Pakiet makr TAP

Bogusław Jackowski

Tabele a...

Tabele a TeX. TeX został wyposażony w rozbudowaną maszynerię umożliwiającą składanie nawet niezwykle trudnych tabel, jednakże notacja TeX-owa w tym przypadku jest w stanie zniechęcić nawet najbardziej zawziętych TeX-owców. Mam tu oczywiście na myśli makra dostępne w formacie plain. W trochę lepszej sytuacji są LaTeX-owcy, ale i oni mieliby na co ponarzekać.

Próby „obłaskawienia TeX-a” niezbyt się udawały, bo proponowane zestawy makr na ogół nosiły silne piętno sposobu myślenia autora, co utrudniało ich użytkowanie przez TeX-owców o innym podejściu.

Okazało się jednak, że od dawna istnieje zestaw makr o nazwie TABLES, trafiający – moim zdaniem znakomicie – w tak zwane przeciętne zapotrzebowania. Autorem tego zestawu jest Michael J. Ferguson z Kanady. Ferguson wymyślił prostą notację, wystarczającą do *prostego i czytelnego* zapisu tabel w przeważającej liczbie przypadków, a przecież i tak w nietypowych, skomplikowanych przypadkach nie da się uniknąć komplikacji kodu TeX-owego.

Tabele a PostScript. Pakiet Fergusona ma jednak dość istotną wadę – nie wykorzystuje możliwości oferowanych przez PostScript, takich jak rysowanie ukośnych linii czy też wypełnianie tła wybranych rubryk kolorem, podczas gdy praktyka złośliwie podsuwa tego rodzaju zadania. To skłoniło moich kolegów i mnie do przebudowania pakietu TABLES w taki sposób, żeby nie tracąc (w miarę możności) nic z Fergusonowego podejścia dołożyć parę notacyjnych „gwizdków i dzwonków” oraz zapewnić łącze z PostScript-em. W ten sposób powstał pakiet makr o nazwie TAP. Znaczenie nazwy łatwo rozszyfrować: „T” to TeX, „A” – AWK, „P” – PostScript. Czemu TeX i PostScript, to oczywiste. Ale skąd tu AWK?

Tabele a AWK. Wiele edytorów tekstu pozwala na rysowanie tabel w łatwy sposób. Na przykład shareware'owy edytor `QEdit` (godny polecenia także dla wielu innych zalet) umożliwia osobom nawet o znikomej wprawie tworzenie dość skomplikowanych tabel.

Każdy, kto miał do czynienia z tabelami TeX-owymi, zdaje sobie na pewno sprawę, że złożenie za pomocą TeX-a takiej na przykład tabeli

<table>
<tr><td>rubryka 1</td><td>rubryka 2</td><td>rubryka 3</td><td>rubryka 4</td></tr>
<tr><td colspan="2">rubryka 1 i 2</td><td colspan="2">rubryka 3 i 4</td></tr>
<tr><td></td><td colspan="2">rubryka 2 i 3</td><td rowspan="2">wiersz 3 i 4</td></tr>
<tr><td colspan="3">rubryka 1, 2, 3</td></tr>
</table>

wymaga od TeX-owca sporej wprawy i sporego zachodu, podczas gdy „wklepanie” takiej tabeli za pomocą np. `QEdit`-a to kwestia kilkunastu – a przy pewnej wprawie kilku – minut. Chciałoby się móc przetłumaczyć postać tekstową na postać TeX-ową. Okazuje się, że AWK idealnie nadaje się do sporządzenia stosownego translatora. Program jest wprawdzie – jak na AWK-a – stosunkowo długi (przeszło dwieście linii kodu), ale wynik automatycznego przekładu jest zadowalający:

<table>
<tr><td>rubryka 1</td><td>rubryka 2</td><td>rubryka 3</td><td>rubryka 4</td></tr>
<tr><td colspan="2">rubryka 1 i 2</td><td colspan="2">rubryka 3 i 4</td></tr>
<tr><td></td><td colspan="2">rubryka 2 i 3</td><td rowspan="2">wiersz 3 i 4</td></tr>
<tr><td colspan="3">rubryka 1, 2, 3</td></tr>
</table>

Program ten nosi nazwę TAPCV (*TAP converter*).

Tabelologia stosowana

To, czego dowiedzieliśmy się dotychczas, wystarcza, byśmy mogli przystąpić do praktycznych ćwiczeń w składaniu tabel z użyciem pakietu TAP.

Proste tabele. Weźmy na pierwszy ogień prościutką tabelkę, dającą się przedstawić za pomocą znaków ASCII następująco:

1	2	3	4
11	22	33	44

Z zapisu ASCII nie wynika, czy liczby mają być umieszczone w osi rubryki czy też mają być wyrównane do prawej cyfry. Załóżmy na razie, że chodzi o centrowanie.

Program TEX-owy opisujący skład takiej tabeli wygląda całkiem przejrzyście (za chwilę szczegółowe objaśnienia):

```
1. \begintable
2. \begintableformat &\center \endtableformat
3. \=
4. \B!:  1 |  2 !  3 |  4 \E!
5. \-
6. \B!: 11 | 22 ! 33 | 44 \E!
7. \=
8. \endtable
```

Generuje on następujący wynik:

1	2	3	4
11	22	33	44

Instrukcje TEX-owe `\begintable` i `\endtable` (wiersz 1 i 8) oznaczają odpowiednio *rozpocznij składanie tabeli* i *zakończ składanie*. Zagnieżdżenie instrukcji `\begintable` i `\endtable` jest niedopuszczalne. Oznacza to, że tabela w zasadzie nie może zawierać innej tabeli.

Na szczęście istnieje metoda obejścia ewentualnych trudności. Każdą tabelę można złożyć osobno, umieszczając ją w pudełku pionowym za pomocą operacji `\setbox... \vbox...`, a następnie za pomocą jednej z operacji `\box`, `\copy`, `\unvbox` lub `\unvcopy` umieścić ją jako element wybranego pola tabeli zewnętrznej.

Instrukcje `\begintableformat` i `\endtableformat`, pojawiające się w wierszu 2, definiują preambułę tabeli, służącą do określenia sposobu justowania rubryk. Instrukcja `&\center` pojawiająca się między nimi niesie informację, że zawartość wszystkich rubryk ma być umieszczona w osi; pozostałe dwie możliwości oprogramowane w pakiecie `TAP` to oczywiście `\left` (dosunięcie zawartości pola do lewej) oraz `\right` (dosunięcie do prawej). Poszczególne rubryki w preambule oddziela się znakiem prostego podwójnego cudzysłowu „`"`” (p. niżej).

I znów wydawać by się mogło, że jest to drastyczne ograniczenie w stosunku do operacji `\halign`. A jednak praktyka pokazuje, że inne sposoby justowania niż *oś-lewo-prawo* trafiają się na tyle rzadko, że nie warto się na tę okazję specjalnie przygotowywać, zwłaszcza że z reguły wiąże się to ze skomplikowanymi lub nietypowymi przypadkami, a specjalne przypadki tak czy owak wymagają specjalnego podejścia.

Instrukcje `\-` oraz `\=` (wiersz 3, 5 i 7) oznaczają odpowiednio cieńszą i grubszą kreskę poziomą, zaś instrukcje reprezentowane za pomocą tzw. znaków aktywnych (*active characters*) „`|`” oraz „`!`” (wiersz 4 i 6) oznaczają odpowiednio grubszą i cieńszą kreskę pionową. Dopuszczalny jest też niewidzialny separator rubryk (można go interpretować jako niewidzialną kreskę pionową) oznaczany za pomocą znaku aktywnego „`"`” (podwójnego prostego cudzysłowu). Ten sam znak wykorzystywany jest też w preambule tabeli jako separator rubryk.

Instrukcje `\B` i `\E` (wiersz 4 i 6) oznaczają początek i koniec rubryki poziomej. Każda z instrukcji oczekuje jako parametru separatora rubryk, czyli jednego z wymienionych wyżej znaków aktywnych: „`|`”, „`!`” lub „`"`”. Znaczenie parametru jest oczywiste – definiuje on rodzaj kreski na skraju rubryki (niewidzialna, cienka bądź gruba) – patrz dokumentacja pakietu `TAP`.

Do omówienia pozostaje tylko dwukropek pojawiający się po instrukcji `\B!`. Określa on jakiej wielkości „drut” czyli `\strut` (niewidoczna TEX-owa `\vrule` o określonej wysokości i głębokości i zerowej szerokości) ma poprzedzać każdą rubrykę poziomą. Jest to parametr opcjonalny – pominięcie go oznacza, że rubryka ma mieć naturalną wysokość. W tym przypadku oznaczałoby to, że kreski poziome „przykleiłyby” się do cyfr, co dałoby efekt estetycznie nie najlepszy:

1	2	3	4
11	22	33	44

Długość „drutu” można określić za pomocą jednego z pięciu znaków:

`:   ^   _   +   -`

oznaczających odpowiednio:

- `:` przedłużenie w górę i w dół,
- `^` przedłużenie w górę,
- `_` przedłużenie w dół,
- `+` standardową wysokość rubryki,
- `-` rubrykę o wysokości zero („spłaszczoną”).

Jak przed chwilą widzieliśmy, brak któregokolwiek z wymienionych znaków oznacza, że rubryka będzie miała wysokość naturalną. Jeżeli taki jest zamysł, lepiej jest nie pomijać parametru, tylko użyć znaku „0” (cyfry zero).

Na pierwszy rzut oka taka rozbudowana „drutologia” wydać się może przesadą. Jednak raz jeszcze na obronę przywołać można praktykę: znaku „`:`” należy użyć jeżeli rubryka pozioma pojawia się między kreskami poziomymi, znaku „`^`” – jeżeli rubryka pojawia się pod kreską, znaku „`_`” – jeżeli rubryka pojawia się nad kreską, znaku „+” – jeżeli rubryka pojawia się między zwykłymi rubrykami.

Poniższy przykład demonstruje zastosowanie omówionych przed chwilą znaków:

```
\begintable
\begintableformat &\center \endtableformat
\=
\B!^  1  |  1  !  1  |  1  \E!
\B!+  2  |  2  !  2  |  2  \E!
\B!_  3  |  3  !  3  |  3  \E!
\-
\B!:  4  |  4  !  4  |  4  \E!
\-
\B!^  55  | 55  ! 55  | 55  \E!
\B!+  66  | 66  ! 66  | 66  \E!
\B!_  77  | 77  ! 77  | 77  \E!
\=
\endtable
```

Efekt składu przedstawia się następująco:

1 2 3	1 2 3	1 2 3	1 2 3
4	4	4	4
55 66 77	55 66 77	55 66 77	55 66 77

Chrzest bojowy za nami – teraz można ruszyć do przykładów nieco trudniejszych.

Skomplikowane tabele. Tytuł brzmi groźniej niż powinien, bowiem skomplikowane tabele okażą się nie takie znowu trudne do składania. Zacznijmy od pierwszego przykładu.

Oto odpowiedni kod TEX-owy:

```
1. \begintable
2. \begintableformat &\center \endtableformat
3. \=
4. \B!:rubryka 1|rubryka 2|rubryka 3| rubryka 4  \E!
5. \=
6. \B!:  @2 rubryka 1 i 2 |   @2 rubryka 3 i 4   \E!
7. \-
8. \B!:         |  @2 rubryka 2 i 3 |            \E!
9. \B!-                 @3 \-       |wiersz 3 i 4\E!
10. \B!:    @3 rubryka 1, 2, 3       |            \E!
11. \=
12. \endtable
```

I proszę – nie taki diabeł straszny, jak go malują. W stosunku do tego, co już wiemy, pojawił się tylko jeden nowy znak, mianowicie „@”. Jest to także znak aktywny, a jego znaczenia łatwo się domyślić – oznacza on sklejenie kilku rubryk w jedną rubrykę; liczba następująca po tym znaku określa liczbę rubryk które mają być sklejone. Przykład ten także demonstruje zastosowanie znaku „-” (wiersz 9, instrukcja \B!-), decydującego o spłaszczeniu rubryki – widać kiedy ten zabieg jest przydatny.

Nawiasem mówiąc kod generowany automatycznie jest inny (nie jest optymalny), ale efekt optyczny jest taki sam – por. poniższy skład i skład na str. 41:

<table>
<tr><td>rubryka 1</td><td>rubryka 2</td><td>rubryka 3</td><td>rubryka 4</td></tr>
<tr><td colspan="2">rubryka 1 i 2</td><td colspan="2">rubryka 3 i 4</td></tr>
<tr><td></td><td colspan="2">rubryka 2 i 3</td><td rowspan="2">wiersz 3 i 4</td></tr>
<tr><td colspan="3">rubryka 1, 2, 3</td></tr>
</table>

Z praktycznego punktu widzenia interesujące jest składanie tabel na zadaną szerokość. Pakiet TAP oczywiście na to pozwala. Wystarczy w tym celu zmiennej \desiredwidth (rejestr typu *dimen*) nadać żądaną wartość. Jeżeli zawartość rejestru \desiredwidth jest większa od zera, tabela zostanie złożona na zadaną szerokość. Na przykład poprzedzenie omawianej tabeli instrukcją \desiredwidth = \hsize spowoduje proporcjonalne rozciągnięcie rubryk tak, aby ostateczna szerokość wyniosła \hsize:

<table>
<tr><td>rubryka 1</td><td>rubryka 2</td><td>rubryka 3</td><td>rubryka 4</td></tr>
<tr><td colspan="2">rubryka 1 i 2</td><td colspan="2">rubryka 3 i 4</td></tr>
<tr><td></td><td colspan="2">rubryka 2 i 3</td><td rowspan="2">wiersz 3 i 4</td></tr>
<tr><td colspan="3">rubryka 1, 2, 3</td></tr>
</table>

Jeżeli wymuszenie szerokości tabeli dotyczy tylko jednej konkretnej tabeli, lepiej jest użyć konstrukcji \thistable{\desiredwidth = \hsize }.

Stosunkowo trudnym elementem przy składaniu tabel są pola zawierające nie pojedynczy wiersz, ale wiele wierszy. Pakiet TAP został wyposażony w makra \X i \Y ułatwiające skład w takich przypadkach. Efektem składu w pierwszym przypadku jest \vbox, a w drugim \vtop, aczkolwiek \X i \Y nie są po prostu skróconymi nazwami tych operacji TEX-owych. Oba makra oczekują parametru (koniecznie w nawiasach klamrowych), zawierającego materiał do ułożenia w wierszach. Wymuszenie łamania wierszy uzyskuje się za pomocą podwójnego w-tył-ciacha \\. Między ciągiem kontrolnym a nawiasem klamrowym otwierającym można podać opcjonalnie szerokość, na jaką ma być łamany tekst. Jeżeli szerokość nie zostanie podana, wszystkie łamania wierszy muszą być wykonane ręcznie.

Domyślnie justowanie jest takie samo jak justowanie w danej rubryce pionowej.

Załóżmy dla przykładu, że w ostatniej tabeli tekst w wierszu 9 ma być przełamany po słowie „wiersz”. Należy w tym celu użyć konstrukcji `\X{wiersz\\3 i 4}` lub `\Y{wiersz\\3 i 4}`. W tym przypadku nie ma większego znaczenia rozróżnienie czy pudełko jest `\vbox`-em czy `\vtop`-em, ponieważ zawartość każdego z pól tego akurat wiersza jest umieszczana ostatecznie w pudełku o zerowej wysokości i szerokości.

Gdy tekst jest nieco dłuższy, niewygodne staje się ręczne łamanie. Należy wówczas wyspecyfikować szerokość. Na przykład użycie konstrukcji `\X 25mm {wiersz 3~i~4 w~pudełku}` da w rezultacie następującą tabelę:

<table>
<tr><td>rubryka 1</td><td>rubryka 2</td><td>rubryka 3</td><td>rubryka 4</td></tr>
<tr><td colspan="2">rubryka 1 i 2</td><td colspan="2">rubryka 3 i 4</td></tr>
<tr><td></td><td colspan="2">rubryka 2 i 3</td><td rowspan="2">wiersz 3 i 4
w pudełku</td></tr>
<tr><td colspan="3">rubryka 1, 2, 3</td></tr>
</table>

Niekiedy zdarza się, że któreś z pól ma być justowane inaczej niż reszta danej rubryki pionowej, na przykład cała rubryka jest umieszczona w osi, a jedno pole chcemy justować do lewej. Można wówczas zastosować konstrukcję `\left{` ⟨ *zawartość pola* ⟩ `}`. Kontynuując ten sam przykład – użycie konstrukcji `\left{\X {wiersz\\3 i 4}}` dałoby następujący efekt:

<table>
<tr><td>rubryka 1</td><td>rubryka 2</td><td>rubryka 3</td><td>rubryka 4</td></tr>
<tr><td colspan="2">rubryka 1 i 2</td><td colspan="2">rubryka 3 i 4</td></tr>
<tr><td></td><td colspan="2">rubryka 2 i 3</td><td rowspan="2">wiersz
3 i 4</td></tr>
<tr><td colspan="3">rubryka 1, 2, 3</td></tr>
</table>

Wreszcie – justowanie liczb dziesiętnych w taki sposób, by przecinek dziesiętny znajdował się w jednym pionie wymaga skorzystania z naturalnej techniki TEX-owej jaką jest stosowanie „niewidzialnej cyfry”, czyli pustego pudełka o rozmiarach cyfry. Jest to sensowne tylko wówczas, gdy wszystkie cyfry mają tę samą szerokość (dla rodziny Computer Modern szerokość ta wynosi $^1/_2$em).

Pakiet `TAP`, wzorem pakietu `TABLES`, wykorzystuje tyldę „~” jako niewidzialną cyfrę. W makrach `plain`-owych tylda jest znakiem aktywnym, oznaczającym niełamliwą spację (wiązanie), zatem takie znaczenie tyldy ma sens jedynie w trybie pionowym, tzn. przy łamaniu akapitów. Ponieważ zawartość pól tabeli z reguły jest składana w trybie poziomym, wewnątrz tabeli tyldę można wykorzystać do innego celu. Cyfry w tabeli możemy więc zapisać tak:

```
...
\=
\B|^  ~~~~12,3~~~  \E|
\B|+  ~~~~~0,12~~  \E|
\B|+  ~12345,678~  \E|
...
```

co daje

12,3
0,12
12345,678

Lista ciekawych aspektów składu tabel jest wprawdzie jeszcze daleka od wyczerpania, na przykład nie omówione zostało zagadnienie zmiany parametrów tabeli takich jak grubość kresek czy długość „drutów”. Zainteresowanych szczegółami pozostaje odesłać do dokumentacji.

Kolorowe tabele

Korzystanie z POSTSCRIPT-u pozwala na operowanie praktycznie dowolnymi barwami. W niniejszym artykule z konieczności trzeba się będzie ograniczyć do szarości, chociaż z drugiej strony, jak wiadomo, szare jest piękne...

Technika kolorowania tła wybranych pól w pakiecie TAP bazuje na pomyśle dwukrotnego składania tabeli: tabela jest najpierw składana próbnie z zaznaczeniem położenia przeciwległych naroży wybranych rubryk, a następnie składana jest powtórnie z wykorzystaniem informacji o położeniu naroży. W obecnej wersji pakiet TAP pozwala na wypełnienie całego pola dowolnym kolorem (w modelu *cyan-magenta-yellow-black*), na wypełnienie kolorem połowy pola z podziałem wzdłuż przekątnej oraz na poprowadzenie wzdłuż przekątnej kolorowej linii o zadanej grubości.

Zaznaczanie naroży zostało zaimplementowane za pomocą instrukcji `\special`, które wstawiają do pliku `DVI` kod POSTSCRIPT-owy rejestrujący położenie danych punktów w POSTSCRIPT-owym układzie współrzędnych. POSTSCRIPT-owe operacje realizujące to zadanie zostały ochrzczone – dla krótkości – mianem kotwic (*anchors*).

Makra wykorzystujące informację o położeniu kotwic też są zaimplementowane za pomocą instrukcji `\special`, generujących stosowny kod POSTSCRIPT-owy odwołujący się do zarejestrowanych uprzednio położeń kotwic.

Kod wygenerowany przez makra TEX-owe jest umieszczany w pliku wynikowym przez sterownik POSTSCRIPT-owy. Należy pamiętać, że korzystanie z instrukcji `\special` związane jest zawsze z konkretnym sterownikiem. Pakiet TAP współpracuje ze sterownikiem DVIPS autorstwa Tomasa Rokickiego, aczkolwiek przystosowanie pakietu do innego sterownika nie powinno nastręczać większych trudności.

Zobaczmy, jak w praktyce wygląda operowanie kotwicami. Poniższy przykład pokazuje, w jaki sposób wypełnić szarym kolorem tło jednego z pól:

```
 1. \beginanchtable
 2. \begintableformat &\center \endtableformat
 3. \rectfill {0 0 0 .15} {kotwa}
 4. \=
 5. \uranchor 1 {kotwa}
 6. \B!:  1 |  2 \E!
 7. \llanchor 1 {kotwa}
 8. \-
 9. \B!: 3 | 4 \E!
10. \=
11. \endanchtable
```

1	2
3	4

Kilka słów komentarza:

– zamiast pary instrukcji: `\begintable–\endtable` użyta została para: `\beginanchtable–\endanchtable`;
– w wierszu 5 pojawia się polecenie umieszczenia kotwicy w pierwszej rubryce pionowej w górnym prawym narożniku (*upper right*) pola kolejnej rubryki poziomej; liczba występująca po instrukcji `\uranchor` oznacza właśnie numer rubryki pionowej, parametr „kotwa” pojawiający się po liczbie oznacza nazwę danej kotwicy (nadawaną przez użytkownika);
– w wierszu 7 pojawia się polecenie umieszczenia kotwicy w pierwszej rubryce pionowej w lewym dolnym narożniku (*lower left*) pola poprzedzającej kotwicę rubryki poziomej; podobnie jak poprzednio, liczba występująca po instrukcji `\llanchor` oznacza numer rubryki pionowej, parametr „kotwa” pojawiający się po liczbie oznacza nazwę nadawaną kotwicy przez użytkownika; konieczne jest „rzucenie” dokładnie dwóch kotwic o tej samej nazwie – w lewym dolnym i w prawym górnym narożniku, przy czym nie muszą się one odnosić do tej samej rubryki poziomej czy pionowej;
– wreszcie w wierszu 3 pojawia się polecenie wypełnienia prostokąta określonego za pomocą pary kotwic o nazwie „kotwa” 15-procentową szarością `\rectfill {0 0 0 .15} {kotwa}` (pierwszy parametr zawiera odpowiednie składowe CMYK); proszę zauważyć, że operacja wypełnienia pojawiła pozornie się *zanim* kotwice zostały rzucone; w istocie operacje wykorzystujące kotwice można umieścić w dowolnym miejscu tabeli (w odróżnieniu od instrukcji `\llanchor` i `\uranchor`), istotna jest jedynie ich wzajemna kolejność.

Gdybyśmy rubrykę w wierszu 9 ujęli w parę kotwic `\uranchor 2 {kotwa'}` oraz `\llanchor 2 {kotwa'}`, i dołożyli w dowolnym miejscu operację `\rectfill {0 0 0 .15} {kotwa'}`:

```
\beginanchtable
\begintableformat &\center \endtableformat
\rectfill {0 0 0 .15} {kotwa}
\rectfill {0 0 0 .15} {kotwa'}
\=
\uranchor 1 {kotwa}
\B!:  1 |  2 \E!
\llanchor 1 {kotwa}
\-
\uranchor 2 {kotwa'}
\B!: 3 | 4 \E!
\llanchor 2 {kotwa'}
\=
\endanchtable
```

otrzymalibyśmy „szachownicę”:

1	2
3	4

Oczywiście położenie kotwic zmienia się odpowiednio wraz ze zmianą zawartości i szerokości tabeli. Na przykład poprzedzenie ostatniej tabeli instrukcją `\thistable {\desiredwidth=5cm}` wywołałoby następujący skutek:

1	2
3	4

Oprócz wypełnienia tła kolorem często przydatna jest operacja narysowania przekątnej w danym polu – oto przykład:

	Avia	Baltic	Calisia
Avia		1 : 5	0 : 1
Baltic	0 : 0		7 : 21
Calisia	3 : 2	13 : 21	

Służy do tego operacja `\diagstroke`, posiadająca postać bardzo podobną do operacji `\rectfill`. Różnica sprowadza się do dwóch dodatkowych parametrów, zresztą całkiem naturalnych, podawanych przed współrzędnymi CMYK: zaraz po nazwie powinna się pojawić bez nawiasów klamrowych wielkość typu *dimen* oznaczająca grubość kreski, a następnie jedno ze słów kluczowych „up" (przekątna „rosnąca"), „down" (przekątna „malejąca") lub „both" (obie przekątne), podane także bez nawiasów klamrowych, precyzujące, wzdłuż której z przekątnych ma być rysowana kreska. W powyższym przykładzie po rzuceniu kotwic nazwanych „A", „B" i „C" użyte zostały trzy operacje \diagstroke:

```
\diagstroke 0.4pt both {0 0 0 1} A
\diagstroke 0.4pt both {0 0 0 1} B
\diagstroke 0.4pt both {0 0 0 1} C
```

Jak widać w stosunkowo prosty sposób można zatrudnić POSTSCRIPT z całym jego bogactwem do składania tabelek. Jedynie drobna część tego bogactwa została wykorzystana w pakiecie TAP. Nie ma, rzecz jasna, obowiązku poprzestania na skromnej ofercie pakietu TAP – wręcz odwrotnie, zachęcałbym do prób poszerzenia POSTSCRIPT-owych możliwości pakietu, lub chociaż informowania jego autorów o rozszerzeniach wartych uwzględnienia.

◇ Bogusław Jackowski
jacko@ipipan.gda.pl

Pakiet TAP dostępny jest w archiwum GUST w katalogu: /pub/TeX/GUST/contrib/BachoTeX96/tap070.zip. Artykuł przedstawiony powyżej jest wyborem z publikacji B. Jackowskiego udostępnionej na Konferencji w Bachotku.
(*Opr. StaW*)

Standardy

SGML w praktyce

Piotr Bolek

Co to jest SGML

SGML (*Standard Generalized Markup Language*) jest nazwą języka zdefiniowanego w standardzie ISO 8879 opublikowanym w październiku 1986 r. Twórcą standardu jest dr Charles Goldfarb. SGML jest językiem stosunkowo złożonym i bywa określany jako:

- język oznaczania (etykietowania) tekstu (ang. *tagging language*),
- narzędzie do opisywania logicznej struktury tekstów,
- schemat łączenia i odwoływania się do plików,
- system tekstowej bazy danych,
- podstawa systemów hipertekstowych i multimedialnych,
- narzędzie umożliwiające tworzenie szablonów dla dokumentów tekstowych,
- narzędzie pozwalające wykorzystać zakodowany nim tekst w sposób nieprzewidziany przez autora,
- przenośny, niezależny od architektury sprzętu i oprogramowania sposób kodowania tekstu,
- narzędzie umożliwiające zapisywanie dowolnych struktur hierarchicznych,
- rozszerzalny język opisu dokumentów,
- standard komunikacji między różnymi typami sprzętu i programami użytkowymi,
- metajęzyk umożliwiający tworzenie języków prezentacji struktury dokumentów.

Jak pisze C. Goldfarb wszystko to jest prawdą, ale nie całą. SGML jest po części tym wszystkim, ale jako całość jest czymś jeszcze więcej.

Potrzeba istnienia takiego standardu jak SGML wynikała z problemów, które pojawiają się w trakcie elektronicznego przetwarzania tekstów. Częste są w takich przypadkach trudności z konwersją tekstów podczas przenoszenia między różnymi programami (edytorami, systemami składu itp.) Przejście z jednego programu do innego często prowadzi do utraty informacji o strukturze, formatowaniu tekstu, albo nawet jego zawartości merytorycznej. Problemy istnieją także w przypadku wykorzystania w dokumencie niestandardowego zestawu znaków. SGML pozwala rozwiązywać takie problemy. Definiuje on zasady tworzenia języków opisujących logiczną strukturę dokumentów. Dzięki zestawowi deklaracji pozwala także jednoznacznie określić sposób kodowania informacji zastosowany przy tworzeniu dokumentu. Ułatwia więc przenoszenie dokumentów między różnymi systemami.

Standard służy jedynie do opisywania logicznej struktury dokumentów, nie determinuje ostatecznej formy prezentacji informacji, która może być docelowo przekształcana w najróżniejszy sposób. Dokument zakodowany z wykorzystaniem SGML-a może służyć jako postać wyjściowa do formatowania tej samej informacji w różny sposób i prezentacji z użyciem różnych mediów np. w formie drukowanej na papierze, w postacji hipertekstu albo tekstowej bazy danych. Bardzo prosty przykład tego typu zastosowania będzie zaprezentowany na zakończenie artykułu.

Istnieje wiele narzędzi służących do przetwarzania dokumentów zakodowanych w SGML-u. Wiele z nich jest dostępnych na zasadach public domain, coraz częściej pojawiają się także w produktach komercyjnych filtry służące do przekształcania informacji do postaci SGML.

Sposób kodowania dokumentów w SGML-u

Dokument w SGML-u zaczynać się powinien od zestawu deklaracji (*SGML declaration*) określających pewne parametry (np. długość identyfikatorów używanych do oznaczania elementów struktury), definiujących zestaw znaków używany w dokumencie oraz znaki specjalne służące do oznaczania różnych typów identyfikatorów. Deklaracje te nie zawsze występują w postaci jawnej, często ukryte są w systemie służącym do przygotowywania lub przetwarzania tekstu.

Drugim ważnym elementem jest Definicja Typu Dokumentu (ang. *Document Type Definition – DTD*). DTD określa w sposób formalny logiczną strukturę dokumentu – podobnie jak style dokumentu w LaTeX-u (struktura ta musi być zachowana jednak bardziej rygorystycznie niż w LaTeX-u – nie byłoby możliwe opuszczenie jednego poziomu nagłówków np. bezpośrednio po `\chapter` użycie `\subsection`, co w LaTeX-u daje efekt nieco dziwaczny, ale nie powoduje błędu).

Definicja DTD może znajdować się wewnątrz dokumentu, w jego nagłówku, jeżeli jednak ma być użyta przy tworzeniu wielu tekstów, zwykle zapisywana jest w oddzielnym pliku.

Elementy. W DTD zdefiniowane są wszystkie logiczne części danego typu dokumentu – elementy (ang. *elements*), które mają podobne zastosowanie jak LaTeX-owe środowiska. Każdy element składa się z etykiety otwierającej (ang. *start-tag*) i zamykającej (ang.*end-tag*). Etykieta otwierająca ma zwykle postać `<elem>`, a zamykająca `</elem>`. Inaczej niż w LaTeX-u, nie wszystkie elementy występujące w tekście muszą mieć zaznaczone jawnie etykiety otwierające i zamykające. W deklaracji SGML możliwe jest ustawienie parametrów dopuszczających różne formy minimalizacji wpisywanych etykiet. Minimalizacja taka może polegać na możliwości opuszczenia etykiety (`OMITTAG`), użyciu skróconej wersji etykiety (`SHORTTAG`), grupowaniu etykiet (`RANK`), albo automatycznym rozpoznawaniu etykiet (`DATATAG`). Dodatkowo możliwe jest używanie skrótów (ang. *short references*), pozwalających w pewnych kontekstach używać zamiast etykiet zdefiniowanych uprzednio symboli, lub ich sekwencji.

Po dopuszczeniu możliwości opuszczania etykiet można pominąć etykietę otwierającą lub zamykającą, jeżeli da się z kontekstu wydedukować konieczność jej istnienia. Jeżeli np. zdefiniujemy elementy `chap`, `title` i `p` w następujący sposób:

```
<!ELEMENT chap - - (title,p+)>
<!ELEMENT title O O (#PCDATA)>
<!ELEMENT p - O (#PCDATA)>
```

to możemy napisać w dokumencie:

```
<chap>
<title>Tytuł rozdziału
<p>Pierwszy akapit tekstu
</chap>
```

a nawet:

```
<chap>
Tytuł rozdziału
<p>Pierwszy akapit tekstu
</chap>
```

i będzie to równoważne z:

```
<chap>
<title>Tytuł rozdziału
</title>
<p>Pierwszy akapit tekstu
</p>
</chap>
```

W powyższym przykładzie możemy także zobaczyć jak wygląda definiowanie i używanie elementów. Element chap zdefiniowaliśmy jako składający się z tytułu (element `title`), z następującym po nim (przecinek „`,`" określa następstwo elementów) co najmniej jednym akapitem (znak plus „+" oznacza wystąpienie co najmniej jednokrotne). Element `title` zdefiniowany został jako zawierający dane tekstowe (`#PCDATA` ang. *parsed characted data*), bez żadnych elementów niższego poziomu. Znaki „`- -`", „`O O`" i „`- O`" występujące w definicjach elementów `chap`, `title` i `p` określają czy możliwe jest pominięcie odpowiednich etykiet otwierających lub zamykających. Znak minus „`-`" na pozycji pierwszej oznacza, że etykieta otwierająca musi wystąpić, na pozycji drugiej, że wymagana jest etykieta zamykająca. Litera „`O`" zezwala na pominięcie odpowiedniej etykiety.

Atrybuty. Każdy z elementów może zawierać atrybuty określające pewne cechy szczególne np. fakt numerowania listy wyliczeń:

```
<list type="numbered">
<item> Pierwszy
<item> Drugi
</list>
```

Element typu `list` ma w tym przykładzie atrybut `type`, któremu przypisano wartość `numbered`. W DTD zawarte są definicje atrybutów wszystkich elementów wraz z ich typami i dopuszczalnymi wartościami.

Definicje. W dokumencie SGML istnieje także możliwość definiowania pewnych powtarzalnych części jako pewnego rodzaju definicji zwanych w terminologii SGML-a *entities*. Definicja może zastąpić pewien często używany tekst i jest w takim przypadku podobna do bezparametrowej definicji TEX-owej np.:

```
<!ENTITY sgml "Standard Generalized
    Markup Language">
```

W tekście dokumentu każdy napis `&sgml;` zostanie zastąpiony przez `Standard Generalized Markup Language`.

Inny rodzaj definicji może być używany w pliku DTD, dając możliwość definiowania pewnych wspólnych cech kilku elementów naraz. Przykład tego typu definicji pojawi się w przykładowym DTD na końcu artykułu.

Inne składowe dokumentu SGML. W dokumencie SGML może wystąpić jeszcze wiele innych składowych, np. wyróżnione sekcje (ang. *marked sections*), instrukcje formatujące (ang. *procesing instructions*), poddokumenty (ang. *subdocuments*), odwołania do znaków niedostępnych z klawiatury (ang. *character references*). Nie ma tutaj miejsca na szczegółowe omówienie ich definiowania i sposobu wykorzystania.

Praca z SGML-em

W praktyce najważniejszą sprawą jest wstępne zidentyfikowanie logicznej struktury dokumentu i zbudowanie hierarchii elementów tej struktury. Następnie należy stworzyć lub wykorzystać istniejący DTD, określający strukturę dokumentu formalnie.

Po zidentyfikowaniu wszystkich logicznych elementów tekstu i wybraniu odpowiedniego typu dokumentu należy określić strukturę dokumentu wstawiając do tekstu etykiety (ang. *tags*), które zaznaczają typy poszczególnych elementów dokumentu. Wprowadzenie etykiet do tekstu może mieć miejsce w trakcie tworzenia dokumentu i czasem może być zupełnie niewidoczne dla użytkownika. Etap ten może stanowić koniec pracy z tekstem. Dalsza obróbka może być dokonywana przez innych. Zwykle należy jednak jeszcze dokonać weryfikacji dokumentu, czyli sprawdzić czy struktura stworzonego dokumentu jest zgodna z definicjami zawartymi w DTD. Do weryfikacji struktury służą programy zwane parserami, czytające DTD oraz dokument i stwierdzające czy dany dokument ma strukturę określoną w definicji czy nie. Czasem parser może także dokonać przekształcenia struktury dokumentu na postać kanoniczną (rozwiniętą), w której jawnie występują wszystkie etykiety otwierające i zamykające wraz ze wszystkimi atrybutami. Wyjście z parsera może być wykorzystane w etapie następnym – przygotowaniu dokumentu do prezentacji w sformatowanej postaci.

Formatowanie tekstu to etap, którym standard się już nie zajmuje, ale etap ten jest zwykle niezbędny, aby zakodowana informacja mogła być w sensowny sposób wykorzystana. Dokument może być sformatowany w sposób prosty bezpośrednio z dokumentu źródłowego, przez prostą edycję przy pomocy standardowych narzędzi, służących do przetwarzania tekstu (sed, awk, Perl), albo po uprzednim przetworzeniu tekstu przez parser na postać „rozwiniętą".

Przykład

Na koniec chciałbym przedstawić bardzo prosty przykład prezentujący opisane powyżej cechy SGML oraz możliwości, które daje ten sposób kodowania informacji. Przykładem tym jest przygotowywanie tabel do publikacji w WWW. Nie jest to przykład bardzo skomplikowany, stanowi on próbę pokazania możliwości praktycznego wykorzystania SGML-a.

Problem i propozycja rozwiązania. Prezentacja tabel w WWW sprawia problemy, ponieważ brak jest jednoznacznego sposobu ich kodowania w języku HTML. W definicji standardu HTML3 istnieje co prawda dosyć ciekawy sposób tworzenia tabel, ale większość używanych przeglądarek nie obsługuje języka HTML3. Sposób zapisywania tabel rozumiany przez przeglądarkę Netscape też nie jest całkiem wygodny, chociaż daje duże możliwości i jest częściowo zgodny ze standardem HTML3. Większość przeglądarek zupełnie jednak nie radzi sobie z wyświetlaniem tabel.

W zaproponowanym rozwiązaniu tabele tworzone są w kilku wariantach – oddzielnie dla Netscape'a w postaci przez niego zrozumiałej, a oddzielnie dla pozostałych przeglądarek – w postaci preformatowanego tekstu ASCII (tabelki są w tym przypadku składane z wykorzystaniem programu `groff` z preprocesorem `tbl`). Obie te wersje (oraz dodatkowa trzecia LaTeX-owa) są tworzone z tej samej wersji źródłowej zapisanej w SGML-u.

DTD. Aby umożliwić przetwarzanie tabel należy przede wszystkim przygotować odpowiednią definicję typu dokumentu – DTD. Do tego celu została wykorzystana zmodyfikowana i nieco uproszczona wersja definicji tabel z DTD dla języka HTML 3.0.

```
<!-- tab.dtd - DTD dla tabelek -->

<!ELEMENT TABLE - - (TR+)>
<!ATTLIST TABLE
        border (border) #IMPLIED
        colspec CDATA   #IMPLIED
        units  (en|pixels|relative) en
        width NUMBER    #IMPLIED
        padding NUMBER #IMPLIED
        spacing NUMBER #IMPLIED
        >

<!ENTITY % cell "TH | TD">
<!ENTITY % horiz.align
               "left|center|right|justify">
<!ENTITY % vert.align
               "top|middle|bottom|baseline">

<!ELEMENT TR O O (%cell)* >
<!ATTLIST TR
        align  (%horiz.align) #IMPLIED
        >

<!ELEMENT (%cell) - O (#PCDATA)>
<!ATTLIST (%cell)
        colspan NUMBER    1
        align  (%horiz.align) #IMPLIED
        >

<!ELEMENT TAB O O ANY>
```

Narzędzia. Do przetwarzania tabel użyty został parser `sgmls` Jamesa Clarka, oraz program `sgmlspl` napisany w Perlu, korzystający z biblioteki `SGMLS.pm` (autorem programu i biblioteki jest David Megginson z Kanady).

Program `sgmlspl` działa na danych wyjściowych z parsera `sgmls`. Argumentem tego programu jest plik ze specyfikacją, w którym w postaci kodu w Perlu zdefiniowane są akcje, które należy wykonać po napotkaniu kolejnych elementów struktury dokumentu. Dla każdego z formatów wyjściowych konieczny jest oczywiście inny plik ze specyfikacją. Kod specyfikacji nie będzie tutaj przedstawiany ponieważ jest dosyć specyficzny i stosunkowo skomplikowany, byłby więc mało zrozumiały nawet dla osób znających Perla. Zainteresowanych odsyłam do dokumentacji biblioteki `SGMLS.pm` i programu `sgmlspl`.

Przykładowa tabelka. Oto przykładowa tabelka zapisana zgodnie ze zdefiniowaną w `tab.dtd` specyfikacją.

```
<!doctype tab system>

<TABLE width=440 border padding=3
                  COLSPEC="lrrrr">
<TR ALIGN=CENTER>
  <TD>
  <TD COLSPAN=4>Zasoby
  <TR>
  <TD>
  <TD align=center colspan=2>Udokumentowane
  <TD align=center colspan=2>Wydobywalne
<TR ALIGN=left>
  <TD>Surowce
  <TD>mld t <TD> %
  <TD> mld t <TD> %
<TR>
  <TD> Węgiel kam.
  <TD> 36,18 <TD> 63,1 <TD> 14,47 <TD> 69,8
<TR>
  <TD>Węgiel brun.
  <TD>  2,74 <TD>  4,8 <TD>  2,46 <TD> 11,9
<TR>
  <TD>Gaz ziemny
  <TD>  0,17 <TD>  0,3 <TD>  0,14 <TD>  0,7
<TR>
  <TD>Ropa naftowa
  <TD> 0,005 <TD> 0,01 <TD> 0,004 <TD> 0,01
<TR>
  <TD>Energia wód
  <TD> 18,29 <TD> 31,9 <TD> 3,65 <TD> 17,6
</TABLE>
```

Tabelka ASCII. Oto przykładowa tabelka sformatowania programem `groff` i preprocesorem `tbl`:

```
+------------+-------------------------------+
|            |            Zasoby             |
+------------+---------------+---------------+
|            | Udokumentowane | Wydobywalne  |
+------------+-------+-------+-------+-------+
|Surowce     | mld t | %     | mld t | %     |
+------------+-------+-------+-------+-------+
|Węgiel kam. | 36,18 |  63,1 | 14,47 | 69,8 |
+------------+-------+-------+-------+-------+
|Węgiel brun. |  2,74 |   4,8 |  2,46 | 11,9 |
+------------+-------+-------+-------+-------+
|Gaz ziemny  |  0,17 |   0,3 |  0,14 |  0,7 |
+------------+-------+-------+-------+-------+
|Ropa naftowa | 0,005 |  0,01 | 0,004 | 0,01 |
+------------+-------+-------+-------+-------+
|Energia wód | 18,29 |  31,9 |  3,65 | 17,6 |
+------------+-------+-------+-------+-------+
```

Taką tabelkę można umieścić na stronie HTML przeznaczonej dla „ubogich” przeglądarek WWW.

Tabelka HTML. A tak wygląda ta sama tabelka po przetworzeniu na HTML oglądana pod Netscapem:

	Zasoby			
	Udokumentowane		Wydobywalne	
Surowce	mld t	%	mld t	%
Węgiel kam.	36,18	63,1	14,47	69,8
Węgiel brun.	2,74	4,8	2,46	11,9
Gaz ziemny	0,17	0,3	0,14	0,7
Ropa naftowa	0,005	0,01	0,004	0,01
Energia wód	18,29	31,9	3,65	17,6

Tabelka LaTeX. Na koniec ta sama tabelka w LaTeX:

	Zasoby			
	Udokumentowane		Wydobywalne	
Surowce	mld t	%	mld t	%
Węgiel kam.	36,18	63,1	14,47	69,8
Węgiel brun.	2,74	4,8	2,46	11,9
Gaz ziemny	0,17	0,3	0,14	0,7
Ropa naftowa	0,005	0,01	0,004	0,01
Energia wód	18,29	31,9	3,65	17,6

Literatura

1. Charles F. Goldfarb: *The SGML Handbook*, Oxford University Press, 1995,
2. Martin Bryan: *SGML, an author's guide to the Standard Generalized Markup Language*, Addison-Wesley, 1988.
3. David Megginson: Dokumentacja do biblioteki SGMLS.pm i programu sgmlspl, `http://www.uottawa.ca/~dmeggins/SGMLSpm/sgmlspm.html`, `http://www.uottawa.ca/~dmeggins/sgmlspl/sgmlspl.html`
4. Daniel Gilly: *Unix in the Nutshell*, O'Reilly & Associates, 1994.

◇ Piotr Bolek
`P.Bolek@ia.pw.edu.pl`