PL ISSN 0209-1593

1 1987

prace naukowo-badawcze

Instytutu Maszyn Matematycznych

rok XXIX

prace naukowo-badawcze
Instytutu
Maszyn
Matematycznych

Zdzisław Wrzeszcz, Jan Ryżko, Romuald Synak, Andrzej Sikorski: Komputery optoelektroniczne 3

Zdzisław Wrzeszcz, Wojciech Przyłuski, Jerzy Kasprzyk, Andrzej Rowicki: Podstawy implementacji systemów eksperckich na mikrokomputerze 39

rok XXIX 1/1987

Adres redakcyjny: Instytut Maszyn Matematycznych
Branżowy Ośrodek INTE
02-078 Warszawa, ul. Krzywickiego 34
tel. 28-37-29 lub 21-84-41 w. 244

Prace naukowo-badawcze Instytutu Maszyn Matematycznych

Zdzisław WRZESZCZ, Jan RYŻKO
Andrzej SIKORSKI, Romuald SYNAK

Komputery optoelektroniczne

1. WPROWADZENIE

Nowoczesna maszyna cyfrowa jest jednym z najbardziej złożonych produktów myśli i technologii. Sfera jej zastosowań obejmuje już niemal wszystkie dziedziny działalności człowieka: przemysł (gdzie komputer stosuje się do planowania, projektowania wyrobu i produkcji), usługi i handel, banki i administracja różnego szczebla, szkolnictwo, służba zdrowia, obronność. Osobno trzeba wymienić naukę i badania naukowe, gdyż pochodzące stąd problemy obliczeniowe są źródłem stale rosnących wymagań, mają więc znaczny wpływ na kierunki rozwoju komputerów.

Różnorodność zastosowań sprzyja wyodrębnieniu pewnych klas komputerowych. Na przykład w przemyśle, usługach, szkolnictwie, administracji wymaga się rozwiązań standardowych, o stosunkowo niskiej mocy obliczeniowej, łatwych w obsłudze i jednocześnie tanich. Natomiast badania naukowe, szczególnie badania przestrzeni kosmicznej, biologia, fizyka ciała stałego oraz obronność, potrzebują środków o wielkich mocach obliczeniowych (tzw. superkomputerów).

Podejmowanie coraz bardziej złożonych problemów obliczeniowych przy jednoczesnym dążeniu do ułatwień w ich formułowaniu (programy) i rozwiązywaniu (obsługa systemu) wymaga ciągłej poprawy parametrów systemu i jednocześnie komplikuje strukturę wewnętrzną jego modułów; parametry te dotyczą przede wszystkim pamięci, szybkości wykonywania operacji przez procesor i wydajności transmisji informacji w systemie. W okresie ostatnich 50 lat technologia realizacji komputera przekształciła się z technologii układów elektromechanicznych, poprzez układy na lampach elektronicznych i tranzystorach, w technologię krzemowych układów scalonych wielkiej skali integracji. Przy czym pojemność pamięci, począwszy od wartości kilkudziesięciu słów osiąga obecnie wiele megabajtów w jednym konstrukcyjnym module. Oprócz stałej poprawy parametrów rozwojowi podlega także architektura systemu komputerowego.

Technologia układów optycznych i światło jako nośnik informacji wniosą bezsprzecznie wiele nowego. Korzyść główna, jakiej oczekujemy po technologii optycznej będzie wynikać z różnicy fizycznych własności fotonu i elektronu: szybkość transmisji sygnału elektrycznego jest ograniczona rezystancją, kapacytancją i induktancją przewodników, co przy fotonach nie występuje; ponadto szersze pasmo fotonów ułatwia połączenia, wiązki świetlne mogą się krzyżować bez wzajemnych oddziaływań, gdyż fotony nie mają ładunku. Stwarza to dogodne warunki do zwiększenia gęstości upakowania układów w komputerze (funktorów, rejestrów, pamięci), co jest niezmiennym dążeniem konstrukcyjnym. Droga sygnału optycznego może być dowolnie ukształtowana za pomocą światłowodu i to bez ograniczenia pasma transmisji, jak to ma miejsce w przypadku przewodnika elektrycznego. Wprowadzenie układów optycznych umożliwia równoległe przetwarzanie wielkich bloków informacji, np. obrazów, a także ich pamiętanie; informacja może mieć postać cyfrową, ale też analogową - stosownie do potrzeby problemu.

Mimo tych ewidentnych zalet technologia układów optycznych wchodzi do praktyki komputerowej bardzo powoli. Chociaż początków badań nad optycznymi układami liczącymi należy szukać w XIX w. (wg J.W. Goodmana), to jednak pewien postęp w badaniach tego kierunku widać dopiero po opracowaniu lasera (lata sześćdziesiąte), a następnie diody laserującej (AT&T, Bell Laboratories, lata siedemdziesiąte). Dotychczas nie uzyskano konstrukcji optycznego układu binarnego, który działałby zadowalająco i jednocześnie umożliwiał budowę struktur złożonych jak w przypadku elektronicznych układów wielkiej skali integracji (LSI, VLSI).

Po okresie znacznych inwestycji w badaniach nad układami optycznymi w informatyce nastąpił regres i dopiero ostatnio wzrosło zainteresowanie tymi problemami w związku z osiąganiem kresu możliwości rozwiązań elektronicznych.

Spadek tempa światowych badań nad komputerami optycznymi ma wiele przyczyn; główne z nich to:

1 - brak odpowiednich materiałów na elementy układów optycznych,

2 - ciągły rozwój technologii komputerów elektronicznych,

3 - brak silnych motywacji aplikacyjnych dla komputera optycznego.

W połowie lat siedemdziesiątych również w Polsce podjęto badania nad optycznymi układami komputerowymi (IMM, WAT, CLO). Prace te, mimo dobrych wyników, przerwano w początku lat osiemdziesiątych głównie z powodu braku możliwości finansowania. Powstaje więc pytanie o obecny status komputera optycznego (optoelektronicznego) i rozwój badań nad nim w świecie, a także w Polsce.

Wobec niedojrzałości wyników prac nad układami i podsystemami niejsna jest również kompletna postać (architektura) badanego obiektu. To właśnie zagadnienie chcemy w niniejszej pracy naświetlić przede wszystkim, ponieważ zahamowanie badań powoduje, że poglądy na architekturę komputera optycznego, jego realizację i zastosowania są silnie zróżnicowane. Stan ten został wyrażony dobitnie w dyskusji panelowej poświęconej przyszłości przetwarzania optycznego; dyskusja miała miejsce w styczniu 1986 na sympozjum O-E/LASE, Los Angeles [1]. Spotkania te - zakończyły się co prawda wspólnym poglądem na architekturę przyszłego komputera optycznego, lecz jest to sformułowanie tak ogólne, że odpowiada każdej technologii realizacji.

Jesteśmy zdania, że droga rozwoju będzie prowadzić przez połączenie technologii układów elektronicznych i optycznych, a powstały w ten sposób komputer optoelektroniczny dziedziczy w znacznym stopniu cechy architektoniczne i sferę aplikacji komputera elektronicznego, dlatego niniejszą pracę rozpoczynamy od omówienia rozwoju architektury systemu komputerowego przechodząc z kolei do prezentacji składników systemu. Składniki te rozwijają się w różnym tempie, co jest już dostateczną przeszkodą nie pozwalającą obecnie konstruować komputerów wyłącznie optycznych. Jaką drogą pójdzie rozwój architektury komputera optycznego, tego nie wiemy [4]. Jesteśmy jednak przekonani, że rozwój ten przez długi czas będzie zbieżny z rozwojem komputera elektronicznego.

Po przedstawieniu architektury systemu i jego członów (procesory, pamięci, urządzenia we/wy, połączenia) omawiamy krótko prawdopodobne kierunki zastosowań. Ponadto, podajemy też nieco informacji o roli holografii w obecnych pracach nad komputerem.

Niniejsze opracowanie jest wynikiem działania zespołu autorów, jednak tworzy ono określoną całość. Wynika to zarówno z przyjętego poglądu na rozwój architektry komputera optoelektronicznego (wiąże się z tym dobór materiału opracowania), jak też ze wspólnie zdobytych doświadczeń badawczych. Autorstwo opracowania przedstawia się następująco:

- rozdziały 1., 2., 3., 8., 9. - Z. Wrzeszcz,
- rozdział 4., 7. - J. Ryżko
- rozdział 5. - R. Synak,
- rozdział 6. - A. Sikorski.

2. ARCHITEKTURA SYSTEMU KOMPUTEROWEGO

2.1. Zarys rozwoju

Komputery mechaniczne, elektryczne, elektroniczne, optyczne to to ten sam rodzaj urządzenia przeznaczonego do przetwarzania informacji. Ten sam, ale nie taki sam, gdyż rozwój zdolności funkcjonalnych komputera rośnie silnie wraz ze zmianą technologii. O ile pierwsze maszyny matematyczne będące konstrukcjami w pełni mechanicznymi były zdolne wykonywać jedynie pojedyncze rodzaje operacji arytmetycznych (B. Pascal - 1642, G. Leibnitz - 1671, C. Babage - 1834), to już wprowadzenie przekaźników elektromechanicznych umożliwiło wydatnie zwiększenie liczby rodzajów operacji (Aiken, MARK 1 - 1944). Dopiero jednak wprowadzenie lamp elektronowych jako elementów do budowy podstawowych układów funkcjonalnych maszyny pozwoliło J. von Neumanowi (i jego licznym współpracownikom Uniwersytetu w Princeton, 1943-1946) zbudować uniwersalną maszynę programowaną [16]. Komputer ten znany pod skróconą nazwą IAS jest właśnie komputerem pierwszej generacji [17], a jego architektura odznacza się tym, że w pamięci maszyny są gromadzone dane wyjściowe do obliczenia, program obliczenia i wyniki pośrednie. Elementy programu zawierające dane i instrukcje sterujące są pobierane sekwencyjnie, a sekwencja ta jest ustalana przez tzw. licznik instrukcji, którego zawartość również może podlegać modyfikacji. Takie rozwiązanie architektury systemu miało zasadnicze znaczenie zarówno dla rozwoju konstrukcji komputera, jak i jego aplikacji.

Następna generacja komputerów powstaje dzięki wprowadzeniu tranzystora jako elementu układu podstawowego (IBM, maszyna 7094, rok 1960) [28]. Nowe własności funkcjonalne i parametry wiążą się z zastosowaniem rejestrów wskaźnikowych i układów do realizacji arytmetyki zmiennego przecinka, procesorów do operacji wejścia/wyjścia, lampy promieniowej jako pamięci. Taki rozwój środków technicznych, umożliwił wprowadzenie języków wysokiego poziomu (FORTRAN, COBOL) oraz standardowych podprogramów; jest to więc początek tworzenia oprogramowania w sposób systemowy. W tym czasie w Polsce powstała maszyna ZAM-41 (Instytut Maszyn Matematycznych, zespół pod kierunkiem prof. L. Łukaszewicza - 1963).

Trzecia generacja komputerów wiąże się z opracowaniem podstawowego układu funkcjonalnego w formie scalonej. Nie ma potrzeby stosowania technologii łączenia (lutowania) oddzielnych elementów, co pociągało za sobą stratę czasu, wysokie koszty produkcji, duże gabaryty urządzenia, niską niezawodność, a więc same kłopoty. Dzięki wprowadzeniu układu scalonego powstają maszyny (np. IBM 360 - 1964 r., IBM 370 - 1970 r.), których pamięci wewnętrzne osiągają pojemności rzędu kilkuset KB.

Technologia układów scalonych umożliwiła następujący rozwój komputera:

- zastępując układy na elementach dyskretnych znacznie zwiększono upakowanie układów i zmniejszono gabaryty; jednocześnie wzrosła niezawodność,
- obok pamięci ferrytowych pojawiły się znacznie szybsze pamięci półprzewodnikowe, o większej pojemności,
- szerokie zastosowanie znalazła technika mikroprogramowania upraszczająca konstrukcję procesorów,
- pojawiły się pierwsze techniki realizacji procesów współbieżnych umożliwiające wieloprogramowość i przetwarzanie równoległe zadań obliczeniowych,
- powstały metody wspólnego użytkowania zasobów procesorowych i pamięciowych przez kilka procesów (programów).

Wzrost efektywności użytkowej komputera jest więc olbrzymi.

Czwarta generacja komputerów powstała około roku 1975. Zainicjowanie prac przypisać trzeba G.M. Amdahlowi [19], byłemu pracownikowi IBM, który dostrzegł możliwości zamknięcia procesora typu IBM 370 w kilku zaledwie układach wielkiej skali integracji (AMDAHL 470).

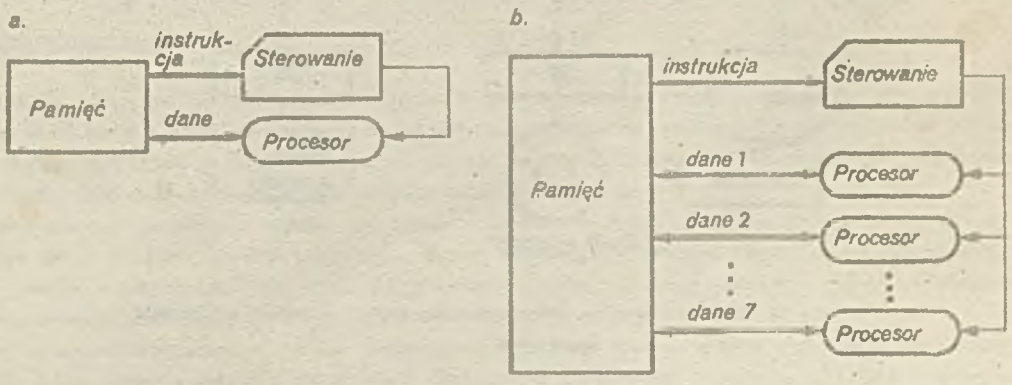
Jesteśmy obecnie u progu piątej generacji komputerów. Podobnie jak poprzednio i ta generacja również bardzo zależy od postępu technologii układów podstawowych [20], choć nie wyłącznie. Z założenia zmiany architektury komputera V generacji wiążą się przede wszystkim z innym typem przetwarzania. Mówimy tu o przetwarzaniu i gromadzeniu wiedzy, procesach wnioskowania w miejsce (a raczej obok) procesów obliczeniowych. Języki tracą swój imperatywny charakter na rzecz deklaratywnych zdolności opisu problemu i jego rozwiązania. Wszystko to wymaga ogromnych pojemności pamięci, wielowymiarowych układów podstawowych, nowych układów dostępu do informacji itd., a więc takich cech, jakich spodziewamy się po układach optycznych.

Realizacja opisanych postulatów architektonicznych w technice układów elektronicznych nastręcza wiele trudności. Wydaje się, że pokonanie tych trudności za pomocą układów optoelektronicznych będzie łatwiejsze.

Rozwój badań nad komputerem optycznym musi naszym zdaniem, łączyć się z głównym nurtem rozwoju komputerów elektronicznych; chodzi również o to, by nie marnować ogromnego dorobku zarówno w aplikacji komputerów, jak też w badaniach nad systemem, zwłaszcza w zakresie organizacji systemu oraz języków formułowania i rozwiązywania problemów.

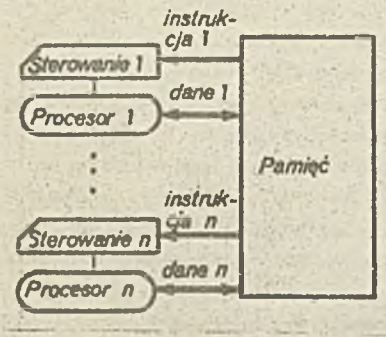
Potencjalne możliwości techniki optycznej, tj. olbrzymia szybkość działania układów, równoległe przetwarzanie itd. sprawiły, że uwaga konstruktorów skupia się w głównej mierze na linii prowadzącej do tzw. superkomputerów. Linię tę wyznaczają np. zastosowania w badaniach naukowych (np. badania dynamiki molekuł i badania fizyki ciała stałego [8]), w prognozowaniu pogody na znacznych obszarach (ważne zarówno dla gospodarki, jak dla komunikacji i celów obronnych)[9]. Zastosowania takie wymagają komputerów o olbrzymich mocach obliczeniowych (jak np. CRAY [10]).

Komputery elektroniczne rozwijają się w temple wyznaczonym przez rozwój technologii układów podstawowych i pamięci; przyjrzyjmy się kolejnym postaciom struktury systemu. Jak już wyjaśnialiśmy, w I generacji komputera elementy programu realizowane sekwencyjnie są przesyłane między pamięcią i procesorem (rys. 1a), i właśnie to przesyłanie uznano za pierwsze "wąskie gardło" organizacji von-neumanowskiej, gdyż przy ograniczonej wydajności transmisji zwiększenie szybkości działania procesora nic nie daje. Wprowadza się więc wiele procesorów przetwarzających dane równolegle (rys. 1b), a następnie stosuje rozwiązania wielomodułowe, gdzie każdy moduł skupia zarówno przetwarzanie, jak i sterowanie (rys. 2). System komputerowy musi zawierać

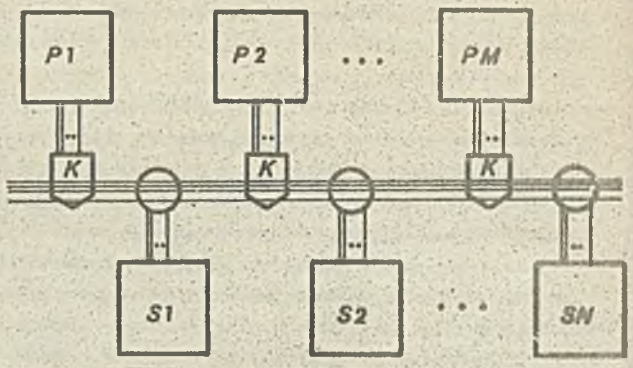


Rys. 1. a - struktura SISD, b - struktura SIMD; (SISD - Single Instruction Singular Data, SIMD - Single Instruction Multiple Data)

również obok procesorów, urządzenia wejścia i wyjścia. Urządzenia te z natury wolniejsze od procesorów opóźniają realizację całego procesu obliczeniowego. Wprowadza się więc wiele urządzeń we/wy rozdzielając ich zadania w celu zwiększenia szybkości lub dubluje się urządzenia, aby zwiększyć niezawodność systemu. Magistrala systemowa musi więc odpowiadać zarówno wymogom szybkości, jak też niezawodności. Wymaganiom tym odpowiada struktura przedstawiona na rys. 3 umożliwiająca łączenie dowolnego procesora w dowolnym urządzeniu dynamicznie; struktura ta ma wręcz charakter wieloprocesorowy i wielourządzeniowy[7].



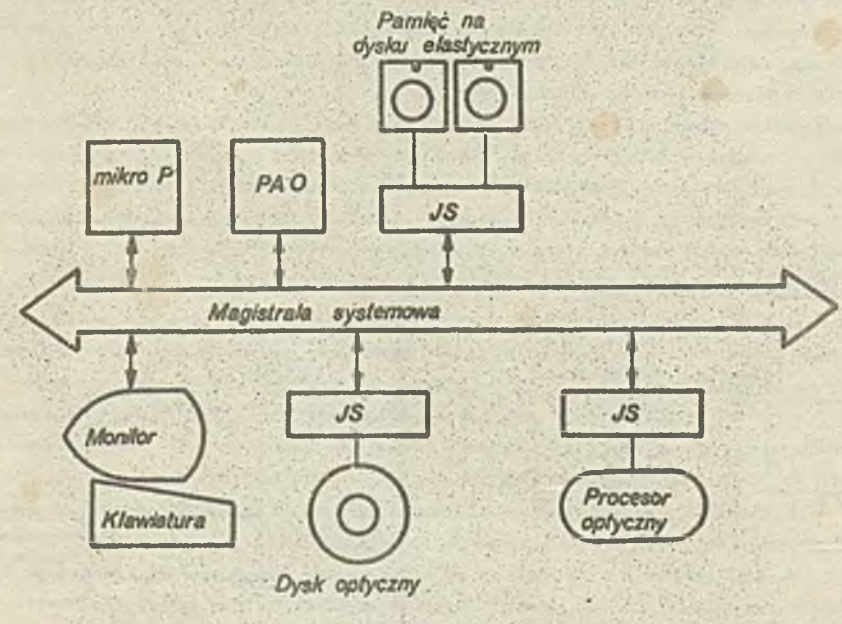
Rys. 2. Struktura MIMD (MIMD - Multiple Instruction Multiple Data)



Rys. 3. Struktura POLYBUS: P - procesor, S - urządzenie, K - klucz

Innym bardzo ważnym rodzajem komputerów są tzw. minikomputery i mikrokomputery. Są to tanie, masowo stosowane komputery do zadań codziennych zarówno w gospodarce, jak i w innych dziedzinach (nauka i nauczanie, administracja, służba zdrowia itd.). Rozwój urządzeń optoelektronicznych, takich jak pamięci optyczne, czytniki dokumentów, drukarki laserowe itd. poważnie wpłynie na rozwój systemów mini- i mikrokomputerowych (rys. 4).

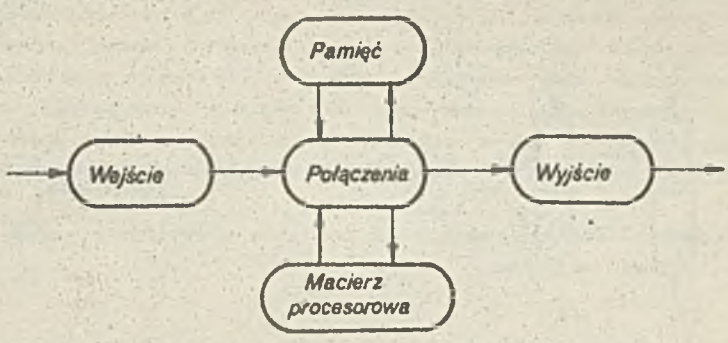
Mówiliśmy dotychczas o skupionej formie systemu, jednak systemy komputerowe mogą działać w grupach rozproszonych tworząc sieci komputerowe. Połączenia odgrywają tu zasadniczą rolę.



Rys. 4. Struktura systemu mikrokomputerowego: JS - jednostka sterująca, PAO - pamięć operacyjna

2.2. Architektura komputera optycznego

Zagadnienie architektury optycznego systemu komputerowego skupia w ostatnim czasie uwagę wielu badaczy. To, co przez wiele lat uważano za optyczny komputer okazuje się być optycznym urządzeniem przetwarzania sygnału, a w najlepszym przypadku specjalnym procesorem. Dyskutujący zagadnienie architektury badacze (m. in. panel dyskusyjny Air Force Office of Scientific Research, skupiający takich uczonych, jak, H.J. Caulfield, S. Collins, C. Guest, B.K. Jenkins, D. Psaltis, A.A. Sawchuk [1]) dochodzą do wniosku, że przyszły komputer optyczny będzie mieć architekturę podobną do architektury komputera elektronicznego (rys. 5); różnica natomiast leży w realizacji członów tej architektury, a także w repertuarze ich funkcji.



Rys. 5. Architektura komputera optycznego wg [1]

Ważnym elementem architektury komputera optycznego są połączenia występujące na różnych poziomach zarówno w module, jak i między modułami.

Część przetwarzająca będzie zawierać cały zestaw procesorowy, co pozwoli na równoległą realizację składników zdekomponowanego obliczenia złożonego, np. dwu- lub trójwymiarowe struktury danych, lecz także różnorodne typy danych, z których każdy może być obsługiwany przez oddzielny procesor. Do zestawu procesorowego mogą wchodzić zarówno procesory cyfrowe, jak i analogowe.

Pamięć optyczna obejmuje wszystkie adresowalne poziomy pamięciowe, począwszy od rejestrów procesorowych i urządzeniowych, poprzez poziom pamięci operacyjnej aż do pamięci plików. Poszczególne poziomy pamięci nie muszą występować w formie skupionej i będą związane z odpowiednimi modułami przetwarzającymi. Ze względu na optyczny charakter swoich układów, pamięci będą adresowane zarówno przez indeks pozycji, jak też przez zawartość; ten drugi typ adresacji ma szczególne znaczenie przy realizacji algorytmów szukania, porównania według wzoru, sortowania itp.

Optyczne urządzenia wejścia/wyjścia mają przede wszystkim za zadanie przyjęcie i wydanie informacji w postaci obrazu. Ponadto muszą też służyć jako przetworniki elektryczno-optyczne (gdyż sygnały elektryczne będą nadal występować w otoczeniu komputera), optyczno-optyczne (różne formy źródeł i detektorów), bufory informacyjne, formatery. Urządzenia we/wy będą zawierać preprocesory i postprocesory, co daje im rangę funkcjonalnie samodzielnego modułu.

2.3. Kierunki dalszego rozwoju

Omówiliśmy dotychczas komputery optoelektroniczne będące w znacznym stopniu rozwinięciem architektonicznym (i technologicznym) komputerów elektronicznych. Komputery takie będą z pewnością realizowane przez okres kilku, kilkunastu najbliższych lat; długości tego okresu nie można określić nie tylko ze względu na znaczne trudności badawcze i techniczne, lecz także z racji zmienności zainteresowania sponsorów programów badawczych [1, 31].

W miarę poprawy parametrów modułów optycznych odpowiedzialnych za pamiętanie struktur przestrzennych (równoległe przetwarzanie takich struktur, sterowanie, łączenie masowo występujących prostych elementów przetwarzających itd.) udział elektroniki zmaleje, a architektura komputera zyska nową postać, różną od von-neumanowskiej i post-von-neumanowskiej architektury komputerów optoelektronicznych pierwszego okresu.

Prace nad optoelektronicznymi maszynami matematycznymi drugiego okresu wiążą się z badaniami nad sieciami neuronowymi [32, 33], a tym samym z badaniami nad emulacją działania mózgu człowieka. Wprawdzie rozwinięte struktury superkomputerów i obecne maszyny sztucznej inteligencji [34] z powodzeniem rozwiązują wiele trudnych i złożonych problemów, np. z zakresu modelowania kompozycji nowych struktur chemicznych, diagnostyki, problemów szachowych itp. są to jednak tak zwane problemy dobrze zdefiniowane. Rozwiązywanie nawet prostych, lecz źle zdefiniowanych problemów (np. rozumienie i tworzenie

mowy potocznej, wnioskowanie na podstawie zdrowego rozsądku, poruszanie się w złożonej dynamicznej, 3-wymiarowej przestrzeni za pomocą detektorów 2-wymiarowych z zakłóceniem) jest dalece niezadowalające. Trudności te usprawiedliwiają poszukiwanie nowych architektur, na przykład na wzór inteligentnych systemów biologicznych. Stąd też z jednej strony trwają próby zdefiniowania sposobu rozwiązywania problemów przez inteligentny system biologiczny (ISB), z drugiej zaś - próby konstrukcji technicznych modeli ISB. Patrząc na ISB jako na pewien zestaw układów i sterowania przyjmuje się obecnie, że podstawowymi elementami układowymi IBS są neurony, jako elementy przetwarzające oraz pewien system synapsów (specjalnych kontaktów do łączenia z dendrytami innych neuronów), dendrytów (pasywnych linii przesyłania impulsów) i aksonów (elektrycznie aktywna forma zbliżona funkcjonalnie do dendrytu), które łączą neurony w złożoną sieć [35]. Mózg ludzki może zawierać około 10^{10} neuronów, a każdy neuron może łączyć się z 10 tysiącami innych neuronów poprzez synapsy, których zdolności przekaźnikowe mogą ulegać zmianie. Wprawdzie działanie elementów IBS jest stosunkowo powolne, to jednak możliwości obliczeniowe IBS są ogromne; wiązać to należy z rozłożeniem obciążeń równomiernie na elementy komunikacji i elementy przetwarzania. W każdej chwili, w działaniach są zaangażowane znaczne liczby elementów środowiska IBS. Dwie pierwsze cechy IBS to redundancja i rozproszenie zasobów obliczeniowych, następnymi zaś są zdolność uczenia się i zdolność do samoorganizacji. Pozwala to na przystosowanie zasobów obliczeniowych do problemów jedynie częściowo opisanych. Ustalając wzorce problemowe można wykorzystać zasady rozwiązań dla różnych problemów, jest więc możliwość wybitnie efektywnego postępowania - tworzenia uniwersalnego systemu rozwiązywania.

Modelowanie sieci neuronowych kieruje uwagę badaczy na układy optyczne, zwłaszcza ze względu na ich ogromne możliwości tworzenia połączeń i przetwarzania równoległego.

X1	X2	ANDG
0	0	0
0	1	0
1	0	0
1	1	1

a.

X	NOTG
0	1
1	0

b.

Rys. 6.a - tablica prawdy dla funktora ANDG. b - tablica prawdy dla funktora NOTG; (ANDG - bramka iloczynu logicznego, NOTG - bramka negacji logicznej)

3. PROCESORY

3.1. Optyczny procesor binarny

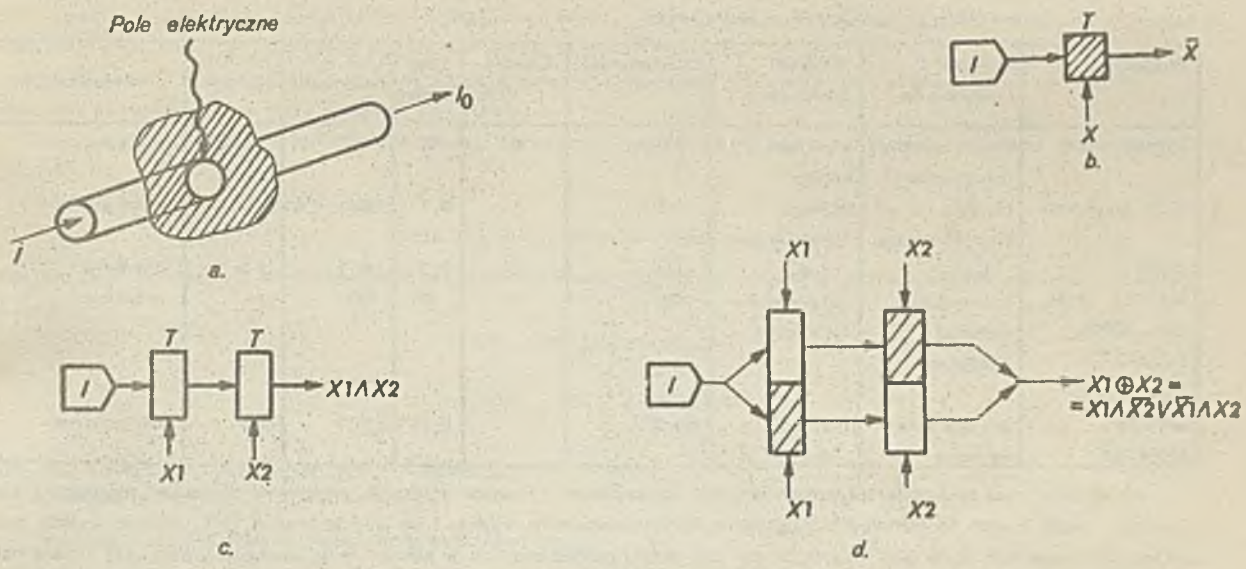
Zadaniem procesora jest wykonywanie operacji arytmetycznych i logicznych, tworzących zbiór operacji podstawowych. Zbiór ten obejmuje takie operacje, na które może być zdekomponowane dowolne obliczenie przy formułowaniu algorytmu wykonania obliczenia na określonych danych. Dane w komputerze są reprezentowane jako liczby pamiętane w pamięci, a następnie przesyłane do rejestrów procesora i poddane sekwencji operacji odpowiednio do programu.

Współczesne komputery wykorzystują notację binarną, stąd elementy maszyny mają charakterystyki dwupoziomowe, a elementy pamięciowe - bistabilne. Słowo maszynowe zawiera więc dowolną treść reprezentowaną przez liczbę binarną, tj. zbiór zer i jedynek. Wynikiem operacji jest również liczba binarna. Stąd podstawowe elementy maszyny są niczym innym jak elementami realizującymi funkcje logiczne lub boolowskie (od nazwiska G. Boola, 1815-1864). Zbiór operacji podstawowych może być zrealizowany za pomocą pewnego zestawu funktorów logicznych, tj. ANDG, NOTG lub ORG, NOTG, których działanie opisują tablice prawdy (rys. 6).

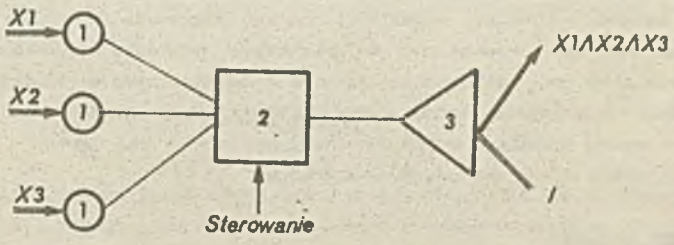
Konstrukcję optycznego funktora logicznego można przedstawić w najprostrzej postaci jako pewien modulator, którego transmitancja optyczna T ulega zmianie pod wpływem sygnału X. Modulator taki jest umieszczony na trakcie włókna optycznego wiodącego światło ze źródła I. Sygnał X sterując transmisją T zmienia jej wartość, a dzięki temu zmienia wartość parametru światła we włóknie optycznym [5]. Układ taki realizuje więc funkcję negacji sygnału, jest więc funktorem NOTG. Działanie tego funktora oraz działanie funktora bardziej złożonego, pokazano na rys. 7.

Realizacja techniczna maszyny wymaga uzupełnienia zestawu funktorów o stabilne elementy pamięciowe z możliwością zapisu i odczytu zawartości. Ponadto funktor optyczny może mieć, oprócz modulatora, również wzmacniacz sygnału po to, by nie ograniczać traktu przetwarzającego do kilku elementów ze względu na straty i by umożliwić budowę funktorów ze sprzężeniem zwrotnym (rys. 8).

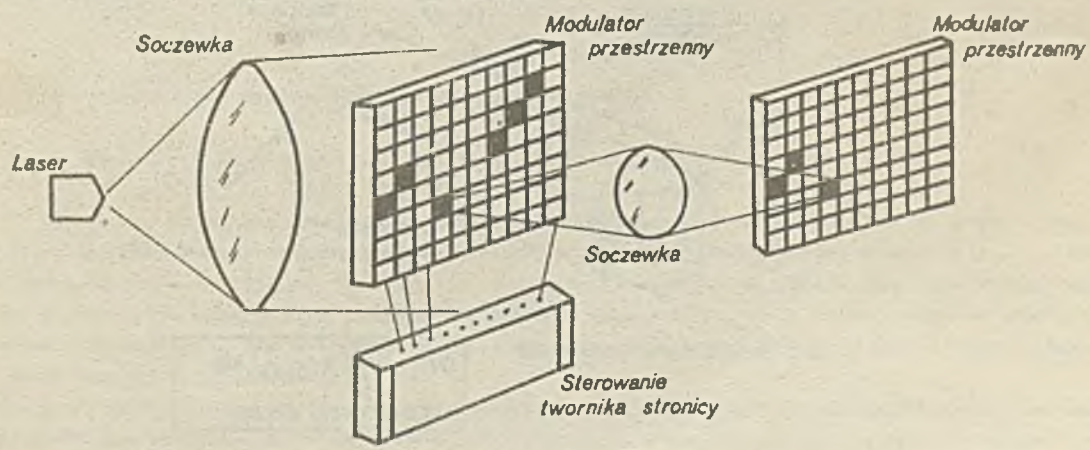
Na rysunku 9 pokazano pierwszy stopień traktu przetwarzania w procesorze optycznym. Istotnymi elementami są tu modulatory przestrzenne [3] będące macierzowymi strukturami złożonymi z binarnych elementów optycznych. Informacja w pamięci bloku danych (np. obrazu), wprowadzona do pierwszego modulatora ulega transformacji w wyniku działania drugiego modulatora.



Rys. 7.a,b - koncepcja optycznej realizacji funktora NOTG
c - funktora ANDG, d - oraz różnicy symetrycznej



Rys. 8. Schemat technicznej realizacji podstawowego układu optoelektronicznego:
1 - fotodioda, 2 - wzmacniacz elektroniczny sygnału, iloczyn sygnału,
3 - sterowane lustro



Rys. 9. Trakt przetwarzania w optycznym procesorze binarnym

Podstawowymi materiałami niezbędnymi do budowy modulatorów przestrzennych są materiały o nieliniowym współczynniku załamania (tabl. I). Idealnymi materiałami będą takie, które pozwolą budować optyczne odpowiedniki tranzystora zarówno w aspekcie działania elementu, jak też technologii wytwarzania.

Tablica I. Przykłady realizacji modulatorów przestrzennych

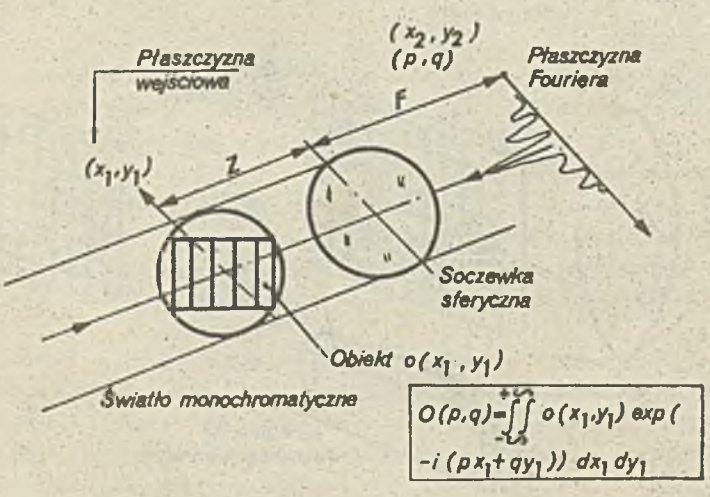
Producent	Materiał modulatorów	Materiał adresowalny	Rozróżnialność paralinii mm	Czułość μ J/cm²	Czas zapisu ms	Czas wymazywania ms	Czas pamiętania ms	Adresowanie
Hughes, USA	ciekłe kryształy nematyczne	siarczek kadmu	30	6	10	15	15	optyczne
ITEK, Sumimoto	bizmut, tlenek krzemu	bizmut, tlenek krzemu	6	5	0,1	0,1	2 h	optyczne
ZSRR	j.w.	j.w.	10	5	0,1	0,1	2 h	optyczne
HARRIS, NCR, CBS, ERIM, FUJINON, HONEYWELL	termoczuły materiał plastyczny	poly-n-winylokarbozol	1400	5	10	100	lata	optyczne
LITTOW, SEMETEX	granaty żelazowe	matryca elektrod	128x128	-	0,001	0,001	lata	elektroniczne

Przez długi czas realizacja optycznych układów podstawowych (zwanych potocznie optycznymi bramkami) nastręczała wiele trudności. Jedna z nich to różnorodność zjawisk wykorzystywanych na wejściu i na wyjściu bramki [21], dlatego wyjście jednej bramki nie mogło być użyte jako wejście drugiej bez układu pośredniczącego w postaci diody detekcyjnej badającej zmianę natężenia sygnału. Dioda ta powoduje regenerację sygnału za pomocą diody laserującej wytwarzającej sygnał z odpowiednią fazą. Po roku 1980 sytuacja uległa znacznej poprawie. Opracowano wiele modeli układów wykazujących nieliniowość optyczną już na niskich poziomach energii [22, 23, 24], chociaż w dalszym ciągu były to dziesiątki watów na bit [25].

Na drodze do uzyskania optycznych bramek o odpowiednich własnościach (wysoki kontrast pomiędzy logicznym zerem i jedynką, dostęp do elementu w strukturze wielobitowej, wysokie wzmocnienie pozwalające wysterować co najmniej dwa następne układy, mały pobór mocy przy wysokiej szybkości przełączania, przywracanie pierwotnego poziomu logicznego po każdym przełączeniu) stoi niewątpliwie brak odpowiednich materiałów w rodzaju krzemu w układach elektronicznych. Dlatego też wiele środków trzeba poświęcić na te badania. Rodzaje materiałów branych pod uwagę to m. in. materiały wykazujące zmiany współczynnika załamania w wyniku pobudzania o znacznym natężeniu (niobian litu, bizmut - tlenki krzemu, tytanat baru i in.). Niezwykle interesujące wydają się być też związki nieorganiczne.

3.2. Optyczny procesor analogowy

Przetwarzanie analogowe jest starszą formą przetwarzania optycznego. Zasadniczym jego atutem jest prostota układu przetwarzającego i równoległość (jednoczesność) przetwarzania danych będących w istocie dwuwymiarowym obrazem, np. fotografią sceny. Na rysunku 10 pokazano podstawowy stopień traktu przetwarzania informacji, którego zasadniczym elementem



Rys. 10. Układ optycznej transformacji Fouriera

jest soczewka optyczna. W płaszczyźnie wejściowej tej soczewki jest umieszczony transparent zawierający obraz $o(x_1,y_1)$ pewnego obiektu. W płaszczyźnie wyjściowej natomiast uzyskuje się widmo przestrzenne $o(p, q)$ obrazu z wejścia, na który pada skolimowana wiązka światła spójnego (np. z lasera). Następny taki stopień może zrealizować odwrotną transformację, przywracając rzeczywistą postać.

Analogowe techniki przetwarzania obrazu bazują więc na:
- fourierowskiej transformacji funkcji reprezentującej obraz obiektu, może to być obiekt realny,
- określeniu cech widma przestrzennego pierwotnego obrazu w sposób równoległy,
- porównaniu cech widma badanego z cechami wzorców zawartych w pamięci komputera (przy analizie lub rozpoznawaniu),
- usuwaniu elementów widma (przy przetwarzaniu obrazu).

Chcąc przybliżyć działanie układu transformacji fourierowskiej przedstawimy pewne podstawowe zależności matematyczne [27], [26].

Rozkład światła w płaszczyźnie widma

$$E(p,g)=\iint E(x_1,y_1)\exp(-i(px_1+qy_1))dx_1\,dy_1$$

gdzie $E(x_1,y_1)$ rozkład pola elektrycznego w płaszczyźnie obiektu (X_1,y_1) jest proporcjonalny do rozkładu elementów obiektu $o(x_1,y_1)$

$$E(x_1,y_1)=E_o(x_1,y_1)\;o(x_1,y_1)$$

$$p=2\pi x_2/\lambda F;\quad q=2\pi y_2/\lambda F$$

$E_o(x_1,y_1)$ - rozkład skolimowanego światła spójnego o długości fali λ,
F - ogniskowa obiektywu.

Rozkład natężeń światła w płaszczyźnie wyjściowej (x_2,y_2)

$$I(p,q)=E(p,q)\;E^*(p,q)$$

gdzie * - wartość sprzężona.

Rozbudowanie traktu przetwarzania o dodatkowy stopień transformacji optycznej oraz wprowadzenie w płaszczyźnie (x_2,y_2) filtru dopasowanego pozwalają zbudować tzw. procesor filtracji dopasowanej. Istotnym elementem jest tu filtr dopasowany, którego transparent można opisać za pomocą funkcji

$$H(p,q)=\frac{O^*(p,q)}{|N(p,q)|^2}$$

gdzie $O^*(p,q)$ jest sprzężonym przekształceniem fourierowskim funkcji obiektu $o(x_1,y_1)$, a $N(p,q)$ - fourierowskim przekształceniem funkcji szumu $n(x_1,y_1)$. Jeśli $n(\ )$ jest tzw. szumem białym /$n(x_1,y_1)$ - funkcja Gaussa/, to

$$H(p,q)=O^*(p,q)$$

Jeśli teraz w płaszczyźnie wejściowej (x_1,y_1) umieścimy transparent obrazu $o(x_1,y_1)$, a w płaszczyźnie widmowej (x_2,y_2) - jego filtr dopasowany, to rozkład pola elektrycznego w płaszczyźnie (x_2,y_2)

$$E(x_2,y_2)\sim O(p,q)\;O^*(p,q)$$

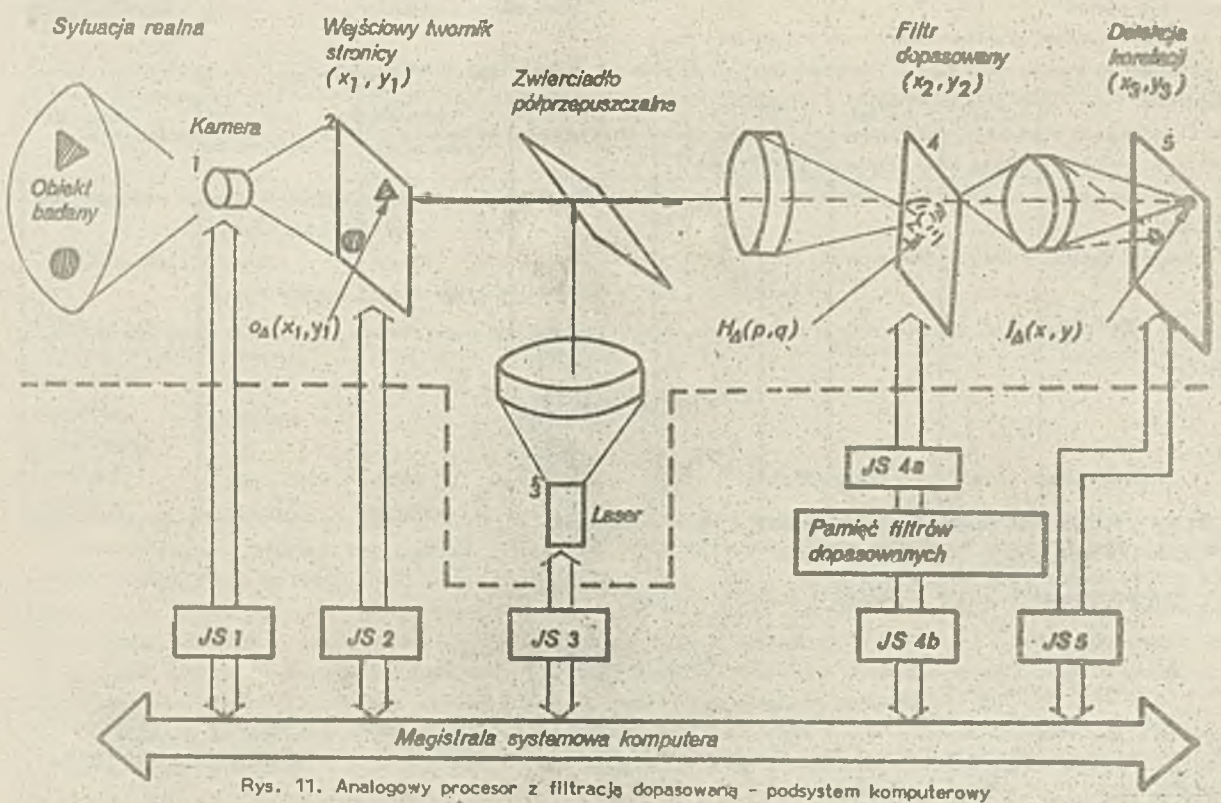
a rozkład pola w płaszczyźnie wyjściowej drugiego stopnia będzie mieć postać

$$E(x_3,y_3)=A\iint_{-\infty}^{\infty} O(p,q)\;O^*(p,q)\exp(-i(px_3+qy_3))dpdq=A\iint_{-\infty}^{\infty} dx_1,y_1)\;o(x_3+x_1,y_3+y_1)dx_1dy_1$$

Tak więc rozkład $E(x_3y_3)$ jest funkcją autokorelacji sygnału (obrazu) wejściowego $o(x_1y_1)$. Jeśli np. w płaszczyźnie wejściowej (x_1y_1) umieścimy transparent dwóch obiektów, z których jeden o kształcie trójkąta jest obiektem o (x_1,y_1), którego istnienie i położenie pragniemy rozpoznać, to umieszczając w płaszczyźnie widmowej (x_2,y_2) filtr dopasowany H (p,q) tego obiektu uzyskamy w płaszczyźnie korelacji (x_3,y_3) maksimum intensywności światła w tym samym położeniu względem obiektu $o_o(x_1,y_1)$ co w płaszczyźnie wejściowej. Jest to więc metoda pozwalająca wykryć istnienie poszukiwanego obiektu i jego względne położenie w scenie łącznie z innymi obiektami.

Jak to wynika z podanych wyżej formuł operacja wykrywania i lokalizacji obiektu dotyczy całości obiektu, bez potrzeby od dzielnej obróbki jego elementów, jak to się dzieje w technice stosującej proces binarny. Tym też należy tłumaczyć wielką szybkość obliczeń w technice stosującej procesor analogowy.

Ogólną zasadę realizacji procesora z filtrem dopasowanym pokazuje rys. 11 (część nad linią przerywaną).



Rys. 11. Analogowy procesor z filtracją dopasowaną - podsystem komputerowy

3.3. Praca procesora analogowego w systemie komputerowym

Analogowy procesor optyczny powinien realizować swoje operacje automatycznie. Oznacza to, że jego elementy, takie jak: 1 - układ wprowadzania informacji (kamera), 2 - obiektowy twornik stronicy, 3 - laser, 4 - twornik filtrów dopasowanych, 5 - detekcja sygnału korelacji, przyjmują sterowanie przez jednostki sterujące z głównego programu komputera rozwiązującego problem. Włączenie procesora analogowego do systemu wymaga więc opracowania m. in. (rys. 11):

- jednostek sterujących /JS (1)...JS(5)/ transformujących sygnały magistrali systemowej komputera na przebiegi sterujące elementami optycznymi,
- specjalnego systemu operacyjnego nadzorującego pracę jednostek,
- języka programowania, którego elementy m. in. umożliwiają formułowanie wyrażeń dotyczących rozpoznawania obiektów i zawierających operacje realizowane przez analogowy procesor optyczny.

Szersze omówienie tych zagadnień wykracza poza ramy niniejszej pracy.

4. PAMIĘCI

Pamięć stanowi obok procesora istotny podzespół każdego komputera. Oba te bloki są ściśle sprzężone ze sobą i informacja jest przesyłana pomiędzy nimi.

Zadaniem pamięci jest przechowywanie danych i programów oraz szybkie ich odczytywanie i zapisywanie. Ze względu na zasadę dostępu do poszczególnych komórek, pamięci dzielą się na adresowe i asocjacyjne; odczyt asocjacyjny odbywa się przez kojarzenie zawartości z wzorcem.

Poza wysoką niezawodnością, od pamięci dużej pojemności informacyjnej wymaga się zwykle szybkiej realizacji zadań (zapis, odczyt) i niskiej ceny za jednostkę pamiętanej informacji. Żądania te są sprzeczne z sobą, co w rezultacie doprowadziło do powstania hierarchii pamięci - od bardzo szybkich i drogich do bardzo tanich i wolnych. Do realizacji wykorzystywane są różne technologie, ciągle udoskonalane i wypierane przez inne. Jedną z nich jest technologia optyczna, z którą wiąże się duże nadzieje, zwłaszcza od czasu wynalezienia lasera.

Dotychczasowy rozwój pamięci zmierzał do utworzenia tego podzespołu jako konstrukcyjnie i funkcjonalnie samodzielnego modułu. Pozwala to obecnie łączyć pamięci optyczne również z komputerem elektronicznym.

4.1. Aktualny stan pamięci komputerowych

Współczesne komputery potrzebują kilka rodzajów pamięci, które ogólnie możemy podzielić na wewnętrzne i zewnętrzne. W pierwszej grupie najważniejsza jest pamięć operacyjna, umożliwiająca dostęp do dowolnego adresu (RAM - Random Access Memory), która przez wiele lat realizowana była na rdzeniach ferrytowych, a obecnie niemal wyłącznie na półprzewodnikach. Pojemność tej pamięci, nawet dla mikrokomputerów, sięga kilku milionów bajtów (8-bitowych jednostek informacji) z tendencją do dalszego wzrostu. Cykl pracy (odczyt - zapis) można ocenić na ok. 200 ns (czas dostępu rzędu 100 ns) z tendencją malejącą. Trzecim istotnym parametrem pamięci operacyjnej (już ekonomicznym, a nie technicznym) jest cena. Koszt miliona bitów tej pamięci wynosi obecnie (dane z wiosny 1987 r.) kilkanaście dolarów i spada dość szybko. Prognozy mówią o 2,5 dolara za Mbit w 1990 roku.

Oprócz pamięci operacyjnych do pamięci wewnętrznych należą pamięci stałe (ROM - Read Only Memory) o zbliżonym do pamięci operacyjnych czasie odczytu i nieco mniejszej pojemności (rzędu części Mbajta). Czasem zalicza się tu również niewielkie szeregowe pamięci dyskowe o stałej zawartości, których jednak czas dostępu jest znacznie dłuższy, rzędu pojedynczych milisekund.

Ostatnią wreszcie grupę pamięci wewnętrznych stanowią bardzo szybkie pamięci zwane czasami pamięciami notatnikowymi (ang cache), które mają krótki cykl rzędu pojedynczych nanosekund i na ogół niewielką pojemność - do kilku tysięcy bitów. Ten rodzaj pamięci wewnętrznych jest ściśle związany z procesorem i trudno wyodrębnić go z jednostki centralnej procesora, która często obejmuje również część pamięci operacyjnej.

Pamięci zewnętrzne natomiast to jednostki o znacznie większych pojemnościach, sięgających Gbajtów i szeregowym dostępie. Umożliwiają one znaczne prędkości przesyłania informacji do kilkunastu Mbajtów na sekundę, lecz czas dostępu jest dłuższy i wynosi od kilkunastu do kilkuset milisekund. Cena (uwzględniając koszt stacji napędowych) jest rzędu pojedynczych dolarów za Mbajt i nadal spada. Dominującym rozwiązaniem są różnego rodzaju systemy magnetyczne (dyski, taśmy, układy na domenach cylindrycznych), lecz tu właśnie po raz pierwszy oferowane są handlowo pamięci optyczne w postaci dysków o dużej pojemności. W zależności od pojemności stosowane są wśród pamięci magnetycznych dyski elastyczne (rzędu 1 Mbajta), dyski typu Winchester (od kilkunastu do kilkuset Mbajtów) oraz systemy taśmowe (podobny zakres).

Wśród pamięci zewnętrznych wyróżnić można grupę o dużych pojemnościach i stosunkowo długich czasach dostępu zwaną pamięciami archiwalnymi. Przewiduje się, że wśród tych pamięci najszybciej rozwijać się będą dyski optyczne.

4.2. Podstawowe cechy

Zarówno w całym komputerze optycznym, jak i w pamięciach wiązka światła laserowego wykorzystywana jest do procesów zapewniających realizację wymaganych funkcji. W tym przypadku chodzi o zapis i odczyt informacji. Rozwiązanie takie budzi nadzieję na osiągnięcie korzystniejszych niż dotychczas parametrów.

Wyeliminowanie, częściowe przynajmniej, połączeń przewodowych powoduje uproszczenie konstrukcji i przyspiesza przesyłanie informacji. Wykorzystanie własności optycznych (czy magnetoptycznych lub akustooptycznych) materiałów rozszerza zakres wykorzystywanych ośrodków i zwiększa możliwość znalezienia lepszej realizacji. Silne skupienie wiązki pozwala uzyskiwać znaczne gęstości informacji, a możliwość odczytania jej w dowolnej płaszczyźnie pozwala na swobodny dostęp do dowolnego adresu. Dodatkowe możliwości stwarza holografia przestrzenna, pozwalająca na przechowywanie w tej samej objętości ośrodka wielu różnych informacji w zależności od kąta padania wiązki.

Przenoszenie i zapamiętywanie informacji za pomocą fotonów, a nie elektronów stwarza też korzystniejszy stosunek sygnałów istotnych do zakłóceń, gdyż fotony nie mając ładunku są mniej wrażliwe na pola elektromagnetyczne.

Podobnie jak w procesorach istnieją tu duże możliwości zrównoleglania operacji zarówno przez stosowanie wielu tanich źródeł światła (lasery półprzewodnikowe), jak i dzielenie wiązki z jednego źródła.

Technika świetlna pozwala wreszcie na realizację pamięci asocjacyjnych, które w odróżnieniu od adresowych wyszukują informację zgodnie z zadanymi jej cechami, a więc podobnie jak działa pamięć człowieka.

4.3. Klasyfikacja pamięci

Oceniając aktualny stan pamięci komputerowych mówiono o hierarchii tych urządzeń, przy czym podział następował tu według wartości podstawowych parametrów pamięci: pojemności, szybkości i ceny. Ograniczając się do pamięci optycznych

i uwzględniając wspomniany już wcześniej fakt, że optyczne pamięci dyskowe dominują wśród rozwiązań, które przetrwały próbę czasu, przedstawimy tu dwa sposoby klasyfikacji tych pamięci: jeden - odnoszący się do rodzaju zapisu i dotyczący głównie pamięci dyskowych oraz drugi - bardziej ogólny, przeprowadzany według zasad działania i zjawisk fizycznych wykorzystywanych w procesach zapisywania, pamiętania i odczytywania informacji.

Pierwsza klasyfikacja wyodrębnia:

- pamięci stałe, do których informację zapisuje wytwórca, a odbiorca może je tylko odtwarzać, tak jak przy płytach dźwiękowych,
- pamięci z jednokrotnym zapisem, do których użytkownik raz może zapisać swą informację, a następnie odczytywać ją wielokrotnie,
- pamięci wymazywalne, do których użytkownik może wielokrotnie zapisywać, odczytywać i wymazywać informację.

Druga klasyfikacja jest mniej spójna ze względu na różne kryteria stosowanych tu podziałów. Będzie to zatem raczej wyliczanie różnych realizacji pamięci optycznych znanych z literatury. Podziały te bowiem zachodzą na siebie i konkretne rozwiązanie należy przyporządkować określonym grupom.

Kryterium stosowanym również w pamięciach magnetycznych jest istnienie części ruchomych w urządzeniu, w tym przypadku chodzi o mechaniczne kierowanie wiązką światła lub nośnikiem informacji. Wśród rozwiązań z mechanicznym przesuwem znalazłyby się więc wszystkie dyski optyczne, które dalej można dzielić ze względu na wymiary geometryczne, a także pamięci z obrotowymi płaszczyznami odbijającymi wiązkę świetlną. Wśród pamięci bez części ruchomych są natomiast układy z deflektorami akustooptycznymi i elektrooptycznymi.

Inny podział można przeprowadzić ze względu na wykorzystywane zjawiska fizyczne i mechaniczne działania. Należałoby na przód wszystkim wymienić pamięci holograficzne, które ciągle rokują duże możliwości, aczkolwiek zadowalającego rozwiązania praktycznego nie udało się znaleźć (zob. następny podrozdział). Wśród tych pamięci też istnieją podziały według realizacji poszczególnych podzespołów, a więc źródła światła, deflektora, modulatora, ośrodka przechowywania hologramów i detektorów.

Wśród pamięci dyskowych możemy wyróżnić wytłaczane matrycą wzorcową (odpowiadające pamięciom stałym z pierwszej klasyfikacji) i dyski termiczne, w których w procesie zapisu wykorzystywana jest energia cieplna do zmiany struktury warstwy rejestracji. Wyróżniamy trzy rodzaje dysków termicznych:

- z wypalaniem dziurek w warstwie metalicznej zapisu,
- z wypukłościami na powierzchni warstwy,
- z odwracalnymi przemianami strukturalnymi w warstwie rejestracji.

Ten ostatni rodzaj może być stosowany do wielokrotnego zapisu i kasowania. We wszystkich tych przypadkach odczyt polega na wykorzystaniu różnicy współczynnika odbicia w miejscu, gdzie została zapisana informacja.

Również do wielokrotnego zapisu mogą być stosowane dyski termomagnetyczne, w których ośrodek rejestracji stanowi warstwa orientowana magnetycznie, przy czym wektor magnetyzacji jest równoległy lub prostopadły do płaszczyzny warstwy. Zapis odbywa się przez podgrzanie energią wiązki świetlnej w obecności pola magnetycznego. W procesie odczytu wykorzystuje się promień lasera o mniejszej mocy, w którym na skutek efektu Kerra zostaje po odbiciu skręcona płaszczyzna polaryzacji. Stosowane są również dyski, gdzie liniowo spolaryzowany promień przechodzi przez magnetooptyczny ośrodek rejestracji i na skutek efektu Faradaya również ma skręconą płaszczyznę polaryzacji.

Dyski te mogą być wielokrotnie kasowane i spełniają wszystkie funkcje rejestracji na taśmach czy dyskach magnetycznych.

Odrębna, jak wspomniano, jest optyczna pamięć asocjacyjna, która może być niezmiennicza względem przesunięcia (ang. shift invariance), co wymaga nieliniowych elementów dyskryminacyjnych [37].

4.4. Rys historyczny, stan badań i techniki

Idea optycznego zapisu i odczytu informacji cyfrowej nie jest nowa. W początkowym okresie rozwoju komputerów używano pamięci na lampach oscyloskopowych i informację przechowywano w postaci obrazu na ekranie tych lamp.

Jeszcze przed wynalezieniem lasera podejmowano prace nad skupieniem do tego celu światła niekoherentnego. Wraz z rozwojem techniki laserowej prace nad pamięciami optycznymi, zwłaszcza holograficznymi, podjęło wiele laboratoriów na całym świecie, obiecując uzyskanie wkrótce rewelacyjnych wyników. Nie było w drugiej połowie lat sześćdziesiątych większej firmy komputerowej, która nie prowadziłaby badań w tym zakresie. Wydawało się, że będzie można zrealizować pamięć spełniającą wszystkie wymagania komputera, a więc dużą pojemność, szybki dostęp i umiarkowaną cenę. Mówiło się nawet o zastąpieniu pamięci wewnętrznych i zewnętrznych jedną pamięcią holograficzną. Model firmy IBM [38] miał więc pojemność 10^8 bitów i czas dostępu 10 μs. We Francji Narodowe Centrum Badań Telekomunikacyjnych (CNET) opracowało urządzenie do zapisu i wyświetlenia informacji analogowej [39] z czasem dostępu do strony (jednej spośród 50 tysięcy) około jednej sekundy.

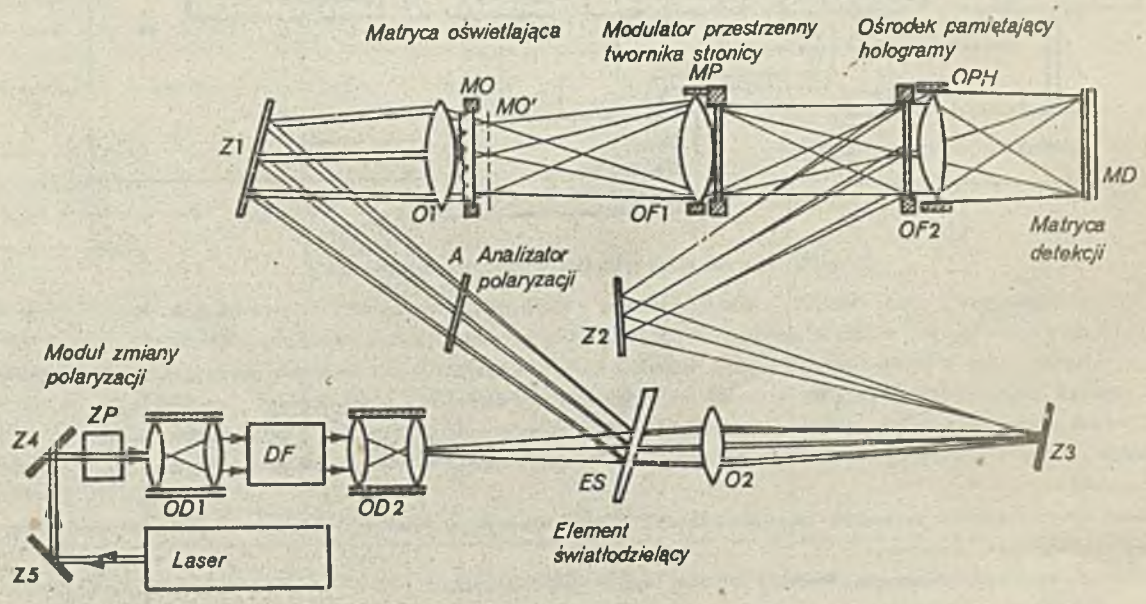
Jednakże opracowanych modeli czy prototypów nie udało się wprowadzić do produkcji. Niezawodność tych urządzeń okazała się niewystarczająca. Nie znaleziono odwracalnego materiału do przechowywania hologramów, który spełniałby wszystkie stawiane

mu wymagania. Wymagania na układy optyczne okazały się zbyt wysokie w stosunku do możliwości wykonawczych lub dopuszczalnych kosztów. W rezultacie większość ośrodków przerwała prace. Jeszcze w 1980 r. nakłady na te pamięci szacowano na ok. 10 mln dolarów, co stanowiło prawie trzecią część całego rynku urządzeń holograficznych. W roku 1985 obroty tego rynku [39] wzrosły natomiast do 82 mln dolarów, lecz pamięci holograficzne nie są wymieniane wśród produkowanych urządzeń.

W tym czasie głównie pod wpływem rozwoju przemysłu fonograficznego, udoskonalono technikę cyfrowego zapisu świetlnego która może być wykorzystana zarówno do produkcji płyt dźwiękowych czy wizyjnych wysokiej jakości, jak i pamięci, w pierwszym etapie stałych, o dużej pojemności i niezawodności. Już w połowie lat sześćdziesiątych szukano urządzenia do zapisu sygnałów wizyjnych, eksperymentując m. in. z układami elektrostatystycznymi (tzw. system CED). Najlepszym okazał się system Laser Vision (LV) zawierający miniaturowe wgłębienia na powierzchni dysku. W stosunku do układów taśmowych upraszczało to napęd, a wytłaczanie dysków nie było skomplikowane. Prace podjęło wiele firm. Aktualny standard opracowany został przez firmę Philips pod koniec 1978 roku. Stosowane są dyski o średnicy 30 cm (12 cali), rzadziej o średnicy 36 lub 20 cm (14 i 8 cali).

W roku 1982 firmy Philips i Sony opracowały nowy standard znany pod nazwą kompakt dysk (CD) o średnicy 12 cm (5¼ cala), który okazał się bestsellerem elektroniki użytkowej. Cena urządzenia odtwarzającego spadła z tysiąca do poniżej 300 dolarów z dalszą tendencją spadkową. Ten rodzaj dysków stosowany jest do zapisu stałej informacji cyfrowej (CD ROM).

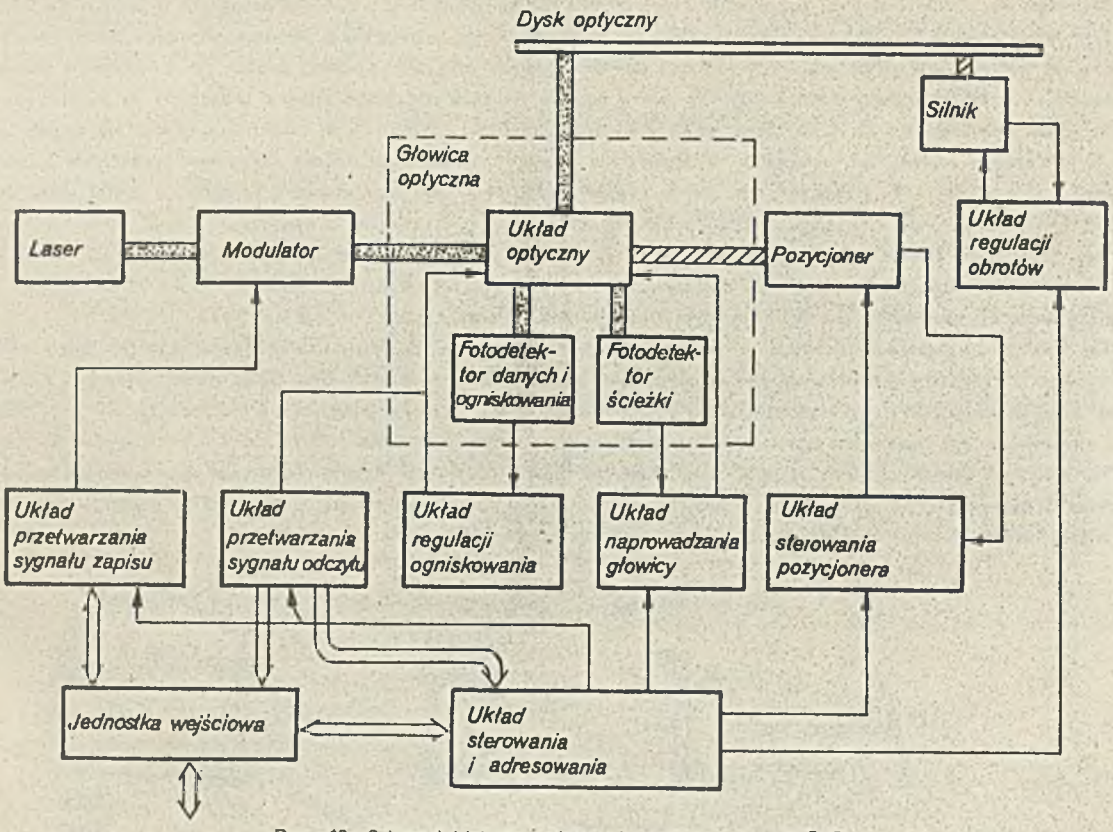
Chcąc zilustrować omawiane tendencje w rozwoju komputerowych pamięci optycznych przedstawiamy krótko dwa rozwiązania. Na rysunku 12 pokazany jest schemat układu optycznego projektu pamięci holograficznej [40]. Wiązka laserowa odchylana jest przez układ deflektora DF, a następnie dzielona w układzie ES. Część zapisana przechodzi przez matrycę MO oraz modulator MP i modulowana jest zapisywaną informacją. Następnie wraz z wiązką odniesienia pada na ośrodek przechowywania hologramów OPH, w którym obie wiązki interferują ze sobą powodując zapis hologramu. Przy odczycie pada tylko wiązka odniesienia, która odtwarza zapisany obraz na matrycy detektora. Jeżeli zachodzi potrzeba wymazania informacji, dokonuje się tego za pomocą wiązki odniesienia o zwiększonej mocy lub przy dłuższym czasie naświetlania.



Rys. 12. Schemat układu optycznego pamięci holograficznej z tomem płaskim: DF - zespół odchylający deflektora, OD1, OD2 - zespoły optyczne deflektora, OF1, OF2 - obiektywy transformacji Fouriera, O1, O2 - obiektywy kierujące wiązkami, Z1, Z2, Z3, Z4, Z5 - zwierciadło kierujące wiązkami światła

Jako deflektory odchylania stosuje się układy elektromechaniczne, akustooptyczne (opracowane w omawianym rozwiązaniu) i elektrooptyczne. Jako modulatory przestrzenne wykorzystuje się układy zrealizowane na ciekłych kryształach, ceramice ferroelektrycznej, kryształach elektrooptycznych, siarczku kadmu, a także modulatory membranowe, akustooptyczne i modulatory ze skanowaniem. Jako ośrodek przechowywania hologramów wykorzystuje się materiały magnetooptyczne (MnBi, TeO, stopy telluru), fotochromowe (szkła fotochromowe, $SiTiO_3$, hologenki alkaliczne, związki organiczne), kryształy elektrooptyczne ($LiNbO_3$, $LiTaO_3$, $Ba_2NaNb_5O_{15}$ i $BaTiO_3$), układy dielektryków i fotoprzewodników, układy ciekły kryształ - fotoprzewodnik i materiały półprzewodnikowe. Matryca fotodetektorów może zawierać elementy do odczytu bezpośredniego bądź gromadzące ładunek.

Rysunek 13 przedstawia natomiast schemat dyskowej pamięci optycznej. Podstawowym zespołem jest tu głowica optyczna, która służy zarówno do zapisywania, jak i odczytu informacji. Zawiera ona układ optyczny do skupiania wiązki laserowej i kierowania jej na dysk, a także detekcji wiązki odbitej. Ponadto znajdują się tu elementy sygnalizowania błędów ogniskowania i położenia głowicy. Ustawienie głowicy i jej odległości od dysku dokonuje się automatycznie.



Rys. 13. Schemat blokowy optycznej pamięci dyskowej [42]

Wiązka światła jest zmodulowana przy zapisie informacją z komputera, a następnie kierowana na dysk. Przy odczycie natomiast na dysk podawana jest wiązka o mniejszym natężeniu, które po odbiciu zależy od zapisanej informacji. Jest ona kierowana do detektora, stąd przez układ przetwarzania i jednostkę wejściową pamięci sygnał wyjściowy przesyłany jest do komputera. Układ regulacji prędkości zapewnia stałą prędkość transmisji danych. Dysk obraca się zwykle z prędkością do 30 obr./s.

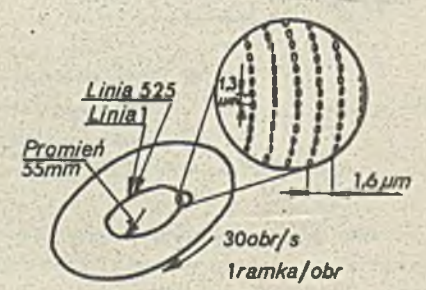
Ośrodek pamięciowy powinien odznaczać się dużą czułości, rozdzielczością, trwałością i odpowiednim stosunkiem sygnału do zakłóceń. Głowica powinna być lekka, co pozwoliłoby skrócić czas dostępu. W tym zakresie nie uzyskano na razie zadowalających wyników.

Zapis cyfrowy wybrano ze względu na wyjątkowo dobrą jakość odtwarzanej informacji. Rysunek 14 ilustruje makro i mikroskopowe wymiary dysku użytkowego [41].

Dyski do zapisu dźwiękowego i informacji cyfrowej mają analogiczną postać. Dodatkowo stosowane są różne kody korekcyjne, które obniżają częstotliwość występowania błędu z 10^{-4} do 10^{-9}, a nawet do 10^{-20}, co spełnia wszelkie wymagania w tym zakresie.

Stacje komputerowych dysków optycznych wyposażone są w odpowiednie interfejsy i odznaczają się krótszymi czasami dostępu aniżeli układy do odtwarzania dźwięku czy obrazu. Zwykle stosowane są tu układy z jednokrotnym zapisem, gdyż użytkownik przygotowuje informację, którą potem wielokrotnie chce odczytywać. Zapis taki powinien być prosty, nie wymagający dodatkowych procesów chemicznych.

Dyski umożliwiające wielokrotny zapis i kasowanie nie osiągnęły natomiast jeszcze etapu masowej produkcji. Najbardziej zaawansowane technicznie są materiały magnetooptyczne, a wśród nich cienkie warstwy ziem rzadkich i metali przejściowych.



Rys. 14. Wymiary makro- i mikroskopowe dysku wizyjnego

4.5. Pamięci optyczne oferowane na rynku

Jako przykłady dostępnych pamięci optycznych podawano do niedawna dyski o dużych średnicach (8-14 cali). W tablicy II przytoczono za pracą [42] parametry czterech takich rozwiązań. Widzimy, że są to dyski o pojemności 1-4 Gbajtów, czasach dostępu 85 do 250 ms i prędkości transmisji od 478 kb/s do 1,5Mb/s. Dla porównania dyski magnetyczne typu Winchester o zbliżonej średnicy mają podobnie pojemności, znacznie mniejszą gęstość ścieżek (31,5 na cm), a za tym - mniejszą gęstość informacji na jednostkę powierzchni (ok. 1,2 Mbit/cm^2 w porównaniu z 35 Mbit/cm^2 dla dysku Optimen). Krótszy jest natomiast czas dostępu (18 ms). Szybkość przesyłania informacji jest tu tego samego rzędu.

Tablica II. Parametry techniczne optycznych pamięci dyskowych z jednokrotnym zapisem

Firma	Optimen	Thomson-CSP	Storage Technology	NEC
Model	Optimen 1000	GD 1001	7640	
Pojemność, Gbajt	1	1	4	1,3
Średnica dysku	12"	12"	14"	30 cm
Średni czas dostępu, ms	157	200	85	250
Prędkość transmisji	625 kb/s	478 kb/s	1,5 Mbajt/s	810 kb/s
Współczynnik błędu				
- bez korelacji			10^{-5}	
- z korelacją	10^{-12}	10^{-12}	10^{-13}	10^{-12}
Prędkość obrotowa dysku, obr/min	1122	1122	1800	900
Liczba bajtów w sektorze	1024	1024		512
Liczba sektorów	25	25		69+6
Gęstość bitów/mm	570	570	1375	1575
Liczba ścieżek/mm lub odległość między nimi	570	570		1,57 µm
Materiał podłoża	szkło	szkło	Al	PMMA
Grubość podłoża, mm				1,2
Liczba laserów	1	1	2	1
Interfejs	SCSI	SCSI		SCSI zmodyfikowany
Masa	22,7 kg	25 kg		
Wymiary, cm (wysokość x szerokość x długość)	17,8x44,7x61	17,8x44,7x61	139,7x132x81,3	

Dysk magnetyczny o średnicy 5,25 cala ma natomiast pojemność 2-4-krotnie mniejszą przy gęstości zapisu ok. 3,4 Mbit/cm^2 i czasie dostępu 30 ms.

Ostatnio [43] pojawiły się również optyczne dyski pamięci stałych o średnicy 5¼ cala. Czterech wytwórców (w większości japońskich) oferuje 10 rodzajów stacji dyskowych o średnim czasie dostępu 0,5-0,7 s, a maksymalnym 0,65-2 s. Pojemność tych dysków wynosi 540 Mbajtów, a więc 1,5-raza więcej niż dysków magnetycznych o tych rozmiarach. Ceny stacji wahają się od 400 do 1795 dolarów, przy czym można na nich również odtwarzać dyski dźwiękowe o średnicy 120 mm. Interfejs większości tych stacji jest zgodny z szyną IBM PC, ale niektóre z nich mają interfejs SCSI. Dyski ROM kosztują od 5 do 25 dolarów.

Stacje dysków optycznych 5¼ cala z jednokrotnym zapisem wytwarza lub zamierza wytwarzać w 1987 r. [43] siedmiu wytwórców. Mają one mniejszą pojemność 115-400 Mbajtów i krótszy średni czas dostępu (0,1-0,22 s). Kosztują drożej: 1500-4380 dolarów, a dyski 40 do 225 dolarów. Stosowany interfejs to SCSI, ESDI lub własny. W przeciwieństwie do ROM nie ma tu jeszcze standardu formatu.

Dyski wymazywalne nie mają jeszcze dostępnych w handlu swych stacji napędowych. W końcu 1986 r. [43] przewidywano pojawienie się takich stacji (i dysków) w ciągu 1-4 lat. Wielu wytwórców demonstrowało już jednak prototypy. Zastosowano w nich środki magnetooptyczne, ze zmianą fazy i układ barwnik-polimer. Pierwsza z tych technologii jest najbardziej zaawansowana i takie firmy jak KerDix z Boulder i Plasmon Data Systems z San Jose oferują próbki ośrodka. Stosunek sygnału do

zakłócenia przekracza tu 45 dB, przy czym oferowane są też testery do oceny ośrodków. Ośrodki ze zmianą fazy to warstwy chalkonidbwe oparte na selenie i tellurze. Niektóre z tych ośrodków mają stosunek sygnału do zakłócenia 95 dB. Ostatnia technologia to organiczne pokrycie dysku złożone z dwóch warstw: elastomerowej i termoplastycznej.

Już po napisaniu, a przed opublikowaniem tej pracy, pojawiły się nowe dane [36] obejmujące ofertę 11 stacji pamięci stałych (CD ROM), przy czym firmy Hitachi i Sony przedstawiały po 2 modele. Pojemność tych dysków wahała się od 540 do 680 MB. Najczęściej spotykanymi tu interfejsami tyły PC i SCSI, a wymiary stacji zawierały się między 4x14x20 a 12x25x36 cm. W dwóch wypadkach podano ceny wyrobów - wynosiły one 600 (przy partii co najmniej 100 szt.) i 869 dolarów.

Drugą grupą oferowanych stacji dysków optycznych były duże urządzenia (tzw. "Juke-box") zawierające po 20 do 150 dysków o średnicach od 8 do 14 cali. Pojemność ich wynosiła od 30 do 1020 GB. Wszystkie pięć prezentowanych modeli (różnych firm) miało interfejs SCSI, większość miała też RS232. Wymiary oczywiście są duże - od 42x90x85 cm do 80x180x225 cm. Cena takiej stacji, zawierającej 66 dysków po 400 MB każdy, wynosiła poniżej 10 tysięcy dolarów.

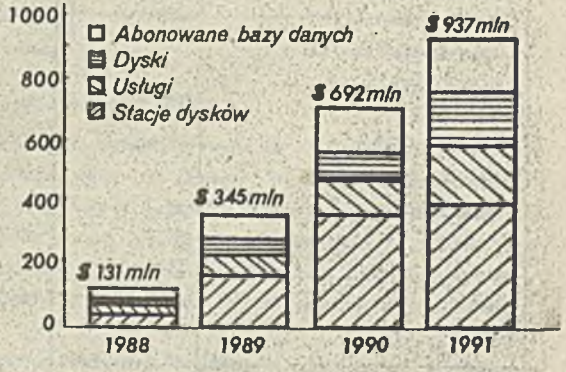
Największy natomiast wybór jest wśród stacji dysków z jednokrotnym zapisem. Aż dziewiętnastu producentów (w tym tylko jeden europejski - ATG) podawało parametry 23 modeli o pojemności od 120 MB do 34 GB, średnio 662 MB. Prawie 70% tej oferty to dyski o średnicy 5,25 cala, a ponad 80% ma interfejs SCSI. Czas dostępu waha się tu od 60 do 400 ms, średnio 123 ms. Wymiary stacji zawierają się pomiędzy 4x14x20 cm a 18x45x61 cm.

Pojawiły się też pierwsze handlowe rozwiązania dysków wymazywalnych. Jest to model ME D5010 firmy Olympus o średnicy 5,25 cala i pojemności 240 MB wykorzystujący technikę magnetyczno-optyczną ze zmianą fazy. Produkcja na szerszą skalę przewidywana była na koniec 1988 roku. Inne firmy, jak Sharp i Sony, też mają pierwsze modele, lecz nieco mniej zaawansowane. Dyski magnetyczno-optyczne wytwarzają też japońskie firmy Daicel i Sumitomo, amerykańskie 3M, Verbatim (która pierwsza opracowała model o średnicy 3,5 cala i pojemności 50 MB), Dupont i Dei, niemiecka Hoechst i angielska Plasmon.

4.6. Perspektywy rozwoju

Jesteśmy w okresie, kiedy dyskowe pamięci optyczne niedawno pojawiły się na rynku i rozwój ich obecnie jest znacznie szybszy aniżeli pamięci o ustabilizowanej pozycji.

Rysunek 15 pokazuje [44] przewidywania firmy Input z Mountain View, według których najbliższe lata przyniosą taki rozwój pamięci CD ROM, jaki miał miejsce na początku lat osiemdziesiątych w dziedzinie komputerów osobistych. Chociaż w roku 1987 sprzedaż tych pamięci nie przekroczy 30 mln dolarów, prognozy na rok 1991 mówią o blisko miliardowych obrotach. Dzieje się tak dlatego, że technika CD ROM jest już obecnie tańsza aniżeli zbiory na papierze lub mikrofilmie, a pod koniec lat osiemdziesiątych stosunek kosztów za jednostkę przechowywanej informacji przekroczy 100.



Rys. 15. Przewidywania rozwoju optycznych pamięci dyskowych typu CD ROM (CD ROM - Compact Disk Only Memory)

Inne przewidywania [45] mówią o 2 mld dolarów za dyskowe pamięci optyczne w 1991 r., przy czym obejmuje to wszystkie trzy rodzaje dysków optycznych. Według tych prognoz pamięci typu ROM stanowić będą ilościowo 42% wszystkich stacji (442 tysiące), a wartościowo 9% (174 mln dolarów), pamięci z jednokrotnym zapisem odpowiednio 33% (345 tys), 74% (1,5 mln dolarów), a pamięci wymazywalne 25% (260 tys) i 17% (345 mln dolarów).

Wśród pamięci stałych 95% stanowić będą dyski typu kompakt, ze względu na opanowaną technologię. Wśród pamięci o jednokrotnym zapisie najpierw rozwijać się będą systemy o pojemności od 1 do 3 Mbajtów, a później systemy o pojemności mniejszej od 1 Mbajtu. Pamięci wymazywalne rozwiną się pod koniec omawianego okresu i pojemność ich nie przekroczy 1 Mbajtu.

Jeszcze bardziej optymistyczna jest trzecia prognoza opracowana przez First & Sullivan Ltd [46] (nb. jeden egzemplarz tego raportu kosztuje 1900 dolarów), przewiduje oma mianowicie skok z 14 mln dolarów w 1986 r. do 1,74 mld dol. w roku 1988 i 2,5 mld w roku 1990.

Tak więc najpostrzeźniejsze nawet prognozy mówią o znacznym, wynoszącym kilkadziesiąt procent w skali rocznej rozwoju dyskowych pamięci optycznych w najbliższych latach. Przewidywania te oparto na dotychczasowym rozwoju technologii, a nie można wykluczyć przyspieszenia tego tempa dzięki nowym materiałom, technologiom i rozwiązaniom systemowym.

Ostatnio [47] pojawiły się doniesienia o pierwszych wymazywalnych pamięciach optycznych. Według tych źródeł stacja dysków optycznych firmy Tandy pozwala na zapis i odczyt na dysku o średnicy 5,25 cala przy zastosowaniu

polimerowego ośrodka ze zmianą fazy. Przewidywano pojawienie się tego wyrobu w sprzedaży pod koniec 1984 roku. Prace nad podobnymi urządzeniami prowadzą też amerykańskie firmy Maxtor i Optical Storage International oraz japońskie Sony, Nikon, Sharp i Hitachi.

Inne rodzaje pamięci optycznych, poza dyskowymi, aczkolwiek nie osiągnęły tego stopnia rozwoju, by stać się produktami rynkowymi, znajdują się często w stadium udoskonalanych prototypów i mogą pojawić się z konkurencyjnymi parametrami.

5. OPTOELEKTRONICZNE URZĄDZENIA WEJŚCIA I WYJŚCIA

Urządzenia do wprowadzania i wyprowadzania informacji stanowią najbardziej zróżnicowaną pod względem konstrukcji i technologii dziedzinę techniki komputerowej. Wynika to z dużej liczby rodzajów tych urządzeń, stale zresztą wzrastającej oraz ustawicznego wzrostu wymagań odnoszących się do parametrów technicznych i jakościowych.

W tradycyjnych systemach komputerowych, służących głównie do przetwarzania danych i obliczeń numerycznych, istniała potrzeba wprowadzania dużej liczby danych alfanumerycznych i drukowania wyników na papierze. Potrzeby te były zaspokajane za pomocą czytników kart lub taśmy papierowej oraz szybkich drukarek uderzeniowych. Pojawienie się minikomputerów, a potem komputerów personalnych stworzyło nowe możliwości zastosowań techniki komputerowej, a jednocześnie spowodowało konieczność rozwoju nowych urządzeń wejścia-wyjścia. Szczególne znaczenie miało zastosowanie monitora ekranowego, który wraz z klawiaturą pełni funkcję urządzenia do wprowadzania informacji, a przy tym jest wykorzystywany jako urządzenie wyjściowe i służy do prowadzenia dialogu z maszyną. Oprócz klawiatury do wprowadzania danych służą rozmaitego rodzaju manipulatory (pióra świetlne, mysz i inne).

Zastosowanie monitorów umożliwiło również wyprowadzanie informacji nie tylko w postaci alfanumerycznej, ale i graficznej Związany z tym rozwój grafiki komputerowej i nowych zastosowań komputerów spowodował potrzebę opracowania urządzeń do wprowadzania informacji graficznej, a także rozszerzenia tradycyjnych funkcji drukarek komputerowych o możliwości drukowania obrazów. Obecnie więc nawet stosunkowo proste systemy komputerowe na potrzeby biurowe czy projektowania wspomaganego komputerowo dysponują wieloma urządzeniami we-wy.

Jeszcze większe zróżnicowanie tych urządzeń istnieje w systemach pełniących wyspecjalizowane funkcje w różnych dziedzinach (np. w przemyśle, medycynie, wojsku, telemetrii itd.). Powstają nowe urządzenia pozwalające na bezpośrednie wprowadzenie specyficznych danych do komputera, który je przetwarza, a wynik tego przetwarzania jest kierowany do specjalizowanego urządzenia wyjściowego. Jako przykład wskażmy tu komputerowe systemy przygotowania fotoskładu, w których wykorzystuje się po stronie wejścia urządzenia do automatycznego czytania maszynopisów, a na wyjściu - urządzenia do naświetlania tekstu i obrazów na papierach lub błonach fotograficznych.

Elementy i podzespoły optoelektroniczne już od wczesnego rozwoju komputerowych urządzeń we-wy znajdowały w nich zastosowanie. W czytnikach taśmy papierowej lub kart perforowanych wykorzystywano elementy fotopółprzewodnikowe, a w pulpitach sterowniczych komputerów powszechnie stosowano różnego rodzaju wskaźniki optyczne. Udział optoelektroniki wzrastał w miarę rozwoju techniki komputerowej i jej zastosowań. Urządzeniami optoelektronicznymi są: monitor ekranowy, liczne urządzenia służące do odczytu informacji, a także urządzenie do drukowania informacji na papierze lub zapisu na błonie fotograficznej. Spektakularnym przykładem rozwoju optoelektronicznych urządzeń wyjściowych mogą być drukarki laserowe.

Ograniczona objętość niniejszego opracowania nie pozwala na pełniejsze przedstawienie problematyki optoelektronicznych urządzeń służących do wprowadzania i wyprowadzania informacji. Ograniczono się zatem do omówienia tych urządzeń, których rozwój osiągnął już stan pewnego ustabilizowania i których zastosowania mają charakter bardziej uniwersalny.

5.1. Urządzenia wejściowe

W systemach komputerowych służących do automatyzacji pracy biurowej lub komputerowego wspomagania projektowania zachodzi potrzeba wprowadzania do komputera informacji cyfrowej odwzorowującej zapisane w trwały sposób (np. na papierze) znaki, rysunki, obrazy, a w wielu wyspecjalizowanych systemach dokonuje się zamiany na postać cyfrową obrazów rzeczywistych obiektów. Zależnie od rodzaju badanego obrazu stosuje się różne metody konwersji, wśród których główną rolę odgrywają metody optoelektroniczne, szczególnie w czytnikach znaków i czytnikach obrazów.

Urządzenia te pracują na zasadzie zamiany sygnału świetlnego, powstającego w wyniku odbicia światła od badanego znaku lub obiektu na impuls elektryczny. Zamiany takiej można dokonać wykorzystując: fotoczułe elementy półprzewodnikowe, lampy analizujące (widikony), przetworniki CCD. Oprócz przetwornika obrazu czytniki zawierają następujące zespoły: źródła światła oświetlające badany obraz, układ optyczny kierujący odbite światło do przetwornika pamięci służącej do przechowywania odczytanej informacji binarnej, układ analizujący i przetwarzający informację.

Jeżeli przetwornikiem jest pojedynczy element światłoczuły (np. fotodioda), obraz jest poddawany stopniowej analizie, np. za pomocą skanowanej wiązki światła laserowego lub światła przechodzącego przez otwór znajdujący się w wirującej tarczy. Przy użyciu jako przetwornika linijki fotodetektorów, obraz jest przesuwany stopniowo względem tej linijki i przetwarzany jest od razu cały rząd punktów obrazu. Metoda ta szczególnie nadaje się do czytników znaków. Zastosowanie matrycy fotodetektorów lub lampy analizującej umożliwia jednoczesne przetwarzanie całego obrazu lub dużej jego części, co pozwala uzyskać dużą szybkość przetwarzania i dlatego sposób ten jest stosowany w czytnikach obrazu.

5.1.1. Przetworniki obrazów [50, 51]

● Fotodetektory półprzewodnikowe. Do detekcji sygnału optycznego wykorzystuje się fotorezystory, fotodiody i fotoogniwa. Fotorezystory wytwarzane są z takich materiałów, jak siarczek kadmu, siarczek ołowiu, selenek ołowiu i mogą być stosowane w przypadku, gdy szybkość przetwarzania nie jest duża. Fotoogniwa ze względu na stosunkowo duże rozmiary są wykorzystywane jako pojedyncze elementy fotoczułe. Największe zastosowanie mają fotodiody, które wytwarza się w postaci pojedynczych diod, linijek i matryc diodowych. Produkowane są liczne typy takich elementów, przeważnie z Ge, Si, GaAs, dostosowane widmowym zakresem pracy (od ultrafioletu do bliskiej podczerwieni), szybkością i czułością przetwarzania do różnych celów. Wykonuje się więc nie tylko fotodiodę ze złączem p-n, ale także z barierą Schottky'ego, heterozłącza, diody ostrzowe i diody p i n. Te ostatnie odznaczają się dużą prędkością działania. Bardzo dużą czułość w połączeniu z dużą szybkością uzyskuje się w fotodiodach lawinowych. Wykonuje się również fototranzystory oraz układy fotodetektorów wyposażone w dodatkowe elementy wzmacniające lub przełączające.

● Lampy analizujące (widikony). Dzięki postępowi, który nastąpił w konstukcjach wyrzutni elektronowej oraz układów sterowania promieniem elektronowym, współczesne lampy analizujące odznaczają się wysoką rozdzielczością (ponad 600 linii) przy znacznie zredukowanych wymiarach pola analizującego. Średnica lamp analizujących zmalała dwukrotnie i osiągnęła wartość 13 mm. Oprócz zmian konstrukcyjnych następuje ciągły postęp w dziedzinie warstw fotoczułych płytki sygnałowej. Zamiast trójsiarczku antymonu stosowanego w tradycyjnym widikonie wykorzystuje się warstwy z tlenku ołowiu, selenku cynku, selenku kadmu, heterozłączowe warstwy z selenu, arsenu i telluru, wielodiodowe mozaiki krzemowe. Warstwy te umożliwiają uzyskanie optymalnych dla danych potrzeb właściwości, tzn. czułości, rozdzielczości, bezwładności i wartości prądu ciemnego. Na przykład bardzo szeroką charakterystykę widmową, od ultrafioletu do podczerwieni i dużą czułość mają mozaiki krzemowe, a dużą czułość, równomierną charakterystykę widmową w zakresie światła widzialnego i bardzo małe prądy ciemne ma selenek kadmu.

● Matryce CCD. Działanie przetwornika CCD (charge coupled device) polega na wykorzystaniu magazynowania ładunku elektrycznego i jego przesuwaniu w układzie liniowo połączonych elementarnych komórek półprzewodnikowych (typu MOS). Matryca złożona jest z wielu takich linii i ładunek zgromadzony w wyniku oświetlenia sprzężonej z tymi komórkami warstwy fotoczułej jest przesuwany przez podanie do komórek odpowiednich impulsów napięciowych. Pomimo, że czułość przetworników CCD, a także ich rozdzielczość są gorsze niż lamp analizujących, mają one wiele zalet: nie wymagają wysokiego napięcia, cewek i układów odchylających itd., a przede wszystkim odznaczają się idealną geometrią i stabilnością położenia obrazu. Należy przypuszczać, że dalszy postęp w tej dziedzinie doprowadzi do poprawy parametrów przetworników CCD, a także wyeliminuje, trudne obecnie do uniknięcia, uszkodzenia komórek podczas procesu produkcji. Osiągana obecnie wielkość przetworników CCD wynosi 2048x2048. Powierzchnia fotoczuła zajmuje wtedy obszar 55x66 mm. Zakres widmowy przetworników CCD jest bardzo szeroki (od 100 do 1100 nm).

5.1.2. Czytniki znaków

Czytniki znaków służą do odczytu znaków zapisanych w określonej konwencji. Mogą to być znaki literowe i cyfrowe, znaki specjalne, kody graficzne itd. Zależnie od rodzaju znaków czytniki znaków dzieli się na:

- czytniki kodów kreskowych,
- czytniki znaczników,
- czytniki znaków alfanumerycznych.

Najliczniejszą kategorię stanowią czytniki znaków alfanumerycznych, których dalszy podział przedstawiono na rys. 16

● Czytniki kodów kreskowych - OBR (optical bars reader) - rozpoznają kod kreskowy, lokalizujący położenie i grubość kresek. Stosowane są do automatyzacji operacji kasowych w domach towarowych.

● Czytniki znaczników - OMR (optical mark reader) - lokalizują pozycję znacznika znajdującego się na dokumencie i korelują je z przypisanym znacznikiem. Mają zastosowanie przy sprawdzaniu testów, dokumentów, rachunków.

● Czytniki znaków alfanumerycznych - OCR (optical character reader) - służą do odczytu znaków zapisanych na maszynie często w określonym standardzie np. OCR-A, OCR-B, Courier 12 lub ręcznie. Zależnie od klasy zastosowań odczytują informację zapisaną na różnych formatach papieru i z różną prędkością. Spotykamy więc następujące ich rodzaje:

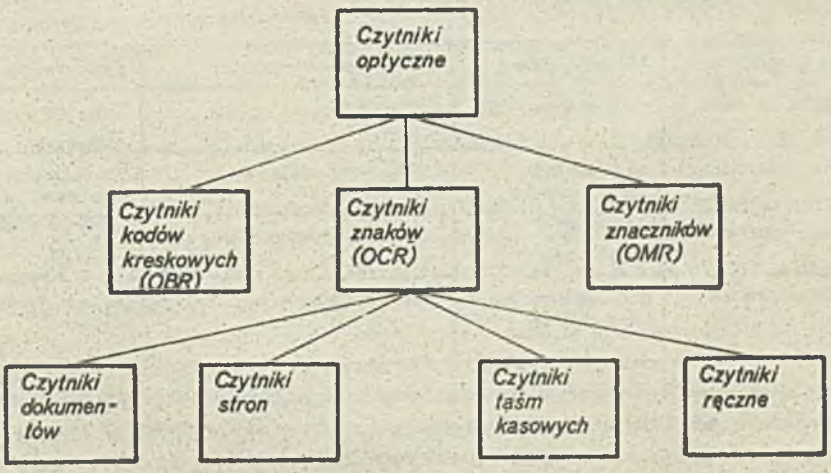
- czytniki dokumentów odczytują informację z dokumentów o niewielkim formacie (rachunków, kuponów). Jednorazowo odczytują 5 linii, znaki muszą mieć określony standard i określone położenie,

- czytniki stron służą do odczytu dokumentów o różnych rozmiarach, różnych znakach i dowolnym położeniu znaków. Zastosowanie znajdują w biurach, poligrafii itd.,

- czytniki taśm kasowych są to wyspecjalizowane urządzenia do odczytu utargów z kas automatycznych,

- czytniki podręczne służą do odczytu kwitów, metek i innych dokumentów, przy czym jednorazowo czyta się tylko jedną linię tekstu.

Rozpoznawanie znaków w czytnikach znaków może odbywać się na różnej zasadzie: porównania tablic wzorcowych, analizy przerw, wykrywania pewnych własności, testowania krzywizny itp. Układ służący do tego celu (ewentualnie mikrokomputer) znajduje się w czytniku. Jako przetworniki obrazów najczęściej są stosowane linijki lub matryce fotodiod lub elementy CCD.



Rys. 16. Optyczne czytniki informacji znakowej

5.1.3. Optyczne czytniki obrazów

Optyczne czytniki obrazów umożliwiają odwzorowanie w pamięci elektronicznej obrazu podzielonego na pewną liczbę punktów elementarnych (pikseli). Zapamiętać można stopień szarości poszczególnego punktu. Czytniki stanowią zwykle jedno z urządzeń systemu umożliwiającego przetwarzanie obrazów; system taki obejmuje ponadto komputer do przetwarzania obrazów, monitor ekranowy i urządzenia do trwałego zapisania obrazów. Podstawowym zespołem czytnika jest zwykle kamera zawierająca lampę analizującą lub matrycowy przetwornik CCD. Możliwe jest również stosowanie innego sposobu odczytu obrazu, o czym wspomniano w rozdziale 2.1. Zależnie od zastosowań obraz może być badany w różnych zakresach światła (tabl. III), stosuje się wtedy odpowiednie przetworniki obrazu i zależnie od zastosowań wykonuje różnego rodzaju operacje przetwarzania obrazów.

Tablica III. Urządzenie do wprowadzania informacji graficznej

Światło	Metoda konwersji	Zastosowania
Ultrafiolet	lampa analizująca, przetwornik CCD	medycyna, badania materiałowe, astronomia
Widzialne	lampa analizująca, przetwornik CCD, laser, matryca fotodiod	rozpoznawanie i przetwarzanie obrazów, roboty
Podczerwień	lampa analizująca, przetwornik CCD z warstwą pyroelektryczną	wojsko, badania materiałowe

5.2. Urządzenie do zobrazowania informacji

Najpopularniejszym urządzeniem służącym do wizualnego przedstawienia informacji alfanumerycznej lub graficznej są monitory ekranowe wykorzystujące lampy obrazowe (kineskopy) monochromatyczne lub kolorowe. Jednak w wielu zastosowaniach, np. wojskowych, przemysłowych, a także w komputerach osobistych przenośnych, lampa jest zastępowana ekranem płaskim (ciekłokrystalicznym, plazmowym lub fluoroscencyjnym). Urządzenia takie zabierają znacznie mniej miejsca, nie wymagają dużych napięć zasilających, są odporne na trudne warunki pracy, a przy tym odznaczają się takimi korzystnymi cechami, jak duży kontrast, duża jasność świecenia i duży kąt obserwacji. Płaskie ekrany i tablice wyświetlające dorównują już pod względem liczby wyświetlanych znaków typowym monitorom (25x80 znaków), chociaż przeważnie są stosowane tablice o mniejszej liczbie znaków.

W kalkulatorach, urządzeniach zewnętrznych, pulpitach operacyjnych itd. są stosowane moduły wskaźników wyświetlających, zawierające zwykle jeden rząd znaków. Do ich budowy wykorzystuje się przeważnie diody świecące i ciekłe kryształy.

Na drugim biegunie znajdują się wielkie ekrany służące do zobrazowania informacji prezentowanej podczas konferencji, do celów reklamowych, w wojsku itd. Istnieje wiele sposobów ich realizacji i trudno obecnie wskazać, które z nich będą miały przewagę. Podobne funkcje co wielkie ekrany mogą spełniać projektory sterowane przez komputer.

Podział i zastosowanie urządzeń służących do zobrazowania informacji przedstawiono w tabl. IV.

Tablica IV. Podział urządzeń do zobrazowania informacji

Nazwa urządzenia	Wielkość obrazu	Główny sposób realizacji	Zastosowanie
Wskaźniki	rząd znaków	LCD, LED	kalkulatory, pulpity operacyjne
Tablice wyświetlające	od kilkunastu do 25 rzędów znaków	LCD, ekran plazmowy	komputery przenośne wojskowe, przemysłowe
Monitory ekranowe	kilkadziesiąt rzędów znaków	lampa elektronopromieniowa	komputery wszystkich rodzajów
Wielkie ekrany	przekątna ekranu kilka metrów	brak standardowych rozwiązań	wojsko, zarządzanie, reklama

W dalszych rozdziałach przedstawiono krótką charakterystykę środków technicznych służących do zobrazowania informacji. Ograniczono się do omówienia najczęściej spotykanych rozwiązań, a przy tym należy dodać, że technika zobrazowania informacji jest dziedziną rozwijającą się szczególnie intensywnie. Prowadzone są liczne prace zmierzające nie tylko do poprawienia parametrów obecnie istniejących urządzeń, ale mające na celu wprowadzenie nowych jakości, np. zobrazowania przestrzennego.

5.2.1. Wskaźniki i tablice wyświetlające [52]

Wskaźniki z diodą świecącą (LED). Wskaźniki te są wykonywane ze złącz elektroluminescencyjnych w kształcie segmentów (wąskich prostokątów lub mozaiki kropek (małych kółek)), których odpowiednie kombinacje pozwalają na uzyskanie pożądanych znaków. Matryce kropkowe są bardziej uniwersalne, gdyż dzięki nim można otrzymać większą różnorodność znaków i elementy graficzne. Podstawową zaletą diod są: możliwość wykorzystania ich w warunkach małego oświetlenia i małe czasy przełączeń pozwalające na ich multipleksowe sterowanie. Dzięki tej cesze można zmniejszyć znacznie moc potrzebną do ich wysterowania i liczbę przewodów doprowadzających, a także poprawić czytelność obrazu dzięki większej intensywności świecenia przy zasilaniu impulsowym. Wskaźniki diodowe świecą zwykle w kolorze czerwonym, wytwarzane są również diody promieniujące światło pomarańczowe, żółte i zielone.

Wskaźniki wykonuje się w postaci modułów 4- lub 8-znakowych, przy czym znaki mogą być złożone z 7 lub 16 segmentów albo mozaiki kropek 5x7. Wysokość znaku wynosi od ok. 3 do 20 mm. Często moduły są wyposażone w układy sterujące diodami, a nawet pamięci i mikroprocesory, które pozwalają na programowe sterowanie modułu.

Wyświetlacze ciekłokrystaliczne (LCD). Wyświetlacze tego rodzaju, podobnie jak diody świecące, są stosowane w wielu wyrobach powszechnego użytku. Ich zaletą jest mała moc potrzebna do wysterowania, wymagają jednak dobrego oświetlenia, aby uzyskać zadowalający kontrast (lub też zastosowanie elementów transmisyjnych z dodatkowym źródłem światła, co jednak niweczy zaletę małej mocy). Jako materiał krystaliczny stosuje się obecnie przeważnie nematyki o teksturze skręconej, skręcające płaszczyznę polaryzacji światła o 90°. Ciekły kryształ jest umieszczony między dwoma polaryzatorami, obróconymi względem siebie o 90°, światło ulega polaryzacji po przejściu przez pierwszą płytkę i przechodzi przez ciekły kryształ, a następnie przez drugą płytkę, potem ulega odbiciu i wraca dając jasne pole widzenia. Skręcenia płaszczyzny polaryzacji w nematyku można uniknąć przez przyłożenie pola elektrycznego do tego materiału za pomocą przezroczystych elektrod, gdyż powoduje to

wydłużenie molekuł nematyka równolegle do kierunku pola. Następuje wówczas rozproszenie światła w drugim polaryzatorze, powierzchnia elektrody jest więc ciemna.

Wskaźniki wykorzystujące to zjawisko dają jednak niezadowalający kontrast i czas przełączenia, a także niekorzystny kąt optymalnego widzenia. Poprawę tych parametrów można uzyskać przez lepsze zdefiniowanie progowego napięcia aktywizującego kryształ, co otrzymuje się w przypadku obrotu płaszczyzny polaryzacji o 160 do 270° ("superskręcony nematyk"). Innym ulepszeniem wskaźników ciekłokrystalicznych jest konstrukcja dwuwarstwowa GH (guest-host, gość i gospodarz) utworzona z ciekłego kryształu "gospodarza" i barwnika organicznego "gościa", który nadaje kryształowi właściwości dwubarwne. Materiał jest kolorowy gdy jest oglądany wzdłuż osi optycznej (a tak się go właśnie sytuuje). W wyniku podania napięcia molekuły barwnika obracają się razem z molekułami nematyka, co powoduje zniknięcie zabarwienia pod elektrodami. Wytwarzanych jest kilka rodzajów wskaźników wykorzystujących to zjawisko. W połączeniu z kolorowym tłem dają one różne efekty kolorystyczne, np. wielokolorowe znaki.

Podobnie jak diody świecące, wskaźniki ciekłokrystaliczne wykonuje się w postaci modułów zawierających znaki złożone z segmentów lub kropek. Wysokość znaków sięga 25 mm, a ich liczba w rzędzie od 2 do 20. Moduł może zawierać kilka rzędów znaków.

- Ekrany plazmowe. Wskaźniki takie pracują na zasadzie zjawiska wyładowania jarzeniowego w plaźmie, które następuje pod wpływem napięcia przekraczającego napięcie zapłonu. Ekran składa się z dwu płytek szklanych z naniesionymi przezroczystymi elektrodami paskowymi, usytuowanymi na jednej płytce w kierunku X, a na drugiej w kierunku Y. Między płytkami znajduje się gaz, który zaczyna świecić na przecięciu dwu pobudzonych elektrod. Barwa świecenia zależy od składu gazu i jest czerwono-pomarańczowa dla mieszaniny helu i neonu. Ekrany mogą mieć różne wielkości, aż do wielkości odpowiadającej lampowemu monitorowi, tj. 512x512 elektrod X, Y. Często układ sterujący scala się razem ze wskaźnikiem. Napięcie pracy wynosi ok. 180 V.

- Wskaźnik fluorescencyjny. Działanie polega na wysyłaniu światła przez anodę lampy próżniowej pokrytej luminoforem przy jej bombardowaniu elektronami emitowanymi z katody. Wskaźnik składa się z segmentów, przed którym jest umieszczona katoda wykonana w postaci drobnoziarnistej siatki. Świecenie następuje po przyłożeniu napięcia do wybranych segmentów (kilkadziesiąt woltów), barwa jest zielona lub zielononiebieska. Wskaźniki fluorescencyjne wykonuje się w postaci modułów zawierających pojedynczy rząd znaków lub kilka rzędów, przy czym liczba znaków może wynosić do 40. Wysokość znaków sięga od kilka do kilkunastu mm. Układy sterujące często połączone są bezpośrednio ze wskaźnikiem. W porównaniu ze wskaźnikiem plazmowym wskaźniki fluorescencyjne emitują jaśniejsze światło, mają szerszą charakterystykę widmową (co daje możliwość filtrowania barw), wymagają mniejszego napięcia zasilającego. Z kolei wskaźniki plazmowe odznaczają się mniejszym czasem przełączania, co umożliwia budowę większych ekranów za pomocą multipleksowanych modułów plazmowych.

- Wskaźniki elektroluminescencyjne. Zasada ich pracy polega na świeceniu cienkiej warstwy luminofonu umieszczonej między pobudzanymi napięciowo elektrodami, z których jedna przepuszcza światło. Wadą tych wskaźników są duże straty mocy przy zmianie napięcia. Obecnie jednak ich znaczenie wzrosło, gdyż dzięki tej technice osiągnięto duży postęp w budowie wskaźników wielokolorowych [53].

Prace nad wskaźnikami i ekranami kilkukolorowymi, a nawet o pełnej gamie kolorów są prowadzone we wszystkich opisanych wyżej dziedzinach wyświetlania, a niektóre rozwiązania osiągają już stadium produkcji seryjnej. Dąży się również do uzyskania obrazów przestrzennych za pomocą konstrukcji hybrydowych - przełączników ciekłokrystalicznych i lampy elektronopromieniowej.

5.2.2. Monitory ekranowe

Monitory ekranowe wykorzystujące lampę obrazową są obecnie powszechnie stosowanym urządzeniem komputerowym. Wytwarzane są w licznych wersjach, różniących się rodzajem kineskopu (kolorowy, jednobarwny), rozdzielczością, rozmiarem ekranu, sposobem formowania znaków i obrazów, możliwościami lokalnego przetwarzania danych, oprogramowaniem itd.

Stosuje się więc monitory monochromatyczne (kolor zielony, bursztynowy, szary) i kolorowe o rozmiarze ekranu od 12 do 20 cali i liczbie punktów wyświetlanych dochodzącej do 2048x2048. W pierwszych monitorach (alfaskopach) wyświetlanie znaków następowało w wyniku pobierania zgodnie z przesłaną informacją znaków zapisanych w pamięci stałej. Obecnie przesyłanie sygnału do lampy ekranowej odbywa się po uprzednim zapełnieniu pamięci obrazu, w której obraz jest pamiętany w postaci tabeli punktów zawierającej informację dotyczącą jego stopnia starości lub koloru (technika rastrowa). Oprócz tej metody jest stosowana technika wektorowa, polegająca na bezpośrednim sterowaniu promieniem elektronowym.

Postęp w dziedzinie monitorów ekranowych idzie dwutorowo. Z jednej strony jest to doskonalenie parametrów lamp kineskopowych i wprowadzanie nowych konstrukcji (można oczekiwać pojawienia się ekranów o spłaszczonej konstrukcji po udanych próbach zbudowania przez firmy Siemens i Matsushita kolorowych lamp telewizyjnych, a z drugiej - opracowywanie specjalizowanych procesorów graficznych. Pozwalają one na znaczne odciążenie komputera od sterowania monitorem, zwiększają możliwość monitora jako urządzenia stosowanego np. w systemach CAD, ułatwiają kontakt użytkownika z monitorem poprzez możliwość wykorzystania języków wyższego rzędu itd. Do takich celów produkowane są obecnie specjalizowane procesory

monitora jako urządzenia stosowanego np. w systemach CAD, ułatwiają kontakt użytkownika z monitorem poprzez możliwość wykorzystania języków wyższego rzędu itd. Do takich celów produkowane są obecnie specjalizowane procesory o bardzo wielkiej skali integracji.

5.2.3. Wielkie ekrany i projektory ekranowe

Aby otrzymać duże formaty zobrazowania informacji, stosuje się różnorodne metody. Wysoką jakość odwzorowania (barwne obrazy) uzyskuje się w projektorach telewizyjnych. Obraz otrzymuje się przez nałożenie na ekran obrazów z trzech sterowanych lamp projekcyjnych R, G, B. Przekątna ekranu wynosi ponad 1 m (np. firmy Sony 50 cali).

Jednokolorowe ekrany można wykonać ze wskaźników plazmowych. Firma Displays Inc. oferuje np. ekran złożony z 48 modułów, każdy zawierający dwa rzędy 20 znaków. Znak jest utworzony z matrycy 5x7 punktów, a jego wysokość wynosi 1 cal, co pozwala na łatwy odczyt z odległości ok. 20 m. Rozmiar ekranu wynosi 40x64 cali.

Innym sposobem jest zastosowanie kluczy świetlnych zrobionych z elementów ciekłokrystalicznych. W opracowywanym przez firmę From-Lyte Systems [54] ekranie o przekątnej 2,44 m zastosowano je łącznie ze światłowodami, które kierują do nich światło zielone, czerwone i niebieskie, co umożliwia otrzymanie obrazów kolorowych.

5.3. Urządzenia wyjściowe

W niniejszym rozdziale omówimy optoelektroniczne urządzenia służące do trwałego zapisu znaków i obrazów na papierze lub materiałach fotograficznych. Scharakteryzujemy najważniejsze rodzaje takich urządzeń, a nieco więcej miejsca poświęcimy drukarkom laserowym ze względu na ich szczególnie intensywny rozwój.

5.3.1. Podstawowe rodzaje optoelektronicznych urządzeń wyjściowych

Głównym czynnikiem wpływającym na budowę i parametry urządzenia do zapisu znaków lub obrazów jest rodzaj materiału, na którym dokonuje się tego zapisu. Niewątpliwie najpopularniejszym materiałem jest i pozostanie jeszcze długo papier. Oprócz niego stosuje się materiały fotograficzne - błony i papiery fotoczułe.

Jeśli chodzi o urządzenia umożliwiające wydruk na papierze, to wprowadzenie rozwiązań optoelektronicznych nastąpiło z chwilą wyprodukowania przez firmę IBM drukarki laserowej model 3800. Drukarka ta, podobnie jak i inne tego rodzaju, które wkrótce pojawiły się, wprowadziły nową jakość w stosunku do konwencjonalnych drukarek uderzeniowych: znacznie większą prędkość wydruku, możliwość drukowania różnorodnych znaków i obrazów itd. Drukarki laserowe pracują na zasadzie oddziaływania promienia laserowego na warstwę fotoprzewodzącą [55]. Podobnie jak w zwykłej kopiarce kserograficznej warstwa ta jest naniesiona na powierzchnię obracającego się bębna, co pozwala w wyniku kolejnych naświetleń modulowaną impulsowo wiązkę światła, odchylaną wzdłuż tworzącej bębna, uzyskać obraz utajony w postaci zmian potencjałów odpowiadający zapisywanej informacji. Obraz ten jest następnie wywoływany przez podanie na bęben proszku (tonera), a potem przenoszony na papier.

Oprócz drukarek laserowych opracowano i inne drukarki wykorzystujące proces kserograficzny. Zastosowano w nich odmienny sposób naświetlania - za pomocą linijki fotodiod (firma Kodak) [56] lub ciekłokrystalicznych kluczy świetlnych przepuszczających w wybranych punktach światło białe (firma Epson) [57]. Rozwiązania te mają na celu uniknięcie stosowania dość złożonych elementów, które są niezbędne przy naświetlaniu laserowym, tj. układów optycznych, obrotowego zwierciadła wielopłaszczyznowego i modulatora. Dzięki masowej produkcji drukarek laserowych nastąpiło jednak obniżenie kosztów wymienionych elementów i wydaje się, że drukarki laserowe długo zachowają przewagę nad pozostałymi.

Co do urządzeń przeznaczonych do zapisu znaków na materiałach fotograficznych, to istnieje ich kilka rodzajów, z których najważniejsze to:

- urządzenia COM (Computer Output on Microfilm),
- rekordery laserowe,
- naświetlarki poligraficzne.

Pierwsze z tych urządzeń służą do zapisu za pomocą kamery fotograficznej dużej liczby danych na błonie fotograficznej. Dzięki dużemu zmniejszeniu znaków można uzyskać znaczną oszczędność miejsca przy magazynowaniu wydruków w stosunku do zapisów na papierze. Wydaje się jednak, że ten sposób archiwizowania danych ustąpi miejsca ich przechowywaniu na dyskach optycznych.

Rekordery laserowe są przeznaczone do otrzymywania na papierze światłoczułym obrazów o wielu stopniach szarości. Ich liczba wynosi od kilkunastu do ponad 100, co w połączeniu z dużą liczbą elementarnych punktów, z których tworzony jest

obraz (100-144 mm^2) pozwala na uzyskanie bardzo precyzyjnych zdjęć. Naświetlenie kropek odbywa się podobnie jak w drukarce laserowej, tj. za pomocą odchylonego i modulowanego (tym razem analogowo) promienia laserowego. Rekordery znajdują zastosowanie w wielu dziedzinach, w których trzeba uzyskać obraz jakiegoś obiektu lub jego cech poprzez pomiar zdalny, sondowanie, naświetlanie itp. oraz obróbkę komputerową otrzymanych wyników. Przykładowe dziedziny zastosowań to medycyna, badania satelitarne, badania nieniszczące materiałów itd.

Naświetlarki poligraficzne są przykładem specjalistycznych urządzeń podłączanych bezpośrednio do komputera. Stosowane są w systemach przygotowania fotoskładu, którego zadaniem końcowym jest naświetlenie na błonie fotograficznej przetworzonego tekstu w postaci znaków o docelowym wyglądzie. Zadanie to realizuje się obecnie głównie za pomocą promienia laserowego w sposób podobny jak w drukarkach laserowych. Wymagana dokładność formowania znaków z pojedynczych punktów jest tu jednak znacznie wyższa niż w drukarkach, gdyż liczba punktów dochodzi do kilku tysięcy na mm^2. Poprzednio do naświetlania znaków stosowano lampy elektronopromieniowe. Warto też dodać, że urządzenia optoelektroniczne znajdują zastosowanie także w innych fazach produkcji poligraficznej [58].

Podstawowe rodzaje optoelektronicznych urządzeń wejściowych zestawiono w tabl. V.

Tablica V. Klasyfikacja optoelektronicznych urządzeń wyjściowych

Nazwa urządzenia	Materiał	Rodzaj urządzenia
Drukarki kserograficzne	papier	laserowe ciekłokrystaliczne diodowe
Rekordery	papier fotograficzny	laserowe kamera fotograficzna
Naświetlarki poligraficzne	błona, papier fotograficzny	laserowe lampa elektronopromieniowa

5.3.2. Drukarki laserowe

Drukarki laserowe produkuje się w wielu wersjach. Głównym czynnikiem decydującym o ich budowie jest wymagana prędkość drukowania, którą zwykle określa się liczbą stron formatu A4 drukowanych w ciągu jednej minuty.

Zależnie od prędkości można wyodrębnić 3 rodzaje drukarek [58]:

- wolne - 6-20 s./min,
- średniej prędkości - 20-70 s./min,
- bardzo szybkie - ponad 70 s./min.

Drukarki wolne stanowią najmłodszą generację drukarek - są produkowane zaledwie od 4 lat. Odznaczają się bardzo małymi gabarytami, co pozwala umieścić je na stole obok komputera osobistego i wykorzystać do prac biurowych. W drukarkach tych jako źródło światła wykorzystuje się lasery półprzewodnikowe, a warstwę kserograficzną stanowi materiał organiczny, który jest czuły na światło podczerwone. Jego trwałość jest jednak bardzo niska i dlatego po wydrukowaniu kilku tysięcy egzemplarzy bęben musi być zmieniony. W najbardziej obecnie popularnych drukarkach wykorzystujących mechanizm laserowy i kserograficzny, opracowany przez firmę Canon, dokonuje się tego wymieniając cały zespół kserograficzny, który jest wykonany w postaci oddzielnego modułu. Drukarki wolne stosuje się wtedy, gdy wymagana przeciętna wydajność miesięczna nie przekracza 20 000 egzemplarzy.

Drukarki o średniej prędkości działania są przeznaczone do pracy w systemach rozproszonych, gdzie mogą obsługiwać kilka komputerów. Wykorzystuje się w nich lasery helowo-neonowe o mocy 5 mW i dość trwały materiał fotoprzewodzący, jakim jest selen domieszkowany tellurem w celu zwiększenia czułości na światło czerwone. Wydajność miesięczna sięga 500 000 stron.

Drukarki bardzo szybkie znalazły zastosowanie w dużych ośrodkach komputerowych, w których trzeba drukować do kilku tysięcy arkuszy miesięcznie. Z tego względu wymagana trwałość bębna kserograficznego jest bardzo duża. Osiąga się to stosując selen amorficzny, którego maksymalna czułość spektralna występuje przy długości fali ok. 400 nm. Dlatego stosuje się w nich lasery argonowe lub kadmowo-helowe. Innym rozwiązaniem jest zastosowanie lasera He-Ne o mocy 25 mW w połączeniu z warstwą organiczną naniesioną na taśmie, której kolejne odcinki opasują bęben w miarę jej zużywania.

Największą dynamikę sprzedaży wykazują drukarki wolne, których bieżąca produkcja wynosi już kilkaset tysięcy sztuk rocznie. Produkcja drukarek o średniej prędkości wykazuje stałą tendencję wzrostową, przy obecnym poziomie kilkadziesiąt tysięcy sztuk rocznie. Produkcja natomiast drukarek bardzo szybkich ustabilizowała się już od kilku lat na poziomie 5000 sztuk.

Podstawowe cechy drukarek laserowych zestawiono w tabl. VI.

Tablica VI. Podstawowe cechy drukarek laserowych

Rodzaj drukarki i zastosowanie	Wydajność drukowania A4/min	Rodzaj lasera	Wydajność miesięczna A4	Nośnik i jego trwałość (liczba arkuszy)
Wolne (komputery personalne)	<20 (8-10 typ)	diodowy	$<20\ 000$	organiczny od 1000-10 000
O średniej prędkości działania (systemy zdecentralizowane)	20-70	He-Ne (5 mW)	od 20 000 do 500 000	selen domieszkowany tellurem od kilkudziesięciu do kilkuset tys.
Bardzo szybkie (centra obliczeniowe)	>70	Ar.He-Cd He-Ne (25 mW)	od 500 000 do 5 000 000 (3 zmiany)	selen amorficzny 1 000 000 organiczny 10 000

6. POŁĄCZENIA

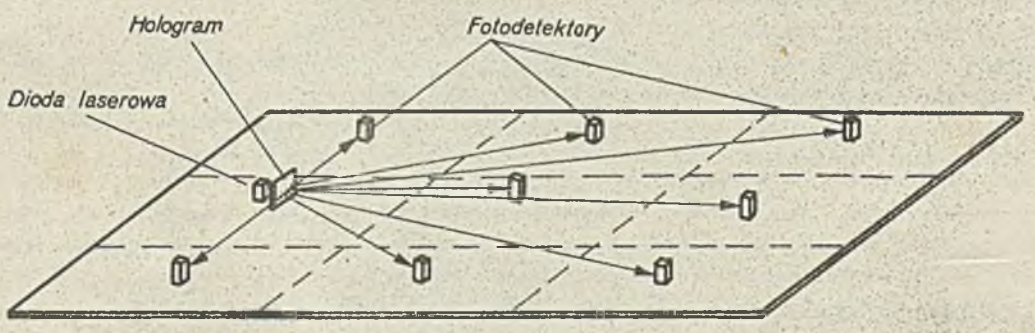
Połączenia modułów operacyjnych w systemie komputerowym odgrywają niezmiernie ważną rolę. W komputerach o architekturze klasycznej (patrz rozdział 2.)to właśnie połączenie pomiędzy pamięcią i procesorem decydowało o możliwościach zwiększania mocy obliczeniowej maszyny. Zwiększenia tej mocy poszukiwano przez zwiększenie liczby modułów operacyjnych i zwielokrotnienie dróg połączeń pomiędzy nimi. Jednak przy skończonych i dość znacznych wymiarach modułów operacyjnych i łączących przewodów prowadzi to do wydłużenia drogi sygnału, a stąd do zróżnicowanego opóźnienia tego sygnału. Zastępując sygnał elektryczny sygnałem świetlnym likwidujemy te trudności, ponadto stwarzamy możliwość przesyłania informacji o złożonej strukturze czasowo przestrzennej, np. odpowiadającej obrazom scen realnych.

W niniejszym rozdziale omówiomy ważniejsze techniki połączeń optycznych w komputerach szerokiego stosowania i w superkomputerach.

Za pomocą światła widzialnego lub promieniowania w bliskiej podczerwieni można przesyłać informacje zarówno na pakiecie jak i pomiędzy pakietami jednego modułu, a także pomiędzy modułami komputera oraz pomiędzy komputerami współpracującymi z sobą w sieci komputerowej. Stosowanie światła jako nośnika przesyłanej informacji pozwala uzyskać największą możliwą prędkość przesyłania informacji, a także uniknięcia zakłóceń elektromagnetycznych i szkodliwych przesłuchów. Światło może rozchodzić się zarówno w swobodnej przestrzeni, jak też może być przesyłane w ośrodku przezroczystym, np. światłowodem.

6.1. Połączenia wewnętrzne komputerów

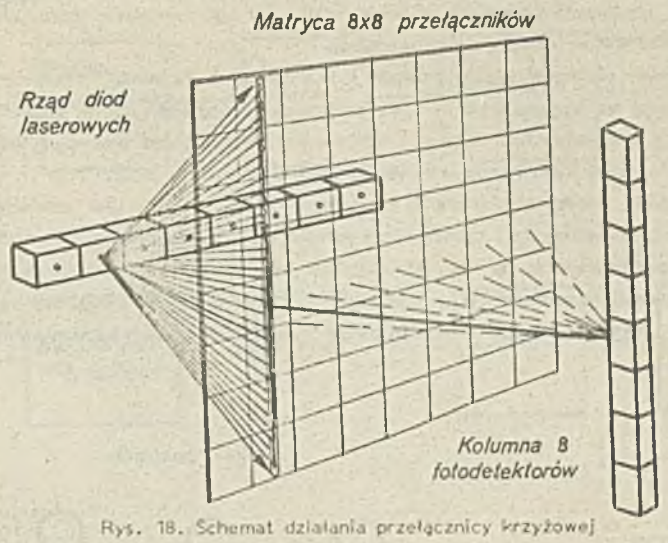
Przesyłanie impulsów światła w swobodnej przestrzeni może być stosowane do przesyłania informacji pomiędzy układami rozmieszczonymi na jednym pakiecie lub pakietach sąsiednich. Przykładem takiego zastosowania jest przesyłanie impulsów zegarowych do poszczególnych układów rozmieszczonych na pakiecie (rys. 17). Impulsy zegarowe odbierane są przez fotodiody rozmieszczone w różnych sektorach pakietu. Fotodiody te są następnie źródłem impulsów zegarowych dla swojego sektora.



Rys. 17. Przesyłanie impulsów zegarowych na pakiecie za pomocą światła

Światło emitowane przez diodę laserową jest kierowane na fotodiody i skupiane na nich za pomocą odpowiednio wykonanego hologramu. Według takiej zasady działania grupa Goodmana w Stanford ma zamiar zrealizować przesyłanie impulsów zegarowych wewnątrz dużego układu scalonego [4].

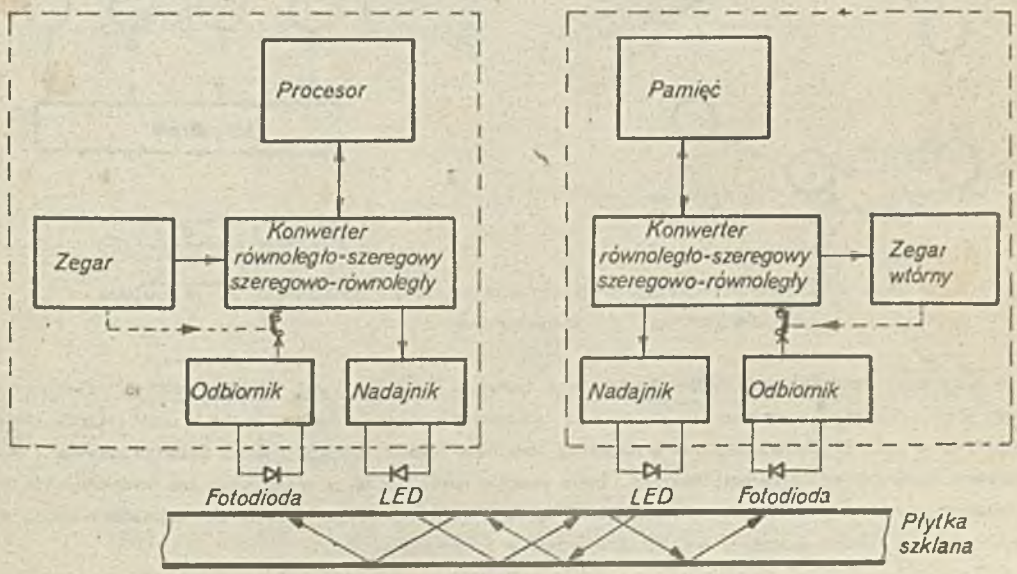
Wykorzystując przesyłanie światła w swobodnej przestrzeni można też zrealizować przełącznicę umożliwiającą przesyłanie informacji z dowolnego z n-źródeł do dowolnego z n-odbiorników. Schemat działania takiej przełącznicy krzyżowej 8x8 przedstawiony jest na rys. 18. W przełącznicy takiej światło emitowane przez każdą z diod laserowych kierowane jest przez odpowiednią kolumnę przełączników optycznych na kolumnę fotodetektorów. Każdy z elementów matrycy przełączników w zależności od jego wysterowania przepuszcza lub nie przepuszcza światła. Jako układy kierujące wiązką światła mogą być zastosowane hologramy.



Rys. 18. Schemat działania przełącznicy krzyżowej

Przełącznica taka o 32 diodach laserowych i 32 fotodetektorach została wykonana w Uniwersytecie w Los Angeles, a planuje się wykonanie przełącznicy o wymiarach 256x256 elementów [4]. Przesyłanie światła w swobodnej przestrzeni zostało też wykorzystane przez konstruktorów japońskich do transmisji informacji pomiędzy modułami komputera Dialog H. W komputerze tym poszczególne moduły wyposażone w nadajnik z diody laserowej i odbiornik z fotodiody lawinowej rozmieszczone są dookoła wypukłego zwierciadła tak, że impulsy emitowane przez dowolny nadajnik trafiają jednocześnie do odbiorników wszystkich pozostałych modułów. W ten sposób zrealizowana została magistrala systemowa poprzez przesyłanie światła w swobodnej przestrzeni.

Magistralę optyczną łączącą moduły mikrokomputera procesor i pamięć wykonano na Uniwersytecie w Duisburgu (RFN)[60] w postaci płytki szklanej pokrytej warstewką srebra. Schemat tej magistrali przedstawiono na rys. 19. Sygnały optyczne emitowane przez diodę LED jednego z modułów wprowadzane do płytki przez okienko w jej metalicznym pokryciu rozchodzą się wzdłuż niej odbijając się od pokrytych srebrem ścian, a następnie przez odpowiednie okienko padają na fotodiodę drugiego modułu. Szeregowe przesyłanie impulsów wymaga zastosowania konwerterów równoległo-szeregowych i szeregowo-równoległych.



Rys. 19. Schemat magistrali optycznej łączącej procesor z pamięcią

Przesyłanie informacji zarówno pomiędzy różnymi modułami komputera, jak i pomiędzy pakietami tego samego modułu można też zrealizować za pomocą łączy światłowodowych. Tego typu połączenia wykonano w komputerach ESS-5 stosowanych przez AT&T w międzymiastowych centralach telefonicznych.

6.2. Połączenia między komputerami

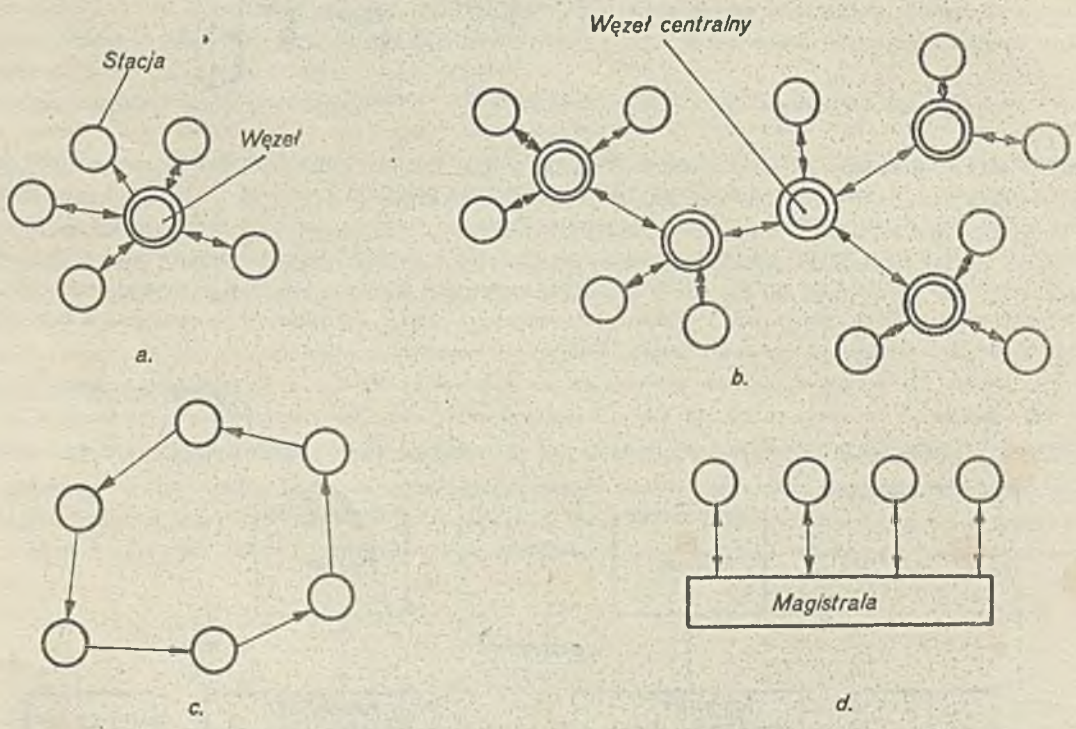
Podczas gdy przesyłanie danych wewnątrz komputera za pomocą światła jest na ogół jeszcze w fazie koncepcji i eksperymentalnych modeli, to przesyłania informacji pomiędzy komputerami poprzez światłowody staje się coraz powszechniejszą praktyką.

Przesyłanie informacji między komputerami odbywa się w sieciach otwartych (WAN) lub sieciach lokalnych (LAN).

W sieciach otwartych współpracować z sobą mogą dowolne komputery gdziekolwiek umieszczone stosując uzgodniony protokół wymiany informacji. Do przesyłania informacji używane są na ogół ogólnodostępne łącza telekomunikacyjne, które w wielu krajach coraz częściej są realizowane w technice światłowodowej.

Sieć lokalna jest to zbiór systemów komputerowych i inteligentnych terminali rozmieszczonych na ogół na niewielkim obszarze i połączonych ze sobą za pomocą środków urządzeniowych i programowych w sposób umożliwiający szybką wymianę informacji pomiędzy każdym z systemów. Sieć lokalna umożliwia korzystanie przez proces użytkowy, prowadzony przez dowolny komputer sieci, z zasobów urządzeniowych i programowych wszystkich systemów sieci.

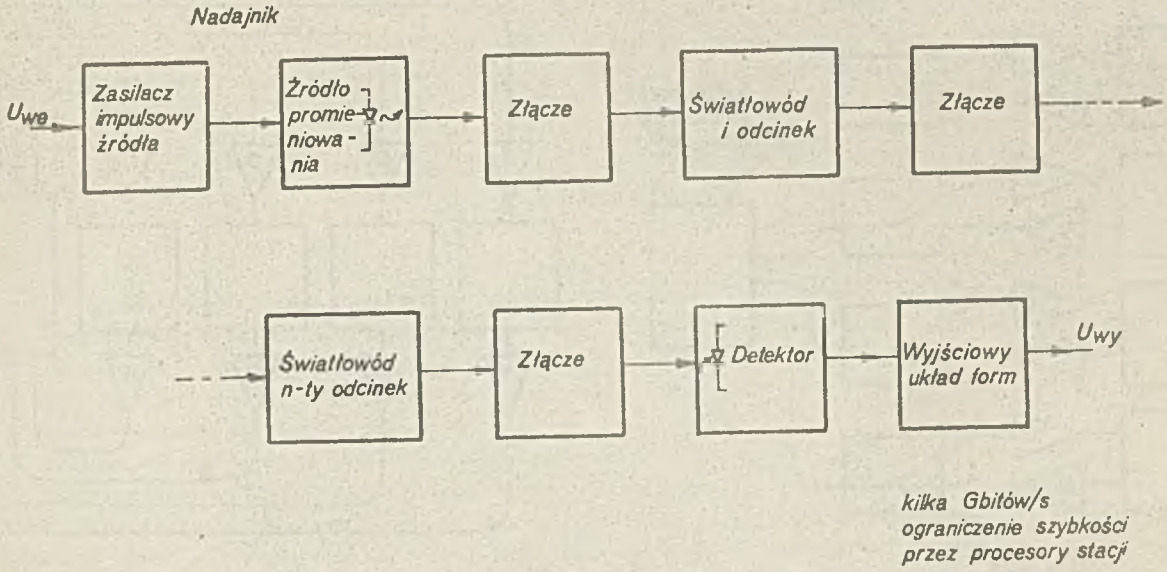
W sieci lokalnej połączenia między komputerami są stałe, a dostęp do nich jest dzielony w czasie. Połączenia między stacjami (tzn. komputerami, terminalami itp.) tworzą różne konfiguracje; podstawowe z nich, jak: gwiaździsta, drzewiasta, pierścieniowa i magistralowa przedstawione są na rys. 20. Możliwe jest też tworzenie konfiguracji mieszanych, łączących na przykład kilka odcinków magistrali w konfigurację gwiaździstą itp. Niezależnie od konfiguracji, sieci różnią się zasadami dostępu stacji do wspólnego łącza. Najpopularniejsze z nich to zasady rywalizacji z wykrywaniem konfliktu i przekazywanie uprawnień



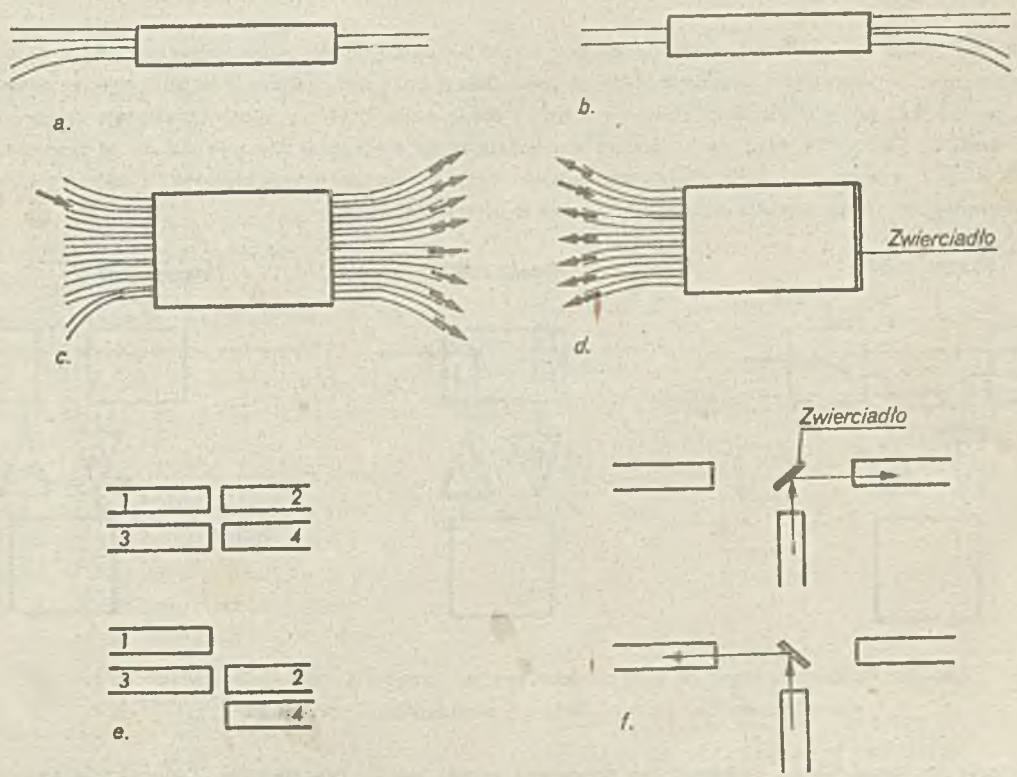
Rys. 20. Różne konfiguracje sieci komputerowych: a - gwiaździsta, b - drzewiasta, c - pierścieniowa, d - magistralowa

Sieci o wszystkich konfiguracjach można zrealizować w technice światłowodowej, chociaż do tego najlepiej nadaje się sieć pierścieniowa, w której stacje łączone są kolejno ze sobą jednokierunkowymi łączeniami punkt - punkt. Łącze takie schematycznie przedstawiono na rys. 21. Składa się ono z nadajnika impulsów światła, odcinków kabla światłowodowego i odbiornika połączonych złączami rozłączalnymi i nierozłączalnymi. Takie obecnie istniejące łącza mogą przesyłać informacje na odległość kilku, a nawet kilkudziesięciu kilometrów z przepływnością binarną do kilku Gbitów/s [61]. Szybkości działania sieci są więc ograniczone nie przez łącza, lecz przez procesory organizujące przepływ informacji w sieci.

Chociaż wszystkie konfiguracje sieci można by zrealizować za pomocą złącz punkt - punkt, wiele sieci znacznie można uprościć stosując rozmaite, specjalne elementy optyczne, takie jak: sprzęgacze, odsprzęgacze, przełączniki itp., pozwalające na kierowanie światła do różnych światłowodów. Niektóre z tych elementów przedstawione są na rys. 22. Zastosowanie ich pozwala na znaczne zmniejszenie liczby nadajników i odbiorników oraz uproszczenie układów elektronicznych.

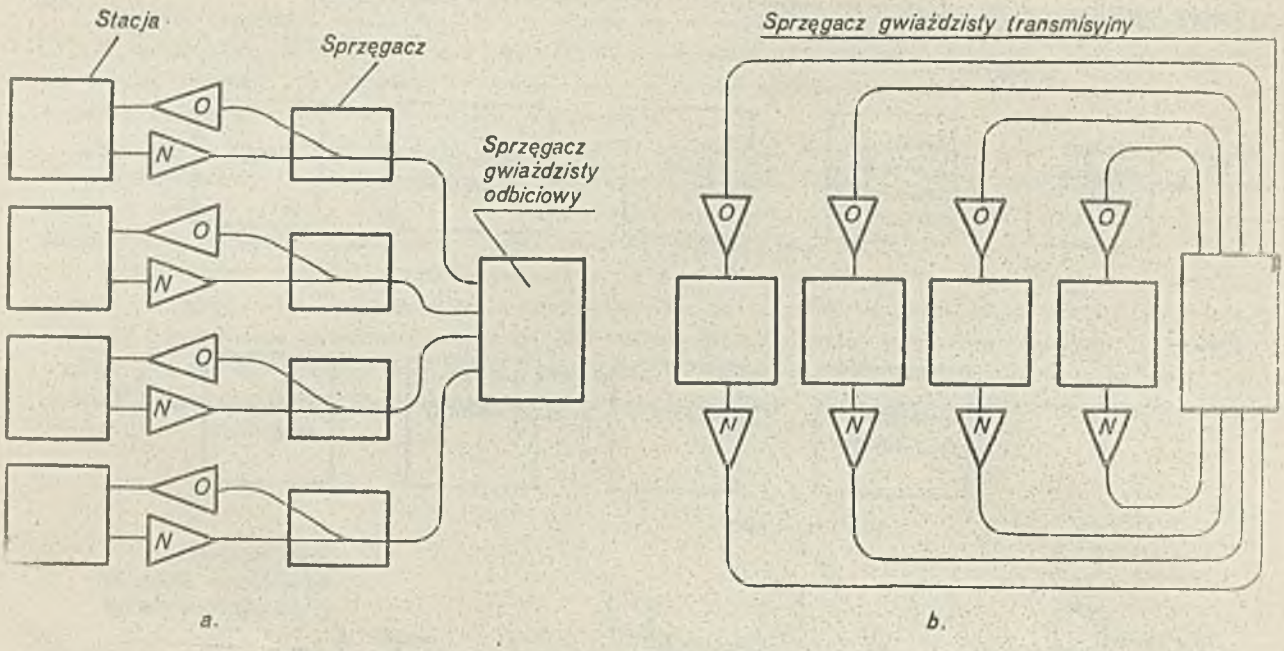


Rys. 21. Łącze światłowodowe punkt - punkt



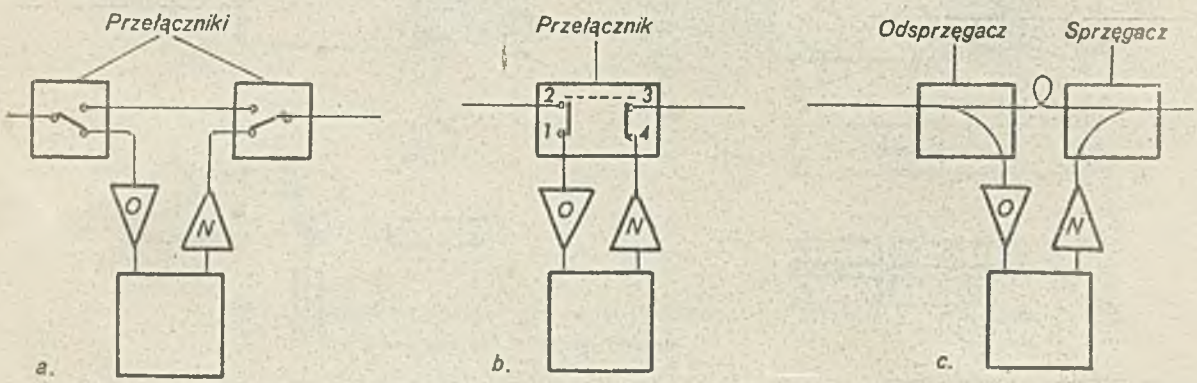
Rys. 22. Elementy optyczne stosowane w sieciach światłowodowych: a - sprzęgacz, b - odsprzęgacz, c - sprzęgacz gwiaździsty transmisyjny, d - sprzęgacz gwiaździsty odbiciowy, e - przełącznik z przesuwem światłowodów, f - przełącznik z obrotowym lustrem lub pryzmatem

Przykłady zastosowania sprzęgaczy gwiaździstych odbiciowego i transmisyjnego przedstawiono na rys. 23. W sieciach tych impulsy światła wysyłane przez dowolną stację dochodzą do wszystkich stacji. Wyjątkowa prostota takich sieci związana jest niestety z ograniczeniami liczby stacji oraz odległości połączeń, gdyż światło wchodzące do sprzęgacza musi zostać rozdzielone na wszystkie wychodzące światłowody, co wielokrotnie zmniejsza moc przesyłanej wiązki światła.



Rys. 23. Sieci o konfiguracji magistralowej zrealizowane jako gwiaździste ze sprzęgaczem: a - odbiciowym, b - transmisyjnym

Przełączniki mogą być wykorzystane np. w sieciach o konfiguracji pierścieniowej do podłączania stacji do sieci (rys. 24 a i b). Pozwalają one na odłączanie i dołączanie stacji do sieci. Stacja może być podłączona do sieci poprzez odsprzęgacz i sprzęgacz (rys. 24 c). Przy takim podłączeniu występuje optyczne obejście stacji, którym sygnały przesyłane są wprost do następnej stacji w przypadku uszkodzenia danej stacji. Sygnały dochodzące przez obejście są znacznie słabsze od sygnałów pochodzących z nadajnika stacji i są wykrywane tylko wtedy, gdy nadajnik stacji nie emituje swoich sygnałów. Odbiór przez następną stację sygnałów o zmniejszonej mocy sygnalizuje jednocześnie awarię poprzedniej stacji.



Rys. 24. Podłączenie stacji do sieci pierścieniowej za pomocą: a - przełączników (jak na rys. 22f), b - przełącznika (jak na rys. 22e), c - odsprzęgacza i sprzęgacza

Sprzęgacze pozwalają też na dwukierunkowe przesyłanie światła jednym światłowodem, takie właśnie zadanie mają sprzęgacze w sieci przedstawionej na rys. 23.

Wśród dalszych elementów optycznych można by jeszcze wymienić multipleksery i demultipleksery optyczne, pozwalające na jednoczesne przesyłanie jednym światłowodem kilku wiązek światła różniących się długością fali, emitowanych z kilku różnych źródeł promieniowania. Pozwala to na dalsze zwiększenie przepływności binarnych łączy światłowodowych.

Coraz szersze stosowanie techniki światłowodowej w sieciach komputerowych wynika z następujących zalet tej techniki: dużej przepływności binarnej przy dłuższych odległościach między stacjami, odporności sieci na zakłócenia elektroniczne, niesemitowania fal elektromagnetycznych dzięki temu nie ma możliwości podsłuchu informacji bez uszkodzenia kabla i dotarcia do wnętrza światłowodu, mniejszych wymiarów i wagi kabla światłowodowego w stosunku do kabla koncentrycznego.

Dzięki powyższym zaletom światłowodowe sieci komputerowe zakładane są przede wszystkim w zakładach przemysłowych, bankach oraz służą do różnych celów militarnych. Szerokość pasma przenoszenia wraz z dodatkową możliwością multipleksowania fali pozwala realizować za pomocą światłowodów nowy rodzaj sieci o zintegrowanych usługach (ISDN), w której oprócz przesyłania danych komputerowych możliwe jest przesyłanie innego typu informacji, jak rozmowy telefoniczne, videotelefon, faksymile, tv itp. [52, 53].

7. ROLA HOLOGRAFII

Holografia jest techniką zapisu i odtwarzania obrazów przestrzennych. Wynaleziona w końcu lat czterdziestych przez Dennisa Gabora, została spopularyzowana przez grupę naukowców z uniwersytetu Michigan z E. Leithem i J. Upatnieksem na czele. Odkrycie lasera w 1962 r. przyczyniło się do gwałtownego rozwoju holografii.

W informatyce [48] możeby wyróżnić trzy dziedziny zastosowania holografii:

1. Użycie komputerów do tworzenia hologramów.
2. Wykorzystanie hologramów do pamiętania danych komputerowych.
3. Łączenie holografii i komputerów do tworzenia komputerów hybrydowych.

Pierwszy punkt mówi o komputerowej syntezie hologramów binarnych. Technikę tę stosuje się do realizacji holograficznych elementów optycznych wykorzystywanych do różnych celów, jak wizualizacja obrazów dużych rozmiarów, czytniki optyczne i drukarki laserowe, a także do kontrolowania jakości soczewek w aparatach optycznych (np. fotograficznych).

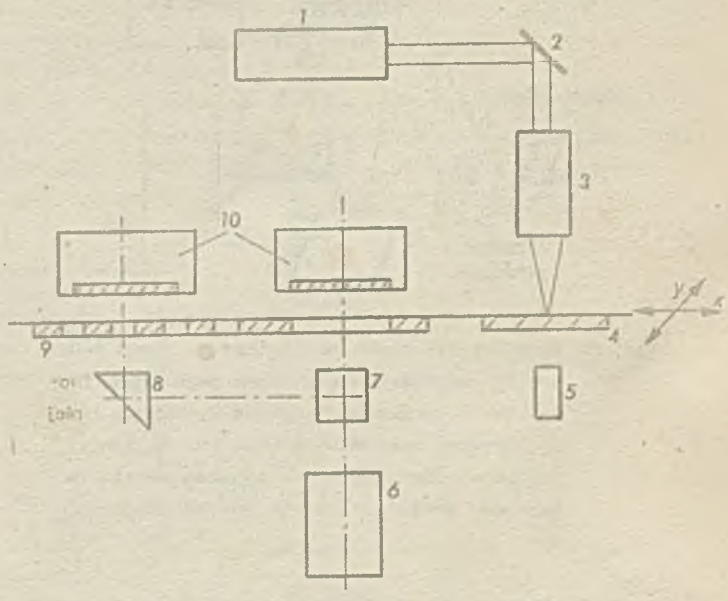
W połączeniu ze wspomaganiem komputerowym hologramy syntetyzowane pozwalają na oglądanie złożonych przedmiotów pod różnymi kątami. Można również porównywać te hologramy z rzeczywistymi i weryfikować np. zgodność projektu i modelu.

O zastosowaniach pamięciowych holografii była mowa w rozdziale 4., przy czym pamięci holograficzne dzielą się na dwie zasadnicze grupy, jeśli chodzi o zasadę pracy: stronicowe (ang. page oriented) i asocjacyjne. Pamięć omawiana w rozdziale 4. należy do pierwszej grupy.

Realizacji technicznej doczekały się też stronicowe pamięci stałe [49], w których nośnikiem informacji jest materiał fotograficzny.

Stała pamięć holograficzna składa się z dwóch zasadniczych części: urządzenia do zapisu informacji i osobnego urządzenia do jej odczytu. Takie rozwiązanie ułatwia obsługę i dodatkowo wpływa na zmniejszenie kosztów całej pamięci.

Przy zapisie należy zapewnić warunki interferencji, przede wszystkim dużą stabilność mechaniczną układu zapisującego, co nie jest wymagane w takim stopniu przy odczycie. Przyczynami niestabilności mogą być wibracje mechaniczne lub przemieszczenia otaczającego powietrza o charakterze akustycznym lub cieplnym. Stała pamięć holograficzna może być wykorzystana jako archiwum danych. Jedno centralne urządzenie służy wtedy do zapisu danych, a odczyt odbywa się za pomocą osobnych tanich urządzeń odczytujących. W pamięciach holograficznych stałych nie stosuje się sterowanych modulatorów przestrzennych. Mikrohologramy w tych pamięciach wykonuje się przy użyciu transparentów otrzymanych za pomocą specjalnych urządzeń.



Rys. 25. Schemat optyczny urządzenia do zapisu transparentów [49]: 1 impulsowy laser azotowy, 2 zwierciadło, 3 mikroobiektyw, 4 transparent, 5 fotodioda, 6,7,8 elementy systemu oświetlania czujnika położenia, 9 siatka czujnika położenia, 10 czujniki optyczne

Transparenty takie można wytwarzać używając impulsowego lasera dużej mocy. Wykorzystanie tej metody daje możliwość uzyskania transparentów o dużym kontekście między ciemnymi a jasnymi polami. Oprócz tego ma to także tę zaletę, że nie wymaga obróbki chemicznej, jak to ma miejsce przy wykorzystaniu materiałów światłoczułych. Transparenty także można wykonywać na urządzeniu przedstawionym na rys. 25.

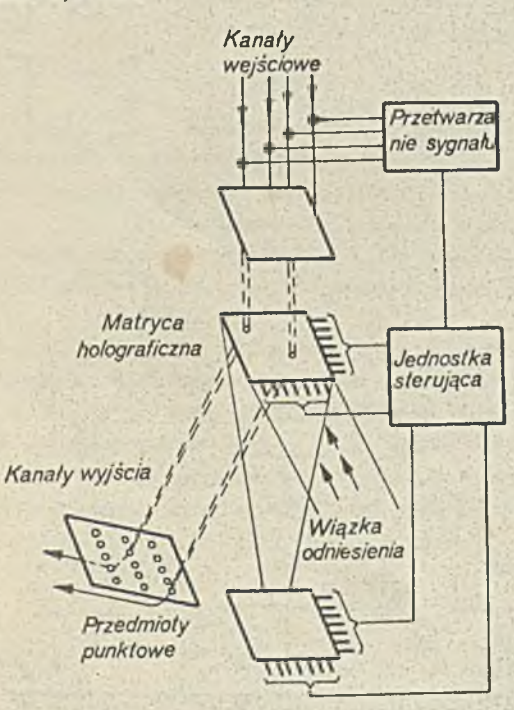
Urządzenie to wykonuje transparenty o następujących parametrach: liczba bitów 98x96, średnica otworu 100±10 μm, krok bitów 200±10 μm, wymiary transparentu 25x25 mm, wymiary części roboczej transparentu 19,6x19,2 mm, czas wykonywania transparentu 6 min.

Transparent wykonany jest z płyty szklanej o grubości 2 mm, pokrytej warstwą niobu o grubości 2000 Å, która odznacza się dobrym przyleganiem do szkła i niewysoką energią wymaganą do jej odparowania [49].

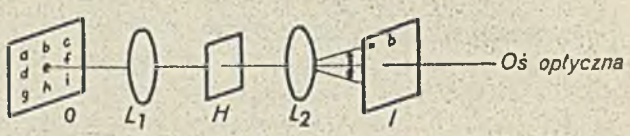
Działanie tego urządzenia polega na wypalaniu otworów w odpowiednich miejscach transparentu za pomocą lasera 1. Promień lasera jest zogniskowany za pomocą obiektywu 3 w płaszczyźnie transparentu 4 . Moment przepłatania transparentu jest określony za pomocą sygnału uzyskiwanego z fotodiody 5. Sygnał ten steruje automatycznym urządzeniem przesuwu transparentu w płaszczyźnie zogniskowania wiązki laserowej.

Przesuwanie odbywa się za pomocą liniowych silników prądu stałego z niezależnym wzbudzeniem, wyposażonych w czujniki szybkości.

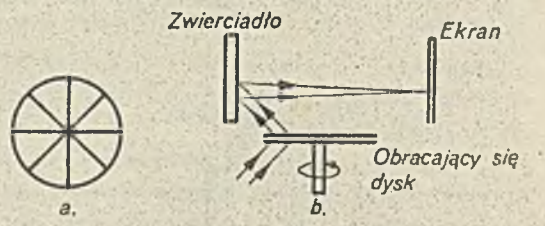
Dokładność wykonania transpartentu zapewnia optoelektroniczny czujnik połączenia wykonany w formie rastru; dokładność ta wynosi 0,01 mm.



Rys. 26. Struktura przełącznika holograficznego. Zespół światłowodów zakończonych soczewkami skupiającymi tworzy matrycę wejściową. Promienie wychodzące z niej są odchylane przez matrycę odchylania do matrycy wyjściowej. Optyczny system adresowy pozwala na ustalenie i zmianę parametrów matrycy holograficznej



Rys. 27. a - skaner utworzony przez umieszczenie siatek holograficznych na dysku, b - dysk w układzie odczytu



Rys. 28. Rozpoznawanie znaków za pomocą holografii. Przykład dotyczy rozpoznania litery b w obiekcie (tekście) O, H - hologram litery b, I - płaszczyzna obrazu, L_1 i L_2 - soczewki

Przykładem zastosowania holografii w trzeciej z wymienionych dziedzin są przełączniki holograficzne [39], które mogą być użyte do połączeń w sieci światłowodowej. Przełączniki takie charakteryzują się znacznie większym pasmem przenoszenia w porównaniu z przełącznikami elektronicznymi lub optoelektronicznymi. Są one zwykle matrycami o adresowaniu dwuwymiarowym i pozwalają łączyć dowolny element matrycy wejściowej z dowolnym elementem matrycy wejściowej. Matryca utworzona jest przez końce światłowodów wraz z soczewkami skupiającymi (rys. 26). Połączenia między wejściem a wyjściem ustala się przez wybór komórki holograficznej odpowiadającej temu wejściu i zapisaniu w niej hologramu odpowiadającemu wyjściu. Aby określić żądany kierunek wyjścia należy określić rozstęp prążków i kierunek siatki w płaszczyźnie odchylania. Trzeci parametr, tzn. kierunek prążków w głąb materiału powinien być wyregulowany. Oznacza to, że hologramy muszą być tworzone w sposób

odwracalny, dostatecznie szybko, a więc występują te same problemy co w pamięci holograficznej. Francuska realizacja (CNET) takiego przełącznika na termoplastyku Steybelite Ester 10 pozwala na kilkadziesiąt przełączeń na sekundę przy wydajności dyfrakcyjnej kilkunastu procent. Inne rozwiązanie, zawierające drugą matrycę odchylającą, zapewnia równoległość promieni wyjściowych.

Układy holograficzne stosowane są również do systemów wyblerania opracowanych dla czytników optycznych. Jeśli hologram obraca się wokół wiązki odniesienia, wiązka ugięta zakreśla koło. Aby uzyskać linię prostą, należy umieścić na dysku wiele identycznych siatek dyfrakcyjnych. Rysunek 27 przedstawia taki dysk holograficzny i jego pracę.

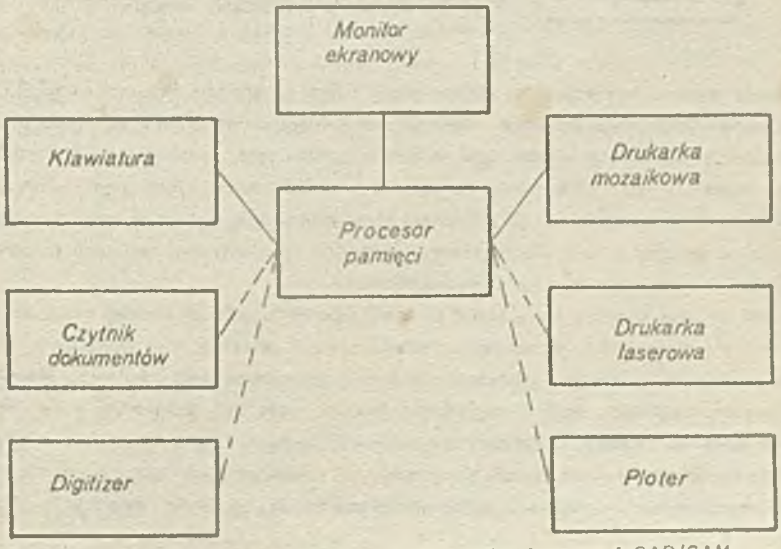
Jeszcze innym zastosowaniem holografii, tym razem do przesyłania informacji, są filtry holograficzne również w postaci siatki dyfrakcyjnej, którą uzyskuje się przez interferencję dwóch wiązek z tego samego źródła.

Holografia wreszcze może odgrywać znaczną rolę w samym przetwarzaniu informacji, a zwłaszcza przy rozpoznawaniu kształtów dwu- lub trójwymiarowych. Hologram uzyskany za pomocą światła spójnego jest transformacją Fouriera oświetlanego obiektu. Rozpoznawanie można więc przeprowadzić przez autokorelację (rys. 28), wtedy przedmiot zawierać może poszukiwany element, którego hologram umieszcza się w płaszczyźnie równoległej do płaszczyzny przedmiotu. W rezultacie uzyskuje się obraz tego elementu w płaszczyźnie wyjściowej, przy czym jeśli w przedmiocie element jest inaczej zorientowany lub ma inną wielkość, odtwarzanie może być znacznie zniekształcone. Przedmiot może zawierać wiele poszukiwanych elementów i wszystkie one zostaną rozpoznane.

8. OBSZARY ZASTOSOWAN

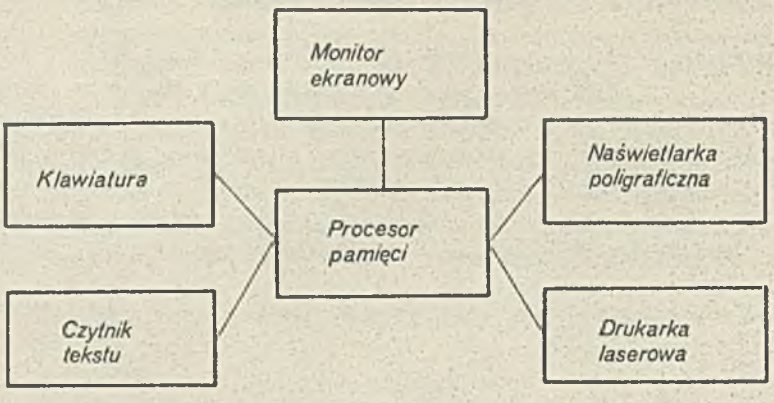
Użycie technologii optycznej do budowy układów i modułów komputerowych nie przyczyni się w najbliższych latach do powstania komputera optycznego o odrębnej architekturze. Przeciwnie, komputer optoelektroniczny będzie wkomponowany w główny nurt rozwoju komputerów. Dziedzicząc zgromadzone doświadczenie badawcze i problemy aplikacyjne, komputer OE umożliwi efektywne rozwiązywanie tych problemów. Optyczne środki komputerowe w formie modułów (pamięci, procesory, wejście/wyjście) oraz połączeń spowodują znaczną poprawę mocy obliczeniowej. Procesory danych cyfrowych działając na strukturach dwu- i wielowymiarowych pozwalają na szybkie wykonywanie obliczeń skomplikowanych funkcji analitycznej, co jest warunkiem istotnym w zadaniach dotyczących modelowania złożonych zjawisk, np. z zakresu fizyki ciała stałego i fizyki cząstek elementarnych [8]. Utworzenie środowiska obliczeniowego, zawierającego trójwymiarową pamięć i procesory, w formie przestrzennego odpowiednika ośrodka, ułatwia intuicyjną jego reprezentację, a sam proces obliczeniowy imituje zachowanie się elementów i całego ośrodka. Dotychczas obliczenia takie były realizowane na superkomputerach typu CRAY [10]. Podobne struktury obliczeń wystąpują w modelach predykcji pogody [9]. Inną dziedziną wymagającą wielkiej liczby obliczeń na danych przestrzennych jest kreacja form przestrzennych, np. w projektowaniu lub twórczości artystycznej [14].

Technika optyczna umożliwia też efektywną implementację takich funkcji, jak korelacja przestrzenna i analiza widma danych. Funkcje te są niezbędne w operacjach dostępu do pamięci, rozpoznawania itp. Operacje stanowiące składniki operacji złożonych, jak np. rozpoznawanie form przestrzennych w porównaniu z wzorcem [11], są podstawą budowy systemów z wielką bazą danych; systemy takie są niezbędne do badania przestrzeni kosmicznej i do sterowania obiektami w tej przestrzeni, do rozpoznawania obiektów w działaniach wojskowych [15, 31].

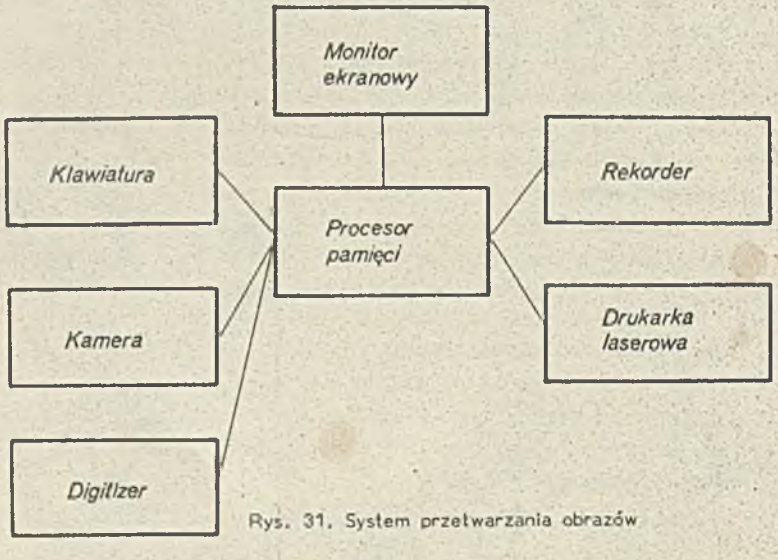


Rys. 29. Komputer personalny w zastosowaniu biurowym i CAD/CAM

Badania przestrzeni kosmicznej i sterowanie zachowaniem się obiektów w tej przestrzeni jest wyjątkowo trudnym i wymagającym typem aplikacji. Jest tak szczególnie wtedy, gdy obiektów w przestrzeni jest wiele i poruszają się one ze znaczną szybkością. Obok trudności przy wykonywaniu sprzętu nie mniejsze kłopoty napotyka projektant oprogramowania, który musi opisać tę złożoną sytuację, jej dynamikę i stworzyć podstawy do wnioskowania i decyzji [12]. Stopień skomplikowania programów jest bardzo wysoki, a zapis tych programów tradycyjnymi językami wysokiego poziomu (typu FORTRAN, PASCAL, ADA) staje się niemal niemożliwy.



Rys. 30. System komputerowy przygotowania fotoskładu



Rys. 31. System przetwarzania obrazów

Mając na względzie wysokie wymagania na niezawodność takich programów (ich poprawności), inżynierskie programowanie zmierza w kierunku języków sztucznego intelektu o charakterze deklaratywnym [29]. Języki te pozwalają formułować problem i jego rozwiązanie stosując reprezentację wiedzy ujętą w formie symbolicznej, odpowiedniej dla problemu rzeczywistego. Pamiętanie informacji symbolicznej i jej przetwarzanie za pomocą urządzeń optoelektronicznych stwarza możliwości implementacji języków typu PROLOG, LISP na komputerach OE, a dzięki temu organizację systemów komputerowych w formie systemów z bazą wiedzy [30]. Optyczne systemy z bazą wiedzy stworzą narzędzie do efektywnej realizacji oprogramowania dla wyżej wymienionych trudnych aplikacji, np. badania w przestrzeni kosmicznej.

Przedstawione zastosowania odnoszą się głównie do tzw. superkomuterów o wielkiej mocy obliczeniowej [13]. Szeregując maszyny według ich mocy obliczeniowej, na początku takiego szeregu ustawimy mikrokomputery, które również mogą być wyposażone w optyczne moduły komputerowe i połączenia w formie optycznych sieci lokalnych. Mikrokomputer wyposażony w optyczną pamięć dyskową oraz optyczne wejście do analizy obrazów może być doskonałym stanowiskiem roboczym projektanta. Należy spodziewać się szybkiego rozwoju i wielkoseryjnej produkcji systemów tego typu. A oto kilka przykładów systemów mikrokomputerowych do określonych klas zastosowań. Na rysunku 29 pokazujemy konfigurację przystosowaną do prac projektowania wyrobów i procesów produkcyjnych. Modułami optoelektronicznymi są tu: koder informacji, drukarka laserowa i kreślak

Wysoka jakość pracy tych urządzeń podnosi wydatnie efektywność i ogólną użyteczność systemu dla danej klasy prac. Rysunki 30 i 31 prezentują odpowiednio system do przygotowania fotoskładu i do przetwarzania obrazów w zastosowaniach przemysłowych. Musimy pamiętać, że dołączenie optoelektonicznych urządzeń do komputera elektronicznego wymaga odpowiednich jednostek sprzęgających, systemu operacyjnego i języka programowania.

9. PODSUMOWANIE

Komputer został stworzony po to, by wyręczał człowieka w wykonywaniu żmudnych i złożonych zadań obliczeniowych. I w istocie, komputer spełnia pokładane w nim nadzieje stając się naczelnym narzędziem nie tylko w obszarach branych pierwotnie pod uwagę (tj. tablicowanie funkcji, księgowość, obliczenia naukowe), lecz obejmuje niemal wszystkie dziedziny działalności człowieka, staje się głównym organem w procesach sterowania, zarządzania, produkowania. W tych funkcjach komputer zdecydowanie przewyższa zdolności człowieka. Są jednak klasy problemów obliczeniowych, w których człowiek nadal zdecydowanie przewyższa zdolności komputera, chodzi tu m. in. o rozpoznanie wzorów (obrazów) i ich kojarzenie w czasie rzeczywistym, w odniesieniu do informacji niepełnej. Fakt ten uzasadnia poszukiwanie innych technologii realizacji niż technologia elektroniczna i innych architektur komputera. Nie może być jednak mowy o gwałtownych zmianach; nowe powstaje stopniowo, dziedzicząc doświadczenie badawcze i problemy.

Bieżące badania i prace konstrukcyjne nad komputerem optoelektronicznym (OE) można powiązać z jego następującymi formami:

1. Komputer szerokiego (masowego) stosowania. Forma ta powstaje przez wzbogacenie konfiguracji komputera elektronicznego w moduły OE, tj. pamięci, procesory, drukarki itd.), ułatwia przygotowanie systemu do zadania i zwiększa efektywność aplikacji.
2. Superkomputery. Dzięki technice optycznej możemy osiągnąć ogromne szybkości działania (najkrótszy impuls optyczny uzyskany w AT&T, Laboratoria Bella [18] wynosi 0,008 ps, podczas gdy najszybszy tranzystor przełącza się w czasie aż 5 ps; działanie równoległe; przez prostą soczewkę można przepuścić około 10 000 niezależnych wiązek światła, co umożliwia jednoczesny dostęp do wielu elementów procesorowych; wiązki światła nie wpływają na siebie wzajemnie, większa jest więc możliwość upakowania połączeń w układzie i pomiędzy układami.
3. OE Komputer z bazą wiedzy. Stosując układy optyczne oczekuje się efektywnej implementacji funkcji do operacji pamiętania struktur ikonicznych, odczytu przez pobudzanie wiązką reprezentującą jedynie fragment tej struktury (wyrażający np. określoną cechę), a więc odczyt przez kojarzenie itd.; stworzy to warunki do budowy bardziej wydajnych maszyn sztucznej inteligencji.
4. OE Komputer-mózg. Technologia układów optycznych otwiera drogę do badań nad komputerem w formie sieci neuronowej [33], wynika to m. in. z olbrzymiej zdolności realizacji połączeń optycznych; struktury neuronowe są podstawą do wyjątkowo szybkiej realizacji funkcji rozpoznawania wzorów (obrazów).

Obecnie w świecie najbardziej zaawansowane są badania nad komputerem masowym. Osiągnięto już poziom produkcji rynkowej modułów optycznych, co z kolei umożliwiło stosownie do zadań kompozycję systemów OE.

Następne z kolei badania dotyczą superkomputerów; prowadzi je zaledwie kilka krajów. Prace nad OE komputerem z bazą wiedzy są w pełni nowe i wymienić tu można jedynie pojedyncze prace wstępne. Podobnie sprawa przedstawia się z pracami nad komputerem optycznym-mózgiem. Wyniki praktyczne tych dwóch kierunków są sprawą przyszłości.

Uważamy, że w kraju trzeba koniecznie zintensyfikować prace podstawowe wszystkich czterech wymienionych kierunków badań. Prace te należy prowadzić w następujących obszarach:

- Architektura, zasady implementacji funkcji i organizacji komputerów optoelekttronicznych; algorytmy działania systemu, języki problemowe.
- Modulatory przestrzenne i ich materiały.
- Funktory logiczne i ich materiały; układy do urządzeń we/wy.
- Pamięci.
- Połączenia.

x x x

Niniejsze opracowanie powstało z inspiracji profesora Adama Smolińskiego - Przewodniczącego Polskiego Komitetu Optoelektroniki SEP. Główne tezy pracy były poddane dyskusji w Zespole Roboczym ds. Optoelektroniki Informatycznej PK Opto, a następnie zaprezentowane na walnym spotkaniu członków tego Komitetu. Do sformułowania treści pracy przyczynili się więc wszyscy członkowie Zespołu Roboczego - uczestnicy wspomnianej dyskusji. Szczególnie jednak pragniemy podziękować docentowi B. Wołczakowi i doktorowi J. Gajdzie za ich opracowanie nt. roli holografii i pamięci holograficznej [64, 65], którego część wykorzystaliśmy. Bardzo też pomocne były dla nas dyskusje z dr. M. Daszkiewiczem oraz z dr. A. Dubikiem.

LITERATURA

[1] Holmes L., Panel urges broad effort in optical computing, Laser Focus/Electro-Optics, July 1986 112-118.
[2] Vander Lugt A., Coherent Optical Processing. Radiation, A Division of Harris-Intertype Corporation, Raport 1977.
[3] Wrzeszcz Z., Własności twornika stronicy pamięci holograficznej w maszynie matematycznej, Prace Naukowo-Badawcze Instytutu Maszyn Matematycznych, Warszawa 1977.
[4] Bell T.E., Optical Computing: A Field in Flux, IEEE Spectrum, August 1986.
[5] Ellon H.P., Morozov N.N., Optoelectronic switching systems in telecommunications and computers, Marcel Dekker Inc., New York 1984.
[6] Ruschitzka M. /ed/., Parallel and large-scale computers: performance, architecture, applications, 10-th IMACS World Congress on Systems Simulation and Scientific Computation, Montreal, Canada, 8-13 August 1982.
[7] Ameling W., Parallelism in Computer Architecture, w: [6].
[8] Bowler K.C., Pawley G.S., Molecular dynamics and Monte Carlo simulations in solidstate and elementary particle physics, Proc. IEEE, 72, 1 (1984), 42-55.
[9] Williamson D.L., Swartztrauber P.N., A numerical weather prediction model , Proc. IEEE, 72, 1 (1984), 56-58.
[10] CRAY-2 System. Auerbach Data World, Auerbach Publishers Inc., 1982, poz. 102. 3004. 150.
[11] Duda R.O., Hart P.E., Pattern Classicication and Scene Analysis. Wiley - Interscience Publ., New York 1973.
[12] Nilsson N.J., Principles of Artificial Intelligence, Tioga Publ. Co., Palo Alto 1982.
[13] Flynn M., Some computer organizations and their effectiveness, IEEE Transactions on Computers, C-21 (1972), 948-960.
[14] Demos G. i inni, Digital Scene Simulation: Synergy of Computer Technology and Human Creativity, Proc. IEEE, 72, 1 (1984), 22-31.
[15] Dróżdż H., Kołosowski W., Rozpoznanie kosmiczne, Wojskowy Przegląd Techniczny, 1986, nr 7-8. 299-301.
[16] Trauby A.H. (ed.), The Collected Works of John von Neumann, vol. 5, New York, Macmillan (1963), 34-79
[17] Bell C.G., Newell A. Computer Structures, New York, Mc Graw-Hill (1971), 92-119.
[18] Hecht J., Optical Computers, Hegh Technology, February 1987, 44-49.
[19] Amdahl G.H., Architectural concepts for highperformance, general-purpose computers, Information Processing 83, R.E.A. Mason (ed.), IFIP, North-Holland 1983.
[20] Moto-OKa T., Fuchi K., The architectures in the fifth generation computers, Information Processing 83, R.E.A. Mason (ed.), IFIP, North-Holland 1983.
[21] Basov N.G. i inni. Application of lasers to computers, [w:] Lasers Handbook, Arecchi F.T., Schulz-Dubois E.O. (ed.), North-Holland 1972.
[22] Abraham E. i inn, The optical digital computer, Scientific America, February 1983, 85-93.
[23] Smith P.W., Tomlinson W.J. Bistable optical devices promise subpicoseCond switching, Spectrum, 8, June 1981, 26-33.
[24] Smith S.D. i inni, Restoring optical logic: demonstration of extensible all-optical digital systems, Optical Engineering, 26, 1 (1987), 45-52.
[25] Huang A., Architectural considerations involved in the design of an optical digital computer, Proc. IEEE, 72, 7 (1984), 780-786.
[26] Coller J.R. i inni, Optical Holography, Academic Press, New York 1971.
[27] Preston K., Coherent Optical Computers, McGraw-Hill, New York 1972.
[28] Hayes J.P., Computer Architecture and Organization, McGraw-Hill, New York 1978.
[29] Eisenback S., Sadler C., Declarative Languages: An overview, BYTE, 10, 8 (1985), 181-200.
[30] Newell A., Simon H.A., Human Problem Solving, Prentice-Hall Inc., Englewood Cliffs, New York 1972.
[31] Hecht J., Light modulators help crunch image data, High Technology, June 1985, 69-72.
[32] Minsky M., Papert S., Perceptrons, An Introduction to Computational Gometry, MIT Press, Cambrige 1969.
[33] Hopfield J.J., Neural Networks and physical systems with emergent collection computational abilities. Proceedings of the National Academy of Sciences, vol. 79 (1982), 2554-2558.
[34] Shaw D.E., On the range of applicability of an artificial intelligence machine, Artificial Intelligence, 32, 2 (1987), 151-172
[35] Stevens J.K., Reverse engineering the brain. Według tłumaczenia na rosyjski, [w:] Realuost i prognozy iskustviennogo intelekta, MIR, Moskva 1987.
[36] Disques optiques: consensus et fusions, Minis et Micros, 1988, nr 293-4, 38.
[37] Psaltis D., Hong J., Shift-invariant optical associative memories, Optical Engineering 26, 1 (January 1987), 10-15
[38] Holographic memory system exhibits rapid access, Computer Design, December 1968.
[39] Une nouvelle dimension pour d'informatique: 1 holografie, Micro Systemes, Fevrier 1987.

[40] Koncepcja zestawu zawierającego pamięć holograficzną, moduły przetwarzania informacji graficznej i moduły interface'u, Archiwum opracowań IMM, Warszawa 1978, nr 42.
[41] Design of optical storage products, Laser Focus, September 1985.
[42] Synak R., Optyczne pamięci dyskowe, Techniki Komputerowe, 24, 1 (1986).
[43] Leibson S., Optical-disk drives target standard 5 ¼ in sites, EDN (1986.12.25).
[44] CD ROM market will reach $939 million by 1991, EDN (1986.09.13).
[45] Optical storage shipments to hit $2 billion by 1991, Computer Age-World Trade 10 (1986).
[46] Les disques, revolution de la micro-informatique, Micro Systemes, December 1986.
[47] Erasable.cal media promises data-storage break-throughs, Computer Design 1988, nr 11.
[48] Encyclopedia of Computer Science and Technology, 9, Marcel Dekker Inc. 1978.
[49] Kubota K., Kondo M., Sugama S., Nishida N., Skaguchi M. Holographic disk with data transfer rate: its application to an audio response memory, Applied Optics, 6 (1980).
[50] Holt J., 1985 trends in detector technology, Laser Focus, December 1985, 76-80.
[51] Ravich L.E. Pyroelectric detectors and imaging, Laser Focus, July 1986, 104-115.
[52] Fleming T., Display modules span existing technologies, suit diverse uses, EDN, December 25, 1985, 124-130.
[53] Biancomano V., Flat - panel displays drive hard and fast to win high-resolution color race. Electronic Des., May 28, 1987 81-90.
[54] LCDs for large-screen TV industry, Laser Focus, July 1986, 100-101.
[55] Beiser L., Laser scanning and recording, developments and trends, Laser Focus, February 1985, 88-96.
[56] LED array technology yields faster, more reliable printer, Computer Des., October 15, 1986, 94.
[57] Warren C., Laser printers draw from copier technology, Mini-Micro-Systems, October 1986, 87-99.
[58] Bruno M.H., Gutenberg goes electronic, Laser Focus, August 1986, 114-119.
[59] Hug W.F., The market for lasers in reprographics, Laser Focus, August 1986, 99-101.
[60] Gosch J., Light beams carry data in glass bus, Electr. Weik, August 6, 1984.
[61] Sikorski A., Technika światłowodowa w lokalnych sieciach komputerowych, Techniki Komputerowe 1987, nr 5-6.
[62] Nicholson P.J., Fiberoptics for multiple-service local area networks, Laser Focus, November 1983.
[63] Finnie G., Big vendors back fibre standard, Communication Systems Worldwide, February 1987.
[64] Wolczak B., Borkowska A., Gajda J., Wybrane zagadnienia optocyfroniki, Prace Naukowe Politechniki Szczecińskiej nr 283, IAP nr 9, Szczecin 1986.
[65] Wolczak B., Gajda J., Struktura pamięci holograficznych (maszynopis nadesłany autorom artykułu).

OPTOELECTRONIC COMPUTERS

The potential of optoelectronic computers is enormous. Research works in optical processing technology last a long time, nevertheless the optical computers have not reached the technical maturity of electronic computers. The purpose of this paper is to present an optoelectronic computer system architecture and review the principles of optoelectronic realizations. The chapter on system architecture contains a thesis according to which the development of computer system architecture is of continous character despite changes of the technology. Thus the architecture of an optoelectronic computer system should evolve from its predecessor - the electronic computer system architecture, at last for the nearest future (5-10 years). Important feature of the computer architecture is a system modularity. This feature allows to consider separately the optoelectronic modules describing their behavior mechanisms, the research status and research directions. The chapters on processor, memory, input/output devices, internal and external connections are followed by chapters which illustrate the role of holography in an optoelectronic computer and a possible area of application.

KOMPUTERY OPTOELEKTRONICZNE

Potencjalne możliwości komputerów optoelektronicznych są wprost niezwykłe. Prace nad technologią optycznego przetwarzania informacji trwają już długi czas, jednak komputery optyczne nie osiągnęły dojrzałości komputerów elektronicznych. Celem niniejszej pracy jest prezentacja architektury optoelektronicznego systemu komputerowego oraz przegląd zasad optoelektronicznej realizacji. Rozdział poświęcony architekturze systemu zawiera tezę, zgodnie z którą rozwój architektury systemu komputerowego ma charakter ciągły bez względu na zmiany technologii. Wobec tego architektura optoelektronicznego systemu komputerowego powinna wynikać z architektury elektronicznego systemu, co najmniej na najbliższą przyszłość (5-10 lat). Ważną cechą architektury komputerowej jest jej modularność systemowa. Cecha ta pozwala rozpatrywać oddzielnie moduły optoelektroniczne opisując ich mechanizmy zachowania, status badań i ich kierunki. Po rozdziałach dotyczących procesora, pamięci, urządzeń wejścia/wyjścia, połączeń wewnętrznych i zewnętrznych następują rozdziały ilustrujące rolę holografii w komputerze optoelektronicznym oraz pewien prawdopodobny obszar jej zastosowań.

ОПТОЭЛЕКТРОННЫЕ КОМПЬЮТЕРЫ

Потенциальные возможности оптоэлектронных компьютеров очень большие. Работы над разработкой технологии оптической обработки информации продолжаются уже несколько лет, но все таки оптические компьютеры не пользуются популярностью электронных компьютеров. Цель данной работы заключается в представлении архитектуры оптоэлектронной компьютерной системы, а также в просмотре правил оптоэлектронной реализации. Раздел посвящен архитектуре системы содержит тезис, согласно которому развитие архитектуры имеет непрерывный характер, независимо от изменений технологий. Из этого следует, что архитектура оптоэлектронной компьютерной системы должна вытекать из архитектуры электронной системы по крайней мере в течение ближайших 5-10 лет. Важной особенностью компьютерной архитектуры является ее системная модульность. Эта особенность дает возможность рассматривать отдельные оптоэлектронные модули, описывать механизм их поведения, статус исследований и их направления. После разделов описывающих процессор, память, устройства ввода/вывода, внутренние и внешние соединения следуют разделы рассматривающие роль голографии в оптоэлектронном компьютере, а также определенную, предполагаемую область ее применений.

Prace naukowo-badawcze Instytutu Maszyn Matematycznych

Zdzisław WRZESZCZ
Wojciech PRZYŁUSKI
Jerzy KASPRZYK
Andrzej ROWICKI

Podstawy implementacji systemów eksperckich na mikrokomputerze

1. WPROWADZENIE

Niniejsze opracowanie zawiera część wyników rocznego studium w zakresie implementacji systemów eksperckich na komputerach personalnych. Wykorzystując wyniki fundamentalnych badań nad sztuczną inteligencją podjęliśmy pracę, której celem było zebranie podstawowych wiadomości niezbędnych do realizacji systemów eksperckich na komputerach popularnych i łatwo dostępnych w kraju. Wybór mikrokomputera jako maszyny interpretującej język sztucznej inteligencji wiąże się z zamiarem przybliżenia systemów eksperckich do praktyki w wielu dziedzinach, zwłaszcza w działalności przemysłowej, gdzie komputery personalne odgrywają coraz większą rolę. Rozwój technologii podzespołów i układów elektronicznych spowodował, że moc obliczeniowa dzisiejszych mikrokomputerów profesjonalnych przewyższa moc obliczeniową wielu maszyn sprzed dziesięciu lat, zwanych wtedy komputerami wielkimi (ang. mainframe) .

Pracę niniejszą rozpoczynamy (rozdział 2.) od prezentacji pierwowzoru systemu eksperckiego, tj. systemu z bazą wiedzy. Kluczowe znaczenie ma tu hipoteza Newella dotycząca modelu mechanizmów myślenia twórczego u człowieka. Rozdział ten kończymy sformułowaniem struktury funkcjonalnej systemu z bazą wiedzy. W rozdziale 3., po ogólnej charakterystyce zastosowań i funkcji systemu, omawiamy główne typy reprezentacji wiedzy oraz związane z tym trzy główne typy systemów eksperckich. I tak opisując reprezentację wiedzy za pomocą reguł produkcji przedstawiamy regułę produkcji jako podstawowy element języka systemów eksperckich pewnej klasy. System ekspercki to w gruncie rzeczy program komputerowy napisany w języku interpretowanym przez sprzęt określonego komputera. Dlatego też rozdział 4. zawiera omówienie cech wybranych języków sztucznej inteligencji oraz charakterystykę mikrokomputerów wybranej klasy (PC XT/AT) interpretujących te języki.

Dwa kolejne rozdziały (5. i 6.) poświęcamy omówieniu języków LISP i PROLOG oraz ich konkretnym realizacjom na komputerze klasy PC XT/AT. Prezentowane są te aspekty, które wiążą się

z mechanizmami systemu eksperckiego. Obydwa te rozdziały zawierają też przykładowe programy; celem tych przykładów jest ilustracja działania. W rozdziale 7. omówiono język SMALLTALK, który jak się wydaje może znaleźć zastosowanie w systemach eksperckich, choć nie cieszy się on taką popularnością jak języki LISP i PROLOG.

Opracowanie niniejsze jest wynikiem działania zespołu autorów: Z. Wrzeszcz: kierownictwo opracowania, rozdziały 1.,2.,3.,4., W. Przyłuski: rozdział 5., J.Kasprzyk: rozdział 6., A.Rowicki: rozdział 7.

2. SYSTEMY Z BAZĄ WIEDZY - ARCHETYP SYSTEMU EKSPERCKIEGO

Myślenie twórcze, a szczególnie rozwiązywanie problemów absorbowało uwagę filozofów od wieków [35] . Obecnie badania takie są dziedziną nauk poznawczych (ang. congnitive sciences) i skupiają wysiłki nie tylko gnoseologów, lecz także psychologów, lingwistów i cybernetyków. Inżynierskim odpowiednikiem powyższej dziedziny jest sztuczna inteligencja, której przedmiotem badań są systemy z bazą wiedzy mające zdolność rozwiązywania problemów, w przybliżeniu tak, jak to czyni człowiek, pod warunkiem umieszczenia wiedzy o problemie w pamięci systemu. System ekspercki jest szczególną postacią systemu z bazą wiedzy.

2.1 Model procesów twórczych u człowieka

Mówimy o problemie, gdy pragnąc czegoś nie wiemy, jak to osiągnąć. Przedmiot naszych pragnień może mieć postać materialną (np. jabłko na wysokim drzewie) lub abstrakcyjną (np. dowód twierdzenia) . Podejmując działalność prowadzącą do osiągnięcia celu wykonujemy wiele akcji, z których część ma charakter fizyczny (np. przemieszczanie się osobnika rozwiązującego problem, branie do ręki przedmiotów itd.) , część zaś odnosi się do organów percepcji osobnika (patrzenie, słuchanie) ; trzeci rodzaj to działanie umysłu: porównywanie elementów sytuacji, pamiętanie tych elementów, pamiętanie relacji między elementami itd.

Podjęcie akcji trzeciego rodzaju może być poprzedzone przez akcje drugiego rodzaju: osobnik rozwiązujący problem musi przyjąć do wiadomości sytuację początkową, zapamiętać ją, rozpoznać problem i podjąć decyzję o działaniu prowadzącym do celu, działać w sytuacji problemowej tak, by ten cel osiągnąć lub do niego się przybliżyć.

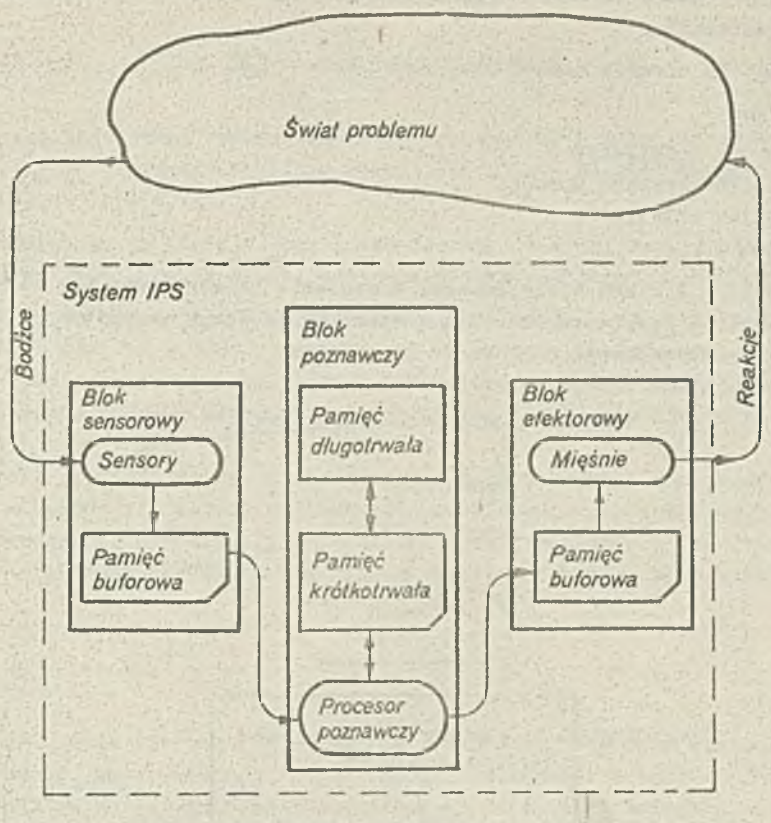
Powyższy opis pozwala nam wyróżnić pewne istotne zdolności systemu inteligentnego. Są to:
- odbiór bodźców otoczenia,
- wnioskowanie,
- interpretowanie relacji między faktami (obiektami itp.) ,
- interpretowanie znaczenia faktu,
- rozpoznanie prawdy stwierdzeń,
- uczenie się,
- korygowanie swoich działań na podstawie doświadczeń,
- adaptacja do nowej sytuacji.

Powstaje więc pytanie o to, jak zbudowany jest system o takich zdolnościach.

Zgodnie z hipotezą A. Newella i H. Simona [49] człowiek rozwiązując problemy działa jak pewien system przetwarzania informacji (ang. intelligent processing system - IPS) . System IPS ma następujące składniki [48]:
- receptory bodźców zewnętrznych,
- zbiór symboli fizycznych - każdy element tego zbioru jest reprezentantem obiektu ze świata problemu, mogą to być obiekty fizyczne i relacje między tymi obiektami podporządkowane określonym procesom fizycznym lub obiekty abstrakcyjne i związane z nimi zależności: symbol fizyczny jest repliką obiektu ze świata problemu, powstałą w pamięci IPS w wyniku odebranego bodźca,
- symbole fizyczne wraz z relacjami tworzące pewne asocjacje (wyrażenia) pamiętane przez pamięć długotrwałą,
- stały i ograniczony zbiór procesów podstawowych przeznaczonych do tworzenia nowych symboli fizycznych, porównywania symboli fizycznych, modyfikacji symboli fizycznych, wprowadzania

nowych asocjacji oraz kierowania asocjacji do otoczenia systemu,

- pamięć krótkotrwałą do reprezentatywnego ujęcia problemu i pamiętania bieżącego,
- pamięć długotrwałą do pamiętania asocjacji fizycznych - metod rozwiązywania problemu,
- układ interpretacyjny określający sekwencję procesów podstawowych.



Rys. 1. Model realizacji procesów twórczych u człowieka jako pewien system przetwarzania informacji

Obraz systemu IPS przedstawiono na rys. 1.

Zaproponowany przez Newella i Simona model mechanizmów myślenia twórczego u człowieka został zaakceptowany przez środowisko nauk poznawczych i przyczynił się do znacznego postępu w badaniach czynności myślenia u człowieka [36, 37, 1]

2.2 System z bazą wiedzy

Hipoteza Newella i Simona ma również ogromne znaczenie dla rozwoju badań nad komputerami, a szczególnie nad ich inteligentną wersją. Badania te, rozpoczęte w latach powojennych (1945-1954) pracami N. Wienera [72] i A.M. Turinga [67] , zatrzymały się przez pewien czas na tzw. systemach rozwiązujących problemy ogólne (ang. general problem solvers) [50] i systemach mechanicznego dowodzenia twierdzeń [7] . Dopiero przyjęcie symbolicznej reprezentacji wiedzy (symbole fizyczne Newella) oraz heurystyki jako podstawy do procesu wnioskowego (asocjacje fizyczne Newella) , spowodowało zasadniczy postęp w pracach nad komputerowymi systemami z bazą wiedzy. W roku 1960 opracowano język LISP do przetwarzania symbolicznego [40] . Język ten po dzień dzisiejszy jest uważany za podstawowy język sztucznej inteligencji.

Komputerowy system z bazą wiedzy (w dalszym ciągu będziemy używać określenia skróconego: system BW) jest funkcjonalnym odpowiednikiem systemu IPS. Oznacza to, że:

- przyjęto koncepcję symboli fizycznych i asocjacji fizycznych,

- przyjęto również koncepcję procesów podstawowych,
- w systemie istnieje pamięć krótkotrwała do pamiętania danych dla bieżącego kroku rozwiązywania, zwana często bazą danych,
- system zawiera pamięć długotrwałą, gromadzącą metody rozwiązywania problemu; nazywana jest ona bazą wiedzy,
- system zawiera również układ interpretacyjny, określający sekwencję procesów podstawowych dla bieżącego kroku rozwiązania.

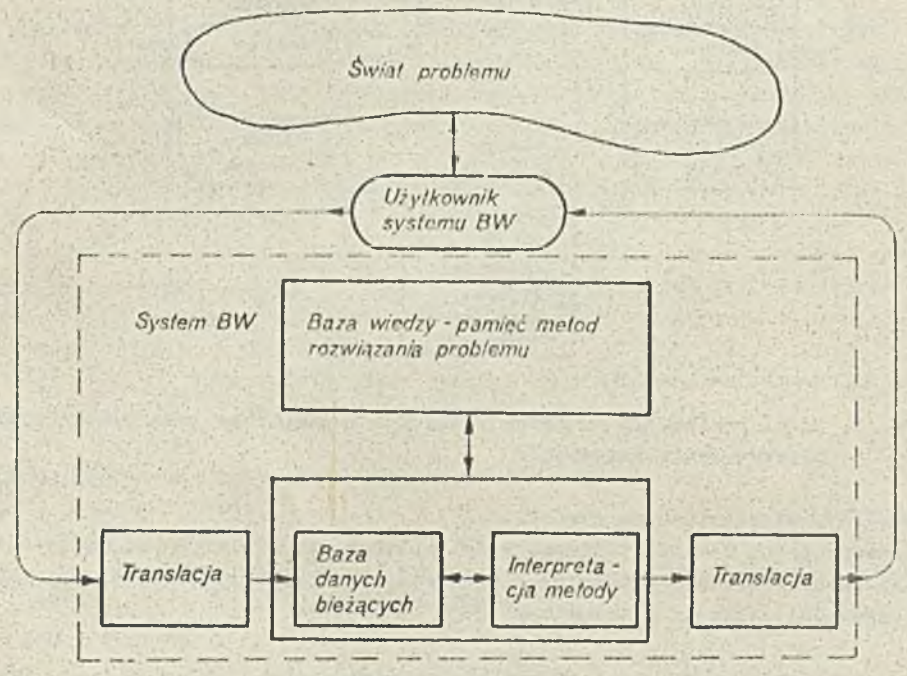
Działanie systemu BW określa następująca procedura:

(1) WYBÓR CELU,

(2) WYBÓR METODY ROZWIĄZANIA,
ZASTOSOWANIE WYBRANEJ METODY,

(3) OCENA WYNIKU ROZWIĄZANIA,
POWRÓT DO (1).

W miejsce bloku percepcji i bloku efektorowego, w systemie BW wprowadza się blok translacji wejściowej - tłumaczenia z języka użytkownika systemu BW na język wewnętrzny systemu BW i blok translacji wyjściowej - tłumaczenia w kierunku przeciwnym.
Strukturę systemu BW pokazano na rys. 2.



Rys. 2. System BW i jego otoczenie

Osobnego omówienia wymagają metody rozwiązywania problemu; bez tego działanie systemu będzie niezrozumiałe.

Metody rozwiązywania są dość zróżnicowane i zależą ściśle od typu reprezentacji wiedzy, typy zaś reprezentacji wiedzy, mimo że odpowiadają paradygmatowi symbolu fizycznego, są zależne od specyfiki problemu.

Różnorodność metod rozwiązywania istnieje nawet w obrębie tej samej klasy problemów. Interesująca nas klasa systemów BW (tzw. systemy eksperckie) preferuje pewne typy reprezentacji, takie jak reguły produkcji czy też ramy. Zagadnienie to omówimy w następnym rozdziale.

Na koniec należy nadmienić, że systemy eksperckie nie są jedyną rozwijającą się klasą systemów BW. Trzeba tu wymienić systemy:

- do tłumaczenia i generacji języka naturalnego [70],
- percepcyjno-efektorowe robotów przemysłowych [4],

- wielkich baz danych [51] ,
- sterowania procesami realnymi [68] , [76] .

Istnieją koncepcje opracowania systemu BW zdolnego realizować wszystkie te aplikacje. Może to jednak zrodzić potrzebę dynamicznej zmiany typu reprezentacji wiedzy i metodyki rozwiązywania. Cechy takie mają mieć tzw. komputery V generacji [45] .

3. BUDOWA SYSTEMU EKSPERCKIEGO

System eksperoki jest pewną szczególną formą systemu BW. System ten odwzorowuje mianowicie zachowanie eksperta rozwiązującego problemy określonej klasy. Ekspert dysponuje dwoma rodzajami wiedzy [41, 47, 26, 44, 27, 6] :

- wiedzą ogólnie dostępną, na którą składają się formuły, teorie, normy, ustalenia projektowe i realizacyjne oraz inne podobne treści dostępne dzięki publikacjom, dokumentom, nauczaniu,
- wiedzą specjalistyczną wyniesioną z doświadczenia, mającą charakter nieformalny, heurystyczny.

System eksperoki zawiera w swojej pamięci oba te rodzaje wiedzy, a wiedza heurystyczna ma tu znaczenie zasadnicze dla procesu rozwiązywania. Jest to więc istotna różnica w porównaniu ze starszym rozwiązaniem systemu mechanicznego wnioskowania [7] , w którym nacisk kładziono na metody formalne [15] .

System ekspercki gromadzi wiedzę w formie symboli fizycznych i ich asocjacji, podobnie jak jego prototyp - system BW. Metody poszukiwania rozwiązania przez system eksperoki nie polegają jednak na prostym przeszukiwaniu przestrzeni rozwiązań, co przy bardziej złożonych problemach jest nieefektywne; heurystyczna wiedza eksperta wybitnie przyśpiesza proces rozwiązania.

3.1 Rodzaje zastosowań

Eksperckie rozwiązanie systemu BW ma już swoją historię. Pierwsze opracowanie trzeba przypisać Feigenbaumowi [16] , którego system DENDRAL opracowany w latach międzywojennych w języku konwencjonalnym został następnie przebudowany do postaci bazującej na regułach produkcji. System DENDRAL jest do dziś wykorzystywany praktycznie.

Systemy eksperckie opracowano i zastosowano w praktyce do zagadnień, takich jak:

- analiza (np. interpretacja danych [5]),
- diagnostyka (określenie uszkodzeń w systemach technicznych [25] , określenie przyczyny chorób [60]),
- ciągła interpretacja sygnałów (intensywna terapia [14]),
- planowanie (programowanie eksperymentu w genetyce molekularnej [64]),
- projektowanie (konstruowanie urządzeń elektronicznych [39]),
- nauczanie [62] ,

Powyższe wyliczenie nie wyczerpuje możliwości aplikacyjnych systemu eksperckiego [66, 76, 52].

3.2 Odwzorowanie funkcji eksperta [65, 63, 9, 41, 26, 44]

Zakładamy, że system ekspercki odwzorowuje zachowanie człowieka, eksperta określonej działalności. Oznacza to, że system ekspercki powinien naśladować wszystkie te funkcje, które ekspert wykonuje rozwiązując problem z danej dziedziny. Rozpatrzmy więc te funkcje:

(1) Wnioskowanie: fakty dotyczące problemu oraz wiedza o metodach rozwiązywania podobnych problemów służą ekspertowi w procesie tworzenia rozwiązania konkretnego problemu.

Funkcja ta ma znaczenie podstawowe, lecz ekspert musi wykonywać również inne, pomocnic funkcje:

(2) żądanie i przyjmowanie nowych danych o problemie,
(3) wyjaśnianie swojego postępowania w procesie wnioskowym,

Oprócz funkcji (1), (2), (3) ekspert może:

(4) wykonywać dynamiczną reorganizację bazy wiedzy stosownie do specyfiki problemu,
(5) łamać przyjęte reguły wnioskowe,
(6) określać wcześniejsze związki między konkretnymi problemami a wiedzą eksperta: problem może być podejmowany jedynie wtedy, gdy mieści się w profilu wiedzy eksperta,
(7) określać rozwiązania przybliżone w przypadku, gdy problem zbliża się do granicy możliwości wynikających z wiedzy eksperta.

Implementacja funkcji (1) w systemie eksperckim może mieć różne formy, zależnie od przyjętej metody reprezentacji wiedzy. Funkcja (2) stanowi ważne uzupełnienie dla funkcji (1), zwłaszcza na etapie budowy systemu prototypowego, a także w przypadku rozszerzeń możliwości użytkowych systemów eksperckich. Funkcja (3) ma charakter pomocniczy, jednak może mieć znaczny wpływ na prawidłowość reakcji użytkownika w dialogu z systemem za pomocą funkcji (2) i (

Współczesne systemy eksperckie nie uwzględniają funkcji (4) - (7) bądź też uwzględniają je częściowo. Jak sądzimy, przyczyna leży w niedostatecznej znajomości podstaw ich implementacji.

Zróżnicowanie implementacji funkcji eksperta może występować również ze względu na rodz zastosowania. Na przykład systemy eksperckie wspomagające decyzję dotyczącą zachowania urząd w czasie realnym wymagają dodatkowo funkcji:

(8) przyjmowania sygnałów z procesów świata realnego,
(9) wysyłania sygnałów do procesów świata realnego.

3.3 Reprezentacja wiedzy

Zgodnie z koncepcją systemu BW zarówno dane o problemie, jak też metody rozwiązywania są ujmowane w formie pewnych symboli zwanych symbolami fizycznymi oraz asocjacji tych symbo Pozostaje jednak do wyjaśnienia, jaką konkretną formę przyjmują owe symbole przed przekazaniem ich komputerowi, który wykonuje funkcje systemu eksperckiego.

Sposób opisu problemu powinien być tak dobrany, aby obok sytuacji problemowych umożliwiał również odwzorowanie metod rozwiązywania problemu [75, 10, 63, 71, 34, 57, 77, 2, 12, 46, 29, 21, 22] .

W niniejszym rozdziale przedstawimy najpierw reprezentacje typu deklaratywnego, a następnie reprezentacje typu funkcyjnego. Sposób reprezentacji symboli fizycznych ma duży wpływ na konstrukcję całego systemu eksperckiego.

3.3.1 Reprezentacja typu deklaratywnego

Przyjmujemy, że punkt wyjścia stanowi pewna sytuacja problemowa określona przez zbiór elementów i ich relacji. Oczywiście, elementy te mogą mieć charakter materialny lub abstrakcyjny. Sytuacja ta ulega przemianom w miarę rozwiązywania problemu. Zachodzi więc potrzeba wprowadzenia zapisu (deklaracji) tych sytuacji.

Najprostszym przypadkiem jest pojedyńczy element. Niech to będzie element, o którym wiemy bardzo mało, na przykład

(OPORNIK)

Zapis

(GRUPA1 OPORNIK TRANZYSTOR DIODA)

zawiera już większą wiedzę: wiemy, że elementy takie jak opornik, tranzystor i dioda stanowi pewną grupę zwaną GRUPA1.

Postępując podobnie możemy zapisać na przykład zadanie polegające na stwierdzeniu przynależności opornika do określonej grupy

(ZADANIE1 GRUPA2 OPORNIK)

Podane zapisy są przykładami reprezentacji w formie symbolu wektorowego lub w skrócie v-symbolu. Ogólnie zapis ten wyrazimy następująco:

<v-symbol> ::= (<identyfikator symbolu fizycznego>{<identyfikator symbolu fizycznego>})
<identyfikator symbolu fizycznego> ::= <znak ∈ Z>{<znak ∈ Z>}

W powyższej definicji znak jest dozwolonym elementem zestawu znaków Z, które jednoznacznie interpretuje komputer implementujący projektowany system eksperoki. Przykładem może być zestaw znaków alfanumerycznych według standardu ASCII, popularnie stosowany w komputerach personalnych.

Zapis za pomocą v-symbolu pozwala więc ująć zarówno sytuację problemową, jak i sam problem do rozwiązania.

Element sytuacji problemowej może mieć wiele właściwości, stanów fizycznych, jeśli jest to element materialny, itd. Wiedzę o tym możemy reprezentować efektywnie przez jednorodne zapisy trójek symbolicznych w sposób następujący:

<identyfikator elementu><atrybut><wartość>

Na przykład:

(OPORNIK OPORNOŚC 300)

W określonym typie problemu może okazać się bardzo przydatne podanie wielu właściwości danego elementu.

Na przykład zapis:

(OPORNIK12 OPORNOSC 300 TEMPERATURA 100°C)

określa fakt, że OPORNIK12 o oporności 300 ma temperaturę powierzchni 100°C. Tym samym przeszliśmy od prostego odnotowania elementów sytuacji problemowej, do zapisu złożonych faktów w postaci symboli z atrybutami lub krócej a-symboli:

<a - symbol> ::= <element><atrybut><wartość>{<atrybut <wartość>}

Podobnie jak w poprzedniej definicji i tu określenia stojące z prawej strony przyjmują znaczenie identyfikatorów symboli fizycznych. Ponieważ jednak identyfikatory te mogą pochodzić z języka naturalnego, to ich zapisy mogą podlegać pewnym restrykcjom. Na przykład:

<element> ::= <identyfikator I rodzaju>
<atrybut> ::= <identyfikator I rodzaju>
<wartość> ::= <identyfikator II rodzaju>
<identyfikator I rodzaju> ::= <znak ∈ A><znak ∈ Z>{<znak ∈ Z>}
<identyfikator II rodzaju> ::= <znak ∈ Z>{<znak ∈ Z>}

gdzie A - zestaw znaków alfabetu łacińskiego.

Najbardziej rozwiniętą formą deklaratywnej reprezentacji wiedzy są sieci semantyczne. Pierwszeństwo zastosowania sieci semnatycznych do zapisu wiedzy przypisuje się Quillianowi [54].

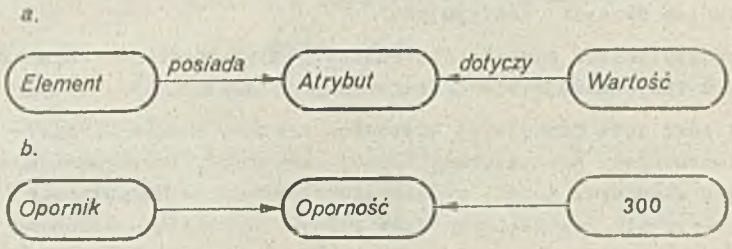
W najprostszej formie sieć semantyczną można określić jako zbiór węzłów i łączących te węzły gałęzi. Węzłami mogą być elementy sytuacji problemowej, a więc [30]

- obiekty fizyczne (np. opornik, układ scalony itd.),
- obiekty konoeptualne (np. schemat, projekt-5 itd.),
- atrybuty obiektu (np. zimny, gorący itp.),
- klasa, zbiór obiektów (nadzbiór),
- obiekt jako składnik obiektu złożonego, np. ręka człowieka.

Połączenia reprezentują:

- relacje między obiektem i deskryptorem np. (opornik) jest (element bierny),
- relacje między obiektami,
- relacje między klasą obiektów a egzemplarzem,
- relacja między składnikiem a obiektem złożonym.

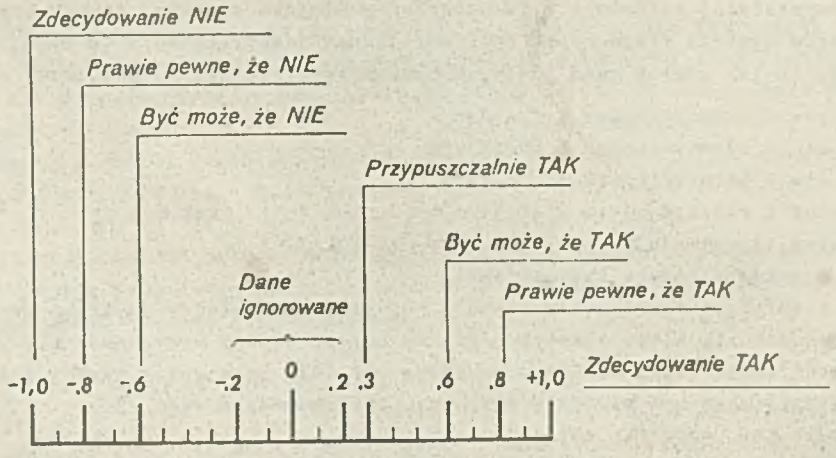
Zwróćmy uwagę, że trójka symboliczna może być traktowana jako przypadek sieci semantycznej. Przykład taki pokazano na rys. 3.



Rys. 3. Trójka symboliczna jako sieć semantyczna:
a. postać ogólna,
b. przykład

● Reprezentacja faktów niepewnych. Istnieje wiele problemów, w których deklaracja faktów - elementów sytuacji problemowej - będzie wymagać odnotowania wątpliwości, niepewności. Może to się odnosić zarówno do wartości atrybutu elementu, jak i do samej konkluzji. Krytykując statyczne podejście do tego zagadnienia Shortliffe w swym systemie MYCINE [80] wprowadza współczynnik ufności (ang. certainty factor) wyrażający wiarygodność określonego składnika danych lub wiedzy.

Istnieje wiele sposobów reprezentowania wiarygodności. Zagadnienie to można nawet rozszerzyć na reprezentację wiedzy zdrowego rozsądku [77] , bardzo ważną w pewnych typach problemów (zwłaszcza w pierwszej fazie rozwiązania).



Rys. 4. Skala współczynników pewności systemu MYCINE

Rysunek 4 pokazuje skalę współczynników ufności, ograniczoną wartościami -1,0 oraz +1,0. Współczynnik -1,0 wyraża przeświadczenie, że fakt jest fałszywy, zaś +1,0 wyraża przeświadczenie, że fakt jest prawdziwy. Rozwiązanie to pochodzi z systemu MYCINE.

3.3.2 Reprezentacje typu funkcjonalnego

Opisane wcześniej reprezentacje typu deklaratywnego nie ujmują metod rozwiązywania problemu. Są one stwierdzeniami dotyczącymi sytuacji problemowej. Reprezentacje funkcjonalne mają za zadanie opis zarówno sytuacji problemowej, jak też i metod rozwiązywania problemu.

W praktyce systemów eksperckich trzy spośród sposobów reprezentacji wiedzy znalazły szczególnie szerokie zastosowanie, są to:

- ramy,
- programowanie logiczne,
- reguły produkcji.

Ograniczmy się więc do omówienia trzech powyższych sposobów zapisu wiedzy.

3.3.2.1 Rama

Pierwszeństwo sformułowania ramy jako konstrukcji językowej do reprezentacji wiedzy należy przypisać Minskyemu [43] , który zastosował ramę do zapisu wiedzy o scenie zawierającej obiekty materialne, a następnie do analizy tak opisanych scen. Minsky skupiał uwagę na semantyce sceny i jej reprezentacji przez ramę. Późniejsze prace nad ramami koncentrowały się na definicji problemu i jego rozwiązywaniu [17, 8] .

Własności ramy. Każda rama jest pewną jednostką konceptualną korespondującą z określoną jednostką ze świata problemu, taką jak obiekt rzeczywisty, grupy obiektów rzeczywistych, obiekty abstrakcyjne, obiekty wirtualne, zorganizowane grupy obiektów, zdarzenia, zagadnienia, własności itp.

Rama cechuje się pewną strukturą utworzoną z sekwencji działek, które zawierają określenia dotyczące jednostek ze świata problemu. Działka limituje te informacje.

Działka może zawierać:

- wartość atrybutu,
- wskaźnik do innej ramy,
- regułę lub zbiór reguł,
- procedurę do obliczania wartości i inne.

W ogólnej formie rama zawiera dwa typy działek, są to działki typu własnego z informacją o danej jednostce ze świata problemu oraz działki typu komunikacyjnego służące do tworzenia sieci na podobieństwo sieci semantycznej [54] . Sieć ram tworzy bazę wiedzy o problemie i przez to staje się głównym ogłonem systemu eksperckiego bazującego na ramach (system BWR).

Rama jest tak skonstruowana, że odwzorowuje przede wszystkim strukturę i własności problemu.

Proces wnioskowania. Mówiąc ogólnie, w wyniku procesu wnioskowania w systemie na bazie ram, następuje powiększenie zbioru stwierdzeń uznanych za prawdziwe. Kolejne kroki procesu bazują na strukturalnych własnościach ramy i taksonomii wpisanej w konstrukcję ramy. Główną metodą wnioskowania jest dziedziczenie. Jeśli na przykład rama o nazwie OBIEKTY.FIZYCZNE ma połączenie do podklasy UKŁADY.ELEKTRON, zaś rama UKŁADY.ELEKTRON ma połączenia do podklasy będącej ramą MIKROPROCESORY, to mechanizm wnioskujący wyprowadzi wniosek (przeświadczenie) , że MIKROPROCESORY są podklasą OBIEKTY.FIZYCZNE bez odnoszenia się do pomocy innych mechanizmów. Jest to więc szybka realizacja wnioskowania w porównaniu np. do systemów na bazie programowania logicznego.

Inny rodzaj mechanizmów to wprowadzenie ograniczeń w postaci klasy wartościującej (ang. Value Class) oraz wartości kardynalnej (ang. cardinality) System KEE [31]. Oba te ograniczenia stają się wejściami określonych działek. Odnosząc to do ramy MIKROPROCESORY zapiszemy

```
Działka-Własna:   UŻYTKOWNIK
     WartKlasa:   INŻYNIER.ELEKTRONIK
     KardKlasa:   10000
```

Ograniczenie takie pozwoli wnioskować o zapotrzebowaniu na układy scalone w określonych warunkach.

Opisane wyżej ograniczenia zawierają warunki konieczne, które nie są dostateczne, dzięki czemu istnieje możliwość wykluczenia przynależności egzemplarza do danej klasy. Natomiast stwierdzenie przynależności można uzyskać wprowadzając warunki dostateczne. Można to wykonać za pomocą reguł produkcji określających przynależność do klasy.

Użycie reguł produkcji jako treści działki dostarcza też możliwości zbudowania sprzęgu z otoczeniem i aktywnego śledzenia zachowania tego otoczenia. Sprzęg i układy z nim związane pełnią funkcję demonów z modelu Selfridge'a [58].

Ramy mogą tworzyć taksonomie przez użycie dwóch mechanizmów, z których pierwszy określa połączenie z tytułu przynależności danej ramy do klasy wyższej, drugi zaś dostarcza połączeń do klas podporządkowanych.

W ciągu ostatnich kilku lat ramy zyskały znaczną popularność w rozwiązywaniu pewnych problemów o dobrze zdefiniowanej strukturze i przejrzystym porządku hierarchicznym. Są to często zagadnienia natury konstrukcyjnej (inżynierskie). Budowa ramy odwzorowuje tu podstawowe bloki konstrukcyjne [2, 31, 3].

Aby zilustrować powyższe omówienie reprezentacji wiedzy, podajemy przykład zapisu ramy dla mikroprocesora jako elementu konstrukcyjnego. Przykład ten odpowiada konwencji systemu narzędziowego KEE (rys. 5).

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  Rama: MIKROPROCESORY w bazie wiedzy PROJEKTOWANIE
   Nadklasy: ELEMENTY.KONSTRUKCYJNE
   Podklasy: ELEMENTY.ELEKTRONICZNE, ELEMENTY.PÓŁPRZEWODNIKOWE
   Przynależność: KLASY.OBIEKTÓW.FIZYCZNYCH
---- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    .
    .
    .
DziałkaPrzynal: WYSOKOSC z OBIEKTY.FIZYCZNE
   KlasaWartości: LICZBY.NATURALNE
   Liczebność.Min: 1
   Liczebność.Max: 1
   Jednostki: MILIMETRY
   Komentarz: "Wysokość w milimetrach"
   Wartości: Nie znane

DziałkaPrzynal: WAGA z OBIEKTY.FIZYCZNE
   KlasaWartości: Liczby.NATURALNE
   Liczebność.Min: 1
   Liczebność.Max: 1
   Jednostki: GRAMY
   Komentarz: "Jednostki w gramach"
   Wartości: Nie znane
.
.
.
DziałkaWłasna: NAJB.ZŁOŻONY z KLASA.OBIEKTÓW.FIZYCZNYCH
   Klasa Wartości: MIKROPROCESORY
   Liczebność.Min: 1
   Liczebność.Max: 1
   Komentarz: "Najbardziej złożony mikroprocesor"
   Wartości: Nie znane
.
.
.- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Rys. 5. Przykład zapisu ramy dotyczącej mikroprocesora

3.3.2.2 Programowanie logiczne

Problem można również ująć w formie klauzul stanowiących wariant rachunku predykatów [32, 33, 28, 61].

Najprostszą formą wyrażenia jest atom stwierdzający określoną relację między określonymi obiektami. Wyrażenie K (Jan, Adam) opisuje relację podległości służbowej: Jan jest kierownikiem Adama. Operator implikacji odwrotnej:- służy do zapisu bardziej złożonych obiektów. Na przykład Jan jest kierownikiem Adama, jeśli Jan jest kierownikiem pracowni

P (Jan, Adam) :- K (Jan, Adam)

Bardziej złożona struktura

P (Jan, Tom) :- K (Jan, Adam) , K (Adam, Tom)

mówi, że Jan jest przełożonym Toma, jeśli Jan jest kierownikiem Adama i Adam jest kierownikiem Toma.

Powyższe zdania wyrażały relacje między stałymi obiektami, tj. były to wyrażenia dotyczące szczególnych faktów, w których są zaangażowane konkretne obiekty. Bardziej ogólne wyrażenia dotyczące wielu obiektów określonego rodzaju zapiszemy stosując zmienne w miejsce stałych obiektów, np.

P (x, y) :- K (x, y)

Powyższy zapis oznacza, że x jest przełożonym y, jeśli x jest kierownikiem pracowni, w której pracuje y .

Fakty bądź konkluzje zapisywane są jako zdanie pewne, niezależne od przesłanek, np.

M (Tom) :-
Ka (Anna) :-

co oznacza: Tom jest mężczyzną, Anna jest kierowniczką.

- Proces wnioskowania. Celem rozwiązania określonego problemu dane są:
  - zbiór reguł wnioskowania
  - zbiór faktów
  - procedura wnioskowania

poszukujemy natomiast konkluzji w postaci nowych faktów dotyczących konkretnych obiektów.

Jedna z najczęściej stosowanych procedur wnioskowych wykorzystuje zasadę rezolucji [55]. Rozpatrzmy przykład. Dane są dwie klauzule

P (Jan, Adam) :- K (Jan, Adam)
D (Jan, Tom) :- P (Jan, Adam) , P (Adam, Tom)

Druga klauzula mówi, że Jan jest dyrektorem Toma, jeśli Jan jest przełożonym Adama i Adam jest przełożonym Toma. Podstawienie atomu lewej strony pierwszej klauzuli w miejsce pierwszego atomu prawej strony drugiej klauzuli daje w konkluzji

D (Jan, Tom) :- K (Jan, Adam) , P (Adam, Tom)

czyli: Jan jest dyrektorem Toma, jeśli Jan jest kierownikiem Adama i Adam jest przełożonym Toma. Przykład powyższy dotyczy rezolucji przy dwóch identycznych atomach. Weźmy jednak bardziej ogólny przypadek ze zmiennymi, wtedy atomy nie będą identyczne

P (x, Adam) :- K (x, Adam)
D (Jan, z) :- P (Jan, y) , P (y, z)

Dokonujemy unifikacji zmiennych w obu regułach, tj. zmiennej x odpowiada Jan, zaś zmiennej y odpowiada Adam. W konkluzji otrzymamy

D (Jan z) :- P (Jan, Adam) , P (Adam, z)

- Gramatyka reguł i faktów. Zarówno reguły, jak i fakty mają postać klauzul, a zapis problemu do rozwiązania jest po prostu zbiorem takich klauzul.

Klauzula ma następującą postać ogólną

<klauzula> ::= <atom> : -{,<atom>}|<atom> : - atom>{,<atom

<atom> ::= <identyfikator relacji>(<term{,<term>})

<term> ::= <identyfikator stałej>|<identyfikator zmiennej>|<funkcja>

<funkcja> ::= <identyfikator funkcji>(<term>{<term>})

<identyfikator relacji> ::= <znak alfabetu>{<symbol 1>}

<identyfikator funkcji> ::= <znak alfabetu>{<symbol 1>}

<identyfikator stałej> ::= <mały znak alfabetu>{<symbol 1>}

<symbol 1> ::= <znak alfabetu>|<cyfra>

Wyrażenie zapisane zgodnie z powyższą gramatyką stwierdza, że jego lewa strona(przed operatorem : -)jest prawdziwa tylko wtedy, gdy każdy atom prawej strony wyrażenia ma wartość prawdy.

3.3.2.3 Reguła produkcji

Ogólna postać struktury zawierającej wiedzę o problemie ma następującą postać [70, 23] .

jeśli <sytuacja> to <akcja>

lub krócej

<sytuacja> → <akcja>

Pierwszeństwo zastosowania takiej konstrukcji przypisuje się [13] Postowi [53] . Działanie systemu eksperckiego z reprezentacją wiedzy w postaci reguły produkcji zasadza się na tak zwanej przestrzeni stanu problemu. Zgodnie z tą zasadą sytuacja problemowa odpowiada pewnemu stanowi procesu rozwiązującego, który jest pamiętany na bieżąco w bazie danych systemu.

Lewa strona reguły wyraża pewien warunek w odniesieniu do stanu bazy danych. Operacja dopasowania (ang. matching) bada zgodność wzorca sytuacji określonego wybraną regułą ze stanem rozwiązania zawartym w bazie danych. Jeśli wynik badania jest pozytywny, to następuje wykonanie prawej strony reguły, a więc będzie zrealizowana określona akcja ze zbioru akcji dopuszczalnych. Główna z tych akcji dotyczy zmiany stanu rozwiązania. Zmiana w bazie danych powoduje zmianę warunków realizacji dla innych reguł, które mogą zadziałać w następnym cyklu. Wybór reguł ma charakter niedeterministyczny, natomiast w obrębie danej reguły działanie systemu jest zdeterminowane.

Prawa część reguły zawiera operator (jeden lub więcej) , który realizuje daną akcję. Ogólnie biorąc są to operatory:

(1) działające na bazę danych w pamięci operacyjnej, które:
- wprowadzają nowe dane,
- usuwają część danych z pamięci;

(2) odnoszące się do użytkownika systemu, tj. operatory, które:
- żądają wprowadzenia nowych danych,
- wysyłają komunikat o przebiegu procesu wnioskowania,
- wysyłają komunikat o możliwych lub koniecznych akcjach użytkownika systemu, skierowanych na środowisko problemu,

(3) odnoszące się do urządzeń ze środowiska problemu w przypadku, gdy system eksperoki jest sprzężony ze środowiskiem problemu za pomocą przetworników: wtedy będą to operatory, które:
- wprowadzają dane z urządzeń środowiska problemu i kierują je do PO jako elementy bazy danych,
- kierują do urządzeń środowiska problemu komendy określające wykonanie przez te urządzenia wybranych operacji.

Reguła produkcji jest więc pewnym minimalnym programem definiującym działanie systemu eksperckiego będącego szczególną postacią systemu BW.

●Struktura systemu eksperckiego BWP. W systemie niezbędny jest pewien układ interpretacji reguły, którego zadanie polega na:

- dopasowaniu lewej strony reguły do stanu rozwiązania zawartego w bazie danych systemu oraz
- wykonanie akcji prawej strony reguły w przypadku dopasowania.

Układ ten wraz z selektorem reguł tworzy układ sterowania procesem wnioskowania - układ SPW. W literaturze angielsko języcznej układ SPW nosi często nazwę maszyny wnioskującej (ang. inferenoe engine) .

Ponadto w strukturze systemu eksperokiego BWP znajdują się:

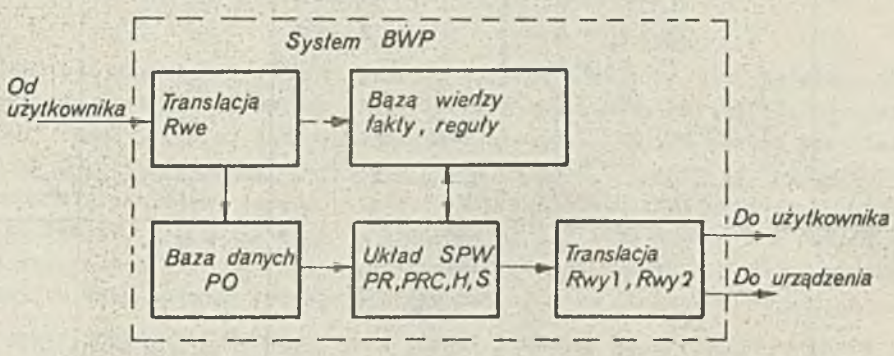
- ozłon translaoji wejściowej - jego zadanie polega na tłumaczeniu problemu na język reprezentaoji wewnętrznej systemu,
- ozłon translaoji wyjściowej - tłumaczenie wyników, poleceń, komentarzy na język użytkownika systemu.

Budowa układu sterowania prooesem wnioskowania jest zależna od wybranego kierunku prowadzenia wywodu.

Proces wnioskowania można prowadzić:

- od przesłanek do wniosków (w skróoie nazywany "wiązanie w przód" od ang. forward-chaining) lub też
- od wniosków do przesłanek("wiązanie wstecz" od ang. backward chaining).

Możliwe jest też postępowanie mieszane.



Rys. 6. Struktura systemu eksperokiego z bazą wiedzy w formie reguł produkoji

Rozpatrzmy działanie pewnego przykładu systemu eksperokiego BWP, którego strukturę zamieszczamy na rys. 6. W systemie tym wyróżniamy:

PO - pamięć operacyjną gromadzącą wyjściowe dane o problemie oraz bieżący stan problemu; jest to baza danych o problemie,

PR - pewna pomocnicza pamięć reguł gromadząca reguły produkoji w liczbie LICZBA-R służące do rozwiązania bieżącego problemu,

PRC - pamięć reguł częściowo dopasowywanych; w wyniku zastosowania procedury M1 (lsPR, PO) elementy lewej strony niektórych reguł są dopasowane do treści PO.

S - rejestr pomocniczy ujawniający prawdziwość hipotezy zawartej w H,

H - rejestr hipotezy,

Rwe - rejestr wejściowy

Rwy1 - rejestr wyjściowy do komunikacji z użytkownikiem

Rwy2 - rejestr wyjściowy do komunikaoji z urządzeniem

Układ SPW wykonuje m.in. dwie operacje dopasowania:

- M1 (lsPR, PO) - dopasowania lewej strony reguł zawartych w PR do zawartości PO Reguły, których lewa strona (lsPR) ma wszystkie elementy zgodnie z PO, powodują przeniesienie treści swojej prawej strony do PO. Reguły, których część elementów lewej strony jest zgodna z PO, są wyróżnione przynależnością do PRC,
- M2 (H, PO) - dopasowanie rejestru hipotezy do PO: w przypadku zgodności rejestr S:= TRUE.

Aby zilustrować działanie nowego uproszczonego mechanizmu SPW, posłużymy się równie

prostym zestawom reguł wziętym od Watermanna [69] ; zestaw reguł umieszczamy w pamięci reg
PR. Przebieg procesu wnioskowania pokazujemy w tabl. I.

Tablica I. Przebieg procesu wnioskowania

Operacja	PR	PRC	PO	S
	A · B · C → D D · F → G A · J → G B → C F → B L → J G → K		A F	
M2 (H, PO)				FALSE
M1 (1s PR, PO)	(A) · B · C → D D · (F) → G (A) · J → G B → C (F) → B L → J G → K	(A) · B · C → D D · (F) → G (A) · J → G	A F B	
M2 (H, PO)				FALSE
M1 (1s PR, PO)	(A) · (B) · C → D D · (F) → G (A) · J → G (B) → C (F) → B L → J G → K	(A) · (B) · C → D D · (F) → G (A) · J → G	A F B C	
M2 (H, PO)				FALSE
M1 (1s PR, PO)	(A) · (B) · (C) → D D · (F) → G (A) · J → G (B) → C (F) → B L → J G → K	D · (F) → (G) (A) · J → (G)	A F B C D	
M2 (H, PO)				FALSE
M1 (1s PR, PO)	(A) · (B) · (C) → D (D) · (F) → G (A) · J → G (B) → C (F) → B L → J G → K	(A) · J → G	A F B C D G	
M2 (H, PO)				FALSE
M1 (1s PR, PO)	(A) · (B) · (C) → D (D) · (F) → G (A) · J → G (B) → C (F) → B L → J (G) → K	(A) · J → G	A F B C D G K	
M2 (H, PO)				TRUE

Działanie układu SPW w obrębie pojedyńczego pakietu reguł można więc opisać następującym prostym programem:

```
procedure WNIOSKOWANIE ( );
   .
   .
   .
begin
  N := ϕ ;
repeat
    N := N+1 ;
    M1 (ls PR,PO) ;
    M2 (H, PO ) ;
until (S = TRUE) or (N = Liczba-Reguł) ;
if (S = TRUE) then (K = TRUE) else (K = FALSE)
end;
```

Wnioskowanie z "wiązaniem wsteoz" polega, ogólnie mówiąc na tym, że pierwotna hipoteza K w rejestrze H zostaje zastąpiona (oczywiście w przypadku, gdy M2 (H,PO) : FALSE) przez lewą stronę tej reguły, której strona prawa jest zgodna z K . W naszym przykładzie jest to G→K . Dowodząc G poszukujemy zgodności dla D·F→G, A·J→G . Reguły te tworzą tzw. zbiór konfliktowy. Zgodnie z określoną zasadą rozwiązywania konfliktu [42] wybieramy jedną z tych reguł - niech to będzie D·F→G. Staramy się dopasować jej lewą stronę i z chwilą wykonania tego zadania wprowadzamy G do PO, a następnie K do PO., dowodząc w ten sposób pierwotnej hipotezy. Mamy tu więc pewnego rodzaju wywód z nawracaniem (ang. baok tracking) .

●Zapis reguły produkcji. Zarówno zapis treści problemu w PO, jak też zapis jego rozwiązania w PR (dany w postaci zbioru reguł) musi być oparty na ustalonym typie danych. Ogólnie biorąc na treść pamięci PO składają się PO-elementy:

<PO-element > ::= < v-symbol >| < a-symbol>

Treść PO-elementu może ulegać zmianie w trakcie rozwiązywania problemu.

Podany wyżej zapis PO - elementów musi być zgodny z zapisem elementów (wzorców) lewej strony reguły i z zapisem elementów - argumentów operacji prawej strony reguły. A oto zapis reguły produkcji:

<reguła produkcji> ::= < nazwa reguły><sytuacja>→<akcja>
<sytuacja> ::= < element> <wzorzec>{&<wzorzec>}
<wzorzec > ::= < atrybut><wartość>
<atrybut > ::= < identyfikator stałej> | <identyfikator zmiennej>
<akcja > ::= WNIES < wzorzec > | MODYF < wartość wzorca No.>|
USUN < wzorzec No.> | PISZ < wartość>{< wartość >}

Zadanie funkcji M( ) w układzie SPW polega na stwierdzeniu dopasowania wzorców reguły (opisu sytuacji) do elementów stanu rozwiązania (elementów w PO).

Sprawdzanie dotyczy dopasowania każdego składnika we wzorcu i elemencie pamięci. Zasada dopasowania składników jest następująca:

- symbol stały lub liczba odpowiada jedynie identycznym stałym,
- zmienna odpowiada dowolnemu symbolowi lub liczbie; jeśli dana zmienna występuje w kilku wzorcach danego zapisu sytuacji (danej reguły), to wszystkie egzemplarze tej zmiennej przyjmują tę samą wartość.

Występujące z prawej strony funkcje tłumaczymy następująco. Funkcja WNIEŚ tworzy w pamięci operacyjnej nowy element (patrz przykład tablicy I), funkcja MODYF zmienia w danym elemencie PO wartość o numerze no. wartości na wartość nową, funkcja USUŃ usuwa wzorzec o numerze no.wzorca z danego elementu. Ostatnia z funkcji PISZ odnosi się do użytkownika wypisując ustalony przez daną regułę tekst. Szczególnie funkcja ta może być używana do żądań nowych danych.

Przedstawiona tu konstrukcja reguły produkcji jest zbliżona do rozwiązań zastosowanych w językach - narzędziach do tworzenia praktycznych systemów eksperckich. Są to np. języki z rodziny OPS [18, 19, 20] opracowane w Instytucie Technologii MIT i w Carnegie-Mellon-University (USA).

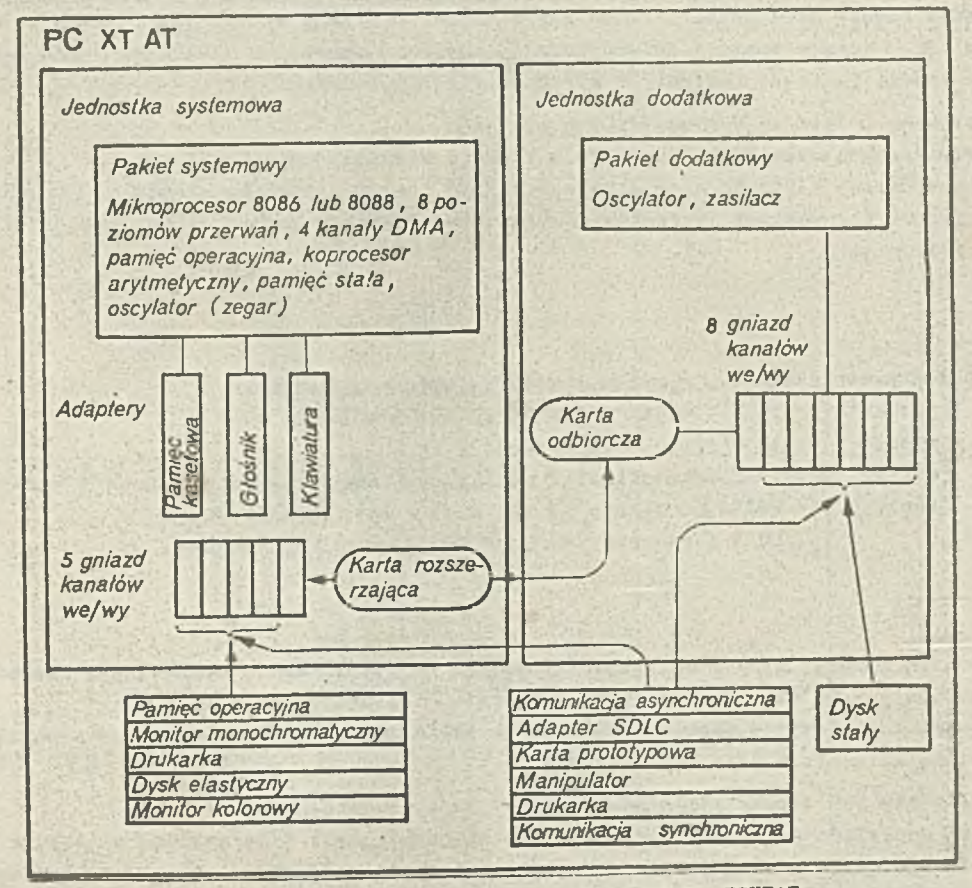
4. JĘZYKI IMPLEMENTACJI

W poprzednich rozdziałach omówiliśmy główne mechanizmy systemu eksperckiego. Rozdział niniejszy poświęcimy wyborowi języka do zapisu systemu eksperckiego, który ma być interpretowany przez mikrokomputer.

4.1 Charakterystyka mikrokomputera

Już na wstępie założyliśmy, że chodzi nam o możliwie najszersze stosowanie systemów eksperckich w praktyce. Założenie to skłania do wyboru środków komputerowych możliwie najłatwiej dostępnych, takich które są lub będą produkowane w kraju. Jednocześnie wybrane komputery muszą mieć znaczną moc obliczeniową.

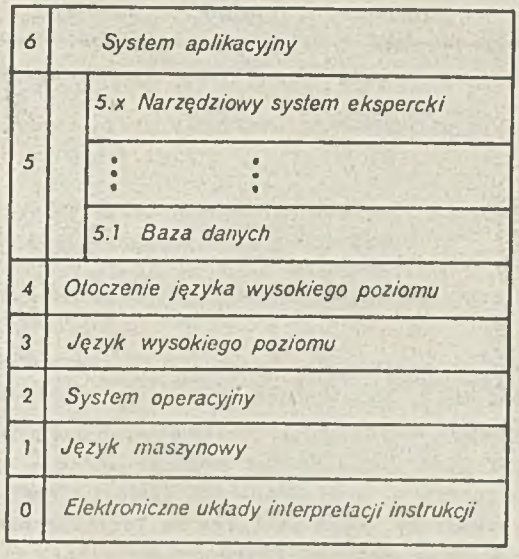
Wybór padł na komputery MAZOVIA 1016, 2016 kompatybilne z komputerami PCXT, PC AT. Mikrokomputery linii MAZOVIA są jednocześnie zgodne z linią rozwoju maszyn o architekturze M16-i zdefiniowanej w wyniku prac w ramach Sekcji Architektury SM EMC [56] . Główne charakterystyki mikrokomputera wybranej klasy przedstawiamy na rys. 7.



Rys. 7. Struktura mikrokomputera klasy PCXTAT

4.2 Wybór poziomu i typu języka implementacji

System eksperoki to w gruoie rzeozy program komputerowy napisany w języku interpretowanym przez sprzęt komputera danego typu. W mikrokomputerze profesjonalnym, podobnie jak w dużej maszynie uniwersalnej, można wyróżnić wiele poziomów interpretacyjnych: poziom 1 (tuż nad układami elektronicznymi do interpretacji podstawowej listy instrukcji) zajmuje język maszynowy, poziom 2 - to poziom systemu operacyjnego itd. Całość pokazano na rys. 8.



Rys. 8. Poziomy oprogramowania mikrokomputera profesjonalnego

Ze względu na wymóg efektywności programowania nowych systemów eksperckich wybiera się zwykle co najmniej język poziomu wysokiego. Pisanie programów dla systemów sztucznej inteligenoji ma charakter eksperymentalny. Programy takie kształtują się w miarę rozwoju pojmowania problemu przez projektanta systemu. Dlatego też do pisania programu SE potrzebny jest nie tylko język wysokiego poziomu, ale też odpowiednie jego otoczenie [73] .

Dokonując wyboru typu języka wysokiego poziomu bierzemy pod uwagę przede wszystkim cechy problemów i systemów do rozwiązywania tych problemów: cechy te przedstawiliśmy w rozdziałach poprzednich. Wynika stąd następująca lista głównych zdolności wybranego języka wysokiego poziomu:

- łatwa manipulacja strukturami symbolicznymi,
- przetwarzanie list symboli i ich połączeń jako struktur podstawowych,
- dopasowanie do wzorca - jedna z podstawowych i masowo występujących operacji układu interpretacji,
- możliwość przekazywania procedur jako danych i tworzenia procedur w trakcie procesu wnioskowego,
- możliwość zapisania systemu w postaci zbioru małych, zrozumiałych dla twórcy modułów,
- możliwość formułowania struktur danych przez użytkownika,
- możliwość formułowania struktur sterujących,
- możliwość komponowania procedur i danych,
- zmienne bez typu,
- rekursja.

Język bazowy dla SE powinien ponadto dysponować pewnym otoczeniem programistycznym, które umożliwiałoby [11] :

- automatyczną alokację pamięci,
- komunikację "użytkownik - system".

W tablicy II podajemy zestawienie wymienionych zdolności w odniesieniu do ozterech języków LISP, PROLOG, FORTRAN,C, z których dwa pierwsze to standardowe języki sztucznej inteligencji, żaś FORTAN i C to języki tradycyjne typu imperatywnego.

Tablica II. Cechy języków implementacji

Zdolność języka	LISP	PROLOG	FORTRAN	C
Manipulacje symbolami	+	+	-	-
Przetwarzanie list	+	+	-	-
Dopasowanie do wzorca	-	+	-	-
Procedury jako dane	+	+	-	+
Kod jako struktura własna	+	+	-	-
Zamienne bez typu	+	+	-	-
Rekursja	+	+	-	+
Procedury	+	+	+	+

Jak to wynika z powyższego zestawienia język PROLOG uzyskał maksimum punktów, tylko w jednym punkcie gorszy jest LISP. Dlatego też obydwa te języki uznaje się obecnie jako bazowe języki programowania systemów sztucznej inteligencji, dla nich też są budowane specjalne maszyny, tzw. LISP - maszyny [59] i PROLOG - maszyny [38] .

Interesującym również językiem jest SMALLTALK [24] - przedstawiciel grupy języków obiektowych. Język ten, jak sądzimy, może posłużyć do formułowania systemów eksperckich na bazie ram; rama wystąpi wtedy jako obiekt.

Wymienione trzy rodzaje języków (tj. LISP, PROLOG, SMALLTALK) opiszemy w kolejnych trzech rozdziałach. W dwóch pierwszych przedstawimy, obok ogólnych charakterystyk języka, również ważniejsze dane o konkretnych implementacjach.

L I T E R A T U R A

[1] Anderson J.R., Bower G.H., Human Associative Memory, Holt, New York 1973.

[2] Brachman R.J. i inni, KRYPTON: A functional approach to knowledge representation, IEEE Computer, October 1983, 67-73.

[3] Brachman R.J., Schmolze J.G, An overview of the KL-ONE knowledge representation system, Congnitive sci., 9,2,

[4] Brady M., Artificial intelligence and robots, Art. Intell., 26, 1 (1985) , 79-121

[5] Buchanan B.G., Feigenbaum E.A., DENDRAL and Meta-DENDRAL: Their application dimension, Art. Intell., 11 (1978), 5-24.

[6] Burns N.A. i inni, The portable inference engine; fitting significant expertise into small systems, IBM System Journal, 25,2 (1986) , 236-243.

[7] Chang C.L., R.C.T. lee, Symbolic logic and Mechanical Theorem Proving, Academic Press, New York 1973.

[8] Charniak E.,A common representation for problem solving and language-comprehension information, AI, 16, 3 (1981) , 225-255.

[9] Clancy W.J., The epistemology of a rule - based expert system a framework for explanation AI, 20,3 May 83 , 215-251.

[10] Clark K.L., McCabe F.G., PROLOG: a language for implementing expert systems, Machine Intelligence, 10, J.E. Hayes i inni (eds.) , 455-475.

[11] Corlett R.A., Features of artificial intelligence languages and their environments, Software Engineering Journal, 1, 4 (1986), 159-164.

[12] Davies R., Shrobe H., Representing structure and behavior of digital hardware, IEEE Computer, October 1983, 75-82.

[13] Davies R., King J., An overview of production systems. Machine Intelligence 8, E.W.Elcock, D. Michie (eds.) , Ellis Horwood, New York 1977.

[14] Fagen J.M., VM: Representing time-dependent relations in a medical setting, Standford Univ., Stanford, June 1980.

[15] Feigenbaum E.A., Feldman J., Computers and Thought, McGraw-Hill, New York 1963.

[16] Feigenbaum E.A., Buchanan B.G., Lederberg, On generality and problem solving: A case study using the DENDRAL program, Machine Intelligence 6; D. Meltzer, D. Michie (eds.) , University Press, Edinburgh 1971.

[17] Fikes R., Kehler T., The role of frame based representation in reasoning, Comm. ACM, 28, 9 (1985) , 904-920.

[18] Forgy C.J., Mc Dermott J., OPS: A domain-idependent production system language, IJCAI-5, 933-939.

[19] Forgy C.L., OPS5 User's Manual, Carnegie Mellon Univeristy 1981.

[20] Forgy C.L., The OPS83 Report, Carnegie-Mellon University 1986.

[21] Funt B.V., Fraser S., Analogical modes of Reasoning and process modeling, IEEE Computer, October 1983, 99-104.

[22] Funt B.V., Problem solving with diagrammatic representations, AI, 13, 3 (1980) , 201-230.

[23] Georgeff M.P., Procedural control in production systems, AI, 18, 2, March 1982 , 175-201.

[24] Goldberg A., Robson D., Smalltalk-80: The language and its implementation, Reading, MA, Addison-Wesley 1983.

[25] Gottschalk G.R., Vandoom R.N, A rule-based system to diagnose malfuctioning computer peripherals, Hewlett-Packard Journal, November 1986, 48-53.

[26] Harman P., King D. Expert Systems - Artificial Intelligence in Business, J. Wiley, New York 1985.

[27] Hayes-Roth i inni (eds.) Buiding Expert Systems, Addison-Wesley, London 1983.

[28] Hayes P.J., In defense of logic. Proc. Fifth Int. Joint Conf. on AI, Cambridge, MA, August 1977 , 559-565.

[29] Havens W., Mackworth A., Reperesenting knowledge of visual world, IEEE Computer, October 1983, 90-96.

[30] Hendrix G.G., Encoding knowledge in partitioned networks, w: [54] .

[31] Kehler T.R., Clemenson G.D., An Application development system for expert systems, Syst. Softw., 3, 1 January 1984, 212-224.

[32] Kowalski R., Logic programming. Str. 133-145 w Information Processing 83, R.E.A. Mason (ed.), North-Holland 1983,

[33] Kowalski R., Logic for Problem Solving, North-Holland 1979.

[34] Langley P., Representational issues in learning systems, IEEE Computer, October 1983, 47-51.

[35] Leibnitz G.W., Nowe rozważania dotyczące rozumu ludzkiego, PWN, Warszawa 1955 (tłum. z francuskiego).

[36] Lindsay P.P., Norman D.A., Procesy przetwarzania informacji u człowieka, wprowadzenie do psychologii, PWN, Warszawa 1984 (tłum. z angielskiego).

[37] Loftus G.R., Loftus E.F., Human memory: The processing of information, New York 1976.

[38] Manuel T., LISP and PROLOG machines are proliferating, Electronics, November 1983, 132-137.

[39] Marks K.M., Goser K.E, An expert system shell for standardization of VLSI process data base and knowledge base, Microprocessing and Microprogramming, 21 (1987), 523-530.

[40] McCarthy J., History of LISP, SIGPLAN Notices, 13, 217-223.

[41] McDermott J., R1: A Rule-based configurer of computer systems: Art.Intell., 19, 1 (1982), 39-88.

[42] McDermott J., Forgy C., Production system conflict resolution strategies, w: [70], 171-179.

[43] Minsky M.,The Psychology of Computer Vision, P.H. Winston (ed.), McGraw-Hill, New York 1975.

[44] Mosley J.D., Expert systems as engineering tools.EDN, no. 13, 1986, 91-100.

[45] Moto-Oka, Fuchl K., The architectures in the fifth generation computers, zawarte w "Information Processing 83", R.E.A. Mason (ed.), North-Holland 1983.

[46] Mylopoulos J. i inni, Building Knowledge-based systems: the PSN experience, IEEE Computer, October 1983, 83-89.

[47] Newell A., Heuristic programming: ill-structured problems; Arnofsky J.S. (ed.), Progress in Operations Research, Wiley, New York 1969, 361-414.

[48] Newell A., Simon H.A., Human Problem Solving, Prentice-Hall, Inc., Englewood Cliffs 1972.

[49] Newell A., Simon H.A., Computer science as empirical inquiry: Symbols and search, Comm. ACM, 19, 3 (1976), 113-126.

[50] Newell A., Simon H.A., GPS, a program that simulates human thought, w: [15].

[51] Nilson N.J., Principles of Artificial Intelligence, Palo Alto 1980.

[52] Parker R., An expert for every office, Computer Design, Fall 1983, 37-46.

[53] Post E., Formal reductions of the general combinatorial problem, Am. Journal Math., 65 (1943), 197-268.

[54] Quillian M.R., Semantic Memory, Bolt Beranek Newman, Combridge, May 1966.

[55] Robinson J.A., Generalized resolution principle, Machine Intelligence, vol. 3, D, Michie (ed.) , American Elovier, New York 77-94.

[56] Rodzina M16-1, Ogólne Charakterystyki Techniczne, Redakcja 3-84, XXV posiedzenie SS-4. SM EMC.

[57] Schubert L.K., i inni, Determining type, part, color ane time relationships, IEEE Computer, October 1983, 53-60.

[58] Selfridge O.F., Pattern recognition and modern computers. Proc. of the 1955 Joint Computer Conference, 7 (1955) , 91-93.

[59] Shell B., Family of personal LISP machines speeds AI program development, Electronics, November 3 (1983), 153-156.

[60] Shortliffe E.H., Computer - based Medical Consultations: MYCINE, Am. Elsevier, New York 1976.

[61] Simon H.A., Siklassy L., (eds.) , Representation and Meaning: Experiments with Information Processing systems, Prentice-Hall, Inc., Englewood Cliffs 1972.

[62] Sleeman D., Brown J.S., Intelligent Tutoring Systems, Academic Press, London 1982.

[63] Sowa J.F., Conceptual Structures, Addison-Wesley, Reading 1984.

[64] Stefik J., Planning with constraints, Art.Intell., 16, 2 (1981) , 111-140.

[65] Stefik M. i inni, The organization of expert systems, a tutorial. AI, 18, 2, March 1982 135-173.

[66] Taylor J.H., Frederick D.K., An expert system architecture for computer - aided control engineering, Proc. IEEE, 72 , 12 (1984) , 1795-1805.

[67] Turing A.M., Computing machinery and intelligence. Mind, 59 (1950) , 433-450.

[68] Warman E.A., Computer Application in Production and Eugineering CAPE 83, North-Holland 1983.

[69] Watermann D.A., An introduction to production systems, AISB European Newsletter, Issue 25, 7-10.

[70] Watermann D.A., Hayes-Roth F., (eds.) , Pattern Directed Inference Systems, Academic Press, New York 1978.

[71] Webber B.L., Logic and natural language, IEEE Computer, October 1983, 43-46.

[72] Wiener N., Cybernetics. Sci. Americain, 179, 1948.

[73] Wilcox B., Reflections on building two go programs, SIGART Newsletters, October 1985 94 , 29-43.

[74] Winograd T., Understanding Natural Language, Academic Press 1972.

[75] Woods W.A., What's important about Knowledge representation? Computer (IEEE computer Society) , October 1983, 22-27.

[76] Wrzeszcz Z.,A model of complex job control computer systems, Prace Naukowo-Badawcze Instytutu Maszyn Matematycznych, XXVIII, I (1986), 3-29.

[77] Zadeh L.A., Commonsense knowledge representation based on fuzzy logic, IEEE Computer, October 1983 , 61-65.

5. LISP

Niniejszy rozdział poświęcamy charakterystyce języka LISP oraz bardziej szczegółowemu przedstawieniu pewnych jego uzupełnień niezbędnych do realizacji systemów eksperckich. Jak wyjaśniono w rozdziale 4., język LISP w swej podstawowej formie nie ma zdolności dopasowania do wzorca, zdolność tę wprowadzamy w postaci funkcji MATCH. Ponadto podajemy zapisy bazy danych i reguł oraz wprowadzamy funkcję FORWARD-CHAIN niezbędną do sterowania procesem wnioskowania. Technikę pisania programów eksperckich za pomocą języka LISP ilustrujemy przykładami używając języka GCLISP będącego realizacją COMMON LISP na komputery PC-XT/AT.

5.1 Omówienie własności języka LISP

Autorem języka LISP (list processing language) jest John Mc Carthy, który na podstawie wcześniejszych prac A. Newella, C. Shawa i H. Simona podjął około 1960 r. w MIT udaną próbę zbudowania funkcyjnego języka programowania. LISP uzyskał pewną popularność w środowiskach rozwijających sztuczną inteligencję oraz w środowiskach naukowców pracujących nad semantyką programowania. Praktyczne jednak zastosowanie mógł on wtedy znaleźć tylko przy wykorzystaniu największych istniejących wówczas komputerów i nie mógł konkurować w dziedzinie efektywności ani algorytmów numerycznych z takimi językami, jak FORTRAN i ALGOL. Dążenie do poszerzenia bazy użytkowników powodowało, że ulegał on ewolucji w kierunku imperatywności, rozbudowywano jego środowisko (MAC-LISP i INTERLISP), tworzono liczne wersje adaptujące praktycznie wszystkie koncepcje języków programowania (LOGLISP - koncepcje z PROLOGU, XLISP z języka SMALLTALK). Wreszcze powstał jego standard COMMON LISP i jego wersja na komputery personalne GCLISP. Równocześnie olbrzymi postęp wydajności środków obliczeniowych został dodatkowo wzmocniony konstrukcją komputerów o specjalnej architekturze do efektywnych obliczeń w LISP-ie (SYMBOLIC 3600, LMI, Explorer TI i inne). Obecnie ze względu na bogactwo środowiska i wydajność specjalnego sprzętu LISP jest najbardziej popularnym językiem w USA w dziedzinie sztucznej inteligencji. Projektuje się mikroprocesor o wysokim stopniu integracji dla LISP-u [1], buduje się języki lispowe do równoległego przetwarzania (MULTI-LISP). Pewne problemy wynikają z braku wysoko wykwalifikowanych programistów, wysokiej ceny sprzętu specjalizowanego, trudności semantycznych wynikających z imperatywnych rozszerzeń oraz rozwijającej się konkurencji PROLOGU. Podstawy teoretyczne LISP-owi daje język symbolicznych obliczeń - Lambda rachunek Churcha [2], na którym wzorował się John McCarthy. W LISP-ie podtrzymano zasadę beztypowości w znaczeniu braku syntaktycznego rozróżnienia między danymi a funkcjami, która również występuje w językach niskiego poziomu maszyn von Neumana (słowo i instrukcja). Jednocześnie zachowano pełną funkcjonalność języka wysokiego poziomu przejawiającą się w strukturalnej jasności i semantycznej ekspresywności. Semantykę LISP-u opisano w samym języku LISP w postaci jego interpretera. Jest on punktem wyjścia do różnych implementacji na tzw. maszynach SECD [2] oraz na układach scalonych.

5.1.1 Składnia

Składnia LISP-u ma wyjątkowo prostą postać

<atom> ::= <literał> | <numerał>
<s - wyr> ::= <atom> | (<s-wyr> . <s-wyr>)

wyróżnia się atomy T jako reprezentacje prawdy i NIL (identyczny z( )) jako reprezentację fałszu oraz listy pustej. Składnia symbolicznych wyrażeń (s-wyrażeń), w większości realizacji LISP-u jest reprezentowana w pamięci komputera przez adresy do opisu atomów i adresy do par adresów podwyrażeń. W nowszych realizacjach ta informacja adresowa jest umiejętnie cechowana (tagowana).
Składnia ta, chociaż teoretycznie wystarczająca, praktycznie jest uciążliwa przez ekspansję nawiasów i kropek. Wprowadza się więc zapis skrótowy

Np. (A B C . D) $\underset{\text{def}}{=}$ (A . (B . (C . D)))
oraz (A B C D) $\underset{\text{def}}{=}$ (A . (B . (C . (D . NIL))))

Wyrażenia postaci (A B C D) noszą nazwę list, przy czym warto zauważyć, że A,B,C,D mogą być dowolnymi s-wyrażeniami, np. listami. LISP, który powstał jako język programowania przeznaczony do przetwarzania listowych struktur danych, wyposażony jest w duży repertuar funkcji służących do przetwarzania list. Istnieją więc funkcje: pozwalające na dostęp do poszczególnych elementów listy, służące do wydzielenia określonych podlist, tworzące listy z żądanych elementów, wyliczające liczbę elementów listy itp. Wśród predykatów LISP-u istotną grupę stanowią te, które służą do rozpoznawania typu różnych s-wyrażeń języka. Istnieje więc również predykat pozwalający na identyfikację wyrażeń typu lista. Wśród elementów typu lista są programy definiujące funkcje. Definicje te pisane są w formalizmie rachunku lambda przy użyciu funkcji pierwotnych lub wcześniej zdefiniowanych.
Należy w tym miejscu podkreślić fakt, że programy wprowadzania informacji dopuszczają jedynie struktury drzew binarnych ze zlepionymi liśćmi (tzn. złożone z par kropkowanych i atomów), ale w trakcie obliczeń mogą powstać dowolne grafy binarne, które są traktowane przez interpreter jako programy. Mamy tu pełną analogię z von-neumanowskim komputerem, w którym słowo jest instrukcją, jeżeli znajduje się w rejestrze instrukcji i jest liczbą, jeżeli znajduje się w rejestrze akumulatora.

Sposób użycia składni LISP-u w pisaniu programów ilustrują następujące przykłady:

1. Aplikacje funkcji "f" do argumentów "x" i "a"; f(x, a) zapisuje się w tzw. Cambridge Polish notacji: (f x a), co jest listą LISP-u złożoną z trzech atomów f, x, a.
2. LAMBDA abstrakcją po zmiennej "x" formy f(x, a) (tzn. uczynienie ze wzoru f(x, a) definicji funkcji $x \longmapsto f(x,a)$: $\lambda x.f(x, y)$) zapisuje się w listowej notacji LISP-u (LAMBDA (x)(f x a)), w której LAMBDA jest wyróżnionym atomem, a nawiasy oznaczają odpowiednie listy.
3. Wprowadzenie nowej funkcji do środowiska zapisuje się w postaci (DEFINE <Literał><s-wyr>), w którym literał jest identyfikatorem funkcji, a symboliczne wyrażenie jest definicją w postaci LAMBDA abstrakcji z ewentualnym użyciem rekursywnym tego identyfikatora.

Bardzo ważnym rodzajem s-wyrażeń LISP-u są tzw. wyrażenia warunkowe, pozwalające tworzyć programy z rozgałęzieniami. Do budowy tych wyrażeń służy specjalna funkcja. Wyrażenia warunkowe stanowią uogólnienie konstrukcji budowanych za pomocą operatora IF w językach ALGOL czy FORTRAN. Należy zaznaczyć, że wyrażenia warunkowe oraz definicje rekurencyjne stanowią podstawowe narzędzia do budowania programów w LISP-ie. W LISP-ie istnieją też specjalne funkcje, których zadaniem jest wprowadzenie i wyprowadzenie danych i to na trzech różnych poziomach: pojedyńczych znaków, słów i wyrażeń symbolicznych. Szczególnie ważne jest, że w LISP-ie używa się danych do przedstawienia programów, które można dzięki temu przetwarzać tak jak dane. Tak więc możemy zdefiniować program wykonujący inne programy, zwany interpretatorem, a przez to określić znaczenie programów LISP-u w samym LISP-ie.

5.1.2 Semantyka

Rozróżnia się dwa rodzaje opisu semantyki języka programowania: denotacyjny i operacyjny. W pierwszym używa się języka logiki do opisania modeli matematycznych, w których interpretuje się operacje języka programowania i bada ich własności. W drugim używa się języka programowania do opisania algorytmu interpretacji na abstrakcyjnym automacie realizującym język opisu. Następnie transformuje się, z zachowaniem treści, algorytm interpretacji i automat do realizacji efektywnego programu na fizycznym istniejącym sprzęcie. Jest to złożona praca konstrukcyjna z wieloma trudnymi decyzjami do podjęcia. Podejście denotacyjne jest równie niełatwym procesem logicznej analizy, ale prowadzi on do rozwiązań trudnych do osiągnięcia na drodze operacyjnej. Opiszemy teraz w ujęciu operacyjnym semantykę LISP-u z tym LISP-em jako językiem opisu i człowiekiem jako realizującym go automatem abstrakcyjnym. Będzie to mała i prosta, ale nowoczesna wersja LISP-u zwana SCHEME, przeznaczona do realizacji w układzie scalonym.
Funkcje pierwotne:
(x - jest zmienną, wx - jest wartością zmiennej x)

(ATOM x) przyjmuje wartość T gdy wx jest atomem,
NIL w przeciwnym przypadku.

(NUMBERP x) przyjmuje wartość T, gdy wx jest numerałem,
NIL w przeciwnym przypadku.

(EQ x y) przyjmuje wartość T, gdy wx jest identyczne z wy
NIL w przeciwnym przypadku.

(NULL x) przyjmuje wartość T, gdy wx=NIL,
NIL w przeciwnym przypadku.

(IF x y z) przyjmuje wartość wy, gdy wx ≠ NIL,
wz w przeciwnym przypadku.

(CAR x) przyjmuje wartość pierwszego wyrażenia pary kropkowanej wx,
nieokreślona, jeśli wx nie jest parą kropkowaną.

(CDR x) przyjmuje wartość drugiego wyrażenia pary kropkowanej wx,
nieokreślona, jeśli wx nie jest parą kropkowaną.

(CADR x) równa jest z definicji (CDR (CAR x)) (analogicznie dla innych skrótów)
(CONS x y)równa jest z definicji (wx . wy).
(LIST x1 x2...xn)równa jest z definicji (wx1 wx2 ... wxn).

Opisany w tych prymitywach interpreter traktuje zmienne nie związane operatorem Lambda w definicji funkcji w sposób nowoczesny, biorąc ich wartości ze stanu otoczenia w momencie definicji, a nie w momencie aplikacji, rekursywne zaś wywołanie funkcji w ostatnim obliczanym wyrażeniu w jej definicji traktuje jako pętlę.

Interpretacja programów (tzn. symbolicznych wyrażeń) polega na rekursywnym wywoływaniu funkcji EVAL, APPLY, EVLIS i innych. Zasadniczą rolę odgrywają EVAL i APPLY.

```
(DEFINE EVAL
   (LAMBDA (EXP EIN)
           (IF (ATOM EXP)
               (IF (NUMBERP EXP) EXP
                   (VALUE EXP ENV))
               (IF (EQ (CAR EXP)'QUOTE)
                   (CADR EXP)
                   (IF (EQ (CAR EXP) 'LAMBDA)
                       (LIST '&PROCEDURE
                       (CADR EXP)
                       (CADDR EXP)
                       ENV)
                       (IF (EQ (CAR EXP) 'IF)
                           (IF (EVAL (CADR EXP) ENV)
                               (EVAL (CADDR EXP) ENV)
                               (EVAL (CADDDR EXP) ENV))
                           (APPLY (EVAL(CAR EXP) ENV)
                            (EVLIS (CDR EXP)ENV)))))))))


(DEFINE APPLY
   (LAMBDA (FUN ARGS)
           (IF (PRIMOP FUN)
               (PRIMOP-APPLY FUN ARGS)
               (IF (EQ (CAR FUN) '&PROCEDURE)
                   (EVAL (CADDR FUN)
                         (BIND (CADR FUN)
                               ARGS
                         (CADDDR FUN)))
                   (ERROR)))))

(DEFINE EVLIS
   (LAMBDA (ARGLIST ENV)
           (IF (NULL ARGLIST) '()
               (CONS (EVAL (CAR ARGLIST) ENV)
                     (EVLIS (CDR ARGLIST)
                            ENV)))))
```

EVAL klasyfikuje wyrażenie i kieruje jego obliczeniem w zadanym otoczeniu (stan zmiennych):

- gdy wyrażenie jest numerałem, jego wartością jest ten numerał,
- gdy warażenie jest innym atomem, jego wartością jest wartość zdefiniowana w otoczeniu,
- gdy pierwszym elementem formy jest QUOTE, wtedy wartością jest drugi element formy,
- gdy pierwszym elementem jest LAMBDA, wtedy EVAL tworzy listę składającą się z zastrzeżonego słowa &PROCEDURE, listy zmiennych funkcji, definicji funkcji, aktualnego otoczenia (informacje te będą wykorzystane w APPLY),

- gdy pierwszym elementem jest IF, wtedy oblicza drugi element i w zależności od tego, czy jest NIL - przechodzi do obliczenia trzeciego lub czwartego elementu,
- w pozostałych przypadkach stosuje funkcję APPLY do rezultatu ewaluacji pierwszego elementu (funkcja) i listy wartości argumentów uzyskanych za pomocą funkcji EVLIS, w której wykorzystywany jest rekurencyjnie EVAL.

Funkcja APPLY rozróżnia dwa rodzaje funkcji: prymitywne i zdefiniowane przez użytkownika. Wartości funkcji prymitywnych oblicza się bezpośrednio. Dla funkcji użytkownika, które wyróżniają się wystąpieniem na pierwszej pozycji &PROCEDURE, APPLY rozszerza otoczenie z momentu definicji o argumenty funkcji oraz ich wartości (patrz funkcja BIND) i wywołuje EVAL dla definicji funkcji i tego otoczenia.

Otoczenie jest listą złożoną z par List. Elementy pierwszej listy są identyfikatorami, których wartości znajdują się na drugiej liście. Kolejne pary list realizują leksykalną, blokową strukturę. Do operacji na otoczeniu używa się następujących funkcji:

```
(DEFINE BIND
   (LAMBDA (VARS ARGS  ENV)
           (IF(= LENGTH VARS)
              (LENGTH ARGS))
              (CONS (CONS  VARS ARGS)  ENV)
              (ERROR))))

(DEFINE VALUE
   (LAMBDA (NAME  ENV) (VALUE1  NAME
   (LOOKUP NAME ENV))))
(DEFINE VALUE1
   (LAMBDA (NAME  SLOT)
           (IF (EQ  SLOT '&UNBOUND)  (ERROR)
           (CAR SLOT))))
(DEFINE LOOKUP
   (LAMBDA (NAME ENV)
           (IF (NULL ENV)'&UNBOUND
              (LOOKUPI
            NAME  (CAAR ENV)
           (CDAR ENV) ENV))))
(DEFINE LOOKUPI
   (LAMBDA (NAME VARS VALS ENV)
           (IF (NULL  VARS)
              (LOOKUP NAME (CDR  ENV))
              (IF (EQ NAME (CAR  VARS)) VALS
                  (LOOKUPI
               NAME (CDR  VARS)
              (CDR VALS)  ENV)))))
```

BIND rozszerza otoczenie o parę list: listę nazw i listę wartości sygnalizując błąd, gdy listy te różnią się długością. VALUE jest połączeniem z LOOKUP, w którym wydziela się przypadki niezdefiniowania wartości. LOOKUP dla zmiennej i otoczenia dostarcza tę część listy wartości, której CAR jest żądaną wartością zmiennej. W [4] opisano ciąg transformacji tego interpretera do maszyny SECD realizowanej w układzie scalonym.

5.2 Dopasowanie do wzorca

Fundamentalnym pojęciem przy konstruowaniu programów z dziedziny sztucznej inteligencji, a zwłaszcza dotyczących systemów eksperckich opartych na regułach, jest pojęcie dopasowania do wzorca (ang. pattern-matching). Sam LISP nie ma wbudowanych narzędzi służących dopasowaniu, jednak dobrze się je implementuje w tym języku. Poniżej przedstawione zostaną dwa sposoby takiej implementacji: pierwszy - zawierający większość najistotniejszych mechanizmów występujących przy problemie dopasowania do wzorca, drugi - stanowiący modyfikację pierwszego pod kątem pisania programów w języku polskim.

Dopasowanie do wzorca jest procesem porównywania między sobą wyrażeń symbolicznych. Wyrażenia te, noszące odpowiednie nazwy wzorzec i dana, są listami. Dana służy bardzo często do reprezentowania stwierdzeń (ang. assertions) dotyczących badanego modelu. Wzorzec natomiast służy do identyfikacji żądanej danej spośród być może bardzo wielu danych opisujących rozważany model.

Wzorzec może zawierać pewne specjalne wyrażenia np.: ?, + które wykorzystywane są przy identyfikacji danej. Jeśli przykładowo mamy zbiór danych postaci:

```
(A1   JEST KWADRATEM )
(A2   JEST PODOBNA DO A1)
(A2   MA      4   KĄTY RÓWNE )
```

Wtedy używając wzorca

```
(A1   JEST   ? )
```

identyfikujemy pierwszą z tych danych.
Natomiast używając wzorca

```
(A2   + )
```

identyfikujemy dwie pozostałe dane.

Jeśli wzorzec nie zawiera żadnych specjalnych wyrażeń, jest on dopasowany do danej wtedy tylko, jeśli na odpowiadających sobie pozycjach występują te same wyrażenia, czyli wtedy, gdy wzorzec i dana są identyczne.
W naszym przykładzie wzorzec

```
(A2   JEST PODOBNA DO A1)
```

identyfikuje drugą z rozważanych danych.

Szczegółowy opis procesu dopasowania do wzorca przedstawimy poniżej, omawiając definiowaną w LISP-ie funkcję MATCH.

5.2.1 Funkcja MATCH - wersja 1.

Funkcja MATCH, realizująca w LISP-ie proces dopasowania do wzorca, jest funkcją trójargumentową. Jej argumentami są: wzorzec, dana i lista asocjacji.

Lista asocjacji jest rodzajem kontekstu, w którym dokonuje się porównania wzorca i danej. Postać listy asocjacji jest następująca:

```
((V1   E1) (V2   E2) ...(VN   EN))
```

gdzie E1...EN mogą być dowolnymi wyrażeniami LISP-u, natomiast V1...VN są tzw.zmiennymi wzorcowymi (ang. pattern variables) .

Funkcja MATCH jest zdefiniowana w sposób rekurencyjny, przy czym kolejno analizowane i porównywane są elementy list wzorca i danej. Jeśli element wzorca nie jest wyrażeniem specjalnym, to zostaje uznany za dopasowany do odpowiadającego mu elementu danej tylko wtedy, gdy są to elementy identyczne. W przypadku, gdy elementem wzorca jest wyrażenie specjalne, działanie funkcji jest następujące:

- jeśli wyrażenie specjalne ma postać: ?, wtedy odpowiadający mu jeden element danej zostaje uznany za dopasowany do tego elementu wzorca;
- jeśli wyrażenie specjalne ma postać: +, wtedy odpowiadający mu ciąg kolejnych elementów danej może zostać uznany za dopasowany do tego elementu wzorca;
- jeśli wyrażenie specjalne ma postać:(>V), gdzie symbol > nosi nazwę wskaźnika wzorca (ang. pattern indicator), a symbol V nosi nazwę zmiennej wzorcowej, wtedy odpowiadający mu jeden element danej D zostaje uznany za dopasowany do tego elementu wzorca oraz lista asocjacji zostaje rozszerzona o asocjację (V D);
- jeśli wyrażenie specjalne ma postać:(<V), gdzie < jest wskaźnikiem wzorca, a V zmienną wzorcową, wtedy na liście asocjacji zostanie odnaleziona wartość zmiennej V i podstawiona do wzorca w miejsce rozważanego wyrażenia specjalnego;
- jeśli wyrażenie specjalne ma postać: (+V), gdzie + jest wskaźnikiem wzorca, a V zmienną wzorcową, wtedy odpowiadający mu ciąg kolejnych elementów danej może zostać uznany za dopasowany do tego elementu wzorca i ponadto na liście asocjacji odnaleziona zostanie

asocjacja zmiennej V i odpowiednio rozszerzona o element lub ciąg elementów danej. Jeśli nie zostanie odnaleziona asocjacja zmiennej V, wtedy zostanie ona utworzona i dołączona do listy asocjacji;

- jeśli wyrażenie ma postać: (RESTRICT ? PRED1...PREDN), gdzie RESTRICT jest wskaźnikiem wzorca, ? jest wskaźnikiem ograniczenia (ang. restriction indicator), a PRED1...PREDN są predykatami ograniczającymi (ang. restriction predicates), wtedy odpowiadający mu jeden element danej zostaje uznany za dopasowany do tego elementu wzorca o ile ten element danej spełnia wszystkie predykaty PRED1...PREDN;
- jeśli wyrażenie specjalne ma postać: (RESTRICT(>V) PRED1...PREDN), gdzie RESTRICT jest wskaźnikiem wzorca,(>V) jest wskaźnikiem ograniczenia, a PRED1...PREDN są predykatami ograniczającymi, wtedy odpowiadający mu jeden element danej D zostaje uznany za dopasowany do tego elementu wzorca o ile ten element danej spełnia wszystkie predykaty PRED1...PREDN; ponadto lista asocjacji zostanie rozszerzona o asocjację (V D).

Wynik działania funkcji MATCH występuje w trzech rodzajach:

- NIL - jeśli proces dopasowania danej do wzorca nie zakończył się powodzeniem,
- T - jeśli proces dopasowania danej do wzorca zakończył się powodzeniem i jednocześnie lista asocjacji jest listą pustą,
- Lista asocjacji - jeśli proces dopasowania danej do wzorca zakończył się powodzeniem i jednocześnie lista asocjacji jest niepusta.

Pełna definicja funkcji MATCH (wersja 1) w LISP-ie została przedstawiona w rozdziale 5.6.2.

Przedstawimy teraz kilka wywołań funkcji MATCH, w których uwzględniono różne postacie wzorca.

```
(MATCH  `(A1 JEST ?)  `(A1 JEST  KWADRATEM)  NIL)
T

(MATCH  `(+  JEST +)  `(A1 JEST  KWADRATEM)  NIL)
T

(MATCH `((>FIGURA)  JEST ?)  `(A1 JEST  KWADRATEM)  NIL)
((FIGURA A1))

(MATCH `((<FIGURA)  JEST ?)  (A1 JEST  KWADRATEM)
       `((FIGURA A1)))
((FIGURA A1))

(MATCH `(A2 MA  (+ OPIS))       `(A2 MA 4 KĄTY RÓWNE)
      `((OPIS (4 BOKI RÓWNE))))
((OPIS  (4 KĄTY RÓWNE 4 BOKI RÓWNE)))

(MATCH `(A2 MA    (RESTRICT (>KĄTY)  NUMBERP)  KĄTY  RÓWNE)
       `(A2 MA     4 KĄTY RÓWNE)  NIL)
((KĄTY 4))
```

5.2.2. Funkcja MATCH - wersja 2.

Przedstawimy teraz drugą wersję funkcji MATCH, która stanowi modyfikację wersji pierwszej pod kątem pisania programów w języku polskim.

Dla programów z dziedziny sztucznej inteligencji problemy dialogu z użytkownikiem oraz formułowania stwierdzeń w bazie danych stanowią o komunikatywności, a co za tym idzie przydatności tych programów. Dlatego też realizacja programu w języku polskim wydaje się być ważna i potrzebna.

Wykonano dwie modyfikacje:

- po pierwsze - wyrażenie specjalne wzorca + będzie miało charakter bardziej uniwersalny. Odpowiednia zmiana w definicji funkcji MATCH jest następująca: jeśli wyrażenie specjalne jest postaci: +, wtedy odpowiadający mu ciąg być może pusty kolejnych elementów danej może zostać uznany za dopasowany do tego elementu wzorca;
- po drugie - jeśli element wzorca nie jest wyrażeniem specjalnym, wtedy zostanie uznany za dopasowany do odpowiadającego mu elementu danej tylko wtedy, gdy element wzorca stanowi początkową część (w sensie ciągu znaków) elementu danej.

Obie powyższe zmiany wprowadzone zostały w celu umożliwienia identyfikowania żądanych danych za pomocą słów kluczowych, a w szczególności za pomocą znaczących części słów kluczowych. Aby zaprezentować nowe możliwości identyfikacji, na które pozwala druga wersja funkcji MATCH, rozważmy poniższy przykład.

Niech zbiór danych ma następującą postać:

(CZWOROKĄT MA 1 PARĘ RÓWNYCH BOKÓW)
(CZWOROKĄT MA 2 BOKI RÓWNOLEGŁE)
(CZWOROKĄT MA DWIE PARY RÓWNYCH KĄTÓW)
(CZWOROKĄT MA JEDNĄ OŚ SYMETRII)
(CZWOROKĄT JEST TRAPEZEM)

Jeśli użytkownik chciałby z tego zbioru danych opisujących w języku naturalnym jakiś czworokąt wybrać dane dotyczące np. boków tego czworokąta, to używając 1 wersji funkcji MATCH miałby z tym sporo kłopotów. Wynika to z faktu różnorodnych możliwości deklinacyjnych rzeczownika BOK oraz różnych możliwości szyku zdania w języku polskim. Stosując natomiast 2 wersję funkcji MATCH możemy to zrobić używając wzorca

(+ BOK +)

który identyfikuje wszystkie dane zawierające na jakiejkolwiek pozycji słowo, którego częścią znaczącą jest BOK.

Pełna definicja funkcji MATCH (wersja 2.) została przedstawiona w rozdziale 5.6.3.

Jakkolwiek głównym powodem, dla którego analizujemy pojęcie dopasowania do wzorca są systemy eksperckie, to jednak dla wielu programów z dziedziny sztucznej inteligencji dopasowanie do wzorca jest pojęciem kluczowym. Tak jest również dla programów, których celem jest podtrzymywanie dialogu z użytkownikiem. Przykład takiego programu o nazwie "PRZYJACIEL" został przedstawiony w rozdziale 5.6.4. Wykorzystuje on oczywiście funkcję MATCH (wersja 2). Jest on wzorowany na programie psychiatrycznym DOCTOR, którego uproszczona wersja została przedstawiona w [3].

5.3 Implementacja reguł produkcji

Bardzo często wiedza eksperoka może być reprezentowana jako zbiór reguł następującej postaci:

Jeśli <przesłanka 1 jest prawdziwa> i
⋮
<przesłanka n-ta jest prawdziwa>
to <wniosek 1 jest prawdziwy> i
⋮
<wniosek k-ty jest prawdziwy> n, k ≥ 1

Reguły tej postaci zwane są często regułami produkcji (ang. production rules - rozdział 3.2.2.3)

Dwoma istotnymi składnikami systemu eksperckiego z regułami produkcji są: baza danych zawierająca zbiór stwierdzeń (ang. assertions) będących faktami dotyczącymi danego problemu oraz zestaw reguł reprezentujący wiedzę ekspercką danej dziedziny. W LISP-ie stwierdzenia mogą być reprezentowane jako listy atomów. Wszystkie takie stwierdzenia zebrane będą w listę o nazwie FAKTY.

Przykładowa lista FAKTY może być wprowadzona do bazy danych w sposób następujący:

```
(SETQ  FAKTY `(
      (CZWOROKĄT  MA 4 OSIE SYMETRII)
      (CZWOROKĄT  MA WSZYSTKIE  KĄTY RÓWNE)
      (CZWOROKĄT  MA WSZYSTKIE BOKI RÓWNE)
             ))
```

Istnieje wiele sposobów reprezentowania reguł. Przyjmiemy, że reguła będzie reprezentowana jako lista postaci:

```
(REGUŁA  <nazwa>
   (JEŚLI  <przesłanka  1>
           :
           <przesłanka  n>)
   (TO     <wniosek  1>
           :
           <wniosek  k>))          n,k ⩾ 1
```

Zakładamy, że wszystkie reguły zebrane będą w listę o nazwie REGUŁY.

Przykładowa lista REGUŁY może zostać wprowadzona w sposób następujący:

```
(SETQ REGUŁY `(
   (REGUŁA  NUMER1
      (JEŚLI((>FIGURA)   JEST KWADRATEM))
      (TO    ((<FIGURA   JEST ROMBEM)))
   (REGUŁA  NUMER2
      (JEŚLI ((>FIGURA)  MA WSZYSTKIE KĄTY  RÓWNE))
      (TO    ((<FIGURA)  JEST PROSTOKĄTEM)))
   (REGUŁA  NUMER 3
      (JEŚLI ((>FIGURA)  MA  4 OSIE SYMETRII))
      (TO    ((<FIGURA)  JEST  KWADRATEM)))
            ))
```

5.4 Sterowanie procesem wnioskowania

Przyjęty tu sposób działania systemu opiera się na zasadzie, "wnioskowania w przód" i będzie zrealizowany w LISP-ie poprzez funkcję FORWARD-CHAIN. Aby opisać działanie tej funkcji, zacznijmy od tego, jak stosuje się pojedynczą regułę.

Próba zastosowania pojedynczej reguły polega na dopasowaniu (funkcja MATCH) wszystkich przesłanek tej reguły do stwierdzeń zawartych na liście FAKTY. Jeśli wszystkie przesłanki zostaną dopasowane, wtedy za pomocą wszystkich wniosków tej reguły sformułowane zostaną wstępnie nowe stwierdzenia. Teraz nastąpi umieszczenie na liście FAKTY tych wstępnie sformułowanych stwierdzeń, których na liście FAKTY jeszcze nie ma. Będziemy mówili, że zastosowanie pojedynczej reguły zakończyło się sukcesem, jeżeli co najmniej jedno z wstępnie sformułowanych stwierdzeń zostanie umieszczone na liście FAKTY.

Zauważmy również, że brak sukcesu przy zastosowaniu reguły może być spowodowany dwiema przyczynami. Pierwsza - to fakt, że nie udało się dopasować przesłanek tej reguły do aktualnie istniejących stwierdzeń. Druga - to fakt, że jeśli nawet przesłanki dopasowano do stwierdzeń, to żadne z wstępnie sformułowanych stwierdzeń nie jest nowe.

Możemy teraz omówić działanie całego systemu (funkoja FORWARD-CHAIN) . Działanie to możemy podzielić na cykle. Pojedynczy cykl polega na tym, że system próbuje zastosować wszystkie reguły z listy REGUŁY w kolejności ich występowania na tej liście. Przy czym zuważmy, że jeśli próba zastosowania jakiejś reguły zakończyła się sukcesem wtedy rozszerzona została lista stwierdzeń FAKTY i próba zastosowania następnej reguły przebiega na podstawie nowej rozszerzonej listy.

Jeżeli w czasie pojedynczego cyklu próba zastosowania co najmniej jednej z reguł zakończyła się sukcesem, wtedy zainicjowany zostanie następny cykl. Tak więc zakończenie działania systemu nastąpi jedynie po tym cyklu, podczas którego żadna próba zastosowania reguły nie zakończyła się sukcesem.

Dla przykładu przeanalizujemy teraz działanie systemu dla wprowadzonej w poprzednim rozdziale 5.3 listy FAKTY oraz listy REGUŁY.

Cykl pierwszy

Przesłanki reguły nr 1 nie można dopasować do żadnego stwierdzenia z listy FAKTY, a więc reguły nr 1 nie można zastosować.

Przesłanka reguły nr 2 zostanie dopasowana do drugiego stwierdzenia z listy FAKTY. Wstępnie zostanie sformułowane stwierdzenie (CZWOROKĄT JEST PROSTOKĄTEM) , które umieszczone będzie na liście FAKTY.

Lista FAKTY ma teraz postać:

```
((CZWOROKĄT  JEST  PROSTOKĄTEM)
 (CZWOROKĄT  MA 4  OSIE SYMETRII)
 (CZWOROKĄT  MA WSZYSTKIE KĄTY RÓWNE)
 (CZWOROKĄT  MA WSZYSTKIE BOKI RÓWNE))
```

Przesłanka reguły nr 3 zostanie dopasowana do drugiego stwierdzenia z listy FAKTY. Wstępnie zostanie sformułowane stwierdzenie (CZWOROKĄT JEST KWADRATEM) , które umieszczone będzie na liście FAKTY.

Lista FAKTY ma teraz postać:

```
((CZWOROKĄT JEST KWADRATEM)
 (CZWOROKĄT JEST PROSTOKĄTEM)
 (CZWOROKĄT MA  4  OSIE  SYMETRII)
 (CZWOROKĄT  MA WSZYSTKIE KĄTY RÓWNE
 (CZWOROKĄT MA WSZYSTKIE BOKI RÓWNE))
```

Ponieważ w cyklu pierwszym otrzymaliśmy dwa sukcesy (dwukrotnie zastosowanie reguły zakończyło się sukcesem) , rozpocznie się cykl drugi.

Cykl drugi

Przesłanka reguły nr 1 zostanie dopasowana do pierwszego stwierdzenia z listy FAKTY. Wstępnie zostanie sformułowane stwierdzenie (CZWOROKĄT JEST ROMBEM) , które umieszczone będzie na liście FAKTY.

Lista FAKTY ma teraz postać:

```
((CZWOROKĄT JEST  ROMBEM)
 (CZWOROKĄT JEST  KWADRATEM)
 (CZWOROKĄT JEST  PROSTOKĄTEM)
 (CZWOROKĄT MA  4 OSIE SYMETRII)
 (CZWOROKĄT MA WSZYSTKIE  KĄTY RÓWNE)
 (CZWOROKĄT MA WSZYSTKIE BOKI RÓWNE))
```

Przesłanka reguły nr 2 zostanie dopasowana do piątego stwierdzenia z listy FAKTY. Wstępnie zostanie sformułowane stwierdzenie (CZWOROKĄT JEST PROSTOKĄTEM) , ale ponieważ istnieje

już ono na liście FAKTY, więc reguły nr 2 nie można zastosować.

Przesłanka reguły nr 3 zostanie dopasowana do czwartego stwierdzenia z listy FAKTY. Wstępnie zostanie sformułowane stwierdzenie (CZWOROKĄT JEST KWADRATEM), ale ponieważ istnieje już ono na liście FAKTY, więc reguły nr 3 nie można zastosować.

Ponieważ w cyklu drugim otrzymaliśmy jeden sukces, rozpocznie się cykl trzeci.

Cykl trzeci

Przesłanki wszystkich reguł zostaną dopasowane do odpowiednich stwierdzeń, ponieważ jednak wstępnie sformułowane stwierdzenia są już na liście FAKTY, to żadnej z reguł nie można zastosować.

Ponieważ w cyklu trzecim nie otrzymaliśmy żadnego sukcesu, następuje zakończenie programu eksperckiego.

Ostateczna lista FAKTY ma postać:

```
((CZWOROKĄT JEST ROMBEM)
 (CZWOROKĄT JEST KWADRATEM)
 (CZWOROKĄT JEST PROSTOKĄTEM)
 (CZWOROKĄT MA 4 OSIE SYMETRII)
 (CZWOROKĄT MA WSZYSTKIE KĄTY RÓWNE)
 (CZWOROKĄT MA WSZYSTKIE BOKI RÓWNE))
```

Pełna definicja funkcji FORWARD-CHAIN została przedstawiona w rozdziale 5.6.5.

Na koniec zauważmy, że zbiór stwierdzeń i zestaw reguł choć są oczywiście istotnymi składnikami systemu eksperckiego nie stanowią jeszcze kompletnego systemu eksperckiego (patrz rozdział 3.2 dotyczący funkcji eksperta).

Budujący system eksperckl należałoby więc uwzględnić przynajmniej następujące elementy

Możliwość skutecznego dialogu użytkownika z systemem (np. w języku naturalnym),

Możliwość analizy procesu dedukcyjnego systemu, (np. poprzez umożliwienie użytkownikowi zadania pytań typu: Jak wywnioskowane zostało dane stwierdzenie ? lub Do jakich wniosków użyte zostało dane stwierdzenie ? patrz [3] funkcje HOW, WHY).

Możliwość rozbudowy systemu eksperckiego, aby w miarę przybywania nowych informacji dotyczących danej dziedziny wiedzy, można było uwzględnić te informacje w systemie, rozszerzając odpowiednio zestaw reguł lub zbiór stwierdzeń czy też modyfikując system dialogowy.

Tak więc budowa w LISP-ie praktycznie przydatnego systemu eksperckiego wymaga wielu skomplikowanych konstrukcji programowych pod kątem danej dziedziny wiedzy, symulujących kontakt użytkownika z ekspertem tej dziedziny.

5.5. System CZWOROKĄTY - przykład formułowania

Opiszemy teraz przykładowy system ekspercki o nazwie CZWOROKĄTY, który stanowi próbę ilustracji tego, jak można w LISP-ie implementować systemy z regułami produkcji.

System przeznaczony jest dla użytkownika, który chciałby poznać na poziomie szkolnym zasady określania typu czworokątów na podstawie ich różnych cech geometrycznych oraz własności relacji podobieństwa dla czworokątów.

Na liście FAKTY system może umieścić dowolną liczbę stwierdzeń użytkownika opisujących czworokąty. Umieszczanie stwierdzeń odbywa się w trybie dialogowym, przy czym użytkownik może używać języka naturalnego. Użycie języka naturalnego stało się możliwe dzięki zastosowaniu funkcji MATCH (wersja 2.) oraz posługiwaniu się zestawami słów kluczowych.

Przyjęto jednak pewne ograniczenia.

W pojedynczym stwierdzeniu użytkownik musi podać swoją nazwę czworokąta (np. symboliczną) oraz opisać ten czworokąt za pomocą dokładnie jednej z podanych cech:

- liczba osi symetrii,
- liczba boków równoległych,

- liczba boków równych,
- liczba kątów równych,
- typ (trapez, romb, prostokąt, równoległobok, kwadrat).

Użytkownik może ponadto podać stwierdzenie dotyczące podobieństwa między dwoma czworokątami używając nazw tych czworokątów. Niepodanie nazwy dla jakiegoś czworokąta spowoduje nadanie mu nazwy NIL. Jeśli mimo tych ograniczeń byłyby kłopoty z rozpoznaniem stwierdzenia użytkownika, system dialogowy przewiduje ponowne sformułowanie stwierdzenia przez użytkownika i rozpoczęcie procesu jego rozpoznania od początku.

Po rozpoznaniu wszystkich stwierdzeń użytkownika i przeformułowaniu ich do pewnej znormalizowanej postaci nastąpi proces stosowania zestawu reguł, czyli użycie funkcji FORWARD-CHAIN opisanej w rozdziale 5.4.
Zestaw składa się z 22 reguł, które dzielą się na 4 grupy. Reguły grupy 1 opisują zwrotność, symetryczność i przechodniość relacji podobieństwa. Reguły grupy 2 opisują relacje podobieństwa dla czworokątów. Reguły grupy 3 opisują zasady dobierania nazw dla czworokątów. Reguły grupy 4 opisują, jak zidentyfikować czworokąt na podstawie podanych cech.

Efektem działania systemu będzie uzupełnienie listy FAKTY o wszystkie wnioski, jakie można wyciągnąć ze stwierdzeń użytkownika w zakresie typu czworokątów oraz relacji podobieństwa między nimi.

Pełna definicja funkcji CZWOROKĄTY stanowiącej opisany powyżej system ekspercki podana została w rozdziałach 5.6.6 i 5.6.7. Rozdział 5.6.6 zawiera program główny, w którego skład wchodzą: instrukcja dla użytkownika, system dialogowy oraz uruchomienie funkcji FORWARD-CHAIN. W rozdziale 5.6.7 przedstawiono zestaw reguł.

Aby w GCLISP-ie uruchomić system ekspercki CZWOROKĄTY, należy uzupełnić GCLISP o specjalne narzędzia eksperckie, a więc:

- definicję funkcji MATCH - wersja 2.,
- definicję funkcji FORWARD-CHAIN,
- definicję funkcji CZWOROKĄTY,
- zainicjowanie zmiennej REGUŁY.

System ekspercki możemy uruchomić wywołując funkcję CZWOROKĄTY.

5.6 Zapis funkcji i przykładu w języku GC LISP

Wszystkie przykłady niniejszego opracowania powstały na podstawie systemu GOLDEN COMMON LISP, który stanowi wersję COMMON LISP-u na komputery personalne.

5.6.1 Omówienie języka GC LISP

GOLDEN COMMON LISP jest produktem GOLD HILL COMPUTERS. Jest przeznaczony dla maszyn IBM PC: XT, AT lub w pełni z nimi kompatybilnych oraz DEC Rainbow i Wang Professional Computer.

Stosowanie GCLISP-u - wymaga co najmniej:

- jednej stacji dysków elastycznych dla dwustronnych dyskietek o podwójnej gęstości,
- 512 K bajtów pamięci,
- systemu operacyjnego PC-DOS (lub MS-DOS) wersji 2.0 lub wyższych (np. wersji 3.0).

System GCLISP zawarty jest na pięciu dyskietkach elastycznych 5 1/4 cala i w jego skład wchodzą:

- interpreter LISP-u,
- edytor GMACS,
- narzędzia uruchamiania programów,
- system nauczania San Marco LISP Explorer.

Dokumentację GCLISP-u stanowią:
- users guide (100 stron),

- reference manual (200 stron)
- "LISP" - Winston, Horn ([3]),
- "COMMON LISP Reference Manual" - Steele .

San Marco LISP Explorer wersja 1.00 jest produktem San Marco Associates z roku 1984. Jest on przeznaczony do nauki podstawowych elementów języka LISP, a powstał jako uzupełnienie do drugiego wydania książki P.H. Winstona i B.K.P. Horna "LISP" z 1984 ([3]). Uczenie zorganizowano w formie pokazu slajdów. Użytkownik steruje pokazem zmieniając slajdy w miarę potrzeb. San Marco LISP Explorer zajmuje dwie dwustronne dyskietki. Materiał podzielony jest na 60 rozdziałów i obejmuje w sumie 645 slajdów.

W porównaniu z podręcznikiem San Marco LISP Explorer ma następujące udogodnienia:

- natychmiastowe sprzężenie zwrotne
  - przedstawione przez San Marco LISP Explorer przykłady są prezentowane w formie pytań i odpowiedzi, przy czym odpowiedź pojawia się na ekranie dopiero na żądanie użytkownika,
  - w każdej chwili użytkownik może wykonać własne przykłady;
- dynamiczne ilustracje
  - w skład programu nauczania wchodzi San Marco Inspector dostarczający dynamiczny obraz graficzny wykonywania programu. Jest to szczególnie przydatne przy analizie programów wykorzystujących procedury rekursywne,
  - ponadto San Marco LISP Explorer zawiera program demonstracyjny Blocks World. Jest to ilustracja rozwiązania problemu nowego sytuowania bloków w układzie wieloblokowych piramid przy jednocześnie ograniczonym miejscu na przestawiane bloki;
- ciekawe przykłady dotyczące:
  - rozwiązywania problemu poszukiwań,
  - systemów eksperckich opartych na produkcji,
  - interfejsu dla języka naturalnego,
  - reprezentowania wiedzy.

Edytor GMACS stanowi integralną część systemu GCLISP. Jest on edytorem specjalnie dostosowanym do języka LISP. Zapewnia wystarczającą liczbę buforów i jest dostatecznie szybki. Oprócz drobnych udogodnień takich jak: równoważenie nawiasów, wcinanie wyrażeń, ustawianie pozycji kursora GMAS daje bardzo istotne udogodnienia, a mianowicie: ewaluowanie definicji funkcji, ewaluowanie dowolnych s-wyrażeń, wyświetlanie listy argumentów, dokumentacji i makrorozszerzeń wyrażeń. Przy pamięci komputera 512 Kbajtów edytor GMACS daje możliwość edycji programów o objętości do 20 Kbajtów.

5.6.2 Zapis funkcji MATCH - wersja1.

```
; Definicje funkcji pomocniczych.

(DEFUN SHOVE-GR (VARIABLE ITEM A-LIST)
    (APPEND A-LIST (LIST (LIST VARIABLE ITEM))))

(DEFUN SHOVE-PL  (VARIABLE ITEM A-LIST)
    (COND ((NULL A-LIST)  (LIST(LIST VARIABLE (LIST ITEM))))
          ((EQUAL VARIABLE (CAAR A-LIST))
           (CONS (LIST VARIABLE (APPEND (CADAR A-LIST)  (LIST ITEM)))
                 (CDR A-LIST)))
          (T (CONS (CAR A-LIST)
                   (SHOVE-PL VARIABLE ITEM (CDR A-LIST)))))))

(DEFUN PATTERN-INDICATOR (L)
    (CAR L))
```

```
(DEFUN PATTERN-VARIABLE (L)
    (CADR L))

(DEFUN PULL-VALUE (VARIABLE A-LIST)
    (CADR (ASSOC VARIABLE A-LIST)))

(DEFUN RESTRICTION-INDICATOR (PATTERN-ITEM)
    (CADR PATTERN-ITEM))

(DEFUN RESTRICTION-PREDICATES (PATTERN-ITEM)
    (CDDR PATTERN-ITEM))

(DEFUN TEST (PREDICATES ARGUMENT)
    (COND ((NULL PREDICATES) T)
          ((FUNCALL (CAR PREDICATES) ARGUMENT)
           (TEST (CDR PREDICATES) ARGUMENT))
          (T NIL)))

; Definicja funkcji MATCH.

(DEFUN MATCH (P D ASSIGNMENTS)
    (COND ((AND (NULL P) (NULL D))
           (COND ((NULL ASSIGNMENTS) T)
                 (T ASSIGNMENTS)))
          ((OR (NULL P) (NULL D)) NIL)
          ((OR (EQUAL (CAR P) '?)
               (EQUAL (CAR P) (CAR D)))
           (MATCH (CDR P) (CDR D) ASSIGNMENTS))
          ((EQUAL (CAR P) '+)
           (OR (MATCH (CDR P) (CDR D) ASSIGNMENTS)
               (MATCH P (CDR D) ASSIGNMENTS)))
          ((ATOM (CAR P)) NIL)
          ((EQUAL (PATTERN-INDICATOR (CAR P)) '>)
           (MATCH (CDR P) (CDR D)
                  (SHOVE-GR (PATTERN-VARIABLE (CAR P))
                            (CAR D)
                            ASSIGNMENTS)))
          ((EQUAL (PATTERN-INDICATOR (CAR P)) '<)
           (MATCH (CONS (PULL-VALUE (PATTERN-VARIABLE (CAR P)) ASSIGNMENTS)
                        (CDR P))
                  D
                  ASSIGNMENTS))
          ((EQUAL (PATTERN-INDICATOR (CAR P)) '+)
           (LET ((NEW-ASSIGNMENTS (SHOVE-PL (PATTERN-VARIABLE (CAR P))
                                            (CAR D)
                                            ASSIGNMENTS)))
             (OR (MATCH (CDR P) (CDR D) NEW-ASSIGNMENTS)
                 (MATCH P (CDR D) NEW-ASSIGNMENTS))))
          ((AND (EQUAL (PATTERN-INDICATOR (CAR P))
                       'RESTRICT)
                (EQUAL (RESTRICTION-INDICATOR (CAR P)) '?)
                (TEST (RESTRICATION-PREDICATES (CAR P)) (CAR D)))
           (MATCH (CDR P) (CDR D) ASSIGNMENTS))
          ((AND (EQUAL (PATTERN-INDICATOR (CAR P))
                       'RESTRICT)
```

```
      (EQUAL (CAR (RESTRICTION-INDICATOR (CAR P))) '>)
      (TEST (RESTRICTION-PREDICATES (CAR P)) (CAR D)))
 (MATCH (CDR P) (CDR D)
        (SHOVE-GR (CADR (RESTRICTION-INDICATOR (CAR P)))
                  (CAR D)
                   ASSIGNMENTS)))))
```

5.6.3 Zapis funkcji MATCH - wersja 2.

Definicje funkcji pomocniczych.

```
(DEFUN SHOVE-GR (VARIABLE ITEM A-LIST)
   (APPEND A-LIST (LIST (LIST VARIABLE ITEM))))

(DEFUN SHOVE-PL (VARIABLE ITEM A-LIST)
   (COND ((NULL A-LIST) (LIST (LIST VARIABLE (LIST ITEM))))
         ((EQUAL VARIABLE (CAAR A-LIST))
          (CONS (LIST VARIABLE (APPEND (CADAR A-LIST) (LIST ITEM)))
                (CDR A-LIST)))
         (T (CONS (CAR A-LIST)
                  (SHOVE-PL VARIABLE ITEM (CDR A-LIST))))))

(DEFUN PATTERN-INDICATOR (L)
   (CAR L))

(DEFUN PATTERN-VARIABLE (L)
   (CADR L))

(DEFUN PULL-VALUE (VARIABLE A-LIST)
   (CADR (ASSOC VARIABLE A-LIST)))

(DEFUN RESTRICTION-INDICATOR (PATTERN-ITEM)
   (CADR PATTERN-ITEM))

(DEFUN RESTRICTION-PREDICATES (PATTERN-ITEM)
   (CDDR PATTERN-ITEM))

(DEFUN TEST (PREDICATES ARGUMENT)
   (COND ((NULL PREDICATES) T)
         ((FUNCALL (CAR PREDICATES) ARGUMENT)
          (TEST (CDR PREDICATES) ARGUMENT))
         (T NIL)))
```

```
(DEFUN EQUALPOL (WZOR DANA)
    (COND ((STRING-SEARCH (STRING WZOR) (STRING DANA)
                                        0 (LENGTH (STRING WZOR)))  T)
          (T  NIL)))


;  Definicja funkcji MATCH.


(DEFUN MATCH (P D ASSIGNMENTS)
    (COND ((OR (AND (NULL P) (NULL D))
               (AND (EQUAL (CAR P) '+) (NULL (CDR P)) (NULL D)))
           (COND ((NULL ASSIGNMENTS) T)
                 (T ASSIGNMENTS)))
          ((OR (NULL P) (NULL D)) NIL)
          ((OR (EQUAL (CAR P) '?)
               (EQUAL (CAR P) (CAR D))
               (AND (ATOM (CAR P)) (ATOM (CAR D)) (EQUALPOL (CAR P)
                                                            (CAR D))))
           (MATCH (CDR P) (CDR D) ASSIGNMENTS))
          ((EQUAL (CAR P) '+)
           (OR (MATCH (CDR P) D ASSIGNMENTS)
               (MATCH (CDR P) (CDR D) ASSIGNMENTS)
               (MATCH P (CDR D) ASSIGNMENTS)))
          ((ATOM (CAR P)) NIL)
          ((EQUAL (PATTERN-INDICATOR (CAR P)) '>)
           (MATCH (CDR P) (CDR D)
                  (SHOVE-GR (PATTERN-VARIABLE (CAR P))
                            (CAR D)
                            ASSIGNMENTS)))
          ((EQUAL (PATTERN-INDICATOR (CAR P)) '<)
           (MATCH (CONS (PULL-VALUE (PATTERN-VARIABLE (CAR P)) ASSIGNMENTS)
                        (CDR P))
                  D
                  ASSIGNMENTS))
          ((EQUAL (PATTERN-INDICATOR (CAR P)) '+)
           (LET ((NEW-ASSIGNMENTS (SHOVE-PL (PATTERN-VARIABLE (CAR P))
                                            (CAR D)
                                            ASSIGNMENTS)))
             (OR (MATCH (CDR P) (CDR D) NEW-ASSIGNMENTS)
                 (MATCH P (CDR D) NEW-ASSIGNMENTS))))
          ((AND (EQUAL (PATTERN-INDICATOR (CAR P))
                       'RESTRICT)
```

```
     (EQUAL (RESTRICTION-INDICATOR (CAR P)) '?)
     (TEST (RESTRICTION-PREDICATES (CAR P)) (CAR D)))
 (MATCH (CDR P) (CDR D) ASSIGNMENTS))
((AND (EQUAL (PATTERN-INDICATOR (CAR P)) 'RESTRICT)
      (EQUAL (CAR (RESTRICTION-INDICATOR (CAR P))) '>)
      (TEST (RESTRICTION-PREDICATES (CAR P)) (CAR D)))
 (MATCH (CDR P) (CDR D)
        (SHOVE-GR (CADR (RESTRICTION-INDICATOR (CAR P)))
                  (CAR D)
                  ASSIGNMENTS)))))
```

5.6.4 Program podtrzymujący dialog - PRZYJACIEL

PRZYJACIEL

```
(DEFUN PRZYJACIEL ()
  (FORMAT T "
```

PRZYJACIEL

Program PRZYJACIEL prowadzi konwersacje w języku naturalnym. Użytkownik powinien każda swoją wypowiedź zamykać w okrągłych nawiasach.Aby zakończyć dialog należy użyć w swojej wypowiedzi zwrotu DO WIDZENIA.

```
")
(SETQ *PRINT-LEVEL* NIL)
(SETQ *PRINT-LENGTH* 30)
(SETQ L T)
(SETQ WSK T)
(SETQ LICZ T)
(PRINT '(Opowiedz mi o swoich klopotach))

(TERPRI)

(DO ((S (READ) (READ)) (A-LIST NIL NIL) (DOM) (SYN) (CORK) (BRAT)
     (SIOSTR) (MAZ) (ZON) (MATK) (OJC) (DZIEC))

    (NIL)

    (COND ((> (LENGTH S) 15)
           (PRINT '(Bardzo prosze uzywaj troche krotszych zdan)))

          ((MATCH '(?) S NIL)
           (PRINT '(Nie odpowiadaj tak lakonicznie)))


          ( (AND (OR (MATCH '(+ SPAC +) S NIL)
                     (MATCH '(+ SPIAC +) S NIL)
                     (MATCH '(+ SENN +) S NIL)
                     (MATCH '(+ SEN +) S NIL)
                     (MATCH '(+ ZMECZ +) S NIL))
                 (OR (MATCH '(+ JESTEM +) S NIL)
                     (MATCH '(+ CHCE +) S NIL)
                     (MATCH '(+ CZUJE +) S NIL)
                     (MATCH '(+ MAM +) S NIL)))
            (PRINT '(Myslalem ze jeszcze porozmawiamy - ale trudno))
            (RETURN 'DOBRANOC))
```

```
((OR (MATCH '(+ POLITY +) S NIL)
     (MATCH '(+ GOSPODAR +) S NIL)
     (MATCH '(+ REFORM +) S NIL)
     (MATCH '(+ MIEDZYNARODOW +) S NIL))
  (PRINT '(Do takich tematow nie jestem zaprogramowany))
  (RETURN 'DO_WIDZENIA))


((OR (MATCH '(+ PIENI +) S NIL)
     (MATCH '(+ GOTOWK +) S NIL)
     (MATCH '(+ FORS +) S NIL)
     (MATCH '(+ FINANS +) S NIL)
     (MATCH '(+ ZARAB +) S NIL)
     (MATCH '(+ ZAROB +) S NIL))

  (PRINT '(Podaj ile zlotych ci brakuje /cyframi bez spacji/))
  (TERPRI)
  (SETQ S (READ))

(COND ((SETQ A-LIST (MATCH '(+ (RESTRICT (> P) NUMBERP) +) S NIL))

  (COND ((< (MATCH-VALUE 'P A-LIST) 10000)

(PRINT '(Taka sume mozesz zarobic nawet na panstwowej posadzie)))

        ((>= (MATCH-VALUE 'P A-LIST) 10000)

          (PRINT '(Graj w TOTOLOTKA)))))
(T  (PRINT '(Jesli nie chcesz rozmawiac o pieniadzach to opowiedz mi o swojej pracy)))))

((MATCH '(+ DOM +) S NIL)

  (SETQ DOM T)

  (PRINT '(Opowiedz mi o swojej rodzinie)))

((MATCH '(+ SYN +) S NIL)  (SETQ SYN T)
  (PRINT '(Opowiedz  mi wiecej o swoich dzieciach)))
((MATCH '(+ CORK +) S NIL) (SETQ CORK T)
  (PRINT '(Opowiedz mi wiecej o swoich dzieciach)))
((MATCH '(+ BRAT +) S NIL) (SETQ BRAT T)
  (PRINT '(Powiedz mi cos jeszcze o swoim rodzenstwie)))
((MATCH '(+ SIOSTR +) S NIL) (SETQ SIOSTR T)
  (PRINT '(Powiedz mi cos jeszcze o swoim rodzenstwie)))
((OR (MATCH '(+ MAZ +) S NIL)
     (MATCH '(+ MEZ +) S NIL))   (SETQ MAZ T)
  (PRINT '(Jakie jest Twoje malzenstwo?)))
((MATCH '(+ ZON +) S NIL) (SETQ ZON T)
  (PRINT '(Jakie jest Twoje malzenstwo?)))
((MATCH '(+ MATK +) S NIL) (SETQ MATK T)
  (PRINT '(Opowiedz jeszcze o swoich rodzicach)))
((MATCH '(+ OJC +) S NIL) (SETQ OJC T)
  (PRINT '(Opowiedz jeszcze o swoich rodzicach)))
((MATCH '(+ DZIEC +) S NIL) (SETQ DZIEC T)
  (PRINT '(Jak uklada sie Twoje zycie rodzinne?)))



((OR (AND (MATCH '(+ CZAS +) S NIL)
          (NOT (MATCH '(+ CZASEM +) S NIL))
          (NOT (MATCH '(+ CZASAMI +) S NIL)))
     (MATCH '(+ CZASU +) S NIL))
  (PRINT '(Czas to pieniadz - Jak wygladaja Twoje finanse?)))

((MATCH '(+ KOMPUTER +) S NIL)

(PRINT '(Jesli juz mowimy o komputerach powiedz czy masz w domu swoj komputer)))


((OR (MATCH '(+ ZDROW +) S NIL)
     (MATCH '(+ CZUJE +) S NIL)
     (MATCH '(+ BOL +) S NIL)
     (MATCH '(+ CHOR +) S NIL))
  (PRINT '(Czy masz temperature?))
  (TERPRI)
  (SETQ S (READ))
  (COND ((OR (MATCH '(+ TAK +) S NIL)
             (AND (NOT (MATCH '(+ NIE +) S NIL))
                  (MATCH '(+ MAM +) S NIL)))
    (PRINT '(Nie siedz przy komputerze tylko idz do lekarza)))
        ((OR (MATCH '(+ NIE WIEM +) S NIL)
             (MATCH '(+ NIE MIERZYL +) S NIL))
(PRINT '(W Twoim wieku nie bagatelizuje sie tak waznych spraw jak zdrowie!)))
        (T
 (PRINT '(Nie zajmujmy sie wiec glupstwami opowiedz lepiej o swojej pracy)))))
```

```
        ((MATCH '(+ PRAC +) S NIL)

           (PRINT '(Opowiedz wiecej o swojej pracy))
           (TERPRI)
           (SETQ S (READ))
           (COND ((OR (MATCH '(+ PENSJ +) S NIL)
                      (MATCH '(+ PODWYZ +) S NIL)
                      (MATCH '(+ ZAROB +) S NIL)
                      (MATCH '(+ ZARAB +) S NIL)
                      (MATCH '(+ PIENI +) S NIL)
                      (MATCH '(+ FINANS +) S NIL)
                      (MATCH '(+ PLAC +) S NIL))
(PRINT '(Pieniadze szczescia nie daja ale bez nich tez nie jest wesolo)))
                  ((OR (MATCH '(+ WYMOWIEN +) S NIL)
                       (MATCH '(+ WYMOWIL +) S NIL)
                       (MATCH '(+ WYPOWIEDZEN +) S NIL))
(PRINT '(Lepiej skonczmy rozmowe i lec szukac pracy)))
                  ((OR (MATCH '(+ DYREK +) S NIL)
                       (MATCH '(+ SZEF +) S NIL)
                       (MATCH '(+ KIEROW +) S NIL)
                       (MATCH '(+ ZWIERZCHN +) S NIL))
(PRINT '(No uklady nie ma rady)))
                  (T
(PRINT '(Moze jednak lepiej porozmawiajmy o Twoim zdrowiu)))))

        ((MATCH '(+ PRZYJA +) S NIL)

           (PRINT '(Gdzie pracuje Twoj przyjaciel?)))

        (DOM

           (SETQ DOM NIL)

           (PRINT '(Mowiles wczesniej o domu)))

       (SYN   (SETQ SYN NIL)
          (PRINT '(Wspominales wczesniej o swoim synu)))
       (CORK  (SETQ CORK NIL)
          (PRINT '(Powiedz cos jeszcze o swojej corce)))
       (BRAT  (SETQ BRAT NIL)
(PRINT '(Mowilismy wczesniej o Twoim bracie - Czym on sie zajmuje? )))
       (SIOSTR (SETQ SIOSTR NIL)
          (PRINT '(Chyba wspominales o swojej siostrze)))
       (MAZ   (SETQ MAZ NIL)
          (PRINT '(Opowiedz mi o pracy Twojego meza)))
       (ZON   (SETQ ZON NIL)
          (PRINT '(Czy Twoja zona pracuje?)))
       (MATK  (SETQ MATK NIL)
          (PRINT '(Mowiles wczesniej o swojej matce)))
       (OJC   (SETQ OJC NIL)
          (PRINT '(Mowiles wczesniej o swoim ojcu)))


       (DZIEC (SETQ DZIEC NIL)
          (PRINT '(Opowiedz mi jakie masz klopoty ze swoimi dziecmi))
          (TERPRI)
          (DO ((S (READ) (READ)) (SZKOLA) )
              (NIL)
              (COND ((MATCH '(+ SZKOL +) S NIL)
                     (SETQ SZKOLA T)
                     (PRINT '(Opowiedz o stosunku do rodzicow)))
                    ((OR (MATCH '(+ NAUKA +) S NIL)
                         (MATCH '(+ UCZ +) S NIL)
                         (MATCH '(+ LEKCJ +) S NIL))
                     (PRINT '(A w domu czy dzieci Ci pomagaja)))
                    ((AND (OR (MATCH '(+ POMOC +) S NIL)
                              (MATCH '(+ POMAGA +) S NIL)
                              (MATCH '(+ UCZYNN +) S NIL)
                              (MATCH '(+ OPIEKU +) S NIL))
                          (NOT (MATCH '(+ NIE +) S NIL)))
                     (PRINT '(To sprawa najwazniejsza - gratuluje)))
                    ((OR (MATCH '(+ NIEGRZECZN +) S NIL)
                         (MATCH '(+ BIJE +) S NIL)
                         (MATCH '(+ NIEPOSLUSZN +) S NIL)
                         (MATCH '(+ LEN +) S NIL)
                         (MATCH '(+ KONFLIKT +) S NIL))
                     (PRINT '(To bardzo zle - dodaj cos jeszcze)))
                    (SZKOLA   (SETQ SZKOLA NIL)
                     (PRINT '(Mowiles wczesniej o szkole)))
                    (T
                     (COND (L
                              (PRINT '(Powiedz jeszcze cos o dzieciach))
                              (SETQ L NIL))
                           (T (PRINT '(Porozmawiajmy o innych sprawach))
                              (RETURN))))) (TERPRI)  ))
```

```
((MATCH '(+ DO WIDZENIA +) S NIL)
   (PRINT '(Dziekuje za rozmowe)) (RETURN 'DO_WIDZENIA))

(T    (COND (LICZ  (PRINT '(Moze dodaj cos jeszcze))
                   (SETQ LICZ NIL))

              (WSK   (SETQ WSK NIL)
               (PRINT '(Na jaki inny temat moglibysmy porozmawiac))
               (TERPRI)
               (SETQ S (READ))
               (COND ((OR (MATCH '(+ DOM +) S NIL)
                          (MATCH '(+ RODZIN +) S NIL)
                          (MATCH '(+ ZDROW +) S NIL)
                          (MATCH '(+ PRAC +) S NIL)
                          (MATCH '(+ DZIEC +) S NIL)
                          (MATCH '(+ PIENI +) S NIL)
                          (MATCH '(+ OGOL +) S NIL))
                      (PRINT '(Znakomicie - zaczynaj)))
                     (T
(PRINT '(Proponuje jednak zebysmy porozmawiali o Twoich klopotach rodzinnych)))))

              (T

                 (PRINT '(Nie mamy juz chyba o czym rozmawiac))

                 (RETURN 'DO_WIDZENIA)))))

     (TERPRI)))


(DEFUN MATCH-VALUE (KEY A-LIST)

  (CADR (ASSOC KEY A-LIST)))
```

5.6.5 Zapis funkcji FORWARD-CHAIN.

```
; Definicje podstawowych funkcji pomocniczych.

(DEFUN FILTER-ASSERTIONS (PATTERN INITIAL-A-LIST)
  (DO ((ASSERTIONS FAKTY (REST ASSERTIONS))
       (A-LIST-STREAM (MAKE-EMPTY-STREAM)))
      ((NULL ASSERTIONS) A-LIST-STREAM)
    (LET ((NEW-A-LIST (MATCH PATTERN (FIRST ASSERTIONS) INITIAL-A-LIST)))
      (COND (NEW-A-LIST (SETQ A-LIST-STREAM
                              (ADD-TO-STREAM NEW-A-LIST A-LIST-STREAM)))))))

(DEFUN FILTER-A-LIST-STREAM (PATTERN A-LIST-STREAM)
  (COND ((EMPTY-STREAM-P A-LIST-STREAM) (MAKE-EMPTY-STREAM))
        (T (COMBINE-STREAMS
             (FILTER-ASSERTIONS PATTERN (FIRST-OF-STREAM A-LIST-STREAM))
             (FILTER-A-LIST-STREAM PATTERN (REST-OF-STREAM A-LIST-STREAM))))))

(DEFUN CASCADE-THROUGH-PATTERNS (PATTERNS A-LIST-STREAM)
  (COND ((NULL PATTERNS) A-LIST-STREAM)
        (T (FILTER-A-LIST-STREAM (FIRST PATTERNS)
                                 (CASCADE-THROUGH-PATTERNS (REST PATTERNS)
                                                           A-LIST-STREAM)))))

(DEFUN SPREAD-THROUGH-THENS (RULE-NAME IFS THENS A-LIST)
  (DO ((THENS THENS (REST THENS))
       (ACTION-STREAM (MAKE-EMPTY-STREAM)))
      ((NULL THENS) ACTION-STREAM)
    (LET ((IFS (REPLACE-VARIABLES IFS A-LIST))
          (THENS (REPLACE-VARIABLES THENS A-LIST)))
      (COND ((REMEMBER-THENS THENS)
             (REPORT-ACTION RULE-NAME IFS THENS)
             (SETQ ACTION-STREAM (COMBINE-STREAMS THENS ACTION-STREAM)))))))

(DEFUN REMEMBER-THENS (THENS)
  (DO ((THENS THENS (REST THENS))
       (SWITCH NIL))
      ((NULL THENS) SWITCH)
    (WHEN (REMEMBER (FIRST THENS)) (SETF SWITCH T))))
```

```
(DEFUN REPORT-ACTION (RULE-NAME IFS THENS)
  (FORMAT T "~%Regula ~a stwierdza: ~a" RULE-NAME (FIRST THENS))
  (MAPCAR #'(LAMBDA (E) (FORMAT T "~%      ~a" E)) (REST THENS))
  (FORMAT T "~%      ponieważ: ~a" (FIRST IFS))
  (MAPCAR #'(LAMBDA (E) (FORMAT T "~%                ~a" E)) (REST IFS)))

(DEFUN REPLACE-VARIABLES (S A-LIST)
  (COND ((ATOM S) S)
        ((MEMBER (FIRST S) '(< >))
         (SECOND (ASSOC (PATTERN-VARIABLE S) A-LIST)))
        (T (CONS (REPLACE-VARIABLES (FIRST S) A-LIST)
                 (REPLACE-VARIABLES (REST S) A-LIST)))))

(DEFUN FEED-TO-THENS (RULE-NAME IFS THENS A-LIST-STREAM)
  (COND ((EMPTY-STREAM-P A-LIST-STREAM) (MAKE-EMPTY-STREAM))
        (T (COMBINE-STREAMS
            (SPREAD-THROUGH-THENS RULE-NAME
                                  IFS THENS
                                  (FIRST-OF-STREAM A-LIST-STREAM))
            (FEED-TO-THENS RULE-NAME
                           IFS THENS
                           (REST-OF-STREAM A-LIST-STREAM))))))


(DEFUN USE-RULE (RULE)
  (LET* ((RULE-NAME (SECOND RULE))
         (IFS (REVERSE (REST (THIRD RULE))))
         (THENS (REST (THIRD (REST RULE))))   ;GCLISP does not have FOURTH.
         (A-LIST-STREAM (CASCADE-THROUGH-PATTERNS
                         IFS
                         (ADD-TO-STREAM NIL (MAKE-EMPTY-STREAM))))
         (ACTION-STREAM (FEED-TO-THENS RULE-NAME IFS THENS A-LIST-STREAM)))
    (NOT (EMPTY-STREAM-P ACTION-STREAM))))



;  Definicja funkcji FORWARD-CHAIN.


(DEFUN FORWARD-CHAIN ()
  (DO ((RULES-TO-TRY REGULY (REST RULES-TO-TRY))
       (NO-PROGRESS T))
      ((AND (NULL RULES-TO-TRY) NO-PROGRESS)
       (FORMAT T "~%Niczego wiecej nie mozna wywnioskowac."))
    (WHEN (NULL RULES-TO-TRY)
          (SETQ RULES-TO-TRY REGULY NO-PROGRESS T)
          (FORMAT T "~%Kolejne przejscie przez reguly..."))
    (WHEN (USE-RULE (FIRST RULES-TO-TRY)) (SETQ NO-PROGRESS NIL))))



;  Pomocnicze procedury dostepu.


(DEFUN COMBINE-STREAMS (S1 S2) (APPEND S1 S2))

(DEFUN ADD-TO-STREAM (E S) (CONS E S))

(DEFUN FIRST-OF-STREAM (S) (FIRST S))

(DEFUN REST-OF-STREAM (S) (REST S))

(DEFUN EMPTY-STREAM-P (S) (NULL S))

(DEFUN MAKE-EMPTY-STREAM () NIL)



;  Definicja funkcji realizujacej umieszczenie nowego stwierdzenia
;  na liscie FAKTY.


(DEFUN REMEMBER (NEW)
  (COND ((MEMBER NEW FAKTY :TEST 'EQUAL) NIL)
        (T (SETQ FAKTY (CONS NEW FAKTY)) NEW)))
```

6.6.6 Przykładowy system eksperoki - CZWOROKĄTY (ozęśó I)

CZWOROKĄTY

(ozęśó I - program główny)

Definicja funkcji CZWOROKĄTY, która stanowi program główny reprezentowanego systemu eksperckiego. W skład tego programu wohodzą: zasady korzystania z systemu, budowa bazy danych (lista FAKTY) i wyprowadzenie wniosków.

```
(DEFUN CZWOROKĄTY ()
(FORMAT T "
                Przykładowy system eksperoki

                        CZWOROKĄTY
    System podaje typ (trapez, romb, równoległobok, kwadrat, prostokąt) dla
    opisanych przez użytkownika czworokątów oraz określa między nimi relacje
    podobieństwa.

                Instrukcja dla użytkownika

    Na liście FAKTY system może umieścić dowolną liczbę stwierdzeń użytkownika opisujących
czworokąty. Użytkownik może używać języka naturalnego. W pojedynczym stwierdzeniu musi podać
swoją nazwę czworokąta (np. symboliczną) oraz opisać ten czworokąt za pomocą dokładnie jed-
nej z podanych cech:

                1. liczba osi symetrii
                2. liczba boków równoległych
                3. liczba boków równych
                4. liczba kątów równych
                5. typ (trapez, romb, prostokąt, równoległobok, kwadrat) .

Ponadto użytkownik może podać stwierdzenie dotyczące podobieństwa między dwoma czworokątami
używając nazw tych czworokątów.

    Efektem działania systemu jest uzupełnienie listy FAKTY o wszystkie wnioski jakie można
wyciągnąć ze stwierdzeń użytkownika w zakresie typu czworokątów oraz relacji podobieństwa mię-
dzy nimi. Stwierdzenia wprowadzone przez użytkownika zostaną zapamiętane na liście FAKTY-WPRO-
WADZONE.
    Aby uruchomić system należy napisać (START). ")
  (SETQ *PRINT-LENGTH* NIL)
  (SETQ FAKTY NIL)
  (TERPRI)
  (DO ((S (READ) (READ)))
      (NIL)
      (COND ((MATCH '(START) S NIL) (RETURN))
            (T (PRINT '(Aby uruchomić program należy napisać (START)))
               (TERPRI))))
  (STWIERDZENIE)
  (TERPRI)
  (DO ((S (READ) (READ)))
      (NIL)
      (COND ((MATCH '(TAK) S NIL) (STWIERDZENIE) (TERPRI))
            (T (SETQ FAKTY-WPROWADZONE FAKTY)
```

```
                    (PRINT '(Czekaj na wyniki))
                    (RETURN))))
    (FORWARD-CHAIN) )
```

Definicja funkcji STWIERDZENIE, która umożliwia użytkownikowi umieszczenie pojedynczego stwierdzenia na liście FAKTY.

```
(DEFUN STWIERDZENIE ()
(PRINT '(Podaj stwierdzenie))
(TERPRI)
(DO ((S (READ) (READ)) (ST))
    (NIL)
    (COND ((MATCH '(TAK) S NIL) (SETQ FAKTY (APPEND ST FAKTY))
                                (PRINT '(Czy chcesz podać nowy fakt))
                                (RETURN))
          ((MATCH '(NIE) S NIL) (PRINT
                                '(Spróbuj jeszcze raz-Podaj stwierdzenie))
                                (TERPRI))
          ((MATCH '(+ SYMETRII +) S NIL)
              (COND ((OR (MATCH '(+ 1 +) S NIL))
                         (MATCH '(+ JEDNA +) S NIL))
                     (SZUKANIE-NAZWY)
                     (SETQ ST1 '(NAZWA MA JEDNĄ OŚ SYMETRII))
                     (SETQ ST (LIST ST1))
                     (PRINT '(Czy chodzi o stwierdzenie że: ,ST1))
                ((OR (MATCH '(+ 2 +) S NIL)
                     (MATCH '(+ DWIE +) S NIL))
                     (SZUKANIE-NAZWY)
                     (SETQ ST1 '(NAZWA MA DWIE OSIE SYMETRII))
                     (SETQ ST (LIST ST1))
                     (PRINT '(Czy chodzi o stwierdzenie że: ,ST1)))
                ((OR (MATCH '(+ 4 +) S NIL)
                     (MATCH '(+ CZTERY +) S NIL)
                     (SZUKANIE-NAZWY)
                     (SETQ ST1 '(NAZWA MA CZTERY OSIE SYMETRII))
                     (SETQ ST (LIST ST1))
                     (PRINT '(Czy chodzi o stwierdzenie że: ,ST1)))
                (T (PRINT '(Spróbuj jeszcze raz-Podaj stwierdzenie))
                   (TERPRI))))
          ((MATCH '(+ KĄT +) S NIL)
              (COND ((AND (OR (MATCH '(+ 1 +) S NIL)
                              (MATCH '(+ JEDNĄ +) S NIL)
                              (MATCH '(+ DWA +) S NIL)
```

```
                    (MATCH `(+ 2 KĄTY +) S NIL))
                (NOT (MATCH `(+ PO +) S NIL)))
                    (SZUKANIE-NAZWY)
                    (SETQ ST1 `(,NAZWA MA JEDNĄ PARĘ RÓWNYCH KĄTÓW))
                    (SETQ ST (LIST ST1))
                    (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
            ((OR (MATCH `(+ 2 PARY +) S NIL)
                 (MATCH `(+ DWIE +) S NIL)
                 (MATCH `(+ PO +) S NIL))
                  (SZUKANIE-NAZWY)
                  (SETQ ST1 `(,NAZWA MA DWIE PARY RÓWNYCH KĄTÓW))
                  (SETQ ST (LIST ST1))
                  (PRINT `(Czy chodzi o stwierdzenie że:,ST1)))
            ((OR (MATCH `(+ 4 +) S NIL)
                 (MATCH `(+ CZTERY +) S NIL)
                 (MATCH (+ WSZYSTKIE +) S NIL))
                  (SZUKANIE-NAZWY)
                  (SETQ ST1 `(,NAZWA MA WSZYSTKIE KĄTY RÓWNE))
                  (SETQ ST (LIST ST1))
                  (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
              (T (PRINT `(Spróbuj jeszcze raz-Podaj stwierdzenie))
                 (TERPRI))))
((AND (MATCH `(+ BOK +) S NIL)
      (OR (MATCH `(+ RÓWNE +) S NIL)
          (MATCH `(+ RÓWNY +) S NIL)))
      (COND ((AND (OR (MATCH `(+ 1 +) S NIL)
                      (MATCH `(+ JEDNA +) S NIL)
                      (MATCH `(+ DWA +) S NIL)
                      (MATCH `(+ 2 BOKI +) S NIL))
                  (NOT (MATCH `(+ PO +) S NIL)))
                  (SZUKANIE-NAZWY)
                  (SETQ ST1 `(,NAZWA MA JEDNĄ PARĘ RÓWNYCH BOKÓW))
                  (SETQ ST (LIST ST1))
                  (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
             ((OR (MATCH `(+ 2 PARY +) S NIL)
                  (MATCH `(+ DWIE +) S NIL)
                  (MATCH `(+ PO +) S NIL))
                  (SZUKANIE-NAZWY)
                  (SETQ ST1`(,NAZWA MA DWIE PARY RÓWNYCH BOKÓW))
                  (SETQ ST (LIST ST1))
                  (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
             ((OR (MATCH `(+ 4 +) S NIL)
                  (MATCH `(+ CZTERY +) S NIL)
                  (MATCH `(+ WSZYSTKIE +) S NIL))
                  (SZUKANIE-NAZWY)
                  (SETQ ST1 `(,NAZWA MA WSZYSTKIE BOKI RÓWNE))
                  (SETQ ST (LIST ST1))
                  (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
```

```
            (T (PRINT `(Spróbuj jeszoze raz-Podaj stwierdzenie))
               (TERPRI))))
((AND (MATCH `(+ BOK +) S NIL)
      (MATCH `(+ RÒWNOLEG +) S NIL))
       (COND ((AND (OR (MATCH `(+ 1 +) S NIL)
                       (MATCH `(+ JEDNA +) S NIL)
                       (MATCH `(+ DWA +) S NIL)
                       (MATCH `(+ 2 BOKI +) S NIL))
                   (NOT (MATCH `(+ PO +) S NIL)))
                   (SZUKANIE-NAZWY)
                   (SETQ ST1 `(,NAZWA MA JEDNĄ PARĘ BOKÒW RÒWNOLEGLYCH))
                   (SETQ ST (LIST ST1))
                   (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
              ((OR (MATCH `(+ 2 PARY +) S NIL)
                   (MATCH `(+ DWIE +) S NIL)
                   (MATCH `(+ PO +) S NIL))
                   (SZUKANIE-NAZWY)
                   (SETQ ST1 `(,NAZWA MA DWIE PARY BOKÒW RÒWNOLEGLYCH))
                   (SETQ ST (LIST ST1))
                   (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
            (T PRINT `(Spróbuj jeszoze raz-Podaj stwierdzenie))
               (TERPRI))))
((MATCH `(+ PODOBN +) S NIL)
   (SETQ NAZWA1 (MATCH-VALUE 'N1
                  (MATCH `(+(RESTRICT (>N1)
                                      NIEKLUCZ) +
                             (RESTRICT (>N2)
                                      NIEKLUCZ) +)
                          S NIL)))
   (SETQ NAZWA2 (MATCH-VALUE 'N2
                  (MATCH `(+(RESTRICT (>N1)
                                      NIEKLUCZ) +
                             (RESTRICT (>N2)
                                      NIEKLUCZ) +)
                          S NIL)))
   (SETQ ST1 `(,NAZWA1 JEST PODOBNA DO ,NAZWA2))
   (SETQ ST (LIST ST1))
   (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
((MATCH `(+ TRAPEZ +) S NIL)
  (SZUKANIE-NAZWY)
  (SETQ ST1 `(,NAZWA JEST TRAPEZEM))
  (SETQ ST (LIST ST1))
```

```
      (PRINT `(Czy chodzi o stwierdzenie że:  ST1)))
   ((MATCH `(+ ROMB +) S NIL)
      (SZUKANIE-NAZWY)
      (SETQ ST1 `(,NAZWA JEST ROMBEM))
      (SETQ ST (LIST ST1))
      (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
   ((MATCH `(+ RÒWNOLEGŁOBOK +) S NIL)
      (SZUKANIE-NAZWY)
      (SETQ ST1 `(,NAZWA JEST RÒWNOLEGŁOBOKIEM))
      (SETQ ST (LIST ST1))
      (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
   ((MATCH `(+ PROSTOKĄT +) S NIL)
      (SZUKANIE-NAZWY)
      (SETQ ST1 `(,NAZWA JEST PROSTOKĄTEM))
      (SETQ ST (LIST ST1))
      (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
   ((MATCH `(+ KWADRAT +) S NIL)
      (SZUKANIE-NAZWY)
      (SETQ ST1 `(,NAZWA JEST KWADRATEM))
      (SETQ ST (LIST ST1))
      (PRINT `(Czy chodzi o stwierdzenie że: ,ST1)))
(T (PRINT `(Spróbuj jeszcze raz-Podaj stwierdzenie)) (TERPRI)))))
```

Definicje funkcji pomocniczych.

```
(DEFUN NIEKLUCZ (DANA)
  (NOT (KLUCZ DANA)))
(DEFUN KLUCZ (DANA)
  (MEMBER DANA `(JEST DO MA I ORAZ OSIE OS SYMETRII JEDNA 1 2 4 CZTERY
                 DWA DWIE WSZYSTKIE PO W BOKI KATY RÒWNE RÒWNYCH
                 RÒWNOLEGŁE RÒWNOLEGŁYCH PARE PARA PARY PODOBNA PODOBNYCH
                 PODOBNY PODOBNE BOKÒW KĄTÒW SA FIGURA FIGURY TO
                 CZWOROKĄT CZWOROKĄTA POSIADA)))

(DEFUN SZUKANIE-NAZWY( )
  (SETQ NAZWA (MATCH-VALUE 'N (MATCH `(+ RESTRICT (>N) NIEKLUCZ) +) S NIL))))

(DEFUN MATCH-VALUE (KEY A-LIST)
  (CADR (ASSOC KEY A-LIST)))
```

5.6.7 Przykładowy system eksperoki - CZWOROKĄTY (część II)

CZWOROKĄTY

(część II - reguły)

```
(SETQ REGUŁY '(

  Reguły 1-3 opisują zwrotność, symetryczność i przechodniość relacji podobieństwa

 (REGUŁA NUMER1
  (JEŚLI  ((>FIGURA)  JEST  ?))
  (TO     ((<FIGURA)  JEST PODOBNA DO (<FIGURA))))

 (REGUŁA NUMER2
  (JEŚLI ((>FIGURA)  JEST PODOBNA DO (>FIGURA1)))
  (TO     ((<FIGURA1)  JEST PODOBNA DO (<FIGURA))))

 (REGUŁA NUMER3
  (JEŚLI ((>FIGURA)  JEST PODOBNA DO (>FIGURA1))
         ((<FIGURA1)  JEST PODOBNA DO (>FIGURA2)))
  (TO    ((<FIGURA)  JEST PODOBNA DO (<FIGURA2))))

  Reguły 4-5 opisują relacje podobieństwa dla czworokątów

 (REGUŁA NUMER4
  (JEŚLI ((>FIGURA)  JEST KWADRATEM)
         ((>FIGURA1)  JEST KWADRATEM))
  (TO    ((<FIGURA)  JEST PODOBNA DO (<FIGURA1))))

 (REGUŁA NUMER5
  (JEŚLI ((>FIGURA)  JEST (>TYPU))
         ((>FIGURA1)  JEST PODOBNA DO (<FIGURA)))
  (TO    ((<FIGURA1)  JEST (<TYPU))))

  Reguły 6-9 opisują zasady dobierania nazw dla czworokątów

 (REGUŁA NUMER6
  (JEŚLI ((>FIGURA)  JEST KWADRATEM))
  (TO    ((<FIGURA)  JEST ROMBEM)))

 (REGUŁA NUMER7
  (JEŚLI ((>FIGURA)  JEST KWADRATEM))
  (TO    ((<FIGURA)  JEST PROSTOKĄTEM)))

 (REGUŁA NUMER8
  (JEŚLI ((>FIGURA)  JEST ROMBEM))
  (TO    ((<FIGURA)  JEST RÓWNOLEGŁOBOKIEM)))

 (REGUŁA NUMER9
  (JEŚLI ((>FIGURA)  JEST PROSTOKĄTEM))
  (TO    ((<FIGURA)  JEST RÓWNOLEGŁOBOKIEM)))

  Reguły 10-22 opisują  jak zidentyfikować czworokąt na podstawie podanych cech

 (REGUŁA NUMER10
  (JEŚLI  ((>FIGURA)  MA CZTERY OSIE SYMETRII))
  (TO     ((<FIGURA)  JEST KWADRATEM)))
```

```
(REGUŁA NUMER11
  (JEŚLI ((>FIGURA) MA WSZYSTKIE KĄTY RÓWNE)
         ((<FIGURA) MA WSZYSTKIE BOKI RÓWNE))
  (TO    ((<FIGURA) JEST KWADRATEM)))

(REGUŁA NUMER12
  (JEŚLI ((>FIGURA) MA DWIE OSIE SYMETRII)
         ((<FIGURA) MA DWIE PARY RÓWNYCH KĄTÓW))
  (TO    ((<FIGURA) JEST ROMBEM)))

(REGUŁA NUMER13
  (JEŚLI ((>FIGURA) MA WSZYSTKIE BOKI RÓWNE))
  (TO    ((<FIGURA) JEST ROMBEM)))

(REGUŁA NUMER14
  (JEŚLI ((>FIGURA) MA DWIE OSIE SYMETRII)
         ((<FIGURA) MA DWIE PARY RÓWNYCH BOKÓW))
  (TO    ((<FIGURA) JEST PROSTOKĄTEM)))

(REGUŁA NUMER15
  (JEŚLI ((>FIGURA) MA WSZYSTKIE KĄTY RÓWNE))
  (TO    ((<FIGURA) JEST PROSTOKĄTEM)))

(REGUŁA NUMER16
  (JEŚLI ((>FIGURA) MA DWIE OSIE SYMETRII))
  (TO    ((<FIGURA) JEST RÓWNOLEGŁOBOKIEM)))

(REGUŁA NUMER17
  (JEŚLI ((>FIGURA) MA DWIE PARY RÓWNYCH KĄTÓW)
         ((<FIGURA) MA DWIE PARY RÓWNYCH BOKÓW))
  (TO    ((<FIGURA) JEST RÓWNOLEGŁOBOKIEM)))

(REGUŁA NUMER18
  (JEŚLI ((>FIGURA) MA DWIE PARY BOKÓW RÓWNOLEGŁYCH))
  (TO    ((<FIGURA) JEST RÓWNOLEGŁOBOKIEM)))

(REGUŁA NUMER19
  (JEŚLI ((>FIGURA) MA JEDNĄ OS SYMETRII)
         ((<FIGURA) MA JEDNĄ PARE RÓWNYCH BOKÓW))
  (TO    ((<FIGURA) JEST TRAPEZEM)))

(REGUŁA NUMER20
  (JEŚLI ((>FIGURA) MA JEDNĄ OS SYMETRII)
         ((<FIGURA) MA DWIE PARY RÓWNYCH KĄTÓW))
  (TO    ((<FIGURA) JEST TRAPEZEM)))

(REGUŁA NUMER21
  (JEŚLI ((>FIGURA) MA DWIE PARY RÓWNYCH KĄTÓW)
         ((<FIGURA) MA JEDNĄ PARE RÓWNYCH BOKÓW))
  (TO    ((<FIGURA) JEST TRAPEZEM)))

(REGUŁA NUMER22
  (JEŚLI ((>FIGURA) MA JEDNA PARE BOKÓW RÓWNOLEGŁYCH))
  (TO    ((<FIGURA) JEST TRAPEZEM)))
```

L I T E R A T U R A

[1] Sussman G. i inni, SCHEME 79-LISP on Chip. Computer, July 1987.

[2] Glaser H. i inni, Principles of Funotional Programming, Prentice Hall, 1984.

[3] Winston P.H. i inni, LISP. II wyd. Addison Wesley PC.

[4] Steele G.L. jr,Sussman G.J., Design of a LISP-Based Microprocessor, Commun ACM, 23, 11 (1980) , 628-645.

6. PROLOG

W początkach lat siedemdziesiątych Robert Kowalski z Uniwersytetu Edynburskiego i Alan Colmerauer z Uniwersytetu Marsylskiego prowadzili prace nad zastosowaniem logiki pierwszego rzędu do specyfiki zadań realizowanych na komputerach w postaci deklaratywnej. Pierwszy z nich zajmował się aspektami teoretycznymi [6] , [7] a drugi również realizacją, która była pierwszą wersją języka Prolog (1972) . Opis języka podał Rousel (1975) .

6.1 Podstawy teoretyczne języka

Podstawy teoretyczne Prologu opierają się na znanych faktach z rachunku predykatów pierwszego rzędu:

- każda formuła logiczna rachunku predykatów może być sprowadzona do semantycznie jej równoważnego skończonego zbioru klauzul (klauzulą nazywamy uniwersalnie kwantyfikowaną dysjunkcję formuł atomowych lub negacji formuł atomowych) ,
- istnieje reguła wywodu zwana rezolucją (wykorzystująca algorytm unifikacji [8]), za pomocą której można wywieść klauzulę "fałsz" wtedy i tylko wtedy, gdy zbiór ten jest sprzeczny, tzn. nie ma interpretacji spełniającej wszystkie klauzule,
- formuła jest implikacją niesprzecznego zbioru formuł wtedy i tylko wtedy, gdy dołączenie jej negacji do tego zbioru daje sprzeczny zbiór formuł.

W konsekwencji powyższego, jeśli dołączenie do niesprzecznego zbioru formuł negacji formuły egzystencjonalnej pozwoli wywieść metodą rezolucji fałsz, wtedy uzyskujemy interpretację (między innymi zmiennych kwantyfikatora) spełniającą formułę podkwantyfikatorową, a gdy tego fałszu nie możemy uzyskać, to w każdej interpretacji (spełniającej wyjściowy zbiór formuł) jest nie spełniona formuła podkwantyfikatorowa, w rezultacie czego jej negacja wynika z tego zbioru formuł.

Istnieje więc problem praktyczny uzyskania drogą wywodu przez rezolucję fałszu lub wykazanie niemożliwości tego wywodu.

W przypadku pełnego rachunku predykatów wielkość drzewa wywodu przekreśla realność tego rozwiązania. Ograniczamy się więc z pozytywnym skutkiem do Hornowskiego podzbioru klauzul przyjmując specjalną strategię w stosowaniu zasady rezolucji.

Klauzule Hornowskie są to klauzule mające najwyżej jedną nie zanegowaną formułę atomową. Dzielą się więc na cztery klasy dające się zapisać, przy użyciu operatora implikacji, w następującej postaci:

- A (x,b...) $\Leftarrow$ True prawa strona jest pustą koniunkcją równoważną True
- A (x,b...) $\Leftarrow$ B_1 (x,...) & B_2 (a,y,...) & ... & Bn (z,...)
- False $\Leftarrow$ C_1 (x,y,...) & ... & Cn (x,...)

  lewa strona jest pustą dysjunkcją równoważną False
- False $\Leftarrow$ True jest to klauzula Hornowska z pustym zbiorem formuł atomowych i pustym zbiorem negacji formuł atomowych równoważna fałszowi .

Powyższe cztery przypadki formuł odpowiadają odpowiednio faktom bazy, regułom wnioskowania bazy, pytaniom (query) oraz pozytywnemu wynikowi stosowania zasady rezolucji.

6.1.1 Zasada rezolucji dla klauzuli Hornowskich

Zasada rezolucji opierająca się na algorytmie unifikacji ma szczególnie prostą postać dla klauzul hornowskich. Ilustruje to poniższy przykład:

np : $\Leftarrow$ A1 (x,f (y)) & A2 (a,g (x)),

A1 (h (z) , f (a)) $\Leftarrow$ B1 (f (z) , c) & ... BN (...)

daje $\Leftarrow$ B1 (f (z) , f (a)) & ... BN (...) & A2 (a,g (h (z)))
przy uwzględnieniu formuły unifikacyjnej [x/h (z) , y/a]

6.1.2 Program w Prologu i jego interpretacja

Programem w Prologu nazywamy zbiór faktów i reguł bazy oraz pytanie. Rezultatem działania programu może być prosta odpowiedź tak lub nie, gdy w pytaniu brak zmiennych, podanie wartościowania zmiennych występujących w pytaniu, które je spełnia, lub odpowiedź, że spełniona jest negacja pytania (tzn., że wszystkie wartościowania zmiennych spełniające formuły bazy nie spełniają pytania).

Uzyskanie tego wyniku jest zadaniem interpretera Prologu. Stosuje on następującą strategię: Wybiera pierwszy cel (atomową formułę) pytania. Szuka w uporządkowanej bazie faktów i reguł pierwszej pozycji wobec której może zostać zastosowana procedura unifikacji. Gdy jest to fakt, to pamięta, który to fakt oraz jaki jest wynik dokonanego wartościowania i przechodzi do następnego celu. Natomiast, gdy jest to reguła, to pamięta, która to reguła oraz wynik dokonanego wartościowania, zastępuje cel zestawem celów z prawej części reguły i przechodzi do analizy pierwszego z nich.

Gdy w przeszukiwaniu bazy dla jakiegoś celu dojdzie on do jej końca i cel nie zostanie zunifikowany, wtedy w dalszej części bazy poszukuje możliwości innej jego unifikacji przy uwzględnieniu kolejnych wartościowań (por. p. 3.3.2.2 Programowanie logiczne).

W ten sposób albo osiągany jest pusty zbiór celu równoważny wywodowi fałszu i wartościowanie będące odpowiedzią na pytanie (możemy teraz uzyskać nastpną odpowiedź ponawiając pytanie w zachowanym stanie użycia bazy), albo odpowiedź negatywną, że brak jest możliwości spełnienia pytania, przez wartościowania zmiennych zgodne z bazą.

Z powyższego wynika, że Prolog jest relacyjnym językiem programowania, w którym możemy uzyskać wszystkie rezultaty dla danych i wszystkie dane prowadzące do zadanego rezultatu oraz wszystkie możliwe pary danych i rezultatów.

Z innego punktu widzenia można patrzeć na Prolog jako uogólnioną relacyjną bazę danych lub bazę wiedzy ze względu na logiczny charakter zawartej informacji. Baza wiedzy jest niesprzeczna (ponieważ zasadą rezolucji nie jesteśmy w stanie osiągnąć jej sprzeczności), ma wyłącznie pozytywną informację, a brak informacji w bazie oznacza negację tej informacji.

W Prologu są także możliwe inne strategie przeszukiwania bazy i kolejności rozważania celów. Można stosować "szeroką" strategię analizowania celu najpierw z pierwszego poziomu, później z drugiego itd., stosowania reguł oraz rozważanie wszelkich alternatyw użycia reguł z bazy jednocześnie. Takie strategie mają teoretyczną przewagę nad wcześniej opisaną, ponieważ zawsze znajdują rozwiązanie, jeżeli ono istnieje. Wymagają one jednak znacznie większych i szybszych pamięci na konwencjonalnych komputerach. Prowadzone są prace nad sterowaniem strategią poszukiwań środkami logicznymi (dwu poziomowa logika) lub środkami gramatycznymi (gramatyki Wijngartena). Wreszcie szuka się rozwiązań w wieloprocesorowości.

Przyjęcie określonej realizacji daje duże korzyści praktyczne. Pozwala wzmocnić środki wyrazu przez wprowadzenie do pytań i prawych stron reguł negacji i dysjunkcji oraz mechanizmów skracających przeglądanie bazy (operacji "cut") i wbudowanych predykatów. Środki te pozwalają stworzyć środowisko programowania dla Prologu dorównujące środowisku języków typu LISP-u. W związku z tym podkreślić należy, że programowanie w "czystym" Prologu jest bardzo trudne.

6.1.3 Środowisko Prologu

Środowisko Prologu tworzy się definiując tzw. wbudowane predykaty, których spełnienie powoduje często uboczne efekty. Są to predykaty wejścia, wyjścia, komunikacji ze zbiorami, definiujące nowe operatory, zwiększające bazę wiedzy.

Często w Prologu jest wprowadzany funktor odpowiadający zapisom listowym:

[ ] - lista pusta, [X | T] - lista z głową X i resztą T.
[A, B | T] - lista z dwoma pierwszymi elementami A,B oraz resztą T.

Uwzględniona jest możliwość jawnego zapisu wartości logicznych;
"true" oznacza predykat zawsze spełniony, "fail" predykat zawsze nie spełniony.
Var (X) oznacza nieinstancjonowanie zmiennej, tzn., że nie jest jej przypisywana aktualna wartość w aktualnym procesie wnioskowania.
Atom (X) oznacza, że X jest atomem.
Integer (X) oznacza, że X jest liczbą całkowitą.
Atomic (X) oznacza, że atom (X) lub integer (X) .

W środowisku Prologu klauzule mogą być traktowane jak termy. Możemy skonstruować term reprezentujący w bazie klauzulę, wstawić ją do bazy lub usunąć z bazy.

Clause (X, Y) oznacza, że X unifikuje się z głową, a Y z ciałem klauzuli.

Call (X) wywołuje cel X i spełnia się wtedy i tylko wtedy, gdy jest spełnione X ,

Asserta (X) wstawia klauzulę X do bazy przed klauzule pasujące do X .

AssertZ (X) wstawia klauzulę X do bazy za klauzule pasujące do X .

Retract (X) usuwa X z bazy.

W Prologu można budować i analizować dowolne termy.

. Functor (T,F,N) - jego spełnienie daje dla termu jego "Functor" F i liczbę argumentów, a dla F i N term F(_1,...) z nie podstawionymi zmiennymi.

Arg (N, T, A) wybiera n-ty argument termu T.

Operacja X -..L /univ/ działa w sposób następujący:

f (a,b) -..X X -.. [f, a, b]

X = [f,a,b] X = f (a, b)

Name (A, L) powoduje, że L staje się listą numerów liter występujących w atomie A.

Repeat modyfikuje nawroty - Jest programowane jak repeat, co odpowiada repeat :- repeat.

Wprowadza się także definicje złożonych celów:

X;Y - infixowy operator ";" oznacza spełnienie, jeśli spełniony X lub spełniony Y.

Call (X) - X musi być instancjonowane przez term interpretowany jako cel.

Not (X) - X musi być instancjonowane przez term interpretowany jako cel. Spełnienie oznacza niespełnienie X.

X = Y jest spełniony, jeśli term X unifikuje się z Y

Wprowadza się także operatory arytmetyczne +, -, / , mod.

X is Y ewaluuje wyrażenie Y, gdy X nie jest instancjonowane i porównuje rezultaty, gdy X jest instancjonowane.

Środowisko zawiera także rozbudowany system predykatów uruchomieniowych:

trace, spy,...itd. oraz predykaty pomocnicze:

random (R, N) N jest liczbą losową między 1 i R,

gensym (H, X) powoduje generacje identyfikatorów np.: gensym (student, X) powoduje, że X = studenti.,

findall (X, G, L.) spełnienie powoduje, że L staje się listą wszystkich instancji X spełniających cel G.

Środowiska zawierają również programy przetwarzające reguły gramatyk bezkontekstowych w program służący komunikacji z użytkownikiem w języku quasi-naturalnym (np. simple w mikro-Prologu) .

W miejsce klasycznego zapisu odwrotnej implikacji Wniosek ⇐ Przesłanka stosowane są w praktyce dwa równoważne W if P oraz W:-P.

6.1.4 Wnioski

Mimo dużego postępu, programowanie logiczne znajduje się jeszcze we wstępnej, eksperymentalnej fazie rozwoju. Widać brak ustabilizowanej składni, świadome błędy w implementacjach wprowadzane w celu przyśpieszenia zbyt wolnego działania programu, brak doświadczenia z dużymi programami, brak szerokiego rozpowszechniania różnych technik programowania. Również wsparcie sprzętowe do realizacji języków programowania logicznego jest jeszcze niewystarczające. Jednak wśród kompetentnych znawców problematyki ugruntowany jest pogląd, że przyszłość należy do tego stylu programowania. W szczególności Prolog, pomyślany początkowo jako język wysokiego poziomu, obecnie jest widziany jako język podstawowy z bezpośrednią realizacją w sprzęcie V generacja komputerów . Zajmie on więc w programowaniu pozycję analogiczną do pozycji lpgiki pierwszego rzędu w matematyce, wykorzystując jej sprawdzone techniki i środki wyrazu. Zrealizowanie tych zamierzeń być może doprowadzi do przełamania kryzysu softwarowego [1] . Otworzą się możliwości nowych zastosowań typu bazy wiedzy, możliwości efektywnego budowania i modyfikowania narzędzi do tworzenia systemów problemowo zorientowanych przez specjalistów nie będących programistami, możliwości efektywnej specyfikacji algorytmów i języków programowania. Nie zostaną oczywiście wyeliminowane inne języki programowania lub inne architektury komputerowe, lecz wyraźniej określi się zakresy ich efektywnych zastosowań.

6.2 Ogólna charakterystyka Turbo-Prologu i jego implementacji

Implementacja Prologu rozpowszechniana przez firmę Borland International Inc. Scotts Valley, CA cieszy się dużym zainteresowaniem i uznaniem wśród użytkowników komputerów personalnych standardu IBM PC [10] .

Wśród wielu, kontrowersyjnych nieraz, poglądów wyrażanych przez użytkowników, za szczególnie znamienne uznać można te, które wskazują na takie podstawowe zalety tej implementacji, jak: wyjątkowo duża szybkość działania, bardzo rozbudowane i dogodne środowisko, relatywnie niska cena.

Bliższej analizie cech tej implementacji poświęcony będzie kolejny rozdział.

6.2.1 Otoczenie Turbo-Prologu

Kompilator Turbo-Prologu wyposażony jest w otoczenie interakcyjnie współdziałające z użytkownikiem. Składa się na nie pięć okien zobrazowanych na ekranie monitora. Górne okno zawiera zebrane w jednej linii główne menu. W jego skład wchodzą podstawowe opcje umożliwiające użytkownikowi realizację takich funkcji, jak edycja, kompilacja i wykonanie programu, ustawienie opcji kompilatora, wykonywanie operacji na zbiorach, ustawianie parametrów charakteryzujących poszczególne okna oraz zakończenie pracy.

Okno informacyjne, "Message", informuje użytkownika o bieżącej aktywności translatora. Są to komunikaty o zbiorach wejściowych i wyjściowych oraz informacje o postępach w przebiegu kompilacji.

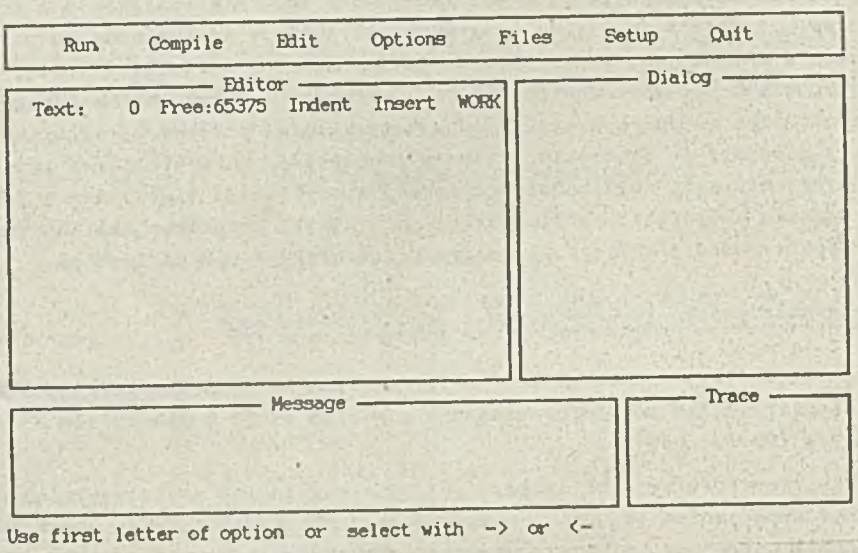
Okno dialogowe, "Dialog", jest aktywne, tzn. dostępne dla użytkownika, od momentu rozpoczęcia wykonywania programu. Za jego pośrednictwem przyjmowane są dane wejściowe podawane przez użytkownika i przedstawiane rezultaty działania programu.

Okno "Trace" jest aktywne wtedy, gdy program lub jego fragment, ma włączoną opcję "trace". Okno to umożliwia śledzenie, krok za krokiem, działania programu napisanego w Prologu. Turbo-Prolog ogranicza się do tego typu działań wspomagających uruchamianie programów i nie jest wyposażony w żadne bardziej zaawansowane środki tego typu.

Wchodzący w skład Turbo-Prologu edytor działa bardzo podobnie do edytora Word Star pracującego w sposób standardowy tzn. w modzie tekstowym. Jest on także bardzo podobny do edytora wchodzącego w skład Turbo-Pascala. Edytor Turbo-Prologu umożliwia korzystanie z edytora pomocniczego, który pozwala na przeglądanie lub edycję innych, dodatkowych zbiorów źródłowych. Wywoływanie edytora pomocniczego jest możliwe podczas pracy w normalnym edytorze Turbo-Prologu, a po zakończeniu korzystania z edytora pomocniczego następuje powrót do

normalnego edytora z dokładnym odtworzeniem stanu pracy edytora i stanu zbioru bieżącego, podlegającego uprzednio edycji, z uwzględnieniem operacji wykonanych za pomocą edytora pomocniczego. Umożliwia to zakończenie edycji zbioru bieżącego niezależnie od operacji przeprowadzanych w związku z wywołaniem edytora pomocniczego.

Edytor główny i edytor pomocniczy wykorzystują odpowiednie okna "Editor" i "Auxiliary Edit".



Rys.9. Zestaw okien otoczenia Turbo-Prologu w wersji wyjściowej

6.2.2 Kompilator

Kompilator Turbo-Prologu umożliwia uzyskanie znajdującego się w pamięci skompilowanego programu wynikowego i wykonywanie tego programu lub generację programu wynikowego w postaci zbioru dostępnego spoza środowiska Turbo-Prologu. Kompilacja przebiegająca w pamięci jest bardzo szybka. Zbliża się ona do szybkości odpowiadającej natychmiastowej, bezpośredniej interpretacji osiąganej przy pracy z interpreterem Prologu.

Jeżeli kompilator wykryje błąd, to uaktywnione jest okno edytora, a w nim wskazywane jest miejsce w programie, w którym wystąpiło odstępstwo od prawidłowej syntaktyki lub inny błąd. Związany z tym błędem komunikat jest wyświetlany w dolnej linii okna. Jeżeli rozmiary komunikatu przekraczają długości linii aktualnie wykorzystywanego okna edytora, to komunikat jest "obcięty". Celowe jest w związku z tym odpowiednie rozszerzenie okna edytora za pomocą opcji "Setup". Lista komunikatów dotyczących błędów zawarta jest w podręczniku Turbo-Prologu.

Wykrycie błędu powoduje także automatyczne przejście do pracy pod kontrolą edytora. W efekcie możliwe jest natychmiastowe poprawienie błędu, a uzyskana w ten sposób nowa wersja programu może być od razu przetestowana w wyniku podjęcia ponownej próby jego kompilacji.

Turbo-Prolog daje dwie możliwości skorzystania ze skompilowanego programu. Wybór jednej z nich wiąże się z rozmiarami programu.

Jeżeli kompilowany program jest stosunkowo mały, to można wygenerować zbiór typu "EXE", który daje się wykonywać pod kontrolą systemu operacyjnego DOS. Generacja tego zbioru jest możliwa przy pozostawaniu w otoczeniu Turbo-Prologu. Osiąga się to generując za pomocą kompilatora zbiór zawierający kod wynikowy programu, który za pomocą programu linkującego (łączącego) jest przekształcany na zbiór typu "EXE". Wykorzystywany jest tutaj program linkujący Microsoft rezydujący jako LINK.EXE.

Następnie można albo wykonywać nowe programy napisane w Prologu, albo powrócić do otoczenia Turbo-Prologu. Ze względów praktycznych najkorzystniejsza jest sekwencja działań, na którą składają się: powrót do otoczenia Turbo-Prologu, wyjście z otoczenia Turbo-Prologu i wykonanie skompilowanego programu. Uzyskuje się w ten sposób najkorzystniejsze warunki pracy programów z rozbudowanymi operacjami rekursji, dla których pamięć częściowo zapełniona rezydującym Turbo-Prologiem może okazać się niewystarczająca.

W przypadku dużych programów należy wykorzystać opcję generacji kodu wynikowego, wyjść z otoczenia Turbo-Prologu i następnie uzyskać program wynikowy linkując go (łącząc) pod kontrolą systemu operacyjnego DOS.

Możliwa jest także kompilacja dająca w wyniku zbiór typu "OBJ", ale ma ona raczej znaczenie pomocnicze. Program jest wtedy wywoływany i wykonywany z otoczenia Turbo-Prologu przy wykorzystywaniu standardowego zestawu okien. Ograniczone są wtedy możliwości użycia ekranu, gdyż jako "tło" pozostaje zawsze system okien Turbo-Prologu. Najwyraźniej zaznacza się też ograniczenie pamięci.

Wśród dziesięciu dyrektyw określających różne warianty rodzaju pracy kompilatora występują między innymi takie, które pozwalają na:

- włączanie programów źródłowych pochodzących z innych zbiorów tekstowych,
- uzyskanie modularności powstającego oprogramowania,
- włączanie funkcji śledzenia wykonywania programu ("trace").

Możliwe jest także włączenie pewnych opcji użytecznych tylko podczas uruchamiania programu, takich np. jak komunikaty kompilatora.

Programy w Turbo-Prologu mogą współpracować z programami skompilowanymi w Pascalu, Języku C, FORTRAN-ie i ASSEMBLER-rze. Pozwala to również na pisanie podprogramów w tym z wymienionych języków, w którym wykonują się one najbardziej efektywnie, a następnie wywoływanie ich w programie napisanym w Turbo-Prologu.

6.2.3 Język

Składnia Turbo-Prologu w znaczący sposób różni się od składni typowego Prologu [3]. Występują takie różnice, jak silniej zaznaczona strukturyzacja programu i wprowadzenie typów danych. Cechy te zwiększają czytelność programów napisanych w Turbo-Prologu w porównaniu z typowym Prologiem.

Można przyjąć, że właśnie te cechy będą pozytywnie przyjęte przez osoby mające praktykę w programowaniu w takich powszechnie używanych językach, jak Pascal i Modula, bo ułatwią im początkowo w istotny sposób korzystanie z Turbo-Prologu.

Program w Turbo-Prologu jest podzielony na sekcje deklaracji, z których część jest opcjonalna.

Pierwsza sekcja deklaracji nosząca nazwę "domains" zawiera deklaracje lokalne deklarowanych przez użytkownika typów danych i list.

Sekcja "global domains" umożliwia dostęp do typów danych z innych modułów programu.

Sekcja "predicates" definiuje nazwy lokalnych predykatów oraz liczbę i typy argumentów dla każdego predykatu.

W sekcji "global predicates" zadeklarowane są predykaty dostępne z innych modułów programowych.

Opcjonalna sekcja "Goal" pozwala na określenie celu logicznego, który jest realizowany przez program. Jeżeli chce się wypróbować działanie programu dla różnych, kolejno zadawanych celów, to sekcję tę opuszcza się, a cele zadaje podczas wykonywania programu. Sekcja ta jest

jednak niezbędna w tych programach, które będą poddawane kompilacji w celu uzyskania zbiorów zawierających ich kod wynikowy.

Sekcja "database" zawiera predykaty, które odznaczają się tym, że w czasie pracy programu można dodawać do nich fakty lub fakty usuwać.

Sekcja "clauses" jest ostatnią sekcją deklaracji wchodzącą w skład programu. Zawiera ona wszystkie fakty i reguły. Wszystkie klauzule zawarte w tej sekcji muszą być zgodne z deklaracjami zawartymi w sekcji "predicates". Turbo-Prolog nie zezwala na definiowanie predykatów mających nazwy takie same, jak predykaty wbudowane, nawet wtedy gdy liczba i typ argumentów są inne, niż użyte w predykacie wbudowanym.

Turbo-Prolog pozwala na wykorzystywanie i obsługę następujących typów danych: znaki ("characters"), liczby całkowite ("integers"), liczby rzeczywiste ("real numbers"), łańcuchy ("strings"), atomy zwane także symbolami ("atoms", "symbols") i zbiory ("files") Charakterystyczne jest, że znak i łańcuch mogą być używane zamiennie, a różnica między nimi występuje tylko na poziomie wewnętrznej obsługi.

Są dwa specjalne typy danych. Typ "regdom" jest używany w predykatach odnoszących się do komunikacji z BIOS-em w celu uzyskania dostępu do wewnętrznych rejestrów procesora 8088, a typ "dbasedom" natomiast - przy wykorzystywaniu dynamicznych baz danych o ograniczonych rozmiarach.

Turbo-Prolog jest także wyposażony w predykaty służące do konwersji typów danych i prostych operacji na łańcuchach. W odróżnieniu od standardowego Prologu nie jest możliwe w Turbo-Prologu używanie symboli lub zmiennych instancjonowanych do oddziaływania na przepływ informacji z wejść i do wyjść. Turbo-Prolog wykorzystuje i obsługuje także struktury listowe.

6.2.4 Predykaty wbudowane

Turbo-Prolog jest bogato wyposażony w predykaty wbudowane. Wiele z tych predykatów przeniesiono bezpośrednio z Turbo-Pascala. Służą one do wykonywania operacji czytania i pisania, obsługują ekran, pozwalają na korzystanie z łańcuchów, a także wykonują konwersję typów. Istnieją także predykaty pozwalające na wykorzystywanie systemu zbiorów, komunikację z systemem operacyjnym oraz tzw. predykaty językowe.

W zakresie funkcji wykorzystywanych do różnego rodzaju procesów obliczeniowych Turbo-Prolog dysponuje funkcjami i predykatami umożliwiającymi m.in. znajdowanie wartości bezwzględnej, obliczanie pierwiastka kwadratowego, generację liczb losowych, obliczanie logarytmów oraz funkcji trygonometrycznych. Turbo-Prolog jest także wyposażony w operatory MOD i DIV i umożliwia operacje na bitach za pomocą predykatów wykonujących operacje AND, OR, NOT, XOR oraz przesuwanie w lewo i w prawo.

Możliwości te są jednak wyraźnie ograniczone w porównaniu z językami przewidzianymi do wykonywania obliczeń.

Predykaty realizujące wczytywanie informacji pozwalają na obsługę różnych typów danych, nawet takich, które nie są używane w standardowym Prologu. Wyjątkowo rozbudowane są predykaty służące do wyprowadzania danych. Są to predykaty wieloargumentowe, pozwalające na wspólną obsługę wielu argumentów, którymi mogą być dane różnych typów. Możliwe jest także formatowanie wyprowadzonych danych.Do oddziaływania na kierunek przepływu informacji służą specjalne predykaty obsługujące system wejścia/wyjścia - "readdevice" i "writedevice".

Predykaty odnoszące się do zbiorów systemowych pozwalają programowi na otworzenie dostępu do zbioru w celu uzyskania operacji czytania, pisania i dołączania danych. Możliwa jest obsługa zarówno zbiorów sekwencyjnych, jak i zbiorów o bezpośrednim dostępie. Wskaźnik pozycji w zbiorach o bezpośrednim dostępie może być interpretowany w odniesieniu do początku lub końca zbioru.

Predykaty, które w Turbo-Prologu obsługują ekran i okna, wykonują funkcje związane z tekstami i grafiką o wysokiej rozdzielczości. Pozwalają one na sterowanie kursorem oraz tworzenie, usuwanie i wybieranie okien oraz usuwanie ich zawartości. Grafika o wysokiej rozdzielczości obejmuje predykaty realizujące elementarne, podstawowe operacje związane z wykonywaniem wykresów.

Realizowana jest także tzw. grafika żółwia ("turtle graphics").

Ze względu na fakt, że implementacja Turbo-Prologu uwzględnia typy danych, wprowadzony został zestaw predykatów umożliwiających uzgodnienie różnych ich typów. Predykaty te pozwalają na uzgadnianie (konwersję) danych takich typów, jak odpowiednio: znaki, łańcuchy znaków, liczby całkowite i rzeczywiste oraz małe i duże litery.

W implementacji Turbo-Prologu położono także duży nacisk i poświęcono wiele uwagi udostępnieniu funkcji związanych z tzw. niskim poziomem programowania. Predykaty obsługujące tę kategorię zadań realizują także takie funkcje, jak: odwoływanie się do BIOS-u, czytanie i ustawianie czasu systemowego oraz daty, wywoływanie edytora Turbo-Prologu, wykonywanie dyrektyw DOS-u w czasie działania programów w Turbo-Prologu, a także bezpośrednią obsługę i dostęp do portów wejścia/wyjścia. Za pośrednictwem właściwych predykatów można także wykonywać operacje POKE i PEEK, których argumentami są bajty i słowa zawarte w pamięci.

6.2.5 Dynamiczna baza danych

Języki sztucznej inteligencji wraz ze swymi aplikacjami, a w szczególności także systemy eksperckie, opierają się w poważnym stopniu na wykorzystywaniu baz danych do zapamiętywania i pobierania zbieranych informacji w celu ich odpowiedniego wykorzystania.

W tym zakresie Turbo-Prolog odbiega w znacznym stopniu od Prologu standardowego. Prolog standardowy ma możliwości włączania różnego rodzaju mechanizmów kontaktowania się z bazami danych podczas działania programu. Użytkownik może dzięki temu wykorzystywać wejścia akceptujące dowolne klauzule, reguły i fakty jako dane w czasie pracy programu. Dodatkowo, ze względu na fakt, że w standardowym Prologu nie są wymagane deklaracje (np. deklaracja typu, deklaracja predykatu), możliwe jest w nim osiągnięcie elastyczności programowania znacznie przewyższającej możliwości Turbo-Prologu, który zawsze wymaga wstępnego umieszczenia w programie odpowiednich deklaracji.

Tak więc próby osiągnięcia w Turbo-Prologu elastyczności w dostępie do dynamicznej bazy danych są krępowane przez kontrolę dotyczącą typu danych oraz predykatów. Predykaty, które są wprowadzane do dynamicznej bazy danych muszą być w odpowiedni sposób wyspecyfikowane w sekcji "database" przed uruchomieniem programu. Ta cecha poważnie ogranicza elastyczność, którą odznacza się standardowy Prolog. W konsekwencji implementacja Turbo-Prologu zatraca w tym zakresie cechy charakterystyczne dla Prologu standardowego ze względu na brak oryginalnego, pozbawionego ograniczeń, mechanizmu realizującego funkcje bazy danych.

Ponadto Turbo-Prolog pozbawiony jest funkcji umożliwiających automatyczne odzyskiwanie obszarów pamięci po ich wykorzystaniu (brak tzw. garbage collection).

W Turbo-Prologu nie została także zaimplementowana pewna liczba predykatów Prologu standardowego. Jest to ważne ze względu na fakt, że wraz z tymi predykatami stracono pewne silne mechanizmy stanowiące cechy standardowego Prologu. Jako przykład można podać porównywanie list, którego możliwości są w Turbo-Prologu bardzo ograniczone.

6.2.6 Programowanie modularne

Turbo-Prolog pozwala na budowanie programów o strukturze modularnej. Niezbędne jest w tym celu użycie odpowiednich dyrektyw kompilatora oraz włączenie zbioru zawierającego sekcje "global domains" i "global predicates". Przygotowuje się także tzw. zbiór biblioteczny dla każdej modularnej konstrukcji programowej, który wylicza nazwy wszystkich modułów składowych. Natomiast kompilator generuje zbiór zawierający tablice symboli dla każdego włączonego modułu.

6.2.7 Uwagi dotyczące eksploatacji

Należy podkreślić, że w przypadku Turbo-Prologu występują dwa zasadniczo różniące się sposoby pracy:

- praca z wykorzystaniem otoczenia Turbo-Prologu,
- wykonywanie programów, których kod wynikowy został wygenerowany przez kompilator, bez wykorzystywania otoczenia Turbo-Prologu.

W pierwszym przypadku użytkownik dysponuje pełnym zestawem udogodnień, w jakie jest wyposażone otoczenie Turbo-Prologu (system okien, system menu, edytory, udogodnienia przy uruchamianiu itp.), ale możliwości tak wykonywanych programów są ograniczone rozmiarami wolnej pamięci wyczerpywanej w dużym stopniu przez rezydujące otoczenie. Uniemożliwia to wykonywanie programów rozbudowujących znacznie stos (programy z głęboką rekursją) i zmniejsza szybkość ich wykonywania.

W drugim przypadku uzyskuje się maksymalną szybkość wykonywania programów, maksymalne możliwości rekursji, rozbudowy stosów, ale traci się możliwości i udogodnienia, jakie daje otoczenie Turbo-Prologu.

Można więc powiedzieć, że pierwszy sposób jest odpowiedni dla pracy związanej z pisaniem i uruchamianiem programów, natomiast drugi odpowiada korzystaniu z uruchomionych i przetestowanych programów z rozbudowaną rekursją i wykorzystuje maksymalnie możliwości Turbo-Prologu w zakresie szybkości i efektywności pracy programów.

6.2.8 Podsumowanie

Turbo-Prolog jest implementacją wyposażoną w wiele użytecznych cech, takich jak: bardzo dobre interakcyjne otoczenie, szybki kompilator, rozbudowane możliwości obsługi ekranu i okien, grafika o wysokiej rozdzielczości, dostęp do programowania w innych językach i na poziomie assemblera oraz funkcje matematyczne.

W Turbo-Prologu wystąpują jednak istotne ograniczenia. Wprowadzenie typów danych powoduje ograniczenie możliwości zaawansowanego przetwarzania symbolicznego. Podobna sytuacja występuje w związku z wykorzystywaniem dynamicznych baz danych. Natomiast brak niektórych ważnych predykatów standardowego Prologu pozbawia użytkownika związanych z nimi możliwości, a przy korzystaniu z programów napisanych w standardowym Prologu konieczne stają się często zasadnicze przeróbki, w skrajnym zaś przypadku taka adaptacja może okazać się zupełnie niemożliwa.

W tej sytuacji, mimo niewątpliwych zalet Turbo-Prologu, można się spotkać z opiniami kwestionującymi jego przydatność w poważnych, wielkoskalowych zastosowaniach.

Podsumowując, można stwierdzić, że Turbo-Prolog w stosunku do innych wersji Prologu na IBM PC (mikroProlog, Prolog V) wyróżnia się następującymi zaletami:

- możliwość kompilowania programów zwiększająca efektywność ich wykonywania,
- środowisko szczególnie dogodne dla programisty ze względu na wbudowany edytor i system menu,
- bogaty wybór predykatów zapewniających obsługę operacji na znakach i łańcuchach,
- możliwość bezpośredniego dostępu do pamięci,
- szerokie możliwości w zakresie definiowania i obsługi okien, grafiki, syntezy dźwięku,
- możliwość dołączania fragmentów programów napisanych w innych językach.

Nie można jednak pominąć pewnych cech Turbo-Prologu, które świadczą na jego niekorzyść. Należą do nich:

- konieczność deklarowania wszystkich definiowanych predykatów i dziedzin ich argumentów będąca nietypowym wyjątkiem wśród innych realizacji Prologu,
- brak niektórych funkcji występujących w innych realizacjach Prologu oraz inne nazwy niektórych predykatów,
- wrażliwość na błędy kompilacji o charakterze błędów "non-fatal" i uboga diagnostyka błędów.

6.3 Przykład

Poniżej przedstawiony zostanie przykład będący ilustracją pewnych metod, które mogą być wykorzystywane w systemach eksperckich realizowanych w Prologu. Przykład napisany został w Turbo-Prologu i wykorzystuje charakterystyczne dla Turbo-Prologu predykaty wbudowane, ułatwiające zapisanie programu oraz zorganizowanie komunikacji z użytkownikiem i bazą danych [10].

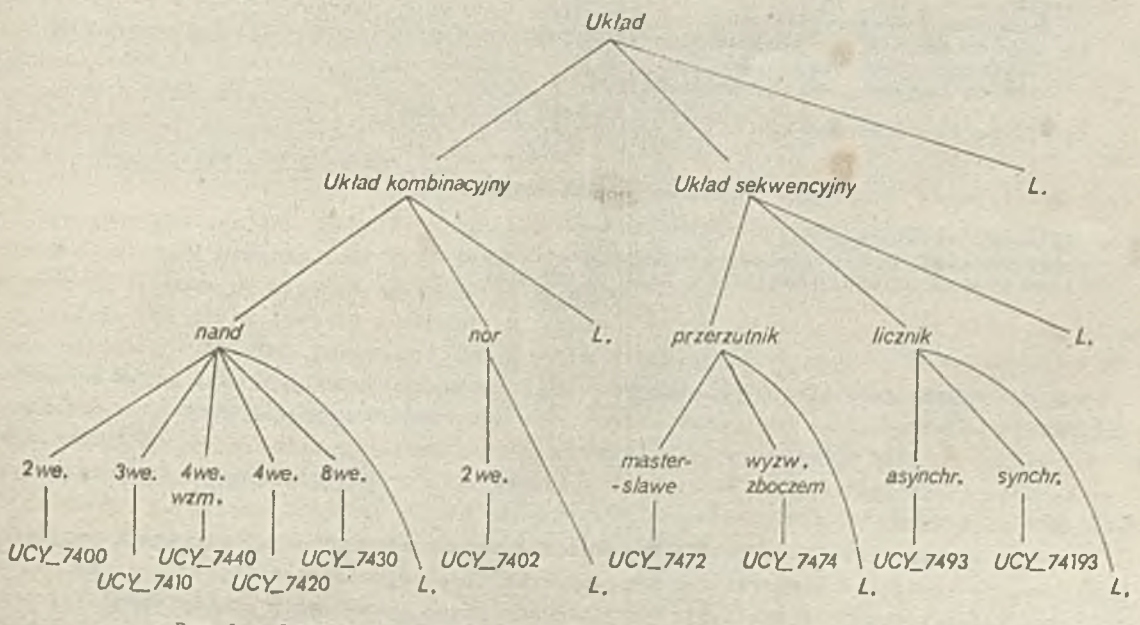
Omawiany przykład spełnia funkcje małego systemu doradczego, w którym wiedza eksperta została zgromadzona w zestawie reguł wprowadzonych do programu jako klauzule. Proces dochodzenia do rozwiązania jest sterowany przez dialog, który użytkownik prowadzi z systemem oraz wspomagany tworzoną przez program dynamiczną bazą danych.

Obiektami, których dotyczy system, są cyfrowe układy scalone TTL, sklasyfikowane według ich podstawowych funkcji logicznych i cech konstrukcyjno-aplikacyjnych istotnych dla konstruktora sprzętu komputerowego. Efektem działania systemu jest podanie konkretnego typu układu scalonego spełniającego wymagania konstruktora. Zestaw informacji wykorzystywanych przez system może być w prosty sposób rozbudowany przez dodanie reguł opisujących nowe układy lub ich klasy. Realne wydaje się rozszerzenie opisywanego systemu tak, aby zawarte w nim reguły obejmowały 100-200 typów cyfrowych układów scalonych, co odpowiada objętości "średniego" katalogu użytkowanego przez konstruktora wykorzystującego podstawowe typy układów scalonych.

6.3.1 Reprezentacja wiedzy w systemie przykładowym

Wiedza o zestawie cyfrowych układów scalonych TTL, która jest wykorzystywana przez system, jest zawarta w zestawie reguł. Merytoryczna zawartość reguł została dostosowana do wymagań i potrzeb użytkownika. Większość reguł ma charakter rozbudowanych definicji obiektów. Definicje te są zapisane w składni Prologu i w związku z tym stanowią klauzule, które mogą być bezpośrednio dołączone do programu. Formę klauzul mają także reguły charakteryzujące pewne klasy obiektów.

Poniżej przedstawiono drzewo reprezentujące klasyfikację typów układów scalonych objętych systemem oraz zestaw reguł opisujących to drzewo i włączonych do programu.



Rys.9. Drzewo klasyfikacji cyfrowych układów scalonych uwzględnianych przez przykładowy system ekspercki; L. - odesłanie do literatury

```
układem_jest(ucy_7400) if
   to_jest(uklad_kombinacyjny) and
       tak(czy,jest_to_uklad_typu_nand) and
       tak(czy,moze_miec_dwa_wejscia).

układem_jest(ucy_7410) if
   to_jest(uklad_kombinacyjny) and
       tak(czy,jest_to_uklad_typu_nand) and
       tak(czy,moze_miec_trzy_wejscia).

układem_jest(ucy_7440) if
   to_jest(uklad_kombinacyjny) and
       tak(czy,jest_to_uklad_typu_nand) and
       tak(czy,moze_miec_cztery_wejscia) and
       tak(czy,powinien_miec_zwiekszona_obciazalnosc).

układem_jest(ucy_7420) if
   to_jest(uklad_kombinacyjny) and
       tak(czy,jest_to_uklad_typu_nand) and
       tak(czy,moze_miec_cztery_wejscia).

układem_jest(ucy_7430) if
   to_jest(uklad_kombinacyjny) and
       tak(czy,jest_to_uklad_typu_nand) and
       tak(czy,moze_miec_osiem_wejsc).

układem_jest(ucy_7402) if
   to_jest(uklad_kombinacyjny) and
       nie(czy,jest_to_uklad_typu_nand) and
       tak(czy,jest_to_uklad_typu_nor) and
       tak(czy,moze_miec_dwa_wejscia).

układem_jest(ucy_7472) if
   to_jest(uklad_sekwencyjny) and
       tak(czy,to_jest_przerzutnik) and
       tak(czy,to_jest_przerzutnik_master_slave).

układem_jest(ucy_7474) if
   to_jest(uklad_sekwencyjny) and
       tak(czy,to_jest_przerzutnik) and
       tak(czy,to_jest_przerzutnik_wyzwalany_zboczem_impulsu).

układem_jest(ucy_7493) if
   to_jest(uklad_sekwencyjny) and
       nie(czy,to_jest_przerzutnik) and
       tak(czy,to_jest_licznik) and
       nie(czy,to_jest_licznik_asynchroniczny).

układem_jest(ucy_74193) if
   to_jest(uklad_sekwencyjny) and
       nie(czy,to_jest_przerzutnik) and
       tak(czy,to_jest_licznik) and
       tak(czy,to_jest_licznik_synchroniczny).

to_jest(uklad_kombinacyjny) if


    tak(czy,dziala_niezaleznie_od_stanow_poprzednich).

to_jest(uklad_sekwencyjny) if
    nie(czy,dziala_niezaleznie_od_stanow_poprzednich) and
    tak(czy,pamieta_conajmniej_jeden_stan_poprzedni).
```

W ten sposób prezentowana wiedza jest wykorzystywana przez program, który uwzględniając informacje podawane przez użytkownika formułuje odpowiedź wskazującą konkretny obiekt.

Używane są przy tym ogólne mechanizmy działania Prologu, które z punktu widzenia wykorzystywania zapisanych reguł, można interpretować jako przeglądanie drzewa reprezentującego wiedzę o obiektach.

Przeglądanie odbywa się metodą "od wniosku do przesłanek" czemu odpowiada przyjmowanie hipotezy, że spełniony jest wniosek zawarty w kolejnej regule stanowiącej zarazem klauzulę Prologu. Wniosek ten jest następnie weryfikowany przez sprawdzenie prawdziwości kolejnych, związanych z nim przesłanek. Proces ten jest kontynuowany aż do momentu wykrycia przesłanki, która okazuje się fałszywa, lub do momentu, gdy badanie wszystkich przesłanek zawartych w regule zostanie zakończone skutkiem pozytywnym.

W pierwszym z tych dwu przypadków uznaje się, że reguła nie jest spełniona i przechodzi się do kolejnej reguły, która jest traktowana w taki sam sposób jak poprzednie.

W drugim przypadku efektem badania przesłanek jest potwierdzenie hipotezy o spełnieniu wniosku, a w konsekwencji, przyjęcie, że obiekt został "dopasowany" do zespołu cech reprezentowanych w regule. Obiekt znaleziony w ten sposób stanowi rozwiązanie.

W bardziej skomplikowanych opisach, w zestawie reguł, mogą pojawić się reguły, które są brane pod uwagę podczas sprawdzania przesłanek innych reguł opisujących obiekty. W takich regułach hipotetyczny wniosek nie jest wskazaniem na konkretny obiekt, a raczej na cechę pewnego zespołu obiektów. W konsekwencji uzyskuje się możliwość wyróżnienia i wskazania za pomocą reguły pewnego podzespołu obiektów.

Tego typu postępowanie pozwala na stworzenie opisu zestawu obiektów. Wykorzystywanie tego opisu podczas działania programu można interpretować jako przeszukiwanie pewnego drzewa opisującego zależności pomiędzy obiektami i ich cechami.

W takim drzewie obiekty występują jako liście, a jego struktura odwzorowuje zestaw i hierarchię zależności zawartych w regułach.

Przeszukiwanie drzewa odpowiada funkcjonowaniu "maszyny wnioskującej", której działanie ma na celu uzyskanie rozwiązania.

Scharakteryzowany powyżej mechanizm działania "maszyny wnioskującej" może być wspomagany komunikacją z dynamiczną bazą danych, która gromadzi uzyskane informacje o faktach. Dzięki temu raz uzyskana informacja o określonym fakcie wykorzystywana jest następnie automatycznie przy stosowaniu kolejnych reguł. Redukuje to dialog z użytkownikiem sprowadzając go do przekazywania wyłącznie istotnie nowych informacji.

Godny uwagi jest fakt, że sposób zapisu reguł nie jest obojętny dla efektywności procesu poszukiwania rozwiązania. I tak na przykład prawdopodobne jest, że reguły podawane przy pierwszej próbie opisu zespołu obiektów będą zawierać nadmiarowy zestaw przesłanek; część z nich może być zbyteczna (np. ze względu na cechy przyjętego systemu klasyfikacji lub ze względu na wykorzystywanie dynamicznej bazy danych). Również kolejność przesłanek w regułach i kolejność reguł mogą wpływać niekorzystnie na proces dochodzenia do rozwiązania. Spowodować to może niepotrzebne rozbudowanie dialogu i zwiększenie liczby kroków, w których dochodzi do komunikowania się z dynamiczną bazą danych.

Można przyjąć, że zastosowanie hierarchicznego podziału obiektów oraz wykluczenie przesłanek nadmiarowych w istotny sposób zwiększa szybkość działania systemu i redukuje dialog do niezbędnych rozmiarów.

6.3.2 Charakterystyka programu

Program realizujący funkcje omawianego, przykładowego systemu eksperckiego jest napisany w Turbo-Prologu i wykorzystuje możliwości, jakie daje translator Turbo-Prologu zainstalowany na typowym komputerze osobistym standardu XT/AT.

Przede wszystkim jest to standardowy system okien i związanych z nimi menu. Po ściągnięciu programu z dyskietki jego tekst ukazuje się w oknie "Edit", tak że rozpoczynająca program sekwencja komentarza staje się widoczna dla użytkownika. Sekwencja ta zawiera informację o sposobie rozpoczęcia pracy. Konwersacja z systemem odbywa się za pośrednictwem okna "Dialog" i może być rejestrowana na drukarce. Formaty komunikatów dostosowane są do maksymalnie rozszerzonego okna "Dialog".

Poniżej załączono przykład takiej konwersacji z systemem przedstawiający konwersację prawidłową oraz reakcje systemu na niepoprawne odpowiedzi użytkownika.

```
start

Odpowiadaj tak/nie.


czy ten uklad dziala_niezaleznie_od_stanow_poprzednich? tak
czy ten uklad jest_to_uklad_typu_nand? tak
czy ten uklad moze_miec_dwa_wejscia? nie
czy ten uklad moze_miec_trzy_wejscia? tak
Twoim ukladem moze byc: ucy_7410


 Jezeli chcesz uzyskac symbol kolejnego ukladu scalonego, to nacisnij
klawisz spacji, a nastepnie napisz START w odpowiedzi na GOAL:

Odpowiadaj tak/nie.              True
start

Odpowiadaj tak/nie.


czy ten uklad dziala_niezaleznie_od_stanow_poprzednich? tak
czy ten uklad jest_to_uklad_typu_nand? nie
czy ten uklad jest_to_uklad_typu_nor? tak
czy ten uklad moze_miec_dwa_wejscia? tak
Twoim ukladem moze byc: ucy_7402


 Jezeli chcesz uzyskac symbol kolejnego ukladu scalonego, to nacisnij
klawisz spacji, a nastepnie napisz START w odpowiedzi na GOAL:

Odpowiadaj tak/nie.              True
start

Odpowiadaj tak/nie.


czy ten uklad dziala_niezaleznie_od_stanow_poprzednich? nie
czy ten uklad pamieta_conajmniej_jeden_stan_poprzedni? tak
czy ten uklad to_jest_przerzutnik? tak
czy ten uklad to_jest_przerzutnik_master_slave? tak
Twoim ukladem moze byc: ucy_7472


 Jezeli chcesz uzyskac symbol kolejnego ukladu scalonego, to nacisnij
klawisz spacji, a nastepnie napisz START w odpowiedzi na GOAL:

Odpowiadaj tak/nie.              True

start

Odpowiadaj tak/nie.


czy ten uklad dziala_niezaleznie_od_stanow_poprzednich? niezaleznie
Niepoprawna odpowiedz
Odpowiadaj tak/nie

czy ten uklad dziala_niezaleznie_od_stanow_poprzednich? tak
czy ten uklad jest_to_uklad_typu_nand? jest
Niepoprawna odpowiedz
Odpowiadaj tak/nie

czy ten uklad jest_to_uklad_typu_nand? tak
czy ten uklad moze_miec_dwa_wejscia? tak
Twoim ukladem moze byc: ucy_7400


 Jezeli chcesz uzyskac symbol kolejnego ukladu scalonego, to nacisnij
klawisz spacji, a nastepnie napisz START w odpowiedzi na GOAL:

Odpowiadaj tak/nie.              True
start

Odpowiadaj tak/nie.


czy ten uklad dziala_niezaleznie_od_stanow_poprzednich? nie
czy ten uklad pamieta_conajmniej_jeden_stan_poprzedni? nie

 System nie posiada informacji o typie ukladu spelniajacego te wymagania.
Uzytkownikowi zaleca sie studiowanie literatury. Moze to byc np pozycja:
W.Sasal, Uklady scalone serii UCA64/UCY74. Parametry i zastosowania.,
WKiL, Warszawa 1985r.

 Jezeli chcesz uzyskac symbol kolejnego ukladu scalonego, to nacisnij
klawisz spacji, a nastepnie napisz START w odpowiedzi na GOAL:

Odpowiadaj tak/nie.              True
```

```
start

Odpowiadaj tak/nie.


czy ten uklad dziala_niezaleznie_od_stanow_poprzednich? tak
czy ten uklad jest_to_uklad_typu_nand? tak
czy ten uklad moze_miec_dwa_wejscia? nie
czy ten uklad moze_miec_trzy_wejscia? nie
czy ten uklad moze_miec_cztery_wejscia? nie
czy ten uklad moze_miec_osiem_wejsc? nie

 System nie posiada informacji o typie ukladu spelniajacego te wymagania.
Uzytkownikowi zaleca sie studiowanie literatury.  Moze to byc np pozycja:
W.Sasal.  Uklady scalone serii UCA64/UCY74.  Parametry  i  zastosowania..
WKiL, Warszawa 1985r.

 Jezeli chcesz uzyskac symbol kolejnego ukladu scalonego, to nacisnij
klawisz spacji,  a  nastepnie napisz  START  w  odpowiedzi  na  GOAL:

Odpowiadaj tak/nie.                 True
```

Poniżej zamieszczono tekst programu wykonującego funkcje omawianego systemu przykładowego "TYP UKLADU SCALONEGO".

```
/*          System przykladowy
          "TYP UKLADU SCALONEGO"
          ======================

    Napisz START  w odpowiedzi na GOAL:
    jezeli chcesz rozpoczac/kontynuowac

                KONSULTACJE
               =============

                                        ver.5.2.
*/
  database
      xtak(symbol,symbol)
      xnie(symbol,symbol)
  predicates
      start
      ukladem_jest(symbol)
      to_jest(symbol)
      tak(symbol,symbol)
      nie(symbol,symbol)
      usun_fakty
      pamietaj(symbol,symbol,symbol)
      pytaj(symbol,symbol)
  clauses
      start:-
       write("\n\nOdpowiadaj tak/nie.    \n\n"),
       ukladem_jest(X),!,
       write("\nTwoim ukladem moze byc: ",X),
       nl,nl,usun_fakty.
      start:-
       write("\n\n System nie posiada informacji "),
       write("o typie ukladu spelniajacego te wymagania. \n"),
       write("Uzytkownikowi zaleca sie studiowanie literatury."),
       write("  Moze to byc np pozycja:\n"),
       write("W.Sasal,  Uklady scalone serii UCA64/UCY74.  "),
       write("Parametry  i  zastosowania.,\nWKiL, Warszawa 1985r.\n"),
       usun_fakty.
      tak(X,Y) :- xtak(X,Y),!.
      tak(X,Y) :- not(xnie(X,Y)),pytaj(X,Y).
      nie(X,Y) :- xnie(X,Y),!.
      nie(X,Y) :- not(xtak(X,Y)),pytaj(X,Y).
      pytaj(X,Y):-
       write("\n",X," ten uklad ",Y,"? "),
       readln(Odpowiedz),pamietaj(X,Y,Odpowiedz).
      pamietaj(X,Y,tak):-asserta(xtak(X,Y)).
      pamietaj(X,Y,nie):-asserta(xnie(X,Y)),
       fail.
      pamietaj(X,Y,Z):-Z<>tak,Z<>nie,
       write("\nNiepoprawna odpowiedz"),
       write("\nOdpowiadaj tak/nie\n"),
       pytaj(X,Y).
      usun_fakty:-retract(xtak(_,_)),fail.
      usun_fakty:-retract(xnie(_,_)),fail.
      usun_fakty:-
       write("\n Jezeli chcesz uzyskac symbol"),
       write(" kolejnego ukladu scalonego, "),
       write("to nacisnij\nklawisz spacji,  a  nastepnie"),
       write(" napisz  START  w  odpowiedzi  na  GOAL:"),
       write("\n\nOdpowiadaj tak/nie.              "),
       readchar(_).
```

Z punktu widzenia struktury programu celowe jest zwrócenie uwagi na sekcję realizującą funkcje dynamicznej bazy danych "database", a w związku z tym wykorzystanie odpowiednich predykatów wbudowanych służących do jej obsługi: "asserta" - wprowadzenie faktu do dynamicznej bazy danych oraz "retract" - zerowanie dynamicznej bazy danych.

Komunikacja z użytkownikiem odbywa się za pomocą predykatów wbudowanych "readchar" i "readln" - wczytywanie odpowiednio znaku i tekstu z urządzenia bieżącego oraz "write" - pisanie na urządzenie bieżące.

Klauzule programu realizują sterowany dialogiem proces przeglądanie zestawu reguł zadanych przez eksperta. W takiej sytuacji zbiór reguł spełnia funkcje zadawanej przez eksperta statycznej bazy danych.

W efekcie tego procesu uzyskuje się odpowiedź - wskazanie typu układu scalonego oraz możliwość kontynuacji polegającą na ponownym rozpoczęciu dialogu prowadzącego do uzyskania kolejnej odpowiedzi.

6.3.3 Możliwości modyfikacji programu przykładowego

Opisany powyżej program przykładowy ma już cechy praktycznej użyteczności (w razie potrzeby może być łatwo uzupełniony nowymi regułami opisującymi dość duży, użyteczny praktycznie, zestaw elementów). W takiej sytuacji celowe może być zastosowanie bardziej zaawansowanych metod programowania w Turbo-Prologu.

Taką dodatkową metodą, której użyteczność zaznaczyłoby się na poziomie programu przykładowego, byłoby wprowadzenie generowanych i obsługiwanych przez program okien zamiast standardowych, jak "Edit", "Dialog" itp.

Dzięki takiemu podejściu można całkowicie "przesłonić" standardowy układ okien i menu wyświetlanych na ekranie monitora, a w konsekwencji usunąć elementy zobrazowania, które nie są związane z działaniem systemu. W obrębie w ten sposób generowanego okna można następnie prowadzić konwersację analogicznie jak dotychczas w oknie "Dialog". Można także w obrębie takiego okna wprowadzić własne menu, co pozwala na uproszczenie i przyspieszenie komunikacji z systemem zastępując fragment dialogu przez wybór jednej z proponowanych opcji. Mogłoby to być np. wybranie istotnej liczby wejść układu określonego typu z listy możliwych opcji, zastępujące wybór wynikający z kilku lub więcej sekwencji pytanie - odpowiedź.

Przy stosowaniu okien generowanych przez program zastosowanie znajdują następujące predykaty wbudowane w Turbo-Prologu: makewindow (...), shiftwidow (...), removewindow (...), clearwindow (...), window_str (...), window_attr (...).

Przy rozbudowie, która wymagałaby znacznego zwiększenia liczby typów układów i zakresu informacji przekazywanych użytkownikowi, celowe mogłoby być wprowadzenie statycznej bazy danych. Jest ona przygotowana i zapisana w postaci stałego zbioru dyskowego wykorzystywanego przez program.

Zastosowanie takiego rozwiązania pozwala na rozszerzenie w dogodny sposób zakresu informacji, które mogą być wykorzystywane podczas pracy systemu i przekazywane jego użytkownikowi.

Z formalnego punktu widzenia statyczna baza danych jest zbiorem klauzul hornowskich, w których prawa strona jest równoważna stałej logicznej "true", a lewa przedstawia wprowadzony do bazy fakt:

```
fakt (a,b, ....) :— true.
```

W praktyce Turbo-Prologu zapis jest uproszczony; pomija się jawne zapisywanie implikacji faktu przez stałą "true" i pozostawia jedynie zapis faktu. W tej sytuacji statyczna baza danych może stanowić po prostu fragment programu (składa się z klauzul), a jej rozmiary są limitowane praktycznie tylko przez pojemność twardego dysku i nie ma przeszkód, by wynosiły nawet 200 Mb.

Takie potraktowanie bazy danych ma jeszcze jedną zaletę: nie odbiega w istotny sposób od tradycyjnego traktowania bazy danych, nie jest więc nowym obciążeniem dla programisty zapoznającego się z Prologiem (Turbo-Prologiem).

L I T E R A T U R A

[1] Clark K.L., An Introduction to Logic Programming., Introductory Readings in Expert Systems., D. Michie ed., Gordon and Breach Science Publishers 1986.

[2] Clark K.L., McCabe F.G, Prolog: A Language for Implementing Expert Systems. Machine Inteligence 10. Ellis Horwood.

[3] Clocksin W.F., Mellish C.S., Programming in Prolog, 2d ed, New York: Springer-Verlag 1984.

[4] Kluzniak F., Spakowicz S., Prolog, WNT, Warszawa 1983.

[5] Kluznak F., Szpakowicz S., Prolog for Programmers. Orlando, FL; Academic Press 1985.

[6] Kowalski R.A., Logic for Problem Solving, North-Holland Elsevier, New York 1979.

[7] Kowalski R.A., i inni, The Semantics of Predicate Logic as a Programming Language. Memo 73, Dept. of Computational Logic School of Artifical Intelligence, University of Edninburgh.

[8] Robinson J.A., A Machine Oriented Logic Based on the Resolution Principle, JACM, vol. 12, 1965.

[9] Shammas N.C., Turbo Prolog. An Easy-to-use Nonstandard Implementation of Prolog for the IBM PC., Byte, September 1986.

[10] Turbo-Prolog Owner's Handbook. Borland, International 1986.

7. SMALLTALK

Pierwsze prace nad językiem Smalltalk rozpoczął Alan Kay w formie Xerox w roku 1971. W następnych latach zostały opracowane kolejne wersje języka: Smalltalk-72, -74,-76,78 i -80. Ostatnio pojawiły się wersje języka Smalltalk opracowane na komputery personalne; dużą popularność zdobył Smalltalk V.

7.1 Podstawowe pojęcia

Język Smalltalk jest językiem obiektywnym. Podstawowymi pojęciami języka są klasa class, obiekt (object) i komunikat (message) . Podstawowe pojęcia (klasa, obiekt) pochodzą z Simuli 67 - pierwszego języka obiektowego. Inspiracji do języków obiektowych można doszukiwać się w programowaniu strukturalnym. W algolu pojawiły się pierwsze elementy strukturalne - blok, procedura. Z pojęciem bloku łączy się pojęcie bloku dynamicznego, związane z realizacją bloku, a więc z rezerwacją pewnego obszaru pamięci komputera na zmienne, wyniki i oczywiście na sam blok. Na ogół po wykonaniu bloku czy też procedury użytkownik nie ma do nich bezpośredniego dostępu; dostęp realizuje się przez ponowne wykonanie bloku czy też procedury. W Simuli i w innych językach obiektywnych występuje twór programowy zwany obiektem, do którego można się odwoływać wielokrotnie bez konieczności wykonania programu zawartego w obiekcie. Wynika to z faktu, że obiekt może być wartością zmiennej. Obiekt można traktować jako egzemplarz utworzony według pewnego wzorca zwanego klasą. Klasa jest konstrukcją syntaktyczną, a więc jest również tworem statycznym. Odwołanie się do klasy powoduje utworzenie obiektu dynamicznego, określonego strukturą klasy, nazywanego obiektem klasy (reprezentowaniem klasy) , w skrócie - obiektem. Dokładniej mówiąc, obiekt zostaje utworzony w wyniku wykonania określonego programu zawartego w definicji klasy. Teoretycznie z klasy można utworzyć (wygenerować) nieskończenie wiele obiektów mających wspólne cechy (atrybuty) określone przez klasę. Pojęcie obiektu i klasy stanowi istotny krok pojęciowy w rozwoju języków programowania. Powyższe pojęcia ułatwiają modularyzację programowania i znaczną dekompozycję rozwiązywanych problemów. Opracowany na U.W.język obiektowy Loglan mimo znacznych zalet nie uzyskał popularności.

Język Smalltalk zyskuje ostatnio coraz większą popularność dzięki bogatemu otoczeniu programowemu i rozbudowanej grafice. Rozbudowane implementacje Smalltalku (zawierające znaczną liczbę zdefiniowanych klas) można traktować jako język deklaratywny, gdyż komunikat określa to, "czego oczekujemy odwołując się do klasy", nie precyzując sposobu uzyskania pożądanej informacji. Natomiast w zaimplementowanych klasach podany jest sposób (algorytm) uzyskania pożądanej informacji. Na przykładzie Simuli omówimy podstawowe pojęcia języków obiektowych, takie jak klasa, obiekt, dziedziczenie.

Podstawowym nowym pojęciem w Simuli w stosunku do innych języków programowania (nieobiektowych) jest pojęcie obiektu. Obiekt jest "reprezentowany" przez samodzielny fragment programu (blok w terminologii algolowskiej) zdefiniowany w deklaracji klasy.

Formalna deklaracja klasy zawiera

class <nazwa klasy (lista parametrów formalnych)>.
<zbiór wartości> ; <specyfikacja parametrów > ;
begin <zbiór deklaracji lokalnych klasy> ;
<lista instrukcji> ;
end

Pozycja <zbiór wartości> w deklaracji klasy jest związana ze sposobem przekazywania parametrów formalnych. W przypadku, gdy sposób przekazywania parametrów jest oczywisty (wynika z typów parametrów) , to ta pozycja w deklaracji klasy nie występuje. Jeżeli w deklaracji klasy nie występuje lista instrukcji, to rola klasy sprowadza się do definicji struktury danych.

Utworzenie obiektu klasy następuje w wyniku wykonania wyrażenia new <nazwa klasy (lista parametrów aktualnych)> ;

Każdy obiekt klasy ma własną kopię zmiennych, a więc ma zdolność reprezentowania stanu dynamicznego. Obiekt klasy tak długo "żyje", jak długo są do niego odwołania.

Tytułem przykładu zdefiniujemy teraz klasę "liczba zespolona"
Deklaracja klasy będzie miała następującą postać

```
class   liczba zespolona  (x,y)  ; real x,y;
    begin
    real r, alfa ;
    r:= sqrt( x..2 + y..2)  ;
    alfa:= arctan  (x,y)  ;
    end
```

Wykonanie wyrażenia

```
new   liczba zespolona  (3,4)  ;
```

powoduje utworzenie obiektu, którym jest liczba zespolona

3 + i 4.

W językach obiektowych można się odwoływać do obiektów. Wymaga to wprowadzenia nowego typu zmiennych. W Simuli wprowadzono nowy typ zmiennych, których wartościami mogą być obiekty danej klasy. Typ ten nazwano typem referencyjnym (reference type).

Deklaracja zmiennej tego typu jest napisem o postaci

ref (<nazwa klasy>) <nazwa zmiennej> ;

Jeżeli A jest nazwą klasy, a X nazwą zmiennej, to mówimy, że zmienna X jest zmienną referencyjną typu A.

Zmienną referencyjną, której wartością jest obiekt danej klasy uważa się za nazwę tego obiektu. Umożliwia to rozróżnienie obiektów tej samej klasy.

Nadawanie wartości zmiennym referencyjnym realizuje się za pomocą instrukcji przypisywania, nazywanej instrukcją przypisania referencyjnego.
Instrukcja ta ma następującą postać:

<nazwa zmiennej> :- <wyrażenie referencyjne>

Wartością wyrażenia referencyjnego jest obiekt danej klasy.

Jeżeli chcemy się odwołać do obiektu, to musimy zadeklarować zmienną referencyjną i nadać wartość tej zmiennej za pomocą instrukcji przypisania referencyjnego. W przypadku odwoływania się do obiektu klasy liczba zespolona, należy napisać następujący ciąg deklaracji:

```
ref  (liczba zespolona)  Z  ;
Z: - new   liczba zespolona  (3, 4)  ;
```

Teraz możemy odwołać się do obiektu klasy liczba zespolona przez zmienną Z. Dostęp do zmiennych x,y,r, alfa uzyskujemy przez następujące wyrażenie Z.X Z.y, Z.r i Z.alfa. Wykonanie ciągu instrukcji

```
print  (Z.x)  ;
print  (Z,y)  ;
print  (Z.r)  ;
print  (Z.alfa)  ;
```

dla obiektu new liczba zespolona (3,4) da nam w wyniku następujący ciąg liczb: 3, 4, 5, 0,927 .

Wracając do definicji klasy należy podkreślić, że w deklaracji klasy jest konieczna specyfikacja parametrów formalnych (określenie typu parametrów). W deklaracji klasy dopuszcza się następujące typy parametrów formalnych:

- proste (real, integer, Boolean, character),

- tekstowe (text),
- referencyjne,
- złożone, tzn. tablice typów prostych, tekstowych i referencyjnych.

Należy zwrócić uwagę, że parametrem formalnym klasy nie może być procedura, etykieta ani też klasa.

Możliwe są tylko dwa sposoby przekazywania parametrów, a mianowicie: przez wartość i przez referencję.

Parametry typów prostych są przekazywane tylko przez wartość, natomiast typów referencyjnych oraz typów złożonych, takich jak tablice typów referencyjnych i tablice typów tekstowych, tylko przez referencję. Wyjątek stanowią parametry typu tekstowego oraz tablice typów prostych, które mogą być przekazywane na oba sposoby.

Jeżeli chcemy parametr typu text czy też typu tablica typów prostych przekazać przez wartość, to należy nazwę tych parametrów poprzedzić słowem value i wstawić w deklaracji klasy między listę parametrów formalnych i zbiór specyfikacji. Niepodanie nazw tych parametrów po słowie value spowoduje przekazanie tych parametrów przez referencję.

W języku Simula-67 istnieje możliwość zdefiniowania nowej klasy B, która będzie rozszerzeniem klasy A o nowe pożądane własności. Tak określoną klasę B nazywamy podklasą klasy A lub klasą prefiksowaną przez klasę A, co zapisujemy w sposób następujący:

A class B ; .....

Nazwanie klasy B podklasą klasy A wynika z tego, że obiekty tej klasy tworzą podzbiór w zbiorze klasy A. Obiekty tego podzbioru mają cechy zdefiniowane zarówno w klasie A, jak i w klasie B. Nazwa klasa prefiksowana bierze się bezpośrednio z postaci deklaracji podklasy.

Prefiksowanie umożliwia definiowanie dowolnie rozbudowanych klas. Wynika to stąd, że klasa, która jest prefiksowana sama musi być z kolei prefiksem innej klasy. W ten sposób można utworzyć całą hierarchię klas. Mechanizm prefiksowania daje możliwość definiowania pojęć złożonych na bazie pojęć prostszych uprzednio zdefiniowanych. Klasa prefiksująca wprowadza podstawowe pojęcia, za pomocą których klasa prefiksowana definiuje nowe pojęcia. Hierarchia klas odpowiada więc hierarchii pojęć, w której pojęcia bardziej złożone są podrzędne względem prostszych.

Rozpatrzmy następujący ciąg deklaracji klas

class A

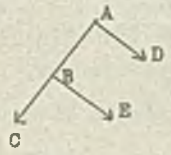
A class B

B class C

A class D

B class E

Klasy te tworzą pewną hierarchię, co graficznie można przedstawić w sposób następujący



Strzałki na tym grafie określają kierunek dziedziczenia własności.

Na prefiksowanie klas w Simuli-67 są nałożone pewne ograniczenia: klasa prefiksująca musi być zadeklarowana na tym samym poziomie, na którym jest użyta jako prefiks.

7.2 Ogólna charakterystyka Smalltalku

Język Smalltalk jest językiem obiektywnym, którego główne koncepcje, jak obiekt i klasa zostały zaczerpnięte z Simuli. W Smalltalku silnie podkreślona jest hierarchia klas. Ogólną charakterystykę Smalltalku podamy opierając się na implementacji Smalltalk-80.

Język Smalltalk-80 jest wyposażony w bogaty zestaw środków programowych , takich jak:

- edytory tekstowe i graficzne,
- narzędzia symulacyjne,
- narzędzia ułatwiające i przyspieszające działania systemu.

W języku można wyróżnić dwie warstwy:

- język programowania,
- język komunikowania.

Nie można jednak przedstawić ostrego podziału; istnieją bowiem wzajemne powiązania. Obiekty są podstawowymi składnikami systemu Smalltalk-80. Obiekty mogą reprezentować np.:

- liczby,
- ciągi znaków,
- kolejki,
- słowniki,
- figury geometryczne,
- katalogi zbiorów,
- edytory tekstowe,
- programy,
- kompilatory,
- procesy obliczeniowe,
- operacje finansowe,
- zobrazowanie graficzne informacji.

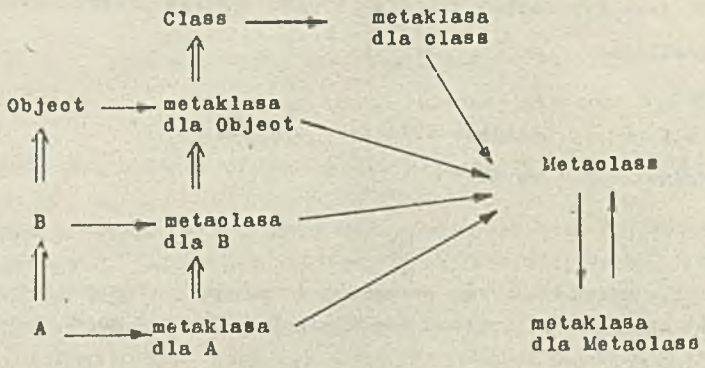
Działanie systemu Smalltalk-80 polega na komunikowaniu się zbioru obiektów między sobą. Obiekt składa się z prywatnej pamięci i zbioru operacji. Każdy obiekt jest obiektem klasy (instance of a class). W przeciwieństwie do Simuli-67 w Smalltalku-80 obiektem klasy może być klasa. Klasa, której reprezentantem obiektem klasy jest klasa nazywamy metaklasą. Jeżeli tworzymy nową klasę, to automatycznie jest tworzona (generowana) dla niej w systemie metaklasa. Klasa opisuje implementację zbioru obiektów mających wspólne cechy. Indywidualny obiekt wygenerowany z klasy nazywa się obiektem (reprezentantem) tej klasy. Klasa opisuje pamięci prywatne obiektów oraz operacje wykonywane przez obiekty. Metaklasy są podobne do klas, gdyż zawierają opis algorytmów używanych przez obiekty do wykonywania określonych operacji. Metaklasy różnią się jednak tym od klasy, że nie mogą być obiektami metaklasy (wygenerowane z metaklasy). Są one obiektami pewnej klasy zwanej Metaklass. Metaklasy nie mają odrębnej nazwy. Dostęp do metaklasy odbywa się przez przesłanie komunikatu do obiektu tej metaklasy o postaci "class". Natomiast klasa Class jest abstrakcyjną nadklasą (abstract superclass) dla wszystkich metaklas. Class opisuje ogólną strukturę klas. Abstrakcyjną nadklasę tworzymy ze względów formalnych w przypadkach, gdy kilka klas ma wspólne cechy, ale żadna z nich nie jest podklasą względem siebie. Abstrakcyjna nadklasa nie generuje obiektów.

W Smalltalku jest silnie podkreślona hierarchia klas w sensie dziedziczenia własności. Każda klasa ma na ogół więcej niż jedną nadklasę. Każdy obiekt jest obiektem klasy.

Hierarchia klas spełnia następujące warunki:

- każda klasa jest podklasą klasy Object (z wyjątkiem klasy Object, która nie ma nadklasy),
- każdy obiekt jest obiektem klasy,
- każda klasa jest obiektem metaklasy,
- każda metaklasa jest podklasą klasy Class,
- każda metaklasa jest obiektem klasy Metaclass.

Niech A będzie podklasą B. Hierarchię klas i metaklas w systemie Smalltalk-80 można przedstawić w sposób jak na rys. 10.

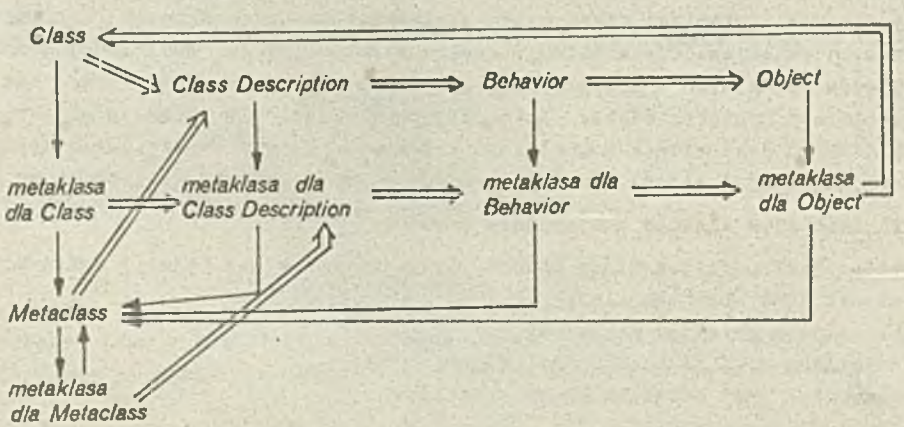


gdzie "—>" oznacza relację bycia obiektem ("A—> metaklasa dla A" oznacza, że klasa A jest obiektem metaklasy dla A)
"⇒" oznacza relację dziedziczenia własności ("A⇒ B" oznacza, że klasa A jest podklasą klasy B) .

Rys. 10. Fragment hierarchii klas i metaklas w Smalltalku-80

Na rys.10 podano tylko fragment hierarchii klas. Jeżeli do systemu wprowadzona jest nowa klasa, to jest ona definiowana jako podklasa klasy w istniejącej w systemie Smalltalk-80 hierarchii klas, na szczycie której jest klasa Object. Dla wprowadzonej klasy automatycznie generowana jest przez system odpowiadająca jej metaklasa. Do definiowania nowych klas są wymagane w systemie dwie klasy Behavior i Class Description. Klasa Behavior definiuje pewne stany wymagane w procesie interpretacji i kompilacji oraz do generowania obiektów. Klasa Class Description jest podklasą klasy Behavior i zawiera dodatkowe informacje potrzebne do pełnej reprezentacji klasy, takie jak reprezentacja zmiennych obiektowych, nazwy klasy i komentarze. Klasa Class Description zawiera dwie podklasy: Class i Metaclass. Klasa Class opisuje reprezentacje zmiennych klasowych oraz operacje na tych zmiennych. Klasa Metaclass steruje procesem generowania metaklasy dla nowo utworzonej klasy.

Wyżej opisane klasy "współdziałając między sobą" ułatwiają opis i generację nowo definiowanych klas. Relacje między tymi klasami przedstawiono na rys. 11.



Rys. 11. Relacje między klasami w Smalltalku-80.
Relacje "—>" i "⇒" mają znaczenie jak poprzednio
(patrz opis rys. 10)

7.2.1 Komunikaty (messages)

Komunikaty są wyrażeniami języka Smalltalk-80. Komunikat określa odbiorcę, podaje selektor i może zawierać argumenty. Wysłanie komunikatu (wartościowanie wyrażenia) jest interpretowane jako polecenie wykonania metody określonej przez selektor w obiekcie będącym odbiorcą komunikatu.

W opisie komunikatu na pierwszym miejscu występuje wyrażenie oznaczające odbiorcę komunikatu (nazwa obiektu), a następnie selektor i ewentualnie argumenty.

Komunikat nie zawierający argumentów jest nazywany komunikatem unarnym, a selektor selektorem unarnym. Wyrażenie

```
theta sin
```

jest komunikatem unarnym. Odbiorcą jest liczba określona przez zmienną theta i selektor sin.

Komunikat binarny zawiera jeden argument i selektor binarny, który jest jednoznakowym lub dwuznakowym symbolem, jak np.

+, -, =, ==, Wyrażenie

```
origin + offset
```

jest komunikatem binarnym o selektorze + i argumencie offset. Inne przykłady komunikatów binarnych:

```
3 + 4
45 + count
total - 1
```

Komunikat kluczowy zawiera jeden lub więcej argumentów oraz selektor złożony z jednego lub kilku słów kluczowych. Każde słowo kluczowe poprzedza jeden z argumentów. Słowo kluczowe składa się z identyfikatora i występującego po nim dwukropka.

Przykładem komunikatu kluczowego jednoargumentowego może być wyrażenie

```
frame moveTo: newlocation
```

gdzie moveTo: jest selektorem, a newlocation argumentem. Przykładem komunikatu dwuargumentowego może być wyrażenie:

```
list at: index put: element
```

w którym selektor jest utworzony ze słów kluczowych at: i put: a argumentami są index i element. Po wysłaniu komunikatu do obiektu zawsze uzyskujemy odpowiedź, która jest również obiektem. Komunikat może być odbiorcą odpowiedzi na komunikat. Odpowiedź na komunikat może być również argumentem dla innego komunikatu. Przykładem komunikatu unarnego opisującego odbiorcę innego komunikatu unarnego może być wyrażenie

```
window frame center
```

Zgodnie z zasadą analizowania komunikatów z lewa na prawo odbiorcą komunikatu o selektorze unarnym frame będzie obiekt window a odbiorcą komunikatu o selektorze unarnym center będzie obiekt uzyskany w wyniku wartościowania wyrażenia window frame.

Jeżeli w wyrażeniu występują różnego typu selektory to najpierw są analizowane komunikaty zawierające selektory unarne następnie binarne a na końcu zawierające selektory kluczowe. Nawiasy również zmieniają kolejność analizowania (wartościowania) wyrażeń. I tak np. wyrażenie 2+3 * 4, zgodnie z przyjętą zasadą wartościowania ma wartość 20, a nie 14. Natomiast wyrażenie 2 + (3 * 4) ma wartość 14 zgodnie z zasadą wartościowania wyrażeń w nawiasach w pierwszej kolejności.

7.2.2 Metody (methods)

Klasa opisuje zbiór obiektów zwanych jej reprezentantami (obiektami klasy — instances). W ramach opisu klasa zawiera zbiór metod określających reakcje reprezentantów na komunikaty. Metoda opisuje ciąg akcji podejmowanych w przypadku uaktywnienia jej przez komunikat. Akcje te polegają na wywołaniu komunikatów, przypisywania wartości zmiennym, wyliczaniu wartości

dla początkowego komunikatu. W opisie metody można wyróżnić następujące ozęści:

- wzór komunikatu (message pattern) ,
- nazwy zmiennych czasowych (roboczych —temporary) ,
- wyrażenia .

Wymienione ozęści metody są rozdzielane pionowymi kreskami(|). Wzór komunikatu składa się z selektora i nazw argumentów. Wyrażenia są rozdzielane kropkami (.) , przy czym ostatnie wyrażenie może być poprzedzone strzałką (↑). Wystąpienie strzałki w metodzie oznacza, że wartością wartościowanego wyrażenia jest wartość wyrażenia poprzedzonego strzałką. W przypadku niewystąpienia strzałki wartością jest odbiorca komunikatu (obiekt, który uaktywnił metodę).

Poszukiwanie metody pasującej do komunikatu (mającej identyczny selektor z komunikatami) zaczyna się od klasy odbiorcy komunikatu i przebiega przez wszystkie jej nadklasy aż do klasy Objeckt (będącej na szczycie hierarchii klas) . Jeżeli metoda nie zostanie znaleziona, to sygnalizowany jest błąd. W przypadku wysłania komunikatu do odbiorcy rozpoczyna się ona od nadklasy klasy zawierającej metodę, w której występuje wysłanie komunikatu.

W wyrażeniach opisujących metody mogą być użyte następujące rodzaje zmiennych:

- zmienne obiektowe odbiornika,
- pseudozmienne self,
- argumenty komunikatu,
- zmienne czasowe (robocze) ,
- zmienne kluczowe,
- zmienne globalne.

Zmienne obiektowe występują w klasie opisującej odbiorcę komunikatu. Pseudozmienna, zmienna self odnosi się do odbiorcy komunikatu. Dostęp do tej zmiennej jest realizowany podobnie jak do innych zmiennych. Zmienna ta różni się od innych tym, że nie można jej nadawać nowej wartości za pomocą instrukcji przypisania.

Zmienne robocze i argumenty mają wspólne cechy. Są one deklarowane w metodzie i istnieją tylko podczas wykonywania metody. Argumenty są automatycznie inicjowane, nie dotyczy to zmiennych roboczych. Można zmieniać wartość zmiennych roboczych za pomocą instrukcji przypisania.

Zmienne klasowe są dostępne dla wszystkich reprezentantów klasy i dla samej klasy. Mimo, że wartości tych zmiennych mogą być zmieniane, to w typowych zastosowaniach są traktowane jako stałe. Są inicjowane w momencie tworzenia klas.

Zmienne globalne są dostępne dla wszystkich obiektów systemu Smalltalk-80. Słownik globalny o nazwie Smalltalk zawiera wszystkie nazwy zmiennych globalnych i ich wartości.

Część metod z różnych klas jest zapisana w języku maszyny docelowej. Są one dostępne bezpośrednio procesorami języka Smalltalk. Metody te nazywane są metodami pierwotnymi. Zawierają one najczęściej używane metody systemowe (np. operacje arytmetyczne, elementarne operacje na obiektach i klasach) , od których może w sposób istotny zależeć efektywność całego systemu. Metody pierwotne są wywoływane przez numery oznaczające je, dokładniej mówiąc są oznaczane przez wyrażenie < primitive: numer metody pierwotnej > .

7.2.3 Klasy

Cechą charakterystyczną Smalltalku jest występująca w formie jawnej hierarchia klas, na szczycie której jest klasa Object. Można powiedzieć, że programowanie w języku Smalltalk sprowadza się do tworzenia lub modyfikowania istniejących klas opisujących obiekty interesujące programistę.

Opis klasy tworzymy według następującego wzorca

nazwa klasy	identyfikator
nazwy zmiennych obiektowych	identyfikatory
metody	metoda . . . metoda

Nazwa klasy rozpoczyna się od dużej litery; nazwy zmiennych rozpoczynają się od małych liter.

Jako przykład podamy klasę liczba zespolona uprzednio zdefiniowaną w Simuli-67 (patrz rozdz. 7.1) . Opis klasy podamy korzystając z podanego wzorca:

nazwa klasy	Liczba zespolona
nazwy zmiennych obiektowych	r , alfa
metody	
rdla: x i: y |x y| ↑ r ← sqrt (x..2 + y..2) alfadla: x i : y |x y| ↑alfa ← arctan (y,x) .	

gdzie " ← " oznacza operację przypisania wartości zmiennej, " ↑ " oznacza, że wartością wyrażenia jest wyrażenie poprzedzone strzałką

Wysłanie komunikatów

Liczba zespolona rdla: 3 i : 4

Liczba zespolona alfadla: 3 i: 4

spowoduje nadanie wartości 5 zmiennej r i wartości 0,927 zmiennej alfa.

Należy podkreślić, że opis klas, metody i komunikaty mogą być opisywane w języku zbliżonym do naturalnego, co było między innymi intencją twórców języka Smalltalk.

Pełny opis klasy wymaga określenia położenia klasy w hierarchii klas, dokładniej mówiąc określenia nadklasy dla klasy definiowanej. Pełny opis klasy należy utworzyć według następującego wzorca:

nazwa klasy	identyfikator
nadklasa	identyfikator
nazwa zmiennych	identyfikatory
nazwa zmiennych klasowych	identyfikatory
metody klasowe	metody . . . metody
metody obiektowe	metody . . . metody

7.2.4 Struktury sterowania i bloki

Struktury sterowania zapewniają odpowiednią kolejność podejmowania akcji. Podstawowe struktury sterowania w Smalltalku-80 zapewniają sekwencyjne wykonywanie (wartościowanie) wyrażeń (np. wartościowanie wyrażeń w metodach). Pozostałe struktury sterowania bazują na obiektach zwanych blokami. Wyrażenie blokowe składa się z ciągu wyrażeń oddzielonych kropkami lub z jednego wyrażenia. Wyrażenia blokowe są oznaczane nawiasami kwadratowymi, np.

```
[index ←——— index + 1]
[index ←——— index + 1.  array at: index put: 0]
```

Wyrażenie blokowe nie jest wartościowane w miejscu jego tekstowego wystąpienia, a dopiero wtedy, gdy do bloku zostanie wysłany specjalny komunikat. Komunikat uaktywniający blok jest unarnym selektorem value (gdy blok nie ma parametrów) albo selektorem kluczowym stanowiącym tylokrotne złożenie klucza value, ile argumentów zawiera blok. Wartością bloku jest wartość ostatniego wyrażenia w bloku, o ile żadne wyrażenie w bloku nie jest poprzedzone strzałką. Jeżeli występuje wyrażenie poprzedzone strzałką, to jego wartościowanie powoduje zakończenie wykonywania bloku i metody, a jego wartość staje się wartością bloku i metody.

Bloki wykorzystuje się głównie do zapisu struktur sterowania. Najprostszymi strukturami sterowania są struktury pierwotne, takie jak:

- wybór warunkowy (conditional selection)
- iteracja warunkowa (conditional repetition)

Wybór warunkowy i iteracja warunkowa są realizowane za pomocą metod w klasie Boolean. Aktywizacja wyboru warunkowego następuje po wysłaniu komunikatów zawierających selektory: ifTrue:, ifFalse:, których argumentami są bloki. Wybór warunkowy jest strukturą podobną do struktury if...then...else występującej w językach algolopodobnych. Na przykład wyrażeniu zapisanym w języku algolopodobnym

```
if   a < b   then
  a:=  a + 1   else
     a: = a - 1 ;
```

odpowiada w Smalltalku-80 następujące wyrażenie

```
a < b
ifTrue:    [a ←a + 1] .
ifFalse:   [a ←a - 1] .
```

Aktywizacja iteracji warunkowej następuje po wysłaniu komunikatów zawierających selektory: whileTrue:, whileFalse, których argumentami są bloki. Wyrażenia bloku wskazane przez selektor są tak długo wartościowane, aż nie zostanie spełniony odpowiedni warunek. Iteracja warunkowa jest podobną strukturą do struktur while i until występujących w językach algolopodobnych. Na przykład

```
while    i < 10   do begin
 x :=    x + a   [i]  ;
 i :=    i + 1
 end;
```

odpowiada w Smalltalku-80 następujące wyrażenie

```
[i < 10]
 whileTrue:    [x ←x +  [a . at:i ] .
                i ←i +  1 ] .
```

7.2.5 Podstawowe klasy Smalltalku-80

Język Smalltalk-80 zapewnia jednolitą składnię ułatwiającą operacje na obiektach, wysyłanie komunikatów i definiowanie klas. Aby ułatwić zachowanie jednolitej składni systemu, Smalltalk-80 zawiera opisy klas, takich jak Object, Class, Message, Compiled Methods, Context oraz podklasy BlockContex i MethodContext. Do koordynacji niezależnych procesów służą klasy Processor Scheduler, Process i Semaphore. Wyróżniony obiekt nil jest jedynym reprezentantem klasy Undefined Object. Powyższe klasy stanowią jądro systemu Smalltalk-80.

System Smalltalk-80 zawiera również klasy służące do opisu podstawowych struktur danych. Klasy te bazują na liczbach i ciągach (zbiorach). Klasa Number służy do opisu wszystkich numerycznych obiektów. Podklasy Float, Fraction, Integer klasy Number opisują specyficzne reprezentacje liczb. Klasa integer zawiera trzy podklasy: Smallinteger, LargePositiveinteger, LargeNegativeinteger. Liczby więc są obiektami (reprezentantami) klas Float, Fraction, Smallinteger, LargePositiveinteger, LargeNegativeinteger.

Klasa Collection służy do opisu struktur danych zawierających ciągi i zbiory obiektów. Podklasami tej klasy są m.in. następujące klasy: Bag, Set, OrderedCollection, LinkedList, MappedCollection. SortedCollection i IndexedCollection, której podklasami są klasy String i Array. Klasy Bag i Set opisują ciągi elementów nieuporządkowanych. W klasie Bag dopuszcza się wielokrotne wystąpienie identycznych elementów a w klasie Set jest niedopuszczalne wystąpienie identycznych elementów. Nazwy pozostałych klas pozwalają z grubsza wyrobić sobie intuicje do opisu jakich obiektów one służą. Pełny wykaz klas systemu Smalltalk-80 jest przedstawiony na rys. 12.

```
Object
   Magnitude
      Character
      Data
      Time
      Number
         Float
         Fraction
         Integer
             LargeNegativeinteger
             LargePositiveinteger
             Smallinteger
   LookupKey
      Association
Link
   Process
Collection
   SequenceableCollection
      LinkedList
          Semaphore
      ArrayedCollection
          Array
          Bitmap
              Display Bitmap
          RunArray
          String
              Symbol
          Text
          ByteArray
      Interval
      OrderedCollection
          SortedCollection
   Bag
   MappedCollection
   Set
      Dictionary
         IdentityDictionary
```

```
Stream
    PositionableStream
        ReadStream
        WriteStream
        ReadWriteStream
            ExternalStream
                FileStream

    Random

File

FileDirectory
FilePage

UndefinedObject
Boolean
    False
    True

ProcessorScheduler
Delay
SharedQueue

Behavior
   ClassDescription
       Class
       MetaClass
Point
Rectangle
BitBit
     CharacterScanner

     Pen

DisplayObject
   DisplayMedium
      Form
        Cursor
        DisplayScreen
   InfiniteForm
   OpaqueForm
   Path

       Arc
         Circle
       Curve
       Line
       LinearFit
       Spline
```

Wcięcia tekstu w przedstawionej hierarchii klas oznaczają relację bycia podklasą.

Rys. 12. Klasy systemu Smalltalk-80

7.3 Uwagi końcowe

Język Smalltalk-80 jest językiem obiektowym. Podstawowymi jego pojęciami są: klasa, obiekt, komunikat, metoda. Składnia języka nie jest złożona, znacznie jednak odbiega oda zasad przyjętych w tradycyjnych językach programowania. Opis wyrażeń języka jest zbliżony do opisu w języku naturalnym, co w zamierzeniu autorów języka miało ułatwić korzystanie ze Smalltalku przez osoby nie mające do czynienia z językami programowania. Takie podejście może prowadzić w niektórych przypadkach do niejasności i nieprecyzyjności określeń oraz sugerować inne znaczenie. Przy braku typów w Smalltalku może to prowadzić do błędów trudno wykrywalnych. Poważnym ułatwieniem w korzystanu ze Smalltalku-80 jest bogate otoczenie programowe, zawierające system okien ułatwiający w trybie interakcyjnym komunikowanie się z systemem, edytory tekstowe i graficzne, narzędzia symulacyjne, narzędzia ułatwiające i przyspieszające działanie systemu. Bogaty zestaw klas powoduje, że programowanie Smalltalku sprowadza się do tworzenia lub modyfikowania istniejących klas interesujących programistę. Język Smalltalk-80 jest reklamowany jako język sztucznej inteligencji. Klasy Smalltalku umożliwiają znacznie dekompozycję problemu oraz ułatwiają utworzenie struktur danych odpowiadających strukturom danych rozważanego problemu. Wyraźnie zaznaczona hierarchia klas ułatwia strukturyzację i dekompozycję problemów. W dostępnej literaturze brak informacji na temat implementacji w Smalltalku systemów eksperckich.

8. PODSUMOWANIE

Zadaniem naszym było zestawienie niezbędnych wiadomości o zasadach budowy systemów eksperckich, a następnie przedstawienie metod implementacji tych systemów stosując takie języki sztucznej inteligencji, które zawierają elementy niezbędne do opisu systemu eksperckiego oraz są interpretowane przez popularne komputery klasy mikromaszyn.

W tym celu przedstawiliśmy:

- pierwowzór systemu eksperckiego,
- architekturę systemu eksperckiego,
- kilka rodzajów systemów eksperckich ze względu na sposoby reprezentacji wiedzy,
- uniwersalne języki sztucznej inteligencji LISP, PROLOG i SMALLTALK,
- konkretne realizacje GC-LISP i Turbo-PROLOG na komputerze wybranej klasy; GC-LISP został wzbogacony o funkcje niezbędne dla procesu wnioskowego w systemie eksperckim.

Wykorzystując powyższe wiadomości podaliśmy kilka przykładów ilustrujących zapisy systemów eksperckich w języku GC-LISP i Turbo-PROLOG.

Chcąc przybliżyć systemy eksperckie do praktycznych zastosowań przyjęliśmy, że komputerem realizującym system eksperckie będzie komputer popularny wśród krajowych profesjonalistów, a zarazem produkowany przez przemysł krajowy. Wybrana klasa komputerów obejmuje maszyny kompatybilne z PC XT/AT oraz ich odpowiedniki produkcji krajowej: MAZOVIA 1016, MAZOVIA 2016, ELWRO-800, ELWRO-801, KRAK-86.

Prezentowana praca zawiera więc wiadomości podstawowe, dotyczące realizacji systemów eksperckich w formie programów lispowych i prologowych na komputerze klasy PC XT/AT.

Osobnym zagadnieniem jest efektywność implementacji systemów według opisanej metodyki, zwłaszcza w odniesieniu do problemów z przemysłu. Ważne to zagadnienie może stać się przedmiotem osobnej pracy.

Niniejsze opracowanie powstało w związku z realizacją w Instytucie Maszyn Matematycznych tematu badawczego finansowanego z Centralnego Programu Badań Podstawowych 2.17 w roku 1987. Koordynatorem programu 2.17 jest Instytut Podstaw Informatyki PAN.

L I T E R A T U R A

[1] Goldberg A., Robson D., Smalltalk-80; the Language and its Implemention, Addison-Wesley, Reading, Mass., 1983.

[2] Goldberg A., Smalltalk-80; the Interactive Programming Environment, Addison-Wesley, Reading, Mass., 1983.

[3] BYTE, August 1981, vol. 6, no. 8(numer specjalny poświęcony systemowi Smalltalk-80)

[4] Oktaba H., Ratajczak W., Simula-67, WNT, Warszawa 1980.

[5] Krępski A., Język Smalltalk-80, IPI PAN, Warszawa 1987.

PODSTAWY IMPLEMENTACJI SYSTEMÓW EKSPERCKICH NA MIKROKOMPUTERZE

Systemy eksperckie zmienią sposób myślenia ludzi o rozwiązywaniu problemów w wielu dziedzinach działalności praktycznej. Zadaniem niniejszej pracy jest omówienie genezy, zasad działania i budowy systemów eksperckich, w stopniu dostatecznym do tworzenia koncepcji takich systemów. Założono, że przejście od konstrukcji konceptualnej do fizycznej będzie się odbywać poprzez zapis systemu za pomocą wybranego języka implementacyjnego realizowanego na mikrokomputerze. Omówienie rozpoczęto od modelu procesów twórczych u człowieka i zasady działania systemu z bazą wiedzy stanowiącego pierwowzór systemu eksperckiego. Zawarcie wiedzy w systemie z bazą wiedzy ma ogromne znaczenie dla działania samego systemu. Opisano więc ważniejsze techniki reprezentacji wiedzy stosowane w systemach eksperckich. Przesłanek do wyróżnienia typu języka implementacyjnego może dostarczyć dyskusja cech języka istotnych dla reprezentacji wiedzy i budowy procedur wnioskowania. Wybrane języki to LISP i PROLOG. Rozdział poświęcony LISP-owi zawiera opis składni, semantyki i środowiska języka; są tu również przykłady systemów, tj. CZWOROKĄTY i PRZYJACIEL oraz dyskusja funkcji systemowych. Rozdział następny dotyczący PROLOG-u ma podobny układ. Rozdział ostatni dostarcza informacji o języku SMALLTALK, który jest językiem typu obiektowego.

MICROCOMPUTER IMPLEMENTATION BASICS OF EXPERT SYSTEMS

Expert systems will alterate the way people think about solving problems in many fields of practical activity. The task of this paper is to discuss genesis, principles of operation and design of expert systems in a degree sufficient to create concepts of such systems. It is assumend that passing from conceptual to physical construction will be done through a system discription porformed by means of a preferential implementation language to be realized by microcomputer. The discussion begins with a model of human creative processors and operation principles of knowledge base system which is an expert system prototype. Involvement of knowledge in a knowledge base system is of utmost importance for the system operation. Thus some techniques of knowledge representation, applied to expert systems, are produced. Preference of implementation language may be evoked discussing these language features which are important for the knowledge representation and desing of inference procedures. LISP and PROLOG are the chosen languages. The chapter on LISP contains descriptions of syntax, semantics and environment of the language. There are also examples of systems i.e., QUADRILATERALS and FRIEND. Besides the chapter contains the discussion of system functions. The next chapter on PROLOG is of similar layout. The last chapter delivers some information on SMALLTALK - an object type language.

ОСНОВЫ РЕАЛИЗАЦИИ ЭКСПЕРТСКИХ СИСТЕМ НА МИКРОКОМПЬЮТЕРЕ

Экспертские системы существенным образом поменяют понятие людей о проблеме решения многих практических проблем. Целью данной работы является рассмотрение происхождения, принципов работы и построения экспертских систем в достаточной степени для создания концепции таких систем.

Понятно, что переход от теоретической конструкции к физической будет производиться через запись разработанной конструкции системы с помощью определенного языка программирования реализованного на микрокомпьютере. Сначала рассматривается модель творческого процесса человека и принцип работы системы с базой знаний, системы являющейся прототипом экспертской системы. Помещение знаний в системе с базой знаний имеет огромное значение для работы самой системы, поэтому большое внимание в данной работе уделяется важнейшим техникам представления знаний, применяемым в экспертских системах. Определение типа языка выработанного для реализации экспертской системы вытекать может из обсуждения свойств языка характерных для представления знаний и построения процедур необходимых для получения правильных выводов. Выбранные языки это LISP и PROLOG. В разделе посвященном языку LISP приведено описание синтаксиса, семантики и среды языка; здесь приведены тоже примеры систем т.е. ЧЕТЫРЕУГОЛЬНИКИ и ДРУГ, а также обсуждение системных функций. Следующий раздел о языком PROLOG имеет похоже содержание. Последний раздел дает информацию о языке SMALLTALK являющимся языком объектного типа.

Cena zł 660.-

Druk IMM zam. 1 /89 nakł. 250 egz.