Stronniczo o stronach kodowych

Bogusław Jackowski

TeX został zaprojektowany w taki sposób, by możliwa była praca z różnorodnymi schematami kodowania znaków – tak zwanymi stronami kodowymi. Przez dłuższy czas implementatorzy najpopularniejszej ostatnio „linii" TeX-owej, wykorzystującej system WEB2C, opierali się wprowadzeniu możliwości zmieniania stron kodowych. Czemu – trudno orzec (pewne sugestie zawiera odpowiedź do ćwiczenia 6.1 podręcznika *The TeXbook*).

Starałem się włączyć do dyskusji, upierając się, że implementacje TeX-a:

(a) albo powinny być dostarczane w postaci *łatwo kompilowalnej* (wówczas strony kodowe sam sobie mogę podmienić);

(b) albo – jeśli interpret\ifWJM ato\else e\fi r TeX-a jest dostarczany w postaci binarnej – zmiana stron kodowych *musi być „wystawiona na zewnątrz"*.

Po długotrwałej wymianie argumentów implementatorzy TeX-a zaakceptowali ostatnio wariant (b). Wydaje mi się, że warto przypomnieć wyjaśnienia, jakie przeszło rok temu przytoczyłem na rzecz zasadności takiego podejścia.

Dziedzictwo języka Pascal

D. E. Knuth zaprogramował TeX-a w języku Pascal. Do obyczaju programowania w tym języku należy rekompilacja programu przy zmianie parametrów – wszystkie parametry TeX-owe są w oryginale tak właśnie ustawiane. Potrzebujemy większej pamięci – zmieniamy jedną liczbę w źródle i kompilujemy TeX-a, potrzebujemy więcej miejsca na fonty – zmieniamy inną liczbę – i znów kompilujemy TeX-a.

Podkreślić należy, że modyfikacje tego typu *nie są zmianą* TeX-a sensu stricto – autor wyraźnie zezwala na tego rodzaju interwencje (wszystkie cytaty w języku angielskim pochodzą ze źródła TeX-a, tj. z pliku TeX.WEB):

> *[...] parameters can be changed at compile time to extend or reduce TeX's capacity.*

Fragment źródła TeX-a, noszący tytuł *System-dependent changes*, zaczyna się od stwierdzenia:

> *This section should be replaced, if necessary, by any special modifications of the program that are necessary to make TeX work at a particular installation.*

D. E. Knuth wyobrażał sobie, że w jakiś taki sposób użytkownicy będą rozwiązywać problem stron kodowych.

Nie wchodząc w szczegóły – źródło TeX-a zawiera tablice o nazwach xchr i xord, za pomocą których definiuje się sposób przekodowania znaków czytanych i pisanych przez TeX-a. TeX ma *na mocy założenia* możliwość przekodowania dowolnego kodu wejściowego na dowolny kod wewnętrzny (odpowiadający układowi znaków w foncie) i odwrotnie.

D. E. Knuth wiedział bowiem, że ASCII nie musi być jedynym systemem kodowania znaków:

> *In order to make TeX readily portable to a wide variety of computers, all of its input text is converted to an internal eight-bit code that includes standard ASCII, the "American Standard Code for Information Interchange." This conversion is done immediately when each character is read in. Conversely, characters are converted from ASCII to the user's external representation just before they are output to a text file.*

Nie przypuszczał jednak, jak sądzę, jakie wieżobablowe zamieszanie ze stronami kodowymi czeka świat, o czym zdają się świadczyć beztroskie słowa w innym miejscu:

> *For example, in countries with an alphabet of more than 26 letters, it is usually best to map the additional letters into codes less than 32.*

Niestety, Pascal nie doczekał się ani powszechnego uznania, ani dobrej standaryzacji. W związku z tym kompilacja TeX-a na różnych platformach nie jest taką łatwą rzeczą, jak zakładał D. E. Knuth. Chwała twórcom systemu WEB2C, który być może uchronił TeX-a przed zagładą...

Ale w związku z tym te implementacje, które nie pozwalają na łatwą rekompilację zmodyfikowanych źródeł, powinny przynajmniej pozwalać na modyfikację *wszystkich* parametrów, które można by zmodyfikować mając dostępne źródło, w szczególności na modyfikację tablic xord i xchr.

Wiem, że tak dobrze być nie może, że „wystawienie na zewnątrz” wszystkich parametrów jest z praktycznego punktu widzenia niemożliwe. Ale im więcej, tym lepiej, a możliwości zmiany *ważnych* parametrów *nie wolno* blokować. Tablice xord i xchr uważam za bardzo ważny fragment kodu TEX-owego (z punktu widzenia przenośności TEX-a), a tym samym uniemożliwienie modyfikacji tych tablic uważam za niedopuszczalne.

Paradygmat INITEX-a

Temat ten pozornie nie wiąże ze stronami kodowymi – a jednak i owszem, jak się niebawem okaże.

TEX był tworzony w czasach, gdy komputery nie miały tak rozbuchanych (częstokroć nie wiadomo po co) pamięci. Stąd koncepcja „jednego TEX-a w dwóch osobach”:

> *[...] there are two versions of TEX. One, called INITEX, has to be run first; it initializes everything from scratch, without reading a format file, and it has the capability of dumping a format file. The other one is called VIRTEX; it is a “virgin” program that needs to input a format file in order to get started. VIRTEX typically has more memory capacity than INITEX, because it does not need the space consumed by the auxiliary hyphenation tables and the numerous calls on* primitive*, etc.*

To dość egzotyczne rozwiązanie miało źródła czysto TEXnologiczne, natomiast konsekwencje są dużo bardziej brzemienne w skutki, niż można by się spodziewać. Chodzi mianowicie o wzorce dzielenia wyrazów:

> *[Data structures] for TEX's hyphenation algorithm is built from a sequence of patterns following a* \patterns *specification. Such a specification is allowed only in INITEX, since the extra memory for auxiliary tables and for the initialization program itself would only clutter up the production version of TEX with a lot of deadwood.*

Zatem wzorce dzielenia wyrazów mogą być przetwarzane jedynie w wersji INITEX-owej. Teoretycznie nie ma przeciwwskazań, żeby na odpowiednio zasobnych w pamięć komputerach TEX nie mógł być zarazem INITEX-em i VIRTEX-em. Dziś tak się TEX-a implementuje: VIRTEX to po prostu TEX, a INITEX to TEX wywołany ze specjalną opcją. Niemniej jednak wiele fragmentów kodu opiera się na założeniu, że wzorce zostały przetworzone, zapamiętane i że nie będą już zmieniane:

> *Each time* \patterns *appears, it contributes further patterns to the future trie, which will be built only when hyphenation is attempted or when a format file is dumped. The boolean variable* trie_not_ready *will change to* false *when the trie is compressed; this will disable further patterns.*

a jedna z podpowiedzi emitowanych przez TEX-a głosi, że

> *All patterns must be given before typesetting begins.*

Nie znam implementacji TEX-a, która by zezwalała na modyfikację zestawu wzorców (tzn. na wykonanie instrukcji \pattern) w trakcie składania dokumentu. Obawiam się, że odpowiednia zmiana TEX-a jest niezwykle skomplikowana, jeśli nie wręcz niewykonalna, podobnie zresztą jak wszystkie inne „proste” usprawnienia, proponowane co jakiś czas.

Odwrotna strona zwierciadła

No i wracamy do naszych baranów, czyli do stron kodowych: wzorce dzielenia wyrazów są zapamiętywane w takim układzie znaków (wewnętrznym), jaki obowiązywał w momencie wykonania instrukcji \pattern. Kodowanie wewnętrzne zostaje niejako „zamrożone” w strukturze danych opisującej wzorce.

Jeśli w trakcie składania użyjemy fontu o innym układzie znaków niż w trakcie tworzenia formatu INITEX-em, na przykład jeśli w trakcie tworzenia formatu obowiązywał układ EC (Cork Encoding), a do składania użyliśmy fontu w egzotycznym i rzadko spotykanym układzie CP 1250, to TEX-owy mechanizm dzielenia wyrazów po prostu *zadziała niepoprawnie.* Za pomocą żonglerki znakami aktywnymi można doprowadzić do pojawienia się właściwych znaków na właściwych miejscach, ale dzielenie wyrazów trzeba zablokować – to jedyne, co można zrobić na poziomie instrukcji TEX-owych.

Znaki aktywne – przez team LATEX-owy uznane, jak się wydaje, za panaceum – są jedynie półśrodkiem, a nawet ćwierćśrodkiem, dotyczą bowiem tylko „jednej strony zwierciadła”, tzn. zewnętrznego kodowania znaków, i rozwiązują problem połowicznie, gdyż znaki aktywne to nie to samo co litery (znaki o kodzie 11), ze wszystkimi licznymi konsekwencjami tej różnicy.

A zwierciadło ma swoją „odwrotną stronę", to znaczy wewnętrzne kodowanie znaków, ściśle związane z układem znaków w foncie, a ten wewnętrzny układ jest z kolei ściśle związany z wzorcami dzielenia wyrazów.

Wypływa stąd ważki wniosek:

> *Jeśli nie można zmienić „w locie" zestawu wzorców, to zmiana kodowania „w locie" pomoże w ogólnym wypadku tyle, co umarłemu kadzidło (też w ogólnym przypadku).*

Można próbować oszukać TEX-a, wmawiając mu, że polski tekst złożony fontami w układzie Mazovii i w układzie CP 1250 to w istocie tekst w dwóch różnych językach, wymagających różnych wzorców dzielenia wyrazów. Podczas tworzenia formatu należałoby zaaplikować TEX-owi różne polskie zestawy wzorców, odpowiednio do użytych układów znaków w fontach. Takie podejście mogłoby działać. Trudno jednak uznać je za praktyczne rozwiązanie problemu, zwłaszcza że omyłkowe niedopasowanie wzorów dzielenia do aktualnego kodowania spowoduje, że TEX po cichutku będzie dzielił wyrazy w sposób niepoprawny.

Warto też zdawać sobie sprawę z tego, że w pliku DVI obowiązuje kodowanie wewnętrzne. To pozornie trywialne spostrzeżenie ma swoje dość głębokie konsekwencje. Oto dwa przykłady ilustrujące zagadnienie:

- Jeśli na przykład używamy instrukcji `\special`, w których pojawiają się polskie napisy (powiedzmy komentarze), to sposób kodowania polskich znaków w tych napisach będzie odpowiadał przyjętemu układowi znaków w foncie i z reguły będzie się różnił od kodowania znaków na wejściu i znaków wysyłanych na ekran, do pliku LOG i do innych plików tekstowych, tworzonych przez TEX-a;
- Jeśli wykorzystujemy program DVITYPE (standardowy program środowiska TEX-owego) do wyekstrahowania tekstu z pliku DVI, to musimy pamiętać, że w utworzonym tekście zobaczymy znaczki-maczki, bo wyekstrahowany tekst zostanie zapisany w kodzie wewnętrznym.

Proponowane czasami rozwiązanie, polegające na „zewnętrznym filtrowaniu" wejścia i wyjścia TEX-a, oznacza, że *wszystko* co pisze i czyta TEX, a więc i plik DVI (a jeśli chodzi o pliki tekstowe, to TEX przecież może czytać coś, co przed chwilą zapisał), musiałoby być przetwarzane przez bardzo skomplikowany system „otoczkowy", który oprócz baczenia na wszystkie bajty sypiące się z TEX-a musiałby jeszcze dokonywać analizy pliku DVI, przynajmniej takiej, jakiej dokonuje DVITYPE.

Rzecz jest teoretycznie możliwa do zrobienia (oczywiście pod wielozadaniowym systemem operacyjnym), ale bynajmniej niełatwa, i w dodatku – ze względu na problem układu znaków „zamrożonego" we wzorcach dzielenia – mało skuteczna.

Mniejszości TEX-owe

Najczęściej wysuwanym argumentem jest to, że koncepcja stron kodowych gryzie się z koncepcją LATEX2ε. To, że w LATEX-u można za pomocą karkołomnych sztuczek (czyt. znaków aktywnych) problem trochę podłatać, sprawy nie załatwia. Jakkolwiek bardzo rozpowszechniony LATEX nie jest jedynym formatem. I tu wkracza na scenę etyka – czy lekceważyć „mniejszości TEX-owe"? Nie należy chyba likwidować wyższych uczelni z tego powodu, że znikoma mniejszość społeczeństwa studiuje!

Argumentem wysuwanym przeciwko możliwości zmieniania stron kodowych jest to, że wówczas wymiana elektroniczna dokumentów znacznie się komplikuje, podczas gdy „wymienialność" dokumentów podnoszona jest jako jeden z najmocniejszych atutów LATEX-a.

Zgoda. Ale przecież TEX nie służy jedynie do elektronicznej wymiany dokumentów. Służy też do składania, i to ładnego składania, skomplikowanego, nietypowego, itd. Mniejsza o procenty (p. wyżej) – skoro TEX się do takich zastosowań nadaje, to nie należy ich utrącać, zwłaszcza jeśli są to zastosowania *zgodne* ze specyfikacją. Możliwość korzystania ze zmiennych stron kodowych w niektórych wypadkach, szczególnie w tych skomplikowanych i nietypowych, jest sprawą pierwszorzędnej wagi.

Epilog

Ku mej radości, ostatecznie zwyciężyło stronnictwo konfigurowalnych stron kodowych. To daje TEX-owi szansę przeżycia ładnych paru lat więcej. I o to chodzi.

◇ Bogusław Jackowski
B.Jackowski@GUST.org.pl