICODE prakticky nepřichází v úvahu. Maximální počet úrovní je devět. V průběhu výběru kandidátů na vzory lze současně vybírat pouze kandidáty shodných délek. Data jsou interně ukládána do statických struktur, pokud během generování paměť dojde, je nutno zasáhnout do zdrojového kódu a rekompilovat.

Samozřejmě lze PatGen využít pro generování vzorů pro jiné jevy než dělení slov, ovšem pouze tak, že se daný problém na dělení slov převede. To může být značně netriviální a navíc lze takto řešit pouze problémy s dostatečně malou abecedou, přibližně pod 240 symbolů. PatGen totiž některé ASCII znaky používá jako výstupní symboly.

PatLib a OPatGen

"My library was dukedom large enough."
—Shakespeare, The Tempest (1611), act 1, sc. 2 l. 109

Rozhodli jsme se tedy PatGen zobecnit. Implementovali jsme knihovnu PatLib (PATtern LIbrary) pro práci se vzory. Generátor vzorů dělení v UNICODE, který využívá služeb knihovny, jsme nazvali OPatGen.

Pro implementaci jsme zvolili méně obvyklou kombinaci v CWEBu psaného C++ z důvodů portability a efektivity a udržování kvalitní dokumentace. Navíc šablony v C++ umožňují odložit přesnou specifikaci typu na co možná nejpozději, což se během analýzy ukázalo jako velká výhoda.

Knihovna PatLib se skládá z manipulátoru s konečným jazykem a generátoru vzorů. Vrstva pro uložení konečných slov je implementována pomocí komprimované varianty datové struktury trie.

Manipulátor jako svou interní abecedu používá čísla pro zachování rozumné efektivity, je však schopen pracovat i s jinými objekty. Manipulátor se vzory je v principu konečným automatem s výstupem omezeným na konečný jazyk.

Poskytuje základní služby typu „vlož vzor“, „dej výstup vzoru“, „odstraň vzor“ a dále umí vydat po jednotlivých vzorech celý uložený jazyk. Výstupní informací vzoru může být také libovolný objekt.

Protože mezi nejčastěji zjišťované charakteristiky kandidátů na vzory patří dvojice čísel udávající, kolikrát kandidát pracuje správně a kolikrát způsobuje chybu, připravili jsme i službu zajišťující pohodlnou práci s těmito údaji.

Generátor implementuje výše popsanou (s PatGENem kompatibilní) strategii generování, tedy opakovaně prochází vstupní data a sbírá statistiky o kandidátech na vzory. Po jednotlivých průchodech vybírá technikou lineárního prahování vzory z kandidátů.

Tím jsme oddělili sémantiku ukládaných informací od jejich reprezentace. Nezáleží tedy na tom, s jakými daty aplikace pracuje. Pokud pro aplikaci vyhovuje námi implementovaná strategie generování, pak stačí doplnit vrstvu rozhraní na soubory. Pokud chceme provést ve strategii generování menší změny, lze předefinovat příslušné metody tříd generátoru a upravit parametry šablon. Pro vytvoření od základu jiné strategie by mohla posloužit alespoň vrstva pro uložení jazyka.

Samozřejmě za větší obecnost a flexibilitu platíme snížením výkonu. Zatímco třeba výstupem vzoru PatGENu je odkaz do hashovací tabulky obsahující dvojici ⟨číslo úrovně, pozice⟩, my musíme mít jako výstupní abecedu třídu s kopírovacím konstruktorem. Naše první zkušenosti ukazují, že na reálných datech (slovníku 170 tisíc slov) je reimplementace PatGENu postavená na knihovně PatLib nejvýše šestkrát pomalejší. Nutno podotknout, že kód současné verze ještě není optimalizován, v této fázi jsme se snažili zachovat „konceptuální čistotu“.

Na OPatGen samotný už zbývá jen realizovat konkrétní rozhraní na soubory. Situaci nám poněkud komplikuje požadavek co možná největší zpětné kompatibility s PatGENem. I v OPatGENu tedy najdeme možnost použití „translate“ souboru. OPatGen může pracovat ve dvou režimech, osmibitovém ASCII nebo plném UTF-8 UNICODE. Rozhodli jsme se manipulaci s UNICODE zvládat ve vlastní režii. Tím dosáhneme větší nezávislosti na systému a možnosti implementace plného UTF-8. Většina běžných operačních systémů dnes totiž podporuje UTF-8 jen omezeně, často pouze do 16 bitů.

Součástí distribuce produktu je také uživatelský manuál, ve kterém jsou zvláště vyznačeny (drobné) rozdíly mezi PatGENem a OPatGENem, aby byl přechod mezi těmito programy co nejsnazší.

Aplikace techniky vzorů u následníků TEXu

"But at least I can point out a minor weakness of TEX's algorithm: all possible hyphenations have the same penalty. This might be ok for english, but for languages like German that have a lot of composite words there should be the ability to assign lower penalties between parts of a composite i.e. Um-brechen should be favored against Umbre-chen."
—Florian Hars [7]

Dělení slov

Připomeňme si, kdy TEX slova dělí. V následujícím popisu zanedbáváme detaily, které nejsou podstatné pro následující úvahy. Pro přesný popis odkazujeme na [12] nebo články [17, 18].

Algoritmus zlomu odstavce má nejvýše tři průchody. V prvním průchodu se slova nedělí a hledá se řešení, při kterém mají všechny řádky hodnotu badness menší nebo rovnu `\pretolerance`. Pokud takové řešení neexistuje, nastupuje druhý průchod.

Ve druhém průchodu se do slov odstavce přidají symboly pro možné dělicí body. Pak se vyhodnocují všechny zlomy, pro něž platí, že všechny řádky mají badness nejvýše `\tolerance`. Pokud neexistuje přijatelné řešení, ve třetím průchodu se navíc povolí roztažitelné mezislovní mezery.

Tento algoritmus má však nepříjemné omezení pro jazyky, v nich jsou častá složená slova. Švy složených slov jsou typografy považována za místa pro dělení vhodnější, ostatní místa jsou vhodná méně. Bohužel TEX nedovoluje klasifikovat dělicí body, pro libovolné místo vybrané jako možný dělicí bod ve druhém průchodu algoritmu zlomu odstavce má pouze jedinou možnou `\hyphenpenalty`.

Možným řešením by bylo vytvoření dvojice vzorů, jedna sada vzorů pro švy složených slov, druhá pro všechny dělicí body. Navrhujeme zavedení penalty za dělení slova na švu, `\compoundhyphenpenalty`. Ta by byla menší než `\hyphenpenalty` a tím by bylo preferováno dělení na švech slov. To samozřejmě vyžaduje zásah do algoritmu zlomu odstavce, tedy se může týkat pouze některého z následníků TEXu.

Rozšíření o tuto klasifikaci dělení přináší otázku, kdy a jak je během zlomu odstavce provádět. Jednou možností je nahradit současný průchod, kdy se dělení provádí, průchodem, ve kterém se zjistí dělicí místa na švech slov a nastaví se jim `\compoundhyphenpenalty`. Dále se zjistí ostatní vhodné dělicí body a těm, které ještě nemají nastavenou penaltu (tj. nejsou na švu slova), se nastaví hodnota `\hyphenpenalty`.

Jinou možností je místo současného průchodu s dělením slov implementovat průchody dva. V prvním by se provedlo pouze dělení na švech slov a pokud by zlom odstavce byl dostatečně kvalitní, ponechal by se. Pokud ne, provedlo by se i dělení v dalších možných místech (s `\hyphenpenalty`) a odstavec by se lámal znovu.

Navíc znalost švů složených slov je výhodná i z dalšího důvodu. Typografická pravidla požadují, aby se na švech nevyskytovaly ligatury. Tedy například slovo šéflékař má být správně vysázeno šéflékař. Bohužel to je nutno TEXu sdělit ručně, proto jsem poslední slovo předchozí věty musel psát `šéf\hskip0ptlékař`.

Bylo by vhodné, aby se všechna slova ze vstupního proudu otestovala na švy složených slov a vstupní proud se příslušně upravil. To samozřejmě přináší další otázky, jako například, zda nevypustit první průchod zlomu odstavce a nezačít rovnou s povoleným dělením na švech slov.

Překladové procesy Ω
Systém Ω umožňuje téměř v jakémkoliv okamžiku zpracování textu ve svém zažívacím traktu aplikovat *překladový proces* (ΩTP, Omega Translation Process). Ten je dosud implementován pomocí, zjednodušeně řečeno, substitucí regulárních výrazů. Tato realizace dostačuje u jednoduchých zobrazení, pro složitější (například dělení slov) není dostatečně efektivní. Proto se pro složitá mapování (dělení slov, aplikace ligatur v daném jazyce, spelling či dokonce grammar checker, kontextové zpracování znaků v arabštině) nabízí pro generování a uložení těchto informací použití techniky vzorů. Knihovna PatLib by pak hrála v efektivní implementaci těchto nástrojů klíčovou roli.

Shrnutí

'I když se neprosadím, chtěl bych věřit, že bude někdo pokračovat v tom, co jsem započal. Ne bezprostředně, ale člověk není sám ve víře v opatrnost.'
—Vincent van Gogh

V článku jsme popsali techniku vzorů, možnosti jejího využití a návrh knihovny pro práci se vzory.

Zdrojové kódy knihovny PatLib a generátoru OPatGen v jejich aktuálních vývojových verzích spolu s dokumentací a dalšími materiály naleznete na adrese `http://www.fi.muni.cz/~xantos/PATLIB`. Doufáme ve využití PatLibu v širokém spektru aplikací, od implementace OPatGenu, přes implementaci překladových procesů sázecího systému Ω, po aplikace v oblastech zpracování přirozeného jazyka či grafiky.

Odkazy

[1] Cezar Câmpeanu, Nicolae Sânteau, and Sheng Yu. Minimal cover-automata for finite languages. In Champarnaud et al. [2], pages 43–56.

[2] Jean-Marc Champarnaud, Denis Maurel, and Djelloul Ziadi, editors. *Automata Implementation, Third International Workshop on Implementing Automata, WIA '98*, Berlin, Heidelberg, 1999. Springer-Verlag.

[3] David Antoš and Petr Sojka. Generování vzorů dělení slov v UNICODE. Str. 23–32, Brno, Czech Republic, Feb 2001. Konvoj.

[4] Patrick Hanks, editor. *The New Oxford Dictionary of English*. Oxford University Press, Oxford, 1998.

[5] Yannis Haralambous. A Small Tutorial on the Multilingual Features of PATGEN2. in electronic form, available from CTAN as `info/patgen2.tutorial`, January 1994.

[6] Yannis Haralambous and John Plaice. Methods for Processing Languages with Omega. In *Proceedings of the Second International Symposium on Multilingual Information Processing, Tsukuba, Japan*, 1997. available as `http://genepi.louis-jean.com/omega/tsukuba-methods97.pdf`.

[7] Florian Hars. Typo-l email discussion list, 4 January 1999.

[8] Piet Hein. *Grooks*. MIT Press, Cambridge, Massachusetts, 1966.

[9] Douglas R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, 1979.

[10] Tao Jiang, Arto Salomaa, Kai Salomaa, and Sheng Yu. Decision problems for patterns. *Journal of Computer and Systems Sciences*, 50(1):53–63, 1995.

[11] Lauri Karttunen, Tamás Gaál, and André Kempe. Xerox finite-state tool. Technical report, Xerox research Centre Europe, Grenoble, June 1997. `http://www.xrce.xerox.com/research/mltt/fssoft/docs/fst-97/xfst97.html`.

[12] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[13] András Kornai. *Extended Finite State Models of Language*. Cambridge University Press, 1999.

[14] Franklin M. Liang and Peter Breitenlohner. `PAT`tern `GEN`eration program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.3 from web2c distribution on CTAN, 1999.

[15] Mehryar Mohri, Fernando C.N. Pereira, and Michael D. Riley. FSM Library — General-purpose finite-state machine software tools, 1998. `http://www.research.att.com/sw/tools/fsm/`.

[16] Emmanuel Roche and Yves Schabes. *Finite-State Language Processing*. MIT Press, 1997.

[17] Petr Sojka. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297, 1995.

[18] Petr Sojka. Hyphenation on Demand. *TUGboat*, 20(3):241–247, 1999.

[19] Petr Sojka. Competing Patterns for Language Engineering. Lecture Notes in Artificial Intelligence LNCS/LNAI 1902, pages 157–162, Brno, Czech Republic, Sep 2000. Springer-Verlag.

[20] Petr Sojka and Pavel Ševeček. Hyphenation in TEX — Quo Vadis? *TUGboat*, 16(3):280–289, 1995.

[21] Alan Turing. Computing machinery and intelligence. *Mind*, (59):433–460, 1950.

Summary: Pattern Generation using PatLib Library and Program OPatGen

Paper describes technique of competing patterns as a method for data mining and effective storage. Development of time- and space-effective hyphenation algorithm from already hyphenated word list is a typical application.

The program PatGen, being nearly twenty years old, doesn't suit today's needs (limitation to eight-bit encodings, monolithic, hard to maintain code, etc.). A new pattern generator, OPatGen, suitable for system Ω, has been designed and implemented from scratch in object-oriented manner. An architecture of OPatGen is outlined. It is based on generic library PatLib for pattern handling.

Possible applications of the pattern technology are listed (multi-level and compound word hyphenation, Thai segmentation, optical character recognition).

David Antoš, Petr Sojka
Fakulta informatiky Masarykovy university, Brno
{xantos|sojka}@informatics.muni.cz