2 1988

informatyka

Prof. Ryszard S. Michalski
o naturze uczenia się

System dydaktyczny do nauki języka ADA/SM

Analizator-tester protokołów

WNT/SIGMA NOT

Cena 200 zł

informatyka

2 1988

Nr 2
Miesięcznik Rok XXIII
Luty 1988

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 01-517 Warszawa, ul. Mickiewicza 18 m. 17, tel. 39-14-34

Zakł. Graf. „Tamka". Zam. 1033-1300/87. Obj. 4,0 ark. druk. Nakład 8650 egz. U-23.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 200 zł
Prenumerata roczna 2400 zł

WYDAWNICTWO
NACZELNA ORGANIZACJA TECHNICZNA
CZASOPISM I KSIĄŻEK TECHNICZNYCH
SIGMA
00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Str.

W NAJBLIŻSZYCH NUMERACH:

- Maciej M. Sysło omawia wybrane metody rozwiązywania trzech przykładowych problemów kombinatorycznych: plecakowego, kolorowania grafu i najkrótszego drzewa rozpinającego w sieci.
- Artur Krępski inauguruje cykl trzech artykułów na temat systemu programowania Smalltalk-80, prezentuje zasadnicze cechy systemu i dąży do jego spopularyzowania w Polsce.
- Jan Bielecki opisuje wybrane elementy ANSI języka C.
- Andrzej Warda przedstawia szczegółowy opis pamięci dyskowych Mazovii 1016.
- Dorota Inkielman prezentuje komputerowy kurs języka Ada dla mikrokomputerów IBM PC, opracowany przez firmy Alsys i Cassie.
- Jarosław Deminet przedstawia Dom Handlowy Nauki, opisuje jego działania zmierzające do zakupu mikrokomputerów klasy IBM PC z Dalekiego Wschodu.

ANTONI KRECZMAR
Instytut Informatyki
Uniwersytet Warszawski

POLSKIE TOWARZYSTWO INFORMATYCZNE

Języki obiektowe (2)

W drugiej części artykułu omówiono dziedziczenie jedno- i wielopoziomowe, dalsze właściwości klas oraz różnice między dziedziczeniem a zagnieżdżaniem.

DZIEDZICZENIE

Tak jak klasy i obiekty, dziedziczenie występuje we wszystkich językach obiektowych. Jest to niezwykle ważne i ciekawe narzędzie. Pozwala tworzyć hierarchię klas (a więc hierarchię w świecie wzorców). Taka hierarchia ze świata wzorców przenosi się automatycznie na hierarchię w świecie obiektów.

Rozpocznę — jak poprzednio — od przykładu. Niech poniższa klasa reprezentuje wzorzec dla typu pojazd:

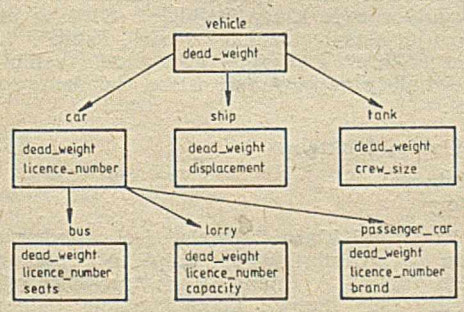
```
vehicle:class
  dead_weight:real;
end vehicle;
```

Jeżeli należałoby rozwinąć tę klasę w naturalny sposób na klasy "car", "ship", "tank" itp., nie trzeba powtarzać w każdym z tych wzorców za każdym razem atrybutów klasy "vehicle". Wystarczy skorzystać właśnie z dziedziczenia. Klasy "car", "ship" i "tank" mogą dziedziczyć wszystkie cechy klasy "vehicle". Przykładowo, definicje tych klas mogą wyglądać następująco:

```
car of vehicle:class (licence_number:integer);
end car;
ship of vehicle:class
  displacement:integer;
end ship;
tank of vehicle:class
  crew_size:integer;
end tank;
```

Można tak postępować dalej, rozbudowując mianowicie klasę "car" na klasy "bus", "lorry", "passenger_car":

```
bus of car:class
  seats:integer;
end bus;
lorry of car:class
  capacity:real;
end lorry;
passenger_car of car:class (brand:text);
end passenger_car;
```



Rys. 1. Hierarchia utworzonych klas

Ponieważ cała hierarchia trochę się skomplikowała, warto podsumować, jakie atrybuty mają poszczególne klasy (rys. 1 — strzałki ilustrują kierunek dziedziczenia). Jak widać na rysunku, atrybuty są dziedziczone i nie ma potrzeby powtarzać ich definicji w klasie dziedziczącej. Obiekty klas dziedzicznych generuje się w taki sposób, jak obiekty klas zwykłych. Przypuśćmy, że istnieją następujące deklaracje:

```
T_32:tank, Mercedes_Benz:bus, Ford:passenger_car;
```

Wówczas można utworzyć obiekty oraz określić ich atrybuty w sposób następujący:

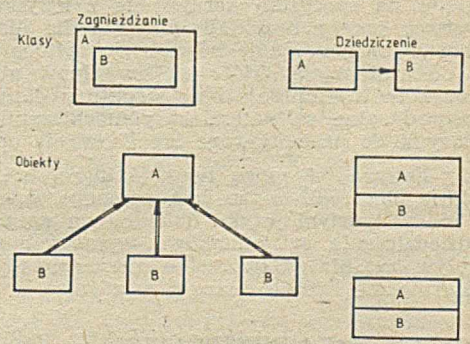
```
T_32:=new tank, T_32.dead_weight:=10_000;
T_32.car_seize:=8;
Mercedes_Benz:=new bus (19876);
Mercedes_Benz.dead_weight:=10_000;
Mercedes_Benz.seats:=89;
Ford:=new passenger_car (1111, "Granada");
```

Jak wiadomo klasa może mieć określone pewne akcje. Jeżeli klasa dziedziczy klasę z akcjami, to akcje te wykonują się w trakcie generowania obiektu klasy przed wykonaniem akcji klasy właściwej. Jeżeli wyżej określone klasy będą miały pewne akcje (po przerobieniu ich deklaracji):

```
vehicle:class (dead_weight:real);
begin
  if dead_weight<0 or dead_weight > 1.0E20
  then
    call alarm
  fi;
end;
bus of car:class (seats:integer);
passenger_seats:integer;
begin
  passenger_seats:=seats—1;
end;
```

to wykonanie instrukcji generowania obiektu "bus" spowoduje, po przekazaniu parametrów, sprawdzenie, czy "dead weight" spełnia warunki brzegowe, a następnie — określenie wartości zmiennej "passenger seats":

```
Mercedes_Benz:=new bus (10_000,19876,89);
```



Rys. 2. Ilustracja różnicy między dziedziczeniem i zagnieżdżaniem

Dziedziczenie jest formą strukturalną zupełnie inną niż zagnieżdżanie. Zagnieżdżenie klasy B w klasie A decyduje o tym, że po generowaniu obiektu klasy A można wygenerować wiele obiektów klasy B zależnych od tego wygenerowanego obiektu klasy A (p. przykłady klasy "domek", gdzie generuje się wiele obiektów klasy "izba"). Z dziedziczeniem jest całkiem inaczej. Jeżeli klasa B dziedziczy klasę A, to decyduje o tym, że każde wygenerowanie obiektu klasy B automatycznie tworzy stowarzyszony obiekt klasy A tylko na potrzeby tego obiektu klasy B. Tworzenie takiego obiektu jest automatyczne i nie trzeba dla niego stosować oddzielnie instrukcji **new** (p. przykład ostatni). Tę różnicę między dziedziczeniem i zagnieżdżaniem zilustrowano na rys. 2.

KLASY I STRUKTURY DANYCH

Jedną z ciekawszych właściwości klasy jest możliwość wyposażenia jej w atrybuty proceduralne. Atrybuty takie mogą być wywoływane w obiektach klasy w sposób zdalny. Daje to programiście narzędzie wyśmienicie nadające się do implementowania struktur danych. Można to zilustrować oklepanym już przykładem struktury stosu:

```
push_down:class (size:integer);
  stack:array [1:size] of object;
  top:integer;
  push:procedure (x:object);
  begin
    if top > size then write ("stack overflow")
    else
      stack [top] :=x; top:=top+1
    fi
  end push;
  pop:function:object;
  begin
  if top <=1 then write ("empty stack")
    else
      top:=top—1; pop:=stack [top]
    fi
  end pop;
begin
  top:=1

end push_down;
```

Klasa "push_down" ma tzw. pole zmiennych wspólnych, tj. tablicę "stack" i zmienną "top", oraz dwie procedury "push" i "pop". Procedury te działają na wspólnym polu danych. Jeżeli utworzy się obiekt klasy "push down" to można wywoływać w sposób zdalny obie procedury, uzyskując zamierzony efekt — tzn. można wstawić i wyjmować z tak utworzonego stosu potrzebne obiekty. Nie należy troszczyć się przy tym o wewnętrzną strukturę tej klasy, gdyż zamknięta „w pudle" użycza swoich możliwości tylko przez „wystawione na zewnątrz" procedury. Obiektów stosu można utworzyć tyle, ile potrzeba w programie, i każdy z nich będzie miał swoje niezależne wspólne pole danych:

```
store1, store2, store3: push_down; ob1, ob2, ob3:object;
n,m:integer;

  ...
  store1:=new push_down (1000);
  store2:=new push_down 200);
  store3:=new push_down (n+m);

  ...
  store1.push(ob1); store2.push(ob2);
  ob3:=store2.pop; store3.push(ob3);
  ...
```

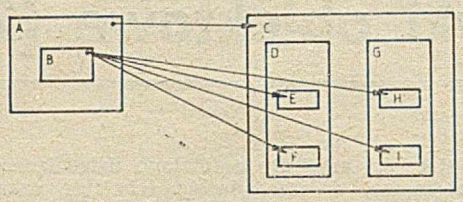
Ze względu na specyfikę korzystania z atrybutów proceduralnych, w języku Smalltalk przyjęto dla nich nietradycyjną nazwę, nazywając je mianowicie metodami, w odróżnieniu od zmiennych, które są zwykłymi atrybutami. Klasa wraz z metodami daje opis struktury danych, atrybuty nieproceduralne są natomiast pomocniczymi jednostkami służącymi do prawidłowego zrealizowania operacji.

Struktury danych też mogą tworzyć hierarchię. W tworzeniu takiej hierarchii może być pomocne dziedziczenie. Jednakże nie w każdym języku obiektowym wszystkie możliwości dziedziczenia są jednakowo wykorzystane. Ma to istotny wpływ właśnie na tworzenie hierarchii struktur danych. Postaram się krótko omówić te różnice.

DZIEDZICZENIE A ZAGNIEŻDŻANIE

Jak wspomniałem w poprzednich punktach, dziedziczenie jest zupełnie inną formą strukturalną niż zagnieżdżanie. Jednakże te dwie formy muszą jakoś współistnieć. Niestety, nietrudno jest wprowadzić do języka wiele konstrukcji, ale znacznie trudniej jest znaleźć dla nich wszystkich efektywną implementację. Ponadto może się zdarzyć, że wprowadzone bardzo swobodne pomysły nie dają jednolitej, niesprzecznej semantyki — a to już może prowadzić do okropnych nieszczęść. Z dziedziczeniem mamy właśnie taką sytuację. Jak w tych czterech językach obiektowych wybrnięto z powyższych kłopotów?

W języku Simula-67 jest zagnieżdżanie i dziedziczenie. Jednakże na dziedziczenie, właśnie ze względów implementacyjnych, nałożono wiele ograniczeń. Najistotniejszy dotyczy związku między poziomem zagnieżdżenia w klasie dziedziczącej i dziedziczonej. Otóż definicja języka mówi, że poziom zagnieżdżenia obu tych klas musi być ten sam, tzn. klasa z „płytszego" poziomu zagnieżdżenia nie może być dziedziczona w „głębszym" poziomie zagnieżdżenia, jakkolwiek jest widoczna. Takie dziedziczenie nazywa się jednopoziomowym. Na rys. 3 zilustrowano sytuację, gdy dziedziczenie nie spełnia takiego warunku.



Rys. 3. Niedopuszczalne dziedziczenie klas w Simuli-67

Klasa A reprezentuje strukturę danych działającą na obiektach klasy B, zagnieżdżonej w klasie A (jest to dosyć typowa sytuacja). Klasa C jest rozszerzeniem klasy A, a więc ją dziedziczy. Klasy E i F są rozszerzeniem klasy B, zatem ją dziedziczą. Natomiast ze względów organizacyjnych, w klasie C należy powiązać ze sobą klasy E i F przez zagnieżdżenie ich w pomocniczej klasie (lub procedurze) D. Podobnie należy postąpić z klasami H i I, zagnieżdżonymi w klasie G. Wówczas poziom klasy B musi być inny niż poziom klas E, F, H oraz I, gdyż między tymi poziomami musi wystąpić poziom klas D i G.

W Simuli-67 jest jeszcze inne nieprzyjemne ograniczenie. Mianowicie klasa, w której zagnieżdżono inną klasę, nie może być używana bezpośrednio do tworzenia obiektów, a jedynie jako pomocnicza klasa, którą można dziedziczyć w bloku (nie będę tego technicznego zjawiska szerzej wyjaśniać, bo to nie ma większego teoretycznego znaczenia). Konstrukcja, którą przedstawiono na samym początku, z "domkiem" w którym zagnieżdżono "kuchnię", nie jest verbatim wyrażalna w Simuli.

W języku Smalltalk można zagnieżdżać jedynie procedury i funkcje (czyli metody). Język ma strukturę płaską — są klasy i dziedziczenie, nie ma natomiast możliwości zagnieżdżenia klas.

W języku Loglan zniesiono ograniczenie Simuli, zarówno dotyczące jednopoziomowości dziedziczenia, jak również ograniczonej używalności klasy, w której zagnieżdżono inną klasę. Wszystkie możliwości dziedziczenia i zagnieżdżenia są wykorzystane.

Język Paragon idzie jeszcze dalej. Dziedziczenie przedstawiano dotąd jako formę strukturalną pozwalającą na kolejne rozszerzenia klasy, a więc definiowana klasa dziedziczyła bezpośrednio co najwyżej jedną klasę. Oczywiście, klasa dziedziczona może być także klasą dziedziczącą, niemniej w jednym kroku dziedziczenia zakładano istnienie tylko jednego przodka. Można wyobrazić sobie bardziej uniwersalną formę strukturalną, pozwalającą na dziedziczenie jednoczesne wielu klas — taki mechanizm dopuszcza Paragon.

Aby zilustrować taki sposób dziedziczenia, wrócę do przykładu dotyczącego klasy "vehicle". Otóż w przykładzie tym występują klasy: "vehicle", "car", "ship", "tank" itd. Klasy "car", "ship" i "tank" dziedziczą "vehicle". Przypuśćmy, że należy wprowadzić klasę dziedziczącą "ship" i "tank", np.:

```
monitor of ship, tank:class

  cannon:integer;
end monitor;
```

W klasie "monitor" klasa "vehicle" jest dziedziczona dwukrotnie, raz za pośrednictwem "ship", a drugi raz za pośrednictwem "tank". A zatem obiekt tej klasy może posiadać dwa razy pole danych klasy "vehicle", ale nie musi. Sprawa pozostaje do dyspozycji projektanta języka. Widać więc pewne niejednoznaczności w semantyce języka, które muszą być rozstrzygnięte. Z drugiej strony, nie uprzedzony użytkownik może mieć wątpliwości w zrozumieniu takiej formy strukturalnej. Co więcej, wydaje się, że w pewnych sytuacjach wygodniej byłoby mieć semantykę z powtarzalnym polem danych, a czasem z jednym.

Autor języka Paragon przyjął, z konieczności, semantykę z jednym polem danych, jeżeli z powodu dziedziczenia

jedna klasa jest wielokrotnie dziedziczona. Wprowadzone w Paragonie dziedziczenie nazywa się umownie wielodziedziczeniem, chociaż ten termin obejmuje nie jedną, ale wszystkie dopuszczalne semantyki.

Dziedziczenie w Simuli i Smalltalku jest jednopoziomowe. W Loglanie dziedziczenie dopuszcza wielopoziomowość. W Paragonie istnieje możliwość wielodziedziczenia wielopoziomowego. Nie ma tu miejsca na analizę trudności semantycznych, które pojawiają się przy tak skomplikowanych konstrukcjach. Jest to zadanie dla projektantów języka i realizatorów jego implementacji. Trzeba jednak zaznaczyć, że pojawienie się języków obiektowych, wraz z różnymi formami dziedziczenia, stworzyło nową dziedzinę informatyki, w której zarówno doświadczenie praktyczne, jak i zdolności teoretyczne odgrywają niebagatelną rolę.

Czytelnik mógłby zapytać, czy języki obiektowe i całe to okropne wielodziedziczenie wielopoziomowe nie jest zbyt trudne w zwykłym programowaniu. Wydaje się, że nie powinien mieć takich obaw. Klasa, obiekt i dziedziczenie są bardzo naturalnymi pojęciami i należy je rozumieć w sposób naturalny. Dziedziczenie wielopoziomowe może być przez użytkownika nawet niedostrzeżone, bo polega na tym, że wolno dziedziczyć to, co jest widoczne — powiedziałbym nawet przekornie, że dziedziczenie jednopoziomowe powinno drażnić użytkownika, bo dlaczego ma istnieć takie dziwne ograniczenie? Wielodziedziczenie jest też naturalnym rozszerzeniem jednodziedziczenia i jeżeli użytkownik nie natrafi jakiejś skomplikowanej pułapki semantycznej, nie będzie się zastanawiał, co może się kryć dziwnego za tymi pojęciami.

Mądre używanie języka obiektowego przynosi użytkownikowi wiele korzyści. Otrzymuje się szybciej produkt szybciej działający. Co więcej, umożliwia ono w elastyczny sposób dopasowanie świata pojęć informatycznych do świata opisywanego. Dziedziczenie ułatwia hierarchizację. A na tym opiera się przecież właściwa organizacja wszelkich systemów, także komputerowych.

LITERATURA

[1] Bartol W. M. et al.: Report on the Loglan 82 programming language. PWN, Warszawa, 1984
[2] Bartol W. M., Kreczmar A., Litwiniuk A., Oktaba H.: Semantics and implementation of prefixing at many levels. Lecture Notes in Computer Science, Vol. 148, pp. 45—80, 1980
[3] Dahl O. J., Myhrahaus B., Nygaard K.: Common base langue. Norwegian Computer Centre, 1970
[4] Ingalls D.: The Smalltalk 76 programming system design and implementation. Proc. POPL, pp. 9—16, 1978
[5] Krause M., Kreczmar A., Langmaack H., Warpechowski M.: Concatention of program modules, an algebraic approach to the semantics and implementation problems. Lecture Notes in Computer Science, Vol. 208, pp. 134—156, 1982
[6] Salwicki A.: On the algoritmic theory of stacks. Lecture Notes in Computer Science, Vol. 64, pp. 452—461, 1978
[7] Sherman M. S.: Paragon. Lecture Notes in Computer Science, Vol. 189, 1985

Nowe książki

WYDAWNICTWA NAUKOWO-TECHNICZNE

Stanisław Borak, Jerzy Klaczak, Stanisław Korczak, Zdzisław Płoski „System operacyjny George 3". Warszawa 1987, 5800 egz., 342 str., cena 370 zł

Książka jest podręcznikiem dla użytkowników maszyny cyfrowej Odra 1305 pracującej pod kontrolą systemu operacyjnego George 3 (przyjmowanie, uruchamianie, planowanie i wykonywanie zadań). Do komunikacji z tym systemem służy język opisu zadań. W książce wyjaśniono zasady obsługiwania użytkownika przez system oraz podano podstawowy zbiór zleceń (instrukcji) umożliwiający rozpoczęcie, prowadzenie i zakończenie pracy, poprawianie błędów za pomocą edytora oraz spis tzw. makrozleceń, służących do posługiwania się językami programowania: Algol, Plan, Fortran itd.

Książka jest przeznaczona dla programistów, projektantów systemów przetwarzania informacji oraz dla studentów kierunków informatycznych.

Rolf Hedtke „Systemy mikroprocesorowe — niezawodność, testowanie, tolerancja błędów" (tłum. z jęz. niemieckiego). Warszawa 1987, 6800 egz., 176 str., cena 500 zł

W książce omówiono problemy związane z zapewnieniem wysokiej jakości systemów mikrokomputerowych: środki dla ochrony przed skutkami uszkodzeń podzespołów; metody testowania układów scalonych, wpływ projektowania sprzętu i oprogramowania na niezawodność systemu; stosowanie technik redundancyjnych; kody wykrywania i korygowania błędów.

Książka jest przeznaczona przede wszystkim dla projektantów i użytkowników systemów mikroprocesorowych; może być książką pomocniczą dla studentów kierunków informatyki i elektroniki wyższych uczelni technicznych.

Theo Pavlidis „Grafika i przetwarzanie obrazów — algorytmy" (tłum. z jęz. angielskiego). Warszawa 1987, 6800 egz., 392 str., cena 420 zł

Książka zawiera omówienie sposobów kodowania i przekształcania obrazów od pełnej skali kolorów lub odcieni szarości (przy czarno-białych obrazach) do obrazów składających się z krzywych, odcinków prostych lub punktów. Szczególny nacisk położono na narzędzia matematyczne, które w przeciwieństwie do urządzenia nie dezaktualizują się. Od Czytelników jest wymagana znajomość podstaw informatyki, rachunku różniczkowego i całkowego, podstaw statystyki i teorii grafów, geometrii, przetwarzania sygnałów.

Książka jest przeznaczona dla programistów, projektantów systemów przetwarzania informacji, pracowników nauki zajmujących się informatyką oraz dla studentów kierunków informatycznych.

RYSZARD S. MICHALSKI
University of Illinois at Urbana-Champaign
USA

O naturze uczenia się — problemy i kierunki badawcze (I)

Artykuł zawiera przegląd zadań i kierunków badawczych w dziedzinie maszynowego uczenia się, mając w zamierzeniu autora służyć jako przewodnik w tej dziedzinie. Omówiono w nim zasadnicze aspekty procesu uczenia się, sklasyfikowano główne kierunki badawcze oraz przedstawiono poglądy autora na temat związków pomiędzy paradygmatami, strategiami i orientacjami uczenia się.

CZY POTRZEBNE SĄ MASZYNY UCZĄCE SIĘ?

Sztuczna inteligencja przeżywa obecnie niebywały rozkwit. Jej idee i wypracowane przez nią metody znajdują zastosowanie w wielu innych dziedzinach. Rozwój systemów ekspertowych, praktyczne implementacje systemów rozumienia języka naturalnego, znaczące postępy w dziedzinie komputerowych systemów wizyjnych i systemów rozumienia mowy, nowe spojrzenie na budowę skutecznych systemów wnioskujących i systemów rozumowania jakościowego (ang. qualitative reasoning) stanowią najbardziej widoczne i najważniejsze osiągnięcia. Szybkie rozszerzanie się sfery wpływów sztucznej inteligencji pozwala oczekiwać w najbliższej przyszłości jej nowych sukcesów.

W tym kontekście ważne staje się pytanie, jakie są ograniczenia obecnych metod sztucznej inteligencji i w jakim kierunku rozwiną się badania w tej dziedzinie. Jedno z oczywistych ograniczeń, wyznaczające zarazem kierunek przyszłych badań, dotyczy **maszynowego uczenia się.** Zagadnienie to jest związane z rozwojem obliczeniowej teorii uczenia się i budową systemów uczących się.

Obecne systemy sztucznej inteligencji, za wyjątkiem tych, których budowa ma na celu badanie maszynowego uczenia się, przejawiają niewielkie lub nie przejawiają żadnych umiejętności uczenia się. Cała wiedza, jaką dysponują, musi być uprzednio przygotowana i zaprogramowana. Jeśli w trakcie ich pracy wystąpi błąd, to nie potrafią samodzielnie go poprawić. Jest on powtarzany tyle razy, ile razy dana procedura jest wykonywana. Systemy te nie potrafią też doskonalić się w miarę zbierania doświadczeń ani gromadzić wiedzy ze swej dziedziny drogą eksperymentowania. Nie są w stanie automatycznie generować dla własnych potrzeb nowych algorytmów, formułować nowych abstrakcyjnych pojęć ani proponować nowych rozwiązań przez analogię do dotychczasowych czy też drogą swoistego procesu odkrywania (ang. discovery). Ogólnie rzecz biorąc, systemom tym brakuje umiejętności **indukcyjnego wyciągania wniosków** z podanej im informacji. Można powiedzieć, że niemal wszystkie istniejące systemy są dedukcyjne, gdyż wyciągają wnioski opierając się na dostępnej wiedzy i nie potrafią samodzielnie przyswajać ani generować nowej wiedzy.

Dla porównania, jedną z najbardziej uderzających cech w inteligentnym zachowaniu się człowieka jest zdolność przyswajania nowej wiedzy, nabierania wprawy i doskonalenia przez praktykę. Wykorzystanie możliwości uczenia się czyni z młodego, niedoświadczonego człowieka wykwalifikowanego inżyniera, wychowawcę, artystę lub lekarza. W powszechnym rozumieniu osoba, która powtarza wciąż te same błędy, raczej nie zasługuje na miano inteligentnej. Umiejętność uczenia się na błędach jest uważana za podstawową cechę zarówno człowieka, jak i całego społeczeństwa [1, 6, 13, 14, 28, 29].

Skoro zdolność uczenia się wydaje się być tak ściśle związana z inteligentnym zachowaniem, obecny stan rozwoju sztucznej inteligencji skłonił niektórych badaczy do wysunięcia tezy, że jednym z najważniejszych celów badań w tej dziedzinie powinno być zrozumienie istoty procesu uczenia się oraz wyposażenie maszyn w zdolność uczenia się [19, 33]. Celowość przezwyciężenia wspomnianych ograniczeń wyznacza więc zadania badawcze na przyszłość.

W tym miejscu rodzi się pytanie: czy taki cel jest osiągalny, a jeśli tak, to czy jest pożądany. Próba odpowiedzi na pytanie o możliwość osiągnięcia tego celu zmusza do sprecyzowania niektórych pojęć. Czy można wskazać ogólne kryteria, których spełnienie zapewnia, że maszyna ma zdolność uczenia się?

Jak tego dowiodły badania w dziedzinie maszynowego uczenia się, zdolność uczenia się nie jest właściwością podlegającą dwuwartościowej ocenie: „tak lub nie". Jest z nią związana cała gama czynności polegających na przetwarzaniu informacji: od bezpośredniego zapamiętywania faktów i prostego organizowania informacji, do bardzo skomplikowanych procesów wnioskowania, prowadzących do tworzenia nowych pojęć i odkrywania nowej wiedzy. Zawsze pociąga to za sobą zmiany w systemie — człowieku lub maszynie — które ulepszają go w określonym sensie.

Odkładając do następnego punktu odpowiedź na pytanie o definicję uczenia się, należy stwierdzić, że maszynowe uczenie się przeżywa obecnie swoje odrodzenie po okresie stabilizacji i powolnego rozwoju. Wysiłki zmierzające do budowy programów przejawiających umiejętności uczenia się zostały w ostatnich latach znacznie zintensyfikowane. Ta młoda dyscyplina ma już na swym koncie pewne sukcesy. Rezultaty niektórych spośród tych wysiłków opisano w [21]. Kontynuacją tej pracy jest książka [22], prezentująca kluczowe zagadnienia i współczesny stan wiedzy w dziedzinie maszynowego uczenia się.

Opierając się na dotychczasowych rezultatach, można dojść do wniosku, że już dziś możliwe jest wyposażenie maszyn w elementarne zdolności uczenia się. Istnieją już programy zdolne formułować nowe pojęcia i wykrywać nie znane dotąd regularności w danych, tworzyć reguły decyzyjne lepsze od reguł formułowanych przez człowieka, dostarczać interesujące analogie, automatycznie uczyć się heurystyk pomagających w rozwiązywaniu problemów lub budować uogólnione plany akcji prowadzących do osiągania określonego celu. Wiele z tych programów omówiono w [21]. Mniej jasne jest, jaki postęp w dziedzinie maszynowego uczenia się można osiągnąć przy użyciu konwencjonalnego sprzętu komputerowego i obecnych metod programowania. Jak zawsze w nauce, na takie pytania można odpowiedzieć, kontynuując badania i budując eksperymentalne systemy uczące się.

Nowe kierunki badań w dziedzinie maszynowego uczenia się otwierają się wraz z rozwojem tzw. maszyn neuronowych (ang. connection machines), systemów komputerowych piątej generacji i innych nowatorskich architektur komputerowych [8, 12]. Na przykład, Hinton, Sejnowski i Ackley [9] opisują uczenie się w tzw. maszynach Boltzmanna. Wiedza gromadzona przez te maszyny jest reprezentowana jako trwałość połączeń między prostymi elementami podobnymi do neuronów. Badania prowadzone w tym kierunku powinny umożliwić pokonanie ograniczeń wczesnych systemów tego rodzaju, takich jak perceptrony [26]. Nowy potencjał badawczy w dziedzinie maszynowego uczenia się wyłania się w związku z rozwojem nowych systemów programowania, a w szczególności programowania

logicznego i jego pierwszego wcielenia w postaci Prologu [31].

Dlaczego konieczna jest budowa maszyn uczących się? Wydaje się, że ich rozwój jest warunkiem dalszego postępu w badaniach nad sztuczną inteligencją i w dyscyplinach pokrewnych. Jest to szczególnie istotne w takich dziedzinach, jak: systemy ekspertowe i systemy z bazami wiedzy, komputerowe systemy wizyjne, rozumienie mowy, rozumienie języka naturalnego, inteligentne systemy edukacyjne oraz prawdziwe przyjazne systemy współpracy człowiek-maszyna. Im bardziej skomplikowane zadanie stawia się przed systemami sztucznej inteligencji, tym więcej wiedzy trzeba w nich gromadzić. Wiedza taka musi obejmować fakty i reguły specyficzne dla danej dziedziny, zdroworozsądkowe heurystyki i ograniczenia, jak również ogólne pojęcia i teorie dotyczące otaczającego świata. Zakres wiedzy każdego systemu musi być tak poszerzony, aby uniknąć powszechnie występującego dziś zjawiska, nazywanego „przepaścią wiedzy" (ang. konwledge cliff) [5] albo „kruchością wiedzy" (ang. brittleness) [24] (rozdział 20), [15]. Polega ono na tym, że system pracuje poprawnie w wyznaczonej mu dziedzinie, natomiast nieznaczne wyjście poza nią drastycznie pogarsza jego właściwości. J. H. Holland w rozdziale 20 [24] omawia uniwersalne algorytmy uczące się, oparte na równoległej architekturze regułowej. Twierdzi on, że rozumowanie indukcyjne w takich systemach pomoże przezwyciężyć „kruchość" dzisiejszych systemów sztucznej inteligencji, biorącą się ze zbyt wąskiego ukierunkowania ich wiedzy.

Wypełnianie bazy wiedzy nowego systemu jest procesem bardzo złożonym, czasochłonnym i podatnym na błędy, wymagającym dużego doświadczenia. Na przykład, budowa systemu ekspertowego wymaga wspólnego wysiłku wysoko wykwalifikowanych specjalistów — przynajmniej jednego eksperta z danej dziedziny oraz inżyniera wiedzy [2, 4, 7]. Zadanie to można uprościć stosując techniki uczenia się, które umożliwiają automatyczne konstruowanie reguł decyzyjnych na podstawie przykładów decyzji eksperta oraz drogą automatycznej analizy zawartość bazy faktów.

Szybki wzrost liczby danych i wiedzy gromadzonej przez społeczeństwo wymaga nie tylko gromadzenia, organizowania i udostępniania informacji, lecz również wykorzystania jej w nowy, twórczy sposób. Ponieważ wiedzę można rozpatrywać jako skondensowaną informację [30], potrzebujemy maszyn potrafiących dokonywać kompresji baz danych i systemów informacyjnych na bazy wiedzy drogą automatycznej analizy pojęciowej ich treści. Jak podkreśla Michie [25], „najbardziej frapującym technicznie, nawet jeśli obecnie nie najważniejszym ekonomicznie, wyzwaniem jest to, w jaki sposób zastosować komputery nie tylko do bezpośredniej obserwacji badań naukowych, prowadzonych za pomocą teleskopów, mikroskopów, komór iskrowych i innych urządzeń, lecz również w procesach rozpoznawania i wnioskowania, dzięki którym chaos danych ostatecznie przeradza się w odkrycie naukowe".

Autor tego artykułu sądzi, że oprócz komputerów pełniących rolę inteligentnych asystentów naukowca i technika, będziemy potrzebowali inteligentnych asystentów osobistych. Obywatele rozwijającego się społeczeństwa informacyjnego będą potrzebowali ich po to, aby poradzić sobie z przytłaczającą masą informacji i złożonością codziennych procesów podejmowania decyzji. Aby asystenci mogli spełniać wyznaczoną im rolę, ich funkcjonowanie i wiedza powinny być dynamiczne; powinni oni łatwo przystosowywać się do zmniejszających się wymagań i mieć umiejętność automodyfikacji. Inaczej mówiąc, powinni dysponować zdolnością uczenia się.

Umiejętność uczenia się jest również niezbędna w dziedzinie wizji komputerowej i rozumienia mowy. Aby zbudować komputerowy system wizyjny, należy wprowadzić do niego wiele transformacji charakterystycznych dla wizji, pojęć geometrycznych oraz opisów fizycznych i funkcjonalnych obiektów wizualnych, które system ma rozpoznawać [38, 39]. Ręczne wprowadzenie tej informacji jest trudne. Znacznie łatwiejsze byłoby nauczenie systemu konkretnych pojęć przez pokazanie mu przykładów tak, aby mógł przyswoić sobie odpowiednie uogólnienia i opisy w sposób, w jaki czynią to ludzie.

System zdolny do rozumienia języka naturalnego i komunikowania się, za jego pomocą z człowiekiem powinien mieć wiedzę o składniowych właściwościach języka [17], jak również uwzględniać wiele pojęć i ich struktur (takich jak ramy, scenariusze (ang. scripts) i schematy), opisujących semantyczne i pragmatyczne aspekty języka [24], rozdział 19 i 21, [32, 37]. Szacuje się, że w zaawansowanym programie rozumiejącym język naturalny, liczba takich pojęć i struktur pojęciowych może osiągnąć dziesiątki tysięcy, a nawet więcej. Zaprogramowanie całej tej wiedzy jest ogromnym zadaniem. Dobrze byłoby więc uprościć je, wykorzystując techniki uczenia się. Gdyby nawet w pewnym momencie maszyna dysponowała pełną wiedzą, system rozumiejący język naturalny nie mógłby i tak długo pracować poprawnie, nie mając zdolności uczenia się. Znaczenie pojęć używanych przez człowieka jest bowiem dynamiczne. Zmienia się ono z upływem czasu, aby dostosować się do nowych okoliczności. Nowe pojęcia powstają i rozwijają się, inne natomiast wychodzą z użycia. Dlatego też, tak jak w podanych wypadkach potrzebny jest system uczący się, zdolny do przyswajania nowych pojęć i struktur pojęciowych drogą uogólniania przykładów lub przez analogię do poprzedniej wiedzy. System taki powinien być zdolny do elastycznego modyfikowania, ukonkretniania i uogólniania starych pojęć.

Inteligentne systemy edukacyjne powinny operować materiałem dydaktycznym na poziomie dostosowanym do poziomu wiedzy ucznia. Aby sprostać temu zadaniu, system musi kontrolować wiedzę ucznia i nadążać za jej zmianami. Pożądane jest uzyskiwanie tej informacji nie drogą cyklicznego i bezpośredniego sprawdzania, lecz drogą uczenia się w trakcie interakcyjnych sesji na podstawie wskazówek, przyjętego modelu ucznia oraz jego zachowania. Tak więc, zdolność uczenia się jest wymagana nie tylko od ucznia, ale także od systemu edukacyjnego [35, 36].

Dzięki umiejętności uczenia się, komputery przyszłości powinny być zdolne do zdobywania wiedzy bezpośrednio z dokumentów i książek drogą konwersacji z ludźmi i uogólniania obserwacji dokonywanych sztucznymi zmysłami. Powinny być zdolne do doskonalenia się przez praktykę i zdobywanie doświadczenia. Przyszłe systemy uczące się nie będą podlegały ograniczeniom, jakim podlegają ludzie (krótka pamięć, trudności w skoncentrowaniu uwagi, niska efektywność, trudności w przekazywaniu nabytej wiedzy). Jeśli powstanie jeden egzemplarz jakiegoś systemu uczącego się, to możliwe będzie zbudowanie teoretycznie nieograniczonej liczby jego kopii, które będą mogły zdobywać wiedzę w różnorodnych dziedzinach. W dodatku każdą nową porcję wiedzy zdobytej przez system uczący się będzie można skopiować do innego systemu szybko i tanio (inaczej niż w wypadku wiedzy zdobywanej przez ludzi, która musi być starannie przekazywana każdemu nowemu uczniowi).

Oczywiście daleko jest jeszcze do tej idealnej wizji, można sobie jednak wyobrazić, że takie systemy uczące się będą mogły być budowane w przyszłości. Warto przy tym rozważyć nie tylko spodziewane korzyści, lecz także niepożądane konsekwencje. Ten drugi problem można zresztą ominąć stwierdzając, że każda nowa technologia daje możliwość użycia jej w niewłaściwym celu, a to jak dotąd nie powstrzymywało człowieka przed ich rozwijaniem. Na ogół uważa się, że zagadnienia te leżą poza dziedziną badań naukowych czy technicznych. Mimo to, należy dokładnie je zbadać, ponieważ stworzenie maszyn zdolnych do samodzielnego zdobywania wiedzy wprowadza dodatkową złożoność i wpływa na wybór drogi dalszego rozwoju maszynowego uczenia się.

Podstawowym problemem jest „nieprzezroczystość" systemów samomodyfikujących się. Przewidywanie zachowania maszyn umiejących uczyć się indukcyjnie jest znacznie trudniejsze niż przewidywanie zachowania maszyn nie mających tej umiejętności. Najważniejszą właściwością maszyn uczących się jest zdolność tworzenia wiedzy mogącej zaskoczyć samych twórców. Może to zresztą prowadzić do niespodziewanych trudności, a nawet niebezpieczeństw, jeśli użyje się takiego systemu do rozwiązania poważnych zadań bez zrozumienia jego ograniczeń. Ponadto zwiększenie nieprzewidywalności działania maszyn uczących się zwiększa niebezpieczeństwo ich nieprawidłowego użycia.

Niektórzy eksperci twierdzą, że przewidywanie zachowania złożonych systemów komputerowych jest już teraz wystarczająco trudne. Wyposażenie komputerów w zdolność uczenia się uważają oni za powiększenie tych trudności, nie zaś za jakościowy krok naprzód. Niezależnie od punktu widzenia należy jednak spodziewać się, że potencjalne korzyści tej technologii sowicie wynagrodzą nam jej niepożądane konsekwencje. Co zaś się tyczy możliwości ich

niewłaściwego użycia, to dlaczego nie można wykorzystać tych sprytnych maszyn uczących się do nadzorowania innych, aby zapobiec ich nadużyciu?

Oprócz trudności w przewidywaniu zachowania maszyn uczących się, warto rozważyć jeszcze jedno zagadnienie, tkwiące korzeniami w samej naturze każdego rodzaju wiedzy różnej od wiedzy zdobytej w wyniku prostej obserwacji faktów. Jak zauważył Hume, a później Popper [11, 28] i inni, wiedza taka ze swej natury ma charakter domyślny. Znaczy to, że w zasadzie nie można udowodnić poprawności żadnej wiedzy tworzonej drogą uogólniania określonych obserwacji lub przez analogię do znanych faktów, choć można dowieść jej błędności.

Jest tak dlatego, że rozumowanie przez indukcję nie zapewnia zachowania „prawdy" (ang. truth-preserving), lecz zachowuje jedynie „fałsz" (ang. falsity-preserving), tzn. z prawdziwych informacji nie zawsze wyprowadza się prawdziwe informacje, natomiast z fałszywych informacji zawsze wyprowadza się fałszywe [21, 22, 23]. Aby to zilustrować, rozważmy zdanie: *„Wszyscy naukowcy w Laboratorium Sztucznej Inteligencji w Massachusetts Institute of Technology MIT są błyskotliwi"*. Drogą dedukcji można wyprowadzić z tego zdania konkluzję, że Roger Light, pracujący w Laboratorium Sztucznej Inteligencji, jest błyskotliwy. Jeśli pierwotna przesłanka jest prawdziwa, to ten wniosek dedukcyjny musi także być prawdziwy. Natomiast przykładem wniosku uzyskanego z pierwotnej przesłanki drogą indukcji jest następujący: *„Wszyscy naukowcy w MIT są błyskotliwi"*. W tym wypadku, prawdziwość pierwotnej przesłanki nie gwarantuje prawdziwości wniosku uzyskanego drogą indukcji. Jeśli jednak pierwotna przesłanka jest fałszywa, to wniosek indukcyjny również jest fałszywy. Tak więc, inaczej niż w systemie dedukcyjnym, poprawne dane wejściowe do systemu indukcyjnego nie gwarantują otrzymania poprawnych wyników. Ponadto każdemu zestawowi danych wejściowych odpowiada nieskończona liczba dających się z niego wyciągnąć wniosków indukcyjnych. Te, które są akurat wybierane, odzwierciedlają preferencje, założenia i ograniczenia, stosowane przy formułowaniu uogólnień [20, 24] rozdział 5.

Z tych powodów, aby można generować wiedzę przydatną dla człowieka, maszyny uczące się powinny być wyposażone w wiedzę dotyczącą wszystkich istotnych ograniczeń i założeń przyjmowanych przez ludzi. Ponieważ jest mało prawdopodobne, aby wszelkie subtelne ograniczenia i preferencje jednostek i społeczeństw były kiedykolwiek poznane przez maszyny, jest możliwe, że wiedza generowana przez nie będzie naruszać niektóre z tych ograniczeń. Warto przytoczyć pogląd Hofstadtera [10]: *„jeżeli program nie będzie dysponował wierną kopią ludzkiego ciała..., to prawdopodobnie będzie miał zupełnie odmienny punkt widzenia na to, co jest ważne, interesujące itd."*. Ponieważ postrzeganie tego, co jest istotne i interesujące, stanowi nieodzowny element tworzenia nowej wiedzy [16], odmienność ta jest znacząca. Gdy wiedza wytworzona przez maszynę zostanie wykorzystana w praktyce, będzie można otrzymać rozwiązania poprawne z technicznego, jednak nie do przyjęcia ze społecznego punktu widzenia.

Wiąże się z tym niebezpieczeństwo zbyt dużego zaufania do wiedzy tworzonej przez maszyny. Podobne zjawisko zaobserwowano, na przykład, u ludzi nadmiernie ulegających wpływowi komputerowej analizy statystycznej, gdy nie rozumieją oni dokładnie założeń leżących u jej podstaw lub u ludzi przypisujących osobowość komputerowemu systemowi konsultacyjnemu, na przykład takiemu jak ELIZA [40]. O ile naukowcy wiedzą o tym, że wiedza uzyskana drogą indukcji jest z natury niepewna, może to być mniej oczywiste dla laików.

Powyższe rozważania prowadzą do istotnego wniosku, że wszelka wiedza wygenerowana przez maszynę powinna być poddana ścisłej weryfikacji ze strony człowieka. Ukazuje to istotny cel badawczy w dziedzinie maszynowego uczenia się: jeżeli ludzie mają rozumieć i akceptować wiedzę generowaną przez maszynę, to systemy uczące się powinny być zdolne do udzielania wyjaśnień. Ponadto wiedza tworzona przez maszyny powinna być wyrażana w formie ściśle odpowiadającej opisom przyjętym przez człowieka i jego modelom myślowym, tzn. powinna spełniać to, co autor nazywa zasadą zrozumiałości [21, 22, 23]. Projektując mechanizmy systemu uczącego się odpowiedzialne za wyjaśnienie, należy dążyć do ułatwienia człowiekowi zrozumienia nie tylko końcowych rezultatów, ale także tkwiących u ich podstaw zasad, założeń i teorii.

Ktoś może wysunąć przypuszczenie, że istnienie zaawansowanych maszyn uczących się wyeliminuje wprawdzie obecne wąskie gardło w przyswajaniu wiedzy, ale wprowadzi za to nowe — w weryfikowaniu wiedzy. Maszyny będą bowiem generowały nową wiedzę w takiej ilości, że ludziom będzie trudno ją sprawdzać i akceptować. Jeśli tak się stanie, to przyszli badacze będą mieli do rozwiązania ciekawy problem, który pozwoli im wypełnić chwile bezczynności. Oczami wyobraźni możemy już ujrzeć badaczy konstruujących wymyślne maszyny uczące się, służące do generowania eksperymentów testujących wiedzę wytworzoną przez inne maszyny.

Mając na uwadze powyższe problemy i pamiętając o tym, jak ważne jest maszynowe uczenie się, spojrzymy nieco dokładniej na zasadnicze cechy procesu uczenia się.

CZYM JEST UCZENIE SIĘ?

Jak zauważono wcześniej, uczenie się — według powszechnego przekonania — pociąga za sobą zmiany w systemie, doskonalące go w pewien sposób. Termin „doskonalenie" wymaga jednak uściślenia. Bezspornie wino „doskonali" się w miarę upływu czasu, nikt jednak nie nazwie tego uczeniem się (przykład ten pochodzi od Steve'a Tanimoto z University of Washington w Seattle). Simon [34] podaje dokładniejszą definicję: *„Uczenie się oznacza zmiany w systemie, które mają charakter adaptacyjny w tym sensie, że pozwalają systemowi wykonać następnym razem takie samo zadanie lub zadania pochodzące z tej samej populacji bardziej efektywnie"*.

Na ogół wszyscy zgadzają się z wymaganiem, aby rezultatem uczenia się była efektywniejsza praca systemu. Są jednak czynności zaliczane do uczenia się, do których trudno jest zastosować kryterium udoskonalania. Minsky w swej wnikliwej teorii procesów myślowych [27], całkowicie odrzuca ten warunek, stwierdzając: *„Uczenie się jest wprowadzaniem zmian do sposobu pracy naszych umysłów"*.

W języku angielskim uczeniem się często nazywa się uzyskiwanie nowej informacji, jak w zdaniu: *„Ponieważ satelita spłonął w atmosferze, astronauta z laboratorium kosmicznego dowiedział się (ang. learned, dosłownie — nauczył się), że satelita miał dodatkową antenę"*. W tym wypadku astronauta uzyskał po prostu pewną informację, która jest bezużyteczna, jeśli chodzi o przyszłą obsługę tego właśnie satelity. **Przyswajanie wiedzy** wydaje się być centralną czynnością w wielu aktach uczenia się. Są jednak przykłady uczenia się, gdzie przyswajanie wiedzy odgrywa raczej małą rolę, szczególnie w tzw. procesach **zdobywania wprawy** (ang. skill acquisition). Zdobywanie wprawy jest to stopniowe doskonalenie zdolności ruchowych lub poznawczych drogą ponawiania prób, z niewielkim lub żadnym udziałem świadomości [3]. Podkreśla się więc jeszcze raz aspekt przyswajania wiedzy w procesie uczenia się, na co zwrócono szczególną uwagę w książce [24].

Aby uzyskać wiedzę o czymkolwiek, trzeba ją oczywiście w jakiś sposób przedstawić, np. w postaci zdań deklaratywnych, procedur, przez połączenie obu tych reprezentacji lub też jeszcze inaczej [18]. Ten fakt i wcześniejsze rozważania prowadzą do następującego wniosku. *„Uczenie się jest konstruowaniem lub modyfikowaniem reprezentacji tego, co jest doświadczane"*.

Pojęcie **doświadczenia** obejmuje tu wszelkie bodźce zewnętrzne, jak również wewnętrzne doświadczenia myślowe. Te bodźce i eksperymenty myślowe służą do postrzegania rzeczywistości, którą system uczący się próbuje reprezentować. Wewnętrzne procesy myślowe mogą być same przedmiotem uczenia się. W ocenie konstruowanych reprezentacji bierze się pod uwagę przede wszystkim: wiarygodność, efektywność i poziom abstrakcji. **Wiarygodność** określa stopień, w jakim reprezentacja odpowiada rzeczywistości. **Efektywność** charakteryzuje przydatność reprezentacji do osiągnięcia danego celu. Im bardziej efektywna jest reprezentacja, tym lepiej funkcjonuje system. To kryterium jest najważniejsze w tych zastosowaniach, w których sprawność działania systemu jest zagadnieniem pierwszoplanowym. **Poziom abstrakcji** odpowiada zakresowi, szczegółowości i precyzji pojęć użytych w opisie. Określa on **moc opisową** reprezentacji. Te trzy kryteria stanowią łącznie o **jakości uczenia się.**

Reprezentacje mogą mieć postać opisów symbolicznych, algorytmów, modeli symulacyjnych, procedur sterujących, planów, obrazów, jak również ogólnych teorii formalnych.

Jeśli rozszerzy się pojęcie reprezentacji, aby był możliwy opis fizycznych i fizjologicznych zmian zachodzących w układzie nerwowym w trakcie zdobywania umiejętności, to rozważane procesy uczenia się obejmują również zdobywanie umiejętności.

Z tego punktu widzenia podstawowym pytaniem we wszelkich badaniach z dziedziny maszynowego uczenia się jest pytanie o formę i metodę użytą do reprezentowania i modyfikowania zdobywanej wiedzy lub umiejętności. Gdy idzie o modyfikację wiedzy, ważne jest określenie tych składowych i tych właściwości reprezentacji, które mają być modyfikowane przez system.

W systematyce badań w dziedzinie maszynowego uczenia się [3] wyróżniono trzy kryteria, jako szczególnie pomocne w klasyfikowaniu i porównywaniu metod: **strategia uczenia się, reprezentacja wiedzy i dziedzina zastosowań.** Strategia uczenia się mówi o rodzaju wnioskowania zastosowanego przez system w trakcie uczenia się. Pewne dodatkowe idee, odzwierciedlające ostatnie osiągnięcia w tej dziedzinie, przedstawiono w drugiej części tego artykułu. Kryteria reprezentacji wiedzy i dziedziny zastosowań są wyczerpująco przeanalizowane w [3], wobec czego nie będą omawiane. Zostaną natomiast przedyskutowane nieco bardziej szczegółowo dwa inne kryteria klasyfikacji: **paradygmaty badawcze** oraz **orientacje uczenia się.** Kryterium paradygmatu badawczego odnosi się tu do metod wykorzystywanych w budowie systemu, natomiast orientacja uczenia się mówi o zakresie i temacie badań.

Tłum. i oprac.
EDMUND PIERZCHAŁA
PIOTR ZIELCZYŃSKI

LITERATURA

[1] Berkson W., Wettersten J.: Learning from Error. Open Court Publishing Company, La Salle (IL), 1984
[2] Buchanan B. G., Schortliffe E. H. (eds.): Rule-based Expert Systems. Addison-Wesley, Reading (MA), 1984
[3] Carbonell J. G., Michalski R. S., Mitchell T. M.: An Overview of Machine Learning. (21)
[4] Davis R., Lenat D. B.: Knowledge-Based Systems in Artificial Intelligence. McGraw-Hill, New York, 1982
[5] Feingenbaum A. E.: First U. S. — China Joint Seminar on Automation and Intelligent Systems, Beijing, 23 May — 1 June 1984
[6] Hayes-Roth F.: Using Proofs and Refutations to Learn from Experience. (21)
[7] Hayes-Roth F., Waterman D. A., Lenat D. B. eds.: Building Expert Systems. Addison-Wesley, Reading (MA), 1983
[8] Hillis W. D.: The Connection Machine (Computer Architecture for the New Wave). AI Meo No. 646, MIT, Cambridge (MA), September 1981
[9] Hinton G. E., Sejnowski T. J., Ackley D. H.: Boltzmann Machines — Constraint Satisfaction Networks that Learn. Technical Report CMU-CS-84-119, Carnegie-Mellon University, Pittsburgh (PA), 1984
[10] Hofstadter D. R.: Gödel, Escher, Bach — Eternal Golden Braid. Vintage Books, New York, 1980
[11] Hume D.: A Treatise of Human Nature. L. S. Selby-Bigge (ed.), Clarendon Press, Oxford, 1988
[12] Kawanobe K.: Present Status of the Fifth Generation Computer Systems Project. ICOT Journal, No. 5, October 1984
[13] Kuhn T. S.: The Structure of Scientific Revolutions. Sec. edition. University of Chicago Press, 1970
[14] Lakatos I.: Falsification and the Methodology of Scientific Research Programmes. Pp. 91—196, Criticism and the Growth of Knowledge. A. Musgrave, I. Lakatos (eds.), Cambridge University Press, 1980
[15] Larkin J., Reif F., Carbonell J., Gugliotta A.: FERMI — Flexible Expert Reasoning with Multi-Domain Inferencing, Cognitive Science, 1985
[16] Lenat D. G.: The Role of Heuristics in Learning by Discovery — Three Case Studies. (21)
[17] Marcus M. P.: A Theory of Syntactic Recognition for Natural Language. MIT Press, Cambridge (MA), 1980
[18] McCarthy J.: Programs with Common Sense. Semantic Information Processing. M. Minsky (ed.), MIT Press, Cambridge (MA), 1968
[19] McCarthy J.: President's Quarterly Message — AI Needs More Emphasis on Basic Research. AI Magazine, Vol. 4, pp. 4—5, 1983
[20] Medin D. L., Wattenmaker W. D., Michalski R. S.: Constraints in Inductive Learning — An Experimental Study Comparing Human and Machine Performance, Cognitive Science, 1985
[21] Michalski R. S., Carbonell J. G., Mitchell T. M. (eds.). Machine Learning — An Artificial Intelligence Approach. TIOGA, Palo Alto (CA), 1983
[22] Michalski R. S.: Theory and Methodology of Inductive Learning. Artificial Intelligence, No. 20, 1983
[23] Michalski R. S. (ed.): Proc. of the International Machine Learning Workshop, Allerton House, University of Illinois, 22—24 June 1983
[24] Michalski R. S., Carbonell J. G., Mitchell T. M. (eds.): Machine Learning. Vol. 2, Morgan and Kaufmann Publishers, 1986
[25] Machice D.: Machine Intelligence and Related Topics. Gordon and Breach Science Publishers, New York, 1982
[26] Minsky M., Papert S.: Perceptrons. MIT Press, Cambridge (MA), 1969
[27] Minsky M.: The Society of Mind. MIT, Cambridge (w druku)
[28] Popper K. R.: The Logic of Scientific Discovery. Basic Books, New York, 1959
[29] Popper K. R.: Objective Knowledge — An Evolutionary Approach, Revised edition. Oxford, 1981
[30] Rendell L. A.: Toward a Unified Approach to Conceptual Knowledge Acquisition. AI Magazine, Vol. 4, No. 4, 1983
[31] Robinson J. A.: Logic Programming — Past, Present and Future, New Generation Computing, Vol. 1, No. 2, 1983
[32] Schank R. C.: Dynamic Memory — A Theory of Reminding and Learning in Computer and People. Cambridge University Press, 1982
[33] Schank R. C.: The Current State of AI — One Man's Opinion, AI Magazine, Vol. 4, No. 1, pp. 3—8, Winter/Spring 1983
[34] Simon H. A.: Why Should Machines Learn? (21)
[35] Sleeman D., Brown J. S. (eds.): Intelligent Tutoring Systems. Academic Press, New York, 1982
[36] Sleeman D. H.: Inferring Student Models for Intelligent Computer-Aided Instruction. (21)
[37] Wingrad T.: What Does it Mean to Understand Language? Perspectives on Cognitive Science, Norman (ed.), Ablex Publishing Corporations, Norwood NJ, 1981
[38] Winston P. H., Binfor T. O., Katz B., Lowry M. R.: Learning Physical Descriptions from Functional Definitions — Examples and Precedents. National Conference on Artificial Intelligence, Washington (DC), 1983
[39] Weizenbaum J.: Computer Power and Human Reason. W. H. Freeman, San Francisco (CA), 1976
[40] Zagorulko N.: Empirical Prediction Algorithms, Computer Oriented Learning Processes, J. C. Simon (ed.), Noordhoff, Leyden, 1976

W dniach 18—20 listopada 1987 r. odbyło się w Katowicach międzynarodowe sympozjum nt.

„Zastosowanie mikrokomputerów w informacji naukowej, technicznej i ekonomicznej"

Miało ono na celu popularyzację zastosowania mikrokomputerów w pracy służb inte, wymianę poglądów i informacji.

Sympozjum, pod protektoratem Urzędu Postępu Naukowo-Technicznego i Wdrożeń, zorganizowano z okazji 35-lecia Instytutu Metali Nieżelaznych w Gliwicach. Oprócz Instytutu imprezę organizował Ośrodek Postępu Technicznego w Katowicach i Komisja ds. INTE Stowarzyszenia Inżynierów i Techników Przemysłu Hutniczego. W Sympozjum wzięło udział ok. 500 uczestników z Ośrodków inte różnego szczebla, ośrodków PAN, wyższych uczelni, bibliotek, ośrodków obliczeniowych oraz studenci kierunków INTE Uniwersytetu Śląskiego. Oprócz Polaków obecni byli specjaliści z Bułgarii, Czechosłowacji, NRD, Węgier i Związku Radzieckiego.

W trakcie trwania sympozjum odbyła się wystawa mikrokomputerów, sprzętu towarzyszącego i oprogramowania do celów inte, na której swoje wyroby prezentowało 10 firm. Na specjalnych pokazach można się było zapoznać ze stosowanymi programami do wyszukiwania inte na Uniwersytecie Śląskim.

Uczestnicy sympozjum mieli także okazję zwiedzenia ekspozycji odbywającej się równocześnie XXIII Międzynarodowej Wystawy Literatury Firmowej.

Ustalono, że za dwa lata odbędzie się kolejne sympozjum na ten temat.

KATARZYNA ŁUBIŃSKA
Instytut Maszyn Matematycznych
Warszawa

System dydaktyczny do nauki języka ADA/SM

Dla języka ADA/SM [1], będącego podzbiorem języka Ada [2] na komputery SM-4, w Instytucie Maszyn Matematycznych opracowano system, o nazwie ADA_USER, służący do nauki i wspomagania programowania w tym języku. Jest to system konwersacyjny, przeznaczony dla początkujących użytkowników.

INFORMACJE PODSTAWOWE

Nauka języka ADA/SM nie polega tylko na przyswojeniu sobie postaci struktur danych i instrukcji. Jest on bowiem oparty na zupełnie nowym aparacie pojęciowym, stanowiącym nierozerwalną całość. Aspekt ten wyklucza projekt systemu uczącego opartego tylko na przedstawianiu konstrukcji językowych.

Celem systemu jest podanie studentom — w serii wykładów i ćwiczeń — intuicyjnych pojęć podstawowych elementów języka ADA/SM oraz sprawdzenie ich zrozumienia. Stosuje się przy tym tradycyjny sposób przekazywania wiedzy i egzekwowania jej od użytkownika.

System ADA_USER obejmuje oprogramowanie nauczania języka ADA/SM wraz z towarzyszącymi narzędziami do wspomagania programowania oraz do wprowadzania scenariuszy. ADA_USER jest programem, w którym dane zawarte w zbiorach scenariusza oraz odpowiedzi na pytania, zadawane przez system, sterują działaniem programu. System działa w dwóch trybach:

- normalnym, gdzie użytkownikiem jest student uczący się języka ADA/SM lub programista uruchamiający program w języku ADA/SM.
- wprowadzającym scenariusze, gdzie użytkownikiem jest nauczyciel projektujący treść wykładów i ćwiczeń.

ADA_USER jest systemem konwersacyjnym, gdzie dialog jest prowadzony przez komputer. Rozpoznawanie odpowiedzi na pytania odbywa się metodami mieszanymi — słów kluczowych, słów kluczowych w kontekście oraz kwestionariuszową. Umożliwia to bardzo łatwe posługiwanie się systemem. Użytkownik pracujący w trybie normalnym nie musi znać żadnego konkretnego języka programowania ani systemu operacyjnego, ani nawet systemu ADA_USER. Zakłada się jednak u niego znajomość podstawowych pojęć informatycznych, takich jak procedura, instrukcja, typ (z przyczyn wyłącznie metodycznych).

W trybie wprowadzania scenariuszy użytkownik powinien znać zasady projektowania dialogu w systemie ADA_USER [3].

DZIAŁANIE SYSTEMU

Podstawową funkcją systemu ADA_USER jest prowadzenie cyklu wykładów i ćwiczeń, sprawdzających wiedzę użytkownika. Wykładem jest porcja wiedzy, która mieści się na jednym ekranie. Ćwiczenia mogą dotyczyć materiału zawartego w jednym lub w kilku wykładach. Mają one formę pytań zadawanych przez system. Odpowiedź użytkownika jest naturalna i zależy od rodzaju pytania. Może to być odpowiedź w języku naturalnym, fragment programu lub numer wariantu z menu, zawartego w treści ćwiczenia. W wypadku niepoprawnej odpowiedzi pojawia się tekst pomocniczy, który objaśnia prawidłową odpowiedź.

Każde ćwiczenie jest oceniane przez system jako prawidłowe lub nie. Po pewnej grupie ćwiczeń, dotyczącej jednego tematu, podawana jest ocena zbiorcza, która określa stopień opanowania przez użytkownika pewnej porcji materiału.

Przykład

ADA_USER	Wykład 15. Typy pochodne. Definicja typu pochodnego tworzy nowy typ (bazowy), którego charakterystyka jest zaczerpnięta od typu zwanego typem macierzystym. Definicja typu pochodnego definiuje dodatkowo podtyp, który jest podtypem tego typu pochodnego, np.: subtype SUBSTO is INTEGER range 1... 100; type STO is new SUBSTO; Typem pochodnym jest typ STO, typem macierzystym typ bazowy od SUBSTO (jest to nienazwany typ w zakresie od INTEGER'FIRST do INTEGER'LAST). Zbiór operacji dla typu pochodnego jest taki sam jak dla typu macierzystego. PRESS KEY ↵
UŻYTKOWNIK	↵
ADA_USER	Wybierz jedną z możliwości: 1 — Następny wykład 2 — Ćwiczenia do tego wykładu
UŻYTKOWNIK	3 ↵
ADA_USER	Podaj cyfrę 1, aby przejść do nowego wykładu lub 2, aby przejść do ćwiczeń.
UŻYTKOWNIK	2 ↵
ADA_USER	Dana jest deklaracja: type A is new INTEGER range 1..100; type B is new A range 10..20; type C is new B range 11..12; Napisz wartość atrybutów w odpowiedniej kolejności: A'FIRST, B'LAST, C'LENGTH, C'BASE'FIRST
UŻYTKOWNIK	To będzie chyba: 1 20 2 i INTEGER'FIRST .↵
ADA_USER	VERY WELL
ADA_USER	Napisz specyfikację pakietu z jedną zmienną typu POCHODNY, gdzie typ POCHODNY będzie typem pochodnym od typu BOOLEAN bez zawężenia, i zainicjuj tę zmienną na TRUE.
UŻYTKOWNIK	package A is type POCHODNY is new BOOLEAN ; C : POCHODNY := TRUE ; —— inicjalizacja end ; ↵
ADA_USER	VERY WELL
ADA_USER	Napisz deklarację typu pochodnego od nienazwanego podtypu typu CHARACTER w zakresie pełnego typu CHARACTER.
UŻYTKOWNIK	type PODCH is new CHARACTER range 'A' .. 'Z' ; a

Mgr KATARZYNA ŁUBIŃSKA ukończyła w 1978 r. Wydział Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. W latach 1978—1983 była zatrudniona w Fabryce Wyrobów Precyzyjnych im. gen. K. Świerczewskiego. Obecnie pracuje w Instytucie Maszyn Matematycznych w Zakładzie Oprogramowania, w zespole zajmującym się budową kompilatorów.

```
ADA_USER        IT IS BAD ANSWER
                Są dwie prawidłowe odpowiedzi:
                type A i new CHARACTER ;
                lub
                type A is new CHARACTER range
                                CHARACTER'FIRST..
                                CHARACTER'LAST;
                YOUR KNOWLEDGE IS NOT GOOD
```

Znak ↵, kończący odpowiedź użytkownika jest konieczny ze względu na możliwość odpowiedzi mieszczącej się w kilku wierszach (np. fragmenty programów).

Oceny systemu są jedynymi tekstami wbudowanymi na stałe w program. Podawane są one w prostych frazach języka angielskiego, ponieważ system może prowadzić dialog w różnych wersjach językowych, zależnie od zadanego mu scenariusza; oceny w języku polskim mogłyby być niezrozumiałe.

System ADA_USER zawiera również pewne elementy wspomagania programowania w języku ADA/SM. Podaje on znaczenie błędów kompilacji oraz przykłady poprawnych konstrukcji językowych. Elementy te mają znaczenie w początkowej fazie uruchamiania programów w języku ADA/SM, np.:

```
ADA_USER        Podaj numer błędu.
UŻYTKOWNIK      707 ↵
ADA_USER        Niezadeklarowana zmienna
                PRESS KEY '↵'
UŻYTKOWNIK      CTRL/Z
ADA_USER        Dziękuję, koniec dialogu.
```

Znak CTRL/Z przerywa dialog w dowolnym momencie wybranym przez użytkownika.

Wprowadzanie scenariusza

Treść wykładów, ćwiczeń, sposób odpowiedzi na ćwiczenia, teksty objaśniające prawidłowe odpowiedzi, kolejność zadawanych pytań nie są integralną częścią systemu, lecz są zadawane przez scenariusz. Mechanizm ten pozwala na łatwe wprowadzanie i modyfikowanie wykładów i ćwiczeń. Scenariusz musi być zaprojektowany przez użytkownika-nauczyciela. Wprowadzanie scenariusza odbywa się w formie dialogu prowadzonego na tej samej zasadzie co dialog edukacyjny. Gwarantuje to pełność i formalną poprawność wprowadzanych danych.

Dotychczas stworzono trzy scenariusze w dwóch wersjach językowych. Realizują one:

— wprowadzanie tekstu,
— wspomaganie programowania,
— część edukacyjną.

System ten, dzięki jego modularnej budowie, można potraktować również jako odrębny pakiet oprogramowania, służący do konwersacji. Można go dołączyć do dowolnego systemu napisanego w języku ADA/SM: systemu nauczania, systemu diagnostycznego lub dla dostępu do bazy danych. Możliwość ta jest dostępna tylko użytkownikom znającym system DOS RW oraz język ADA/SM.

Przykładowo, w samym systemie ADA_USER część edukacyjna i część wprowadzania scenariuszy mają zupełnie inne zadania, działają na innych plikach, a dialog jest prowadzony dokładnie tak samo w obu wypadkach.

ELEMENTY BUDOWY I IMPLEMENTACJA SYSTEMU

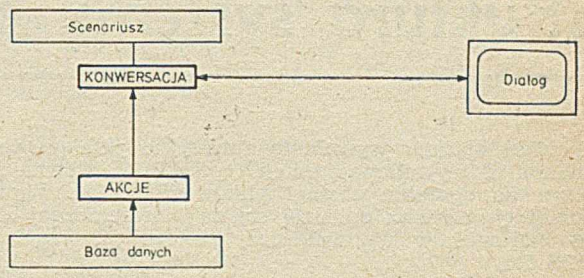
System składa się z dwóch podstawowych modułów (rys.):
- modułu prowadzącego konwersację KONWERSACJA,
- modułu wykonującego akcje dodatkowe AKCJE.

Moduł KONWERSACJA zarządza dialogiem, wyświetla treść pytań, rozpoznaje odpowiedź użytkownika, wybiera następne pytanie.

Moduł AKCJE wykonuje wszystkie działania poza zarządzaniem dialogiem, w zależności od tego, do jakiego zastosowania system ADA_USER został użyty. Na przykład, w dialogu edukacyjnym i wspomagającym programowanie moduł ten ocenia ćwiczenia, podaje dla danego numeru opis błędu lub przykład zastosowania prawidłowej konstrukcji języka ADA_SM.

Żadna funkcja modułu AKCJE nie powinna ingerować w zadania modułu KONWERSACJA.

Wywołaniem modułu AKCJE sterują dane zawarte w scenariuszu. Moduł ten korzysta tylko z danych zawartych w odpowiedzi użytkownika.



Budowa systemu ADA—USER

System ADA_USER zaimplementowano na komputerze SM-4 w języku ADA/SM za pomocą kompilatora tego języka w wersji 1.0. Dla autorów był to pierwszy większy program zaprojektowany i uruchomiony w języku ADA/SM. Dotychczasowe doświadczenia wyniesione przede wszystkim z programowania w Pascalu okazały się nieprzydatne przy programowaniu w języku ADA/SM.

ADA/SM zawiera wszystkie konstrukcje funkcjonalne Pascala i ma ponadto szereg oryginalnych struktur, które mimo pozornej prostoty, rzutują zarówno na strukturę programu, jak i na używane mechanizmy programowe. W systemie ADA_USER korzystano prawie ze wszystkich sekwencyjnych konstrukcji języka ADA/SM, z czego szczególnie wygodne okazały się pakiety i wyjątki.

Gromadzenie danych i związanych z nimi procedur w pakiety, wraz z nałożeniem określonych reguł dostępności, umożliwia utożsamienie modułów funkcjonalnych, strukturalnych z modułami programowymi. Łatwiej jest też określić powiązania między modułami. Pakiety stanowią bardzo silne narzędzie strukturalizacji zarówno projektu jak i programu, co skraca czas i pracochłonność kodowania modułów.

Na fazę uruchamiania programu bardzo wpływa możliwość korzystania z mechanizmu wyjątków, zarówno definiowanych przez programistę, jak i zdefiniowanych pierwotnie w języku (są to wyjątki oznaczające zdarzenia, takie jak: przekroczenie pamięci, przekroczenie zakresów typów, błędy wejścia-wyjścia). Możliwość uniknięcia skutków błędów w programie na poziomie podprogramu lub jednostki kompilacyjnej pozwala uruchamiać program z błędnymi modułami, traktowanymi jako poprawne, co jest ideałem przy wielomodułowych, osobno pisanych i uruchamianych systemach.

Bardzo przydatne jest też używanie typów pochodnych. Powoduje to ujawnienia, już na poziomie kompilacji, błędów polegających na zamiennym używaniu obiektów tego samego typu macierzystego, a spełniających inne funkcje w programie. Format programu wydaje się być też bardzo czytelny z powodu klarownej postaci bloków związanych z instrukcjami złożonymi. Bardzo wygodna okazała się również instrukcja awaryjnego wyjścia z pętli — exit.

Programy w języku ADA/SM pisze się dłużej niż programy w Pascalu (być może odgrywa tu rolę brak doświadczenia programistów), dłużej trwa faza usuwania błędów kompilacji, natomiast nieproporcjonalnie krótki jest czas uruchamiania.

Pełny projekt, implementacja i stworzenie obecnie istniejących scenariuszy zajęło około 3 osobolata, z czego na implementację przypada około 1,5 osoboroku.

dokończenie na str. 12

ANNA SYFERT
ANDRZEJ RAZNOWIECKI
(oprac.)
Warszawa

Struktura systemu operacyjnego PC-DOS (5)

Poniżej opisano wywołania funkcyjne pełniące te same zadania, co wywołania funkcji typu CP/M. Główna różnica polega na sposobie obsługi plików i błędów. Wywołania funkcji typu Xenix opierają się na hierarchicznej strukturze skorowidzów wprowadzonej po raz pierwszy w PC-DOS 2.0.

Wystąpienie błędu podczas wykonywania funkcji staje się dla użytkownika czytelne przez wprowadzenie jednolitych kodów błędów dla wszystkich funkcji. Wykrycie błędu jest programowo łatwiejsze od zrealizowania, ponieważ obecność błędnej funkcji jest sygnalizowana przez ustawienie bitu przeniesienia (reakcja przy użyciu rozkazu JC/JNC). W tym wypadku rejestr AX zawiera kod określający rodzaj błędu. Komunikaty błędów zestawiono w tabeli 1.

Tabela 1. Zestawienie komunikatów błędów dla funkcji typu Xenix

Kod	Znaczenie
01H	Błędne wywołanie funkcji
02H	Plik nie został znaleziony
03H	Nieistniejąca ścieżka do skorowidza
04H	Została przekroczona dozwolona liczba otwartych plików
05H	Dostęp jest zabroniony
06H	Błędny kanał logiczny
07H	Znajdujące się w pamięci tabele kontrolujące podział pamięci zostały zniszczone
08H	Wolny obszar pamięci jest niewystarczający
09H	Adres zadanego obszaru pamięci jest błędny
0AH	Błąd sprzętowy
0BH	Błędny format pliku (przy ładowaniu pliku z rozszerzeniem EXE)
0CH	Błędny kod określający rodzaj dostępu
0DH	Błędne dane (np. przy odwołaniu do urządzenia)
0EH	Nie wykorzystane
0FH	Podana specyfikacja dysku jest błędna
10H	Bieżący skorowidz nie może zostać usunięty
11H	Błędna próba przejścia na inny dysk
12H	Brak innych plików przy przeszukiwaniu

Wywołania funkcyjne typu Xenix w zasadzie nie rozróżniają odmienności działania między dostępem do pliku a dostępem do urządzenia. Również mechanizm dostępu różni się zasadniczo od mechanizmu dla funkcji omawianych dotychczas. W przypadku funkcji typu Xenix użytkownik nie ma już do czynienia z blokiem FCB, lecz jedynie z tzw. uchwytem (ang. handle), tj. numerem przyporządkowanym przez system operacyjny plikowi lub urządzeniu, np. podczas ich otwierania. Oczywiście, dostęp przez ten rodzaj kanału logicznego jest mniej uciążliwy niż budowanie i nadzorowanie bloku FCB. Dla niektórych standardowych urządzeń takie uchwyty zostały już przyporządkowane przez system operacyjny (tab. 2).

Tabela 2. Kanały we-wy zdefiniowane przez PC-DOS

Kanał	Urządzenie
00H	Standardowe wejście (CON)
01H	Standardowe wyjście (CON)
02H	Wyprowadzanie błędów
03H	we-wy przez AUX, w zasadzie sprzęg szeregowy
04H	Wyjście na drukarkę (PRN), w zasadzie przez sprzęg równoległy

Funkcja 39H — założenie skorowidza
Funkcja ta służy w zasadzie do budowy hierarchicznego systemu plików, umożliwiając założenie podskorowidza (ang. subdirectory) w skorowidzu plików, dzięki czemu powstaje struktura drzewiasta z głównym skorowidzem, jako korzeniem.
AH—39H
DS:DX — adres ścieżki do skorowidza, na końcu której powinien zostać założony następny skorowidz (podanie kompletnej ścieżki końcowej) lub adres nazwy skorowidza, który powinien zostać założony w bieżącym skorowidzu (nazwę należy zakończyć przez OOH).
Komunikaty błędów (bit przeniesienia ustawiony):
AX—03H, jeśli ścieżka lub plik nie zostały odnalezione
AX—05H, jeśli istnieje już plik o nazwie przewidzianej dla skorowidza; nazwa jest błędna (zawiera niedozwolone znaki specjalne lub określa urządzenie) lub w nadrzędnym skorowidzu nie ma już więcej miejsca na kolejny element.

Funkcja 3AH — usunięcie skorowidza
Funkcja ta usuwa pusty skorowidz plików. Tak jak w wypadku polecenia RMDIR wszystkie pliki muszą zostać uprzednio usunięte z tego skorowidza.
Parametry wejściowe:
AH—3AH
DS:DX — adres ścieżki, w której ostatnio podany skorowidz powinien zostać usunięty (podanie kompletnej ścieżki końcowej) lub adres nazwy skorowidza, który powinien zostać usunięty z bieżącego skorowidza (nazwę należy zakończyć przez 00H). Komunikaty błędów (bit przeniesienia ustawiony):
AX—03H, jeśli ścieżka lub plik nie zostały znalezione
AX—05H, jeśli skorowidz, który powinien zostać usunięty, nie istnieje lub nie jest pusty, lub podana nazwa nie jest nazwą skorowidza lub jest nazwą skorowidza pierwotnego (ang. root)
AX—10H, jeśli podjęto próbę usunięcia bieżącego skorowidza.

Funkcja 3BH — zmiana skorowidza
Funkcja ta umożliwia przejście z jednego skorowidza bieżącego do innego.
Parametry wejściowe:
AH—3BH
DS:DX—adres nowej nazwy ścieżki (zakończyć przez 00H)
Komunikat błędu (przeniesienie ustawione):
AX — 03H, jeśli nie znaleziono ścieżki

Funkcja 3CH — utworzenie pliku
Funkcja ta zakłada nowy plik. Plik już istniejący zostaje założony na nowo (ma długość 0). Plikowi można przyporządkować specyficzny atrybut. Po założeniu plik jest otwarty dla zapisu i odczytu.
Parametry wejściowe:
AH — 3CH
CX — atrybut pliku
DS:DX — adres ścieżki zawierającej nazwę pliku (nazwę należy zakończyć przez 00H)
Parametr wyjściowy (przeniesienie nie ustawione):
AX — numer kanału logicznego (ang. handle, uchwyt)

Komunikaty błędów (przeniesienie ustawione):
AX — 03H, jeśli nie odnaleziono ścieżki
AX — 04H, jeśli otwarto zbyt wiele plików (liczba równocześnie otwartych plików określona w pliku CONFIG.SYS została przekroczona)
AX — 05H, jeśli brak dostępu do skorowidza, np. skorowidz nie może przyjąć kolejnych plików lub istnieje już plik, którego atrybuty są silniejsze od nowego.

Funkcja 3DH — otwarcie kanału logicznego
Otwiera kanał dostępu do pliku lub urządzenia.
Parametry wejściowe:
AH — 3DH
AL — 00H, otwarcie kanału dla odczytu
AL — 01H, otwarcie kanału dla zapisu
AL — 02H, otwarcie kanału dla odczytu i zapisu
DS:DX — adres nazwy pliku lub urządzenia (zakończony przez 00H)
Parametr wyjściowy (przeniesienie nie ustawione):
AX — numer otwartego kanału
Komunikaty błędów (przeniesienie ustawione):
AX — 02H, jeśli plik nie został znaleziony
AX — 04H, jeśli otwarto za dużo plików
AX — 05H, jeśli przekazana nazwa nie określa żadnego dozwolonego pliku lub urządzenia, np. nastąpi próba otwarcia pliku z ochroną zapisu w celu wykonania zapisu
AX — 0CH, jeśli parametr przekazany w AL jest błędny.

Funkcja 3EH — zamknięcie kanału logicznego
Funkcja ta kończy dostęp do pliku lub urządzenia przez określony kanał (może istnieć wiele otwartych kanałów dla tego samego pliku).
Parametry wejściowe:
AH — 3EH
BX — numer kanału, który powinien zostać zamknięty
Komunikaty błędów (przeniesienie ustawione):
AX — 06H, jeśli numer kanału jest błędny, np. nie został otwarty żaden kanał o tym numerze.

Funkcja 3FH — odczyt z pliku lub urządzenia
Funkcja ta wczytuje określoną liczbę znaków przez kanał. Jeśli przed wczytaniem żądanej liczby znaków zostanie rozpoznany właściwy ogranicznik (ang. delimiter), np. CR przy wczytywaniu z CON lub AUX, lub EOF przy odczycie z pliku, to funkcja przekazuje tylko te znaki, które zostały przeczytane do tego momentu. Jeśli można odczytać więcej znaków niż jest wyspecyfikowane w CX, to funkcja czyta tylko żądaną liczbę znaków. Pozostałe znaki zostają zignorowane bez komunikatu błędu. W wypadku odczytu ze standardowego urządzenia wejściowego (CON) możliwa jest zmiana źródła wprowadzania na inne urządzenie lub plik przez przeadresowanie.
Parametry wejściowe:
AH — 3FH
BX — numer logicznego kanału wejścia
CX — liczba znaków, które mają zostać wczytane
DS:DX — adres bufora wejściowego
Parametr wyjściowy (przeniesienie nie ustawione):
AX — liczba znaków rzeczywiście wczytanych przez funkcję
Komunikaty błędów (przeniesienie ustawione):
AX — 05H, jeśli urządzenie lub plik nie zostały otwarte dla odczytu
AX — 06H, jeśli wyspecyfikowany numer kanału jest błędny lub jeszcze nie został nadany.

Funkcja 40H — zapis do pliku lub na urządzenie
Funkcja ta wyprowadza przez kanał (plik lub urządzenie) określoną liczbę znaków. W wypadku zapisu na standardowe urządzenie wyjściowe (CON) możliwa jest zmiana kierunku wyprowadzania na inne urządzenie lub plik przez przeadresowanie.
Parametry wejściowe:
AH — 40H
BX — numer logicznego kanału wyjścia
CX — liczba znaków, które powinny zostać wyprowadzone
DS:DX — adres bufora wyjściowego
Parametr wyjściowy (przeniesienie nie ustawione):
AX — liczba znaków rzeczywiście wyprowadzonych przez funkcję (jeśli liczba ta jest mniejsza niż zamierzona, co wskazuje na błąd, np. gdy zewnętrzny nośnik danych jest zapełniony). Komunikaty błędów (przeniesienie ustawione):
AX — 05H, jeśli urządzenie lub plik nie zostało otwarte dla zapisu
AX — 06H, jeśli numer wyspecyfikowanego kanału jest błędny lub kanał nie został jeszcze otwarty.

Funkcja 41H — usuwanie pliku
Funkcja ta usuwa plik, który nie ma ochrony zapisu. W celu usunięcia pliku z ochroną zapisu (pliki ukryte mają również ochronę zapisu) należy ją uprzednio usunąć za pomocą funkcji 43H przez wyzerowanie atrybutu.
Parametry wejściowe:
AH — 41H
DS:DX — adres nazwy pliku zakończonej przez 00H (możliwe jest podanie ścieżki do pliku, nazwy zmienne nie są dozwolone)
Komunikaty błędów (przeniesienie ustawione):
AX — 02H, jeśli nie odnaleziono pliku
AX — 03H, jeśli nie odnaleziono ścieżki
AX — 05H, jeśli podany plik ma ochronę zapisu lub próbowano usunąć skorowidz.

Funkcja 42H — przesunięcie wskaźnika zapisu-odczytu
Funkcja powoduje przesunięcie wskaźnika zapisu-odczytu wewnątrz pliku. Wyrównanie zostaje dodane do początku lub końca pliku. Funkcja udostępnia do programu wywołującego nowy stan wskaźnika. Stan oraz wyrównanie są podawane w bajtach.
Parametry wejściowe:
AH — 42H
AL — 00H, wyrównanie zostaje dodane do początku pliku (bezwzględne pozycjonowanie wewnątrz pliku)
AL — 01H, wyrównanie zostaje dodane do bieżącego stanu wskaźnika zapisu-odczytu (względne pozycjonowanie w kierunku końca pliku)
AL — 02H, wyrównanie zostaje dodane do końca pliku (rozszerzenie pliku)
BX — numer kanału logicznego
CX:DX — wyrównanie w bajtach (CX zawiera słowo bardziej znaczące)
Parametr wyjściowy (przeniesienie nie ustawione):
DX:AX — nowy stan wskaźnika zapisu-odczytu (w bajtach; DX zawiera słowo bardziej znaczące)
Komunikaty błędów (przeniesienie ustawione):
AX — 01H, jeśli AL zawiera wartość różną od wyspecyfikowanych powyżej
AX — 06H, jeśli wyspecyfikowany kanał nie został otwarty lub numer kanału jest błędny.

Funkcja 43H — odczyt lub zmiana atrybutu pliku
Funkcja umożliwia odczyt wszystkich atrybutów oraz zmianę następujących atrybutów pliku: ochrona przed zapisem, plik ukryty, plik systemowy, plik zapisany (ang. archive bit).
Parametry wejściowe:
AH — 43H
AL — 00H, odczyt atrybutu
AL — 01H, ustawienie atrybutu
CX — ustawiony atrybut (używane tylko wtedy, gdy AL = =01H)
DS:DX — adres nazwy pliku zakończonej przez 00H
Parametr wyjściowy (przeniesienie nie ustawione):
CX — aktualnie ustawione atrybuty (jeśli podczas wywołania funkcji AL=00H)
Komunikaty błędów (przeniesienie ustawione):
AX — 01H, jeśli funkcja wyspecyfikowana w AL nie istnieje
AX — 03H, jeśli plik nie istnieje lub wyspecyfikowana ścieżka nie jest prawidłowa
AX — 05H, jeśli atrybut wyspecyfikowany w CX nie może zostać ustawiony (w tym wypadku nie zostanie ustawiony żaden z podanych atrybutów).

Funkcja 44H i jej podfunkcje
Ponieważ funkcja 44H jest bardzo złożona, w celu osiągnięcia lepszej czytelności została podzielona na podfunkcje. Podfunkcje 00H, 06H, 07H mogą zostać użyte w odniesieniu do plików, zaś wszystkie podfunkcje mogą zostać użyte w odniesieniu do urządzeń. Komunikaty błędów są wspólne dla wszystkich podfunkcji (przeniesienie ustawione):
AX — 01H, jeśli funkcja wyspecyfikowana w AL nie istnieje
AX — 05H, jeśli próba dostępu została odrzucona (np. wskutek ochrony zapisu lub błędnej specyfikacji dysku)
AX — 06H, jeśli wyspecyfikowany kanał jest błędny lub nie został jeszcze otwarty
AX — ODH, jeśli podano nieważne dane.

Podfunkcja 00H — odczyt informacji o kanale logicznym
Parametry wejściowe:
AH — 44H
AL — 00H
BX — numer kanału logicznego

Parametr wyjściowy (przeniesienie nie ustawione):
DX — zakodowane informacje o kanale.

Podfunkcja 01H — ustawienie informacji o kanale
Parametry wejściowe:
AH — 44H
AL — 01H
BX — numer kanału logicznego
DX — ustawiana informacja o kanale.

Podfunkcja 02H — odbiór znaków sterujących od urządzenia znakowego
Parametry wejściowe:
AH — 44H
AL — 02H
BX — numer kanału logicznego
CX — liczba bajtów, które powinny zostać odebrane
DS:DX — adres bufora wejściowego
Parametr wyjściowy (przeniesienie nie ustawione):
AX — liczba rzeczywiście odebranych znaków.

Podfunkcja 03H — wyprowadzanie znaków sterujących na urządzenie znakowe
Parametry wejściowe:
AH — 44H
AL — 03H
BX — numer kanału logicznego
CX — liczba bajtów, które powinny zostać wysłane
DS:DX — adres bufora wyjściowego
Parametr wyjściowy (przeniesienie nie ustawione):
AX — liczba rzeczywiście wysłanych znaków.

Podfunkcja 04H — odbiór znaków sterujących z urządzenia blokowego
Parametry wejściowe:
AH — 44H
AL — 04H
BL — numer napędu dyskowego
CX — liczba odbieranych bajtów
DS:DX — adres bufora wejściowego
Parametr wyjściowy (przeniesienie nie ustawione):
AX — liczba rzeczywiście odebranych znaków.

Podfunkcja 05H — wyprowadzanie znaków sterujących na urządzenie blokowe
Parametry wejściowe:
AH — 44H
AL — 05H
BL — numer napędu dyskowego
CX — liczba bajtów, które mogą zostać wysłane
DS:DX — adres bufora wyjściowego
Parametr wyjściowy (przeniesienie nie ustawione):
AX — liczba rzeczywiście wysłanych znaków.

Podfunkcja 06H — kontola gotowości do wprowadzania
Parametry wejściowe:
AH — 44H
AL — 06H
BX — numer kanału logicznego
Parametry wyjściowe (przeniesienie nie ustawione):
AL — 00H, urządzenie nie jest gotowe do wprowadzania pliku; odczyt nie jest możliwy, ponieważ został osiągnięty koniec pliku (EOF)
AL — FFH, urządzenie gotowe do wprowadzania, plik może zostać odczytany.

Podfunkcja 07H — kontrola gotowości do wyprowadzania
Parametry wejściowe:
AH — 44H
AL — 07H
BX — numer kanału logicznego
Parametry wyjściowe (przeniesienie nie ustawione):
AL — 00H, urządzenie nie jest gotowe dla wyprowadzania, plik nie może zostać zapisany
AL — FFH, urządzenie jest gotowe do wyprowadzania, plik może zostać zapisany.

Funkcja 45H — duplikowanie kanału logicznego
Funkcja tworzy drugi kanał logiczny dostępu do pliku lub urządzenia. W przypadku pliku wskaźnik zapisu-odczytu przyporządkowany każdemu z obu kanałów wskazuje zawsze również po odwołaniu się do pliku na tę samą pozycję wewnątrz pliku.
Parametry wejściowe:
AH — 45H

BX — numer kanału logicznego, dla którego powinien zostać utworzony drugi kanał
Parametr wyjściowy przeniesienie nie ustawione:
AX — numer drugiego równorzędnego kanału
Komunikaty błędów przeniesienie ustawione:
AX — 04H, jeśli za dużo otwartych plików
AX — 06H, jeśli wyspecyfikowany kanał nie jest otwarty lub jest nieprawidłowy.

Funkcja 46H — wymuszone duplikowanie kanału
Dla już zdefiniowanego kanału zostaje założony drugi kanał. Jeśli drugi kanał jest otwarty, to zostanie zamknięty, a następnie ponownie otwarty dla równorzędnego dostępu do urządzenia przypisanego do pierwszego kanału lub odpowiedniego pliku.
Funkcja ta jest stosowana zawsze wtedy, gdy informacje powinny być wymieniane zawsze przez ten sam kanał logiczny.
Parametry wejściowe:
AH — 46H
BX — numer pierwotnego kanału logicznego
CX — numer drugiego zakładanego kanału
Komunikaty błędów (przeniesienie ustawione):
AX — 04H, jeśli za dużo jest otwartych plików
AX — 06H, jeśli wyspecyfikowany kanał nie jest otwarty (podany w BX) lub jest nieprawidłowy (dotyczy obu kanałów).

Funkcja 47H — odczyt wybranego skorowidza
Funkcja dostarcza ciąg znaków zakończony przez 00H, specyfikujący aktualnie wybrany skorowidz (rozpoczynając od skorowidza pierwotnego); ciąg znaków nie zawiera informacji o wybranym dysku i nie zaczyna się od ukośnika
Parametry wejściowe:
AH — 47H
DS:SI — adres obszaru pamięci o wielkości co najmniej 64 bajtów dla przyjęcia ciągu znaków
DL — dysk, dla którego zostanie dokonany odczyt
Komunikat błędu (przeniesienie ustawione):
AX — 0FH, jeśli specyfikacja dysku w DL jest nieprawidłowa.

System do nauki języka ADA/SM

dokończenie ze str. 9

Prace nad tworzeniem systemów uczących Ady rozpoczęły się zaraz po ogłoszeniu raportu tego języka. Powstał, między innymi, w roku 1985 system Alsys [4] do prowadzenia cyklu wykładów i ćwiczeń z Ady. Był on napisany w języku tradycyjnym, a jego pracochłonność wynosiła 50 osobolat. Oczywiście, ADA_USER jest nieporównywalnie mniejszy od znanych systemów dydaktycznych dla Ady, ale jest to wynikiem konieczności pogodzenia wielkości dostępnej na SM-4 pamięci z wymaganiami dotyczącymi czasu reakcji systemu.

Dzięki prostej budowie ADA_USER może być przydatny w różnych zastosowaniach. Użycie go do nauki innych języków programowania lub innych dziedzin wiedzy wymaga tylko stworzenia i wprowadzenia nowego scenariusza, a użycie go do systemów niedydaktycznych wymaga zmiany modułu AKCJE i stworzenia nowego scenariusza.

LITERATURA

[1] ADA/SM. Opis języka. Dokumentacja firmowa FMiK „Era", Warszawa, 1985
[2] Ada Programing Language. ANSI/MIL-STD-1815A, 1983
[3] ADA_USER. Podręcznik użytkownika. Dokumentacja firmowa FMiK „Era", Warszawa, 1986
[4] Beretz A., Brosgol B.: Developing and Automated Ada Training Product. Proc. Conf. Ada in use. Cambridge University Press, 1985.

ROBERT JAWORSKI
KRZYSZTOF PŁOWIEC
Instytut Maszyn Matematycznych
Warszawa

Monitor ekranowy Mazovii 1016

We współczesnych komputerach, zarówno tych dużych, jak i najmniejszych, jednym z najważniejszych sposobów zobrazowania informacji jest przedstawienie jej na ekranie. Dlatego tak duże znaczenie przywiązuje się do rozwoju monitorów ekranowych i ich sterowników.

Ponieważ w porozumieniu z innymi modułami komputera są to bloki stosunkowo kosztowne, przy ustalaniu wymaganych właściwości monitora ogromne znaczenie ma sprecyzowanie podstawowych jego parametrów, które muszą zostać zachowane oraz tych, które mogą być osiągnięte stosunkowo niewielkim kosztem. Niebagatelne znaczenie ma również istniejące oprogramowanie. Z punktu widzenia wielu producentów nieopłacalne jest wytwarzanie sterowników graficznych, które nie są dostosowane do tego oprogramowania.

Na rynku komputerów osobistych, kompatybilnych z mikrokomputerami IBM PC/XT i IBM PC/AT, można wyróżnić kilka klas sterowników graficznych:

1. **Sterowniki o wysokiej rozdzielczości** przeznaczone do prac projektowych w systemach CAD, nap. IBM PGA.
2. **Sterowniki o dobrych parametrach użytkowych** przeznaczone do pracy z tekstami i drobniejszych prac projektowych, jak również do graficznego zobrazowania danych liczbowych, np. IBM EGA.
3. **Sterowniki standardowe** przeznaczone głównie do pracy z tekstem oraz do graficznego zobrazowania danych liczbowych, np. IBM MDA, IBM CGA. Prace projektowe przy użyciu tych sterowników są również możliwe, jednak bardzo utrudnione, głównie ze względu na zbyt małą rozdzielczość ekranu.

W najpowszechniejszym użyciu znajdują się obecnie sterowniki z grupy 2 i 3. Ogromna większość popularnego oprogramowania może współpracować ze sterownikami grupy 2. Grupa 3 jest natomiast traktowana jako standard i praktycznie każde oprogramowanie może współpracować ze sterownikiem z tej grupy.

PODSTAWOWE WŁASCIWOŚCI

Sterownik monitora do Mazovii 1016 miał być z założenia sterownikiem, z którym będzie pracować możliwie jak największa liczba programów, a jednocześnie niezbyt złożonym. Dlatego też jego parametry zostały tak dobrane, aby można go zaliczyć do grupy 3. Jest on kompatybilny z sterownikami IBM MDA, IBM CGA, HERCULES MGA i COMPAQ, dla których istnieje najwięcej oprogramowania, a które jednocześnie nie stawiają zbyt wysokich wymagań współpracującym monitorom.

Takie podejście było możliwe dzięki temu, że projekt powstawał w czasie, kiedy rynek oprogramowania dla komputerów PC był już dostatecznie rozwinięty. Zachowanie kompatybilności z wymienionymi sterownikami oraz zastosowanie tzw. monitora dwustandardowego (np. typu Mazovia MM12P) powoduje, że cechy funkcjonalne, które dotychczas były osiągalne (na przykład za pomocą pakietów IBM CGA, HERCULES MGA) przy użyciu dwóch monitorów, można otrzymać na jednym ekranie bez konieczności instalowania dodatkowego pakietu i dokonywania jakichkolwiek operacji mechanicznych. W tabeli 1 przedstawiono porównanie różnych rodzajów pracy dla poszczególnych sterowników.

Podstawowym rodzajem pracy sterownika jest praca w konfiguracji jednomonitorowej z monitorem dwustandardowym. Przy pracy z pakietem CGA, monitor ten przedstawia kolory jako odcienie szarości. Nazwa „monitor dwustandardowy" pochodzi stąd, że może on współpracować zarówno ze sterownikami IBM CGA, jak i IBM MDA. Sterowniki te wysyłają sygnały synchronizacji o dwóch różnych częstotliwościach. W pierwotnym rozwiązaniu firmy IBM do każdego sterownika dołączano oddzielny monitor.

Jeżeli monitor dwustandardowy jest dołączony do sterownika Mazovii, to umożliwia on wykorzystanie wszystkich rodzajów pracy. Do sterownika mogą być dołączone również oryginalne monitory IBM lub inne kompatybilne z nimi. Pakiet ma przełączniki konfiguracyjne, którymi można podawać rodzaj dołączonego monitora.
Wówczas sterownik pracuje albo jako IBM CGA, albo jako IBM MDA. Takie rozwiązanie uniemożliwia przypadkowe przełączenie sterownika w inny rodzaj pracy niż wymagany dla dołączonego monitora, co zabezpiecza przed utratą obrazu na ekranie oraz przed uszkodzeniem monitora (p. tabela 2).

Tabela 1. Porównanie różnych rodzajów pracy dla poszczególnych sterowników

Rodzaj pracy ekranu Mazovii	IBM MDA	IBM CGA	COMPAG	HERCULES
Znakowy 40×25, znak 8×8 punktów		Tak	Tak	
Znakowy 80×25, znak 8×8 punktów		Tak	Tak	
Znakowy 80×25, znak 9×14 punktów (16 kolorów znaku na jednym z 8 lub 16 kolorów tła)			Tak	
Znakowy 80×25 (emulacja MDA)	Tak			Tak
Grafika 320×200 punktów, 4 kolory		Tak	Tak	
Grafika 640×200 punktów, jednobarwna		Tak	Tak	
Grafika 720×348 punktów, czarno-biała				Tak

Tabela 2. Przełączniki konfiguracyjne ustalające konfigurację, w jakiej pracuje sterownik

Klucz	Znaczenie
1	ON — zainstalowano dwa sterowniki OFF — zainstalowano jeden sterownik Jeżeli w systemie są zainstalowane dwa sterowniki, przy czym przynajmniej jeden jest sterownikiem Mazovii, to klucz należy ustawić w pozycji ON
5	ON — tryb kolorowy graficzny OFF — tryb monochromatyczny Jeżeli sterownik przy inicjalizacji systemu ma być zainiejowany na tryb monochromatyczny, to klucz należy ustawić w pozycji OFF
7	ON — monitor kolorowy OFF — monitor dwustandardowy Ustawienie klucza w pozycji ON informuje BIOS o dołączeniu monitora kolorowego
8	ON — monitor dwustandardowy Mazovii OFF — monitor IBM monochromatyczny Jeżeli sterownik współpracuje z monitorem monochromatycznym firmy IBM, to klucz należy ustawić w pozycji OFF

Sygnał wizji może być przekazywany liniami koloru i intensywności (R, G, B, I) lub liniami wizji czarno-białej i intensywności (B&W, I). W pracy z monitorem dwustandardowym wizja jest przesyłana zawsze liniami R, G, B, I. Dzięki temu można uzyskać dodatkowy tekstowy rodzaj pracy 80×25 znaków (mozaika znaku 9×14 punktów), z odcieniami szarości. Ten rodzaj pracy jest „widziany" przez oprogramowanie jako tryb znakowy 80×25 pakietu IBM CGA. Daje to możliwość uzyskania na ekranie liter o czytelności takiej, jak dla sterownika IBM MDA, bez utraty możliwości wyświetlania znaków w różnych odcieniach szarości. A zatem, jeżeli jest emulowana praca pakietu IBM CGA z dołączonym monitorem dwustandardowym, to sterownik Mazovii może być traktowany jako pakiet sterownika komputera COMPAQ-DESKPRO.

Z punktu widzenia użytkownika, ekran Mazovii stanowi jedno spójne urządzenie. Aby korzystać z jego wszystkich możliwości nie trzeba wiedzieć, kiedy jaki pakiet ma być emulowany. Jeżeli przełączniki konfiguracyjne są właściwie ustawione, to przełączanie z emulacji jednego pakietu na drugi odbywa się w sposób niewidoczny dla użytkownika.

Jeżeli chcemy, na przykład pracować z programem LOTUS wykorzystującym grafikę monochromatyczną 720×348 punkty, to wystarczy, że przejdziemy do pracy w trybie emulacji MDA/MGA poleceniem systemowym MODE:

A>MODE MONO

i uruchomimy program LOTUS skonfigurowany dla tego rodzaju grafiki.
Podobnie jest przy przechodzeniu do emulacji CGA/COMPAQ. Piszemy wówczas:

A>MODE CO80

lub

A>MODE CO40

i sterownik będzie naśladować pakiet CGA lub COMPAQ.

Jest również możliwa praca komputera Mazovia w konfiguracji dwumonitorowej. Do tego celu są konieczne dwa sterowniki monitora Mazovii i dwa monitory. Jeden sterownik pracuje na stałe w trybie emulacji CGA/COMPAQ, a drugi w trybie emulacji MDA/MGA. Przechodzenie z pracy na jednym ekranie do pracy na drugim odbywa się identycznie jak przy zmianie rodzaju emulacji w wypadku poprzednim.

Konfiguracja taka ma zastosowanie w dwóch wypadkch. Pierwszym z nich jest praca z pewnymi programami graficznymi, np. MS-CHART lub AUTOCAD, które umożliwiają jednoczesne wykorzystanie obu ekranów. Jeden z nich na stałe pracuje jako graficzny, drugi jako tekstowy. Wszelkie komunikaty ukazują się wtedy na ekranie tekstowym. Dzięki temu większa część powierzchni ekranu graficznego może być użyta do przedstawiania rysunków.

Drugi — kiedy konfiguracja dwumonitorowa jest bardzo użyteczna, występuje wtedy, gdy dysponujemy jednocześnie monitorem monochromatycznym lub dwustandardowym i monitorem kolorowym. W tej sytuacji zastosowanie dwóch sterowników umożliwia pracę z tekstem na ekranie monochromatycznym o wysokiej rozdzielczości i nierezygnowanie z możliwości graficznych monitora kolorowego. Monitor taki nie jest najlepszy przy pracy tekstowej, ze względu na niezbyt dużą rozdzielczość, natomiast świetnie nadaje się do prezentowania rysunków, wielobarwnych lub ideogramów.

ZESTAWY ZNAKÓW

Ze względu na specyfikę rynku, sterownik został wyposażony w dwa generatory znaków. Pierwszy generator zawiera polski zestaw znaków, tzn. znaki o kodach 0—127 są identyczne jak dla IBM PC, a znaki typowo polskie wprowadzono na niektórych pozycjach od kodu 128 do 175, w obszarze przeznaczonym na znaki narodowe.

Drugi zestaw znaków odpowiada cyrylicy w odmianie rosyjskiej. Znaki o kodach 0—127 są takie, jak w generatorze poprzednim, a znaki rosyjskie są umieszczone w sposób ciągły na pozycjach od 128 do 175. W połączeniu z klawiaturą Mazovii przeznaczoną do pisania tekstów rosyjskich uzyskuje się komputer pracujący bez kłopotu z drugim zestawem znaków. Dzięki takiemu rozwiązaniu przy pracy z dowolnym alfabetem można korzystać z podstawowego zestawu znaków łacińskich. Oznacza to możliwość przetwarzania tekstów angielskich, angielsko-rosyjskich, angielsko-polskich, jednak bez możliwości pracy z tekstem polsko-rosyjskim.

Generatory znaków są montowane na podstawkach. Umożliwia to wstawianie dowolnego innego zestawu znaków i dostosowywanie zbioru znaków na ekranie do różnych alfabetów narodowych.

Do wszystkich wyżej wymienionych cech pakietu sterującego monitorem użytkownik ma bardzo prosty dostęp. Zestawy znaków są przełączane z klawiatury. Jednoczesne wciśnięcie klawiszy CTRL-ALT-C powoduje wybranie rosyjskiego zestawu znaków. Powrót do podstawowego zestawu znaków następuje przez wciśnięcie klawiszy CTRL-ALT-L.

Przełączenia zestawu znaków mają również wpływ na zestaw znaków w graficznych rodzajach pracy sterownika. Program BIOS komputera Mazovia zapewnia pełne zestawy znaków (po 256 znaków) dla alfabetu polskiego i rosyjskiego przy pracy sterownika w trybach graficznych. Jeżeli użytkownik decyduje się na stosowanie innego generatora znaków (przez wymianę pamięci stałej generatora), to powinien postarać się także o posiadanie identycznego generatora w programie BIOS, wstawionego w miejsce generatora cyrylicy.

Podobnie jak przy zmianie alfabetów postępuje się, aby uzyskać dodatkowy rodzaj pracy tekstowej sterownika. Jest to praca w trybie znakowym 80×25 z mozaiką znaku 9×14 punktów i przedstawieniem kolorów jako odcieni szarości. Jak wspomniano, jest to mutacja trybu znakowego 80×25 z mozaiką znaku 8×8 punktów i kolorową reprezentacją znaków, który był wykorzystywany przy pracy z monitorem kolorowym. Aby uzyskać ten tryb korzysta się z polecenia systemowego MODE:

A>MODE CO80

a następnie zmienia się mozaikę znaku z 8×8 na 9×14 punktów, wciskając jednocześnie klawisze CTR-CLT->. Powrót do mozaiki znaku 8×8 punktów następuje po jednoczesnym wciśnięciu klawiszy CTR-ALT-<. Aby użytkownik nie musiał za każdym razem po przejściu do trybu znakowego ustawiać odpowiedniej mozaiki znaku, rodzaj wybranej mozaiki jest pamiętany i po każdorazowym przejściu do tego rodzaju pracy ustawia się ta mozaika, z którą pracowano poprzednio.

DODATKOWE WŁAŚCIWOŚCI

Oprócz opisanych cech, sterownik ma jeszcze jedną ciekawą właściwość polegającą na możliwości zmiany atrybutów znaków ze standardu IBM MDA na standard IBM CGA (COMPAQ).

W celu wyjaśnienia tej właściwości należy powiedzieć kilka słów o sposobie przechowywania znaków w pamięci ekranu. Każdy znak jest zapisany w postaci dwóch bajtów. Pierwszy z nich, o adresie parzystym, jest kodem ASCII znaku i określa jeden z 256 znaków dostępnych z aktualnie aktywnego generatora. Bezpośrednio po nim następuje bajt o adresie nieparzystym, definiujący cechy tego znaku — zgodnie ze schematem przedstawionym w tabeli 3 (tzw. bajt atrybutu). W rozwiązaniu przyjętym przez firmę IBM, i zachowanym przez emulujący IBM MDA sterownik obra-

Tabela 3. Rozmieszczenie bitów w bajcie atrybutu

Bity	Nazwa	Znaczenie
0—2	RGB	Kolory znaku: czerwony, zielony, niebieski
3	I	Intensywność znaku
4—6	RGB	Kolory tła: czerwony, zielony, niebieski
7	I/B	Intensywność tła lub miganie znaku

zu Mazovii, bajt atrybutu jest interpretowany według tabeli 4.

Jeżeli do pamięci jest wpisany bajt atrybutu różniący się od podanych powyżej, to na etapie jego interpretacji przez sprzęt jest on przekształcany na jedną z postaci standardowych.

Tabela 4. Interpretacja bajtu atrybutu (x odpowiada stanowi bez znaczenia)

B/I	R	G	B	I	R	G	B	Znaczenie
x	0	0	0	x	0	0	0	Wygaszona wizja
x	0	0	0	x	0	0	1	Podkreślenie znaku
x	0	0	0	x	1	1	1	Jasny znak na ciemnym tle
x	1	1	1	x	0	0	0	Ciemny znak na jasnym tle

Sygnalizowana właściwość sterownika obrazu polega na łatwości zmiany interpretacji bajtu atrybutu. Jeżeli pracujemy w trybie emulacji sterownika firmy COMPAQ, to każdy znak jest prezentowany ze wszystkimi wynikającymi z atrybutów stopniami szarości — tła i znaku. Jeżeli chcemy ten sam tekst przetwarzać tak, jak przy emulacji trybu IBM MDA, to wystarczy wcisnąć klawisze:

CTRL-ALT-TAB

aby uzyskać tłumaczenie atrybutów. Tak więc znaki oglądane w kolorze niebieskim tzn. odpowiadającej mu szarości, zostaną teraz podkreślone. Powrót do poprzedniego trybu jest równie prosty i polega na ponownym wciśnięciu klawiszy **CTRL-ALT-TAB.**

KOMPATYBILNOŚĆ Z IBM

Do najważniejszych cech komputerów klasy PC, z punktu widzenia użytkownika, należy ich kompatybilność z oryginałem IBM PC/XT. Podstawowym kryterium funkcjonalnym dla sterownika monitora jest jego kompatybilność z pakietami IBM CGA i IBM MDA.

Sterownik monitora Mazovii jest kompatybilny ze wspomnianymi pakietami pod każdym względem, nawet z pozoru mało istotnym. Identyczne są wyjściowe sygnały sterujące monitorem, identyczne są wszystkie porty wejścia-wyjścia, identyczna jest szybkość współpracy głównego procesora z pamięcią ekranu oraz identyczny jest sposób interpretacji na ekranie zawartości pamięci. Pakiet ten może współpracować z dowolnym innym sterownikiem monitora w konfiguracji dwumonitorowej, np. IBM CGA, IBM MDA, IBM EGA itp.

Jedyną niekompatybilnością sterownika jest niezgodność jego wymiarów (jest około 4 cm wyższy od pakietów firmy IBM) oraz zastosowanie złącza pośredniego zamiast złącza krawędziowego. Z tego względu sterownik nie może być instalowany w komputerach firmy IBM. Może jednak z nim współpracować po zainstalowaniu na łączówce przejściowej.

Zgodność logiczna złącza sterownika z magistralą systemową implikuje możliwość instalowania w komputerze Mazovia 1016, oprócz sterownika monitora, dowolnej innej karty graficznej. Kartę tą instaluje się na łączówce pośredniczącej.

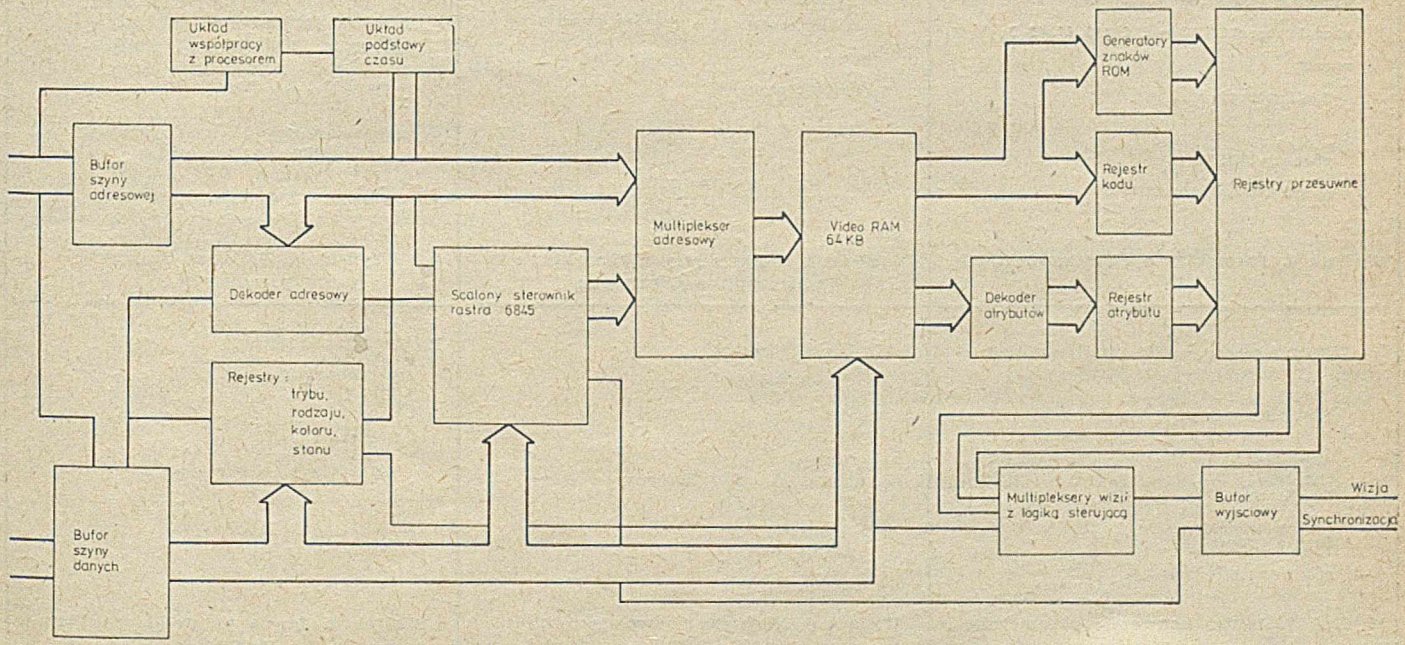
Pakiet HERCULES MGA, który również może być emulowany przez sterownik Mazovii, ma pewną wadę, a mianowicie: powoduje pewne kłopoty przy współpracy z pakietem IBM CGA. Wynika to stąd, że przy uaktywnieniu na nim dodatkowej strony pamięci graficznej dochodzi do konfliktu adresów tej właśnie strony i bufora pamięci pakietu IBM CGA.

Wada ta została wyeliminowana w sterowniku monitora Mazovii. Jeżeli w komputerze zainstalowane są dwa pakiety sterowników monitorów (Mazovii emulujący MDA/MGA i dowolny inny pracujący jako IBM CGA), to jest niemożliwe uaktywnienie drugiej strony pamięci graficznej pakietu emulującego MDA/MGA. Strona ta może być włączona tylko wtedy, gdy jest zainstalowany jeden sterownik monitora.

Tabela 5. Monitor Mazovii w konfiguracji dwumonitorowej

Rodzaj emulacji sterownika Mazovii	Typ drugiego sterownika
IBM MDA, HERCULES MGA	IBM CGA Mazovia jako IBM CGA
IBM CGA, COMPAQ	IBM MDA HERCULES MGA 2 (bez otwierania drugiej strony)

W tabeli 5 przedstawiono możliwości pracy sterownika monitora Mazowii w konfiguracji dwumonitorowej. Schemat blokowy sterownika zamieszczono na rysunku.



Schemat blokowy sterownika monitora Mazovii 1016

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Język C

Skojarzenia parametrów z argumentami

Pewną niedogodnością języka C jest ustalenie, że skojarzenia parametrów z argumentami mogą dokonywać się jedynie przez wartość. Oznacza to, że argumenty funkcji są zawsze traktowane jak wyrażenia właściwe (a nie np. nazwy symboliczne obiektów) i z tego powodu w celu uzyskania efektu charakterystycznego dla skojarzenia przez wskazanie (niezbyt ściśle nazywanego niekiedy skojarzeniem przez nazwę), należy argumentowi nadać postać wyrażenia wskazującego.

Na wydruku 1 przedstawiono program wyjaśniający istotę skojarzenia przez wartość. W funkcji fun występują dwie zmienne lokalne, które są jej parametrami: Ref i Val. W prologu funkcji, zmiennej Ref zostaje przypisane wskazanie zmiennej Alfa, a zmiennej Val zostaje przypisana taka sama dana jak zmiennej Beta. Obie przypisane daną są reprezentowane przez argumenty funkcji fun. Wykonanie operacji *Ref='j' powoduje przypisanie zmiennej Alfa danej 'j'. Wykonanie operacji Val='*' powoduje przypisanie zmiennej Val danej '*'. Ostatnia operacja nie dotyczy więc zmiennej Beta. Wykonanie programu powoduje zatem wyprowadzenie napisu jb.

Podobnie jak w wielu innych językach programowania, w języku C wymaga się, aby każdy argument był zgodny pod względem typu z odpowiadającym mu parametrem. Zgodność ta jest jednak badana dopiero po niejawnej konwersji argumentów. Niejawna konwersja polega na tym, że argumenty typu (char) i (short int) są niejawnie zastępowane argumentami typu (int), a argumenty typu (float) są niejawnie zastępowane argumentami typu (double). Oznacza to, że program przedstawiony na wydruku 2 jest równoważny programowi przedstawionemu na wydruku 3.

dokończenie na str. 19

```
#include <stdio.h>

main()
{
  char Alfa = 'a',
       Beta = 'b';
  fun(&Alfa,Beta);
  printf("%c%c", Alfa,Beta);
}

fun(Ref,Val)
 char *Ref,Val;
{
  *Ref = 'j';
  Val  = '*';
}
```

Wydruk 1. Istota skojarzenia przez wartość

```
#include <stdio.h>

main()
{
  char Chr = 'j';

  fun((int)Chr,(double)44.0);
}

fun(Letter,Number)
  int Letter;
  double Number;
{
  printf("%c-%.0f", Letter,
                    Number);
}
```

Wydruk 2. Niejawne konwersje argumentów

```
#include <stdio.h>

main()
{
  char Chr = 'j';

  fun(Chr,44.0);
}

fun(Letter,Number)
  int Letter;
  double Number;
{
  printf("%c-%.0f", Letter,
                    Number);
}
```

Wydruk 3. Jawne konwersje argumentów

```
#include <stdio.h>

char Letters[2][3] = { {'j'},{'b'} };

main()
{
  fun(Letters);
}

fun(Arr)
  char Arr[2][3];
{
  printf("%c%c", Arr[0][0],
                 (Arr + 1)[0][0]);
}
```

Wydruk 4. Parametry o charakterze tablic

```
#include <stdio.h>

char Letters[2][3] = { {'j'},{'b'} };

main()
{
  fun(Letters);
}

fun(Ptr)
  char (*Ptr)[3];
{
  printf("%c%c", Ptr[0][0],
                 (Ptr + 1)[0][0]);
}
```

Wydruk 5. Parametry o charakterze zmiennych wskazujących

```
#include <stdio.h>

main()
{
  fun(259,1.0 / 3);
}

fun(Char,Real)
  char Char;
  float Real;
{
  printf("%.50f", Char * Real);
}
```

Wydruk 6. Niejawne konwersje parametrów

```
#include <stdio.h>

main()
{
  fun(259,1.0 / 3);
}

fun(Char,Real)
  int Char;
  double Real;
{
  Char = (char)Char;
  Real = (float)Real;

  printf("%.50f", Char * Real);
}
```

Wydruk 7. Jawne konwersje parametrów

```
#include <stdio.h>

int Number[1][1] = { 13 };

main()
{
  int *Fun();

  printf("%d", *Fun(Number));
}

int *
Fun(Par)
  int (*Par)[2];
{
  return Par;
}
```

Wydruk 8. Niejawna konwersja rezultatu i funkcji

```
#include <stdio.h>

int Number[1][1] = { 13 };

main()
{
  int *Fun();

  printf("%d", *Fun(Number));
}

int *
Fun(Par)
  int (*Par)[2];
{
  return (int *)Par;
}
```

Wydruk 9. Jawna konwersja rezultatu funkcji

ZBIGNIEW FRYŹLEWICZ
Centrum Obliczeniowe
Politechnika Wrocławska

Mikrokomputerowy analizator-tester protokołów

Urządzenie, którego cechy funkcjonalne i realizacyjne opisano w tym opracowaniu, powstało w ramach tematu „Sieć Komputerowa Maszyn Jednolitego Systemu — 2, wersja 1", zrealizowanego przez Centrum Obliczeniowe Politechniki Wrocławskiej na zlecenie IKSAiP — Wrocław.

Analiza potrzeb całego tematu i doświadczenia wynikłe z realizacji sieci MSK wskazywały, że zastosowanie specjalistycznych urządzeń monitorująco-testujących istotnie przyspiesza lokalizację błędów transmisji wynikłych z nieprawidłowej implementacji protokołów. Urządzenia podobnego typu produkuje wiele firm elektronicznych, np. [1, 2, 5], lecz na przełomie lat 1983—1984 były one w Polsce niedostępne.

Funkcjonalne cechy urządzenia opisanego poniżej są wypadkową literaturowej analizy działania istniejących urządzeń, wymagań potencjalnych użytkowników i krótkiego czasu przeznaczonego na realizację całości projektu.

CECHY FUNKCJONALNE

Aparatura pomiarowo-diagnostyczna (nazwa firmowa SPS) [4] może być stosowana do:
- monitorowania transmisji przez styk S2,
- testowania komponentów sieci połączonych z SPS stykiem S2.

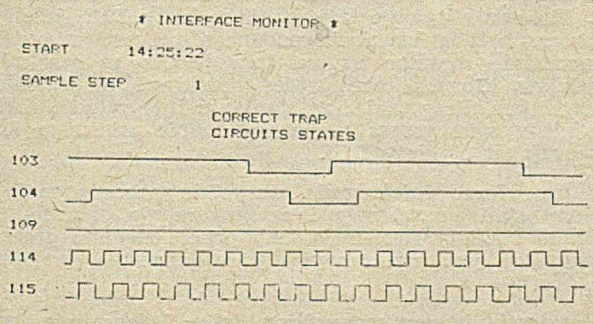
W obu wypadkach parametry styku S2 odpowiadają standardowi RS-232C.

W maksymalnej konfiguracji programowo-sprzętowej użytkownik SPS ma do dyspozycji repertuar 24 komend, na który składają się komendy pomiarowe (6), redagująco-wyprowadzające (9) i pomocnicze (9).

Monitorowanie

SPS umożliwia monitorowanie na dwóch poziomach szczegółowości. Poziom pierwszy — to monitorowanie stanów obwodów styku S2, maksymalnie czternastu, ze zbioru 103—115, 125. Poziom drugi — to monitorowanie transmisji prowadzonej według jednego z następujących protokołów: asynchronicznego, BSC, LAPB, SDLC, X.25/3 (pakietowego).

Prowadzenie monitorowania na poziomie pierwszym jest inicjowane odpowiednią komendą i dialogowym wprowadzeniem odpowiedzi dotyczących numerów monitorowych obwodów i częstotliwości próbkowania (krotność wartości 100 μs). Wyniki uzyskane z przykładowego monitorowania pokazano na rysunku 1.



Rys. 1. Wyniki monitorowania stanu obwodów styku S2

Schemat pomiaru na drugim poziomie szczegółowości jest bardziej złożony i wymaga trzech rodzajów informacji:
- charakterystyki monitorowanego protokołu,
- ogólnych warunków transmisji i pomiaru,
- warunków zakończenia monitorowania, tj. definicji tzw. pułapek.

Przykładową postać dialogu ustalającego warunki monitorowania transmisji według protokołu X.25/3 przedstawia rysunek 2. Przy wprowadzaniu parametrów komend, system podpowiada zakresy wartości i formaty oczekiwanych odpowiedzi. Monitorowanie kończy się w momencie zarejestrowania przez SPS zdarzenia realizującego zdefiniowaną pułapkę lub po wprowadzeniu komendy STOP. W dowolnej chwili monitorowania można uzyskać raport bieżący z jego przebiegu, który zawiera między innymi klasyfikację i liczbę jednostek protokołu zarejestrowanych w strumieniu nadawczym (TX) i odbiorczym (RX). Podobny raport, wzbogacony informacją o czasie trwania i przyczynie zakończenia monitorowania, jest generowany z chwilą jego zakończenia.

```
*MB

     * BIT-ORIENTED PROTOCOL MONITOR *

Line speed (600...9600)          = 2400
Protocol (L/S/X)                 = X
Your tester local to DTE (Y/N)   = Y
Time mark (Y/N)                  = Y
NRZI (Y/N)                       = N

          - Trap events -

Time limit (Y/N)                 = Y
      Value (ms)                 = 2
Traps def. for
      (CS/BS/TX/RX/NO)           = BS

               TX

FCS (Y/N)                        = Y
    No (1...65535)               = 10
Abort (Y/N)                      = Y
    No (1...65535)               = 20
Buffer full (Y/N)                = N
Pattern (Y/N)                    = N

               RX

FCS (Y/N)                        = N
Abort (Y/N)                      = N
Buffer full                      = Y
Pattern (Y/N)                    = Y
      Length (1...6)             = 2
                  # 1            = 11XX00XX
                  # 2            = 1111XXXX
      No (1...65535)             = 10

          * * *
```

Rys. 2. Definiowanie warunków i pułapek monitorowania transmisji według protokołu X.25/3

W trakcie pomiaru SPS zbiera wszelkie istotne informacje o wszystkich obserwowanych jednostkach protokołu, a około 100 ostatnich jednostek z każdego strumienia jest zapamiętywanych w buforze cyklicznym.

Komendy sterujące redagowaniem i wyprowadzaniem są tak dobrane, że pozwalają dokonać analizy jednostek protokołu na różnym poziomie szczegółowości, między innymi zilustrować ich strukturę, wartości parametrów, a także dynamikę wymiany jednostek. SPS rejestruje wszystkie przepływające jednostki, o ile tylko ich struktura, być może szkieletowa, umożliwia synchronizację układu sprzęgającego. Takie nierozpoznane jednostki są w raporcie bieżącym i końcowym specjalnie klasyfikowane, a ich treść można

```
        TX + RX STREAM
RX    10:32:06:540    01 C P U  SABM
TX    10:32:06:580    01 R F U  UA
TX    10:32:06:660    03 C   I  I  N(R)=0  N(S)=0
                      10 00 FB 07 00
RX    10:32:06:720    01 C   I  I  N(R)=0  N(S)=0
                      10 00 FB 00 00
TX    10:32:06:740    01 R   S  RR N(R)=1
RX    10:32:06:800    03 R   S  RR N(R)=1
RX    10:32:14:820    01 C   I  I  N(R)=1  N(S)=1
                      14 32 0B 44 01 01 01 02 05 05 01 43 02 02
                      D7 E2 C9 C5 F9 F0 F1 00 00 3C B3 40
TX    10:32:14:860    01 R   S  RR N(R)=2
```

Rys. 3. Wydruk proceduralny zawartości buforów rejestrujących monitorowanie transmisji według LAPB

wyprowadzić stosując tryb szesnastkowy. Jeżeli wszystkie jednostki monitorowanej transmisji należą do ustalonego protokołu, to ich wyprowadzenie i analizę wygodniej jest prowadzić w trybie procedurowym, tj. zlecając ich dekodowanie, zgodnie z formatem właściwym danemu protokołowi. Przykładem jest rysunek 3 z fragmentem zarejestrowanej transmisji prowadzonej według protokołu LAPB.

Testowanie

SPS umożliwia prowadzenie testowania dwóch rodzajów. Rodzajem pierwszym jest tzw. testowanie standardowe. Realizowana procedura — test 511B według CCITT — pozwala zmierzyć elementową i blokową stopę błędów wybranego kanału transmisji danych. Rodzaj drugi — to tzw. testowanie programowe wykorzystujące specjalnie zbudowany język do nadawania, odbioru i analizy buforów transmitowanych pomiędzy SPS a środowiskiem testowanym.

Tabela 1. Język programowania testów

Kod	Składnia	Opis
00	HALT	Zatrzymanie programu testu
01	EXIT	Koniec definicji programu testu
10	SEND Bi	Nadaj bufor Bi
20	RCEV Bj	Odbieraj dane do bufora Bj
30	CMP Bi, Bj	Porównaj dane w buforach Bi i Bj
31	MASK Bi, Bj	Zeruj bity w Bi zgodnie z Bj
32	TEST Bi, Bj	Zlicz w CO różnicę bitów między Bi i Bj
40	JUMP NK	Skocz bezwarunkowo do NK
41	JPEQ NK	Skocz do NK gdy bufory jednakowe
42	JPNQ NK	Skocz do NK gdy bufory niejednakowe
43	JNS NK	Skocz do NK gdy nadawanie niepoprawne
44	JNR NK	Skocz do NK gdy odbiór niepoprawny
45	JLEQ NK	Skocz do NK gdy Ci równe 0
46	JNLQ NK	Skocz do NK gdy Ci nierówne 0
47	JPTQ NK	Skocz do NK gdy Ti równe 0
50	WAIT n	Czekaj n*20 ms, n $\leq$ 255
51	RUNT n,Ti	Aktywuj stoper Ti z nastawą n*20 ms
52	SET n,Ci	Wpisz n do Ci
53	STPT Ti	Zatrzymaj stoper Ti
60	CLR Ci	Zeruj licznik Ci
62	INC Ci	Zwiększ Ci o 1
63	DEC Ci	Zmniejsz Ci o 1

W testowaniu programowym użytkownik SPS ma do dyspozycji specjalny język programowania testów — TPL. Instrukcje tego języka zestawiono w tabeli 1. TPL dopuszcza operacje na następujących strukturach danych: bufory, liczniki, stopery. Bufory stanowią podstawowe struktury danych, a zawarte w nich informacje są nadawane do (lub odbierane z) urządzenia testowego i podlegają odpowiedniej analizie i obróbce. Liczniki i stopery są strukturami pomocniczymi i umożliwiają łatwą organizację pętli wraz z warunkowaniem działań upływem czasu.

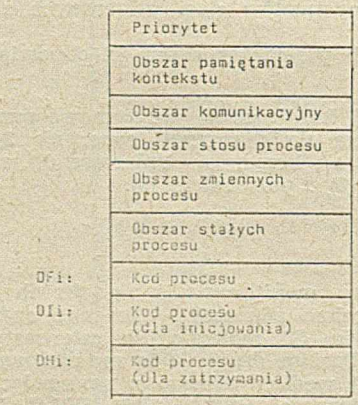
Programując sekwencje testowe, użytkownik SPS ma do dyspozycji oprócz instrukcji języka TPL również inne komendy systemu. Służą one między innymi do podziału przestrzeni dostępnej pamięci na wymaganą liczbę buforów, zapisania i zmodyfikowania inicjalnych wartości danych w tych buforach, zapisania i zmodyfikowania definicji testu oraz uzyskania raportu bieżącego i końcowego z realizacji testu programowego.

Należy podkreślić uniwersalność przyjętego rozwiązania. SPS nie realizuje stałych, wbudowanych sekwencji testowych, lecz oferuje język i dane do generowania takich sekwencji. Istniejące sprzęgi SPS, przeznaczone do transmisji synchronicznej znakowej lub bitowej, dają możliwość testowania urządzeń o różnych sprzęgach sprzętowych i pracujących według różnych protokołów. Najczęściej używane sekwencje testowe mogą tworzyć bibliotekę procedur składowaną na urządzeniach zewnętrznych. Jedynym ograniczeniem implementacyjnym testu programowego jest to, że nie może on być dłuższy od 127 kroków (linii programu).

BUDOWA OPROGRAMOWANIA

SPS należy do klasy systemów działających w czasie rzeczywistym. Budowa oprogramowania aplikacyjnego dla takiego systemu bazuje zwykle na systemie operacyjnym gwarantującym współbieżność, synchronizację i komunikację procesów. Rozwiązaniem alternatywnym jest budowa specjalizowanego monitora, co zwykle gwarantuje krótszy czas reakcji systemu na pojedyncze przerwania, krótszy czas przełączania procesów i mniejszy obszar pamięci przeznaczony na kod systemu (monitora). Te trzy istotne cechy monitora uzyskuje się kosztem rezygnacji z pewnych usług systemu operacyjnego stanowiących o jego uniwersalności, a zatem i przydatności dla wielu aplikacji. Budowę oprogramowania SPS oparto na specjalizowanym monitorze, rezygnując, po analizie, z głębokich modyfikacji firmowego systemu DOPS (zgodny z CP/M). Implementację monitora dla SPS oparto na [6], adaptując go do innego sprzętu i zmieniając pewne usługi zgodnie z wymogami procesów aplikacyjnych.

Oprogramowanie użytkowe zdekomponowano na pewną liczbę statycznych procesów, z których każdy realizował się na pewnej maszynie wirtualnej. Maszynę wirtualną dla pojedynczego procesu tworzy procesor z pamięcią wspólną i dzieloną, system przerwań (źródło pierwotnych zdarzeń w systemie) oraz monitor, dostarczający między innymi usług synchronizacji i komunikacji międzyprocesowej. Monitor, tworząc środowisko dla procesów kodowanych w języku makroasemblera, wymaga, aby każdy z nich miał elementy składowe struktury przedstawione na rysunku 4.



Rys. 4. Elementy składowe struktury procesu

W momencie początkowym monitor zeruje wszystkie tablice, w których utrzymuje informacje o aktualnym stanie całego systemu. Następnie wykonuje inicjowanie każdego z procesów przez wykonanie fragmentów kodów począwszy od etykiet DIi (i — numery procesów). Każdy z procesów ma jeszcze dwa dodatkowe punkty wejścia, tj. DFi — etykieta startu normalnego, DHi — etykieta zatrzymania procesu. Monitor aktywuje proces od etykiety DHi, jeżeli inny proces zażąda usunięcia podanego procesu z systemu. Równocześnie monitor usuwa taki proces ze wszystkich kolejek, w których taki proces oczekiwał na spełnienie swoich żądań. Przy takim zatrzymaniu integralność danych gwarantuje sam proces przez wykonanie własnego kodu, począwszy od etykiety DHi.

Po procedurze inicjowania wszystkie procesy mają ustawiony stan początkowy i są gotowe do „normalnego" startu od etykiety DFi. Gdy dany proces zwalnia procesor, to jego ślad jest przechowywany w „obszarze pamiętania kontekstu".

W systemie jednoprocesorowym tylko jeden proces może wykonywać się w wyróżnionej chwili na procesorze, a pro-

cesy gotowe oczekują w kolejce. Jeżeli procesor staje się wolny, to jest przydzielany procesowi oczekującemu o maksymalnym priorytecie. Proces może znajdować się także w innych kolejkach, tj. czasowej, czasowo-zdarzeniowej i żądań aktywacji. Przesunięcia procesów między kolejkami następują w wyniku realizacji usług monitora. Usługi te, z punktu widzenia procesu, należą do następujących grup wyróżnionych w tabeli 2:

- aktywacji i deaktywacji procesów,
- synchronizacji z innym procesem,
- przesyłania komunikatów między procesami,
- odmierzania czasu.

Jeżeli wymagają one parametrów, to są one przekazywane przez „obszar komunikacyjny".

Tabela 2. Usługi monitora

activate i	— aktywacja procesu i
deactivate i	— deaktywacja procesu i
bye	— bezterminowe zawieszenie procesu
hold n	— zawieszenie procesu na n jednostek
wait e, n	— zawieszenie i oczekiwanie na zdarzenie e co najwyżej przez n jednostek
send e	— sygnalizacja zdarzenia e
clear e	— zerowanie licznika zdarzeń e
send_com i, p	— przesłanie komunikatu w buforze p do procesu i
check_queue	— pobranie bufora z komunikatem z kolejki we (gdy nie pusta) do procesu
get_short_buf	— pobranie krótkiego bufora (z puli buforów systemu)
get_long_buf	— pobranie długiego bufora
free_buf (p)	— zwrot bufora p do puli
start_tic_tim(nj)	— aktywacja czasomierza j z nastawą n jednostek
stop_tic_tim (j)	— wyłączenie czasomierza j
start_mi_tim(n)	— aktywacja minutnika z nastawą n minut
stop_min_tim	— wyłączenie minutnika

REALIZACJA SPRZĘTOWA

SPS zbudowano z modułów systemu MSM firmy Impol 1 [3] oraz specjalizowanych układów sprzęgających wykonanych w Politechnice Wrocławskiej. Podstawowa konfiguracja SPS zawiera 8 modułów połączonych magistralą.

Pierwszy moduł dodatkowy służy do monitorowania obwodów styku S2 w programowanych odstępach czasu. Obwód portu wejściowego zbudowano z wykorzystaniem układu 8255, a blok programowanego generatora przerwań z wykorzystaniem układu licznika 8253. Kaskadowe połączenie dwóch liczników układu 8253 pozwala uzyskać zakres czasów między przerwaniami od pojedynczych mikrosekund do około 2000 s. Ograniczenia programowe powodują, że minimalny okres próbkowania może mieć wartość około 50 μs.

Drugi moduł służy do monitorowania lub sterowania transmisją danych przez styk S2 dla protokołów asynchronicznych lub synchronicznych znakowych. Blok układu transmisji zbudowano z użyciem dwóch układów USART 8251A, które w zależności od trybu pracy są wykorzystywane jako układ nadajnika-odbiornika lub dwa układy odbiorników. Moduł ma własny blok zegara zbudowany z dwóch liczników układu 8253, co umożliwia programowy wybór wewnętrznej lub zewnętrznej (modemowej) podstawy czasu. Systemowy port wejściowy umożliwia odczyt wybranych linii styku S2, a systemowy port wyjściowy umożliwia sterowanie obwodem 111 styku S2, wybór podstawy czasu i maskowanie przerwań generowanych przez układy USART.

Trzeci moduł służy do monitorowania lub sterowania transmisją przez styk S2 dla protokołów synchronicznych bitowych. Blok transmisji jest zbudowany z użyciem dwóch układów 8273, które w zależności od trybu pracy są wykorzystywane jako układ nadajnika-odbiornika lub dwa układy odbiorników. SPS obsługuje układy 8273 w trybie programowego DMA. Moduł ma własny oscylator o częstotliwości 4 MHz i blok zegara zbudowany z układu 8253. Systemowy port wyjściowy umożliwia maskowanie przerwań generowanych przez układy 8273 i wybór podstawy czasu. Systemowy port wejściowy umożliwia odczyt stanu wybranych obwodów styku S2, co zapewnia łatwą diagnostykę zachowań modemu.

Od momentu zdefiniowania zadania do zakończenia testowania tworzenie SPS trwało około 1,5 roku. Do budowy oprogramowania zastosowano dość prymitywne narzędzie (ale jedynie dostępne) jakim był język makroasemblera. W rezultacie wymagało to napisania prawie 22 tys. linii programu źródłowego. Kod wynikowy wszystkich procedur monitorowania i testowania zajął odpowiednio 34 i 26 KB pamięci operacyjnej (plus obszary buforowe). Wszystkie fazy budowy oprogramowania zrealizował pięcioosobowy zespół. W jego skład, oprócz autora, wchodzili: I. Dubielewicz, K. Koleśnik, W. Komorowski i L. Misiura.

Główną charakterystyką użytkową SPS jest zdolność monitorowania i testowania transmisji dupleksowej do szybkości 9600 b/s, co potwierdziły badania. W ich trakcie analizowano również ergonomię obsługi SPS i czytelność generowanych wyników gromadząc na ten temat także uwagi osób spoza kręgu realizatorów. Zmodernizowana, obecna wersja 1.20 jest częściowo wolna od stwierdzonych niedomagań, jakkolwiek pewne udoskonalenia odłożono do momentu budowy urządzenia nowej generacji.

LITERATURA

[1] Data Analyzer DA-10. Wandel und Golterman, 1981
[2] Data Communications Test Sets. Electrodata Inc.
[3] Dokumentacja systemu MSM. PPZ Impol-1, Warszawa
[4] Dubielewicz I., Fryźlewicz Z., Koleśnik K., Komorowski W., Misiura L.: Aparatura pomiarowo-diagnostyczna dla sieci SKJS/2. Dokumentacja eksploatacyjna, Raport SPR 12/86, Centrum Obliczeniowe Politechniki Wrocławskiej 1986
[5] Measurement, Computation, Systems: Katalog firmy Hewlett-Packard, 1984
[6] Thorelli L.-E.: A Monitor for Small Computers. Software Practice and Experience, Vol. 8, pp. 439—450, 1978

Skojarzenie parametrów z argumentami

dokończenie ze str. 16

Jeśli parametr funkcji jest zadeklarowany jako tablica, to jest traktowany jak zmienna wskazująca dane o postaci elementów tej tablicy. Oznacza to, że program przedstawiony na wydruku 4 jest równoważny programowi przedstawionemu na wydruku 5.

Jeśli parametr jest zadeklarowany jako zmienna typu (char) albo (short int), to jest traktowany tak, jakby był typu (int), a jeśli jest zadeklarowany jako zmienna typu (real), to jest traktowany tak, jakby był typu (double). Każdemu z przekształconych parametrów jest przypisywana dana początkowa powstała w konwersji argumentu na pierwotny typ parametru. Oznacza to, że program przedstawiony na wydruku 6 powinien być traktowany tak jak równoważny mu program przedstawiony na wydruku 7.

Po uwzględnieniu wszystkich omówionych tu przekształceń argumentów i parametrów wymaga się, aby odpowiadające sobie parametry i argumenty były dokładnie takiego samego typu. Oznacza to w szczególności, że argumentowi typu (char) może odpowiadać, na przykład, parametr typu (short int), ale nie może odpowiadać, na przykład, parametr typu (real).

W odróżnieniu od funkcji znanych z innych języków programowania, funkcje języka C nie muszą udostępniać rezultatu. Jeśli jednak są wykorzystywane tak jak prawdziwe funkcje, każde ich wywołanie musi kończyć się wykonaniem instrukcji return zawierającej wyrażenie. Wyrażenie takie, poddane ewentualnej konwersji na typ funkcji, stanowi wówczas jej rezultat. Oczywiście wymaga się, aby wspomniana konwersja była wykonalna. Oznacza to, że program taki jak przedstawiony na wydruku 7 jest poprawny i równoważny programowi przedstawionemu na wydruku 8. Wykonanie każdego z tych programów powoduje wyprowadzenie liczby 13.

MICHAŁ KIRPLUK
PIOTR SOBOLEWSKI
Instytut Informacji Naukowej, Technicznej
i Ekonomicznej
Warszawa

Implementacja dedukcyjnej bazy danych Holmes (I)

Zapotrzebowanie na dostęp do wielkich zbiorów informacji stało się impulsem do powstania i rozwoju baz danych. Niezależnie od baz danych rozwijały się systemy sztucznej inteligencji, których wspólną cechą było posiadanie mechanizmów dedukcji, umożliwiających uzyskanie informacji nie zawartych wprost w bazie danych. Powszechnie stosowanym narzędziem implementacji i opisu tych systemów jest logika klasyczna. Logika stanowi dla nich podstawę do reprezentacji wiedzy, jak też przesądza o ich dedukcyjnym charakterze.

W drugiej połowie lat siedemdziesiątych powstała koncepcja połączenia teorii relacyjnych baz danych z logiką klasyczną. Jego celem było uzyskanie systemów mogących efektywnie zarządzać zbiorem danych, przy jednoczesnej możliwości korzystania z mechanizmów wnioskowania. Klasę systemów o powyższych właściwościach nazwano dedukcyjnymi bazami danych.

Rozwiązaniem różniącym się od podejścia relacyjnego było podejście Chena, bazujące na modelu ER (ang. entity-relationship) [3]. Model ten w dalszej części nazywać będziemy obiektowo-relacyjnym. System dedukcyjnej bazy danych Holmes jest systemem łączącym w sobie cechy obiektowo-relacyjnej bazy danych z właściwościami logiki klasycznej [8]. Zastosowanie systemu Holmes ogranicza się do niedużych, specjalizowanych baz danych zawierających zbiór faktów, reguły wnioskowania oraz reguły spoistości logicznej. Klasycznym przykładem może być wiedza na temat drzewa genealogicznego pewnej rodziny. W bazie mamy zbiór obiektów — osoby, faktów — relacje ojcostwa itp., definicje dalszych pokrewieństw oraz reguły prawdziwe dla całej bazy. Posiadanie takiej bazy w systemie Holmes umożliwia zadawanie pytań, na które odpowiedź nie istnieje wprost, a system odpowiada na nie korzystając z mechanizmów wnioskowania. Możliwe jest też znajdowanie obiektów o zadanej formule, jak i kontrolowanie spoistości logicznej bazy, która mogłaby być zachwiana przez zmiany dokonane w bazie.

W artykule opisano implementację systemu dedukcyjnej bazy danych, poprzedzając jej opis podstawowymi wiadomościami dotyczącymi baz tego rodzaju [7].

SYSTEMY RELACYJNE A LOGIKA KLASYCZNA

Większość prac poświęconych teorii baz danych jest oparta na modelu relacyjnym zdefiniowanym przez Codda w 1970 roku [4]. Model ten realizuje odwzorowanie świata rzeczywistego w bazie danych za pomocą zbioru wartości obiektów powiązanych ze sobą relacjami. Relacje te mogą być jedno- lub wieloargumentowe i zmieniane w czasie przez procesy aktualizacji. Upraszczając, relacyjna baza danych może być widziana jako zbiór tabel reprezentujących relacje. Wiersze w tabelach zwane są krotkami. Oznaczają one konkretne, elementarne fakty (np. LECH jest-ojcem WIKTORII). Każda kolumna tabeli ma nazwę zwaną atrybutem. Zawiera ona wartości tego atrybutu pochodzące z jego dziedziny. Pod pojęciem dziedziny rozumiany jest zbiór wszystkich dopuszczalnych wartości atrybutu. Dziedzina całej n-członowej relacji jest iloczynem kartezjańskim dziedzin wszystkich jej członów. Zakłada się, że kolejność krotek w bazie danych jest nieistotna oraz, że w danej relacji nie występują dwie identyczne krotki.

Duże zainteresowanie relacyjnymi bazami danych wynika głównie z prostoty modelu zarówno w praktyce, jak i w teorii. W początkowym okresie teoria relacyjnych baz danych rozwijała się niezależnie od logiki. Dopiero w roku 1978 pojawiły się pierwsze prace wskazujące na jej związki z logiką matematyczną [9]. Szczególnie zastosowanie logiki pierwszego rzędu pozwoliło na precyzyjne zdefiniowanie relacyjnego modelu baz danych, uproszczenie jego opisu, sformalizowanie teorii zależności, a co najważniejsze — rozszerzyło go o mechanizmy wnioskowania dedukcyjnego [5]. Język logiki pierwszego rzędu posłużył do wyrażania faktów elementarnych dotyczących danej rzeczywistości, jak też do wyrażenia ogólnych prawidłowości rządzących tymi faktami. Te ogólne prawidłowości tworzą zbiór zasad spójności określający prawa, z którymi powinna być zgodna spoista logicznie baza danych. Prawa te mogą być opisane za pomocą formuł zbudowanych na wzór wyrażeń logicznych. Bazę danych można uważać za dedukcyjną, jeżeli oprócz mechanizmów klasycznej bazy danych funkcjonują w niej dodatkowo mechanizmy dedukcji. Mechanizmy te działają na pewnych aksjomatach opisujących prawa świata odwzorowanego w bazie danych. W bazie tej nowe fakty można wywieść z faktów wprowadzonych wprost do niej.

W praktyce do opisu systemów dedukcyjnych baz danych używa się rachunku predykatów pierwszego rzędu. Formule w języku logiki odpowiada klauzula w rachunku predykatów, która powstaje z tej formuły po likwidacji kwantyfikatorów i symboli funkcji charakterystycznych. Ogólna postać klauzuli jest następująca:

P1 & P2 & ... & PK → R1 V R2 V ... V Rq

gdzie argumenty predykatów Pi, Rj (i=1...k, j=1...q) są stałymi bądź zmiennymi (zmiennymi indywiduowymi). W zależności od wartości k i q wyróżnia się następujące typy klauzul:

— typ 1 (k=0, q=1): →R(t1,...,tm)

Gdy t1,...tm są stałymi, to klauzula jest faktem w bazie danych. W przypadku, gdy co najmniej jedno ti jest zmienną, klauzula stanowi zasadę spójności.

— typ 2 (k=1, q=0): P(t1,...,tm) →

Gdy każdy argument ti jest stałą, klauzula opisuje negatywny fakt (przy założeniu, że baza nie zawiera negatywnych informacji, jest on nieistotny). Gdy istnieją zmienne ti, może to być zasada spójności lub wartość pusta.

— typ 3 (k>1, q=0): P1&P2&...&Pk →

Może to być zasada spójności.

— typ 4 (k>=1; q=1: P1&P2&...&Pk → R1

Klauzula ta może być zasadą spójności lub definicją predykatu R1 (definicja jest prawem dedukcji).

— typ 5 (k=0, q>1): → R1VR2V...VRq

Jeżeli argumentami Ri (i=1...q) są stałe, to mamy do czynienia z nieokreślonym twierdzeniem, tzn. suma logiczna Ri jest prawdą, lecz nie wiadomo, który wniosek Ri jest prawdziwy.

— typ 6 (k>=1, q>=1): P1&P2&...&Pk → R1VR2V...VRq

Klauzula ta może być interpretowana jako zasada spójności lub definicja nieokreślonej danej.

Dedukcyjne bazy danych można podzielić na dwa rodzaje: dobrze określone i nieokreślone. Dobrze określone bazy danych nie zawierają nieokreślonych klauzul (typ 5 i 6). Natomiast nieokreślone bazy danych dopuszczają te klauzule. Ich nieokreśloność polega na braku możliwości otrzymania jednoznacznych wniosków, jak również na posługiwaniu się niejednoznacznymi faktami.

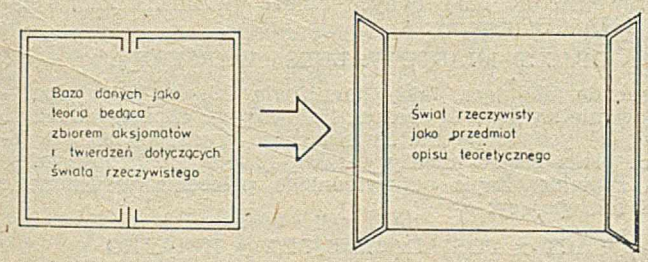
Z punktu widzenia logiki, baza danych może być traktowana w dwojaki sposób: jako „interpretacja" lub jako „teoria". Obydwa podejścia są formalnie równoważne.

Baza danych	
$-\alpha\beta-$ Świat rzeczywisty będący realizacją bazy danych $\alpha 1$	Poziom rzeczywistości
$-\alpha 1-$ Baza danych jako realizacja języka $\alpha 2$	Poziom bazy danych
$-\alpha 2-$ Wyrażenia języka	Poziom języka

Rys. 1. Baza danych jako „interpretacja"

Rozważmy bazę danych jako „interpretację" (rys. 1). W tym podejściu mamy do czynienia z dwoma poziomami realizacji języka bazy danych. Pierwszy poziom to realizacja języka w formie bazy danych, zaś drugi — to realizacja bazy danych w modelowanym przez nią świecie rzeczywistym.

Pytania i zasady spójności wyrażone w języku $\alpha 2$ są wtedy formułami interpretowanymi na podstawie bazy danych. Pytaniom odpowiada pewien podzbiór bazy danych, zaś formułom spoistości odpowiada baza danych spójna logicznie, czyli pewien model w rozumieniu logiki matematycznej [6]. Nieformalnie można stwierdzić, że każdy model w powyższym znaczeniu określa pewną rzeczywistość, którą specyfikują formuły zawarte w bazie danych. Większość klasycznych systemów relacyjnych przy przejściu od pytania do jego interpretacji w bazie danych korzysta z algebry relacji [1, 2].



Rys. 2. Baza danych jako „teoria"

Rozpatrując bazę danych jako „teorię" (rys. 2), można ją uważać za rozszerzenie zbioru twierdzeń zadanego przez definicje i zasady spójności. Fakty stanowiące zawartość bazy danych określa się jako aksjomaty specyficzne. Są one charakterystyczne tylko dla danej rzeczywistości, opisywanej przez tę bazę danych. Zasady spójności oraz definicje będące aksjomatami logicznymi teorii opisują wiele modeli. Odpowiedź na pytanie w tym podejściu polega na udowodnieniu twierdzenia określonej „teorii".

SYSTEM DEDUKCYJNEJ BAZY DANYCH HOLMES

Holmes jest systemem dobrze określonej dedukcyjnej bazy danych opartym na koncepcji obiektowo-relacyjnej. Charakteryzuje go jednolite podejście do modelowania świata (typowe dla relacyjnych baz danych), w którym jedynymi konstrukcjami do opisywania wszystkich związków i zależności są obiekty i relacje zachodzące między nimi.

Opis systemu

Każdy obiekt istniejący w bazie danych jest opisany przez swój typ (relację unarną). Relacje, typy i obiekty są jednoznacznie identyfikowane przez swoje nazwy. Każdy obiekt może być kilku typów i wchodzić w skład wielu relacji. Każda relacja może być widziana jako podzbiór produktu kartezjańskiego określonego na zbiorach obiektów. Elementarnymi jednostkami tego produktu są fakty określające relacje zachodzące między konkretnymi obiektami. W tym ujęciu zbiór faktów elementarnych stanowi zawartość bazy danych systemu. Schemat bazy danych deklaruje i definiuje typy, relacje oraz zasady spójności. Można powiedzieć, że schemat stanowi dla bazy danych „teorię" określającą jej strukturę, przez co baza danych zgodna ze schematem staje się jednym z modeli dla tak określonej „teorii".

W systemie Holmes mamy do czynienia z dwoma rodzajami schematu. Pierwszy to schemat ogólny, wprowadzany wsadowo, w którym możliwe jest zawieszanie i odwieszanie wybranych definicji. Trwałej jego zmiany można dokonać przez edycję pliku zawierającego ten schemat. Wszystkie czynności związane z dokonywaniem zmian w schemacie ogólnym należą do administratora bazy danych i są niedostępne dla zwykłych użytkowników. Mogą oni tylko korzystać ze schematu, jako ze wspólnego, jednakowego dla wszystkich spojrzenia na bazę danych. Oprócz tego każdy użytkownik może dodać do schematu ogólnego własne widzenie bazy danych. Jest ono indywidualnym uzupełnieniem schematu ogólnego, powodującym poszerzenie całkowitego schematu dostrzeganego przez użytkownika. Z tego powodu dostęp do schematu użytkownika ma charakter lokalny. W czasie pracy oba schematy stanowią dla użytkownika spójną całość, przy czym może on dokonywać zmian tylko we własnym schemacie lokalnym. Standardowo, schemat ogólny jest najszerszym schematem dla bazy danych. Schematy indywidualne użytkowników powstają jako uszczegółowione wycinki schematu ogólnego, kopiowane na potrzeby użytkownika. Podejście zastosowane w systemie Holmes daje użytkownikowi duże możliwości rozbudowy własnego widzenia bazy danych. W pewnym stopniu zapewnia ono też ochronę schematu użytkownika.

Podobnie jak w wypadku każdego systemu baz danych, Holmes zapewnia wspomaganie procesu projektowania określonej aplikacji oraz realizację procesów utrzymywania bazy danych i obsługi użytkowników. W fazie projektowania konkretnego zastosowania wprowadzany jest schemat ogólny w postaci pliku deklaracji i definicji. Projektowa-

```
PRIMARY TYPES
        POCIAG
        MIASTO

PRIMARY RELATIONSHIPS
        POCIAG  MA_PRZYSTANEK_NR  INTEGER  NA_STACJI  MIASTO
        POCIAG  STAJE_NA_STACJI  MIASTO  O_GODZ  INTEGER
        POCIAG  JEST_TYPU  ("EXPRESS","POSPIESZNY","OSOBOWY")
                NA_TRASIE_OD_STACJI  MIASTO  DO  MIASTO
        POCIAG  POSIADA_MIEJSCA ("SYPIALNE","DO_LEZENIA","PL1",
                "PL2")
DERIVED TYPES
        EXPRESS
        POSPIESZNY
        OSOBOWY

DERIVED RELATIONSHIPS
        POCIG JEDZIE_OD_STACJI  MIASTO  DO_STCJI  MIASTO
        MIASTO  JEST_STACJA_POCZATKOWA_DLA_POCIAGU  POCIAG
        MIASTO  JEST_STACJA_DOCELOWA_DLA_POCIAGU  POCIAG

DEFINITIONS
  JEDZIE_BEZPOSREDNIO: FORALL x,y,z (x JEDZIE_OD_STACJI  y  DO  z  IFF
        EXIST a1,a2:INTEGER (x MA_PRZYSTANEK_NR a1 NA_STACJI y AND  x
        MA_PRZYSTANEK_NR a2 NA_STACJI z AND a1<a2))

  STACJA_POCZATKOWA: FORALL y,x  (y  JEST_STACJA_POCZATKOWA_DLA_POCIAGU
        x IFF x MA_PRZYSTANEK_NR 1 NA_STACJI y)

  STACJA_DOCELOWA: FORALL y,x  (y  JEST_STACJA_DOCELOWA_DLA_POCIAGU  x
        IFF EXIST z:INTEGER (x MA_PRZYSTANEK_NR z  NA_STACJI  y)  AND
        NOT EXIST v:MIASTO,s:INTEGER (x MA_PRZYSTANEK_NR s  NA_STACJI
        v AND x MA_PRZYSTANEK_NR z NA_STACJI y) IMPLY (s>z))

  EXPRESS: FORALL x (x:EXPRESS IFF EXIST z:MIASTO,y:MIASTO x JEST_TYPU
        "EXPRESS"  NA_TRASIE_OD_STACJI  y  DO_STACJI  z  AND  y
        JEST_STACJA_POCZATKOWA_DLA_POCIAGU  x  AND  z
        JEST_STACJA_DOCELOWA_DLA_POCIAGU x)

  POSPIESZNY: FORALL  x  (x:POSPIESZNY  IFF  (EXIST  y,z:MIASTO  (x
        JEST_TYPU "POSPIESZNY" NA_TRASIE_OD_STACJI  y  DO_STACJI  z))
        AND (NOT EXIST y1:MIASTO, z1:MIASTO (x  JEST_TYPU  "OSOBOWY"
        NA_TRASIE_OD_STACJI y1 DO_STACJI z1)))

  OSOBOWY: FORALL x (x:OSOBOWY IFF EXIST z:MIASTO,y:MIASTO x JEST_TYPU
        "OSOBOWY" NA_TRASIE_OD_STACJI y DO z)

CONSTRAINTS
    START: FORALL x:POCIAG EXIST y:MIASTO
        (y JEST_STACJA_POCZATKOWA_DLA_POCIAGU x)
    TRASA: FORALL x:POCIAG EXIST y:MIASTO,z:MIASTO
        ((y JEST_STACJA_POCZATKOWA_DLA_POCIAGU x AND
        z JEST_STACJA_DOCELOWA_DLA_POCIAGU x) AND (y<>z))
    POLACZENIE:  FORALL  y:MIASTO,z:MIASTO  ((EXIST  x:POCIAG
        x JEDZIE_OD_STACJI y DO_STACJI z) IMPLY (EXIST w:POCIAG
        w JEDZIE_OD_STACJI z DO_STACJI y))
```

Tabela 1. Przykładowy schemat dotyczący sieci kolejowej

niem aplikacji zajmuje się administrator bazy danych. Przez komendy systemu użytkownik ma dostęp do bazy danych zdefiniowanej przez administratora.

Komendy dzielą się na trzy grupy:

- komendy administrowania bazą danych, które pozwalają na kontrolowanie spoistości bazy danych oraz na czasowe zmiany jej głównego schematu;
- komendy modyfikacji i aktualizacji bazy danych, które obejmują mechanizmy dodawania i usuwania obiektów i faktów;
- komendy indywidualnego użytkownika, które umożliwiają korzystanie z funkcji oferowanych przez system, takich jak tworzenie własnego podschematu, wyszukiwanie informacji itp.

Istnieją dwa poziomy współpracy z systemem. Pierwszy poziom służy do projektowania konkretnej, nowej aplikacji, zaś drugi umożliwia korzystanie z aplikacji stworzonej wcześniej. Aby rozpocząć pracę na pierwszym poziomie należy najpierw określić schemat ogólny definiujący konkretne zastosowanie i przygotować plik tekstowy zawierający odpowiedni ciąg bloków deklaracji i definicji. Plik ten można utworzyć korzystając ze standardowego edytora tekstu. Kolejność bloków deklaracji i definicji w pliku oraz treść tych bloków ilustruje tabela 1, w której przedstawiono przykładowy schemat opisujący sieć kolejową.

```
>holmes

        Dzien dobry! , prosze podac haslo
                ----------
              ->piotr_____

          Dysk systemowy: C:\system\holmes
                          /\
     Prosze podac nazwe   /!!\   ____________
  dysku uzytkowego (<CR> gdy !! ) ->B:\_______


          Prosze podac nazwe schematu ogolnego
               nazwa --> max 6 znakow _______
                                     ->gensch
          Prosze podac nazwe bazy danych !!!
               nazwa --> max 6 znakow _______
                                     ->kolej_
          Prosze podac nazwe schematu indywidualnego !!!
               nazwa --> max 6 znakow _______
                                     ->schind


     Prosze podac komende systemu HOLMES !!!
klawisz CR - wyslanie komendy; komenda EXIT - koniec pracy z systemem

:-------------------------------------------------------------------------
APPLY FORMULA FORALL x:POCIAG
      (x:EXPRESS IMPLY x POSIADA_MIEJSCA "KL1")

            /*************************************************************
------------/      <POZYCJA = 172 > ! <DLUGOSC = 131 >
```

Tabela 2. Konwersacja z systemem na poziomie aplikacji

Aby działać na drugim poziomie, użytkownik powinien posiadać zaprojektowany wcześniej plik ze schematem ogólnym. Na poziomie tym system, po skompilowaniu schematu, przechodzi do trybu interpretacji komend. Przykład konwersji użytkownika z systemem na tym poziomie przedstawia tabela 2.

Język systemu

Podstawowym elementem języka, za pomocą którego określany jest zarówno schemat główny, jak i treść komend, jest język formuł. Został on zdefiniowany tak, aby wyrażane w nim formuły były jak najbardziej zbliżone do zdań języka naturalnego. Język formuł systemu Holmes cechuje ponadto moc oraz jednoznaczność właściwa dla języka rachunku predykatów pierwszego rzędu.

Podstawowymi składnikami języka formuł są:

- elementarny term czyli nazwa zmiennej lub obiektu traktowana jako stała;
- opis typu, tj. wyrażenie składające się z nazw typów połączonych operatorami logicznymi AND, OR, NOT oraz nawiasami, służące do określenia zakresu zmiennej;
- elementarna formuła, tj. nazwa n-arnej relacji wraz z argumentami w postaci elementarnych termów.

Formułę bezkwantyfikatorową można zbudować korzystając z opisów typów, elementarnych formuł i operatorów boolowskich AND, OR, NOT, IMPLY, IFF. Operatory boolowskie mają takie samo znaczenie jak w logice (IMPLY to implikacji, IFF — równoważność). Najbardziej ogólną postać formuły stanowi formuła kwantyfikatorowa. Korzysta ona z kwantyfikatorów egzystencjalnych (EXIST) i ogólnych (FORALL), przez co znacznie poszerza się jej zakres znaczeniowy. Przykład formuły pokazano poniżej:

FORALL x:OSOBA,y:PUBLIKACJA (((x NAPISAL y OR x CZYTAL y) AND y:PUBLIKACJA_NAUKOWA AND y OMAWIA_PROBLEM "etyka") IMPLY x ZNA_ZAGADNIENIE "etyka".

W formule tej można wyróżnić m.in. takie elementy, jak: "etyka",x,y — elementarne termy;

:OSOBA — opis typu;
y:PUBLIKACJA_NAUKOWA, x NAPISAL y — elementarne formuły.

Aby formuła miała postać bliższą językowi naturalnemu, dopuszczalne jest stosowanie rozproszonych nazw relacji, tj. postaci infiksowych, np.:
x R1 y R1' z

gdzie R1&R1' jest pełną nazwą tej relacji. Większą czytelność mogą też zapewnić konstrukcje WHO i WHICH. Na przykład, wyrażenie:

x R3 y WHO R4 z WHICH R4 v
jest bardziej czytelne niż równoważne mu wyrażenie:
(x R3 y) AND (y R4 z) AND (z R5 v).

W celu oswojenia Czytelnika z językiem systemu przedstawimy kilka przykładów użycia komend. Po stworzeniu schematu głównego projektant aplikacji może dokonać jego modyfikacji przez zawieszenie (czasowe logiczne usunięcie) wybranych definicji i zasad spójności. Przykładowo, komenda:

SUSPEND EXPRESS

zawiesza definicję o etykiecie EXPRESS. Chcąc wprowadzić nową definicję do schematu ogólnego należy wcześniej sprawdzić jej poprawność w stosunku do schematu i bazy danych. Służy do tego komenda APPLY, np.:

APPLY FORMULA FORALL x: POCIAG
(x:EXPRESS IMPLY x POSIADA_MIEJSCA "KL1")

Do najważniejszych zadań systemu bazy danych należy modyfikacja i aktualizacja danych. Na przykład, chąc wprowadzić do bazy danych obiekt "WARSZAWA" o typie MIASTO piszemy:

ADD OBJECT MIASTO VALUES "WARSZAWA"

Komenda dodająca fakty umożliwia ich grupowe specyfikowanie za pomocą formuł. Załóżmy, że chcemy dodać do bazy danych grupę faktów stwierdzających, że pociągi jadące przez Kutno między godziną 12.00 a 15.00 mają miejsca sypialne. Zapis odpowiedniego polecenia w Holmesie jest następujący:

ADD FACT x POSIADA_MIEJSCA y HOLDING
(x) x STAJE_NA_STACJI "KUTNO" O_GODZ z AND z>12.00 AND z<15.00
(y) y:("SYPIALNE")

Komenda CHANGE umożliwia dokonywanie zmian nazw obiektów. Poniższe użycie komendy CHANGE, odnoszące się tylko do pociągów pospiesznych, zmienia nazwę obiektu "KRAKOW" na "KRAKOW_GL":

CHANGE NAME OF x HOLDING EXIST y:POCIĄG, z:INTEGER
(y MA_PRZYSTANEK_NR z NA_STACJI "KRAKOW" AND y:POSPIESZNY) SETTING "KRAKOW"="KRAKOW.GL"

Operacje wyszukiwania informacji w bazie danych realizują komendy FIND. Na przykład, chcąc znaleźć pociągi jadące przez Warszawę można użyć następującej komendy:

FIND x AS WA_WSKI HOLDING EXIST y:MIASTO
(x JEDZIE_OD_STACJI "WARSZAWA" DO y)

Znalezionym obiektom nadawany jest typ pierwotny WA-WSKI.

Pełny wykaz komend systemu Holmes przedstawiono w tabeli 3.

ACTIVATE	-aktywowanie relacji, typów lub zasad spójności
ADD FACT	-dodanie faktów
ADD OBJECT	-dodanie obiektów
APPLY	-sprawdzanie formuły w oparciu o bazę i schemat
CHANGE	-zmiana nazwy typu lub obiektu
DEFINE RELATIONSHIPS	-definicja relacji
DEFINE TYPE	-definicja typu
DELETE CONSTRAINTS	-usunięcie zasady spójności
DELETE DEFINITION	-usunięcie definicji
DELETE FACT	-usunięcie faktu
DELETE OBJECT	-usunięcie obiektów
EXIT	-wyjście z systemu
FIND	-znalezienie obiektów lub faktów spełniających podaną formułę na czas sesji
KEEP	-zachowanie na stałe faktów lub obiektów znalezionych komendą FIND
LIST DEFINITIONS	-wyświetlanie treści definicji
LIST NAMES	-wyświetlanie nazw wszystkich typów, relacji i zasad spójności
SUSPEND	-zawieszanie relacji, typów lub zasad spójności
VALIDATE	-sprawdzanie spoistości logicznej bazy tj. sprawdzanie wszystkich zasad spójności na schemacie i bazie.

Tabela 3. Komendy systemu Holmes

LITERATURA

[1] Boyce R. F., Chamberlin D. D., Reisner P.: Human factors evaluation of two data base query language — SQUARE and SEQUEL. Proc. NCL, 1975

[2] Chang C. L.: Deduce 2 — further investigations of deduction in relational databases. Plenum Press, New York, 1980

[3] Chen P. P. S.: The entity-relationship model toward a unified view of data. ACM Trans. on Database Systems, Vol. 1, No. 1, 1976

[4] Codd E. F.: A relational model of data for shared data banks. Communications of the ACM, Vol. 13, No. 6, 1970

[5] Gallaire H., Minker J., Reiter R.: Deductive question — answering on relational databases. Plenum Press, New York, 1978

[6] Gburzyński P.: Automatyczne dowodzenie twierdzeń z wykorzystaniem zasady rezolucji. Warszawa, 1978

[7] Getta J., Rybiński H.: Holmes — a deduction augmented database management system. Information Systems, Vol. 9, 1984

[8] Kirpluk M., Sobolewski P.: Projekt i implementacja mechanizmów dedukcji w systemie bazy danych Holmes. Praca magisterska, Politechnika Warszawska, 1987

[9] Nicolas J-M.: First order logic formalization for functional, multivalued and mutual dependencies. ACM, New York, 1978.

Ze świata

W pracowniach IBM

Projekty firmy IBM, które przynoszą jej największe powodzenie, pochodzą w większości z laboratoriów centrum badawczego Tomas J. Watson Research Center. Nazwane tak od nazwiska fundatora IBM, centrum to od 40 lat stanowi awangardę technologii komputerowej. Znajduje się w Yorktown Heights w stanie Nowy Jork. Ponad 2 tys. naukowców pracuje w pięciu dziedzinach: fizyce, matematyce, informatyce, technice łączności oraz technologii układów scalonych. W każdej z tych dziedzin prowadzone są prace od najbardziej teoretycznych, aż do konkretnych zastosowań praktycznych.

Na przykład w dziedzinie fizyki naukowcy z Yorktown Heights zajmują się strukturą materiałów, zjawiskami magnetyzmu i nadprzewodnictwa, biofizyką, astrofizyką, fizyką cząstek oraz fizyką kwantową.

Jeżeli chodzi o matematykę, to prace prowadzone w Thomas J. Watson Research Center obejmują: algebrę liniową, teorię aproksymacji, kombinatorykę, matematyczną koncepcję układów VLSI i modele statystyczne.

Wszystkie te badania zmierzają do jednego celu, którym jest tworzenie lepszych i szybszych komputerów. Dlatego pracuje się nad tworzeniem szybszych podzespołów oraz nad koncepcją architektury komputera, która spełniałaby nakładane wymagania.

Firma IBM opracowała wiele projektów komputerów i od 1979 roku coraz więcej z nich dotyczy architektury równoległej. Jednym z nich jest RP3 (ang. research parallel processor project), który powstał w ubiegłym roku. System operacyjny dla tego komputera jest tworzony w Courant Institute of Mathematical Sciences uniwersytetu w Nowym Jorku. Jeżeli chodzi o architekturę tego komputera, to przypuszczalnie firma IBM wzoruje się na europejskim projekcie Supernode oraz na komputerach serii T firmy Floating Point Systems.

Jednostka RP3 ma zawierać 64 procesory 32-bitowe. Łącząc wiele takich jednostek otrzyma się konfigurację kilkuset procesorów dysponujących wspólną pamięcią wielu GB. Na przykład system zbudowany z 8 jednostek będzie zawierał 512 procesorów z pamięcią wspólną 2 GB. Będzie on miał szybkość rzędu miliarda instrukcji na sekundę, operacje arytmetyczne będą wykonywane z szybkością 800 Mflops (milionów operacji zmiennoprzecinkowych na sekundę).

Komputer RP3 jest wyrazem tendencji do tworzenia architektur całkowicie równoległych. Ale IBM pracuje również nad tworzeniem systemów informatycznych, w których stopień równoległości — tj. liczba procesorów pracujących równolegle — jest mniej ważny, a główny nacisk kładzie się na poprawienie parametrów poszczególnych procesorów.

Jednym z takich projektów jest LCAP. Pracuje nad nim grupa 50 naukowców pod kierunkiem Enrico Clementi. LCAP nie będzie nowym komputerem, lecz raczej wykorzystaniem zależności między już istniejącymi. Chodzi o połączenie w jądrze systemu obliczeń zarówno skalarnych, jak i wektorowych. LCAP znajdzie zastosowanie przede wszystkim w biofizyce i ma być ostatecznym dowodem na to, że firma IBM opanowała już sztukę projektowania i oprogramowania systemów komputerowych o architekturze równoległej.

Innym rodzajem architektury systemów komputerowych, któremu również naukowcy wróżą wielką przyszłość, jest RISC (ang. reduced instruction set computer). W tej dziedzinie, w firmie IBM prowadzone są badania nad rozwojem eksperymentalnego minikomputera IBM 801. Na jego przykładzie widać, jak długi i złożony może być proces przekształcenia projektu w produkt handlowy. W tym wypadku trwał on ponad 10 lat.

Od 1979 roku, w czasie gdy złożone systemy typu 370 odnosiły największe sukcesy, naukowcy z firmy IBM zauważyli, że najczęściej używanymi rozkazami są te najprostsze oraz że większość programów wykorzystuje tylko ok. 20% bogatych list rozkazów. W dodatku znaczna część czasu maszynowego jest zużywana na oczekiwanie na przesłanie między pamięcią centralną i procesorem. Te spostrzeżenia stały się podstawą stworzenia minikomputera 801, którego ogólna charakterystyka jest następująca:

- lista rozkazów ograniczona do rozkazów prostych wykonywalnych w jednym cyklu maszynowym,
- doskonały kompilator,
- równoległe przesyłanie danych,
- sposób adresowania pamięci upraszczający programowanie.

Przede wszystkim więc 801/Risc jest komputerem prostym. Uzyskane wyniki są efektem dobrej współpracy między sprzętem i oprogramowaniem (zwłaszcza kompilatorem). Prostota koncepcji pozwoliła na zamknięcie całego kompletnego komputera w jednym układzie VLSI. Właśnie ten mikroukład jest podstawową częścią systemu PC/RT (czyli 6150), którego projekt firma IBM ukończyła w 1985 roku.

Naukowcy tej firmy są zgodni, że aby zwiększyć szybkość systemu komputerowego trzeba przyspieszyć działanie wszystkich jego składników: pamięci, układów wejścia-wyjścia itd. Ogólnie — bez względu na zastosowany rodzaj architektury — im bardziej rozwinie się technologię scalania do coraz mniejszych rozmiarów, tym szybsze będą komputery. Ale im mniejsze stają się układy scalone, tym bardziej ich elementy są narażone na mikrouszkodzenia spowodowane promieniowaniem. Trzeba będzie więc zająć się tworzeniem układów tolerujących takie uszkodzenia w swojej strukturze.

Opracowała
DOROTA INKIELMAN

Pascal na mikrokomputery IBM PC

Opracowanie stanowi przegląd czterech pakietów Pascala na mikrokomputery IBM PC: MS-Pascal firmy Microsoft, rozszerzona wersja UCSD Pascala firmy Pecan Software Systems, Pro Pascal firmy Prospero Software i Professional Pascal firmy Meta Ware. Informacje o Turbo Pascalu pojawiają się tylko w testach i tabelach porównawczych.

MS-Pascal

Kompilacja programu za pomocą kompilatora MS-Pascal wersji 3.31 odbywa się w trzech etapach. W programie źródłowym można zawrzeć dyrektywy kompilacji lub wpisać je w linii polecenia do systemu operacyjnego przy wykonywaniu pierwszego przebiegu kompilatora. System nie ma wbudowanego edytora. Drugi i trzeci przebieg jest wywoływany z DOS-a bez żadnych parametrów i wykorzystuje pliki tworzone w poprzednich przebiegach. Proces kompilacji i konsolidacji programu może być wywoływany z pliku wsadowego.

Aktualne wersje kompilatorów języka C, Fortranu i Pascala firmy Microsoft umożliwiają wzajemne łączenie plików kodu wynikowego. Konsolidator zawiera zestaw dyrektyw wpływających na takie parametry konsolidacji i generacji kodu wynikowego jak: przydział pamięci, nakładkowanie i użycie biblioteki domyślnej. Jeżeli konsoliduje się programy zawierające znaczną ilość działań zmiennoprzecinkowych, to można konsekwentnie korzystać z kilku bibliotek zmiennoprzecinkowych, np. według normy IEEE lub reprezentacji alternatywnej (tabela 1).

MS-Pascal ma standardowe typy danych: **boolean, char, integer, real.** Ma też pięć podtypów całkowitoliczbowych: krótki — **short integer** (od —127 do 127), bajtowy — **byte** (od 0 do 255), standardowy — **standard** (od —32 768 do 32 767), bezznakowy — **word** (od 0 do 65 535) i długi — **integer4** (od minus do plus 2 bilionów). Typy zmiennoprzecinkowe obejmują standardowy typ **real,** pojedynczej i podwójnej precyzji — **real4** i **real8.**

Tablice jedno- i wielowymiarowe typu **super array** nie mają zdefiniowanej górnej granicy indeksu. Przy deklarowaniu zmiennej w sekcji **var** trzeba podać górną granicę indeksu tablicy. Wskaźniki typów **super array** umożliwiają przesunięcie deklaracji wymiaru do czasu wywołania procedury **new( ).** Procedura przydziela pamięć strukturom dynamicznym i powinna określać górną granicę indeksu tablicy. Dostęp do granicznych indeksów tablicy uzyskuje się za pomocą pierwotnie zdefiniowanych funkcji **upper( )** i **lower( ).**

Są dwa typy napisów: typu **string** o zmiennej długości od 1 do 32 KB (nie istnieje pojęcie długości domyślnej) i typu **lstring** zawierający nie więcej niż 256 znaków. Różnica między nimi wynika ze struktury wewnętrznej: **string** jest tablicą znaków, natomiast **lstring** w pierwszym polu zawiera informacje o długości.

Stałe liczbowe w MS-Pascalu mogą mieć dowolną podstawę, od 2 do 32. Można posługiwać się m.in. stałymi dwójkowymi, ósemkowymi, dziesiętnymi i szesnastkowymi. Stałe można również definiować za pomocą wyrażeń zawierających poprzednio deklarowane stałe, lub — jako rekordy (tabela 2).

W omawianym kompilatorze zaimplementowano również sekcję **value,** tj. część programu, w której inicjalizuje się definiowane zmienne z jednoczesnym przypisaniem każdej z nich jednego lub większej liczby atrybutów. Atrybuty informują kompilator, w jaki sposób manipulować zmiennymi, przechowywać je i określać ich status między kolejnymi wywołaniami procedury.

Oprócz standardowych instrukcji pętli **for..do, while, repeat..until,** kompilator rozpoznaje dwie dodatkowe instrukcje sterujące wykonaniem pętli. Instrukcja **break** powoduje wyjście z aktualnie wykonywanej pętli, a **cycle** pozwala przeskoczyć resztę instrukcji aktualnej iteracji i przejść do następnej iteracji pętli.

W MS-Pascalu istnieją instrukcje **if** i **case.** Instrukcja **case** jest rozszerzona o klauzulę **otherwise.** Wyrażenia warunkowe są obliczane sekwencyjnie. Testowane wyrażenie jest zapisane w postaci serii warunków **and then** lub **or else.** Jeśli warunek **and then** nie jest spełniony, to pozostałej części wyrażenia nie oblicza się. Podobnie prawdziwość warunku **or else** powoduje zakończenie dalszych obliczeń.

Funkcje i procedury mogą zawierać atrybuty sterujące kodem wytwarzanym przez kompilator. Są to:
forward — zapowiedź deklaracji,
extern i **public** — odwołania zewnętrzne,
origin — adresowanie podprogramu,
fortran — sprzęganie wielu języków,
interrupt — sprzężenie ze sprzętem,
pure — optymalizacja.

MS-Pascal zawiera również bogatą bibliotekę konwersji danych, funkcji matematycznych, manipulowania napisami, przydziału pamięci zmiennym dynamicznym, działań na poziomie systemowym, dostępu do systemu plików, obsługi stogu i procedur semaforowych. Nie ma natomiast procedur grafiki wysokiej rozdzielczości, sterowania ekranem, kursorem lub oknami.

Język umożliwia korzystanie z plików binarnych i plików ASCII. Dostęp do pliku może odbywać się w jednym z trzech trybów: sekwencyjnym, bezpośrednim (dostęp swobodny) lub terminalowym (podobny do sekwencyjnego, ale z użyciem drukarek i terminali).

MS-Pascal udostępnia środki do modularnego tworzenia programu w postaci: modułów (tj. programów nie zawierających ciała, tylko deklaracje funkcji i procedur) i jednostek programowych (tj. bardziej rozwiniętych bibliotek złożonych z sekcji sprzęgających i implementacyjnych). W sekcji sprzężenia oprócz deklaracji procedur eksportowanych przez jednostkę, znajdują się stałe, typy, zmienne, nagłówki procedur i funkcji oraz opcjonalnie — inicjalizacja jednostki. Sekcja implementacyjna zawiera kompletną definicję procedury i definicje procedur lokalnych (p. tabela 2).

UCSD Pascal

Wersja 4.21 UCSD Pascala, z własnym środowiskiem i systemem ope-

Tabela 1. Ogólne informacje o omawianych kompilatorach i Turbo Pascalu

	MS-Pascal	UCSD Pascal	Pro Pascal	Profes Pascal	Turbo Pascal
Wersja	3,31	4,21	2,14	2,5	3,0
Kod wrodzony	Tak	Tak	Tak	Tak	Tak
p-kod	Nie	Tak	Nie	Nie	Nie
Liczba przebiegów	3	1	3	1	1
Konsolidator	Tak	Tak	Tak	Nie	Nie
Asembler	Nie	Tak	Nie	Nie	Nie
Program uruchomieniowy	Nie	Tak	Tak	Tak	Nie
Wbudowauy edytor	Nie	Tak	Nie	Nie	Tak
Edytor składniowy	—	Nie	—	—	Nie
Wymagany dysk stały	Nie	Nie	Nie	Tak	Nie
Cena (w dolarach)	300	99,95	390	595	69,95

racyjnym p-System, stanowi, nadbudowę systemu operacyjnego PC-DOS. Plik PSYSTEM.COM służy do ładowania wstępnego p-Systemu. Środowisko składa się z wbudowanego edytora, kompilatora, konsolidatora, programu obsługi plików DOS-a, asemblera, programu uruchomieniowego i monitora.

Edytor o nazwie Edvance jest jednoplikowym, typowym edytorem wbudowanym. Poza możliwościami przesuwania kursora, modyfikowania i formatowania tekstu, ma dwa polecenia obsługi bufora: zwiększenia jego wielkości i zapisywania części na pliku dyskowym. Rejestrując ciągi znaków można również tworzyć nowe funkcje lub makrodefinicje. Co więcej, program może czytać takie funkcje z tekstu. Dozwolone jest również czy program źródłowy na p-kod (opcja redagowanie zagnieżdżone.

Kompilator UCSD Pascala tłumadomyślna) lub kod wrodzony (wymagana jest wtedy dyrektywa kompilatora). Program źródłowy może zawierać szereg dyrektyw zezwalających na podjęcie takich działań, jak: kompilacja warunkowa, sprawdzanie błędów we-wy, sprawdzanie zakresu indeksów, określenie precyzji reprezentacji liczb zmiennoprzecinkowych lub tzw. wyciszona kompilacja — bez wyświetlania informacji o jej przebiegu. Inne wersje kompilatora wytwarzają kod wykorzystujący koprocesor 8087 i bibliotekę BCD. Po wykryciu błędu przez kompilator użytkownik może wznowić kompilację w celu wykrycia innych błędów, zatrzymać kompilator, wywołać edytor wskazujący krytyczwykonywalny, nie trzeba wywoływać komunikaty o błędach.

Konsolidator UCSD Pascala pozwala dołączać procedury w języku asemblera. Można również łączyć procedury w kodzie wrodzonym. Kiedy kompiluje się program źródłowy na kod wykonywalny, nie trzeba wywoływan konsolidatora explicite; kompilator wywoła go automatycznie (p. tabela 1).

W UCSD Pascalu zaimplementowano standardowe typy skalarne (logiczny, znakowy, całkowitoliczbowy i rzeczywisty). Dostępny jest też długi typ całkowitoliczbowy o definicji **integer[n]**, gdzie **n** jest liczbą całkowitą bez znaku reprezentującą liczbę cyfr — nie więcej niż 36. Rzeczywiste typy danych mogą być pojedynczej lub podwójnej precyzji, w zależności od użytej dyrektywy kompilatora. USCD Pascal ma również pierwotnie zdefiniowany typ napisowy **string** o określonej długości domyślnej i maksymalnej — odpowiednio — 80 i 255 znaków (tabela 2).

Procedur i funkcji można używać z wyprzedzeniem, posługując się atrybutem **forward**. Parametry są przekazywane przez wartość lub odwołanie. Język dopuszcza stosowanie tablic jedno- i wielowymiarowych. Górny i dolny indeks każdego wymiaru tablicy jest przypisany do zmiennych, których typ podaje się po określeniu zakresu:

array [dół..góra; typ] of typ.

W obrębie funkcji i procedur dostęp do granicznych indeksów tablicy uzyskuje się za pomocą zmiennych **upper()** i **lower()**. Zaimplementowano również pętle i instrukcje decyzyjne, ale bez rozszerzeń.

UCSD Pascal umożliwia używanie procesów pseudo-współbieżnych (jednoprocesorowe). Procesy muszą być globalne w programie i nie mogą być deklarowane w innych procedurach i funkcjach; niedozwolone są procesy zagnieżdżone. Procedura **start** inicjalizuje wywołanie każdego procesu współbieżnego. Każdy uruchomiony proces musi mieć identyfikator **processid**, priorytet i przypisaną wielkość stosu. Synchronizacja procesów odbywa się przez semafory pierwotnie zdefiniowanego typu i pewną liczbę pierwotnie zdefiniowanych procedur. Semafor monitoruje liczbę procesów oczekujących w kolejce i liczbę procedur. Procedura **seminit** inicjalizuje semafor i liczbę procesów. Procedury **signal** i **wait** wykorzystuje się przy synchronizacji procesów, argumentem obydwu jest **semaphore**. Procesy współbieżne mogą za pomocą procedury **attach** kojarzyć przerwania sprzętowe z semaforem.

Kompilator pozwala wykonywać operacje we-wy w sposób sekwencyjny lub swobodny (w formacie binarnym), ale są dopuszczalne tylko pliki o strukturze sekwencyjnej — ASCII. Możliwa jest obsługa niesformatowanych plików we-wy i wykonywanie blokowych operacji **blockread** i **blockwrite**.

Tabela 2. Porównanie konstrukcji programowych występujących w omawianych odmianach Pascala

	MS-Pascal	UCSD Pascal	Pro Pascal	Profess. Pascal	Turbo Pascal
Stałe dwójkowe	Tak	Nie	Nie	Tak	Nie
Stałe ósemkowe	Tak	Nie	Nie	Tak	Nie
Stałe szesnastkowe	Tak	Nie	Tak	Tak	Tak
Stałe strukturalne	Tak	Nie	Nie	Nie	Tak
Krótkie całkowite (od —127 do 127)	Tak	Nie	Nie	Nie	Nie
Bajtowe (od 0 do 255)	Tak	Nie	Nie	Nie	Tak
Całkowite bezznakowe	Tak	Nie	Nie	Tak	Nie
Całkowite długie	Tak	Tak	Tak	Tak	Nie
Adresy	Tak	Nie	Tak[1])	Tak[1])	Nie
Zmienne absolutne	Tak	Nie	Nie	Tak	Tak
Rzeczywiste podwójnej precyzji	Tak	Tak	Tak	Tak	Nie
Koprocesor 8087	Tak	Tak	Tak	Tak	Tak[2])
Liczby BCD	Nie	Tak	Nie	Nie	Tak[3])
Rzeczywiste rozszerzone	Nie	Nie	Nie	Tak	Nie
Typy wyliczeniowe	Tak	Tak	Tak	Tak	Tak
Typy okrojone	Tak	Tak	Tak	Tak	Tak
Otwarte tablice	Tak	Tak	Nie	Nie[3])	Tak[4])
Tablice uzgodnione	Tak	Nie	Nie	Nie	Nie
Typy proceduralne	Nie	Nie	Nie	Tak	Nie
Funkcje udostępniające struktury i tablice	Nie	Nie	Nie	Tak	Nie
Maks. długość napisu	32 K	255	255	64 K	255
Operacje na bitach i bajtach	Tak	Nie	Nie	Tak	Tak
case z klauzulą **else**	Tak	Nie	Tak	Tak	Tak
Procesy współbieżne	Nie	Tak	Nie	Nie	Nie
Wywołania DOS	Tak	Nie	Tak	Tak[5])	Tak
Grafika wysokiej rozdzielczości	Nie	Tak	Nie	Nie	Tak
Iteratory i rozszerzone pętle **for**	Nie	Nie	Nie	Tak	Nie
Wplatanie kodu maszynowego	Nie	Nie	Nie	Nie	Nie
Przerwania	Tak	Nie	Nie	Tak	Tak
Pętle etykietowane	Nie	Nie	Nie	Tak	Nie
Makro-preprocesor	Nie	Nie	Nie	Tak	Nie
Parametry proceduralne i funkcyjne	Tak	Tak	Tak	Tak	Nie
Sterowanie obrazem i kursorem	Nie	Tak	Nie	Nie	Tak
Operacje na napisach	Tak	Tak	Tak	Tak[5])	Tak
Operatory definiowane przez użytkownika	Nie	Nie	Nie	Tak	Nie
Ograniczenie 64 KB na program i dane	Nie	Nie	Nie	Nie	Tak
Wywołania łańcuchowe	Nie	Tak	Tak	Nie	Tak
Eksportowanie abstrakcyjnych typów danych	Nie	Nie	Nie	Tak	Nie
Moduły	Tak	Nie	Nie	Tak	Nie
Procedury zewnętrzne	Tak	Tak	Tak	Tak	Tak
Włączanie plików (include)	Tak	Tak	Tak	Tak	Tak
Nakładki	Tak	Nie	Tak	Tak	Tak
Segmentacja	Nie	Tak	Tak	Tak	Nie
Jednostki biblioteczne	Tak	Tak	Nie	Nie	Nie

Przypisy:

[1]) Jako funkcje.

[2]) Opcja dostępna w specjalnej wersji.

[3]) Właściwość dostępna przez wskaźniki.

[4]) Ograniczone do napisów.

[5]) Importowane z dostępnych bibliotek zewnętrznych.

UCSD Pascal zawiera pięć jednostek programowych wykonujących inne ważne zadania, takie jak: złożone sterowanie obrazem, przeadresowywanie we-wy i łańcuchowe wykonywanie programów, grafika żółwia, dostęp do sprzętu IBM PC, plików i segmentów danych.

Pro Pascal

Z omawianych czterech pakietów, wersja 2.14 Pro Pascal najbardziej odpowiada Pascalowi według standardów ISO, mimo posiadanie pewnej liczby rozszerzeń. Główny pakiet zawiera kompilator, konsolidator, symboliczny program uruchomieniowy i program zarządzający biblioteką. Pakiet kompilatora zawiera dwa programy usługowe, generator odwołań zewnętrznych i program konfiguracyjny. Pro Pascal nie ma własnego edytora.

Przed dołączeniem wybranej biblioteki do kompilacji trzeba ją skopiować i nazwać PASLIB.OBJ. Jeśli trójprzebiegowy kompilator znajdzie bibliotekę PASLIB.OBJ, to nastąpi automatyczne wywołanie konsolidatora. W przeciwnym razie, trzeba explicite łączyć plik wynikowy z wybraną biblioteką. Można też przeprowadzać selektywną konsolidację tylko tych procedur, do których są odwołania w programie. Jeśli nie poda się nazwy pliku wynikowego, to konsolidator (kompilator) może przejść do trybu interakcyjnego. Oprócz pytania o nazwę pliku wynikowego, konsolidator zadaje kilka pytań dotyczących wielkości stosu, nakładek, utworzenia tablicy symboli, mapy procedur publicznych i zewnętrznych, mapy segmentu. Cztery dostępne biblioteki wykonawcze odpowiadają czterem możliwym trybom pracy: z małą lub dużą pamięcią, z koprocesorem 8087 lub bez niego (p. tabela 1).

Pro Pascal udostępnia następujące podstawowe typy danych: **boolean, char, integer, real, long real** i **string**. Liczby całkowite zaimplementowano jako długie o wartościach od minus do plus 2 bilionów. Typ **long real** dopuszcza wartości z przedziału ($1{,}1^{-308}$; $3{,}6^{308}$) i wykorzystuje szesnastocyfrową precyzję. Typ real ma precyzję siedmiocyfrową i zakres od $5{,}9^{-39}$ do $6{,}3^{38}$. Napisy mają długość dopuszczalną do 255 znaków. Pro Pascal wykorzystuje zwykłe procedury manipulowania napisami i umożliwia deklarowanie zmiennych w obszarze pamięci wspólnej **(common)** do wykorzystania w innych segmentach (p. tabela 2). Funkcje i procedury odpowiadają standardowi ISO, tym samym, ich parametry można przekazywać przez wartość lub odwołanie. Można również użyć parametrów w postaci funkcji i procedur. Kompilator rozpoznaje dyrektywy **forward** i **extern** oraz dodatkowe procedury i funkcje, m.in. do łańcuchowego wywołania programów, konsolidacji nakładek, PEEK i POKE. Zaimplementowane są standardowe pętle i instrukcje warunkowe. Instrukcja **case** zawiera klauzulę **otherwise**.

Specjalna biblioteka procedur zewnętrznych obsługuje porty we-wy. Wykonuje odwołania do systemu operacyjnego i obsługę błędów. Kompilator zawiera procedury wspomagające wywołanie przez program macierzysty (ang. parent) programu potomnego (ang. child). Podczas wykonania programu potomnego program macierzysty jest zatrzymywany i może być wznowiony po zakończeniu programu potomnego. Procesy potomne mogą być zagnieżdżone — istnieją procedury zakończenia tych procesów i zwalniania pamięci przez nie zajmowanej.

Tak jak inne implementacje Pascala, Pro Pascal obsługuje pliki ASCII oraz pliki binarne i umożliwia dostęp swobodny lub sekwencyjny do plików binarnych, a dostęp sekwencyjny — do plików tekstowych. Programy aplikacyjne mogą dołączać dane do końca istniejącego pliku sekwencyjnego. Inne procedury pozwalają przemianowywać i usuwać pliki, udostępniają

Professional Pascal

Professional Pascal, wersja 2.5, ma kilka interesujących rozszerzeń zapożyczonych z Ady i Moduli 2. Kompilator umożliwia obsługę koprocesora 8087 i zawiera liczne programy usługowe do wykonywania takich zadań, jak: we-wy z konsoli, zarządzanie stogiem, przerwania, sortowanie, manipulowanie napisami i usługi systemu operacyjnego. Pakiet nie zawiera edytora, a do jego pracy jest wymagany dysk stały.

Użytkownik może wybierać między pięcioma modelami kompilatora, w zależności od wielkości programów źródłowych. Dyrektywy kompilatora można wywołać bezpośrednio z poziomu DOS-a lub w formie zapożyczonych z Ady pragm (są to dyrektywy umieszczone w tekście źródłowym).

Professional Pascal nie ma własnego konsolidatora i wykorzystuje konsolidator firmy Microsoft, jednakże jest dostępny makro-preprocesor (p. tabela 1).

Omawiany pakiet zawiera wszystkie standardowe typy danych, jak również następujące typy skalarne: **cardinal**, czyli całkowitoliczbowy bezznakowy (inspirowany przez Modulę 2), **long integer**, długi rzeczywisty — **longreal**, i rzeczywisty rozszerzony — **extreal**. Tak więc, istnieją trzy typy zmiennoprzecinkowe. Ich precyzja zależy od maszyny, ale istnieje między nimi zależność:

prec (real) <= prec (longreal) <= <= prec (extreal)

Napisy mają dopuszczalną długość 65 535 znaków. Przy deklarowaniu zmiennych napisowych trzeba w nawiasach określić ich aktualny rozmiar. Nie ma takiego wymagania przy deklarowaniu parametrów typu napisowego w liście parametrów formalnych procedur i funkcji. Wynika z tego, że podprogramy mogą obsługiwać napisy o zmiennej długości.

Stałe liczbowe mogą być przedstawione z podstawą różną od dziesiętnej. Podstawa może przybierać wartości z zakresu od 2 do 16, i oprócz stałych dziesiętnych można posługiwać się dwójkowymi, ósemkowymi i szesnastkowymi. We wszystkich miejscach wystąpienia stałych, mogą pojawić się wyrażenia złożone ze stałych (p. tabela 2).

Zmienne inicjalizuje się w sekcji **value**, znanej z MS-Pascala. Z języka C zapożyczono możliwość formowania typu wyrażenia (ang. type casting). Przydaje się to przy sprawdzaniu wyników wyrażeń zawierających różne typy całkowitoliczbowe. Ścisłą typizację Pascala można ominąć za pomocą zewnętrznej pseudofunkcji przypisującej zmienne. Wyrażenia mogą zawierać operatory przypisania, co pozwala na wielokrotne przypisania między danymi zgodnych typów, np.:

x:= y := 1.23.

Na wskaźnikach i adresach można wykonywać dodawanie i odejmowanie, jak również działania logiczne, do czego służą operatory porównania. Blok deklaracji — pojęcie zapożyczone z Ady — pozwala deklarować zmienne w ciele programu lub procedury.

Professional Pascal ma standardowe pętle i dwie instrukcje sterujące wykonaniem pętli. Instrukcja **cycle**, używana w pętlach **for, while** i **repeat** powoduje pominięcie dalszej części pętli i przejście do następnej iteracji. Instrukcja **break** służy do całkowitego wyjścia z pętli. Instrukcjom pętli i bloku można nadawać nazwy. Nazwy umieszczone po instrukcji **break** wskazują miejsce wznowienia progra-

Tabela 3. Wyniki programu testowego — sito Eratostenesa. Wielkości kodu podano w bajtach (z wyjątkiem UCSD Pascala), a czas w sekundach

Kompilator	Wielkość kodu źródłowego	Wielkość kodu kompilowanego	Czas kompilacji i konsolidacji	Czas wykonania	Uwagi
MS-Pascal	768	29 600	54	11	
UCSD Pascal	4 bloki	2 bloki 3 bloki	19 50	129 18	p-kod wrodzony
Pro Pascal	768	11 008	42	29	
Professional Pascal	768	21 646	74	12	
Turbo Pascal	768	11 495	2,7	13	

mu. Jest to właściwość wygodna przy tworzeniu pętli zagnieżdżonych z pojedynczym wyjściem.

Instrukcja **case** jest podobna do wersji standardowej, z tą różnicą, że może zawierać klauzulę **otherwise.** W operacjach logicznych wykorzystuje się operator warunkowy **and then** w ten sposób, że argument po prawej stronie nie jest obliczany, jeśli argument po lewej stronie jest prawdziwy. Za pomocą tych operatorów można sprawdzać wyrażenia szczególnie podatne na błędy, np. wyrażenie:

i <> 0 and then j/i > 0

eliminuje problem dzielenia przez zero.

W kompilatorze zaimplementowano bardzo silną konstrukcję zapożyczoną z Ady: definiowane przez użytkownika operatory jedno- i dwuargumentowe. Symbol nowego operatora może być dowolnej długości. W języku Professional Pascal wprowadzono nowy rodzaj podprogramu nazwanego **iterator,** stanowiący wszechstronne rozwinięcie pętli.

Język zawiera typy procedurowe (podobnie jak Modula 2) i zezwala na funkcje dowolnego typu. Specjalna składnia pozwala rozróżnić pośrednie wartości funkcji, pojawiające się po prawej stronie instrukcji przypisania, od rekurencyjnych wywołań funkcji. Podczas wywołania podprogramu odbywa się kojarzenie parametrów według odpowiedniej kolejności lub za pomocą nazw parametrów formalnych (jak w Adzie). Procedury i funkcje mogą mieć tablice o zmiennej długości, ponieważ uzyskują dostęp do tablic za pomocą wskaźników.

Professional Pascal zawiera wszystkie operacje wewnętrzne standardowego Pascala, oprócz **pack** i **unpack,** jak również dodatkowe procedury Moduli 2: **increment, decrement, set-include, set-exclude.** Funkcje **maximum** i **minimum** mogą mieć więcej niż dwa argumenty. Najmniejszy i największy indeks tablicy udostępniają funkcje **lowest** i **highest.** Funkcje działają też na typach i zbiorach.

Kompilator rozpoznaje pliki ASCII i pliki binarne. Dostęp do plików binarnych może być swobodny lub sekwencyjny, a do plików ASCII tylko sekwencyjny. Procedura swobodnego dostępu do poszczególnych rekordów pozwala na numerowanie rekordów liczbami typu **long integer;** liczba dostępnych rekordów tym samym zwiększa się do 2 bilionów.

W Professional Pascalu zaimplementowano też znane z Ady jednostki biblioteczne zwane pakietami. Pakiety mogą eksportować do innych programów: stałe, definicje typów, zmienne, procedury i funkcje. Można również eksportować typy abstrakcyjne, których struktura jest niewidoczna w innych programach — cecha zapożyczona z Ady i Moduli 2. Specjalna składnia pozwala określić elementy eksportowane, a tym samym zlokalizować wszystkie pozostałe. Programy importujące, tak jak w Moduli 2, mogą selekcjonować te elementy.

Tabela 4. Wyniki testów zmiennoprzecinkowych. Wielkości kodu podano w bajtach (z wyjątkiem UCSD Pascala), a czas w sekundach

Kompilator	Wielkość kodu źródłowego	Wielkość kodu kompilowanego	Czas kompilacji i konsolidacji	Czas wykonania	Uwagi
MS-Pascal z 8087	384	36 894	64	4	PASCAL. LIB
		29 708	63	4	8087.LIB
		36 878	66	4	MATH. LIB
MS-Pascal bez 8087	384	31 946	65	12	ALTMATH. LIB
		29 954	65	10	DECMATH. LIB
UCSD Pascal	4 bloki	2 bloki	17	36	p-kod
		3 bloki	43	4	kod 8086/87
Pro Pascal	384	5 120	43	4	z 8087
		5 888	47	10	bez 8087
Professional Pascal	384	16 782	75	2,5	z 8087
		21 566	86	18	bez 8087
Turbo Pascal	384	10 334	2,0	7	z 8087
		11 485	2,6	33	bez 8087

Testy

Dla wszystkich czterech kompilatorów wykonano standardowe testy zmiennoprzecinkowe i sito Eratostenesa. Badania przeprowadzono na mikrokomputerze IBM PC/XT z dyskiem stałym 10 MB, 512 KB pamięci i koprocesorem 8087, pod kontrolą systemu operacyjnego PC-DOS 3.1. Kompilację i konsolidację przeprowadzono z pliku wsadowego (z wyjątkiem Turbo Pascala). Zmierzone czasy kompilacji i wykonania we wszystkich przypadkach dotyczą pracy z dyskiem stałym, jedynie programy w UCSD Pascalu uruchamiano z dysków elastycznych. Wszystkie komiplatory tłumaczyły programy testowe bez błędów (w tym celu nieznacznie modyfikowano tekst źródłowy).

Kompilator Pro Pascala utworzył pełen program o najmniejszej wielkości, na drugim miejscu był Turbo Pascal, a największy program utworzył kompilator MS-Pascala. UCSD Pascalowi odpowiadał najkrótszy program, co można przypisać obecności procedur we-wy w jądrze systemu UCSD.

Pod względem szybkości kompilacji i konsolidacji Turbo Pascal okazał się najlepszy, nieznacznie wyprzedzając UCSD Pascal, mimo że ten był uruchamiany z dyskietek. Czasy wykonania wskazują na mocne i słabe strony sprawdzanych kompilatorów (p. tabela 4).

Spośród omawianych kompilatorów nie można wybrać zdecydowanie najlepszego. Każdy ma swoje zalety i wady. Wydaje się, że każdy wybór — szczególnie gdy chodzi o komputery — zależy od potrzeb i umiejętności użytkownika.

MC-Pascal jest odpowiedni do tworzenia dużego oprogramowania i dla programistów o średnich i wysokich umiejętnościach. UCSD Pascal ma szybkość wykonania na praktycznym i pożądanym poziomie. Podobieństwo Pro Pascala do standardu ISO przesądza o jego przydatności do celów edukacyjnych. Professional Pascal jest adresowany do miłośników Pascala, wzbogaconego o interesujące cechy innych języków; ponadto jest odpowiedni do pisania dużych programów. Turbo Pascal, mimo ograniczenia rozmiaru kodu do 64 KB, jest atrakcyjny dla wszystkich programistów, niezależnie od umiejętności.

Tabele porównawcze 1—4 powinny pomóc w ocenie możliwości poszczególnych kompilatorów i w wyborze tego, który najlepiej spełnia wymagania.

Oprac. M. KUC

na podst. BYTE, December 1986

Nowy rekord obliczeniowy

Tych, którzy pragną poznać „dokładną" wartość liczby Π z pewnością ucieszy fakt, że wyliczono ją już i zweryfikowano z dokładnością do 133 554 414 cyfr znaczących. Poprzedni rekord należał do Davida H. Bailey z NASA, który wyznaczył 29 360 000 cyfr znaczących na superkomputerze Cray-2. W lutym 1987 roku, prof. Yasumasa Kanada z Ośrodka Obliczeniowego Uniwersytetu Tokijskiego ustanowił nowy rekord na superkomputerze NEC SX-2 zainstalowanym w zakładach firmy NEC w Fuchu. Obliczenia trwały ponad 45 godzin, a wydrukowanie wyniku pochłonęło 19 000 arkuszy papieru. **(MM)**

Projekt komputerów piątej generacji

W wielu publikacjach porównuje się wpływ japońskiego projektu komputerów piątej generacji (KPG) na światowy przemysł komputerowy do roli, jaką w rozwoju techniki odegrało wystrzelenie pierwszego sztucznego satelity Ziemi. Celem tego projektu jest opracowanie komputera o zupełnie nowej organizacji, zrywającej z tradycją maszyn opartych na modelu Johna von Neumanna. Ma on dokonać przewrotu w technologii przetwarzania informacji, polegającego na przejściu od tradycyjnego przetwarzania danych do przetwarzania wiedzy. W praktyce oznacza to wykorzystanie w procesie obliczeniowym, m.in. wielu metod i narzędzi sztucznej inteligencji. Kluczową rolę w tym projekcie odgrywa programowanie logiczne.

Dane do opracowania artykułu zaczerpnięto głównie z prac [1], [12] i [13].

ORGANIZACJA PROJEKTU JAPOŃSKIEGO

W 1979 r. organizacja JIPDEC (ang. Japan Information Processing Development Association) zwróciła się do profesora Tohru Moto-oki z propozycją współpracy w projekcie KPG. Było to następstwem zlecenia ministerstwa MITI (ang. Ministry of International Trade and Industry), którego celem miało być podsumowanie dotychczasowych osiągnięć nauki, interesujących z punktu widzenia planowanego projektu. W rezultacie powołano komitet przeglądowy KPG, którego przewodniczącym został T. Moto-oka.

Utworzono trzy grupy tematyczne. Zadaniem pierwszej było zbadanie jakiego rodzaju komputerów będzie potrzebowało społeczeństwo lat dziewięćdziesiątych. Kierownictwo tej grupy objął Hajime Karatsu z Matsushita Tsushin Company. Druga grupa skoncentrowała się na architekturze przyszłych komputerów. Badaniami tymi kierował Hideo Aiso z Uniwersytetu Keio. Zadaniem trzeciej grupy — kierowanej przez profesora Kazuhiro Fuchi — było zbadanie teoretycznych podstaw projektu. W ciągu dwóch lat od daty utworzenia komitetu, w jego pracach brało udział ponad 100 naukowców. W październiku 1981 r., na międzynarodowej konferencji w Tokio, po raz pierwszy poinformowano świat o projekcie.

Oficjalnie prace rozpoczęto 1 kwietnia 1982 r., z chwilą utworzenia instytutu ICOT (The Institute for New Generation Computer Technology) — organizacji odpowiedzialnej za realizację projektu KPG. Głównymi członkami ICOT są: Fujitsu, Hitachi, Matsushita, Mitsubishi, NEC, Oki, Sharp i Toshiba. Dyrektorem ICOT mianowano K. Fuchi. Projekt zaplanowano na 10 lat, z budżetem około 855 mln dolarów. Całe przedsięwzięcie podzielono na trzy fazy.

- Faza wstępna (lata 1982—1985) objęła gromadzenie i ocenę rezultatów badań w dziedzinie przetwarzania informacji i wiedzy. Przewidziano również zaprojektowanie narzędzi do zastosowania w przyszłych badaniach. Naukowców wyposażono, m.in. w komputery PSI do prowadzenia wstępnych prac badawczych.
- Faza pośrednia (lata 1985—1989) obejmuje przetestowanie dwóch podstawowych podsystemów KPG — baz wiedzy oraz procesorów wnioskujących.
- Faza końcowa (lata 1989—1992) będzie miała na celu skonstruowanie prototypu komputera piątej generacji, który — jak się planuje — ma być przekazany do produkcji na skalę przemysłową.

INNE PROGRAMY BADAWCZE

Po ogłoszeniu założeń projektu KPG szybko uświadomiono sobie jego znaczenie i przełom, jaki za jego sprawą może dokonać się w informatyce. Dlatego w wielu krajach opracowano analogiczne programy badań.

Odpowiedzią USA na japońskie wyzwanie są dwa programy badawcze. Pierwszy z nich, koordynowany przez agencję DARPA (ang. Defense Advanced Research Project Agency), rozpoczęto w 1984 r. Program zaplanowano na 10 lat, z funduszem około 600 mln dolarów na pierwszych 5 lat. Drugą amerykańską inicjatywą związaną z KPG jest założone przez Williama Norrisa z CDC konsorcjum ponad 18 firm elektronicznych o nazwie MCC (Microelectronic and Computer Technology Corporation). Roczny budżet MCC wynosi 50 mln dolarów. Każda z firm przystępujących do MCC wpłaca 100 tys. dolarów, uzyskując w zamian dostęp do wyników badań i prototypowych projektów. W skład konsorcjum wchodzą, m.in.: Motorola, National Cash Register, National Semiconductor. IBM nie jest członkiem MCC, ale z pewnością poświęca na ten cel część swego kilkumiliardowego budżetu przeznaczonego co roku na cele badawczo-rozwojowe.

W 1982 r. na konferencji w Wersalu przedstawiciele Europejskiej Wspólnoty Gospodarczej podjęli inicjatywę utworzenia wspólnego programu badań i rozwoju zaawansowanej technologii informacyjnej. W wyniku powstał program o nazwie ESPRIT (ang. European Strategic Programme for Research in Information Technology). Badania zaplanowano na 10 lat, z budżetem około 450 mln funtów. Program badań jest w tym wypadku szerszy niż w ramach ICOT i obejmuje m.in. zagadnienia automatyzacji prac biurowych.

Odpowiedzią Wielkiej Brytanii na japoński projekt KPG było ustanowienie komitetu pod przewodnictwem Johna Alveya. Komitet ten opublikował w 1982 r. raport proponujący pięcioletni narodowy program badawczy w dziedzinie zaawansowanych technologii informacyjnych, z budżetem około 350 mln funtów. W 1983 r. rząd brytyjski zaakceptował ten plan prawie bez zastrzeżeń. Określono cztery podstawowe obszary badawcze: inżynieria oprogramowania, technologia VLSI, inteligentne systemy z bazą wiedzy oraz inteligentna komunikacja z użytkownikiem.

OPROGRAMOWANIE PODSTAWOWE KOMPUTERÓW PIĄTEJ GENERACJI

Celem japońskiego projektu KPG jest stworzenie superkomputera do przetwarzania wiedzy o całkowicie równoległej architekturze. W tym celu opracowuje się oprogramowanie podstawowe, które będzie składać się prawdopodobnie z następujących elementów [6]:

- język maszynowy wysokiego poziomu (ang. kernel language) oraz język programowania wiedzy,
- moduł rozwiązywania problemów i wnioskowania,
- system zarządzania bazą wiedzy,
- moduł inteligentnej komunikacji,
- moduł inteligentnego programowania.

Wymienione elementy wyznaczają jednocześnie główne obszary badawcze projektu.

Podstawą do opracowania języka maszynowego oraz języka programowania wiedzy jest programowanie logiczne [9, 10], które ma stanowić pomost między przetwarzaniem wiedzy a całkowicie równoległą architekturą komputera, zrealizowaną przy użyciu technologii VLSI.

Moduły rozwiązywania problemów i wnioskowania oraz zarządzania bazą wiedzy są podstawowymi elementami systemu przetwarzania wiedzy. Zakłada się opracowanie efektywnych mechanizmów wnioskowania, m.in. realizację funkcji wnioskowania współbieżnego i metawnioskowania. Celem wprowadzenia funkcji wnioskowania współbieżnego jest znaczne przyspieszenie procesu obliczeniowego, a także umożliwienie „rozproszonego" rozwiązywania problemów.

Jednym ze sposobów sterowania wnioskowaniem jest zastosowanie, oprócz reguł logicznych poziomu podstawowego, reguł bardziej ogólnych, informujących o sposobie korzystania z reguł poziomu podstawowego. Takie rozwiązanie jest podstawą metawnioskowania. Przykładem implementacji metawnioskowania jest znany w programowaniu logicznym predykat Demo [10]. Jego zadaniem jest pokazanie, że dany cel może być dowiedziony na podstawie określonego zbioru klauzul. Predykat ten dotyczy procesów sekwencyjnych, dlatego w projekcie opracowano predykat o nazwie Simulate, pełniący tę samą funkcję w warunkach współbieżności.

Opracowanie systemu zarządzania bazą wiedzy poprzedzono analizą następujących zagadnień:
- realizacja bardzo dużych baz wiedzy,
- akwizycja wiedzy w języku logiki,
- reprezentacja wiedzy,
- tworzenie systemów ekspertowych.

Pierwszym krokiem w kierunku opracowania bardzo dużej bazy wiedzy jest połączenie systemu dedukcyjnego dla klauzul Horna z relacyjną bazą danych, zarówno na poziomie sprzętu, jak i oprogramowania. Połączono komputer prologowy PSI z bardzo dużą relacyjną bazą danych Delta, za pomocą lokalnej sieci o nazwie INI. Zaprojektowano jednocześnie eksperymentalny system zarządzania bazą wiedzy o nazwie KAISER (ang. Knowledge Acquisition-Oriented Information Supplier). Jedną z jego głównych funkcji jest wspomaganie akwizycji wiedzy.

Celem badań w dziedzinie inteligentnej komunikacji jest opracowanie elastycznych metod porozumiewania się człowieka z komputerem [6]. Wynika to z przyjęcia założenia, że KPG będą służyć wielu ludziom, z których większość nie będzie informatykami. Dąży się więc do osiągnięcia tzw. zgodności poznawczej (ang. cognitive compatibility) [1]. Jest to cel znacznie trudniejszy do osiągnięcia, niż cecha określana jako „user-friendliness" w wypadku tradycyjnych komputerów. Aby go osiągnąć uwzględniono w badaniach, m.in. takie zagadnienia jak: techniki przetwarzania języka naturalnego, techniki analizy językowej oraz badanie aspektów inteligentnej komunikacji między człowiekiem a komputerem [8]. Ponadto rozwija się metody rozpoznawania oraz syntezy mowy. Prowadzi się również prace dotyczące przetwarzania obrazów, których celem jest m.in. zbudowanie języka oraz maszyny do przetwarzania obrazów.

W pracach dotyczących inteligentnego systemu programowania uwzględnia się trzy obszary badawcze:
- specyfikacja programów,
- weryfikacja programów,
- transformacja programów.

Ostatecznym celem tych badań jest opracowanie systemu, który automatycznie przekształca dany problem na efektywny program komputerowy, wyrażony w języku maszynowym [4]. W początkowej fazie projektu rozpoczęto realizację systemu do programowania modularnego oraz weryfikacji programów. Jego zadaniem jest zwiększenie efektywności programowania na każdym z etapów tworzenia programu, tj. na etapie projektowania, kodowania, uruchamiania, testowania i poprawiania. System zaprojektowano, jako zestaw specjalizowanych systemów ekspertowych, którymi zarządza nadrzędny system ekspertowy o nazwie Coordinator.

JĘZYKI PROGRAMOWANIA KOMPUTERÓW PIĄTEJ GENERACJI

Centralne miejsce w projekcie KPG zajmuje programowanie logiczne. Wybór programowania logicznego spotkał się z krytyką ze strony niektórych badaczy (np.: [2], [4], [7] i [15]). Krytyka ta częściowo wynika z właściwości obecnych implementacji Prologu, które nie są pozbawione pewnych słabości i ograniczeń. Twórcy projektu podkreślają jednak, że obecna postać Prologu stanowiła jedynie punkt wyjścia do opracowania bardziej zaawansowanych języków.

Integracja poszczególnych podsystemów w KPG, ma być osiągnięta na bazie programowania logicznego. W tym celu zaplanowano opracowanie serii języków maszynowych: KL0, KL1 oraz KL2, które z jednej strony mają stworzyć element wiążący poszczególne podsystemy, z drugiej zaś stanowić pomost między sprzętem a resztą oprogramowania.

W pierwszej kolejności zdefiniowano język KL0 przeznaczony do bezpośredniej realizacji na sekwencyjnej maszynie wnioskującej o nazwie PSI (ang. Personal Sequential Inference). KL0 można uważać za rozszerzoną wersję Prologu. Język ten jest wyposażony m.in. w mechanizmy przenoszenia sterowania wykorzystujące wielopoziomowy operator obcięcia, a także specjalne operacje niskiego poziomu, takie jak: działania na rejestrach i współpraca z urządzeniami wejścia—wyjścia. Ponieważ KL0 jest językiem maszynowym, opracowano również język użytkownika o nazwie ESP (ang. Extended Self-contained Prolog), przeznaczony m.in. do programowania systemowego. Język ESP jest oparty na Prologu i zawiera dość istotne rozszerzenia o elementy programowania obiektowego (ang. object-oriented programming). ESP wykorzystuje mechanizmy modularyzacji oraz tworzenia hierarchii, niezbędne do realizacji dużych programów (np. systemów operacyjnych). ESP jest wyposażony w mechanizm tworzenia makrofunkcji, a więc w pewnym sensie może być uważany za makroasembler języka KL0. Jego makrofunkcje umożliwiają zapis w notacji typu funkcyjnego, jak również obiektowego [4]. Program napisany w języku ESP jest przed wykonaniem tłumaczony na język KL0.

KL0 zaprojektowano bezpośrednio po rozpoczęciu realizacji projektu i w konsekwencji zawiera on w sobie pewne ograniczenia związane z przetwarzaniem sekwencyjnym. Z tego względu obecnie opracowuje się język KL1. W odróżnieniu od KL0 jest on wyposażony w środki programowania współbieżnego oraz mechanizmy metawnioskowania. Programy logiczne mogą korzystać z dwóch rodzajów współbieżności: koniunkcyjnej (ang. and-parallelism) i alternatywnej (ang. or-parallelism). Pierwsza z nich wynika z faktu, że wywołania w zapytaniu (klauzuli-celu) mogą być uaktywnione i rozwiązywane równolegle. Drugi rodzaj współbieżności opiera się na założeniu, że gdy kilka procedur odpowiada na aktywne wywołanie, to mogą one być wywołane i użyte równolegle. W języku KL1 wszystkie rozwiązania są uzyskane najpierw w podsystemie współbieżności alternatywnej, a następnie zbiór rozwiązań jest przetwarzany jako strumień danych w podsystemie współbieżności koniunkcyjnej [6].

Metaprogramowanie logiczne, ujmując rzecz skrótowo, polega na:
- przetwarzaniu programu, tak jak danych,
- przetwarzaniu danych, tak jak programu,
- traktowaniu rezultatu procesu obliczeniowego jak danych.

Słabością tej techniki programowania jest to, że metaprogramy są wykonywane wolniej niż programy poziomu podstawowego. Metaprogramowanie ma jednak duże znaczenie dla zwiększenia elastyczności mechanizmów wnioskowania. Interpretery Prologu stosują bowiem sztywną strategię sterowania w procesie wnioskowania, co jest przedmiotem krytyki niektórych specjalistów zajmujących się sztuczną inteligencją. W rezultacie zastosowania metaprogramowania uzyskuje się parametryzację metod sterowania, a więc większą elastyczność mechanizmów wnioskowania [6].

Wstępną definicję języka KL1 opublikowano w 1983 r. W jego opracowaniu wykorzystano m.in. wyniki doświadczeń z językami Parlog, Concurrent Prolog oraz Relational Language. Z KL1 jest związany język użytkownika Mandala, przeznaczony do programowania wiedzy. Język Mandala został już zrealizowany, chociaż w pewnym momencie jego rozwój uległ zatrzymaniu, ze względu na nieefektywność implementacji. W celu przezwyciężenia tej trudności, opracowano z powodzeniem technikę optymalizacji, polegającą na transformacji programów przez częściową ewaluację (ang. partial evaluation) [5]. W dalszej fazie projektu przewiduje się integrację modułów (rozwiązującego problemy, wnioskującego i zarządzania bazą wiedzy) za pomocą języka Mandala.

PROTOTYPY KOMPUTERÓW

W końcu 1983 r. opracowano pierwszą wersję komputera o nazwie PSI, przeznaczonego do bezpośredniej realizacji programów w języku KL0. PSI wykonuje obliczenia z szybkością 30 KLIPS (ang. logical inference per second), a jego pamięć operacyjna ma pojemność 16 megasłów (80 MB) [5]. Do budowy komputera użyto szybkich układów TTL. Równolegle z opracowaniem sprzętu, opracowano system operacyjny o nazwie SIMPOS, w całości napisany w języku ESP. Niektórzy specjaliści wyrażali wątpliwość, czy języki programowania logicznego nadają się do pisania tak dużych programów jak systemy operacyjne. SIMPOS jest odpowiedzią na te wątpliwości. Tekst źródłowy systemu liczy około 145 tys. wierszy [5]. Liczba klas (w sensie programowania obiektowego) wynosi 960, a liczba użytych predykatów — około 15 tys. Jak zauważa K. Fuchi, język ESP potwierdził swoją skuteczność jako narzędzie programowania.

Obecnie opracowuje się nową wersję PSI, nazywaną PSI-II. Zakończenie budowy zaplanowano na wiosnę 1987 r. Jak się przewiduje, szybkość wniosko-

wania PSI-II wyniesie około 100 KLIPS, a pojemność pamięci wzrośnie do 64 megasłów. Konstrukcję oparto na układach CMOS Gate Array LSI. Jednocześnie trwają prace nad systemem wielu komputerów PSI połączonych ze sobą za pomocą bardzo szybkiej sieci. Sprzętową część tego systemu o nazwie multi-PSI/V1, już ukończono. System ten ma być przede wszystkim narzędziem do opracowania systemu operacyjnego PIMOS dla maszyn PIM (ang. Parallel Inference Machine). Planuje się również budowę rozszerzonej wersji, nazywanej multi-PSI/V2, wykorzystującej około 20 maszyn typu PSI-II. Oczekuje się, że szybkość systemu wzrośnie w ten sposób ponad trzykrotnie w porównaniu z PSI [5].

Jednocześnie trwają badania nad konstrukcją komputera równoległego PIM. Początkowo przyjęto założenie, że docelowym językiem będzie język nazywany OR-Parallel Prolog. Prace koncentrowały się na konstrukcji trzech eksperymentalnych systemów opartych na organizacji maszyn sterowanych: przepływem danych (ang. data flow architecture), redukcji grafu (ang. graph reduction architecture) oraz organizacji określonej jako „Kabu-wake". Po opracowaniu nowego języka logicznego GHC (ang. Guarded Horn Clauses) [14] badaniami objęto implementację współbieżności zarówno koniunkcyjnej, jak i alternatywnej. Nowy kierunek badań obejmuje również realizację odpowiedniego oprogramowania podstawowego. Zakłada się, że konstrukcja PIM powinna być zakończona do końca etapu pośredniego, a więc do 1989 r. Przewiduje się, że szybkość wnioskowania PIM będzie rzędu kilku MLIPS.

Docelowa wersja sprzętu komputera piątej generacji ma osiągnąć szybkość wnioskowania od 100 MLIPS do 1 GLIPS. Dla porównania, współczesne komputery działają z szybkością od 10 do 100 KLIPS. Jednocześnie planuje się budowę bazy wiedzy tego systemu o pojemności od 100 GB do 1000 GB [8].

* * *

Powszechnie uważa się, że komputery piątej generacji odegrają olbrzymią rolę w takich dziedzinach działalności ludzkiej, jak: przemysł, edukacja, automatyzacja prac projektowych, a także w sektorze wojskowym. Jednym z najbardziej obiecujących zastosowań są systemy ekspertowe — w szczególności będzie możliwa budowa systemów ekspertowych z bardzo dużymi bazami wiedzy. KPG będą jednym z wiodących rodzajów komputerów końca dwudziestego wieku.

Warto przytoczyć opinię profesora E. Feigenbauma, jednego z najwybitniejszych specjalistów z dziedziny sztucznej inteligencji: „Droga do systemów piątej generacji jest najeżona wieloma trudnościami i nieuniknionymi niepowodzeniami, lecz innej drogi nie ma" [3].

KRZYSZTOF MICHALIK

LITERATURA

[1] Bishop P.: Fifth Generation Computers — Concepts, Implementations and Uses. Ellis Horwood, Chichester, 1986

[2] Bobrow D. G.: If Prolog is the Answer What is the Question? Proc. Int. Conf. on Fifth Generation Computer Systems, ICOT, Tokyo, 1984

[3] Feigenbaum E.: Innovation and Symbol Manipulation in Fifth Generation Computers. Proc. Int. Conf. of Fifth Generation Computer Systems. T. Moto-oka (Ed.), North-Holland, Amsterdam, 1982

[4] Fuchi K.: Revisiting Original Philosophy of Fifth Generation Computer Systems Project. Proc. Int. Conf. on Fifth Generation Computer Systems, ICOT, Tokyo, 1984

[5] Fuchi K., Furukawa K.: The Role of Logic Programming in the Fifth Generation Project. Proc. Third Int. Conf. on Logic Programming. E. Shapiro (Ed.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1986

[6] Furukawa K., Yoki T.: Basic Software System. Proc. Int. Conf. on Fifth Generation Computer Systems, ICOT, Tokyo, 1984

[7] Impact 1984: The Fifth Generation and Expert Systems Considered. Expert Systems, Vol. 1, No. 2, October 1984

[8] Kawanobe K.: Current Status and Future Plans of the Fifth Generation Computer Systems Project. Proc. Int. Conf. on Fifth Generation Computer Systems, ICOT, Tokyo, 1984

[9] Kowalski R. A.: Logic Programming in the Fifth Generation. Proc. Fifth Generation Conf., SPL International, London, 1982

[10] Kowalski R. A.: Logic for Problem Solving. North-Holland, Amsterdam, 1985

[11] Michalik K.: Programowanie logiczne, Informatyka, nr 5, 1986

[12] Moto-oka T., Kitsuregawa M.: The Fifth Generation Computer — The Japanese Challenge. Wiley, Chichester, 1985

[13] Simons G. L.: Expert Systems and Micros. MCC Publications, Manchester, 1985

[14] Ueda K.: Guarded Horn Clauses. Technical Report TR-103, ICOT, Tokyo, 1985

[15] Wagner C. C., Binkert P. J.: Hardware Based Semantics. Proc. Conf. on Intelligent Systems and Machines, 23—24 April 1985 Center for Robotics and Advanced Automation, Oakland University, Rochester, 1985.

Czy można lekceważyć niechlujstwo językowe?

Eskimosi mają ponoć z górą dwadzieścia słów na określenie różnych gatunków śniegu, nie używają natomiast rzeczownika abstrakcyjnego „śnieg". Warszawskim mieszczuchom wystarcza ten właśnie rzeczownik abstrakcyjny, w razie potrzeby modyfikowany przymiotnikami (puszysty, brudny, mokry, trzeszczący...). Ale już np. narciarze są dokładniejsi, puchu ze szrenią raczej nie mylą: inaczej się jeździ, to i inaczej nazywa to, po czym się jedzie. Naturalnie, aby się nie pomylić, trzeba wiedzieć. A żeby innych nie zmylić, trzeba używać właściwych słów.

Obserwując, jak się obecnie o informatyce mówi i pisze, dochodzę do wniosku, że ogromnie wielu mówiących i — niestety — sporo piszących albo nie wie, o co chodzi, albo nie panuje nad swoją mową. Anglicy, wśród których sposób wysławiania się jest oznaką pozycji społecznej (rzeczywistej lub udawanej, nieważne!), ważniejszą niż styl ubierania się, otóż ci Anglicy mają świetne, acz nieco brutalne określenie: *a case of verbal diarrhoea with mental constipation*, którym jednoznacznie klasyfikują mówcę lub pisarza nie panującego nad wypowiedzią. (W przybliżonym przekładzie: cierpi na biegunkę słów z obstrukcją myśli.)

Złapany na gorącym uczynku rodzimy gwałciciel polszczyzny tłumaczy się zazwyczaj, że wszyscy tak robią, albo że wszyscy i tak go pojmują. I — w pewnym stopniu — ma rację. Chwasty językowe plenią się jak salmonella latem, podobnie jak salmonella pozornie nie psują smaku potrawy. Ale szkodzą, oj, jak szkodzą.

Wielu osobników wypowiadających się (w mowie i piśmie) o informatyce czerpie natchnienie i wiedzę z tzw. materiałów źródłowych, którymi najczęściej są gazetki i ulotki reklamowe, nierzadko pisane w Hongkongu czy na Formozie (kłopoty, jakie mają ludzie Orientu z wysławianiem się w językach europejskich, są legendarne, ale osobnik o tym pewno nie wie, bo jest „specjalistą", czyli uważa, że wolno mu być nieukiem w każdej dziedzinie oprócz tej jednej — specjalizacji!). Materiały te wyglądają na pisane po angielsku. Osobnik zaś — oczywiście! — uważa, że mowę Byrona opanował doskonale.

W języku angielskim występuje czasownik przechodni *to control*. Z punktu widzenia polszczyzny jest on homonimem, oznaczającym: mieć władzę, panować, opanowywać, rządzić, zarządzać, sprawować nadzór, pohamować, sterować, kierować (czymś), regulować, warunkować (coś), a także sprawdzać, kontrolować (we wszystkich znanych mi słownikach angielsko-polskich polski odpowiednik najbardziej fonetycznie zbliżony do tego angielskiego słowa jest wymieniany na ostatnim miejscu). A w wypowiedziach specjalistów wszystko się „kontroluje". Oczywiście to nie tylko lenistwo, to także niewiedza! Trzeba się bowiem znać na rzeczy, by wiedzieć, kiedy *control character* należy przełożyć na *znak kontrolny*, a kiedy na znak *sterujący!* Gdyby to „informatycy" wymyślali terminologię mechaniczną, mielibyśmy zapewne *kontroler Watta* (przypominam: mamy *regulator!*), tak jak dziś mamy *kontrolery dyskowe* i inne nonsensy.

O ile częste używanie słowa *kontrolować* (i pochodnych) dowodnie świadczy o nieuctwie, a przynajmniej o niezrozumieniu przedmiotu wypowiedzi, o tyle użycie (nawet tylko jeden raz!) słowa *kompatybilny* świadczy o czymś bodajże jeszcze gorszym, o pozerstwie intelektualnym. Osobnik używający tego (nie istniejącego w żadnym polskim słowniku!) wyrazu udaje, że ma do powiedzenia coś więcej lub coś innego niż „zgodny". Otóż nie! Angielskie słowo *compatible* oznacza tylko i dokładnie tyle, co polskie *zgodny*, nawet ta słynna (z procesów rozwodowych) *niezgodność* (np. charakterów) też po angielsku jest *incompatibility*. Po co więc przenosić, czysto fonetycznie, obce słowo, kiedy jest ścisły polski odpowiednik? Tylko chyba po to, by wśród maluczkich wzbudzić szacunek, że się niby opanowało jakieś tam mądrości. Paskudna to cecha charakteru, takie zadawanie szyku. Nic nie poradzę, ile razy słyszę *kompatybilny*, tyle razy widzę brudny kołnierzyk, wybrylantynowane włoski i sygnet z tombaku — atrybuty wiejskiego podrywacza na zabawie w remizie.

Są i trudniejsze przypadki, jest ów słynny *joystick* i jest *interface*. Słowa, które nie mają dokładnych odpowiedników w polszczyźnie. Można oczywiście przyjąć *dżojstik* i *interfejs;* brzmią okropnie, ale chyba błędu nie ma. Tyle że, z nielicznymi wyjątkami (np. *dżem*), polszczyzna nie przyjmuje słów angielskich, inna jest tradycja języka, łaskawsza dla łaciny i niemieckiego, ostatecznie dla języków romańskich niż dla angielskiego, zwłaszcza pod względem fonetycznym. O ile sobie przypominam z lektury książek Meissnera, to, co angielscy piloci nazywają *dżojstikiem*, Polacy latający w RAF-ie nazywali *orczykiem*. Ze zmierzchem konnych zaprzęgów oryginalne znaczenie słowa orczyk zaczyna się zacierać. Fonetycznie poprawne, dobrze osadzone w polszczyźnie słowo może przybrać nowe znaczenie. Podobnie, choć *pośrednictwo* znaczy dotychczas coś innego, nie widzę powodu, dlaczego nie miałoby być używane jako odpowiednik *interface*. Analitycznie, choć nie tradycyjnie, jest ono bardzo dobrym odpowiednikiem.

Garść przykładów: od oczywistych błędów (merytorycznych, nie tylko językowych) — przez oznaki wątpliwej elegancji — do spraw estetycznych. Czy to ważne? Tak, bardzo ważne. Osobnik nie zwracający uwagi na to, jak się wyraża, nie panujący nad swą mową, piszący byle jak, wystawia sobie świadectwo niechluja językowego. A to oznacza jedno z dwojga: albo jest on *homo* nie całkiem *sapiens*, albo lekceważy adresatów wypowiedzi w takim samym stopniu, w jakim pani domu lekceważy gości, przyjmując ich w papilotach i przydeptanych papuciach. Sposób wyrażania jest — niezależnie od tego, czy chemy, czy nie — wizytówką naszej ogłady, kultury i sprawności intelektualnej. Legendy o geniuszach, którzy nie umieli się wysłowić, są właśnie legendami, nikt ich jakoś nie umie poprzeć konkretnymi przykładami. Przykładów przeciwnych, ludzi naprawdę wybitnych, którzy zwracali pedantyczną wprost uwagę na poprawność wypowiedzi, jest mnóstwo. Nie brak ich i wśród informatyków, by wymienić tylko kilku: Dijkstra, Hoare, Naur, Wirth — ich prace są nie tylko doniosłe, są też pięknie napisane.

WMT
(Przedruk z Biuletynu PTI)

Stypendia PTJ

Zarząd Główny Polskiego Towarzystwa Informatycznego zawiadamia, że w terminie do 20 czerwca 1988 członkowie PTI o co najmniej dwuletnim stażu członkowskim mogą składać wnioski o przyznanie stypendiów Towarzystwa. Założeniem stypendiów jest umożliwienie wykonywania prac związanych ze statutowymi celami PTI, do których należy m.in. popieranie działalności naukowej i naukowo-technicznej we wszystkich dziedzinach informatyki i doskonalenie metod jej efektywnego wykorzystania w gospodarce narodowej, a także podnoszenie poziomu wiedzy i etyki zawodowej oraz kwalifikacji członków PTI.

Wnioski powinny zawierać: podanie do Zarządu Głównego PTI, charakterystykę pracy zgłaszanej do stypendium (w tym cel i zakres pracy oraz plan realizacji), informację o sylwetce zawodowej kandydata oraz opinię właściwego oddziału, koła lub sekcji PTI. Do stypendiów kwalifikują się zarówno prace wykonywane indywidualnie, jak i zespołowo. Bliższych informacji udziela sekretariat ZG PTI, Warszawa, telefon 43-74-44.

(KU)

Kreczmar A.: Języki obiektowe (2)

INFORMATYKA 1988, nr 2, s. 1

Druga część charakterystyki języków obiektowych, zawierająca omówienie dziedziczenia, dalszych właściwości klas oraz różnicy między dziedziczeniem a zagnieżdżeniem.

Michalski R. S.: O naturze uczenia się — problemy i kierunki badawcze (1)

INFORMATYKA 1988, nr 2, s. 4

Pierwsza część omówienia zadań i kierunków badań naukowych w dziedzinie uczenia się wspomaganego komputerem.

Łubińska K.: System dydaktyczny do nauki języka ADA/SM

INFORMATYKA 1988, nr 2, s. 8

Charakterystyka systemu do nauki języka ADA/SM przeznaczonego dla minikomputerów SM-4.

Syfert A., Raznowiecki A.: Struktura systemu operacyjnego PC-DOS (5)

INFORMATYKA 1988, nr 2, s. 10

Piąta część charakterystyki systemu operacyjnego PC-DOS, zawierająca omówienie wywołań funkcji typu CP/M.

Jaworski R., Płowiec K.: Monitor ekranowy Mazovii 1016

INFORMATYKA 1988, nr 2, s. 13

Szczegółowa charakterystyka parametrów technicznych i własności użytkowych monitora ekranowego w komputerze osobistym Mazovia 1016.

Bielecki J.: Język C. Skojarzenia parametrów z argumentami

INFORMATYKA 1988, nr 2, s. 16

Sposób i przykłady kojarzenia parametrów z argumentami w języku C.

Fryźlewicz Z.: Mikrokomputerowy analizator-tester protokołów

INFORMATYKA 1988, nr 2, s. 17

Charakterystyka skonstruowanego w Politechnice Wrocławskiej urządzenia do kontroli poprawności protokołów w sieciach komputerowych.

Kirpluk M., Sobolewski P.: Implementacja dedukcyjnej bazy danych Holmes (1)

INFORMATYKA 1988, nr 2, s. 20

Podstawowe wiadomości na temat dedukcyjnych baz danych oraz omówienie implementacji bazy danych Holmes.

Kreczmar A.: Object languages (2)

INFORMATYKA 1988, No. 2, p. 1

Second part of object language characteristics, which contains discussion of succession, further attributes of classes and difference between succession and nesting.

Michalski R. S.: On nature of learning — problems and ressearch directions (1)

INFORMATYKA 1988, No. 2, p. 4

First part of discussion about tasks and research directions in computer assisted learning.

Łubińska K.: Didactic system for ADA/SM language teaching

INFORMATYKA 1988, No. 2, p. 8

Characteristics of a system for ADA/SM language teaching on SM-4 minicomputers.

Syfert A., Raznowiecki A.: Structure of the PC-DOS operating system (5)

INFORMATYKA 1988, No. 2, p. 10

Fifth part of the PC-DOS operating system characteristics, which contains discussion of CP/M-type calls.

Jaworski R., Płowiec K.: Display of Mazovia 1016

INFORMATYKA 1988, No. 2, p. 13

Detailed technical parameters and application features characteristics of Mazovia 1016 personal computer display.

Bielecki J.: C language. Associations of parameters with operands

INFORMATYKA 1988, No. 2, p. 16

The method and examples for association of parameters with operands in the C language.

Fryźlewicz Z.: Microcomputer controlled analyser/tester for protocols

INFORMATYKA 1988, No. 2, p. 17

Characteristics of microcomputer controlled device for protocol checking in computer networks, which was constructed in Wrocław Technical University.

Kirpluk M., Sobolewski P.: Implementation of the Holmes deductive data base system (1)

INFORMATYKA 1988, No. 2, p. 20

Basic information about deductive data bases and discussion of the Holmes data base system implementation.

Kreczmar A.: Objektsprachen (2)

INFORMATYKA 1988, Nr. 2, S. 1

Zweiter Teil einer Charakteristik von Objektsprachen, der eine Besprechung von Vererbung, weiteren Klasseneigenschaften und Unterschieden zwischen Vererbung und Einnisten umfasst.

Michalski R. S.: Über Natur des Lernens — Probleme und Forschungsrichtungen (1)

INFORMATYKA 1988, Nr. 2, S. 4

Erster Teil einer Besprechung von Aufgaben und Forschungsrichtungen im Bereich des rechnerunterstützten Lernens.

Łubińska K.: Didaktisches System für Erlernung der ADA/SM-Programmiersprache

INFORMATYKA 1988, Nr. 2, S. 8

Eine Charakteristik des Systems für Erlernung der ADA/SM-Programmiersprache von SM-4-Minicomputern.

Syfert A., Raznowiecki A.: Struktur des PC-DOS-Beriebssystems (5)

INFORMATYKA 1988, Nr. 2, S. 10

Fünfter Teil einer Charakteristik von PC-DOS-Betriebssystems, der eine Besprechung von CP/M-Typ-Aufrufen umfasst.

Jaworski R., Płowiec K.: Bildschirm von Mazovia 1016

INFORMATYKA 1988, Nr. 2, S. 13

Eine detaillierte Charakteristik von technischen Parametern und Anwendungseigenschaften des Bildschirms in Mazovia 1016-Personal Computer.

Bielecki J.: C-Sprache. Assoziationen von Parametern mit Operanden

INFORMATYKA 1988, Nr. 2, S. 16

Die Method und Beispiele der Assoziationen von Parametern mit Operanden in der C-Sprache.

Fryźlewicz Z.: Mikrorechnergesteuerter Analysator/Tester für Protokolle

INFORMATYKA 1988, Nr. 2, S. 17

Eine Charakteristik der in der Technischen Universität zu Wrocław konstruierten Einrichtung für Richtigkeitsprüfung der Protokolle in Rechnernetzen.

Kirpluk M., Sobolewski P.: Implementierung des Holmes-Deduktionsdatenbanksystems (1)

INFORMATYKA 198, Nr. 2, S. 20

Grundinformation zum Thema der Deduktionsdatenbanken und eine Besprechung von Implementierung des Holmes-Datenbanksystems.

6-letnie doświadczenie software'owe sprawdzone w ponad 1000 zakładach pracy

TYLKO **SYSTEM CSK/32** jeśli zarządzanie Twoim przedsiębiorstwem wymaga minikomputera o mocy obliczeniowej Odry oraz kilkunastu skomputeryzowanych stanowisk pracy

System CSK/32 to:

MIKROKOMPUTERY QUATRO CSK/32
— przystosowane do pracy z kilkunastoma terminalami, pracujące w dowolnym trybie graficznym

OPROGRAMOWANIE
— systemowe WDOS, MIKROLAN
— narzędziowe — MEGA BLOK, TABPLAN, PL-TEKST, BGRAF, TRYS
— aplikacyjne — FK K&K, EM K&K, KADRY, PŁACE

WDROŻENIA, SZKOLENIA
— zespoły ds. wdrożeń i szkoleń pracują w:
Gdyni tel. 248 018, tlx 054792 ● Krakowie tel. 373 573 ● Poznaniu tel. 676 271
Wrocławiu tel. 481 679 ● Warszawie tel. 258 528, tlx 816711
Sopocie — w wyspecjalizowanym Salonie Sprzedaży, ul. Kombatantów 1

EKKA™
computer studio kajkowscy

EO/74/88

MIĘDZYWOJEWÓDZKA SPÓŁDZIELNIA PRACY „SIÓDEMKA"

ŁODŹ, AL. KOŚCIUSZKI 93,

ZAKŁAD INFORMATYKI I SYSTEMÓW KOMPUTEROWYCH
poleca po najniższych cenach w kraju:

— mikrokomputery 8-, 16-, 32-bitowe najwyższej jakości renomowanych firm z całego świata
— urządzenia peryferyjne:
- drukarki — również 24-igłowe i laserowe
- streamery
- napędy dyskowe 3", 5.25"
- monitory monochromatyczne, kolorowe, EGA, HEGA, VGA
- karty rozszerzenia pamięci
- kontrolery
- dyski twarde typu Winchester 20 MB, 40 MB, 60 MB, 80 MB

— systemy wielodostępne i lokalne sieci mikrokomputerowe:
- MULTI-LINK
- D-LINK
- XENIX

— materiały eksploatacyjne:
- dyskietki 5.25" MD2-D
- dyskietki 5.25" MD2-HD
- dyskietki 3" CF2
- dyskietki 3.5" MF 2DD
- taśmy barwiące do wszystkich typów drukarek STAR i NEC

Termin realizacji zamówień **natychmiast** po złożeniu zamówienia. **Bezpłatnie:** szkolenia, kursy, zestawy oprogramowania narzędziowego i użytkowego — przy dostarczeniu kompletnych systemów.

Proponujemy również na wszelkiego rodzaju mikrokomputery programy wspomagające zarządzanie przedsiębiorstwem:
— system finansowo-księgowo-kosztowy
— system zbytu i zaopatrzenia
— system technicznego przygotowania produkcji
— system materiałowy
— system kadrowy i kadrowo-płacowy (również dla pracowników akordowych)
Wymienione systemy pracują w wersjach sieciowych i wielodostępnych.

Wszelkich informacji udzielamy codziennie (oprócz sobót) w siedzibie Zakładu w Łodzi przy Al. Kościuszki 101, tel. 36-51-00, w godzinach 8—16.

EO/1005/87