3 1990

informatyka

Nowy język Turing
Automatyczny sterownik – ASTER
Sterowanie bazą wideodanych

WYDAWNICTWO NOT SIGMA

Cena: 5800 zł

informatyka

nr 3 1990
Marzec
Rok wydania XXV

Miesięcznik
ISSN 0542-9951. INDEX 36124

W numerze

W najbliższych numerach:

- Ilona Bluemke omawia ciekawsze rozwiązania procesorów tablicowych scalonych lub wykonywanych z elementów dyskretnych tolerujących błędy.
- Piotr Dembiński zajmuje się protokołami komunikacyjnymi w sieciach lokalnych.
- Tomasz Żakowicz zaczyna cykl artykułów na temat pakietu programów CDS/ISIS, umożliwiających tworzenie i obsługę systemów baz danych.
- Jacek Kardasiewicz opisuje podsystem projektowania technologii dla typoszeregów części, stanowiący jeden z elementów składowych systemu MikroAPO, wdrożony w wielu zakładach przemysłowych w kraju.
- Sławomir Czepielewski prezentuje środowisko programowe do projektowania pakietów elektronicznych systemów mikroprocesorowych.
- Krzysztof Rzymkowski omawia oprogramowanie systemu sieciowego IBM.

WYDAWCA:

Wydawnictwo Czasopism
i Książek Technicznych
NOT SIGMA
Spółka z o.o.
00-950 Warszawa
ul. Biała 4
skr. poczt. 1004

ADRES REDAKCJI:

Pl. Inwalidów 10, p. 128, 136
01-552 Warszawa
tel. 39-14-34

DRUK:

RSW „Prasa-Książka-Ruch"
Prasowe Zakłady Graficzne
ul. Dworcowa 13, 85-009 Bydgoszcz
zam. 191/90
Obj. 4,5 ark. druk. Nakład 6000 egz.

Cena egz. 5800 zł

WARUNKI PRENUMERATY

Prenumeratorzy zbiorowi jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat wyłącznie na blankiecie „wpłata zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia). Blankiety te będą dostarczane dotychczasowym prenumeratorom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu, w Radach Wojewódzkich NOT bądź w Redakcjach czasopism.

Prenumeratorzy indywidualni osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty. Wpłacać należy na konto: **Państwowy Bank Kredytowy III/O Warszawa nr 370015-7490-139-11.**

Prenumerata ulgowa przysługuje wyłącznie osobom fizycznym członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły. Sposób zamawiania prenumeraty ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po jednym egzemplarzu każdego czasopisma.
Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

Prenumerata ze zleceniem wysyłki za granicę zamawia się tak, jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy.
Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Wpłaty na prenumeratę przyjmowane są w terminach:
do 10 listopada na każdy kwartał, I i II półrocze oraz cały rok następny,
do 28 lutego na II, III i IV kwartał oraz II półrocze,
do 31 maja na III i IV kwartał oraz II półrocze,
do 31 sierpnia na IV kwartał.
Zmiany w prenumeracie można zgłaszać pisemnie tylko w wyżej wymienionych terminach.

Informacji o prenumeracie udziela Zakład Kolportażu Wydawnictwa NOT SIGMA (ul. Bartycka 20, 00-716 Warszawa), skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 wew. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa ul. Mazowiecka 12 (tel. 26-80-16), lub zamówić pisemnie. Zamówienia na egzemplarze archiwalne czasopism przyjmuje: Zakład Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.
Cena egzemplarza: normalna 5800 zł, ulgowa 1160 zł

Uwaga! Podane ceny mają charakter wstępny i mogą ulec zmianie, w związku z powyższym Wydawnictwo zastrzega sobie prawo żądania dopłat do już opłaconej prenumeraty.

RICHARD C. HOLT
Computer Systems Research Institute
University of Toronto
Toronto, prowincja Ontario
Kanada

Turing – język lepszy od Pascala

Turing, opracowany w 1982 roku na Uniwersytecie w Toronto, był wynikiem ponad dwudziestoletnich badań w dziedzinie projektowania języków programowania. Od 1983 roku jest on głównym językiem stosowanym w nauczaniu na tym Uniwersytecie.

Najkrócej można opisać ten język jako super-Pascal z precyzyjną składnią i ścisłą definicją matematyczną. Turing zawiera wszystkie konstrukcje Pascala, a ponadto moduły, wygodne napisy i operacje wejścia–wyjścia. Jego składnia jest bardzo prosta. Kompilatory i interpretery Turinga działają na komputerach IBM PC oraz w środowisku systemów operacyjnych Unix i VMS. Programowanie w tym języku jest ułatwione dzięki stworzeniu zintegrowanego środowiska programowego. Matematyczna definicja języka zapewnia, że napisane w nim programy działają jednakowo na komputerach osobistych, stacjach roboczych, minikomputerach i dużych komputerach, a ponadto są przenośne z jednego komputera na inny.

WPROWADZENIE

Główne cechy Turinga scharakteryzuję za pomocą przykładów. Pierwszy przykład jest pełnym choć prostym programem do wyprowadzania gwiazdek ułożonych w trójkąt:

```
var s: string := "*"
loop
  put s
  s := s+"*"
end loop
```

W pierwszym wierszu programu zadeklarowano zmienną napisową *s* i przypisano jej gwiazdkę „*" jako wartość początkową. Instrukcje w pętli **loop** są wykonywane powtarzalnie: instrukcja **put** służy do wyprowadzania wartości zmiennej *s*, a operacja „+" do spinania bieżącej wartości tej zmiennej ze znakiem „*". Instrukcja pętli spowoduje ostatecznie utworzenie napisu, którego długość przekroczy maksymalną, zależną od implementacji, określoną standardowo jako nie mniejszą niż 255 znaków. Gdy to nastąpi, program zakończy działanie wysyłając komunikat błędu (przy założeniu, że odblokowana jest kontrola błędów czasu wykonania).

Turing dopuszcza swobodny format programu, tzn. powyższy program można zapisać w jednym wierszu lub rozbić go na kilka wierszy. Warto też zwrócić uwagę na brak średników i wiersza nagłówkowego, jak np.:

```
program test (input, output)
```

Deklarację w pierwszym wierszu przykładu można skrócić do postaci:

```
var s:="*"
```

ponieważ typ, w tym przypadku **string**, może być opuszczony, jeśli w deklaracji występuje inicjowanie zmiennej. Choć nie podano tego w przykładzie, w deklaracjach można też inicjować nieskalary, jak tablice i rekordy.

Wejście–wyjście

Na wydruku 1 przedstawiono przykład programu odczytującego plik (strumień) i wyprowadzającego wszystkie wiersze tekstu zawierające napis „*iron*". Pierwszy wiersz programu zawiera komentarz zaczynający się znakiem procentu (%) i kończy znakiem końca wiersza. Zmienna *n* jest zadeklarowana jako całkowitoliczbowa, co wynika z wartości początkowej. Pętla jest wykonana powtarzalnie aż do napotkania znaku końca pliku (*eof*) w strumieniu wejściowym. Instrukcja **get** występuje w postaci umożliwiającej wczytanie całego wiersza do pojedynczej zmiennej o nazwie *line*. Inne postacie tej instrukcji służą do wczytywania słów z automatycznym opuszczaniem odstępów lub określonej liczby znaków.

```
%Wczytaj plik, drukuj wiersze z napisem "iron"
var line: string
var n:=0
loop
    exit when eof
    n := n+1
    get line: *
    if index(line,"iron") not=0 then
        put n:4," ",line
    end if
end loop
```

Wydruk 1

Funkcja *index (line, "iron")* udostępnia zero, jeśli wczytany wiersz nie zawiera napisu „*iron*". Zauważmy, że instrukcje **if** i **loop** kończą się klauzulami **end if** i **end loop**. Podobnie, instrukcja **case** kończy się klauzulą **end case**, a instrukcja **for** – **end for**. Wykonanie instrukcji **put** polega na wyprowadzeniu numeru wiersza w polu o szerokości czterech znaków, następnie jednej spacji i zawartości wiersza. Przykładowo, wiersze 8 i 52 mogą być wydrukowane następująco:

```
 8 and yet iron combines in two
52 mix carefully with iron filings, then
```

Instrukcje **get** i **put** są bardzo wygodne do operowania strumieniami wejściowo–wyjściowymi. Przykładowo, odczytywanie pliku **master** i zapisywanie pliku *newmaster* może wyglądać następująco:

```
var m, nm: int
open (m, "master", "r")                    %Odczyt = "r"
open (nm, "newmaster", "w")                %Zapis ="w"

...

get : m,x
put : nm,x
```

Tablice jako parametry: deklaracje

Turing na bardzo wygodne instrukcje do operowania tablicami, których rozmiar jest znany dopiero w czasie wykonania, co zilustrowano w przykładzie na wydruku 2. Wartość *n* określająca rozmiar tablicy *a* jest wczytywana instrukcją **get**. Zasięg zmiennej *a* rozciąga się od jej deklaracji do końca obejmującego ją bloku, tj. do końca programu. Jest dokładnie tak, jakby tuż przed deklaracją tablicy *a* zaczynał się blok **begin** i rozciągał aż do końca programu. W Turingu, deklaracja może występować wszędzie tam, gdzie występuje instrukcja. Ta właściwość

```
%Wczytaj i zsumuj zestaw liczb
var n: int
%Dynamiczna granica zakresu
get n
var a: array 1..n of real
%
function sum(b: array 1..* of real): real
    var total: real := 0.0
    for j:=1..upper(b)
        total := total + b(j)
    end for
    result total
end sum
%
... obliczenia ...
put "Sum of array: ",sum(a)
```

Wydruk 2

ułatwia modularyzację programów, pozwalając na umieszczenie deklaracji w miejscach, gdzie są rzeczywiście potrzebne, bez rozszerzania ich globalnej widoczności.

W nagłówku funkcji *sum* zadeklarowano jej parametr formalny *b* jako tablicę liczb rzeczywistych, której dolna granica indeksu równa się 1, a górna jest określona przez parametr aktualny. Parametrem aktualnym, może więc być dowolna tablica liczb rzeczywistych, której dolna granica indeksu jest równa 1. Reguła zgodności typów w Turingu określa, że tablice są równoważnych typów, jeśli ich granice są równe a typy elementów równoważne. Oznacza to strukturalną równoważność typów, tj. opartą na postaci tablic, a nie równoważność nazw, jak w Pascalu, gdzie wymaga się użycia identyfikatorów typów.

Podobnie jak w Pascalu, parametry formalne można zadeklarować jako **var** lub nie **var**. Przypisania parametrom **var** powodują zmiany wartości odpowiednich parametrów aktualnych. W przeciwieństwie do Pascala, w Turingu parametry nie zadeklarowane jako **var** są uważane za stałe i nie można im przypisać wartości. Ponieważ w Turingu niedozwolone są efekty uboczne funkcji, nie można używać parametrów formalnych **var**. W Turingu uogólniono pojęcie stałej na wartości, które są obliczane w czasie wykonania, lecz nie mogą być zmienne. Znaczy to, że dla każdego wywołania funkcji *sum* wartość *b* musi pozostać niezmieniona, ale inne wywołanie może ją zmienić. Zgodnie z deklaracją:

```
const answer := sum(a)
```

podobnie jak *b*, *answer* jest uważana za stałą, lecz stałą czasu wykonania a nie czasu kompilacji, ponieważ jej wartość jest znana dopiero podczas wykonania.

Niektóre instrukcje i funkcje

Pętla **for** w Turingu zawiera licznik, opatrzony w naszym przykładzie nazwą *j*. W każdej iteracji pętli jest on traktowany jak stała, a komputer zapewnia, że użytkownik nie może go zmienić. Podobnie jak w Pascalu, wartość licznika pętli może się zwiększać lub zmniejszać. Zakres licznika może być podzakresem jakiegoś typu, jak w poniższym przykładzie:

```
type pixelRange : 1..472
var line : array pixelRange of 0..7
for i: pixelRange
  line(i) := 0
end for
```

Wartość funkcji *sum* jest udostępniana instrukcją **result**, podobną do instrukcji **return**, udostępniającej wartości w językach PL/I i C. Słowo zastrzeżone **return** służy w Turingu do jawnego zakończenia procedury lub programu głównego.

```
function intDollar(i: int) s: string
    pre i>=0
    post s(1)="$" and s(*-2)="."
                and i=strint(s(2..*-3)+s(*-1..*))
    var t: string := intstr(i)
    if length(t)=1 then
        t := "00"+t
    elsif length(t)=2 then
        t := "0"+t
    end if
    assert length(t) >= 3
    result "$" + t(1..*-2) + "." + t(*-1..*)
end intDollar
```

Wydruk 3

Funkcja przestawiona na wydruku 3 ilustruje użycie operacji napisowych. Pobiera liczbę całkowitą *i*, reprezentującą centy, i udostępnia równoważny napis odpowiadający tej wartości w dolarach, np. jeśli *i* równa się 5, to odpowiedni napis będzie równy $ 0,05. Funkcja *intDollar* korzysta z funkcji *intstr* do konwersji liczby całkowitej na napis. Inne funkcje konwersji typów występujące w Turingu to: *realstr*, *strint*, *strreal*, *floor* i *ceil*.

W przykładzie użyto klauzuli **elsif** stosowanej do kaskadowego wyboru instrukcji **if**, która służy do wydłużenia wartości napisu *t* z lewej strony, co najmniej do trzech cyfr. Jeżeli na przykład *i* równa się 5, to *t* ma wartość 005. Instrukcja **result** korzysta z wycinków napisów, zapisywanych jako *t(L..R)*, gdzie *L* oznacza położenie lewej strony wycinka w napisie *t*, a *R* – prawej strony. Gwiazdka „*" reprezentuje w tym zapisie długość napisu *t*. Instrukcją **result** spina się cztery następujące napisy:

- znak dolara „$"
- wszystkie cyfry napisu *t* oprócz dwóch ostatnich, tj. *t*(i..*–2)
- kropkę dziesiętną „."
- dwie ostatnie cyfry napisu *t*, tj. *t*(*–1..*).

Wycinek może wybierać pojedynczy znak (napis o długości 1) posługując się jedną pozycją, np. *t*(1) zawiera pierwszą cyfrę napisu *t*, a *t*(*) – ostatnią.

Omawiany przykład ilustruje też pewien styl programowania w Turingu, polegający na używaniu wielkich liter do rozpoczynania słów występujących w identyfikatorach, jak np. *D* w *intDollar*. W przeciwieństwie do Pascala, w Turingu rozróżnia się wielkie i małe litery, np. identyfikatory *inTDolLar*, *Intdollar* i *iNtDoLlAr* są rozróżnialne w Turingu, lecz nie w Pascalu.

ASERCJE

Turing zawiera zbiór asercji (*pre*, *post*, *assert*, *invariant*) do specyfikowania wymagań dla programów. Ich użycie zilustruję ponownie funkcją *intDollar*, zastępując każdy komentarz odpowiednią asercją (wydruk 4).

```
function intDollar(i: int): string
%-- i musi być nieujemne
%-- wynik jest wartością w dolarach
    var t := intstr(i)
    if length(t)=1 then
        t := "00"+t
    elsif length(t)=2 then
        t := "0"+t
    end if
%-- t ma długość przynajmniej 3 znaków
    result "$" + t(1..*-2) + "." + t(*-1..*)
end intDollar
```

Wydruk 4

W nagłówku funkcji zadeklarowano zmienną *s* przeznaczoną na wynik, poprzedzając nią typ wyniku funkcji. Jedynym miejscem, w którym można użyć zmiennej *s*, jest tzw. warunek ostateczny **post**, ponieważ wartość *s* jest dostępna jedynie po wykonaniu instrukcji **result**, tj. podczas powrotu z wykonania funkcji.

Warunek **pre**, tzw. warunek wstępny, jest wymaganiem narzuconym na wywołanie. Funkcja *intDollar* musi być wywołana z nieujemnym argumentem. Warunek **post** jest wymaganiem, które musi spełnić podprogram. Wynik z funkcji *intDollar* powinien zapewnić prawdziwość warunku **post**. Każda instrukcja *assert* wyraża założenie polega na tym, że zmienna *t* występująca po instrukcji **if** musi zawierać przynajmniej trzy znaki.

Asercje są sprawdzane standardowo w czasie wykonania, choć kompilator może ominąć kontrolę, jeżeli potrafi udowodnić, że warunek nie może być naruszony. Ponieważ w przykładzie zastąpiono asercjami komentarze, można uważać, że są one wykonywalnymi komentarzami. Kompilator generuje kod dla sprawdzenia założeń poczynionych w asercjach. Zauważmy, że błędy mogą wystąpić zarówno w założeniu (asercji), jak i w samym programie. Jeżeli zachodzi potrzeba szybkiego wykonania, to programista może usunąć kontrole i związane z nimi narzuty czasu wykonania, kompilując program ponownie z odpowiednią opcją.

Semantyka formalna Turinga definiuje dla całego programu warunek wstępny, który musi być spełniony przez dane wejściowe, jeśli program ma spełniać swoją specyfikację. Jeżeli dane wejściowe spełniają ten warunek wstępny, to każda asercja będzie prawdziwa podczas wykonania. W przykładzie nie potrzeba sprawdzać asercji w czasie wykonania, ponieważ zagwarantowano, że są prawdziwe. Przykładowo, jeśli zapewniono, że zmienna *i* w funkcji intDollar jest zawsze nieujemna, to przeprowadzenie w czasie wykonania kontroli warunków **assert** i **post** jest niepotrzebne. Nie będę tu podawał przykładów asercji **invariant**, które służą do ustalania wymagań dla pętli i modułów.

WSKAZANIA I UNIE

Wskazania i dynamiczna alokacja pamięci są podobne do odpowiednich cech Pascala. Jednakże w Turingu, obiekty alokowane dynamicznie są dzielone na dwie różne kolekcje. Kolekcję można uważać za tablicę, która początkowo nie ma żadnych elementów. Do dołączania elementów do kolekcji służy instrukcja **new**, a do usuwania ich z kolekcji – instrukcja **free**. Dla określonej kolekcji *c* i wskazania *p*, te dwie instrukcje mają następującą postać:

```
new c, p     %Utworzenie elementu kolekcji c wskazywanego przez p
free c, p    %Usunięcie elementu kolekcji c wskazywanego przez p
```

Po utworzeniu elementu, lecz przed jego usunięciem, można odwoływać się do niego za pomocą wyrażenia *c(p)*.

Na wydruku 5 podano przykład listy jednokierunkowej, zrealizowanej według kolekcji o nazwie *list*. Każdy węzeł (element) listy zawiera pole nazwy *name* i dowiązanie do innego węzła. Zapis *nil(list)* oznacza w tej kolekcji wskazanie *nijakie* (ang. *null*). Zauważmy, że odwołanie do elementu tablicy *a* o indeksie *i*, tzn. *a(i)*, jest notacyjnie podobne do odwołania *c(p)*. To podobieństwo nazywa się **jednolitością odwołań** (ang. *uniform referent*) i umożliwia programowanie w Turingu, w dużym stopniu, bez zwracania uwagi na to, czy struktura danych jest reprezentowana przez tablice czy przez kolekcje. Takie podobieństwo oznacza również, że kolekcje można zdefiniować formalnie w sposób analogiczny do tablic.

```
%Jednokierunkowa lista nazw
var list: collection of
     record
          next: pointer to list
          name: string(30)
     end record
$-- lista pusta
var first,last := nil(list)
$
procedure append(p: pointer to list)
     if first=nil(list) then
          first := p
     else
          list(last).next := p
     end if
     last := p
     list(p).next := nil(list)
end append
$ ...
var item: pointer to list
%-- alokacja elementu
new list,item
list(item).name := "A.M.Turing"
% dopisanie nowego elementu
append(item)
```

Wydruk 5

W Turingu istnieje pewna odmiana rekordów wariantowych, zwanych **uniami** (ang. *union*). Unia jest typem, w którym selekcja aktywnych składowych, tzw. **ewentualności** (ang. *alternative*), może być dokonywana w czasie wykonania programu. Unia umożliwia współdzielenie pamięci przez wartości odpowiadające różnym ewentualnościom. W przykładzie na wydruku 6, który dotyczy rejestrowania pojazdów, przedstawiono użycie i inicjowanie unii oraz wprowadzono typy wyliczeniowe. Wszystkie pojazdy podzielono w nim na samochody pasażerskie i rolnicze oraz pojazdy rekreacyjne. Jeżeli pojazd jest samochodem pasażerskim, to należy zarejestrować jego liczbę cylindrów, jeżeli jest pojazdem rolniczym, to należy zaliczyć go do określonej klasy, np. mleczarka (ang. *dairy*), umieszczając informację w napisie o długości nie przekraczającej 10 znaków. W pozostałych przypadkach jest to pojazd rekreacyjny, o którym nie rejestruje się żadnej informacji.

```
type vehicle: enum(passenger,farm,recreational)
type vehicleRecord:
     union kind: vehicle of
          label vehicle.passenger:
               cylinders: 1..16
          label vehicle.farm:
               farmClass: string(10)
          label: %-- trzecie pole jest puste
     end union
%-- inicjowanie metki i farmClass
var v: vehicleRecord := init(vehicle.farm,"dairy")

%-- sprawdzenie czy metka dotyczy samochodu rolniczego
.farmClass := "vegetable"

tag v,vehicle.passenger
-- sprawdzenie czy metka dotyczy samochodu osobowego
.cylinder := 4
```

Wydruk 6

W pierwszym wierszu programu utworzono typ wyliczeniowy o nazwie *vehicle*, mając trzy wartości, do których można odwoływać się przez ich identyfikatory, poprzedzając je nazwą typu oddzieloną kropką, np. *vehicle farm*. Jest to inaczej niż w Pascalu, gdzie wartość wyliczeniowa jest dana wprost przez swoją nazwę, jak np. *farm*, bez poprzedzenia jej prefiksem – w tym przypadku *vehicle*. Podejście pascalowe sprawia kłopoty, ponieważ niejasny jest zasięg identyfikatorów wyliczeniowych, szczególnie wtedy, gdy typ wyliczeniowy jest zdefiniowany wewnątrz innego typu, gdy występują kolizje nazw lub gdy typy wyliczeniowe są eksportowane z modułów. W Turingu traktuje się typy wyliczeniowe w sposób zbliżony do rekordów, a dostęp do wartości typu uzyskuje się tak, jak do pól rekordu. Dzięki temu unika się niejasności i ułatwia implementację.

W przykładzie, metka[1] *kind* określa aktywną ewentualność unii. Rekord *v* jest inicjowany w deklaracji ustawiającej metkę na wartość *farm*, a zmienną *farm Class* na wartość *dairy* (inicjowanie jest opcjonalne). Każda próba dostępu do pola *cylinders*, podczas ustawienia metki na wartość *farm*, zostanie potraktowana w czasie wykonania jako błąd. Zmiana wartości metki może spowodować zniszczenie zawartości pozostałych pól. Zauważmy też, że jeśli *kind* ma wartość *recreational*, to nie ma innych pól aktywnych oprócz samego pola *kind*. Jeżeli istnieją wspólne pola wszystkich ewentualności unii, to są one zadeklarowane w typie rekordowym, zawierającym unię jako jedno ze swoich pól.

WIĄZANIE

W Turing zastąpiono znaną z Pascala konstrukcję **with** analogiczną konstrukcją **bind**. W poniższym segmencie:

```
type r:
  record
    s, t: string (10)
    i: int
  end record
var a: array i...500 of r
...
bind var x to a(j)
x.s := "Hughes"
```

deklaracja **bind** tworzy zmienną *x* odpowiadającą *j*-emu elementowi tablicy *a*. Zmienną *x* można uważać za parametr formalny **var**, któremu odpowiada parametr aktualny *a(j)*. Każda zmiana *x* jest spowodowana zmianą *a(j)*. Jeżeli pominięte zostanie słowo zastrzeżone **var**, to *x* może być tylko odczytywany, lecz nie modyfikowany. Odpowiedni segment w Pascalu miałby postać:

```
with a[j] do
  begin
  s := "Hughes"
  ...
  end
```

gdzie klauzula **with** czyni pola *a[j]* wprost widocznymi.

Konstrukcja **bind** ma szereg zalet w porównaniu z klauzulą **with**. Najważniejsza z nich polega na możliwości jednoczesnego dostępu do więcej niż jednego zbioru pól zadeklarowanego typu rekordowego, np.:

```
bind var x to a(1), y to a(j)
x.s := y.t
```

W Pascalu nie można napisać odpowiedniego programu, nawet tak, jak poniżej:

```
with a[i], a[j] do...
```

ponieważ pola *a[i]* są przesłonięte przez pola *a[j]*, które stają się jedynie widoczne.

Ponieważ deklaracja **bind** przemianowuje zmienną (lub jej część), jest potencjalnym źródłem **utożsamiania** (ang. *aliasing*). Takiej możliwości zapobiega jednak szereg ograniczeń, z których najważniejsze jest zastrzeżenie, aby zmienna wiązana nie była dostępna w zasięgu deklaracji **bind**. Znaczy to, że w zasięgu zmiennej *x* nie można odwoływać się do tablicy *a*.

PODPROGRAMY I MODUŁY

Poniżej omówię programy parametryczne, posługując się przykładem aproksymacji całki funkcji matematycznej *f*. Wykonanie programu przedstawionego na wydruku 7 powoduje wyprowadzenie tekstu:

Integral of cos from zero to pi is 0.00

W pierwszym wierszu programu zadeklarowano funkcję całkowaną *f*, mającą rzeczywisty parametr i rzeczywisty wynik. Funkcja całkująca *integrate* pobiera jako parametry funkcję *f*, początek (*left*) i koniec (*right*) przedziału całkowania oraz liczbę odcinków aproksymacji – *n*.

[1] W innych językach programowania **metka** (ang. *tag*) jest zwana **wyróżnikiem** (ang. *discriminant*) – przyp. red.

Szerokość odcinka aproksymacji jest obliczana jako stała czasu wykonania. Typ zmiennej *sum* jest rzeczywisty, co wynika z jej wartości początkowej, równej 0,0. Użycie instrukcji **put** ilustruje sterowanie szerokością pola (4 znaki) i liczbą cyfr po kropce dziesiętnej (2 znaki) przy wyprowadzaniu liczb (oczywiście, można też sterować liczbą cyfr wykładnika).

```
function integrate(function f(x: real): real,
                   left,right: real, n: int): real
     const width := (right-left)/n
     var sum := 0.0
     for i: 1..n
          sum := sum + f(left + i*width)
     end for
     result width*sum
end integrate
%-- Przykład całkowania
put "Całka funkcji cosinus od zera do pi wynosi ",
          integrate(cos,0.0,3.14,100):4:2
```

Wydruk 7

Najważniejszą chyba konstrukcją wprowadzoną do języków programowania w ostatnim dziesięcioleciu są **moduły**, zwane także **klasami**, **skupieniami** (ang. *cluster*) lub **pakietami**. Moduły zapewniają przesłanianie informacji dzięki metodzie kontrolowania dostępu do zadeklarowanych obiektów. Na wydruku 8 przedstawiono prosty moduł, który eksportuje dwa podprogramy. Pierwszy podprogram, nazwany *enter*, służy dołączaniu liczb całkowitych do zbioru, a drugi, nazwany *member*, określa, czy dana liczba całkowita należy do tego zbioru. Wybór najlepszej struktury danych i algorytmów do implementacji podprogramów *enter* i *member* zależy od zakresu dołączanych liczb całkowitych i od częstotliwości wykonywania obu operacji. Jeżeli zakres liczb jest niewielki, np. 0 do 20, to można zaimplementować oba podprogramy bezpośrednio, używając typu mnogościowego **set** (wydruk 8).

Typy mnogościowe w Turingu są podobne do typów mnogościowych w Pascalu, przy czym konstruktory mnogościowe w Turingu używają nazw typów. Przykładowo, zbiór pusty w module na wydruku 8 zapisano jako *smallset()*, a nie [ ] jak w Pascalu. Kłopot z zapisem pascalowym polega na tym, że typ konstruktora mnogościowego, takiego jak [ ], jest niejednoznaczny.

```
module intset
     export(enter,member)
     const minval := 0
     const maxval := 20
     type smallset: set of minval..maxval
%-- zbiór pusty
     var s::smallset()
%--
     procedure enter(i: int)
          s := s + smallset(i)
     end enter
%--
     function member(i: int): boolean
          result i in s
     end member
end intset
%--
%-- Dodaj do zbioru 9
intset.enter(9)
%-- Czy 4 należy do zbioru ?
if intset.member(4) then ...
```

Wydruk 8

Jeżeli zakres liczb całkowitych zwiększy się znacznie, np. od 0 do 5000, tzn. zbiór będzie uzupełniany o niewielką liczbę elementów, to przedstawioną implementację modułu *intset* należy zmienić, używając – na przykład – tablic lub kolekcji. Jest to zmiana typowa dla konserwowania programu. Informacja o implementacji jest ukryta wewnątrz modułu, ponieważ nic oprócz dwóch podprogramów nie jest z niego eksportowane. Dlatego można w sposób bezpieczny ponownie zaimplementować moduł, mając gwarancję, że zmiana implementacji typu mnogościowego nie wpłynie w najmniejszym stopniu na pozostałe części programu.

* * *

Turing jest językiem purytańskim, tzn. nie pozwala w żaden sposób na dostęp do sprzętu. Takie podejście ma tę zaletę, że gwarantuje przenośność oraz poprawia strukturalność i czystość programu, które są podstawowymi cechami dobrego programowania. Jednakże w niektórych zadaniach dostęp do sprzętu jest konieczny. Umożliwia to Turing Plus, mniej purytańskie rozszerzenie Turinga zapewniające z nim zgodność pod tym względem, że każdy program w Turingu jest również programem w Turingu Plus.

Turing Plus umożliwia stosowanie **różnych rozmiarów typów liczbowych, rozłączne kompilowanie** podprogramów i modułów, **kontrolę typów**, bezpośrednie **zaglądanie** (ang. *peek*) i **wkładanie** (ang. *poke*) do pamięci, **obsługę wyjątków** i **współbieżność**. Jest on zaprojektowany do zastosowań, w których równie dobrze można używać języka asemblerowego, C lub Ady. Przenośny kompilator języka Turing Plus jest już dość dobrze przetestowany.

Tłumaczył i opracował JANUSZ ZALEWSKI

LITERATURA

[1] Cordy J. R., Graham T.C.N.: Design of an Interpretive Environment for Turing. Proc. ACM SIGPLAN'87 Symp. on Interpreters and Interpretive Techniques, Minneapolis – St. Paul (MN), June 1987

[2] Holt R. C., Cordy J. R.: The Turing Language Report. Technical Report CSRI-153, University of Toronto, December 1983

[3] Holt R. C. et al.: Introduction to Computer Science Using the Turing Programming Language. Prentice-Hall, Englewood Cliffs (NJ), 1987

[4] Holt R. C. et. al.: The Turing Programming Language – Design and Definition. Prentice-Hall, Englewood Cliffs (NJ), 1987.

Nowe książki z dziedziny poligraficznego składu komputerowego

T. Bove, C. Rhodes, Desktop Publishing with Pagemaker for IBM PC/AT, PS/2 and Compatible and for the Macintosh. Wiley, Chichester (UK), 1987

Są to dwa przewodniki, w których na przykładach z różnych publikacji prezentuje się techniki i polecenia umożliwiające tworzenie wzorców pism urzędowych, typowych dokumentów biurowych, dokumentacji technicznych, książek, czasopism itp. Książka jest zalecana wszystkim użytkownikom pakietu Pagemaker – stanowi doskonały dodatek do jego podręcznika użytkownika.

Richard J. Janz, Ventura Publisher for the IBM PC, Mastering Desktop Publishing (Covers Version 1.1). Wiley, 1987

Książka składa się z trzech części. W pierwszej przedstawiono to wszystko, co należy wiedzieć przed rozpoczęciem pracy z pakietem Ventura Publisher. W części drugiej zaprezentowano sprzęg graficzny Ventury oraz opisano dokładnie cztery podstawowe tryby pracy; omówiono także takie zagadnienia, jak definiowanie stylów lub automatyczna paginacja. W części trzeciej, krok po kroku opisano znaczenie i sposób korzystania z poszczególnych funkcji Ventury; forma graficzna, m.in. umieszczenie na marginesie aktualnie omawianego menu, ułatwia korzystanie z książki i odszukiwanie potrzebnych informacji. Ostatni rozdział zawiera wskazówki dotyczące najskuteczniejszego korzystania ze stylów oraz wstępnego formatowania plików tekstowych.

Graham Jones, Desktop Publishing Companion. Wiley, 1989

Książka zawiera odpowiedzi na pytania: Czym jest „desktop publishing", czego wymaga i jakie może przynieść korzyści. Napisana przez autora, który czuje i rozumie zarówno komputery, jak i problemy poligraficzno-wydawnicze. Polecana każdemu, kto zamierza zajmować się tego rodzaju działalnością.

Michael L. Kleper. The Illustrated Handbook of Desktop Publishing. Wiley, 1989

Książka jest napisana jasnym, nietechnicznym stylem, zawiera wiele praktycznych informacji na temat metod i urządzeń do składu tekstów i reprodukcji obrazów, stosowanej terminologii fachowej i wreszcie możliwości zastosowania mikrokomputerów (sprzęt i odpowiednie oprogramowanie) do tzw. małej poligrafii (*desktop publishing*). W dodatku zebrano wiele cennych informacji i wskazówek niezbędnych do pełnego wykorzystania możliwości istniejących systemów DTP. Dzięki swej strukturze, układowi materiału i jasnemu stylowi książka może stanowić podręczną encyklopedię z tej dziedziny.

Bezpieczeństwo zapisu danych – Produkcja firmy BASF

W 1934 roku firma BASF dokonała przełomu technologicznego uzyskując pierwszą na skalę przemysłową produkcję magnetycznego nośnika zapisu informacji. Na tej samej zasadzie, zapisu magnetycznego – pierwotnie stosowanego do zapisu dźwięku – opiera się jeszcze dzisiaj zapis dźwięku, obrazu i danych. Stały postęp i produkcja najwyższej jakości nośników magnetycznych wymaga ogromnej wiedzy w zakresie chemii i znajomości "Know-how".

W wielu dziedzinach firma BASF osiągnęła wiodącą w skali światowej pozycję poprzez wieloletnie badania i wprowadzanie nowych technologii.

Od wielu lat firma BASF jest kompetentnym partnerem handlowym Polski. Firma BASF posiada liczne, technologicznie dojrzałe rozwiązania problemów w dziedzinach chemikalii, barwników, środków pomocniczych, tworzyw, surowców i energii, produktów dla rolnictwa i wyrobów powszechnego użytku.

Produkty firmy BASF torują drogę w nowoczesnej chemii. Ważnym czynnikiem powodzenia jest owocna współpraca specjalistów nauk przyrodniczych i dyscyplin technicznych, od klasycznej chemii i techniki, przez fizykę, aż do biologii i medycyny. W 1987 roku wzrosły w firmie BASF nakłady na badania i rozwój ponownie o 6%, t. j. do 1.613 mln marek.

Punkty ciężkości rozwoju skierowane są na produkty powszechnego użytku, do których należą też nośniki magnetyczne.

Bezpieczeństwo i niezawodność

Szczególnie w przechowywaniu danych ważnym jest ich długotrwałe zachowanie. Zabezpieczenie na dłuższy czas cennych zbiorów danych jest bardzo ważnym wymogiem dla nauki i gospodarki.
Nośniki pamięci firmy BASF wykazują tu najwyższą jakość. Olbrzymie nakłady na badania i rozwój oraz własne nowoczesne miejsca produkcji zapewniają firmie pierwszeństwo w technologii zapisu danych.

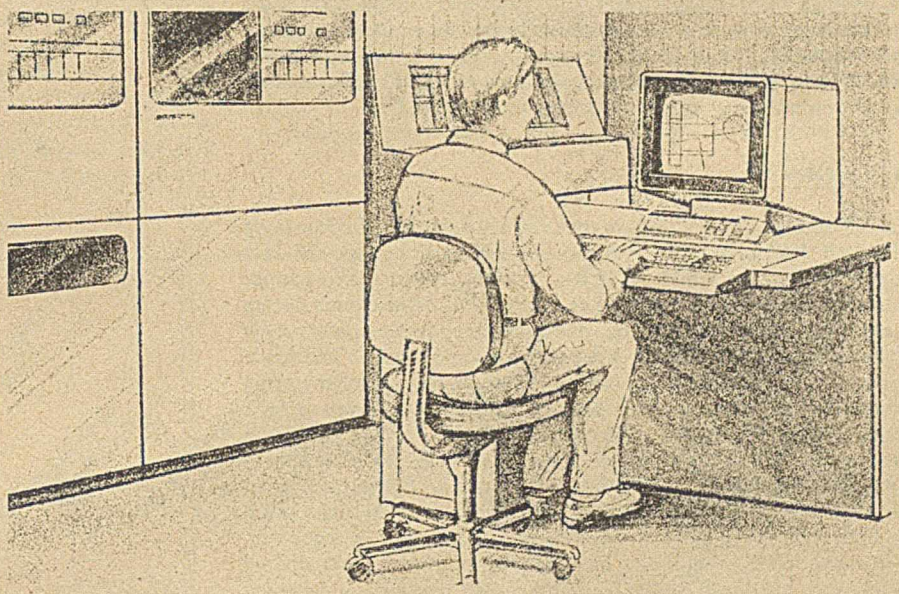
Od nośników danych, po systemy przetwarzania

Kompletne systemy przetwarzania danych dopełniają ofertę firmy BASF w dziedzinie przetwarzania danych.

Już w 1970 roku firma BASF była jedną z pierwszych, która wyszła na rynek z urządzeniami peryferyjnymi, kompatybilnymi co do łącza (tzw. PCM).
Od 1980 roku rozszerzono program na kompletne systemy produkowane w Japonii. Na rynku polskim proponujemy:

- Jednostkę centralną BASF 7/6X zajmującą powierzchnię tylko 1 m2 – najmniejszą jednostkę tej klasy z możliwością rozbudowy, spełnia wszystkie indywidualne wymagania użytkownika.
- Pamięci taśmowe, małe o zwartej budowie, dla których kontroler (sterownik) zintegrowany jest w pierwszym przewijaku łańcucha z możliwością pracy w trzech gęstościach zapisu (800/1600/6250 bpi).
- Drukarki wierszowe o dużej prędkości, małych wymiarach, zwartej budowie i wyjątkowo ciche. Prędkości od 1500 do 2000 linii/min. Osiągane w tej samej obudowie, modułowo rozbudowywalnej z modelu do modelu, z nadzwyczajną jakością druku, przy zastosowaniu taśmy stalowej z trzcionkami. Obecnie około 200 taśm z różnym rodzajem pisma.

Ponadto:
- System telekomunikacji lokalnej i zdalnej.



- Napędy dyskowe ze stałą głowicą (tzw. HDA) o pojemności do 317,5 MB na jednostkę.

Nasz system oparty na nowoczesnej technologii pozwala na oszczędność miejsca i małe zużycie prądu, jest odporny na wahania częstotliwości i napięcia zasilania. Doświadczenie firmy BASF i wypróbowana technika gwarantują niezawodną pracę systemu.

Uwaga!
Sprzęt firmy BASF jest kompatybilny nie tylko ze sprzętem IBM, ale także RIAD.

Elastyczne nośniki danych jak dyskietki, taśmy komputerowe, kasety i kasety Cartridge oraz kasety i pakiety dyskowe będące w programie sprzedaży firmy BASF. Ich niezawodna funkcjonalność działania podwyższa niezawodność całego systemu rejestracji danych.

Przedstawicielstwo BASF w Polsce:

LIM Centre
Al. Jerozolimskie 65/79
00-697 Warszawa

telefony: 300291-5
teleks: 816877 lub 816878
telefax: 300296

BASF

JAN GRZEGORZ CABAŃSKI,
MAREK LEWANDOWSKI,
STANISŁAW MARACH, ZBIGNIEW OCZKI
Ośrodek Badawczo-Rozwojowy
Elektronicznych Urządzeń Specjalizowanych
Toruń

ASTER – automatyczny sterownik transmisji szeregowych dla PC XT/AT

Komputery PC są wyposażone zwykle w jedno lub dwa łącza transmisji szeregowej RS 232C, znane jako urządzenia COM1 i COM2 w systemie operacyjnym DOS. Choć zadowala to wielu użytkowników, często zachodzi konieczność zastosowania kilkunastu lub kilkudziesięciu takich łącz. Wprawdzie dostępne są dodatkowe karty z czterema lub ośmioma łączami, jednakże użytkownik musi wówczas zapewnić programową obsługę sprzętu dla nadania lub odebrania każdego pojedynczego znaku, indywidualnie dla każdego łącza. Zaangażowanie procesora PC obsługą wielu transmisji szeregowych już przy średnich szybkościach transmisji poważnie ogranicza możliwości komputera. Narzuca się więc konieczność uwolnienia procesora PC od czasochłonnych obowiązków obsługi wielu transmisji szeregowych. Oczekiwania te realizuje prezentowany w tym artykule sterownik ASTER.

OPIS OGÓLNY

ASTER jest lokalnym modułowym mikrokomputerem zbudowanym na kartach PC, przeznaczonym do obsługi maksymalnie 56 łącz szeregowych. Pamięć RAM ASTER-a jest zrealizowana jako pamięć wspólna w przestrzeni adresowanej komputera PC. Dwustronny dostęp procesora PC i procesora lokalnego do tej pamięci pozwala na tworzenie oprogramowania sterownika w środowisku komputera PC. Te właściwości łącznie z rozkazami kasowania (*clear on/off*), wstrzymywania (*hold on/off*) i przerwań pracy procesora ASTER-a (*interrupt on/off*), stwarzają możliwości pełnego programowego sterowania lokalnym mikrokomputerem z komputera PC. Załadowane i uaktywnione oprogramowanie lokalnego mikrokomputera ma możliwość zgłaszania przerwań do PC, obsługi przerwań od PC, lokalnych przerwań zegarowych oraz przerwań nadajników i odbiorników transmisji szeregowych. Wymienione właściwości, przy niewątpliwie skromnej i optymalnej architekturze lokalnego mikrokomputera, nie ograniczają inwencji programisty w zakresie zastosowań sterownika. Tak więc, oprócz prezentowanej dalej sieci lokalnej Master NET, można realizować własne koncepcje wykorzystania sterownika ASTER, jak np. popularne w amerykańskich przedsiębiorstwach podsieci lokalne zwane automatycznymi przełącznicami danych ADS (ang. *automatic data station*). ADS łączą w sieć gwiaździstą, po łączach RS 232, kilkadziesiąt komputerów i urządzeń, często bardzo drogich, w celu ich wspólnego wykorzystania. Realne jest również dołączenie w ten sposób komputera PC do sieci teleksowej.

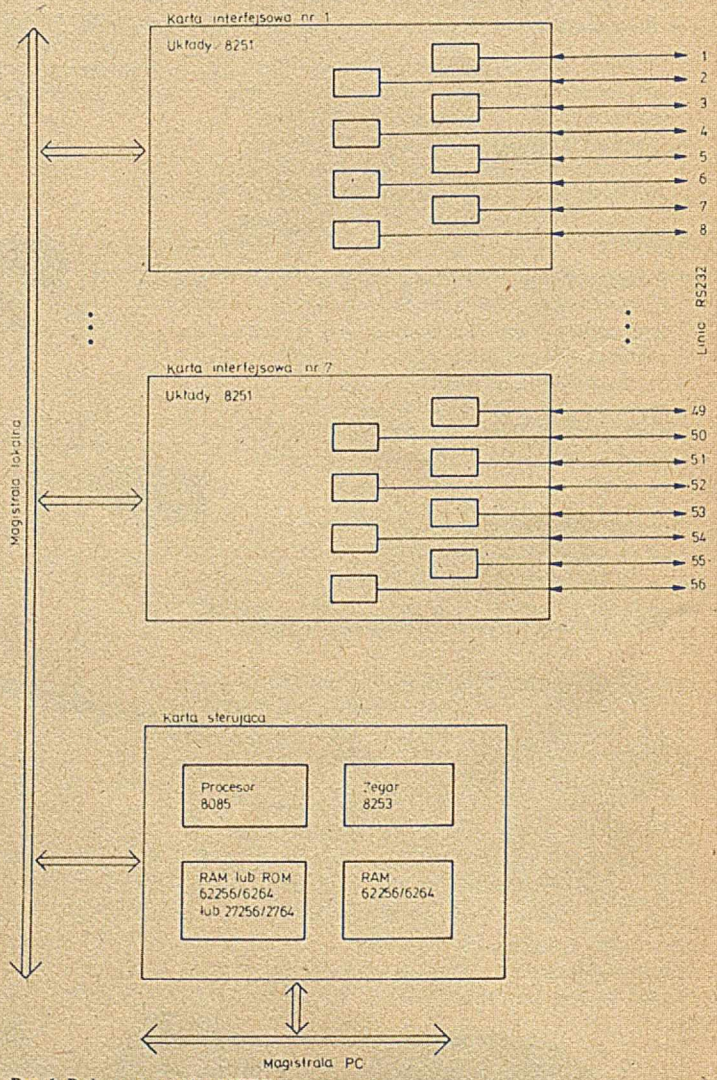
BUDOWA

Sterownik ASTER (rys. 1) ma budowę modułową i składa się z jednej karty sterującej i od jednej do siedmiu kart interfejsowych, zawierających po 8 łączy szeregowych. Możliwe jest również zainstalowanie w komputerze PC drugiego sterownika transmisji szeregowych ASTER, zwykle w oddzielnej obudowie.

Karta sterująca zawiera :

- procesor lokalny LP, Intel 8085, do automatycznej obsługi transmisji,
- programowy dzielnik częstotliwości Intel 8253 do programowej konfiguracji bazowych szybkości transmisji X, Y i czasu względnego,
- jeden lub dwa układy pamięci typu 62256 lub 6264 jako pamięć RAM o dwustronnym dostępie w przestrzeni adresowej komputera PC, do buforowania programu oraz danych.

Pojemność pamięci RAM jak i jej adres w przestrzeni adresowej PC są dostosowywane do wymagań użytkownika (rys. 2). Pamięć RAM bufora programu może być zastąpiona pamięcią ROM typu 27128 lub

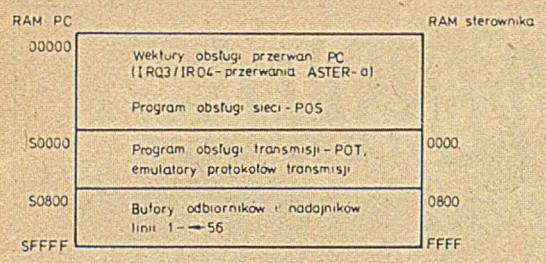


Rys. 1. Budowa sterownika ASTER

2764. Karta sterująca współpracuje z kartami interfejsowymi przez własną magistralę lokalną.

Karty interfejsowe zawierają po 8 układów transmisji napięciowych lub prądowych (0–20 mA), bez optoizolacji lub z optoizolacją (1 kV). Każdy z układów karty może transmitować dane z jedną spośród dwóch bazowych (X, Y) i trzech połówkowych ($X/2$, $X/4$, $X/8$) szybkości transmisji. Szybkości bazowe są ustalane programowo, a połówkowe wybierane sprzętowo, podobnie jak przyporządkowanie układom poszczególnych szybkości.

Oprogramowaniem lokalnego mikrokomputera ASTER jest modelowy program sieciowy MasterNET do obsługi na poziomie sprzętu i protokołu transmisji. Użytkownik PC widzi każdą linię transmisji jako 254-bajtowe bufory odbiornika i nadajnika we wspólnej pamięci RAM. Wysłanie komunikatu ogranicza się do wpisania jego adresu i treści do odpowiedniego bufora we wspólnej pamięci RAM, a odebranie – tylko adresu.



Rys. 2. Przestrzenie adresowe PC i ASTER-a (numer segmentu S = 8, 9, A, B, C, D, E)

Sieć MasterNET

Oparta na sterowniku Aster sieć MasterNET jest siecią otwartą, a w pewnym sensie – tylko strukturą sieci wypełnianą przez użytkownika dla określonego zastosowania. Niezmiennikami tej sieci są: jej topografia, uwarunkowania techniczne sterownika ASTER oraz dwuczłonowość oprogramowania. Oprogramowanie sieci stanowi:

– program obsługi sieci (POS) wykonywany przez procesor PC, opracowany zwykle przez użytkownika,

– program obsługi transmisji (POT) wykonywany przez procesor ASTER-a, opracowywany dla danej sieci i konfiguracji sterownika.

Kształt i funkcje programu POS zależą od konkretnego zastosowania. Program POT ma natomiast sprecyzowane zadanie: obsłużyć transmisje blokowe zlecane przez program POS oraz ewentualne transmisje międzyliniowe. Sposób zlecania transmisji może być różnoraki i sprowadza się do podania kierunku transmisji, adresu i długości bloku oraz protokołu i parametrów fizycznych transmisji. Najbardziej przejrzystą metodę uzyskuje się przy wspólnie ograniczonej długości bloku oraz stałych adresach buforów odbiorników i nadajników (p. tabela). Bez emulacji protokołu tekst nadawany lub odbierany umieszcza się w pewnej części bufora nadajnika-odbiornika, a jego długość równa się 256–*LB*; nadanie i odebranie bajtu *LB* = *FF* kończy transmisję przerwaniem do PC i wyzerowaniem bajtu *LB* tej linii. W trybie emulacji protokołu tekst nadawany lub odbierany umieszcza się od bajtu 02 i kończy binarnym 0; tekst nadawany jest uzupełniany, zaś dane odbierane – obcinane zależnie od protokołu danej linii. Koniec transmisji dekoduje emulator protokołu, sygnalizując to przerwaniem do PC i wyzerowaniem bajtu *LB*.

Bufory komunikatów

Nr linii	Adres	00	01	02–FF	00	01	02–FF
1	S0800	LB	HB	B01	LB	HB	BN1
2	S0A00	LB	HB	B02	LB	HB	BN2
...							
n	Sxx00	LB	HB	B0n	LB	HB	BNn
...							
56	S7E00	LB	HB	B056	LB	HB	BN56

S – numer segmentu równy 8, 9, A, B, C, D, E; xx – bardziej znaczący bajt adresu bufora komunikatorów równy 8+2×(n−1);

LB = 00 koniec transmisji, wolny nadajnik-odbiornik linii, 01 (otwarcie i programowanie linii), 02–FF (mniej znaczący bajt adresu tekstu nadawanego lub odbieranego);

HB – bardziej znaczący bajt adresu tekstu nadawanego lub odbieranego, równy przed transmisją 8–7F (wybór bufora nadajnika lub odbiornika linii), a po transmisji 0–7 (kod błędu), BOn i BNn – bufory odbiornika i nadajnika linii n;

Program obsługi sieci spełnia następujące funkcje:

- ładuje do bufora programu karty sterującej program POT,
- definiuje i modyfikuje parametry sieci,
- inicjuje zerowanie sprzętowe sterownika,
- aktywizuje procesor sterownika,
- zleca sterownikowi realizację transmisji szeregowych dla poszczególnych linii według wcześniej ustalonych parametrów (szybkość, protokół),
- obsługuje przerwanie od sterownika i zgłasza do sterownika.

dokończenie na s. 17

HENRYKA SOBOL
Instytut Podstaw Informatyki PAN
Warszawa

Współbieżność sterowania bazą wideodanych i systemów przetwarzania obrazów

W Laboratorium Architektury Systemów Komputerowych IPI PAN, podczas implementowania różnych algorytmów przetwarzania obrazów, powstał problem archiwowania wszystkich uczestniczących w przetwarzaniu danych (dalej nazywanych obiektami), obrazów źródłowych oraz obrazów przetworzonych, głównie w celu umożliwienia wnikliwej analizy i porównania rezultatów przetwarzania. Należy zwrócić uwagę na to, że dane typu obiektów charakteryzują się nieregularnością struktury oraz dużym zapotrzebowaniem na pamięć (obiekty mają wielkości do 1 MB, najczęściej jednak nie przekraczają 64 KB).

Problem ten rozwiązano w następujący sposób. Autorka, uczestnicząca w tworzeniu systemu przetwarzania obrazów (SPO), przeznaczonego do analizy efektywności działania implementowanych algorytmów, dobudowała do niego system sterowania bazą wideodanych (SSBWD) powstających w trakcie działania tych algorytmów. Przez wideodane rozumie się tutaj obiekty i ich charakterystyki w postaci parametrów przetwarzania oraz parametrów kompresji obiektu. SSBWD zawiera jedynie konieczną liczbę operacji.

WYBÓR SYSTEMU OPERACYJNEGO

Wspomniany SPO zaimplementowano na mikrokomputerze IBM PC/AT z systemem operacyjnym DOS 4.00. Wbudowanie operacji SSBWD praktycznie uniemożliwiło IBM dalsze rozszerzenie systemu o implementacje kolejnych algorytmów przetwarzania obrazów, ze względu na ograniczoną pojemność pamięci operacyjnej (640 KB). Wynikła stąd konieczność oddzielenia SSBWD od SPO, jednak bez utraty dostępu do operacji na bazie wideodanych. Na pojedynczym mikrokomputerze z jednozadaniowym systemem operacyjnym, takim jak PC-DOS, niemożliwe jest efektywne czasowo spełnienie tego warunku, bez użycia specjalizowanych drogich kart o standardzie EMS 4.0 (ang. *Expanded Memory System*). Należy wziąć również pod uwagę umożliwienie dostępu za pomocą sieci komputerowej do bazy wideodanych innym SPO przez nieblokowanie dostępu przez dłuższy czas. Odmiany systemu PC-DOS o nazwie Concurrent DOS, Multilink itp. umożliwiają pracę wielozadaniową, ale wprowadzają jednocześnie drastyczne zmniejszenie wielkości pamięci operacyjnej i to bez zapewnienia jej ochrony przed wzajemnym niszczeniem danych i programów. Każdy użytkownik praktycznie może zniszczyć całą pamięć operacyjną komputera. Praca w trybie sprzętowo kontrolowanej ochrony pamięci operacyjnej, wykorzystanie zwykłych kart rozszerzenia pamięci, pamięć wirtualna, wielozadaniowość i wygodny wielodostęp przemawiają za systemem operacyjnym Unix. Wybrano odmianę tego systemu o nazwie Xenix System V, który może być zainstalowany na komputerze klasy IBM PC/AT.

WSPÓŁDZIAŁANIE SYSTEMÓW SPO i SSBWD

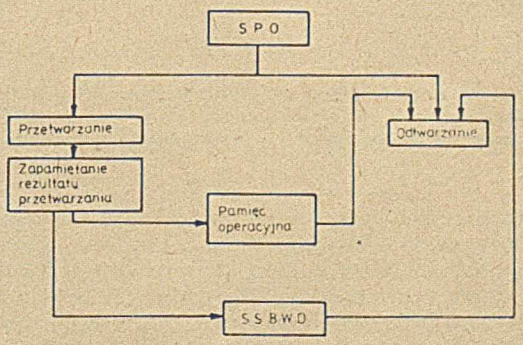
SPO składa się zasadniczo z dwóch części o nazwach: „przetwarzanie" i „odtwarzanie". W części „przetwarzanie" są zawarte kolejne implementacje algorytmów przetwarzania obrazów: w wyniku działania tych algorytmów otrzymuje się obiekty przetworzone. Nie zawsze konieczna jest natychmiastowa analiza rezultatu przetwarzania. Potrzeba taka istnieje jedynie na etapie uruchamiania implementacji danego algorytmu. Aby każde kolejne uruchomienie operacji przetwarzania nie spowodowało całkowitego zniszczenia rezultatu ostatniej, operacji niezbędne jest zapisanie do pliku tego rezultatu, a także parametrów przetwarzania. Część „odtwarzanie" może posłużyć się tak zapamiętanymi danymi umożliwiając wybór obiektu do odtworzenia na podstawie znanych parametrów przetwarzania. Operacja odtwarzania polega na wizualizacji rezultatów przetwarzania według określonego algorytmu. Najczęściej wizualizacja jest poprzedzona dekompresją obiektu.

W SSBWD zaimplementowano następujące polecenia działające na bazie wideodanych:
- wyświetlanie spisu obrazów oryginalnych w celu umożliwienia wyboru oryginału przetwarzania,
- poszukiwanie obrazu skomprymowanego (kompresja sprzętowa przez kartę graficzną VERTICOM), odpowiadającego wybranemu oryginałowi (obraz skomprymowany jest szybciej przekazywany z pamięci masowej na karcie graficznej),
- rejestrowanie nowego obiektu,
- usuwanie obiektów,
- poszukiwanie parametrów obrazu przetworzonego przeznaczonego do odtworzenia.

SSBWD jest interpreterem wymienionych poleceń, z których każde ma parametry wejściowe i wyjściowe. Ze względu ma modułową budowę SSBWD można łatwo rozszerzać o następne funkcje.

Współdziałanie systemów jest oparte na komunikacji procesów obliczeniowych za pomocą standardowych mechanizmów udostępnianych przez system operacyjny Xenix. SSBWD jest uruchamiany jako zadanie w tle czekające na wykonanie określonych poleceń. Zadanie główne to SPO, który przygotowuje zestaw poleceń dla SSBWD i zestaw parametrów wejściowych, jeśli są one konieczne. Po wykonaniu swojego zadania SSBWD przekazuje w razie konieczności parametry wyjściowe. Tak więc synchronizacja SSBWD i SPO jest oparta ze strony SSBWD na oczekiwaniu na zestawy poleceń od SPO, a ze strony SPO – na oczekiwaniu na parametry wyjściowe od SSBWD niezbędne do dalszej pracy lub inne efekty wizualne potwierdzające zakończenie pożądanej przez SPO operacji.

W systemie jednokomputerowym jednoczesne uruchomienia SSBWD i kilku SPO prowadzi do spowolnienia działania SPO (jest to spowodowane pętlą oczekiwania na zestaw poleceń w SSBWD), co znacznie wydłuża oczekiwanie na rezultaty przetwarzania. Jednak w porównaniu z systemem wielokomputerowym łatwiej jest zrealizować komunikację między dwoma procesami SSBWD i SPO działającymi na jednej pamięci operacyjnej. Spośród wielu możliwych sposobów komunikacji między procesami (wspólna pamięć, zmienne systemowe, przetwarzanie potokowe, pliki, synchronizacja za pomocą przekazywania sygnałów) najbardziej efektywną jest komunikacja za pomocą wspólnej pamięci (np. rys. 1), umożliwia to zastosowania tylko w systemie jednokomputerowym. Ale również i pozostałe sposoby komunikacji będą działały z większą szybkością w tym systemie, zwłaszcza jeśli wziąć pod uwagę to, że na naszym rynku brak jest szybkich sieci komputerowych pracujących z systemem operacyjnym Unix.



Rys. 1. Struktura współdziałania w parze systemów SPO i SSBWD; jeżeli istnieje potrzeba odtworzenia rezultatów ostatniego przetwarzania, to pośrednikiem jest pamięć operacyjna; poprzednie rezultaty przetwarzania są osiągalne tylko za pomocą SSBWD

W systemie wielokomputerowym każdy z systemów jest uruchamiany na innym komputerze, a komunikacja między nimi odbywa się za pośrednictwem plików przesyłanych przez sieć umożliwiającą dostęp wszystkich SPO do SSBWD. Takie podejście daje możliwość łatwego manipulowania SSBWD na kilku komputerach w sieci, z zachowaniem możliwości autokonfiguracji (procesy ustalają, gdzie należy wysyłać dane i skąd należy ich oczekiwać).

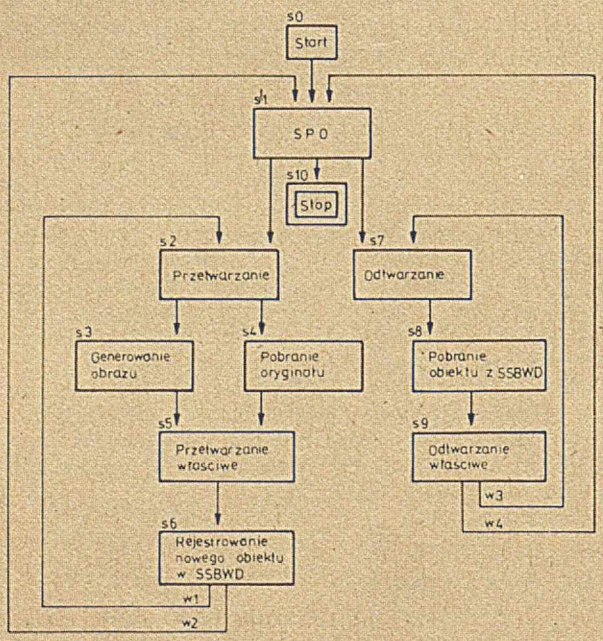
WYBÓR ROZWIĄZANIA I DYNAMIKA WSPÓŁDZIAŁANIA SYSTEMÓW

W początkowej wersji przyjęto rozwiązanie jednokomputerowe, w którym jeden SPO współdziała z SSBWD. Sprzyja to szybkiej synchronizacji obydwu system. SPO jako proces nadrzędny zawiesza proces SSBWD, uruchamiając go jedynie przed rozpoczęciem wykonywania operacji na bazie wideodanch. Jeżeli niezbędne będzie współdziałanie większej liczby SPO z SSBWD, to pozostałe SPO zostaną uruchomione w innych węzłach sieci. Każdy z SPO będzie miał możliwość zdalnego (przez sieć) wznawiania i wstrzymywania procesu SSBWD oraz będzie tworzył indywidualny plik poleceń i ich parametrów wejściowych, a w odpowiedzi będzie otrzymywał swój indywidualny plik z parametrami wyjściowymi. Dzięki takiemu podejściu do obsługi żądań poszczególnych SPO, SSBWD może traktować dany SPO uwzględniając z góry zadany lub zmieniany dynamicznie priorytet.

Obecnie SSBWD jest scentralizowany z zachowaniem w pełni możliwości decentralizacji, przez uruchomienie procesów na różnych komputerach. Zawiera on następujące części składowe:

- bazę wideodanych zawierającą informacje o adresie logicznym obiektu typu obrazu lub rezultatu przetwarzania oraz o parametrach przetwarzania;
- właściwy SSBWD;
- system kompresji i dekompresji jednorodnych grup obiektów;

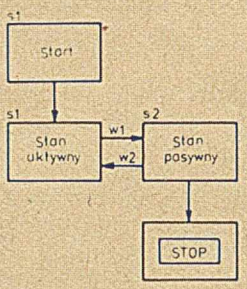
w skład grupy wchodzą tylko obrazy źródłowe lub tylko rezultaty przetwarzania określonego algorytmu; istnieje możliwość selektywnej dekompresji określonych plików ze skomprymowanej grupy za pomocą operacji ekstrakcji, a także operacja odwrotna – dodanie do skomprymowanej grupy określonych plików w postaci skomprymowanej.



Rys. 2. Graf stanów i przejść procesu obliczeniowego odpowiadającego dowolnemu SPO, gdzie *s0*, *s1*,..., *s10* – stany systemu, a *w1*, *w2*, *w3*, *w4* – oznaczają warunki zmiany stanu, tzn. *w1*, *w3* – decyzje kontynuacji przetwarzania lub odtwarzania, *w2*, *w4* – decyzje przejścia do odtwarzania lub przetwarzania

Na rysunku 2 przedstawiono graf procesu obliczeniowego dowolnego SPO, a na rysunku 3 – procesu SSBWD. Dynamika systemów w grafach jest opisana dla konfiguracji, w której kilka komputerów typu IBM PC/AT jest połączonych w sieć pod kontrolą systemu operacyjnego Xenix. Jeden z procesów SPO, oznaczony jako *P1* jest uruchomiony współbieżnie z SSBWD na tym samym komputerze, a pozostałe SPO, których procesy oznaczono jako *P2*, *P3*,... *Pn*, są uruchomione na pozostałych komputerach w sieci (każdy na innym). Warunki w_1–w_4 dotyczą decyzji użytkownika.

W celu skorzystania z funkcji SSBWD dowolny proces, *P1*, *P2*,..., *Pn*, sprawdza, czy SSBWD jest aktywny i jeśli tak, to czeka w kolejce, w przeciwnym razie uaktywnia SSBWD. Jednakże tylko *P1* ma prawo wymuszenia stanu pasywnego dla SSBWD bezwarunkowo, po skorzystaniu z funkcji SSBWD, w celu przekazania większej mocy obliczeniowej swojemu procesowi przetwarzania lub odtwarzania. Procesy *P2*,..., *Pn* powinny wywołać stan pasywny dla SSBWD tylko wtedy, gdy przed skorzystaniem z funkcji SSBWD zastały ten system w stanie pasywnym. W przeciwnym wypadku proces obliczeniowy *P1* otrzymałby dodatkowe obciążenie.

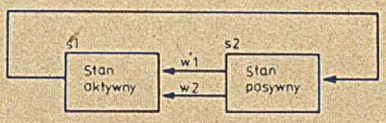


Rys. 3. Graf stanów (*s0*, *s1*, *s2*, *s3*) i istotnych warunków przejść (*w1*, *w2*) w procesie obliczeniowym PO SSBWD

Na rysunku 3 warunki mają następujące znaczenia:

w1 –	(*P1s3*	lub *P1.s5* lub *P1.P1.s9*)	lub
	((*P2. s3*	lub *P2. s5* lub *P2.s9*)	i *PX.s1*(w2)) lub
	((*Pn.s3*	lub *Pn.s5* lub *Pn.s9*)	i *PX.s1*(w2))
w2 –	(*P1.s4*	lub *P1.s6*	lub *P1s.8*) lub
	(*P2.s4*	lub *P2.s6* lub *P2.s8*)	lub
	(*Pn.s4*	lub *Pn.s6* lub *Pn.s8*),	

gdzie zapis *Pn.sn* oznacza stan o numerze *n* procesu o numerze *n*, a zapis *PX.s1*(*w2*) oznacza stan o numerze 1 wywołany warunkiem *w2* w procesie *PX*.



Rys. 4. Graf pomocniczy ilustrujący istotne warunki (*w1*, *w2*) przejścia do stanu aktywnego SSBWD, symbolicznie może on oznaczać proces PX

Rysunek 4 ilustruje dwa różne warunki przejścia do stanu aktywnego SSBWD:

w1 – pod wpływem procesu *P1*,
w2 – pod wpływem *P2* lub *P3* lub... *Pn*.

* * *

Przewidziano rozwój systemu w kierunku rozproszenia SSBWD na różne komputery w sieci według poniższej strategii. Na każdym z komputerów istnieje kopia ostatniej wersji bazy wideodanych zewnętrznych i lokalnych dla danego komputera oraz lokalny SSBWD rozszerzony o sterowanie przepływem informacji (o dokonanej modyfikacji w bazie) do pozostałych komputerów sieci.

Aby zapewnić jednolitość informacji w bazie wideodanych, przed przystąpieniem do modyfikacji, do wszystkich pozostałych lokalnych SSBWD jest rozsyłane zlecenie blokowania odpowiedniego fragmentu bazy. Modyfikacja następuje dopiero po otrzymaniu potwierdzeń blokowania. Po zakończeniu modyfikacji następuje rozesłanie polecenia odblokowania modyfikowanego fragmentu.

Niektóre obiekty w tak utworzonym wielokomputerowym systemie są powielone na innych komputerach, w celu uniknięcia częstego przesyłania ich przez sieć oraz zwiększenia niezawodności.

dokończenie na s. 31

INTERSOFT

00–478 Warszawa, Al. Ujazdowskie 18 m. 14
Tel. 28-01-76

Szanowni Państwo!

Polecamy duży wybór dokumentacji dot. komputerów IBM oraz oprogramowania, m in.:

Przewodnik programisty IBM	144 000 zł
Wprowadzenie do komputerów IBM	32 000 zł
Sidekick	50 000 zł
Wstęp do grafiki (Basic, Turbo, Graphics)	60 000 zł
Autocad v.2.17	86 000 zł
Lotus 1–2–3 v.2.0	100 000 zł
Framework IIp	114 000 zł
System operacyjny DOS 4.0	160 000 zł
System operacyjny DOS 3.3	144 000 zł
Turbo Basic	130 000 zł
Turbo „C" v.2.0	145 000 zł
Aztec „C"	132 000 zł
Zastosowanie języka C dla zaawansowanych	160 000 zł
Turbo Graphics (do TP v.3.0)	74 000 zł
Turbo Pascal v.4.0	43 000 zł
Turbo Pascal v.5.0	270 000 zł
Turbo Pascal v.5.5	306 000 zł
Turbo Power Tools Plus (procedury do TP 4/5)	115 000 zł
Grafika TP v.4.0 i TC v.1.5	110 000 zł
Clipper 87, kompil. do dBase III+	160 000 zł
dBase III+	170 000 zł
dBase III+, praca w sieci	40 000 zł
Fox Base+	140 000 zł
Programowanie w assemblerze	160 000 zł
Eureka	65 000 zł
Poly-Windows	48 000 zł
Instrukcja obsługi PC 1512	150 000 zł
Wordstar 2000	50 000 zł
Modula 2 Logitech	102 000 zł
Informix	180 000 zł
OS-2, opis systemu	130 000 zł
Or-Cad	170 000 zł
Novell, podr. użytkownika i instalatora	180 000 zł

Do większości pozycji dołączamy dyskietki z przykładami (koszt nośnika nie jest ujęty w wyżej wymienionych cenach).

Prowadzimy sprzedaż wszelkiego sprzętu komputerowego po bardzo konkurencyjnych cenach.

Do sprzedawanego przez nas sprzętu dodajemy bezpłatnie pakiety dokumentacji i oprogramowania o wartości zależnej od wielkości sprzedaży.

ZAPRASZAMY!

EO/232/89

KATARZYNA ŁUBIŃSKA
Instytut Maszyn Matematycznych
Warszawa

Rola Ady w wytwarzaniu oprogramowania

W latach osiemdziesiątych Departament Obrony USA, największy na świecie użytkownik oprogramowania, stanął przed koniecznością rozwiązania kilku nieoczekiwanych problemów. Po pierwsze, okazało się, że koszty oprogramowania od roku 1980 (6,07 miliardów dolarów) do roku 1990 wzrosną do 37,99 miliardów dolarów i wzrost ten będzie przebiegał krzywą potęgową. Wielkość nakładów na produkcję oprogramowania przewyższy znacznie wielkości nakładów przeznaczonych na produkcję sprzętu komputerowego.

Po drugie, zmieniła się struktura produkowanego oprogramowania. Systemy wbudowane już w połowie lat siedemdziesiątych stanowiły 58% oprogramowania wyprodukowanego na zamówienie Departamentu Obrony (przetwarzanie danych 19%, naukowe 19%, pozostałe 20%) i udział procentowy tych systemów stale rośnie. Są to systemy taktyczne, komunikacyjne, używane w lotnictwie i marynarce. Przeciętnie zawierają 50–100 tysięcy wierszy. Angażują przy wytwarzaniu i pielęgnacji duże zespoły ludzi, złożone nie tylko z programistów, lecz także ze specjalistów z dziedziny zastosowania i dziedzin pokrewnych.

Po trzecie, zmieniła się struktura wykorzystania dużych programów. Zwiększył się okres ich użycia do około 10–15 lat (dane z 1984 roku). Użytkownicy dużych systemów stanęli przed problemem wzrostu kosztów pielęgnowania programów, które zaczęły być porównywalne z kosztami wyprodukowania oprogramowania. Powstał również problem testowania produktów programowych, którego nie można wykonać w warunkach rzeczywistych, przy użyciu rzeczywistej konfiguracji sprzętu. Występuje to, na przykład, w testowaniu systemów medycznych sterujących zabiegami, systemów militarnych, kosmicznych itp.

Wyprodukowanie takiego oprogramowania przy użyciu dotychczas stosowanych języków programowania i metod inżynierii oprogramowania jest trudne, wymaga ogromnego nakładu kosztów. Systemy zawierają dużo błędów, które usuwane w trakcie eksploatacji programów, często powodują powstanie nowych błędów.

Dotychczasowe rozwiązania, polegające na swobodnym rozwoju języków programowania na wszystkich komputerach osobno, mają wiele wad. Przede wszystkim bardzo duże są koszty przenoszenia oprogramowania. Programy napisane w danym języku (lub językach) programowania są dostępne tylko na komputerze, na którym były pisane. Koszt wyprodukowania takiego systemu nie procentuje przy produkcji dokładnie takiego samego systemu na innym komputerze. Jest to szczególnie widoczne przy programach napisanych w językach asemblerowych, Fortranie i językach specyficznych dla pewnych komputerów. Języki wysokiego poziomu również nie zapewniają przenośności oprogramowania. Trudne jest przenoszenie operacji wejścia-wyjścia, a także wszelkich rozszerzeń udostępniających z języka wysokiego poziomu możliwości sprzętu.

Znaczące są też koszty rozwoju oprogramowania narzędziowego, kompilatorów i środowiska programowego wszystkich używanych języków. Każdy rodzaj komputera, a często każda wersja ma oryginalne kompilatory wszystkich języków użytkowych na tym komputerze. Każdy kompilator wymaga dodatkowo specjalnie przygotowanego środowiska. Sytuację komplikuje fakt, że w jednym komputerze istnieje często kilka kompilatorów tego samego języka i powstają nowe – lepsze. Nie można oczywiście składać systemów z modułów napisanych w języku akceptowanym przez różne kompilatory.

Podejmuje się lokalne próby ujednolicenia oprogramowania dla danego komputera lub danej generacji komputerów, na przykład, wytwarzanie kompilatorów kilku języków z tym samym generatorem kodu (Microsoft), kontynuacja rozwoju jednego kompilatora przy wprowadzaniu nowej generacji komputera (firma DEC), ale są to tylko częściowe rozwiązania tego problemu. Z zagadnieniem tym wiąże się zwiększenie kosztów użytkowania nowych typów komputerów oraz opóźnienie ich pojawienia się na rynku ze względu na konieczność wyprodukowania nowych kompilatorów wraz ze środowiskiem.

Niedoceniane były również problemy pielęgnowania programów. Długi czas użytkowania systemów połączony ze zwiększonym stopniem ich złożoności postawił twórcom tych systemów oraz twórcom oprogramowania narzędziowego dodatkowe wymagania. Regułą jest, że zespoły pielęgnujące duże systemy nie są ich twórcami. Budowa systemu powinna więc być jasna, konsekwentna, często nawet kosztem optymalności. Wykluczone jest programowanie tzw. „trickowe", które aczkolwiek efektowne, wyklucza modyfikowanie i pielęgnowanie programu.

Języki stosowane do tworzenia złożonych programów powinny być łatwe do odczytywania, powinny mieć językowe mechanizmy podziału programu na moduły. Program powinien być doskonale zaprojektowany, dobrze udokumentowany, a poza tym jego struktura powinna być wiernym odbiciem projektu. Ze względu na długi czas użytkowania konieczna jest łatwa modyfikowalność, a także ochrona przed wprowadzaniem błędów podczas modyfikowania.

Ada jako remedium na kryzys

Ada jest pierwszym językiem programowania, w którym istnieją językowe mechanizmy do podziału programu na oddzielnie kompilowane jednostki, przy czym jednostki te mają dobrze (językowo) określone sprzężenia z innymi jednostkami. Kompilator czuwa nad poprawnością nie tylko izolowanych jednostek, ale także nad poprawnością komunikowania się między nimi. System bibliotek, należący do podstawowego wyposażenia środowiska programowego każdego kompilatora Ady, czuwa także nad kolejnością kompilacji i aktualnością poszczególnych modułów. Rozwiązanie to przyjęło się jako element dodawany w innych językach programowania (Pascal, C).

Rozbudowany system kontroli struktur danych narzuca programiście pewien reżim w posługiwaniu się obiektami, co ułatwia projektowanie, pielęgnowanie i uruchamianie programu, ale utrudnia kodowanie. Kodowanie jednak stanowi tak niewielki składnik w kosztach produkcji oprogramowania, że trudności w tej fazie można uznać za pomijalne. Struktury danych niemal całkowicie uniezależniają projekt i realizację systemu od komputera. Nawet operacje arytmetyczne zarówno na literałach, jak i na zmiennych rzeczywistych są niezależne od arytmetyki komputera. Arytmetykę można zdefiniować w programie i używać jej bez wiedzy na temat ograniczeń implementacyjnych.

W Adzie istnieje możliwość generowania i obsługiwania sytuacji wyjątkowych (ang. *exception handling*), co znakomicie ułatwia projektowanie i pielęgnowanie systemów wbudowanych. Do realizowania takich systemów przydaje się również możliwość projektowania procesów równoległych, których sposób komunikowania się i zarządzania wspólnymi zasobami jest kontrolowany przez kompilar.

Bardzo istotna jest w Adzie metoda realizacji wejścia-wyjścia. Jest ono zunifikowane i zmiany implementacyjne w definicji pakietów wejścia-wyjścia są niedopuszczalne. Jest to istotnie nowa wartość w definiowaniu języków wysokiego poziomu; dotychczasowe próby ujednolicenia instrukcji wejścia-wyjścia w poszczególnych językach nie powiodły się z różnych powodów.

Kompilatory Ady przed oddaniem ich do użytku, są atestowane, tzn. sprawdzane przez specjalnie powołany urząd, zarówno pod kątem pełności akceptowanego języka, jak i pod kątem poprawności realizacji kompilatora (ang. *validation*). Kompilatory nie dopuszczone przez ten urząd nie mają prawa nazywać się kompilatorami Ady. Wymóg ten ma na celu zagwarantowanie przenośności programów napisanych w tym języku na różnych komputerach.

Ada, ze względu na swoje zalety i ze względu na naciski Departamentu Obrony USA – na jej powszechne używanie, miała stać się najpopularniejszym językiem programowania. Departament ten postulował nawet, że nie będzie przyjmował prac pisanych w innych językach, co musiało mieć znaczenie na międzynarodowym rynku programistycznym. Duże były też nakłady na produkcję kompilatorów i środowiska programowego Ady.

Wykorzystanie Ady

Ada nie spełniła jednak oczekiwań Departamentu Obrony i swoich twórców. Nie stała się ani jedynym, ani nawet najpopularniejszym językiem programowania. Dość łatwo środowisko programistów przyjęło Adę jako standard dokumentacyjny, nie zaakceptowało jej jednak jako języka implementacyjnego.

Złożyło się na to wiele czynników. Pierwszym i łatwym do przewidzenia była naturalna niechęć programistów do przestawienia się na inny język programowania, choćby był najlepszy. Tym bardziej, że dotychczas większość programów na zamówienie Departamentu Obrony USA była pisana w Fortranie i Cobolu. Ada znacznie odbiega od poziomu tych języków, jest ponadto językiem dla profesjonalistów, trudnym w przyswojeniu i używaniu. Systemy wbudowane wymagają współpracy wielu programistów. Każdego należałoby przeszkolić, co zwiększa koszty i wydłuża czas produkcji systemów, wymaga też odpowiedniej kadry do przeszkolenia programistów.

Sytuację skomplikował fakt, że kompilatory Ady nie dawały się zrealizować przy użyciu standardowych metod stosowanych do produkcji kompilatorów języków wysokiego poziomu. Kompilatory Ady odbiegają wielkością i stopniem złożoności od kompilatorów innych języków. Pierwsze kompilatory Ady pojawiły się wiele lat po ustaleniu standardu języka, były bardzo drogie i trudno dostępne.

W latach osiemdziesiątych nastąpiła na rynku komputerowym inwazja komputerów osobistych. Umasowienie komputerów i ogromne zapotrzebowanie na małe „sprytne" programy, zmieniło nieco strukturę rynku oprogramowania. Problemy inżynierii oprogramowania zeszły na drugi plan wobec tendencji do produkcji programów dowolnej jakości, byle szybciej można było wprowadzić je na rynek (na przykład oprogramowanie firmy Borland na IBM PC). Ada nie była stworzona do takiej specyfiki zapotrzebowania. Poza tym kompilatory Ady są dość trudne do implementowania na małych komputerach, nie realizują też efektywnego kodu, co na mikrokomputerach zaczęło znowu być znaczące.

Brak ściśle ustalonego standardu na środowisko kompilatora Ady również mści się na użyteczności tego języka. Warstwowa struktura środowiska narzucona przez standard Stoneman nie zapewnia szybko takich możliwości, jak naprędce stworzone otoczenia kompilatorów wielu innych języków. Istotny jest również brak konkretnej normy – Stoneman podaje jedynie filozofię konstrukcji środowiska kompilatora Ady.

Środowisko programowe Ady

Środowisko programowe danego języka jest to zbiór narzędzi do projektowania, testowania i pielęgnowania produktów w danym języku. Środowisko zawiera też narzędzia umożliwiające użytkowanie pozajęzykowych możliwości komputera (biblioteki użytkowe).

Z założenia, środowisko Ady ma strukturę warstwową, która została ustalona ze względu na przenośność środowiska, co jest niezbędne dla przenośności produktów programowych. Składa się ono z następujących warstw.

- KAPSE jest to warstwa bezpośrednio otaczająca system, w którym jest umieszczona. Zawiera bazę danych i system wykonawczy (ang. *run-time*). Oddziela część przenośną od nieprzenośnej.
- MAPSE zawiera minimalny zestaw narzędzi w Adzie do opracowania programów. Jest to:
  - edytor,
  - program drukujący i formatujący,
  - translator,
  - konsolidator tworzący kod ładowalny,
  - programy ładujące,
  - analizator odwołań (statycznych i dynamicznych),
  - administrator plików,
  - program zarządzający bazą danych,
  - język dostępu do elementów bazy (najlepiej adopodobny; obecnie najczęściej jest to JCL systemu, w którym działa kompilator, ale są już zintegrowane systemy posługiwania się środowiskiem).
- APSE zawiera dodatkowe narzędzia stosowane podczas całego okresu istnienia produktu, a także ukierunkowane na konkretne zastosowania programu użytkowego.

Obecnie działa wiele typów środowiska programowego poszczególnych kompilatorów Ady (na przykład ALS, AIE lub znane w Polsce środowisko kompilatora Ady-M na IBM PC). Przenośność programów napisanych przy użyciu poszczególnych środowisk jest utrudniona. Bez standaryzacji, przede wszystkim na poziomie KAPSE, nie może być mowy o przenośności programów w Adzie, a przecież wiele elementów języka zostało stworzonych głównie w tym celu.

Okres istnienia produktu programowego

Ada to nie tylko język programowania, to także krok naprzód w inżynierii programowania. W Adzie wprowadzono niektóre nowe pojęcia z dziedziny języków programowania, na przykład przeciążanie (ang. *overloading*), takich samych nazw operacji lub obiektów o różnym typie i właściwościach.

Przy pracach nad Adą wprowadzono pojęcie okresu istnienia programu, w którym produkt programistyczny jest traktowany jako całość od chwili postawienia problemu przez użytkownika, do końca użytkowania gotowego produktu (ang. *life cycle*). Można wydzielić zasadnicze fazy tworzenia produktu programistycznego.

Analiza problemu. Faza ta obejmuje określenie natury problemu, uszczegółowienie sformułowania problemu. Faza ta jest niezależna od implementacji.

Definiowanie potrzeb. Faza ta określa wymagania dotyczące narzędzi programowych, systemu, sprzętu komputerowego i wyspecjalizowanego, koniecznego do realizacji i użytkowania systemu.

Projektowanie. W tej fazie tworzy się strukturę systemu, określa użyte algorytmy, dzieli się projekt na jednostki i określa sposób komunikacji między nimi. Ada zawiera wszystkie mechanizmy niezbędne do projektowania i przez to staje się dobrym językiem projektowania. W fazie projektowania można dzięki temu wykorzystywać mechanizmy środowiska, co zbliża fazę projektowania do implementacji.

Kodowanie. Obejmuje ono wszystkie czynności polegające na określaniu struktur danych i czynności na nich wykonywanych w języku implementacji. Faza ta trwa do końca okresu istnienia produktu. Kodowanie, jego łatwość i efektywność zależy od jakości środowiska. Podczas kodowania można przy użyciu odpowiednich edytorów zapobiec wielu błędom, nadać tekstowi programu odpowiednią postać graficzną, co potem ułatwia pielęgnowanie produktu.

Testowanie. Jest to sprawdzanie poprawności zarówno osobnych modułów, jak i całości implementowanego produktu programistycznego. Testowanie jest niezbędne przez cały okres uruchamiania i użytkowania programu. Szczególną uwagę zwraca się na testowanie w czasie modyfikowania gotowego, użytkowanego produktu. Często poprawki powodują błędy w nieoczekiwanych miejscach programu i sprawiają, że nowa wersja produktu jest gorsza od tej sprzed poprawek (regresja programu). W tej fazie również istotne są stosowane elementy środowiska APSE. Obsługa bibliotek programowych jest narzędziem niezbędnym i bardzo pomocnym przy testowaniu. Pozwala, na przykład, unikać błędów wynikających ze stosowania nieaktualnych modułów. Testowanie może ułatwić również symboliczny program uruchomieniowy oraz inne elementy wspomagające.

Użytkowanie i pielęgnowanie. W czasie użytkowania produktu programistycznego muszą, ze szczególną intensywnością na początku, wystąpić pewne błędy, niedociągnięcia lub sytuacje nieprzewidziane w projekcie. Powinny one być usunięte bez spowodowania regresji produktu. Przy długim okresie istnienia programu należy liczyć się z bardzo kosztownymi zmianami, dotyczącymi różnic zarówno w funkcjach systemu, jak i w konfiguracji sprzętowej. W fazie projektowania i kodowania należy przewidzieć możliwości modyfikacji, choć oczywiście nie wszystko da się przewidzieć. APSE może zawierać elementy środowiska ukierunkowane na pielęgnację gotowego produktu.

dokończenie na s. 25

MIECZYSŁAW KŁOPOTEK, ADAM KOWALSKI, ROBERT MACURA, MACIEJ MICHALEWICZ, ANDRZEJ PACAN, ELŻBIETA SYROPIATKO, SŁAWOMIR WIERZCHOŃ
Instytut Podstaw Informatyki PAN
Warszawa

SYS8688 – system pozyskiwania i weryfikacji wiedzy medycznej (2)

W pierwszej części artykułu przedstawiono założenia koncepcyjne i architekturę opracowanego przez autorów systemu SYS8688, a także omówiono dokładniej dwa spośród czterech jego elementów – system bazy danych i system bazy wiedzy. Poniżej zostaną omówione pozostałe elementy systemu, zakres jego stosowania i dotychczasowe doświadczenia z realizacji.

SYSTEM EKSPERTOWY

Jądrem systemu jest „szkieletowy" system ekspertowy SYS8688/SE. W jego skład wchodzą:
- pamięć robocza (baza faktów),
- moduł wnioskujący (interpreter reguł),
- moduł generowania objaśnień (MGO).

Pamięć robocza stanowi lokalną bazę danych SE, przechowującą fakty trzech rodzajów:
1) początkowe dane o pacjencie,
2) dane uzyskane w wyniku przeprowadzonej dedukcji,
3) konkluzje SE będące celem konsultacji.

Dane pierwszych dwóch rodzajów są przechowywane w postaci czwórek ⟨O,A,W,WP⟩ oraz piątek ⟨O,A,W,WP,T⟩. W celu uniknięcia redundancji w przypadku atrybutów czasowych fakt postaci ⟨O,A,W⟩ (np. temperatura pacjenta jest wysoka) jest pamiętany tylko raz, a każde jego wystąpienie w czasie jest dołączane do listy, której elementami są pary ⟨WP,T⟩. Pamięć robocza może być zainicjowana na podstawie informacji o pacjencie przechowywanej w BD systemu lub przez użytkownika SE, dysponującego informacją uzyskaną podczas wywiadu z pacjentem, jego oględzin, badań laboratoryjnych itp. Zbiór konkluzji stanowi przestrzeń rozwiązań, w której SE wyszukuje jedno bądź kilka rozwiązań uzasadniających aktualny stan chorobowy pacjenta.

Interpreter reguł pełni funkcję struktury sterującej, która decyduje o kolejności sprawdzania reguł. W systemie zastosowano strategię wnioskowania wstecz (ang. *backward reasoning*) [1]. Przebieg procesu wnioskowania jest analogiczny jak w systemach produkcji, z wyjątkiem dopasowywania warunków zawierających atrybuty czasowe. Przykładem tego rodzaju dopasowania może być sprawdzenie, czy w określonym przedziale czasu pacjent gorączkował. Wymaga to przejrzenia listy temperatury skojarzonej z badanym pacjentem i wyselekcjonowania wycinka spełniającego dane ograniczenie czasowe. Jeżeli wycinek zawiera przynajmniej jeden odczyt odpowiadający wysokiej temperaturze, to prawdziwość sprawdzanego warunku zostaje potwierdzona. Jeżeli nie – to warunek nie jest spełniony.

Na dynamikę procesu wnioskowania wywiera również wpływ przyjęty w systemie probabilistyczny model oceny niepewności [7]. Model ten zakłada istnienie dwóch rodzajów niepewności (subiektywizmu), pochodzącej z dwóch różnych źródeł:
- niepewności statycznej, wynikającej z subiektywizmu wiedzy eksperta, uwzględnianej w systemie w postaci WP skojarzonych z regułami,
- niepewności dynamicznej, zależnej od danych o konkretnym pacjencie, wprowadzanej przez użytkownika w postaci WP skojarzonych z faktami.

Jeżeli dopasowanie przesłanki reguły do aktualnego stanu pamięci roboczej zakończyło się sukcesem, to jest wyznaczany finalny WP reguły jako iloczyn WP przesłanki i WP reguły. WP przesłanki jest obliczany zgodnie ze schematem Shortliffe'a [7], według następującej zasady: WP koniunkcji warunków jest równy minimalnemu WP, a WP alternatywy warunków jest równy maksymalnemu WP. Warunkiem akceptacji dopasowanej reguły jest przekroczenie przez WP jej przesłanki pewnej wartości progowej, zadawanej przez użytkownika na początku konsultacji.

Po zakończeniu procesu wnioskowania system przedstawia listę konkluzji uporządkowaną w kolejności malejących WP. Z każdą konkluzją jest skojarzona odpowiednia sugestia terapeutyczna, określająca typowe postępowanie dla danego przypadku. Użytkownik może również wywołać moduł MGO i w ten sposób dokładnie prześledzić tok rozumowania systemu. Szczegółowe uzasadnianie otrzymanych rezultatów ma istotne znaczenie, ponieważ:
– wydatnie zwiększa zaufanie użytkownika do systemu,
– likwiduje wszelkiego rodzaju niejasności wynikające z różnic w terminologii w metodach leczenia itd.,
– może okazać się pomocne w prowadzeniu zajęć o charakterze dydaktycznym.

SYSTEM POZYSKIWANIA WIEDZY

W praktyce medycznych systemów ekspertowych uzyskiwanie reguł od ekspertów jest trudne. Zwykle obawa i ostrożność ekspertów są tak dalece posunięte, że proces tworzenia reguł jest długotrwały, a uzyskany zbiór reguł – kontekstowo niepełny. Co więcej, w nowych działach medycyny nie można mówić o istnieniu pełnej wiedzy nadającej się do formalizacji; naturalny proces generowania wiedzy od obserwacji precedensów do uogólnień jest tu oczywisty. Jednak uogólnienia wynikające z dostępnego materiału są zazwyczaj trudne do wykonania, a otrzymane sugestie wymagają dalszych obserwacji i weryfikacji.

SYS8688 wspomaga proces tworzenia BW oferując użytkownikowi dwa zestawy narzędzi:

- narzędzia przeglądania i testowania aktualnego stanu BW,
- narzędzia pozyskiwania wiedzy.

Cechą charakterystyczną pierwszego zestawu jest to, że jego użycie nie powoduje trwałych zmian w BW. Podstawowym zadaniem tego zestawu jest pomoc w wyszukiwaniu takich miejsc w BW, które wymagają modyfikacji bądź uzupełnienia ze strony eksperta. Zestaw drugi wspomaga proces wprowadzania wiedzy do BW systemu, przy czym poszczególne narzędzia różnią się między sobą stopniem przetworzenia wiedzy eksperta w momencie ich wykorzystania.

Narzędzia przeglądania i testowania BW

Jeden z wielu możliwych scenariuszy może wyglądać następująco. Analizując przypadki, dla których z góry znana jest diagnoza, ekspert stwierdził niepoprawność rozwiązania uzyskanego przez SE. Powstaje zatem konieczność rozpoznania przyczyny niezgodności albo – mówiąc inaczej – ustalenia tzw. kontekstu błędu [2]. W tym celu ekspert wywołuje moduł MGO, pozwalający przeglądać drzewa decyzyjne związane z każdą potwierdzoną przez SE konkluzją. MGO informuje eksperta, które reguły i fakty zostały użyte na różnych poziomach dedukcji, podaje ich WP, teksty reguł itd. Po przeanalizowniu tych informacji ekspertowi nasuwa się przypuszczenie co do przyczyny

nieprawidłowego funkcjonowania SE. W celu całkowitego upewnienia się formułuje kwerendy do BW zadawane w postaci:
- pełnej reguły,
- tylko lewej lub tylko prawej części reguły,
- zbioru atrybutów i konkluzji.

Po ustaleniu kontekstu błędu (np. brak reguły, niepełna bądź niewłaściwa postać istniejącej reguły, nieprawidłowy WP) ekspert dokonuje niezbędnych modyfikacji w BW, wykorzystując w tym celu jedno z narzędzi pozyskiwania wiedzy. Końcowym etapem jest sprawdzenie, na ile wprowadzone zmiany wpłynęły na jakość funkcjonowania SE. W tym celu ekspert ponownie analizuje rozważany przypadek i – jeśli zauważona nieprawidłowość nie została usunięta – omawiany proces powtarza się.

W celu ustalenia zgodności lub niezgodności reguł z istniejącą w BD informacją o pacjentach, ekspert może posłużyć się kwerendami do BD. Warunek selekcji pacjentów powinien zawierać te same atrybuty co przesłanka reguły, a w części informacyjnej kwerendy powinny znajdować się wnioski reguły. Stwarza to szansę konfrontacji reguł zgromadzonych w BW (lub nowych propozycji) z aktualnym stanem BD. Ponieważ jednym z elementów odpowiedzi na kwerendę jest liczebność zbioru części informacyjnej, ekspert może ocenić istotność zjawiska, jego występowanie w danej populacji itd. Może także posłużyć się dostępnym aparatem statystycznym w celu obiektywizacji własnego spojrzenia.

Narzędzia pozyskiwania wiedzy

SYS8688 oferuje cztery metody pozyskiwania wiedzy:
- bezpośrednie wprowadzanie gotowych reguł wnioskowania przez eksperta,
- analizę supozycji, weryfikację i uzupełnianie częściowych propozycji reguł na podstawie BD,
- automatyczne generowanie reguł na podstawie materiału z BD,
- obróbkę statystyczną materiału z BD.

Bezpośrednie wprowadzanie gotowych reguł odbywa się przy użyciu edytora sterowanego składnią reguł, który wchodzi w skład SZBW.

Moduł analizy supozycji. Lekarz zauważa związki między współistniejącymi lub następującymi po sobie objawami, często jednak nie potrafi w pełni sprecyzować swoich spostrzeżeń. Moduł analizy supozycji (*MAS*) – opierając się na intuicji, doświadczeniu i wiedzy lekarza oraz nagromadzonych danych o pacjentach – wspomaga proces tworzenia nowych reguł. *MAS* przyjmuje sugestie dotyczące kształtu przyszłej reguły w postaci reguły niepełnej (tzw. supozycji) [4].

Definicja syntaktyczna supozycji jest wzorowana na składni reguły wnioskowania, różni się tylko występowaniem elementu niewiadomego. W zależności od niepełności informacji zawartej w supozycji można wyróżnić:
- supozycje pierwszego stopnia (nie ma podanego WP),
- supozycje drugiego stopnia
  - prawostronne, gdy nie jest podana wartość atrybutu występującego po prawej stronie,
  - lewostronne, gdy nie są zdefiniowane wartości atrybutów występujących po lewej stronie.

Ze względu na rodzaj wniosku supozycje – podobnie jak reguły – można podzielić na terminalne (po prawej stronie występuje konkluzja) lub nieterminalne (po prawej stronie występuje atrybut). Przykładem proponowanej przez eksperta niepełnej reguły może być następująca:

IF poziom_leukocytów > 9000 **AND**
  kolor_rany = ? **AND**
  bolesność_rany = ? **AND**
  ALL temperatura [**TIME** > moment_oparzenia] = wysoka
THEN

  zakażenie_rany =, ?

MAS przekształca supozycje drugiego stopnia na listę supozycji pierwszego stopnia, podstawiające w miejsce nieznanych wartości atrybutów kombinacje dopuszczalnych ich wartości. Następnie – na postawie materiału zgromadzonego w BD – dla każdego elementu tej listy jest przeprowadzana analiza, potwierdzająca lub odrzucająca badaną postać reguły. W tym celu supozycja pierwszego stopnia jest zamieniana na zbiór supozycji atomowych, czyli supozycji, których przesłanki zawierają tylko jeden warunek. Wyznaczany jest WP każdej supozycji atomowej, a następnie – sumaryczny WP supozycji początkowej przez złożenie WP supozycji atomowych (zgodnie ze schematem Shortliffe'a dla składania reguł o tym samym wniosku [7]). Reguły, które znalazły potwierdzenie (mają WP różny od 0) u wystarczającej lub wyspecyfikowanej liczby pacjentów, są przedstawiane ekspertowi, który ostatecznie decyduje o ich dołączeniu do BW systemu.

Stosując opisaną procedurę do reguł zawartych w BW, można sprawdzać zgodność ich WP z aktualną zawartością BD. Umożliwia to okresowe weryfikowanie i modyfikowanie BW, co – w świetle medycyny – ma ogromne znaczenie ze względu na stały przyrost materiału obserwacyjnego.

Automatyczne generowanie reguł. O ile moduł MAS umożliwia generowanie specyficznych (pojedynczych) reguł, o tyle moduł automatycznego generowania zestawu korzysta z całości zebranego materiału. Główną zaletą zastosowanej metody (wariant algorytmu ID3 [6]) jest jej prostota i uniwersalność. Pozyskiwana wiedza jest reprezentowana za pomocą drzew decyzyjnych, które łatwo można przekształcić na reguły wnioskowania. Wprawdzie formalizm ten nie ma tak bogatych możliwości jak sieci semantyczne lub reprezentacje logiczne pierwszego rzędu (z punktu widzenia logiki operuje się tu jedynie rachunkiem zdań), ale zyskuje się za to na prostocie strategii uczenia się.

Informacją wejściową jest zbiór obiektów reprezentowanych za pomocą uporządkowanych n-tek $\langle x_1^i, x_2^i, ..., x_n^i \rangle$, gdzie x_j^i oznacza wartość j-tego atrybutu charakteryzującą i-ty obiekt. Należy zauważyć, że n-ty (ostatni) atrybut – oznaczany zazwyczaj terminem „klasa" – ma w pewnym sensie znaczenie odrębne od pozostałych. Atrybuty o numerach 1 do n-1 stanowią potencjalne przesłanki generowanych reguł, „klasa" natomiast odpowiada konkluzji reguł (pojęcie „klasa" może oznaczać np. nazwę jednostki chorobowej).

Proces generowania reguł przebiega według poniższego schematu (B oznacza zbiór obiektów):
- Jeżeli B zawiera obiekty należące do jednej klasy, to otrzymuje się liść drzewa opisany nazwą tej klasy;
- Jeżeli B zawiera obiekty z różnych klas, to wybiera się atrybut T i dokonuje podziału zbioru B na rozłączne klasy $B_1, ..., B_m$. Każdy podzbiór B_i zawiera obiekty, dla których atrybut T przyjmuje wartość t_i. Dla każdego podzbioru B_i należy powtórzyć opisaną wyżej procedurę. Przy wyborze atrybutu T stosuje się kryterium

$$\min_{T \in A} \{E(B) - E(B|T)\},$$

gdzie A jest zbiorem atrybutów, natomiast:

$$E(B) = \sum_k (n_k/n)\log_2(n_k/n)$$

$$E(B|T) = \sum_i (n_i/n)\, E(B_i)$$

oznaczają odpowiednio entropię próbki B oraz entropię warunkową. Tutaj n oznacza liczbę elementów należących do B, n_k – liczbę elementów, dla których wartość atrybutu T jest równa t_i.

Powyższa procedura zapewnia minimalność generowanego drzewa decyzyjnego, a jednocześnie porządkuje atrybuty ze względu na ich moc dyskryminacyjną. Przy dużej liczbie atrybutów celowe jest przeprowadzenie wstępnej analizy statystycznej ich istotności. Zadanie to można wykonać przy użyciu procedur z modułu statystycznego (np. testu χ^2 na niezależność statystyczną). Dopuszcza się także operowanie pojęciem uogólnionej klasy, które ma znaczenie w sytuacjach, gdy opisy obiektów są obarczone błędami. Istotnym rozszerzeniem algorytmu jest wzbogacenie go o procedurę wyboru optymalnej liczby wartości atrybutów. Pierwotny algorytm wykazuje bowiem tendencję do faworyzowania atrybutów o większej liczbie wartości. W celu uniknięcia przekłamań tego rodzaju wykorzystuje się procedurę opisaną w [5].

Moduł statystyczny. Klasyczną już metodą uogólniania wyników doświadczeń jest analiza statystyczna. W przypadku medycyny przeważająca część danych ma charakter klasyfikujący lub stopniujący, dla których jedynymi sensownymi relacjami są relacje równości, nierówności, większości i mniejszości. Dlatego analiza statystyczna w SYS8688 ogranicza się do obliczania podstawowych parametrów statystycznych

pojedynczej zmiennej, wyznaczenia pary zmiennych oraz szukania najbardziej znamiennie skorelowanych par w zbiorze wszystkich zmiennych. Bardziej zaawansowana analiza może być przeprowadzona przy użyciu innych pakietów statystycznych (np. STATGRAPH) o standardzie zbiorów zgodnych z dBase III.

Cechą wyróżniającą moduł statystyczny SYS8688 jest jego przystosowanie dla użytkownika nie będącego ekspertem w dziedzinie statystyki. Profesjonalne pakiety (SPSS i inne) dostarczają narzędzi analizy, pozostawiając profesjonalnemu statystykowi sterowanie procesem analizy i wyciągnięcie wniosków na podstawie uzyskanych wyników. SYS8688 sam nadzoruje proces wnioskowania i podaje wyniki w postaci gotowych ocen (np. dobiera testy statystyczne do typu danych, kryterium znamienności do wielkości próbki itp.).

Moduł MS umożliwia weryfikację znamienności reguł wnioskowania – hipotez wygenerowanych przez moduł automatycznego generowania, propozycji stworzonych przy użyciu modułu MAS, czy też reguł sformułowanych przez eksperta – na podstawie materiału z BD. Rezultaty analizy są prezentowane w postaci skomentowanych wyników liczbowych oraz wykresów graficznych. Pojedynczą zmienną obrazują – w zależności od jej typu – diagram kołowy lub histogram, a parę zmiennych – skatergram (zobrazowanie względnej gęstości rozkładu dwóch zmiennych przez zagęszczenie punktów w dwuwymiarowym układzie współrzędnych).

KOMUNIKACJA Z UŻYTKOWNIKIEM

Użytkownikiem systemu SYS8688 może być [3]:
- ekspert (wprowadzanie i modyfikowanie wiedzy),
- lekarz (przeglądanie i modyfikowanie informacji o pacjentach, badania statystyczne, konsultacja z systemem ekspertowym),
- średni personel medyczny (wprowadzanie badań, raporty).

Współpraca z SYS8688 odbywa się w sposób konwersacyjny przez hierarchiczny system „menu", zawierających spis możliwych do wykonania na danym poziomie funkcji. Po wybraniu jednej z funkcji system ochrony sprawdza, czy dany użytkownik ma prawo do jej realizacji, tzn. każdy użytkownik ma wyznaczony zakres kompetencji.

Przy projektowaniu komunikacji z użytkownikiem kierowano się przede wszystkim wygodą w posługiwaniu się systemem. Uproszczono czynności manualne i ograniczono do minimum ilość informacji, którą użytkownik musi zapamiętać do sprawnego korzystania z systemu. Uwzględniając podstawowe wymogi ergonomii, zapewniono daleko idącą jednolitość obsługi w różnych modułach, wspólną dla wszystkich modułów formę wprowadzania i przedstawiania danych oraz uzyskanych rezultatów. W wielu miejscach systemu przewidziano możliwość wyświetlenia na ekranie tekstów pomocniczych, informujących o tym, co można zrobić w danym module i w jakim celu.

* * *

SYS8688 jest uniwersalnym systemem doradczym, który może być wykorzystywany jako aktywny konsultant w rutynowych zadaniach diagnostycznych i (lub) jako asystent przy pracach badawczych. Docelowym obszarem jego zastosowań jest medycyna. W klinice typowego szpitala system funkcjonuje jako asystent lekarza, tzn. na podstawie otrzymanych informacji o pacjencie sugeruje diagnozę i odpowiednią dla niej terapię. Ponadto spełnia istotną funkcję nośnika najnowszej wiedzy medycznej przekazywanej w trakcie bezpośrednich sesji konsultacyjnych.

Wyposażony w mechanizmy gromadzenia danych i pozyskiwania wiedzy, system współpracuje z lekarzem przy prowadzeniu prac naukowo-badawczych. Rezultaty tych prac mogą być – na życzenie lekarza – wprowadzane do BW, co pozwala polepszyć jakość systemu (w sensie trafności stawianych diagnoz) oraz dopasować jego wiedzę do specyfiki konkretnej kliniki. Generowana w ten sposób nowa wiedza medyczna może być łatwo wymieniana między użytkownikami podobnych systemów. Omówione cechy SYS8688 predestynują go także do zastosowań dydaktycznych w placówkach naukowo-badawczych oraz uczelniach medycznych. Wyposażony w możliwości uzasadniania sugerowanych diagnoz, system może być wykorzystywany jako narzędzie wspomagające nauczanie studentów. Zastosowany w systemie formalizm reprezentacji wiedzy nie stawia specjalnych ograniczeń co do obszaru zastosowań. Głównym wymogiem jest, aby rozwiązywane problemy były wyrażalne za pomocą reguł wnioskowania.

SYS8688 został opracowany na komputery typu IBM PC/XT/AT wyposażone w dysk stały. Program napisano w jęz. C. Komunikacja z użytkownikiem odbywa się w języku polskim i obejmuje m.in. wykorzystanie polskich znaków monitora i (lub) drukarki (o ile użytkownik dysponuje ich sprzętową realizacją) oraz polskojęzyczne generowanie mowy.

Doświadczenia przeprowadzone na prototypowych BW wykazały, iż możliwości funkcjonalne systemu pozwalają na rozwiązywanie różnorodnych problemów medycznych, przy czym jakość rozwiązywania jest zbliżona do poziomu eksperta. Szczególnie istotna okazała się możliwość powiązania wnioskowania probabilistycznego, z przetwarzaniem relacji czasowych, co stanowi pewną nowość w dziedzinie medycznych SE.

LITERATURA

[1] Buchanan B.G., Shortliffe E.H.: Rule-Based Expert Systems. Addison-Wesley, Reading (MA), 1984
[2] Davis R., Lenat D.B.: Knowledge-Based Systems in Artificial Intelligence. McGraw-Hill, New York, 1982
[3] Kłopotek M., Pacan A., Syropiatko E.: User's Interface for the Expert and Statistical System SYS8688. Intern. Conf. on Artificial Intelligence. Suwałki, June 1987
[4] Kowalski A., Schutt A.: Analysis of Suppositions. T. Havranek, Z. Sidak, M. Novak (Eds.) COMPSAT'84. Proc. in Computational Statistics. Physica-Verlag, Wien, 1984.
[5] Michalewicz M.: Creative Data Analysis – Rule Generation from a Clinical Data Base. M. Chytil, R. Engelbrecht (Eds.), Medical Expert Systems. Sigma Press, Chichester (UK), 1988
[6] Quinlan J.R.: Learning efficient classification procedures and their application to chess and games. R.S. Michalski, J.G. Carbonell, T.M. Mitchel (Eds.), Machine Learning – An AI Approach Springer Verlag, Berlin, 1983
[7] Shortliffe E.H.: Computer-Based Medical Consultations – MYCIN. American Elsevier, New York, 1976.

ASTER – automatyczny sterownik...

dokończenie ze str. 7

Program obsługi transmisji wykonuje następujące zadania (po aktywizacji procesora sterownika):
- przygotowuje układy transmisji szeregowej do pracy,
- nadaje do linii i odbiera z linii komunikaty z lub do buforów zawartych we wspólnej pamięci RAM, według zleceń programu obsługi sieci, zgodnie z wybranym dla danej linii protokołem transmisji,
- informuje program POS o zaistniałych zdarzeniach i błędach.

Programy obsługi sieci i obsługi transmisji komunikują się przez bufory komunikatów zgodnie z zasadami podanymi w tabeli. Dodatkową możliwość komunikacji zapewnia system przerwań. Bufor komunikatów linii (odbiornika i nadajnika) ma strukturę pozwalającą elastycznie definiować zlecenia dla programu obsługi transmisji. Program obsługi sieci korzysta ze zbioru podprogramów zlecenia transmisji. Możliwe jest wydawanie zleceń bezpośrednio przez pamięć RAM. Program obsługi transmisji składa się z:
- bloku obsługi przerwań,
- emulatorów protokołów (IBM 2740, IBM 3270, BSC, BPPM, MERA 7953 Ne i in.),
- wektora konfiguracji sieci definiującego, który z emulatorów obsługuje daną linię.

* * *

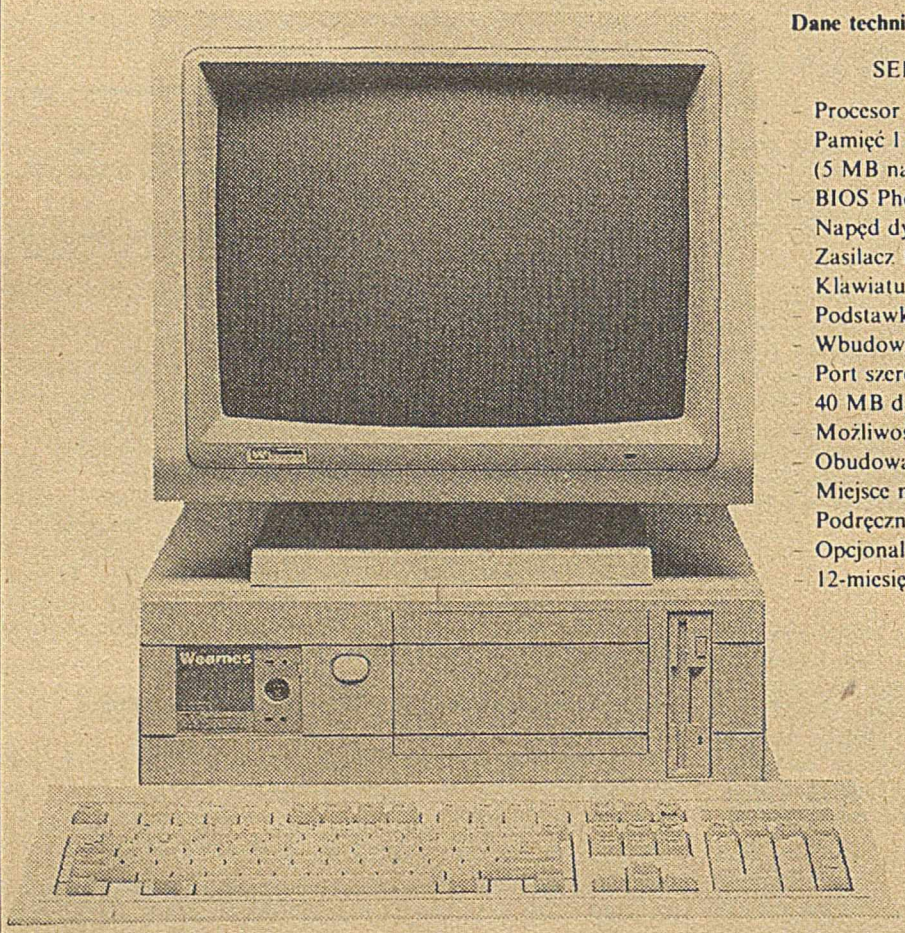
W Polsce dziedziny potencjalnego zastosowania sterownika są bardzo liczne. Wiąże się to z popularnością komputerów PC XT/AT, które są obecnie na biurku naukowca, wspomagają zarządzanie przedsiębiorstwem, jak też sterują procesem produkcyjnym. Zniesienie ograniczenia liczby łącz szeregowych komputera PC, prostota obsługi blokowych transmisji szeregowych z lub do wielu urządzeń, sterowników, monitorów i komputerów, według różnorodnych protokołów komunikacyjnych sieci lokalnej MasterNET, pozwala na dalsze rozszerzenie sfer stosowania sterownika. Wykorzystując sterownik można tworzyć wielostanowiskowe systemy informatyczne, np. w dydaktyce (kilkadziesiąt monitorów nauczających lub egzaminujących), zarządzaniu przedsiębiorstwem (wiele monitorów, komputerów i urządzeń obejmujących sieć informatyczną przedsiębiorstwa), przemyśle (kilkadziesiąt obrabiarek sterowanych numerycznie bezpośrednio przez PC), przetwarzaniu danych (terminale, monitory i komputery obsługujące magazyny, banki) itp.

KOMPUTER
Z WBUDOWANĄ PRZYSZŁOŚCIĄ

Komputer zaprojektowany przez czołowego światowego producenta mikrokomputerów – firmę ADVANCED LOGIC RESEARCH i wyprodukowany przez firmę WEARNES TECHNOLOGY.

Komputer, którego możliwości i cena oszałamiają konkurencję.

Komputer, który nie zestarzeje się, dzięki możliwości rozbudowy do architektury 386SX i 486.



Dane techniczne:

SERIA WEARNES BOLDLINE „M"

- Procesor 80286 12,5 Mhz
- Pamięć 1 MB RAM, możliwość rozbudowy do 16 MB (5 MB na płycie głównej)
- BIOS Phoenix
- Napęd dyskietek 3,5" 1,44 MB
- Zasilacz 110 Watt
- Klawiatura 101 klawiszy
- Podstawka dla koprocesora matematycznego 80287
- Wbudowany sterownik dyskowy z przeplotem 1:1
- Port szeregowy i równoległy
- 40 MB dysk sztywny
- Możliwość korzystania z EMS 4.0
- Obudowa typu „compact"
- Miejsce na dwa napędy 5,25" o wysokości 1/2
- Podręcznik i dyskietka z programem konfiguracyjnym
- Opcjonalna rozbudowa do 386SX i 486
- 12-miesięczna gwarancja

Komputer z wbudowaną przyszłością!

BOLDLINE COMPUTERS – GRUPA MICOMP-TECH
Biuro Informacji Techniczno-Handlowej
ul. Astrów 7, 40–045 Katowice
tel. 51-86-28 (telefaks), teleks 315687 COMP PL

EO/1123/89

MARIUSZ ORMIŃSKI
Komputer Studio Kajkowski
Sopot

Ethernet – od specyfikacji do realizacji praktycznej (2)

Realizacja praktyczna

Przedstawiona w poprzedniej części artykułu specyfikacja sieci Ethernet stanowi podstawę do praktycznej realizacji sieci. Sterowniki wykonuje się m.in. w postaci kart przeznaczonych do komputerów klasy IBM PC. Z uwagi na powszechne stosowanie programu Netbios i przyjęcie go jako standardowej realizacji podwarstwy warstwy łącza danych, odpowiedzialnej za sterowania połączeniami logicznymi, z kolejnych warstw oprogramowania sieciowego aż do warstwy sesji włącznie, czasami spotyka się karty obejmujące układy realizujące funkcje sterownika sieci Ethernet i modułu Netbios.

Wszystkie sterowniki do sieci Ethernet są wykonywane z układów scalonych VLSI, na przykład firmy Intel. Warstwę fizyczną wykonuje się przy użyciu układu 82501 i układu 82502 przeznaczonego do części nadawczo-odbiorczej (ang. *transceiver*). Zdefiniowaną w specyfikacji sieci typu Ethernet warstwę łącza danych, czyli de facto podwarstwę warstwy łącza danych sterującą dostępem do kanału komunikacyjnego, realizuje się przy użyciu specjalizowanego koprocesora 82586.

W Polsce sterownik do sieci typu Ethernet opracowano w Instytucie Informatyki Politechniki Śląskiej. Oprogramowanie umożliwiające jego zastosowanie niemal do wszystkich dostępnych w kraju pakietów oprogramowania warstw wyższych (np. NetWare, MikroLAN, MikroLAN/X), a także własny emulator Netbiosa, wykonano w firmie Computer Studio Kajkowski.

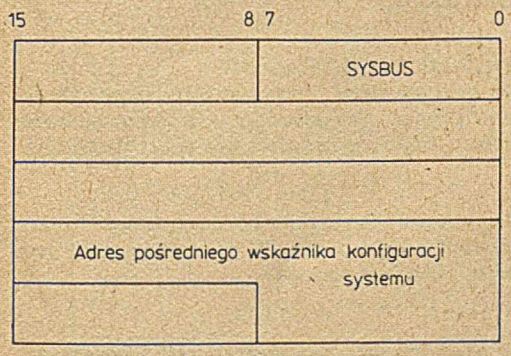
PRACA STEROWNIKA

Zrealizowany praktycznie sterownik wymaga rozwiązania nie tylko funkcji sterowania, ale również stworzenia komunikacji z wyższymi warstwami sieci. Funkcje te są realizowane przez zestaw procedur zwany handlerem realizujących funkcje systemu zarządzania siecią i umożliwiających komunikację sterownika z podwarstwą sterowania połączeniami logicznymi i warstwami wyższymi (w szczególności z modułem (NETBIOS). Sposób realizacji handlera zależy zarówno od możliwości sprzętu, jak i od rozwiązań przyjętych w oprogramowaniu warstw wyższych.

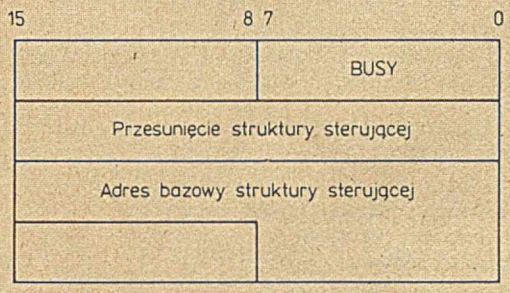
Handler realizuje następujące funkcje:
- inicjowanie swojej pracy i pracy sterownika,
- ustawienie fizycznego adresu stacji,
- ustawianie i kasowanie adresów grupowych,
- wysyłanie i odbieranie ramek,
- nadzorowanie poprawnej pracy sterownika.

Komunikacja ze sterownikiem odbywa się przez wspólną pamięć procesora oraz koprocesora 82586. Wymiana informacji i danych jest synchronizowana sygnałem przerwania do procesora i sygnałem zawiadamiającym dla koprocesora (ang. *channel attention*). Wspólny obszar pamięci podzielono na cztery części:
- obszar inicjujący (ang. *initialization root*),
- strukturę sterującą (ang. *system control block*),
- obszar odbioru ramek (ang. *receive frame area*).

Wspólny obszar pamięci z wyjątkiem obszaru inicjującego oraz buforów nadawczych i odbiorczych, musi zawierać się w ramach jednego segmentu o długości 64 KB.



Rys. 1. Wskaźnik konfiguracji systemu



Rys. 2. Pośredni wskaźnik konfiguracji systemu

W obszarze inicjującym występują dwie struktury (rys. 1,2): wskaźnik konfiguracji systemu (ang. *system configuration pointer*) oraz pośredni wskaźnik konfiguracji systemu (ang. *intermediate system configuration pointer*). Wskaźnik konfiguracji systemu znajduje się na stałe pod adresem OFFFFF6H. W polu SYSBUS znajduje się informacja o tym, czy systemowa szyna danych jest 8- czy 16-bitowa.

Znajdujące się w strukturze pośredniego wskaźnika konfiguracji systemu adresy, po zsumowaniu, określają położenie struktury sterującej w pamięci. Pole BUSY służy do przekazywania informacji o wykonywaniu inicjowania przez koprocesor 82586. Po włączeniu zasilania, przygotowaniu przez handler wszystkich wspólnych obszarów pamięci i wysłaniu sygnału zawiadamiającego do koprocesora, struktury obszaru inicjującego umożliwiają koprocesorowi zainicjowanie i znalezienie struktury sterującej.

Rys. 3. Postać struktury sterującej

Struktura sterująca (rys. 3) stanowi dwukierunkową skrytkę (skrzynkę pocztową) umożliwiającą komunikację między procesorem a dwiema

działającymi równolegle jednostkami funkcjonalnymi koprocesora: jednostką poleceń (ang. *command unit*) i jednostką odbiorczą (ang. *receive unit*). Jednostka poleceń służy do realizacji poleceń przekazywanych do koprocesora, jednostka odbiorcza umożliwia odbiór przesyłanych w sieci ramek.

Pierwsze słowo w strukturze sterującej służy do przekazywania informacji o stanie koprocesora. Jest ono ustawiane i zerowane przez koprocesor. Znaczenia poszczególnych pól są następujące:
CX – zakończenie wykonania polecenia, po którym miało być wygenerowane przerwanie,
RR – zakończenie odbioru ramki,
CNR – jednostka poleceń w stanie zawieszenia,
RNR – jednostka odbiorcza w stanie zawieszenia,
CUS – informacja o stanie jednostki poleceń (nieaktywna, zawieszona, aktywna),
RUS – informacja o stanie jednostki odbiorczej (nieaktywna, zawieszona, gotowa do odbioru, brak zasobów).

Drugie słowo struktury sterującej jest słowem poleceń. Jest ono zawsze ustawiane przez koprocesor. Poszczególne pola mają nastąpujące znaczenie:

ACK-CX – potwierdzenie informacji o wykonaniu poleceń,
ACK-FR – potwierdzenie odebrania ramki,
ACK-CNR – potwierdzenie informacji o zawieszeniu jednostki poleceń,
ACK-RNR – potwierdzenie informacji o zawieszeniu jednostki odbiorczej,
CUC – polecenia dla jednostki poleceń:
– pozostań w stanie bezczynności,
– wykonaj polecenia z listy,
– przystąp do wykonywania poleceń po zawieszeniu,
– zawieś pracę po zakończeniu bieżącego polecenia,
– zaniechaj natychmiast wykonywania poleceń,
RUC – polecenia dla jednostki odbiorczej,
– pozostań w stanie bezczynności,
– rozpocznij odbieranie ramek,
– powróć do odbierania ramek po zawieszeniu,
– zawieś odbieranie ramek,
– zaniechaj natychmiast odbierania ramek,
RES – polecenie ponownego zainicjowania sterownika.

[illegible]ejne dwa słowa struktury sterującej to pola adresowe, wskazujące [illegible] poleceń oraz obszar odbioru ramek. Pozostałe pola zawierają [illegible]ormacje statystyczne o pracy sieci. Informacje te mogą być wykorzys[illegible]ane przez warstwy wyższe.

Lista poleceń składa się z bloków (ang. *command block*) stanowiących poszczególne polecenia dla koprocesora. Dodatkowo występuje również lista deskryptorów buforów nadawczych (ang. *transmit buffer descriptor*) oraz bufory nadawcze.

Obszar odbioru ramek obejmuje listę deskryptorów ramek (ang. *frame descriptor*), listę deskryptorów buforów odbiorczych (ang. *receive buffer descriptor*) oraz bufory odbiorcze.

POLECENIA KOPROCESORA

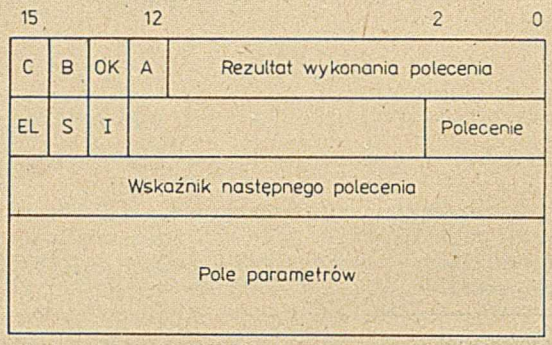
Koprocesor 82586 ma osiem poleceń opisanych poniżej.

Pola w bloku poleceń przedstawionym na rys. 4 mają następujące znaczenie:
C – polecenie zakończone,
B – polecenie w trakcie wykonywania,
OK – polecenie wykonane bezbłędnie,
A – polecenie przerwane,

Rezultat wykonania polecenia – (specyficzne dla danego polecenia informacje o jego wykonaniu):
EL – koniec listy poleceń,
S – nakaz zatrzymania po wykonaniu polecenia,
I – nakaz wysłania przerwania po wykonaniu polecenia,
polecenie – informacja o rodzaju polecenia (jego kod),
wskaźnik następnego polecenia – adres bloku następnego polecenia,
pole parametrów – parametry danego polecenia.

● NOP
Jest to polecenie nakazujące pozostanie w stanie bezczynności. W jego bloku nie występuje pole parametrów.

● IA – SETUP
Polecenie służy do ustawiania indywidualnego adresu węzła w sieci. Adres jest podany w polu parametrów.

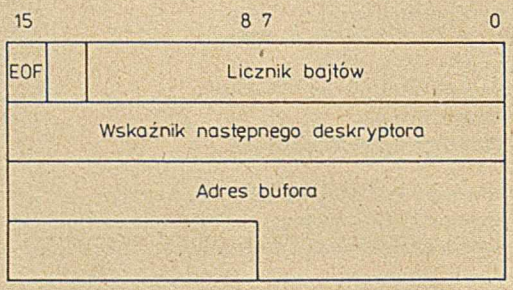


Rys. 4. Struktura bloku poleceń koprocesora 82586

● CONFIGURE
Polecenie umożliwia zmianę parametrów pracy sterownika. W sieci Ethernet polecenie to nie musi być używane, gdyż właściwe parametry są ustawiane automatycznie po włączeniu sterownika.

● MC – SETUP
Polecenie umożliwia ustawienie adresów wielokrotnych węzła w sieci. Pole parametrów składa się z licznika długości oraz listy adresów.

● TRANSMIT
Polecenie służy do nadawania ramek. Pole rezultatu wykonania polecenia zostało przeznaczone na szczegółowe informacje o przyczynach niepoprawnej transmisji, a także na informację o liczbie wykrytych kolizji. W polu parametrów znajduje się wskaźnik listy deskryptorów buforów nadawczych, adres docelowy oraz informacja protokolarna warstw wyższych (rys. 5). Poszczególne pola deskryptora bufora nadawczego mają następujące znaczenie:
EOF – pole zawierające informację o tym, że jest to deskryptor ostatniego bufora dla danej ramki.
Licznik bajtów – określa liczbę bajtów w buforze nadawczym związanym z tym deskryptorem.
Wskaźnik następnego deskryptora – zawiera adres kolejnego deskryptora w liście.
Adres bufora – jest to adres bufora nadawczego związanego z tym deskryptorem.



Rys. 5. Postać deskryptora bufora nadawczego

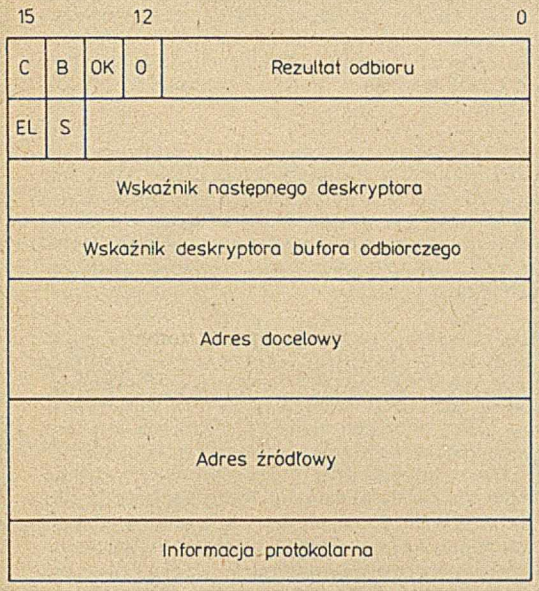
● TIME DOMAIN REFLECTOMETR – TDR
Polecenie umożliwia wykrycie zwarć i przerw w ośrodku transmisji. Jest to realizowane przez pomiar czasu od rozpoczęcia transmisji aż do otrzymania echa. Rezultat pomiaru jest podawany w polu parametrów.

● DUMP
Polecenie umożliwia zapisanie w wybranym obszarze pamięci zawartości wewnętrznych rejestrów koprocesora. W polu parametrów jest podany adres wybranego obszaru pamięci.

● DIAGNOSE
Polecenie powoduje uruchomienie wewnętrznego testu koprocesora. Wynik testu jest podawany w polu rezultatu wykonania polecenia.

NADAWANIE I ODBIÓR RAMEK

Występujące w obszarze odbioru ramek deskryptory ramek (rys. 6) tworzą listę. W deskryptorach tych występują pola wskaźników do deskryptorów buforów odbiorczych. Deskryptory buforów odbiorczych również tworzą listę. Pola C, B, OK, EL, i S deskryptora mają następujące znaczenie:
C – odbiór ramki zakończony,
B – deskryptor ramek zajęty przez jednostkę odbiorczą,
OK – ramka odebrana bezbłędnie,
EL – koniec listy deskryptorów,
S – nakaz zatrzymania po odebraniu ramki.

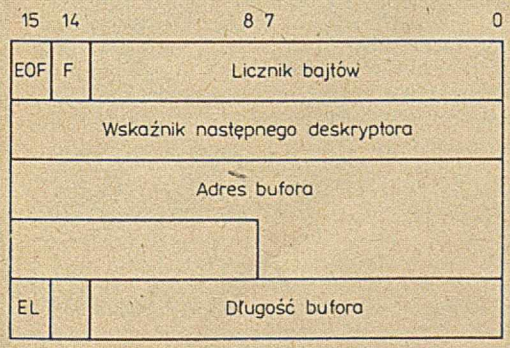


Rys. 6. Postać deskryptora ramek

Pole **rezultatu odbioru** jest przeznaczone na szczegółowe informacje o przyczynach błędnego odbioru ramki. Pola **adresu docelowego, adresu źródłowego** i **informacji protokolarnej** warstw wyższych są wypełniane na podstawie informacji pobranej z nadesłanej ramki.

Deskryptory buforów odbiorczych mają postać jak na rys. 7. Poszczególne pola deskryptora bufora odbiorczego mają następujące znaczenie:
EOF – informacja, że jest to deskryptor ostatniego bufora z odebraną ramką,
F – informacja, że wartość w polu licznika bajtów jest aktualna,
licznik bajtów – liczba bajtów w buforze odbiorczym związanym z tym deskryptorem,
wskaźnik następnego deskryptora – adres kolejnego deskryptora w liście,
adres bufora – adres bufora odbiorczego związanego z tym deskryptorem,
EL – informacja, że jest to ostatni deskryptor w liście,
długość bufora – długość bufora odbiorczego związanego z tym dyskryptorem.

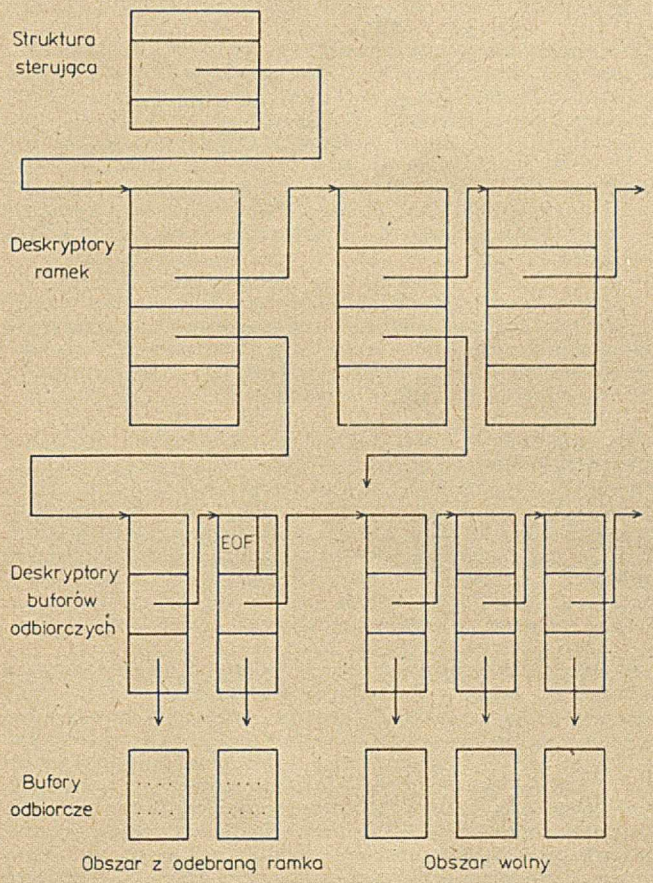
Komunikacja z koprocesorem odbywa się zawsze za pomocą struktury sterującej. Po właściwym jej wypełnieniu wysyła się do koprocesora sygnał zawiadamiający, po otrzymaniu którego koprocesor przystępuje do wykonywania odpowiednich działań na podstawie zawartych w strukturze sterującej informacji. Takim działaniem może być realizacja listy poleceń; może ona być dowolnie długa (ograniczona jedynie długością segmentu – 64 KB). Istnieje możliwość wysłania przerwania lub zawieszenia pracy jednostki poleceń po wykonaniu każdego polecenia. Wykonywanie poleceń może być również zaniechane w dowolnej chwili.



Rys. 7. Postać deskryptora bufora odbiorczego

Nadawanie ramek odbywa się przy użyciu polecenia TRANSMIT. W bloku tego polecenia znajduje się wskaźnik do listy deskryptorów buforów nadawczych. Z każdym deskryptorem jest związany odpowiedni bufor nadawczy. W buforach tych mogą znajdować się kolejne fragmenty pola danych ramki. Koprocesor tworzy ramkę na podstawie podanego w poleceniu TRANSMIT adresu docelowego i informacji protokolarnej oraz znanego sobie adresu źródłowego i znajdujących się w buforach nadawczych danych.

Odbiór ramek jest dokonywany przy użyciu jednostki odbiorczej. Jednostka odbiorcza po nadejściu ramki automatycznie wpisuje ją do obszaru odbioru ramek, w którym jest utworzona lista deskryptorów ramek. W ostatnim deskryptorze występuje znacznik końca listy, lecz w polu wskaźnika do następnego deskryptora jest wpisany adres pierwszego deskryptora w liście. W pierwszym deskryptorze jest podany adres listy deskryptorów buforów odbiorczych. W tej liście również w ostatnim deskryptorze jest ustawiony znacznik końca listy i w polu wskaźnika do następnego deskryptora jest wpisany adres pierwszego deskryptora w liście. Po wpisaniu pierwszej ramki do buforów odbiorczych i wypełnieniu deskryptorów tych buforów, koprocesor wpisuje odpowiednią informację do pierwszego deskryptora ramki oraz adres deskryptora pierwszego wolnego bufora odbiorczego do kolejnego deskryptora ramki. Obszar odbioru ramek po nadejściu ramki został przedstawiony na rys. 8.



Rys. 8. Obszar odbioru ramek

Po otrzymaniu przerwania od koprocesora i zidentyfikowaniu nadejścia ramki jako jego przyczyny, procesor odczytuje odebraną ramkę i przystępuje do zwolnienia wykorzystanych deskryptorów i buforów. Likwiduje w deskryptorze ramki adres listy deskryptorów buforów odbiorczych oraz przenosi znacznik końca listy do tego deskryptora.

dokończenie na s. 31

Profesor Antoni Kiliński 1909–1989

Profesor **ANTONI KILIŃSKI** urodził się 20 października 1909 r. w Antonowie na Litwie. Studia na Wydziale Elektrycznym Politechniki Warszawskiej ukończył w 1935 r. Tytuł docenta uzyskał w 1955 r., profesora nadzwyczajnego – w 1959 r., a profesora zwyczajnego – w 1965 r. W latach 1951–1954 był prodziekanem, a w latach 1956–1960 dziekanem Wydziału Łączności. Wielokrotnie był przedstawicielem Wydziału w Senacie Politechniki Warszawskiej (1960–1968). W latach 1969–1970 Profesor Kiliński był rektorem Politechniki Warszawskiej; od 1 października 1978 r. przeszedł na emeryturę.

Pracę w Politechnice Warszawskiej rozpoczął w 1935 roku jako asystent prof. M. Pożaryskiego. Równolegle, w 1936 r. pracował w Urzędzie Patentowym, a w latach 1937–1939 w Państwowym Instytucie Telekomunikacji na stanowisku kierownika Działu Aparatów Elektroakustycznych. W czasie okupacji ponownie pracował w Urzędzie Patentowym, a jednocześnie brał udział w pracy konspiracyjnej. W powstaniu warszawskim był oficerem łączności w „Kompanii Łużyce" AK. Po upadku powstania został wywieziony do obozu jenieckiego w Grossborn, a następnie do obozu Sandbostel.

Po powrocie do kraju został zatrudniony w Zjednoczeniu Przemysłu Radiotechnicznego, gdzie kierował zorganizowanym przez siebie Centralnym Biurem Konstrukcyjnym w Dzierżoniowie. Jednocześnie był pełnomocnikiem dyrektora Zjednoczenia ds. koordynacji planów produkcji podległych zakładów.

W roku 1948 Profesor Kiliński został powołany do służby wojskowej. Pracował w instytucjach centralnych Wojska Polskiego, a następnie również w Wojskowej Akademii Technicznej, gdzie kierował Katedrą Elektrotechniki Teoretycznej. W tym czasie prowadził także wykłady zlecone w Politechnice Wrocławskiej. W 1951 r. został kontraktowym samodzielnym pracownikiem nauki na Wydziale Łączności Politechniki Warszawskiej i w tym samym roku objął Katedrę Radiofonii, przemianowanej w 1953 r. na Katedrę Konstrukcji Telekomunikacyjnych i Radiofonii.

Po odejściu z wojska, od 1953 r. Profesor Kiliński całkowicie poświęcił się pracy w Politechnice Warszawskiej. Na Wydziale Łączności zorganizował studia z zakresu technologii sprzętu radiotechnicznego. Nadal kierował Katedrą Konstrukcji Telekomunikacyjnych i Radiofonii, a jednocześnie zorganizował Katedrę Technologii Sprzętu Elektronicznego i został jej pierwszym opiekunem. Przy Katedrze tej utworzył Zakład Doświadczalny Budowy Maszyn Matematycznych, którego zadaniem było opracowywanie i produkowanie sprzętu elektronicznego w nowatorskiej na owe czasy dziedzinie produkcji elektronicznej aparatury cyfrowej, stosowanej w atomistyce i elektronicznym przetwarzaniu danych. W Zakładzie opracowywano również konstrukcje i technologie elektronicznych maszyn cyfrowych. W 1963 r. został kierownikiem Katedry Budowy Maszyn Matematycznych, powstałej z dwóch katedr: Katedry Konstrukcji Telekomunikacyjnych i Radiofonii oraz Katedry Technologii Sprzętu Elektronicznego. W 1970 r. Katedra ta została przekształcona w Instytut Maszyn Matematycznych (w 1975 r. przemianowany na Instytut Informatyki), a profesor Kiliński został jego dyrektorem. Na stanowisku tym pozostał do 1978 r., tj. do przejścia na emeryturę.

W początkowym okresie pracy naukowo-badawczej zainteresowania Profesora Kilińskiego koncentrowały się na opracowaniu i konstruowaniu aparatury do rejestracji i reprodukcji dźwięków. Jego dorobek z tego okresu był wystawiany w 1939 r. na Wystawie Światowej w Nowym Jorku. Po wojnie, pracując w przemyśle, skoncentrował się na zagadnieniach niezawodności sprzętu elektronicznego i niezawodności przemysłowych procesów realizacji oraz na technologii sprzętu radiotechnicznego. Rezultatem prac w owym czasie była między innymi książka „Podstawy technologii sprzętu radiotechnicznego" (PWT, 1960 r.). Następnie Profesor zajął się problemem znalezienia teoretycznych podstaw oceny wzajemnego oddziaływania rozrzutów wartości parametrów układu elektronicznego; wyniki prac ogłaszał w wielu publikacjach. Ostatnią Jego publikacją na ten temat była monografia „Przemysłowe procesy realizacji, podstawy teorii".

Innym zagadnieniem, które trafiło na warsztat Profesora było zbadanie możliwości znalezienia teoretycznych podstaw oceny skuteczności stosowania redundancji w układach elektronicznych. Wyniki prac nad tym problemem zostały opublikowane m.in. w podręczniku pt. „Podstawy teorii przemysłowych procesów realizacji" (PW, 1972 r.) i w monografii „Przemysłowe procesy realizacji, podstawy teorii" (WNT, 1976 r.).

Koronnym osiągnięciem teoriopoznawczym Profesora Kilińskiego były wyniki badań opublikowane w rozprawie pt. „Definicje opisowo-analityczne i wartościująco-normatywne podstawowych pojęć teorii niezawodności" (Prakseologia, nr 38, 1971 r.). Profesor sformułował w niej układ twierdzeń ogólnych, z których wywodzą się wszystkie inne twierdzenia teorii niezawodności. Łączne potraktowanie zagadnień niezawodności struktur i procesów wytwarzania, dystrybucji, użytkowania i kształtowania na bieżąco jakości produktów i procesów doprowadziło do stworzenia przez Profesora nowej dyscypliny nazywanej przez prakseologów *ogólną teorią niezawodności*, a przez innych – *polską szkołą niezawodności*.

Poza teorią niezawodności nazwisko Profesora łączy się z początkami rozwoju informatyki w Polsce, pierwszymi krajowymi konstrukcjami maszyn cyfrowych oraz z kształceniem studentów, najpierw w ramach specjalności *Maszyny matematyczne*, a następnie w ramach powołanego z inicjatywy Profesora kierunku *Informatyka*.

Profesor Kiliński promował 23 doktorów, w tym kilkunastu z teorii niezawodności, reszta z zakresu informatyki. Wykaz Jego publikacji zawiera 100 pozycji, w tym kilkanaście monografii, podręczników i skryptów, kilkadziesiąt rozpraw i artykułów naukowych oraz kilkadziesiąt artykułów popularnonaukowych w czasopismach fachowych i prasie codziennej. Przez wiele lat był przewodniczącym Zespołu Dydaktyczno-Wychowawczego Elektroniki oraz Zespołu Informatyki, był przewodniczącym lub członkiem rad naukowych wielu instytutów (m.in. Instytutu Maszyn Matematycznych i Instytutu Tele- i Radiotechnicznego), poza tym członkiem Rady Głównej Szkolnictwa Wyższego i Centralnej Komisji Kwalifikacyjnej.

Za działalność naukową, dydaktyczną i współpracę z przemysłem Profesor otrzymał wiele odznaczeń i wyróżnień, między innymi: Order Sztandaru Pracy I Klasy oraz tytuł honorowy „Zasłużony Nauczyciel PRL", Krzyż Oficerski Orderu Odrodzenia Polski, Medal 40-lecia Polski Ludowej oraz siedem odznaczeń wojskowych.

Mimo odejścia na emeryturę Profesor żywo interesował się pracami Instytutu, cieszył się jego sukcesami, przeżywał jego niepowodzenia. Do ostatnich dni życia był niesłychanie aktywny zawodowo – opracowywał ekspertyzy, recenzje i opinie. Jeszcze w październiku 1988 r. wziął udział jako honorowy gość w konferencji „40 lat informatyki w Polsce" i wygłosił referat „O osiągnięciach Instytutu Informatyki Politechniki Warszawskiej, zastosowanych w praktyce". Referat ten w całości opublikowany w ubiegłorocznym numerze 8–12 *Informatyki* zawiera pełną syntezę osiągnięć Profesora oraz pracujących pod jego kierunkiem zespołów Politechniki Warszawskiej.

W naszej pamięci Profesor pozostanie na zawsze jako jeden z pionierów rozwoju i głównych realizatorów osiągnięć polskiej informatyki.

Cześć jego pamięci

LECH BANACHOWSKI
JAROSŁAW DEMINET
MAREK J. LAO
Instytut Informatyki
Uniwersytet Warszawski

Propozycja nowego programu uniwersyteckich studiów informatycznych (2)

W pierwszej części artykułu omówiono ogólne założenia proponowanego programu studiów uniwersyteckich, podział na blok podstawowy i zaawansowany oraz przedstawiono propozycje przedmiotów wchodzących w skład każdego z bloków. Druga część jest poświęcona szczegółowemu omówieniu przedmiotów bloku podstawowego. Przedstawione programy nie są kompletne – zostały opracowane jedynie zarysy ich treści, z różną dokładnością. Te propozycje mają stanowić początek właściwej dyskusji merytorycznej. W programie każdego przedmiotu wymieniono również przedmioty, których zaliczenie jest niezbędne do przystąpienia do zajęć danego przedmiotu.

INF1: Wstęp do informatyki (30 + 0 godz., egzamin)

Wymagane zaliczenie: brak
Przedmioty równoległe: INF2
Cel wykładu i wskazówki metodyczne
Zapoznanie studentów z problematyką studiów informatycznych, z podstawami organizacji komputerów i języków programowania; wprowadzenie do problemów rozwiązywania zagadnień metodami informatycznymi – analiza problemu, specyfikacja algorytmu, jego koszt i poprawność. Należy zwrócić uwagę na to, że równolegle jest prowadzony wykład ze wstępu do programowania, którego podstawowym celem jest praktyczne rozwiązywanie problemów metodami informatycznymi.
Program wykładu
- *Organizacja komputerów:* podstawowe elementy (procesor, pamięć, urządzenia wejścia–wyjścia), oprogramowanie systemowe, konfiguracja systemu, rozwój języków programowania (maszynowy, asemblerowy, wysokiego poziomu), wewnętrzna reprezentacja danych, translacja języków wysokiego poziomu.
- *Rozwiązywanie problemów i konstrukcja algorytmów:* analiza problemu, strategie rozwiązywania problemów, weryfikacja algorytmów (uwaga: podobne elementy wchodzą również w treść wykładu INF2).
- *Języki programowania:* metoda zapisu algorytmu, podstawy składni i semantyki, interpretery, kompilatory.
- *Algorytmy:* specyfikacja algorytmu, koszt algorytmu, iteracja, niezmienniki, rekurencja, równania rekurencyjne i punkty stałe, elementy dowodzenia poprawności algorytmów.

INF2: Wstęp do programowania (30 + 45 godz., egzamin)

Wymagane zaliczenie: brak
Przedmioty równoległe: INF1
Cel wykładu i wskazówki metodyczne
Nauka programowania i konstruowania algorytmów. Należy zwrócić przede wszystkim uwagę na praktyczne aspekty informatyki: systematyczne rozwiązywanie problemów, poprawność programów, dokumentowanie itp. Wskazane jest, by większą część pracowni stanowiły zajęcia praktyczne przy komputerze, pod nadzorem i opieką pracownika dydaktycznego. Pracownia powinna polegać na uruchamianiu krótkich programów po ich przedyskutowaniu w grupie. Podczas wykładu studenci powinni poznać praktycznie podstawowe konstrukcje pierwszego języka programowania – najbardziej przydatny wydaje się Pascal.
Program wykładu
- *Konstrukcja algorytmów:* rozwiązywanie problemów metodami informatycznymi, strategie rozwiązywania problemów (dekompozycja, rozwiązanie przez analogię, przez uogólnienie), identyfikowanie danych wejściowych i wymagań stawianych przed nimi, reprezentacja algorytmów.
- *Struktury programowe:* struktury sterowania (sekwencyjne, warunkowe, iteracyjne), podprogramy (funkcje, procedury, parametry, zasięg identyfikatorów, zagnieżdżenie podprogramów i rekurencja).
- *Typy danych:* pierwotne typy danych, operacje na obiektach tych typów, strukturalne typy danych: tablice, rekordy, napisy.
- *Konstrukcja programów:* projektowanie (abstrakcja danych, abstrakcja proceduralna, projektowanie zstępujące i metodą kolejnych ulepszeń, ukrywanie informacji), kodowanie i styl kodowania, instrukcje strukturalne, poprawność programów (wybór danych testowych, techniki wykrywania błędów, np. ślady, testowanie modułów), formalna weryfikacja programów (niezmienniki iteracji), dokumentacja (również dokumentacja wewnątrz programu).

INF3: Struktury danych (30 + 45 godz., egzamin)

Wymagane zaliczenie: INF1, INF2
Przedmioty równoległe: MAT1
Cel wykładu i wskazówki metodyczne
Rozwinięcie tematów wprowadzonych w INF2, podstawowe struktury danych używane w algorytmach komputerowych, techniki konstruowania algorytmów metodą kolejnych ulepszeń, przedstawienie podstawowych narzędzi analizy algorytmów i złożoności obliczeniowej. Podobnie jak w INF2, pracownia powinna być prowadzona w większej części przy komputerze i powinna pozwalać na praktyczne sprawdzenie poznawanych metod i narzędzi.
Program wykładu
- *Narzędzia analizy algorytmów:* asercje, niezmienniki, liczniki częstości.
- *Metoda kolejnych ulepszeń:* abstrakcyjne typy danych. *Listy:* listy uporządkowane, stosy i kolejki, listy z dowiązaniami, listy uogólnione.
- *Drzewa:* reprezentacja, przeszukiwanie, binarne drzewa poszukiwań, drzewa z dodatkowymi połączeniami.
- *Grafy:* reprezentacja, poszukiwanie, znajdowanie spójnych składowych, drzewa rozpinające, najkrótsze ścieżki, domknięcia tranzytywne.
- *Rekurencja:* realizacja za pomocą stosu.
- *Wyszukiwanie:* binarne i interpolacyjne.
- *Sortowanie wewnętrzne:* Insertionsort, Quicksort, Heapsort, kolejki priorytetowe.
- *Algorytmy geometryczne:* reprezentacja obiektów geometrycznych, otoczka wypukła, zasłanianie obiektów.
- *Pliki:* pliki w Pascalu (w ramach pracowni).

INF4: Techniki programowania I (30 + 45 godz., egzamin)

Wymagane zaliczenie: INF3, MAT1
Cel wykładu i wskazówki metodyczne
Przedstawienie technik konstruowania algorytmów komputerowych w zaawansowanym programowaniu imperatywnym oraz zapoznanie z metodami organizacji pracy programisty i zespołu. Jest to podstawowy wykład przygotowujący studenta do pracy w charakterze programisty i umożliwiający mu rozpoczęcie projektu INF12. W ramach wykładu i pracowni należy wprowadzić elementy drugiego języka programowania, umożliwiającego programowanie wielkoskalowe. Najbardziej odpowiednia wydaje się Modula, rozszerzająca Pascal omówiony w INF2 i INF3. Pracownia powinna być prowadzona bezpośrednio przy komputerze.
Program wykładu
- *Projekt programistyczny:* specyfikacje algorytmów i programów, projektowanie, kodowanie, dokumentacja, poprawność i testowanie, pielęgnowanie, organizacja pracy zespołowej.
- *Programowanie imperatywne:* metoda „dziel i zwyciężaj", algorytmy rekurencyjne, równania rekurencyjne, abstrakcyjne struktury danych na przykładzie optymalnego drzewa BST, programowanie z nawrotami, przybliżone rozwiązywanie problemów (algorytmy aproksymacyjne), metoda rozgałęzień i ograniczeń, metoda transformacji algorytmów na przykładzie algorytmów wyszukiwania wzorca w tekście.
- *Sztuczna inteligencja:* podstawy logiczne, heurystyki, przeszukiwanie z odcinaniem, reprezentowanie niepełnej wiedzy, systemy doradcze.

INF5: Techniki programowania II (30 + 45 godz., egzamin)

Wymagane zaliczenie: INF4, INF6 (lub równoległe)
Cel wykładu i wskazówki metodyczne
Przedstawienie technik programowania współbieżnego i deklaratywnego (kontynuacja tematyki INF4). Pozostałe uwagi jak w INF4.
Program wykładu
- *Programowanie współbieżne:* współprogramy, metody sekwencyjne programowania równoległego (Dekker), semafory, warunkowe rejony krytyczne, monitory, mechanizmy komunikacji (CSP), zdalne procedury (Ada).
- *Programowanie deklaratywne*

INF6: Wstęp do systemów komputerowych i operacyjnych (30 + 30 godz., egzamin)

Wymagane zaliczenie: INF3
Cel wykładu i wskazówki metodyczne
Wykład propedeutyczny łączący tematykę systemów komputerowych i systemów operacyjnych. Ćwiczenia powinny być głównie poświęcone zagadnieniom z systemów operacyjnych, a pracownia – praktycznemu programowaniu niektórych z omawianych algorytmów (np. obsługa przerwań, algorytmy wywłaszczania). Pracownia powinna prowadzić programowanie w języku asemblera.
Program wykładu
- *Struktura hierarchiczna komputera:* oprogramowanie, oprogramowanie systemowe i podstawowe, maszyna fizyczna.
- *Podstawy architektury komputera:* procesor, rozkazy, tryby adresowania, urządzenia, kanały, przerwania.
- *Wirtualizacja procesora:* rodzaj zadań zajmujących czas procesora, przerwania zegarowe, niepodzielność operacji, przekazywanie procesora, szeregowanie krótko- i długoterminowe, praca w czasie rzeczywistym.
- *Wirtualizacja pamięci:* możliwe rozwiązania sprzętowe, przestrzenie adresów logicznych, wirtualnych i fizycznych, podział pamięci, warstwy ochrony, tryby pracy, pamięć wielopoziomowa, pamięć wirtualna, segmentacja, stronicowanie.
- *Wirtualizacja urządzeń:* klasy urządzeń, zarządzanie urządzeniami, strumienie, poziomy dostępu, buforowanie, spoolery, organizacja plików i katalogów, ochrona dostępu.
- *Programy i zadania:* postać przemieszczalna, biblioteki, ładowanie, interpretowanie poleceń, hierarchie programów, potoki, dziedziczenie uprawnień, zarządzanie terminalem, okna.
- *Problemy synchronizacji:* bufory, rejony krytyczne, monitory, spotkania.
- *Sieci komputerowe*

INF7: Języki programowania i kompilatory (30 + 30 godz., egzamin)

Wymagane zaliczenie: INF4, INF6
Cel wykładu i wskazówki metodyczne
Przedstawienie praktycznych zagadnień związanych z realizacją języków programowania oraz struktur tych języków. Jest to propedeutyczna wersja wykładu z budowy translatorów.
Program wykładu
- *Automaty ze stosem:* definicje, determinizowanie, gramatyki LL i LR, parsery i skanery, abstrakcyjne drzewo wywodu.
- *Postać wynikowa programu:* właściwości języków, reguły widoczności, ich związek z generowanym kodem, zarządzanie zasobami.
- *Problemy kompilacji:* algorytm Sethi-Ullmana, optymalizacja kodu, moduły i oddzielna kompilacja.
- *Języki programowania:* definiowanie semantyki języka, struktury języków programowania.

INF8: Analiza algorytmów (30 + 30 godz., egzamin)

Wymagane zaliczenie: INF4, INF5 (lub równolegle), MAT3, MAT4
Cel wykładu i wskazówki metodyczne
Zapoznanie z podstawowymi metodami analizy złożoności obliczeniowej algorytmów. Zakłada się znajomość materiału z INF3 oraz materiału dotyczącego konstruowania algorytmów z INF4. Wykład należy do bloku podstawowego, ważne jest więc wyważenie ilości podawanych szczegółów i dowodów.
Program wykładu
- *Modele obliczeń:* maszyny Turinga, RAM, drzewa decyzyjne, związki między nimi.
- *Analiza probabilistyczna* na przykładzie algorytmów Quicksort i mieszania.
- *Analiza kosztu zamortyzowanego* na przykładzie struktury danych drzewiastej dla zbiorów rozłącznych i kopców Fibonacciego.
- *Dolne ograniczenia* w modelu drzew decyzyjnych na przykładzie problemów selekcji, metoda wyroczni.
- *Klasy P i NP: NP*-zupełność, algorytmy aproksymacyjne, ocena aproksymacji.
- *Algorytmy równoległe:* modele obliczeń równoległych (systoliki, sieci procesorów o stałej liczbie połączeń, PRAM), przykłady algorytmów (mnożenie macierzy, sortowanie, spójne składowe).

INF10: Metody optymalizacji (30 + 30 godz., egzamin)

Wymagane zaliczenie: MAT1, INF4, INF8 (lub równolegle), MAT6
Cel wykładu i wskazówki metodyczne
Zapoznanie z zadaniami programowania matematycznego i metodami ich rozwiązywania. Nacisk powinien być położony na algorytmiczny aspekt zagadnień. Należy zwrócić uwagę, że poruszane w tym wykładzie problemy są istotnie powiązane z innymi przedmiotami informatycznymi, jak np. Struktury danych, teoria grafów. Podczas zajęć studenci powinni opanować umiejętności: formułowania modeli matematycznych dla praktycznych potrzeb optymalizacji, doboru właściwej metody rozwiązania zagadnienia, komputerowej realizacji przedstawionych algorytmów.
Program wykładu
- *Programowanie liniowe:* modele programowana liniowego, elementy badań operacyjnych, podstawy teoretyczne, metody prymarne i dualne sympleks wraz z zasadami realizacji komputerowych, komputerowe systemy programowania liniowego, złożoności zadań programowania liniowego.
- *Optymalizycja kombinatoryczna:* zadania o macierzach unimodularnych, zagadnienia transportowe, przepływy w sieciach, problemy kombinatoryczne.
- *Programowanie całkowitoliczbowe:* metody odcięć, metody podziału i ograniczeń z użyciem technik programowania liniowego, ocena złożoności i efektywności.
- *Inne zadania optymalizacji dyskretnej:* problem plecakowy i zagadnienia minimalnego pokrycia, zagadnienia przydziału, problem komiwojażera, problemy szeregowania zadań.

INF11: Teoria obliczeń i języków formalnych (30 + 30 godz., egzamin)

Wymagane zaliczenie: MAT1, INF4 (lub równolegle)
Cel wykładu i wskazówki metodyczne
Zapoznanie z podstawowymi elementami lingwistyki matematycznej, języków formalnych rozstrzygalności i złożoności. Należy zwrócić uwagę na przekazanie ważnych tez dotyczących istnienia problemów algorytmicznie nierozwiązalnych oraz rozstrzygalnych i zasadniczo nierozwiązalnych z uwagi na złożoność.
Program wykładu
- *Wprowadzenie:* język formalny, maszyna Turinga, hipoteza Churcha, niedeterminizm, gramatyki i hierarchia Chomsky'ego.
- *Języki regularne:* automaty skończone Rabina-Scotta, automaty niedeterministyczne, minimalizacja automatu, własności języków regularnych, analiza i synteza automatu.
- *Języki bezkontekstowe:* gramatyka bezkontekstowa, wyprowadzenie, jednoznaczność gramatyki, lemat o pompowaniu, niedeterministyczne automaty ze stosem, właściwości języków bezkontekstowych.
- *Automaty komórkowe*
- *Nierozstrzygalność:* uniwersalna maszyna Turinga, problem stopu, twierdzenie Rice'a, nierozstrzygalność problemu Posta odpowiedniości słów, nierozstrzygalne problemy gramatyk bezkontekstowych.
- *Złożoność obliczeniowa:* czasowa i pamięciowa miara złożoności maszyn Turinga, liniowe przyspieszanie i twierdzenie o hierarchii, twierdzenie o lukach i twierdzenie o przyspieszaniu, języki NP-zupełne, przykłady problemów wykładniczo trudnych – względem czasu lub względem pamięci.

INF12: Projekt programistyczny (0 + 45 godz., zaliczenie)

Wymagane zaliczenie: INF5
Cel zajęć i wskazówki metodyczne
Potwierdzenie zdobytych wiadomości i umiejętności. Student powinien wykazać się umiejętnością samodzielnej pracy w charakterze programisty. Powinien wykorzystać wiadomości zdobyte w trakcie wykładów z praktycznej informatyki w zakresie analizy problemu, specyfikacji problemu, projektowania, kodowania w wybranym języku, programowania oraz dokumentowania swojej pracy. Przedmiotem zaliczenia powinien być poprawny program, dobrze udokumentowany i zweryfikowany. W zasadzie praca studenta powinna być indywidualna, wykonywana pod opieką pracownika (dydaktycznego lub technicznego z Ośrodka Obliczeniowego). Podane godziny dotyczą konsultacji w trakcie realizacji projektu, jego oceny oraz przedstawienia w formie seminarium odbiorczego. Zadanie powinno być niebanalne. Może być powiązane z dowolnym przedmiotem studiów (numeryczne, przetwarzanie danych itp.), w każdy przypadku należy zwrócić uwagę na przyjazną reakcję programu, czytelne wyprowadzenie wyników itp. Przedmiot powinien kończyć blok podstawowy. Projekt powinien być prowadzony przez dwa semestry.

INF13: Podstawy elektroniki cyfrowej (30 + 30 godz. laboratorium, zaliczenie)

Wymagane zaliczenie MAT1, INF6
Cel wykładu i wskazówki metodyczne
Zapoznanie studentów z podstawami elektronicznymi działania komputerów, równoważnością rozwiązań sprzętowych i programowych, „informatyzacji" sprzętu przez zwiększenie możliwości mikroprogramów. Przy szybkim postępie mikroelektroniki wydaje się, że podstawowa wiedza z tego zakresu będzie potrzebna wszystkim programistom, w szczególności programistom systemowym. Prowadzenie zajęć ma sens jedynie w przypadku dysponowania laboratorium elektronicznym wyposażonym w niezbędne układy elektroniczne, oscyloskopy, przewody itp.
Program wykładu
Bramki. Układy kombinacyjne. Układy sekwencyjne, licznik. Mikroprogramowanie. Mikroprocesory i ich mikroprogramy. Niestandardowe architektury komputerów: RISC, drzewiaste itp.

MAT1: Matematyka dyskretna (60 + 60 godz., egzamin)

Wymagane zaliczenie: brak
Przedmioty równoległe: MAT2, INF1, INF2
Cel zajęć i wskazówki metodyczne
Zapoznanie z podstawowymi pojęciami i strukturami matematyki dyskretnej, w tym logiki, teorii mnogości, algebry ogólnej i kombinatoryki, które mają związek z informatyką. Istotną częścią zajęć powinno być pokazanie zastosowań przedstawionych pojęć i konstrukcji w informatyce. Bardziej skomplikowane dowody twierdzeń mogą być opuszczone (np. twierdzenia o pełności rozważanych logik).
Program wykładu
- *Logika matematyczna:* rachunek zdań, rezolucja, rachunek pierwszego rzędu; pojęcie modelu, teorii, niesprzeczności, zupełności; modele Herbranda; logiki

nieklasyczne (intuicjonistyczne, niemonotoniczne, modalne), logiki programów.
● *Teoria mnogości:* rachunek zbiorów, funkcje, relacje, zbiory uporządkowane, zbiory dobrze uporządkowane, moc zbioru, liczby kardynalne; praporządki, porządki, kresy, kraty, łańcuchy, iteracyjne rozwiązywanie równań, punkty stałe.
● *Algebra ogólna:* algebra, kongruencja, algebry ilorazowe, twierdzenie o izomorfizmie, algebry wolne, algebry Boole'a, filtry, ideały, grupy, półgrupy.
● *Kombinatoryka:* obiekty kombinatoryczne, reprezentacje komputerowe, podstawowe klasy kombinatoryczne, zasada włączeń i wyłączeń, liczby Bella, Fibonacciego, Stirlinga, funkcje generujące, redundacja, odległości kodowe, kody samokorygujące się.

MAT2: Analiza matematyczna I (45+45 godz., egzamin)

Wymagane zaliczenie: brak
Przedmioty równoległe: MAT1
Cel zajęć i wskazówki metodyczne
Zapoznanie z podstawami teorii funkcji, różniczkowalności dla funkcji zmiennej rzeczywistej, zespolonej i wektorowej. Akcent powinien być położony na umiejętności wykorzystania zdobytej wiedzy w praktyce (w zadaniach), mniejszy na dowodzenie twierdzeń.
Program wykładu
● *Wstępne wiadomości o zbiorze liczb rzeczywistych*
● *Funkcje rzeczywiste:* ciągi liczbowe, funkcje rzeczywiste, zbieżność, ciągłość, szeregi o wyrazach rzeczywistych, rachunek różniczkowy dla funkcji rzeczywistych.
● *Ciągi i szeregi funkcyjne:* zbieżność jednostajna, różniczkowanie ciągów, promień zbieżności szeregów, aproksymacja i interpolacja.
● *Przestrzenie metryczne*
● *Funkcje zespolone i wektorowe:* uogólnienie poprzednich wiadomości.
● *Szeregi potęgowe:* szeregi potęgowe zmiennnej zespolonej, pochodna funkcji zmiennej zespolonej, funkcje elementarne zmiennej zespolonej, szeregi Fouriera.

MAT3: Analiza matematyczna II (45 + 45 godz., egzamin)

Wymagane zaliczenie: MAT1, INF2
Cel zajęć i wskazówki metodyczne
Zapoznanie z podstawami rachunku całkowego i równań różniczkowych. Akcent powinien być położony na umiejętność wykorzystania zdobytej wiedzy w praktyce (w zadaniach), z ograniczeniem dowodzenia twierdzeń. W ramach ćwiczeń powinno się omawiać numeryczne metody rozwiązywania zadań.
Program wykładu
Całka Riemanna. Miara i całka Lebesque'a. Równania różniczkowe zwyczajne pierwszego *rzędu. Funkcje analityczne – informacja.*

MAT4: Rachunek prawdopodobieństwa i statystyka (45 + 45 godz., egzamin)

Wymagane zaliczenie MAT1, MAT3
Cel zajęć i wskazówki metodyczne
Zapoznanie z podstawami probabilistyki i statystyki. Wiele twierdzeń niezbędnych do konstrukcji teorii musi pozostać bez dowodów. Wykład należy oprzeć na ogólnej teorii miary i całki Lebesque'a (znanej z MAT3).
Program wykładu
● *Wprowadzenie do rachunku prawdopodobieństwa:* doświadczenia losowe, modele probabilistyczne, przestrzenie probabilistyczne, modele ciągłe, doświadczenia wieloetapowe.
● *Zmienne losowe i ich rozkłady:* zmienne losowe jedno- i wielowymiarowe, rozkłady, niezależność, dystrybuanta, wartość oczekiwana i momenty, rozkłady warunkowe, funkcje charakterystyczne.
● *Ciągi niezależnych zmiennych losowych i twierdzenia graniczne:* zbieżność stochastyczna, słabe prawa wielkich liczb, zbieżność rozkładów, twierdzenie Poissona, Centralne Twierdzenie Graniczne, informacja o zbieżności z prawdopodobieństwem 1.
● *Wnioskowanie statystyczne:* estymacja parametrów, testowanie hipotez.
● *Procesy stochastyczne:* klasyfikacja, łańcuchy Markowa, zastosowania teorii kolejek.
● *Analiza regresji, korelacji, wariancji*

MAT5: Numeryczna algebra liniowa (60 + 60 godz., egzamin)

Wymagane zaliczenie: brak
Przedmioty równoległe: MAT2, INF1, INF2
Cel wykładu i wskazówki metodyczne
Zapoznanie z podstawowymi pojęciami algebry liniowej i analizy numerycznej. Algorytmy sprowadzania macierzy do różnych postaci powinny być rozważane pod kątem ich właściwości numerycznych. Tematy analizy numerycznej powinny być wplecione w tematy algebry liniowej.
Proces wykładu
● *Wprowadzenie do obliczeń numerycznych*: arytmetyka fl, błąd reprezentacji, działania w fl, błąd przeniesiony i wytworzony w arytmetyce, wskaźnik uwarunkowania zadania, numeryczna poprawność i stabilność algorytmów, normy wektorów i macierzy (**uwaga**: ten fragment powinien być realizowany stopniowo, według potrzeb).
● *Ciała i pierścienie:* definicje, przykłady, pierścień liczb całkowitych, wielomianów, ciało funkcji wymiernych, ciało F_p reszt modulo p.
● *Przestrzenie liniowe:* przestrzeń liniowa, podprzestrzeń, kombinacja liniowa, liniowa zależność i niezależność wektorów, baza, wymiar, współrzędne wektora w bazie, suma i suma prosta, podprzestrzeń generowana, przykłady: K^n, wielomiany, przestrzenie funkcyjne, przestrzeń rozwiązań układu równań liniowych jednorodnych.
● *Układy równań liniowych:* macierz, rząd macierzy, uwarunkowanie nieosobliwego układu liniowego, algorytm eliminacji, rozkład trójkątny, strategie wyboru elementu głównego, obliczanie wyznacznika.
● *Przekształcenia liniowe:* definicje, rząd przekształcenia, opis przekształceń za pomocą macierzy, działania na przekształceniach, izomorfizmy i macierze nieosobliwe, odwracanie macierzy, przestrzeń sprzężona, przekształcenia sprzężone, wartości i wektory własne, wielomian charakterystyczny.
● *Przestrzenie ortogonalne:* iloczyn skalarny w przestrzeni euklidesowej, jego macierz w danej bazie, prostopadłość wektorów, suma ortogonalna, konstrukcja bazy ortogonalnej metodą Schmidta, izomorfizm przestrzeni ortogonalnych, macierze samosprzężone a macierze symetryczne, diagonalizacja, właściwości geometryczne wektorów przestrzeni euklidesowej, wyznacznik Gramma.
● *Transformacja Householdera:* rozkład ortogonalno-trójkątny, zastosowanie do rozwiązywania układów równań itp.
● *Przestrzenie afiniczne*
● *Formy kwadratowe*: sprzężenie formy kwadratowej metodami Lagrange'a i macierzy ortogonalnych, twierdzenie o bezwładności, klasyfikacja zbiorów algebraicznych stopnia 2 w R^2 i R^3, własności geometryczne.

MAT6: Metody numeryczne (60+60 godz., egzamin)

Wymagane zaliczenie: MAT3, MAT5
Cel wykładu i wskazówki metodyczne
Zapoznanie studentów z podstawowymi zagadnieniami i algorytmami w metodach numerycznych. Po wykładzie studenci powinni dobrze orientować się w problemach poprawności i stabilności numerycznej oraz kosztu algorytmów rozwiązywania podstawowych zadań obliczeniowych. Powinni też umieć samodzielnie rozwiązywać proste zagadnienia numeryczne. W ramach ćwiczeń do wykładu studenci powinni opracować i uruchomić programy, porównać ich własności numeryczne itp.
Program wykładu
● *Przypomnienie z MAT5 podstawowych definicji*
● *Algorytmy rozwiązywania układów równań liniowych*: pozostałe, nie omówione w MAT5 (np. metoda Banachiewicza-Cholesky'ego), iteracyjne poprawianie rozwiązań, przykłady metod iteracyjnych, metoda sprzężonych gradientów.
● *Kwadratury i rozwiązywania równań różniczkowych zwyczajnych*: wzory prostokątów, trapezów (przypomnienie z MAT3), kwadratury interpolacyjne, twierdzenie o zbieżności procesu kwadratur, osiągalna dokładność obliczenia całki oznaczonej, proces kwadratur Romberga, całki z osobliwościami, informacja o obliczaniu całek wielokrotnych, informacja o różniczkowaniu numerycznym, metody jednokrokowe, przykłady metod wielokrokowych, zadania z wartościami brzegowymi.
● *Wielomiany i ich numeryczne reprezentacje:* zadanie interpolacyjne Lagrange'a, zadanie interpolacyjne Hermite'a, błąd przybliżenia funkcji jej wielomianem interpolacyjnym, problem Czebyszewa wyboru węzłów, wielomiany ortogonalne.
● *Aproksymacja wielomianami*: aproksymacja średniokwadratowa, zastosowanie wielomianów ortogonalnych, aproksymacja jednostajna, twierdzenie o alternansie, wielomiany dobrze aproksymujące, operatory rzutowe itp.
● *Wielomiany trygonometryczne i kubiczne funkcje sklejane:* wielomiany trygonometryczne, zadania interpolacyjne dla kubicznych funkcji sklejanych, właściwości ekstremalne rozwiązań, aproksymacja funkcji i jej pochodnych kubicznymi funkcjami sklejanymi.
● *Algebraiczne zadanie własne*
● *Rozwiązywanie równań nieliniowych:* metody bisekcji, stycznych, siecznych, osiągalna dokładność, wykładnik zbieżności, wskaźnik złożoności metody iteracyjnej, wielowymiarowa metoda iteracji prostych i metoda stycznych.

Rola Ady w wytwarzaniu...

dokończenie ze s. 13

Do nich mogą należeć systemy testowania, wykonywania automatycznych raportów wyników testowania, co ułatwia zidentyfikowanie regresji systemów.

* * *

Ada jest jak dotychczas jedynym językiem, który w swoich założeniach obejmuje cały okres istnienia programów. Nie ogranicza się tylko do problemów kodowania i testowania. Rozwój Ady wpłynął na rozwój innych popularnych języków programowania, takich jak Pascal lub Fortran. Jaka będzie rola Ady na współczesnym rynku produkcji oprogramowania w kontekście atrakcyjności innych zintegrowanych narzędzi do tworzenia produktów programowych – pokaże przyszłość; tymczasem nie sposób nie docenić roli, jaką spełniła w rozwoju inżynierii oprogramowania.

ZBIGNIEW HUZAR
LUDWIK KUŹNIARZ
Centrum Obliczeniowe
Politechnika Wrocławska

Język Lotos (3)

Specyfikacja systemów

W poprzednich częściach artykułu opisano mechanizmy algebraicznych specyfikacji typów danych oraz behawioralnych specyfikacji procesów. W tej części, na kilku przykładach pokazano łączne wykorzystanie tych mechanizmów do specyfikacji systemów reaktywnych. Ogólna postać specyfikacji w Lotosie ma postać:

specification *Nazwa-specyfikacji*
Lista-bramek-formalnych
Lista-parametrów-formalnych
: Funkcjonalność
Definicje-typów-globalnych
behaviour
Wyrażenie-behawioralne
where
Definicje-lokalne
endspec

Postać ta różni się od schematu definicji procesu możliwością wprowadzania definicji globalnych typów danych, przy czym mogą to być typy biblioteczne bądź typy definiowane tylko dla potrzeb danej specyfikacji.

W rozpatrywanych przykładach pojawi się jeszcze nie omówiona konstrukcja, zwana **wyrażeniem dozorowanym**, zapisywana w postaci

$[r] \rightarrow B$

gdzie r jest równością, nazywaną tutaj dozorem, natomiast B jest dowolnym wyrażeniem behawioralnym.

Interpretacja wyrażenia dozorowanego $[r] \rightarrow B$ jest następująca: jeżeli w danej konfiguracji proces (dla aktualnego wartościowania zmiennych) równość jest spełniona, to dalsze zachowanie procesu przebiega zgodnie z działaniem wyrażenia B; w przeciwnym przypadku dalsze działanie procesu jest puste, czyli równoważne procesowi **stop**.

Przedstawianymi przykładami będą: stos, sortowanie oraz prosty system komunikacyjny.

```
type BoolConst is
  sorts  bool
  opns   false, true : -> bool
endtype
type L_Nat is BoolConst
  sorts   nat
  opns  0 :          -> nat
        s : nat      -> nat
       _≤_ : nat nat -> bool
  eqns
    forall  n : nat
      ofsort bool
             n ≤ n       = true ;
             n ≤ s(n)    = true ;
             n ≤ s(s(n)) = true ;
          s(n) ≤ n       = false;
        s(s(n)) ≤ n      = false;
endtype
type Element is L_Nat
  sorts elem
  opns  a,b,+,*,↑  :       -> elem
                op : elem -> bool
                pr : elem -> nat
  eqns
    ofsort bool
       op(+) = true ;
       op(*) = true ;
       op(↑) = true ;
       op(a) = false;
       op(b) = false;
    ofsort nat
       pr(+) = 0;
       pr(*) = s(pr(+));
       pr(↑) = s(pr(*));
endtype
```

Wydruk 1

SPECYFIKACJA STOSU

Niech będzie dany następujący problem: opisać proces ONP, tłumaczący ciąg symboli reprezentujący wyrażenie arytmetyczne na odwrotną notację polską. Wyrażenie do przetłumaczenia jest podawane na wejściu w punkcie interakcji o nazwie *we*, a odwrotna notacja polska jest wysyłana na wyjście w punkcie interakcji *wy*. Dla uproszczenia zakłada się, że w skład wyrażenia wchodzą argumenty o identyfikatorach *a*, *b* oraz operacje „+", „*", „↑" takie że „↑" ma najwyższy priorytet, „*" następny w kolejności, a „+" najniższy. Wyrażenie nie zawiera nawiasów. Proces tłumaczący wyrażenie korzysta ze stosu, na którym są przechowywane operatory. Tłumaczenie odbywa się według następującego schematu:

– jeśli na wejściu pojawia się argument, to jest on wysyłany bezpośrednio na wyjście,
– jeśli na wejściu pojawia się operacja, to jest ona wysyłana na wyjście przez stos; polega to na tym, że dana operacja, zanim zostanie wysłana na wyjście, jest zapisywana na stos, z którego wcześniej zostają wysłane na wyjście operacje o priorytetach nie mniejszych niż dana operacja.

Zakłada się także, że proces ONP działa w sposób nieskończony.

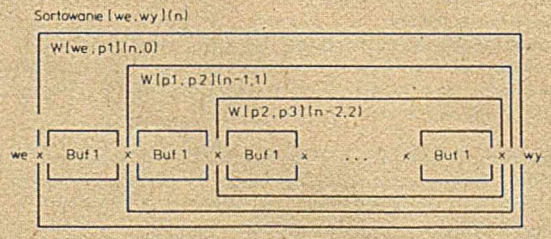
```
type Stos  is
  formalsorts el
  sorts  stos
  opns
         inic     :          -> stos
        szczyt    : stos     -> el
        ze_stosu  : stos     -> stos
        na_stos   : stos el  -> stos
  eqns
    forall x : el, s : stos
      ofsort el
        szczyt(na_stos(s,x)  = x;
      ofsort stos
        ze_stosu(inic) = inic;
        ze_stosu(na_stos(s,x)) = s;
endtype
type StosElem is Stos
  actualizedby Element using
  sortnames elem for el
  opns      pusty : stos -> bool
  eqns
     forall s : stos, x : elem
     ofsort bool
        pusty(inic) = true;
        pusty(na_stos(s,x)) = false;
endtype
```

Wydruk 2

```
process ONP [ we, wy ] ( x : stos ) : noexit :=
    we ? y : elem ;
       ( [ op(y) = false ] -> wy ! y ; exit (x)

       [] [ op(y) = true  ] -> P [ wy ] (x,y)
       )
          >> def z:stos in ONP[in,wy](z)
  where
  process  P [ wy ] ( x : stos, y : elem )
    exit ( stos ) :=
    (   [ pusty(x) = true ] ->   exit( na_stos(x,y)
     [] [ pusty(x) = false] ->
            ( [ pr(y) ≤ pr(szczyt(x))=true ] ->
                                wy!szczyt(x); P[wy](ze_stosu(x),y)
            [] [ pr(y) ≤ pr(szczyt(x))=false] ->
                                exit(na_stos(x,y)
            )
    )
  endprocess
endprocess
```

Wydruk 3



Rys. 1. Struktura systemu sortującego (W oznacza proces wtłaczania)

Podano dwa rozwiązania tego problemu. W jednym stos jest zrealizowany jako abstrakcyjny typ danych (wydruk 2), a w drugim jako proces

```
process ONP [ we, wy ] : noexit :=
hide pusty, szczyt, ze_stosu, na_stos in
  ( ONP_S[ we,wy, pusty, szczyt, ze_stosu, na_stos ]
  || StosElem[ pusty, szczyt, ze_stosu, na_stos ] )
where
 process StosElem [ pusty, szczyt, ze_stosu, na_stos ] : noexit :=
   (   pusty ! true ; exit
     [] na_stos ? x:elem ; S[pusty,szczyt,ze_stosu,na_stos](x))
   ) >> StosElem[ pusty, szczyt, ze_stosu, na_stos ]
   where
    process S[pusty,szczyt,ze_stosu,na_stos](x:elem):exit:=
      (   pusty! false ; S[pusty,szczyt,ze_stosu,na_stos](x)
        [] szczyt  ! x  ; S[pusty,szczyt,ze_stosu,na_stos](x)
        [] ze_stosu     ; exit
        [] na_stos ? y  ; S[pusty,szczyt,ze_stosu,na_stos](y)
                        >> S[pusty,szczyt,ze_stosu,na_stos](x)
      )
    endprocess
 endprocess
 process ONP_S [we,wy,pusty,szczyt,ze_stosu,na_stos]:noexit :=
   we ? x : elem ;
     ( [op(x)=false] -> wy ! x ;  exit
    [] [op(x)=true]  -> P[wy,pusty,szczyt,ze_stosu,na_stos](x)
     )    >>   ONP_S [ we,wy,pusty,szczyt,ze_stosu,na_stos ] )
   where
   process P[wy,pusty,szczyt,ze_stosu,na_stos] (x:elem):exit:=
       szczyt ? y:elem ;
       (  [ pr(x)≤pr(y) = true  ] -> wy ! y ; ze_stosu  ;
              P[wy,pusty,szczyt,ze_stosu,na_stos] (x:elem)
       [] [ pr(x)≤pr(y) = false ] -> na_stos !x ;  exit
       )
     [] pusty ! true;  na_stos ! x; exit
   endprocess
 endprocess
endprocess
```

Wydruk 4

(wydruk 4). Na wstępie zdefiniowano struktury danych używane w obu rozwiązaniach, tj. specyfikację typu Element (definiującego pojedynczy element tłumaczonego wyrażenia – wydruk 1). Wykorzystanie stosu w procesie tłumaczenia wyrażenia na odwrotną notację polską przedstawiono na wydruku 3.

```
specification Sortowanie[we,wy](n: Nat): noexit
 library Boolean endlib
    (* Typ Boolean jest typem standardowym -  jego  rodzaje  i
       operacje są określone następująco:
       type Boolean is
          sorts Bool
          opns true, false: -> Bool
               not: Bool -> Bool
               _and_, _or_, ... : Bool Bool -> Bool
          eqns ...................................
       endtype
    *)
 type LiczbyNat is Boolean
    sorts Nat
    opns  0: -> Nat
          Succ, Pred: Nat -> Nat
          _==_, _<_, _≤_: Nat Nat -> Bool
    eqns forall m,n : Nat
          ofsort Nat
             Pred(0)=0
             Pred(Succ(m))=m
          ofsort Bool
             0 == 0 =true
             0 == Succ(m) =false
             Succ(m) == 0 = false
             Succ(m) == Succ(n) = m == n
             0 < 0 = false
             0 < Succ(m) = true
             Succ(m) < 0 = false
             Succ(m) < Succ(n) = m < n
             m ≤ n = (m == n) or (m < n)      endtype
 behaviour Wtłaczanie[we,wy](n,0)
 where
   process Wtłaczanie[we,wy](i:Nat,j:Nat) : noexit :=
      [Succ(0)<i = true] -> hide p in
     Buf1[we,p](i,j)  |[p]| Wtłaczanie[p,wy](Pred(i),Succ(j))
      [] [i==Succ(0) = true] -> Buf1[we,wy](i,j)
   where
      process Buf1[we,wy](i:Nat,j:Nat) : noexit:=
         (* Proces odpowiada ustalonej  pozycji  w  sortowanym
            ciągu  i służy do wprowadzenia pierwszego elementu
            na tę pozycję. Parametr i  oznacza  numer  pozycji
            liczony  od  lewej  do  prawej  strony  -   rys.1;
            parametr j jest dopełnieniem i do liczby n będącej
            długością sortowanego ciągu.*)
         we?x:Nat; Buf2[we,wy](x,Pred(i),j)
      where
         process Buf2[we,wy](a:Nat,i:Nat,j:Nat): noexit :=
            (*  Proces  służy  do  porównywania  elementu
              wczytanego od  lewego sąsiada z elementem
              zapamiętanym i przesłania większego z  nich  do
              sąsiada z prawej strony - rys.1.*)
            [0<i = true] -> we?x:Nat;
             ([x≤a = true] -> wy!a; Buf2[we,wy](x,Pred(i),j)
              [][a≤x = true] -> wy!x; Buf2[we,wy](a,Pred(i),j))
            [] [i==0 = true] -> Buf3[we,wy](a,j)
         where
            process Buf3[we,wy](a:Nat,j:Nat) : noexit :=
               (* Proces  służy  do  wyprowadzania  elementów
                  uporządkowanego ciągu do sąsiedniego procesu
                  - rys.1.)
               [ 0<j = true] -> wy!a; we?x:Nat;
                                Buf3[we,wy](x,Pred(j))
             [][j==0 = true] -> wy!a; stop
            endproc
         endproc
      endproc
   endproc
endspec
```

Wydruk 5

SPECYFIKACJA SYSTEMU SORTUJĄCEGO

Sortowanie jest czynnością bardzo często spotykaną w systemach informatycznych. Istnieje też wiele algorytmów sortowania. W artykule przedstawiono system sortujący, który wykorzystuje ideę sortowania przez wtłaczanie (rys. 1). Koncepcja algorytmu jest następująca. Niech

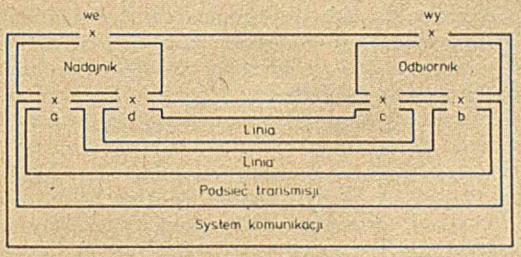
$$c_0 = \langle a_1 a_2,\ldots,a_n \rangle$$

będzie nieposortowanym ciągiem elementów – dalej przyjęto założenie, że są nimi liczby naturalne. Ciąg c_0 będzie kolejno przekształcany na ciągi c_1, c_2,..., c_n. Ciąg c_k, dla $1 \leqslant k \leqslant n$, stanowi posortowany ciąg elementów a_1, a_2,...,a_k z ciągu c_0. Ciąg c_n będzie więc posortowanym ciągiem wszystkich elementów c_0. Postępowanie jest oparte na następujących zasadach:

– ciąg $c_1 = \langle a_1 \rangle$

– jeżeli utworzono ciąg $c_k = \langle a_{i1}, a_{i2},\ldots, a_{ik} \rangle$, przy czym $k<n$, to ciąg c_{k+1} tworzy się w taki sposób, że bierze się element a_{k+1} z ciągu c_0 i następnie, przeglądając ciąg c_k od pozycji pierwszej do ostatniej, znajduje się w nim pozycję m ($1 \leqslant m \leqslant \mu$) taką, że $a_{im-1} \leqslant a_{k+1} \leqslant a_{im}$

Użyty symbol $\leqslant$ oznacza ogólnie pewną relację porządku częściowego na zbiorze elementów ciągu c_0; w tym przypadku jest to zwykła relacja nierówności w zbiorze liczb naturalnych.



Rys. 2. Struktura systemu komunikacyjnego

```
specification System-komunikacji[we,wy] : noexit
   library Boolean endlib
   type LiczbyNat is
     (* zdefiniowany jak w poprzednim przykładzie*)
   endtype
   type transmitowany-komunikat is LiczbyNat
      formalsorts zakłócenie
      formalopns ACK : -> Nat
                 NO-ACK : -> Nat
                 ERR : -> Nat
                 zakł-pocz : -> zakłócenie
                 zniekształcenie  :  Nat zakłócenie
                 korekta : Nat -> Nat
                 następne-zakł : zakłócenie -> zakłócenie
      formaleqns forall z : zakłócenie
                 ofsort Nat
                 zniekształcenie(ACK,z) = ACK;
                 zniekształcenie(NO-ACK,z) = NO-ACK;
   endtype
behaviour
   hide a,b,c,d in
   Nadajnik[we,a,d]
   |[a,d]|Podsieć-transmisji[a,b,c,d]|Odbiornik[we,b,c]
where
   process Nadajnik[we,do-lin,z-lin] : noexit :=
      we?x:Nat; Nadajnik-1[we,do-lin,z-lin](x)
   where
      process    Nadajnik-1[we,do-lin,z-lin](x:Nat)     :
                 noexit :=
         do-lin!x; z-lin?y:Nat;
       ([y==NO-ACK = true] -> Nadajnik-1[we,do-lin,z-lin](x)
      [][y== ACK    = true] -> Nadajnik  [we,do-lin,z-lin])
      endproc
   endproc
   process Odbiornik[wy,z-lin,do,lin] : noexit :=
      z-lin?x:Nat; Odbiornik-1[wy,z-lin,do-lin](x)
   where
      process Odbiornik-1[wy,z-lin,do-lin](x:Nat)
                 :noexit :=
         [korekta(x)==ERR = true ] -> do-lin!NO-ACK;
                                      Odbiornik[wy,z-lin,do-lin]
        [][korekta(x)==ERR = false] -> do-lin!ACK;
                                      wy!korekta(x);
                                      Odbiornik[wy,z-lin,do-lin]
      endproc
   endproc
   process
     System-transmisji[do-lin-1,z-lin-1,do-lin-2,z-lin-2] :
          noexit :=
       Linia[do-lin-1,z-lin-1](zakł-pocz)
     |[]|Linia[do-lin-2,z-lin-2](zakł-pocz)
   where
      process Linia[do,z](zakł:zakłócenie) : noexit :=
         do?x:Nat; z!zniekształcenie(x,zakł);
         Linia[do,z](następne-zakł(zakł))
      endproc
   endproc
endspec
```

Wydruk 6

Nowy ciąg c_{k+1} powstaje przez wtłoczenie elementu a_{k+1} na pozycję m ciągu c_k, z przesunięciem w prawo pozostałych jego elementów:

$$c_{k+1} = \langle a_{i1} \ldots a_{im-1}, a_{k+1}, a_{im}, \ldots, a_{ik} \rangle$$

System sortujący opisywany w Lotosie działa w taki sposób, że przez jedną ze swych bramek wczytuje nieposortowany ciąg o ustalonej długości, a następnie przez drugą bramkę wyprowadza ciąg posortowany. Specyfikację systemu przedstawiono na wydruku 5. Warto zwrócić uwagę na to, że system w przedstawionej postaci umożliwia równoległe wykonywanie wielu operacji, co nie jest jawnie wyrażone w słownym opisie zastosowanego algorytmu.

SPECYFIKACJA SYSTEMU KOMUNIKACYJNEGO

System służy przekazywaniu pewnej postaci komunikatów między nadawcą a odbiorcą. Składa się z nadajnika i odbiornika połączonych podsiecią transmisji, złożoną z pary linii transmisyjnych (rys. 2). Podsieć transmisji jest zawodna – komunikaty wysłane przez linię transmisyjną nie są wprawdzie gubione, lecz mogą być zniekształcane. Nadajnik pobiera komunikaty od nadawcy i wysyła je przez jedną z linii transmisyjnych do odbiornika. Odbiornik dokonuje korekcji przesłanego komunikatu, po czym sprawdza, czy korekcja powiodła się. Jeżeli tak, to odbiornik przesyła skorygowany komunikat do odbiorcy i potwierdza jego odbiór przesyłając komunikat potwierdzenia *ACK* drugą linią transmisyjną do nadajnika (wydruk 6); po odbiorze takiego potwierdzenia nadajnik pobiera od nadawcy kolejny komunikat do wysłania. Jeżeli stwierdzi się, że odebrany komunikat uległ zniekształceniu, to nie wysyła się go do odbiorcy, lecz powiadamia o tym fakcie nadajnik wysyłając drugą linią transmisyjną komunikat błędu *NO-ACK*; nadajnik reaguje na to retransmisją ostatnio wysyłanego komunikatu. Zakłada się przy tym, że komunikaty *ACK* oraz *NO-ACK* nie ulegają zniekształceniu.

Atrybuty jakości oprogramowania – definicje IEEE

dokończenie z III s. okładki

Użyteczność (ang. *usability*) – łatwość z jaką użytkownik może nauczyć się obsługi, przygotowywać dane wejściowe i interpretować wyniki działania systemu lub jego komponentów.

Wielokrotnie używalność (ang. *reusability*) – stopień w jakim moduł programowy może być wykorzystany w więcej niż jednym programie komputerowym.

Współdziałanie (ang. *interoperability*) – zdolność dwóch lub większej liczby składowych lub komponentów do wymiany informacji i jej użycia.

Wydajność (ang. *performance*) – stopień w jakim system lub jego składowa wykonuje swoje funkcje przy narzuconych ograniczeniach, takich jak szybkość, dokładność, wykorzystanie pamięci.

Zabezpieczenie (ang. *security*) – ochrona sprzętu komputerowego lub oprogramowania przed przypadkowym lub zamierzonym dostępem, użyciem, zmodyfikowaniem, zniszczeniem lub ujawnieniem.

Zespolenie (ang. *coupling*) – sposób i stopień współdziałania modułów programowych.

Zgodność (ang. *compatibility*) – zdolność dwóch lub wielu systemów lub ich komponentów do wymiany informacji.

Złożoność (ang. *complexity*) – stopień komplikacji lub trudności projektu lub implementacji systemu lub jego komponentu.

JANUSZ ZALEWSKI

SDS – system wytwarzania oprogramowania w Moduli 2

System SDS (ang. *Software Development System)* amerykańskiej firmy *Interfaces Technologies Corporation* jest zintegrowanym pakietem do programowania w Moduli 2, złożonym z edytora, kompilatora i konsolidatora. Poniżej opisano właściwości tego pakietu oraz porównano go z systemem Turbo Pascal, według tekstów przedstawionych w miesięczniku *Byte*.

Edytor

System SDS ma wbudowany edytor syntaktyczny. Każda konstrukcja składniowa Moduli 2 jest w nim wywoływana przez wciśnięcie kombinacji klawisza ALT z klawiszem literowym. Przykładowo, przyciśnięcie klawiszy ALT-F powoduje wstawienie w miejsce wskazane kursorem następującego szablonu instrukcji FOR:

```
FOR <id> := <expr> TO
      <expr> DO
END;
```

Klawisz TAB służy do przechodzenia między polami oznaczonymi przez nawiasy kątowe w celu wstawienia wartości odpowiednich zmiennych lub wyrażeń. Instrukcje między słowami zastrzeżonymi DO i END wstawia się po wciśnięciu klawiszy ALT-P. Średnik jest dostawiany automatycznie po każdej instrukcji. Jeżeli wstawiana konstrukcja jest szablonem instrukcji, to edytor dokonuje jej automatycznego wcięcia. Edytor dostawia również zawsze prawy nawias odpowiadający każdemu nowo wprowadzonemu lewemu nawiasowi.

W edytorze systemu SDS nie jest możliwe napisanie programu bez użycia opisanych szablonów. Na pierwszy rzut oka wydaje się to bardzo kłopotliwe, ale w praktyce jest bardzo użyteczne. Po pierwsze, w Moduli 2 znacząca jest wielkość liter, np. wszystkie słowa zastrzeżone, jak FOR, TO, END itd., muszą być pisane dużymi literami. Choć nawet dobre maszynistki często popełniają błędy literowe, użycie szablonów całkowicie eliminuje tę trudność. Po drugie, dzięki stosowaniu szablonów unika się błędów składniowych i można szybciej nauczyć się języka, szczególnie przy uprzedniej znajomości Pascala. Choć początkowo traci się trochę czasu na przyswojenie znaczenia różnych kombinacji klawisza ALT z klawiszami literowymi, to w praktyce są one bardzo łatwe do mnemonicznego zapamiętania. Przykładowo, kombinacja ALT-F odpowiada instrukcji FOR, a kombinacja ALT-A tablicy, tzn.

ARRAY [...]

Należy jednak pamiętać, że te same kombinacje mają różne znaczenia w zależności od części programu. W części daklaracyjnej, na przykład, kombinacja ALT-F powoduje wstawienie szablonu funkcji.

Do systemu SDS można oczywiście włączać pliki zapisane przy użyciu innych edytorów. Jednakże edytor SDS jest czuły na składnię, tzn. nie importuje on programów mających błędy składniowe, włączając błędy wynikające z niezgodności różnych deklaracji bibliotecznych samego systemu SDS. Po wykryciu błędu podczas importowania kodu jest on wskazywany, lecz jego poprawienia należy dokonać za pomocą innego edytora.

Edytor systemu SDS ma również pewne dogodne właściwości, jak na przykład możliwość wyszukiwania, kopiowania i zastępowania napisów. Umożliwia także wstawianie bloków kodu do wnętrza innych bloków i na odwrót, wydzielanie zaznaczonych części bloków. Bloki przemieszczane w ten sposób muszą być jednak pełnymi jednostkami syntaktycznymi, np. BEGIN...END lub REPEAT... UNTIL. Pomimo tych zalet, edytor jest jednak dość niewygodny, ponieważ dostęp do wielu jego poleceń jest możliwy dopiero po wciśnięciu klawisza ALT z odpowiednią literą i przejściu do menu niższego poziomu. Te z kolei wymagają wybrania odpowiedniego pola (za pomocą klawisza TAB), a niekiedy jeszcze dodatkowych czynności. To złożone postępowanie może nie przeszkadzać osobom początkującym, ale nie jest wygodne dla doświadczonych programistów, bardzo biegłych w pisaniu na klawiaturze i mających inne przyzwyczajenia.

Po tekście programu zapisanego w tym edytorze można poruszać się dwoma sposobami: wiersz po wierszu używając strzałek w lewo i w prawo (wiersz bieżący jest podświetlany) lub blok po bloku (w znaczeniu składniowym), używając strzałek w górę i w dół. Poruszanie się blok po bloku dotyczy bloków tego samego poziomu; do wnętrza bloków wchodzi się za pomocą strzałek w lewo lub w prawo. Ten sposób poruszania się po tekście jest wystarczająco szybki dla niezbyt długich programów, lecz niedostateczny dla dłuższych tekstów. W takim wypadku konieczna jest możliwość przeglądania tekstu pełnymi stronami.

Trudno jest ocenić, czy do pisania programów w Moduli 2 edytor składniowy jest lepszy w użyciu od typowego edytora tekstu. Odpowiedź zależy w dużym stopniu od czynników subiektywnych i doświadczenia w użytkowaniu pakietu SDS. O ile początkowo edytor tego systemu nie wydaje się zbyt wygodny, to po pewnym okresie użytkowania można się do niego przyzwyczaić.

Kompilacja i konsolidacja

Wybór rodzaju pracy w systemie SDS (redagowanie, kompilowanie lub konsolidowanie) odbywa się przez menu, a wykonanie odpowiednich segmentów – przez nakładkowanie. Współpraca z systemem zainstalowanym na dysku elastycznym jest bardzo wolna, natomiast użycie dysku elektronicznego wymaga około 600 KB pamięci RAM. Istotne jest, że pierwsza wersja systemu SDS miała poważne defekty, jeśli chodzi o pracę z dyskiem elektronicznym; w wersji drugiej defekty te usunięto.

Język akceptowany przez kompilator systemu SDS odpowiada niemal dokładnie standardowemu raportowi (N. Wirth, Modula 2, WNT, Warszawa, 1987). Oprócz wszystkich bibliotek standardowych, system zawiera też pewne dodatkowe moduły, jak np. RealInOut służący do wprowadzania i wyprowadzania liczb rzeczywistych. Moduł ten ma jednak pewne wady. Wyprowadza liczby rzeczywiste tylko w notacji matematycznej, a przy wyprowadzaniu liczby bez poprzedzenia kropki dziesiętnej cyfrą rejestruje tę liczbę jako zero, nie wydając żadnego ostrzeżenia.

W omawianej wersji kompilatora zrealizowano tablice otwarte Moduli 2, co pozwala na deklarowanie procedur z tablicami określonego typu, lecz bez określania wymiarów. Granice tablic są określane przez parametry aktualne w czasie wywołania procedury, co jest znaczną zaletą w porównaniu z Pascalem, szczególnie przy pisaniu procedur bibliotecznych, np. do operacji macierzowych.

System SDS zawiera także sprzężenie graficzne, które umożliwia kreślenie punktów, linii i prostokątów, sprzężenie akustyczne, służące do wytwarzania dźwięków o określonej wysokości, oraz sprzężenie z systemem operacyjnym, umożliwiające wykonywanie niektórych przerwań systemów DOS i BIOS. Za dodatkową opłatą można nabyć tzw. pakiet importowania obcych modułów wynikowych (ang. *foreign object module import package)*, który jest ważnym narzędziem służącym do łączenia modułów Moduli 2 z modułami wynikowymi, napisanymi w języku asemblera mikroprocesora 8086. Ten pakiet ma jednak pewne ograniczenia, z których najważniejszym jest niemożność istnienia oddzielnego segmentu danych w kodzie asemblerowym. Choć podobne ograniczenie istnieje w Turbo Pascalu, firma Borland nie pobiera dodatkowej opłaty za możliwość korzystania z kodu asemblerowego. Wymienione ograniczenie można jednak ominąć, np. umieszczając dane w segmencie kodu i umiejętnie je przeskakując. Jednak przy tworzeniu dużych programów jest to ograniczenie istotne.

System SDS ma wbudowane udogodnienia do tworzenia bibliotek, co umożliwia programistom gromadzenie oddzielnie kompilowanych modułów i łączenie ich z innymi programami. Kod źródłowy i wynikowy przechowywany w takiej bibliotece może być udostępniony tylko przez otwarcie biblioteki z edytora lub kompilatora. W aktualnej wersji systemu biblioteka musi znajdować się na tym samym dysku co kompilator.

Edytor i kompilator systemu SDS współdziałają od samego początku tworzenia programu, tzn. kompilator wykonuje pewne czynności wstępne (nie jest jasne jednak – jakie) zaraz po wprowadzeniu kodu do edytora. Istniejąca w menu opcja wyłączenia tej kompilacji w tle nie działa jednak przy próbie importowania pliku zawierającego niepoprawny fragment kodu.

Dokumentacja i defekty

Dokumentacja systemu SDS zawiera dość dobre wprowadzenie do języka, znacznie lepsze niż, na przykład, odpowiedni podręcznik Turbo Pascala. Dla osób znających wcześniej Pascal będzie to najprawdopodobniej wystarczające, lecz osoby nie programujące w Pascalu muszą zapoznać się przedtem z dobrym podręcznikiem Moduli 2 – jako wprowadzeniem.

Ponieważ wszystkie stałe, zmienne i procedury importowane z bibliotek muszą być zadeklarowane, konieczne jest posiadanie wiedzy, jakie identyfikatory są przechowywane w jakich bibliotekach. Niestety, nie jest to łatwe do określenia na podstawie istniejącej dokumentacji; nawet jej indeks nie jest dokładny i wykazuje niezgodności z rzeczywistą numeracją stron. Poważniejszą wadą jest jednak fakt, że różne biblioteki mogą zawierać ten sam identyfikator o identycznym brzmieniu w dwóch różnych kontekstach. Przykładowo, zmienna Done jest typu BOOLEAN w bibliotece InOut, natomiast w bibliotece Files ma typ wyliczeniowy. Z tego względu, bardzo użyteczne byłoby posiadanie tablicy identyfikatorów występujących we wszystkich bibliotekach, jak również – spisu zawartości tych bibliotek. Podobnie korzystne byłoby przesunięcie opisu użycia kluczy funkcyjnych w łatwiej dostępne miejsce bliżej początku podręcznika.

Ogólnie rzecz ujmując, w dokumentacji Moduli 2 systemu SDS brakuje wielu informacji, z wyjątkiem najprostszych przykładów. O ile we wprowadzeniu zawarto omówienie najbardziej elementarnych konstrukcji języka, zasady pisania i kompilowania programów, to nieco trudniejsze tematy, takie jak tworzenie niezależnie kompilowanych bibliotek, wplatanie kodu maszynowego lub użycie przerwań BIOS-a, wymagają dokładniejszego wyjaśnienia i zilustrowania przykładami.

Pierwsza wersja omawianego systemu miała niezliczoną niemal liczbę defektów. Większość z nich została jednak usunięta w następnych wydaniach. Nadal jednak zdarza się, na przykład, że kompilator wysyła bardzo mylące komunikaty. Przy wywołaniu procedury NEW w celu utworzenia wskaźnika pojawia się komunikat, że zmienna nie jest zadeklarowana, mimo iż faktycznie ją zadeklarowano. Okazuje się, że konieczne jest w tym wypadku zaimportowanie procedur ALLOCATE i DEALLOCATE z biblioteki systemowej. Nawet jeśli w programie nie występują wywołania tych procedur, to są one niezbędne do działania procedur NEW i DISPOSE. Choć podobnym kłopotom jest dość łatwo zaradzić, samo ich istnienie nie najlepiej świadczy o jakości używanego oprogramowania. Pewną pomoc w rozwiązywaniu tych trudności stanowi elektroniczna tablica ogłoszeń utrzymywana przez producenta systemu, na której umieszcza się informacje o usuniętych defektach. Ze względu na koszt połączeń telefonicznych, jest ona jednak użyteczna raczej tylko dla użytkowników z bezpośredniego sąsiedztwa firmy.

Porównanie z Turbo Pascalem

W celu porównania szybkości obu systemów przepisano w Moduli 2 teksty wzorcowe opublikowane w miesięczniku Byte z lutego 1986 r. Poszczególne programy wykonują następujące zadania:
CALC – 10000 operacji mnożenia i dzielenia liczb pojedynczej precyzji,
FLOAT – testowanie funkcji standardowych (operacje zmiennoprzecinkowe),
SIEVE – jedna iteracja metodą sita Erastotenesa do obliczania liczb pierwszych,
TRANS – przesyłanie znakowe pliku o długości 10000 znaków,
BTRANS – przesyłanie blokowe (128 bajtów) powyższego pliku,
LINETEST – wykreślanie linii standardową procedurą,
HEAPTEST – tworzenie i niszczenie zmiennych dynamicznych.

Jak wynika z danych przedstawionych w tabeli, Turbo Pascal w wersji 3.0 ma przewagę we wszystkich przypadkach, z wyjątkiem operacji zmiennoprzecinkowych (mnożenia i dzielenia) za pomocą koprocesora 8087, które są prawie trzykrotnie szybciej wykonywane w systemie SDS. Z drugiej jednak strony, przewaga Turbo Pascala w wykonywaniu operacji arytmetycznych bez koprocesora i w zarządzaniu stertą (ang. *heap*) jest podobnie wielka.

Analizując szybkość kompilacji należy jednak dodać, że zadanie kompilatora Moduli 2 jest nieco trudniejsze, ponieważ musi on wytworzyć kod, który będzie potem łączony z innym kodem. Jest to czynność bardziej złożona niż kompilowanie od razu na kod wykonywalny i wymaga więcej czasu. Ponadto, kompilator Moduli 2 wytwarza pliki typu EXE, a kompilator Turbo Pascala – pliki typu COM. Pliki wykonywalne omówionych programów wzorcowych dla Moduli 2 są 2–2,5 raza większe niż odpowiednie pliki Turbo Pascala. O ile wielkość skompilowanego kodu w Turbo Pascalu nie może przekroczyć 64 KB, to dla omawianego systemu Moduli 2 nie ma takich ograniczeń. Programy mogą być tak długie, jak może pomieścić je dysk lub pamięć operacyjna.

Oprócz bardzo technicznego porównania szybkości i wielkości odpowiednich programów, interesujące jest także porównanie budowy obu języków – mające aspekt bardziej psychologiczny. Jeśli ktoś uważa Pascal za język zbyt sformalizowany (stypizowany) i ograniczający elastyczność, lecz dogodny do tworzenia „piramid", tj. dużych i złożonych programów zbudowanych z wielu modułów, to znajdzie te same cechy w Moduli 2. W Pascalu istnieje chyba tylko jedna w pełni dynamiczna i elastyczna procedura standardowa, WriteLn, dopuszczająca zmienną liczbę parametrów różnych typów. W Moduli 2 zastąpiono ją pięcioma innymi procedurami: WriteChar, WriteString, WriteInt, WriteCard i WriteReal. Deklarując je, należy oczywiście przestrzegać poprawnego użycia dużych i małych liter oraz podać źródło pochodzenia (bibliotekę). Jest jasne, że narażenie na takie niedogodności może skłonić programistę do użycia języka C, który zapewnia również bardziej zwarty kod.

Trzeba jednak powiedzieć, że zupełnie inne cechy języka są decydujące, gdy dziesiątki lub setki programistów pracują nad projektem dotyczącym wyprodukowania setek tysięcy wierszy kodu, który musi być modularny i przenośny. Jeśli potrzeby użytkownika nie sięgają tak daleko i poprzestaje on na programach o długości 3–4 tysięcy wierszy, to nie ma sensu rezygnować z Turbo Pascala. Jest on szybszy, bardziej zwarty i elastyczny, mniejszy, prostszy i tańszy niż jakikolwiek inny język dostępny obecnie na rynku. Z drugiej strony, jeżeli w rachubę wchodzi współpraca z wieloma innymi programistami nad olbrzymim programem, lub jeśli do tej pracy konieczne jest szkolenie, to prawdopodobnie lepsze będzie użycie Moduli 2. Edytor składniowy systemu SDS znacznie ułatwia opanowanie tego języka, niezależnie od tego, czy Modula 2 przewyższy kiedyś popularnością Pascal czy nie. Oprócz omówionych składników, w pakiecie SDS znajdują się także dodatkowe biblioteki, sprzężenie z językiem asemblera i program usługowy Make, wzorowany na swoim odpowiedniku z systemu Unix. Na zakończenie wypada przyznać, że system programowania w Moduli 2 SDS wart jest stosowania, choć nie zastępuje Turbo Pascala.

oprac. **J. Zal.**

Porównanie szybkości obliczeń w Moduli 2 SDS i Turbo Pascalu (czas podano w sekundach)

Program wzorcowy	Modula 2	Turbo Pascal
CALC (obliczenia pojedynczej precyzji bez koprocesora)	52	32
CALC 87 (obliczenia pojedynczej precyzji z koprocesorem)	2	6,5
FLOAT (obliczenia zmiennoprzecinkowe bez koprocesora)	212	65
FLOAT 87 (obliczenia zmiennoprzecinkowe z koprocesorem)	5	3
SIEVE (sito Erastotenesa)	19	13
TRANSD (przesyłanie znakowe z dysku na dysk)	129	124
TRANSM (przesyłanie znakowe z pamięci do pamięci)	98	93
BTRANSD (przesyłanie blokowe z dysku na dysk)	23	22
BTRANSM (przesyłanie blokowe z pamięci do pamięci)	2	1,5
LINETEST (rysowanie linii)	31	17
HEAPTEST (tworzenie i niszczenie zmiennych dynamicznych)	12	< 0,5

Realizacja praktyczna

dokończenie ze s. 21

Podobnie w liście deskryptorów buforów odbiorczych przenosi wskaźnik końca tej listy do ostatniego z użytych deskryptorów oraz we wszystkich użytych deskryptorach kasuje znacznik F. W ten sposób zwolnione deskryptory dołącza się na koniec listy wolnych deskryptorów.

* * *

Przedstawione dwie najniższe warstwy sieci typu Ethernet stanowią podstawę do realizacji sieci obejmującej wszystkie siedem warstw. Wyższe warstwy są realizowane w postaci sprzętowo niezależnych pakietów. W Polsce jest dostępnych kilka rodzajów profesjonalnego oprogramowania sieciowego mogącego, bezpośrednio lub za pomocą Netbiosa, tworzyć pełną siedmiowarstwową sieć Etherent, dla komputerów klasy IBM PC (NetWare, PC Network, Xenix-NET, MikroLAN/X i inne). Wydaje się, że szczególnie interesujące mogą być te rozwiązania, które umożliwiają współpracę oprogramowania sieciowego z wielodostępnymi systemami operacyjnymi. Takie podejście daje możliwości tworzenia systemów informatycznych, które w sposób bardzo elastyczny mogą spełniać najróżniejsze wymagania, pojawiające się w konkretnych przedsiębiorstwach lub instytucjach.

Współbieżność sterowania bazą wideodanych i systemów przetwarzania obrazów

dokończenie ze s. 10

Proponowane rozwiązanie może znaleźć zastosowanie we wszystkich ośrodkach naukowo-badawczych zajmujących się przetwarzaniem niejednorodnych danych niekoniecznie obrazowych, gdzie niezbędne jest usystematyzowane składowanie na bieżąco otrzymywanej po przetworzeniu informacji. Walory sieciowe rozwiązania i rozproszenie systemu baz danych będą przydatne w wielokomputerowych systemach przetwarzania danych o dowolnej strukturze, zwłaszcza w coraz bardziej rozpowszechniających się sieciach mikrokomputerowych operujących bankami danych.

Główne walory proponowanego rozwiązania to niezawodność (dzięki stale obecnym kopiom do odzyskania i systemowi operacyjnemu Unix) oraz elastyczność (dzięki modułowej budowie i prostocie dodawania kolejnych użytkowników).

LITERATURA

[1] Chang S.K., Fu K.S.: Pictorial Information Systems. Springer-Verlag, Berlin, 1980
[2] Xenix System V. Microsoft Corporation. The Santa Cruz Operation Inc., 1987

Przypominamy, że zlecenia na umieszczenie ogłoszeń w INFORMATYCE przyjmuje w bieżącym roku bezpośrednio nasza redakcja. Czekamy na zgłoszenia.

Holt R.C.: Turing – język lepszy od Pascala

INFORMATYKA 1990, nr 3, s. 1

Charakterystyka opracowanego i stosowanego w nauczaniu na Uniwersytecie w Toronto uniwersalnego języka programowania Turing, stanowiącego istotne ulepszenie języka Pascal.

Holt R.C.: Turing – a language better as Pascal

INFORMATYKA 1990, No. 3, p. 1

Characteristics of at Toronto University elaborated and in didactic process applied universal programming language Turing, which presents essential improvement of the Pascal language.

Holt R.C.: Turing – die Sprache, die besser als Pascal ist

INFORMATYKA 1990, Nr. 3, S. 1

Eine Charakteristik der in Toronto-Universität erarbeiteten und in Lernprozess angewendeten Turing-Universalprogrammierungssprache, die eine wesentliche Rationalisierung der Pascal-Sprache darstellt.

Cabański J.G., Lewandowski M., Marach S., Oczki Z.: ASTER – automatyczny sterownik transmisji szeregowych dla PC XT/AT

INFORMATYKA 1990, nr 3, s. 6

Charakterystyka rozwiązań sterownika ASTER, usprawniającego działanie mikrokomputerów klasy PC XT/AT przy obsłudze wielu transmisji szeregowych.

Cabański J.G., Lewandowski M., Marach S., Oczki Z: ASTER – an automatic controller for serial transmission with PC XT/AT

INFORMATYKA 1990, No. 3, p. 6

Characteristics of the ASTER controller solutions, which rationalizes the operation of the PC XT/AT microcomputer by servicing many serial transmissions.

Cabański J.G., Lewandowski M., Marach S., Oczki Z.: ASTER – eine automatische Steuerungsanlage für Seriellübertragung mit Anwendung von PC XT/AT–R chnern

INFORMATYKA 1990, Nr. 3, S. 6

Eine Charakteristik von Lösungen der ASTER Steuerungsanlage, die den Betrieb der PC XT/AT Rechner bei Bedienung von mahreren Seriellübertragungen rationalisiert.

Sobol H.: Współbieżność systemu sterowania bazą wideodanych oraz systemów przetwarzania obrazów

INFORMATYKA 1990, nr 3, s. 9

Charakterystyka rozwiązania programowego systemu sterowania bazą wideodanych oraz systemów przetwarzania obrazów realizowanych na mikrokomputerach klasy PC, AT.

Sobol H.: Concurrency of the control system for videodata base and image processing systems

INFORMATYKA 1990, No. 3, p. 9

Characteristics of program solutions of control system for videodata base and image processing systems realized on microcomputer of the PC AT category.

Sobol H.: Parallelbetrieb des Systems für Steuerung der Videodatenbank und der Bildverarbeitungssysteme

INFORMATYKA 1990, Nr. 3, S. 9

Eine Charakteristik von Programmlösung des Systems für Steuerung der Videodatenbank und der Bildverarbeitungssysteme auf Mikrorechnern der PC AT Kategorie.

Łubińska K.: Rola Ady w wytwarzaniu oprogramowania

INFORMATYKA 1990, nr 3, s. 12

Omówienie zalet oraz dotychczasowego zastosowania języka Ada, a także jego wpływu na usprawnienie procesu produkcji oprogramowania.

Łubińska K.: The part of Ada in software manufacturing

INFORMATYKA 1989, No. 10, p. 12

Discussion of advantages and hitherto application of the Ada language, as well as of its influence on improvement of software manufacturing process.

Łubińska K.: Die Rolle der ADA–Sprache bei Softwareerzeugung

INFORMATYKA 1990, Nr. 3, S. 12

Eine Besprechung von Vorteilen und bisherigen Anwendung der Ada Sprache, sowie von ihrer Einfluss auf Rationalisierung des Softwareeuzezgungsprozesses.

Kłopotek M. i inni: SYS8688 – system pozyskiwania i weryfikacji wierdzy medycznej (2)

INFORMATYKA, 1990, nr 3, s. 15

Druga część charakterystyki rozwiązań programowych oraz sposobu działania systemu ekspertowego dla wybranych obszarów medycyny, zawierająca omówienie pozostałych elementów systemu, zakresu jego stosowania i dotychczasowych doświadczeń i realizacji.

Kłopotek et al.: SYS8688 – a system for medical science acquisition and verification (2)

INFORMATYKA 1990, No. 3, p. 12

Second part od characteristics of program solution and operation method for an expert system in selected medicine areas, which includes discussion of remaining elements of the system, scope of its application and actual experience of realization

Kłopotek u.a.: SYS8688 – ein System für Gewinnung und Verifikation der Medizinwissenschaft (2)

INFORMATYKA 1990, Nr. 3, S. 15

Zweiter Teil einer Charakteristik von Programmlösungen und Betriebsweise eines Expertensystems für ausgewählte Medizinbereite, der eine Besprechung der übergebliebenen Elemente des Systems, seines Anwendungsbereiches und der bisherigen Realisationserfahrungen umfasst.

Ormiński M.: Ethernet – od specyfikacji do realizacji praktycznej (2). Realizacja praktyczna

INFORMATYKA 1990, nr 3, s. 19

Druga część charakterystyki rozwiązań oraz sposobu działania sieci Ethernet, zawierająca omówienie podstaw realizacji tej sieci.

Ormiński M.: The Ethernet – from specification to practical realization (2). Practical realization.

INFORMATYKA 1990, No. 3, p. 19

Second part of characteristics of the Ethernet network solutions and operating method, which includes discussion of the network realization base.

Ormiński M.: Ethernet – von Spezifikation bis praktischer Realisation (2). Die praktische Realisation

INFORMATYKA 1990, Nr. 3, S. 19

Zweiter Teil einer Charakteristik von Lösungen und Betriebsweise des Ethernet-Netzes, der ine Besprechung der Realisationsgrundlagen Dieses Netzes umfasst.

Banachowski L., Deminet J., Lao M.J.: Propozycja nowego programu uniwersyteckich studiów informatycznych (2)

INFORMATYKA 1990, nr 3, s. 23

Druga część szczegółowego omówienia propozycji nowego programu uniwersyteckich studiów informatycznych w Polsce, wzorowane na rozwiązaniach zalecanych przez wiodące światowe organizacje informatyczne.

Banachowski L., Deminet J., Lao M.J.: A proposal for a new program of informatics university studies (2)

INFORMATYKA 1990, No. 3, p. 23

Second part of detailed discussion of the proposal for new informatics university studies program in Poland, which is modelled after solutions recommended by leading informatics worldwide organizations.

Banachowski L., Deminet J., Lao M.J.: Ein Vorschlag des neuen Programmes für Informatik-Universitätsstudien (2)

INFORMATYKA 1990, Nr. 3, S. 23

Zweiter Teil einer detaillierten Besprechung des Vorschlages von neuen Programm für Informatik-Universitätsstudien in Polen, das nach Empfehlungen der Führenden Informatikweltorganisationen erarbeitet wurde.

Huzar Z., Kuźniarz L.: Język Lotos (3). Specyfikacja systemów

INFORMATYKA 1990, nr 3, s. 26

Trzecia część charakterystyki języka Lotos jako narzędzia specyfikacji formalnej, zawierająca omówienie sposobu specyfikowania systemów.

Huzar Z., Kuźniarz L.: The Lotos language (3). Systems specification

INFORMATYKA 1990, No. 3, p. 26

Third part of characteristics of the Lotos language as a tool for formal specification, which includes discussion of the method for systems specification.

Huzar Z., Kuźniarz L.: Lotos-Sprache (3). Spezifikation der Systeme

INFORMATYKA 1990, Nr. 3, S. 26

Dritter Teil einer Charakteristik der Lotos-Sprache als ein Werkzeug für Formalspezifikation, der eine Besprechung der Methode für Systemespezifikation umfasst.

Atrybuty jakości oprogramowania – definicje IEEE

W niniejszym odcinku rubryki terminologicznej podaję definicje atrybutów jakości oprogramowania według słownika IEEE Glossary of Software Engineering Terminology (IEEE Std 729–1989).

Dyspozycyjność (ang. *availability*) – stopień w jakim system lub jego komponent jest dostępny i zdolny do pracy, gdy żąda się jego użycia.

Efektywność (ang. *efficiency*) – stopień wykorzystania zasobów przez system lub jego komponent w celu spełnienia wymaganych funkcji.

Elastyczność (ang. *flexibility*) łatwość z jaką można modyfikować system lub jego komponent w celu użycia go w zastosowaniu lub w środowisku innym niż to, do którego był zaprojektowany.

Integralność (ang. *integrity*) – stopień w jakim system lub jego komponent jest chroniony przed nieupoważnionym dostępem lub zmodyfikowaniem programów komputerowych lub danych.

Konserwowalność (ang. *maintainability*) – łatwość z jaką system programowy lub jego komponent może być modyfikowany w celu poprawienia defektów, dostosowania do zmian środowiskowych, zwiększenia wydajności lub polepszenia innych parametrów.

Krytyczność – (ang. *criticality, severity*) – stopień w jakim wymaganie, moduł, błąd, defekt, awaria lub inny czynnik wpływa na opracowanie lub działanie systemu.

Modularność (ang. *modularity*) – stopień zawartości w systemie lub programie komputerowym dyskretnych komponentów, takich że wymiana jednego komponentu ma minimalny wpływ na pozostałe.

Nadmiarowość (ang. *redundancy*) – obecność dodatkowych komponentów w systemie, przeznaczonych do wykonywania tych samych lub podobnych funkcji, w celu zapobiegania uszkodzeniom lub przywrócenia stanu sprzed uszkodzenia.

Niesprzeczność (ang. *consistency*) – stopień jednolitości i znormalizowania części systemu lub jego komponentu.

Niezawodność (ang. *reliability*) – zdolność systemu lub jego komponentów do wykonywania wymaganych funkcji przez określony czas, w ustalonych warunkach.

Odporność (ang. *robustness*) – stopień w jakim system lub jego komponent może działać poprawnie w obecności nieważnych sygnałów wejściowych lub niesprzyjających warunków środowiskowych.

Ogólność (ang. *generality*) – szerokość zakresu funkcji wykonywanych przez system lub jego komponent.

Poprawność (ang. *correctness*) – stopień w jakim system programowy lub jego komponent jest wolny od defektów projektowych i defektów kodu.

Prostota (ang. *simplicity*) – stopień zrozumiałości i jasności projektu lub implementacji systemu lub komponentu.

Przenośność (ang. *portability*) – łatwość z jaką system lub jego komponenty mogą być przenoszone z jednego sprzętu lub środowiska programowego do innego.

Realizowalność (ang. *feasibility*) – stopień w jakim przy istniejących ograniczeniach mogą być zrealizowane wymagania, projekt lub plany dotyczące systemu lub jego komponentu.

Rozszerzalność (ang. *extendability*) – łatwość modyfikacji systemu lub jego komponentów w celu zwiększenia wielkości pamięci lub zdolności funkcjonalnych.

Spoistość (ang. *cohesion*) – sposób i stopień wzajemnego powiązania zadań realizowanych przez pojedynczy moduł programowy.

Śladowalność (ang. *traceability*) – stopień w jakim można ustalić związki o charakterze poprzednik-następnik lub nadrzędny-podległy między dwoma lub wieloma produktami procesu wytwórczego.

Testowalność (ang. *testability*) – stopień w jakim system lub jego komponent ułatwia ustalenie kryteriów testowania i przeprowadzenie testów w celu stwierdzenia, czy spełnione są te kryteria.

Tolerowanie błędów (ang. *error tolerance*) – zdolność systemu lub jego komponentu do normalnego działania pomimo obecności błędnych sygnałów wejściowych.

Tolerowanie defektów (ang. *fault tolerance*) – zdolność systemu lub jego komponentu do normalnego działania pomimo istnienia defektów sprzętowych lub programowych.

dokończenie na s. 28