Makra

Zgrabne formatowanie tekstów programów komputerowych przy użyciu TEX-a

Marcin Woliński

Przedstawiony tutaj pakiet makr służy do publikowania programów komputerowych. Nie chodzi przy tym o drukowanie tasiemcowych listingów programów, lecz raczej o skład podręcznika języka programowania lub innej książki ilustrowanej algorytmami.

Wprowadzenie

W większości wydawanych obecnie w Polsce książek z dziedziny informatyki przykładowe fragmenty programów są składane czcionką o stałej szerokości znaków w sposób, który TEX-nik nazwałby `"verbatim"`. Tymczasem w staranniej wydanych pozycjach można znaleźć programy (szczególnie w Pascalu) złożone według dosyć skomplikowanych reguł typograficznych, które najprościej będzie przedstawić na przykładzie — zobacz tablicę 1 na następnej stronie.

Dzięki odróżnieniu słów kluczowych od innych identyfikatorów osobie uczącej się języka łatwiej jest się zorientować, z jakimi „częściami mowy” ma do czynienia. Kształt graficzny programu konsekwentnie odzwierciedla przepływ sterowania. Łatwo jest stwierdzić, „który **end** pasuje do którego **begin**”. Taki układ tekstu ma przy tym znaczenie nie tylko dla dydaktyki informatyki — wydaje się, że tak złożony program jest bardziej czytelny, szybciej pozwala zorientować się w istocie algorytmu. Kwestią, którą chcę się zająć, jest

Jak wykonać taki skład w TEX-u?

Spróbujmy złożyć mały fragment programu w języku Pascal:

begin $q := 0;\ r := x;$
while $r \geqslant y$ **do**
  begin $\{ q * y + r = x, r \geqslant y \}$
  $r := r - y;\ q := q + 1$
  end
end

Ręczne dostawienie wszystkich potrzebnych zmian krojów, odstępów i końców linii wydaje się być pracą iście katorżniczą. Bardzo trudno byłoby zachować konsekwencję w stosowaniu wyróżnień, zwłaszcza wcięć. Narzucające się zastosowanie środowiska `tabbing` niewiele w tym względzie pomoże. Istnieje kilka pakietów makr LATEX-owych, które mają zaradzić tej sytuacji, wymagają one jednak nadal skomplikowanego oznaczania tekstu makrami, a wynik często pozostawia wiele do życzenia.

Tymczasem okazuje się, że są w zasięgu ręki znacznie bardziej eleganckie rozwiązania. Otóż profesor Knuth postanowił pewnego razu opublikować dwa duże programy napisane w Pascalu — mianowicie TEX-a i METAFONT. Ponieważ nie miał zamiaru spędzić nad tym dziełem reszty swojego życia zdecydował, że najprościej będzie nauczyć komputer reguł składu Pascala. Szczęśliwie reguły te dają się sformułować w sposób wystarczająco ścisły. Tak oto jedną z funkcji systemu WEB stało się produkowanie „upiększonego” wydruku tekstu programu.

W pewnym uproszczeniu praca z WEB-em wygląda w ten sposób, że plik zawierający tekst programu okraszony obfitymi komentarzami traktuje się narzędziem o nazwie WEAVE i otrzymuje się plik dla TEX-a, w którym program jest opatrzony wszystkimi stosownymi zmianami krojów pisma i odstępami. Drugie narzędzie o nazwie TANGLE pozwala z tego samego tekstu uzyskać działający program.

Ten schemat doskonale nadaje się do produkowania programu wraz ze szczegółową dokumentacją techniczną. Gdy jednak tekst zawiera fragmenty przykładów, które nie mają w sumie stać się jednym działającym programem, użycie WEBa staje się nieco sztuczne. W takiej sytuacji byłoby wygodniej uniknąć użycia zewnętrznego narzędzia preformatującego wydruki. W tym celu trzeba nauczyć samego TEX-a reguł składu programów pascalowych.

Takie zadanie spełnia pakiet pretprin. Zawiera on makra TEX-owe formatujące program dokładnie tak, jak robi to WEAVE. Tekst przykładu powyżej w pliku źródłowym wygląda następująco:

```
\begin{Pascal}
begin q:=0;r:=x;while r>=y do begin(*
|q*y+r=x|, |r>=y| *)r:=r-y;q:=q+1end end
\end{Pascal}
```

Został on tak idiotycznie podzielony na linie, żeby pokazać, że nie ma to żadnego wpływu na kształt wyniku. Co więcej, zawiera on tylko te odstępy, które są *konieczne*, aby kompilator Pascala mógł zrozumieć ten fragment. Proszę zwrócić uwagę na brak odstępu między cyfrą 1 i słowem **end** — odstęp nie jest tu konieczny ponieważ identyfikator w Pascalu nie może zaczynać się cyfrą; na wydruku w tym miejscu zostaje dostawiony koniec linii. (Oczywiście w życiu jestem zwolennikiem pisania większej ilości spacji.)

Algorytm (7.14), choć znacznie przejrzystszy niż (7.13) i nie zawierający skoków, nie jest jeszcze zbyt zrozumiały. Lepszą wersję otrzymuje się wprost z własności (7.9)–(7.12)

begin $x := u$; $y := v$; $k := 1$; (7.15)
while $even(x) \wedge even(y)$ **do**
  begin $x := x$ **div** 2; $y := y$ **div** 2; $k := k * 2$
  end;
  $\{ nwd(u,v) = k * nwd(x,y) \}$
repeat $\{ odd(x) \vee odd(y) \}$
  while $even(x)$ **do** $x := x$ **div** 2;
  while $even(y)$ **do** $y := y$ **div** 2;
    $\{ odd(x) \wedge odd(y) \}$
  if $x > y$ **then** $x := x - y$
  else $y := y - x$
until $(x = 0) \vee (y = 0)$;
if $x = 0$ **then** $nwd := y * k$
else $nwd := x * k$
end $\{ nwd = nwd(u,v) \}$

Tworząc algorytm (7.15) rozumowano w następujący sposób:
(a) Ze względu na własność (7.9) po wykonaniu pętli

while $even(x) \wedge even(y)$ **do** (7.16)
  begin $x := x$ **div** 2; $y := y$ **div** 2; $k := k * 2$
  end

zachodzi równość $nwd(u,v) = k * nwd(x,y)$. Jest jasne, że ta pętla „dopóki” musi się skończyć, gdyż x i y można podzielić przez 2 tylko skończoną liczbę razy.

(b) Pętle

while $even(x)$ **do** $x := x$ **div** 2; (7.17)
while $even(y)$ **do** $y := y$ **div** 2

nie zmieniają wartości $nwd(x,y)$. Wynika to z własności (7.10) i tego, że gdy dochodzi się do pętli (7.17), jest spełniony warunek $odd(x) \vee odd(y)$. Jest tak przy pierwszym dojściu do pętli (7.17), gdyż dzieje się to po wyjściu z pętli (7.16). Po wykonaniu pętli (7.17) jest $odd(x) \wedge odd(y)$, a po instrukcji **if** $x > y$ **then** $x := x-y$ **else** $y := y-x$ albo x jest nieparzyste, a y parzyste, albo na odwrót. Wynika to z implikacji (7.12). Tak więc i przy każdym następnym dojściu do pętli (7.17) x lub y będzie nieparzyste.

Tablica 1: Fragment strony 247 książki S. Alagicia i M. Arbiba „Projektowanie programów poprawnych i dobrze zbudowanych”

Makra można łatwo zaadaptować do składu innych języków programowania. Na przykład poniżej mamy cztery klauzule procedury w Prologu wykonującej różniczkowanie symboliczne:

$$\mathrm{d(X,\ X,\ D)} \leftarrow \mathrm{atomic(X),\ !,\ D = 1.}$$
$$\mathrm{d(C,\ X,\ D)} \leftarrow \mathrm{atomic(C),\ !,\ D = 0.}$$
$$\mathrm{d(U + V,\ X,\ DU + DV)} \leftarrow \quad \mathrm{d(U,\ X,\ DU),\ d(V,\ X,\ DV).}$$
$$\mathrm{d(U * V,\ X,\ DU * V + U * DV)} \leftarrow \quad \mathrm{d(U,\ X,\ DU),\ d(V,\ X,\ DV).}$$

Pakiet pretprin

został napisany w sposób możliwie modularny. Zgodnie z ortodoksyjnymi zasadami budowania kompilatorów najpierw dzieli on ciąg znaków na „słowa” (nazywa się to analizą leksykalną), dla każdego z nich określa, jaka to „część mowy”, a następnie bada, czy słowa układają się w większe jednostki gramatyczne (analiza składniowa). Wtedy wedle dokonanego „rozbioru gramatycznego” produkuje „kod wynikowy”, czyli w naszym przypadku instrukcje formatujące tekst.

Najniższy poziom makr pakietu definiuje symulator automatów skończonych, potrzebny do realizacji analizatora leksykalnego, oraz interpreter reguł gramatycznych potrzebny do analizy składniowej. Muszę przyznać, że napisanie tych makr było dla mnie prawdziwym wyzwaniem (interpreter reguł posługuje się podwójnie łączoną listą, po której przesuwa się w obie strony badając elementy, a czasami wstawiając je i usuwając; operacje na liście nie wymagające jej modyfikowania odbywają się przez „czyste rozwinięcia"). Ten poziom jest niezależny od języka programowania.

Wyższy poziom jest specyficzny dla języka, ale operuje abstrakcyjnymi pojęciami skrzętnie skrywającymi problemy TEX-niczne. Jeżeli wstawić tu automat rozpoznający „słowa" Pascala oraz 54 reguły wymyślone przez prof. Knutha otrzymamy wydruki jak w pierwszych przykładach; jeżeli zamienić te reguły, można składać fragmenty w innych językach, a być może dowolne inne dane o odpowiednio formalnej składni.

Podręcznik użytkownika

makr nie jest skomplikowany. Całość została zrealizowana jako pakiet dla LATEX-a 2e, więc gdzieś po `\documentclass` powinno się znaleźć zaklęcie

```
\usepackage[pascal]{pretprin}
```

(jeżeli składamy Pascala, dla innych języków będą inne opcje). Od tego momentu możemy używać środowiska `Pascal` (uwaga na dużą literę). W obrębie tekstu pascalowego mogą występować komentarze. Tekst komentarza jest w zwykły sposób interpretowany przez TEX-a, mogą w nim występować makra i wzory matematyczne. Ponieważ w obrębie komentarza mogą być użyteczne grupy TEX-owe, wolałem nie zmieniać znaczenia nawiasów klamrowych. W związku z tym komentarze należy zaznaczać nawiasami `(*` oraz `*)`. Jeżeli w komentarzu ma się znaleźć fragment Pascala, powinien on być ujęty w znaki `|`, które w obrębie komentarzy w programach są aktywne. Wszystkie te zjawiska ilustruje początkowy przykład.

Do składu krótkich fragmentów Pascala w obrębie akapitu służy makro `\pascal` przyjmujące jeden parametr. Polecenie

```
\pascal{array [1..20] of real}
```

daje w składzie **array** [1 .. 20] **of** *real*.

Podsumowanie

Bieżący stan pakietu można uznać za wersję „beta". Upiększacz dla Pascala zachowuje się już w sposób dosyć przewidywalny. Pakiety dla Prologu, języka zapytań SQL i funkcyjnego języka SML znajdują się w fazie eksperymentów. W planach na dalszą przyszłość mam język C/C++ (który jest nieco trudniejszy). Analizator Pascala nie zawiera jeszcze wszystkich wodotrysków, które są dostępne w WEAVE'ie (pozwalających na „przebijanie" automatycznie podejmowanych decyzji). Mam jednak nadzieję stopniowo to naprawić.

Podstawowym celem tego przedsięwzięcia było zbadanie, czy w TEX-u da się zapisać nietrywialny algorytm. Nie chodziło przy tym o możliwość formalną — wiadomo, że „pyszczek" TEX-a ma siłę obliczeniową maszyny Turinga. Chciałem sprawdzić, czy nad nietrywialnym algorytmem zapisanym w TEX-u będzie można zapanować. Ten cel został osiągnięty: algorytm został zapisany w sposób ogólny i elegancki (na ile wolno mi to oceniać). Jak wynika z kierunku rozwoju kolejnych wersji makr, dobre efekty daje w TEX-u styl programowania przypominający języki funkcyjne, podczas gdy elementy imperatywne mogą skutecznie utrudnić programowanie. Ze względu na efektywność trzeba jednak osiągnąć jakąś rozsądną równowagę między tymi dwoma aspektami.

Podstawową wadą uzyskanego pakietu jest spowolnienie przetwarzania dokumentu przez TEX-a. Pełna strona programu może być przetwarzana przez czas potrzebny do złożenia kilku do kilkunastu zwykłych stron.

Warto zaznaczyć, że nie ma tu „konfliktu interesów" z WEBem — jeśli celem jest stworzenie działającego programu wraz z jego dokumentacją, należy użyć WEBa. Stosowanie pakietu pretprin jest sensowne tylko w przeciwnym przypadku.

Na zakończenie pozwolę sobie (po raz kolejny) wyrazić zachwyt nad TEX-em, szczególnie nad otwartością tego systemu pozwalającą na zrealizowanie w jego ramach nawet tak dziwacznego projektu, jak analizator składniowy Pascala.

◇ Marcin Woliński
Wolinski@bull.mimuw.edu.pl

Od red.: Tuż przed oddaniem Biuletynu do druku Autor udostępnił omawiany pakiet makr w wersji *beta* – dziękujemy. Można je znaleźć na serwerze GUST:
`/pub/TeX/GUST/contrib/MWolinski/pretprin-0.17.zip`.

(*StaW*)