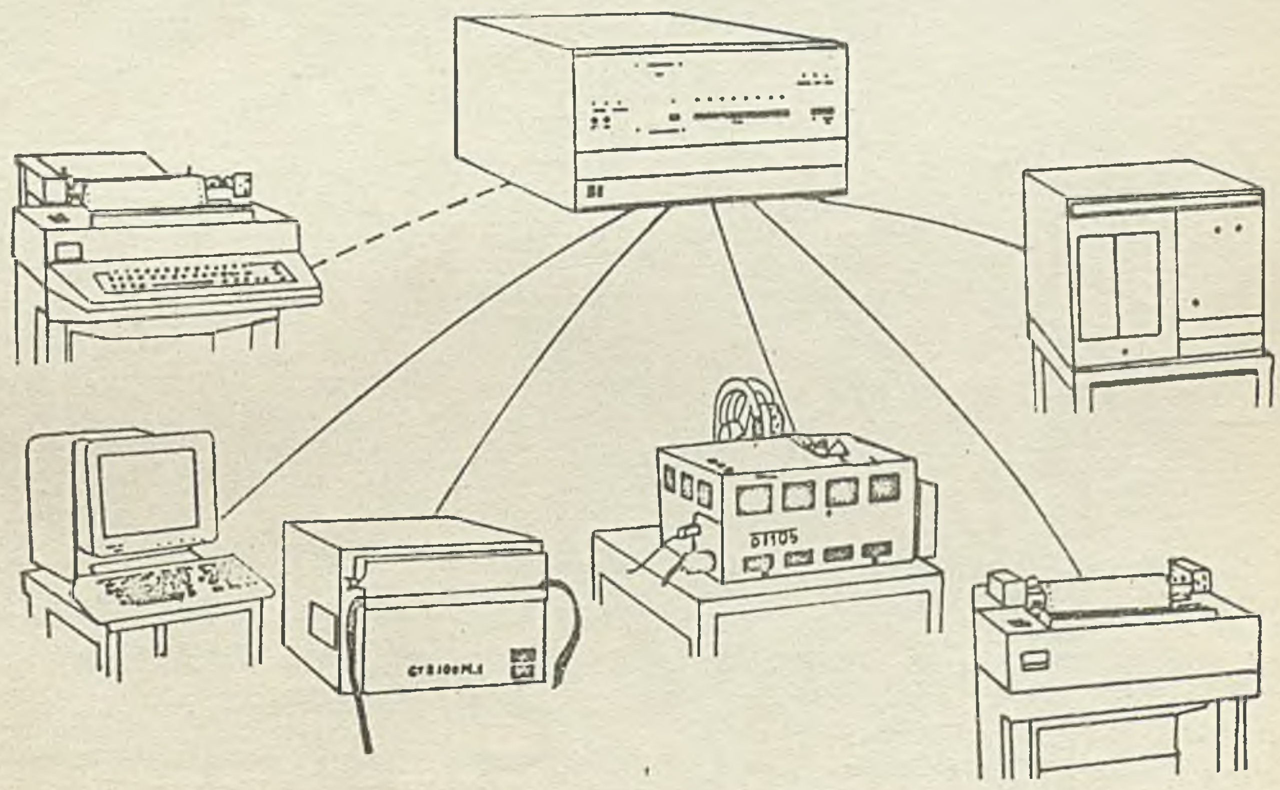
P.3057/81

biuletyn informacyjny

5-6 '81

OBIEKTOWE SYSTEMY KOMPUTEROWE



Zjednoczenie Przemysłu Automatyki i Aparatury Pomiarowej „MERA"
Instytut Maszyn Matematycznych „MERA IMM" Branżowy Ośrodek INTE

Rysunek na okładce: MSWP - Mikroprocesorowy system wspomagania projektowania. Zestaw do projektowania, uruchamiania i testowania sprzętu i oprogramowania urządzeń mikroprocesorowych opracowany w Instytucie Maszyn Matematycznych

Druk IMM zam. 31/82 nakł. 600 egz.

Biuletyn Informacyjny OBIEKTOWE SYSTEMY KOMPUTEROWE

P. 3057/81

Rok XIX Nr 5-6 1981

Spis treści

Poznański Z.: Simula 67 - uniwersalny język programowania. Cz.4 ....s. 3

Olszewski J.: O programowaniu i weryfikacji struktur systemów operacyjnych s.29

Rowicki A.: Pewne metody interpolacyjne i aproksymacyjne stosowane w systemach sterowania numerycznego s.47

Sprawozdania z konferencji s.63

Содержание

Посканьски З.: Simula 67 - универсальный язык программирования. Ч.4 ....с. 3

Ольшевски Е.: Программирование и проверка структур операционных систем с.29

Ровицки А.: Некоторые интерполяционные и апроксимационные методы применяемые в системах вычислительного управления с.47

Отчеты о конференциях...с.63

Contents

Poznański Z.: Simula 67 - the universal programming language Part 4.................p. 3

Olszewski J.: About programming and verification of the operational systems structure p.29

Rowicki A.: Some interpolation and approximation methods utilised in the numerical control systems p.47

Conference reports p.63

D W U M I E S I Ę C Z N I K

Wydaje:

I N S T Y T U T M A S Z Y N M A T E M A T Y C Z N Y C H

Branżowy Ośrodek Informacji Naukowej Technicznej i Ekonomicznej

KOMITET REDAKCYJNY

dr inż. Stanisława BONKOWICZ-SITTAUER, doc.mgr Jan BOROWIEC
mgr Cezary DZIADOSZ /sekretarz redakcji/,
doc.dr inż. Jan ŁYSKANOWSKI,doc.dr hab.inż.Stanisław MAJERSKI,
doc.dr inż. Henryk ORŁOWSKI /redaktor naczelny/,
dr inż. Piotr PERKOWSKI

Opracowanie redakcyjne: mgr Hanna DROZDOWSKA
Opracowanie graficzne: Barbara KOSTRZEWSKA

Adres redakcji: ul.Krzywickiego 34, 02-078 Warszawa
tel.28-37-29 lub 21-84-41 w.244

mgr inż. Zbigniew Poznański
Instytut Technologii Elektronowej

SIMULA 67 - uniwersalny język programowania. Cz.4

W poprzednich częściach przedstawiono dwie systemowe klasy zdefiniowane w języku Simula 67, tj. klasę SIMSET do operacji na strukturach listowych oraz klasę BASICIO do wprowadzania i wyprowadzania danych. Ważną klasą systemową, kierującą Simulę na zagadnienia symulacyjne, jest klasa SIMULATION. Koncepcja tej klasy oparta jest na "wbudowanym" do języka Simula tzw. mechaniźmie quasi-równoległości. W tej części opracowania przedstawiono uproszczoną, w porównaniu z rzeczywistymi możliwościami Simuli, koncepcję mechanizmu quasi-równoległości, jak również pełną definicję klasy SIMULATION. Rozważania zilustrowano kilkoma przykładami.

11. Mechanizm quasi-równoległości

W złożonym procesie można wyróżnić pewne procesy elementarne, które zachodząc jednocześnie, tworzą dany proces. Jeżeli między procesami elementarnymi nie występują żadne interakcje, to w celu zbadania procesu można jego składowe symulować jedną po drugiej. W przeciwnym razie postępowanie takie jest błędne i zmuszeni jesteśmy symulować jednoczesny przebieg procesów elementarnych. Zwykle dysponujemy do tego celu jednym procesorem, w którym w każdej chwili może być wykonywana tylko jedna operacja. W języku Simula zdefiniowano mechanizmy, które "imitują" wspomnianą jednoczesność przez stworzenie tzw. układu quasi-równoległego.

Rozważmy prosty przykład [3] , w którym procesem będzie jednoczesny ruch dwóch kół połączonych masą. Zwróćmy uwagę na to,że ruch ten można rozbić na elementarne obroty każdego z kół o pewien kąt α. A zatem przeniesienie masy realizowane jest obrotem na zmianę kolejnych kół o kąt α. Oczywiście w czasie obrotu jednego z kół, drugie także obraca się, lecz przy tym podejściu do problemu fakt ten nie jest istotny. Napiszmy prosty program opisujący ruch pojedynczego koła:

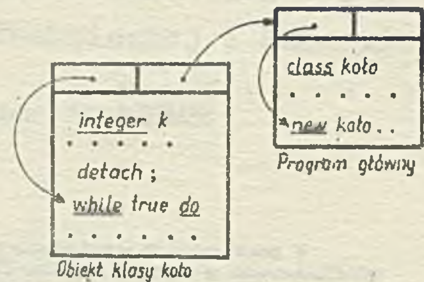
```
begin
  class koło ( k ); integer k;
  while true do
  begin
    obróć koło o kąt α ;
    wstrzymaj koło i aktywuj koło następne;
  end;
end;
```

Po obrocie o kąt α działanie koła jest zawieszone do chwili jego ponownej aktywacji przez następne koło. Należy dodać, że działanie obiektu klasy koło może być wznowione z punktu, w którym zostało zawieszone. Widać stąd, że obiekt klasy koło może znajdować się w stanie aktywnym bądź wstrzymanym.

Z chwilą utworzenia obiektu (new) zostaje on przyłączony do bloku i pozostaje w tym stanie do momentu osiągnięcia symbolu end, sygnalizującego koniec treści obiektu bądź napotkania procedury detach. W pierwszym wypadku obiekt przechodzi w stan zakończony i może pozostawać w pamięci komputera jako struktura danych, w drugim zaś staje się obiektem odłączonym, co czyni go niezależnym składnikiem. W ten sposób można uzyskać układ wieloskładnikowy.

Program główny i odłączone obiekty nazywamy układem quasi-równoległym. Utwórzmy zatem układ quasi-równoległy złożony z obiektów klasy koło:

```
begin
  class koło (k);
  integer  k;
  begin
    detach;
    while true do
    ............
  end;
  while n < N do
  begin
    new koło (n);
    n := n + 1;
  end;
end;
```



rys.62

Zgodnie z tym co powiedziano wyżej, instrukcja new koło powoduje przyłączenie obiektu koło do bloku, w którym został on utworzony. Ponieważ jednak pierwszą instrukcją klasy koło jest detach więc obiekt tej klasy zostaje natychmiast odłączony od bloku stając się niezależnym składnikiem układu quasi-równoległego. Jego SL wskazuje na jednostkę dynamiczną najmniejszego modułu, który zawiera klasę koło, jest nią program główny (rys. 62).

Wprowadźmy następujące oznaczenia:

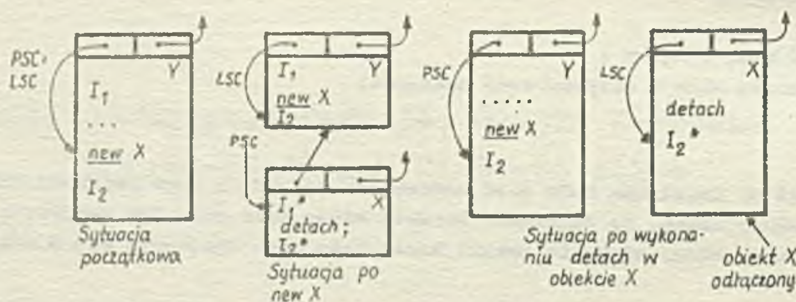
PSC - Program Sequence Control(sterowanie programu), wskaźnik wykonywanej instrukcji programu,

LSC - Local Sequence Control (sterowanie lokalne) , punkt reaktywacji wstrzymanego obiektu będącego w układzie quasi-równoległym.

Wyjaśnimy teraz dokładniej działanie procedury detach.

Wykonanie procedury detach powoduje:

- jeśli X jest referencją do obiektu przyłączonego (pierwsze wywołanie procedury detach w treści obiektu)
  - wstrzymanie wykonywania programu obiektu X,
  - ustawienie punktu reaktywacji LSC obiektu X na instrukcji występującej bezpośrednio po detach,
  - przejście sterowania programu PSC do instrukcji znajdującej się za miejscem, w którym zostało uruchomione działanie obiektu X (rys. 63).



Detach:
- wstrzymanie programu X
- X.LSC := PSC+1
- PSC := Y.LSC

rys. 63 *)

---

*) Wszystkie połączenia statyczne SL jednostek dynamicznych obiektów X, Y przedstawione na rysunku wskazują na jednostki najmniejszych bloków zawierających tekstowo daną klasę.

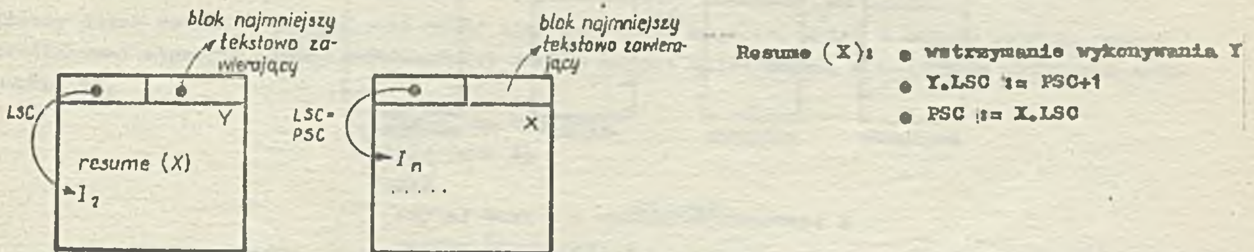
- jeśli X jest obiektem odłączonym (w treści obiektu X procedura detach występuje co najmniej drugi raz)
  - wstrzymanie działania obiektu X,
  - ustawienie punktu reaktywacji LSC obiektu X na instrukcji występującej bezpośrednio po detach,
  - przejście sterowania programu PSC do instrukcji, na którą wskazuje LSC programu głównego, tzn. PSC:=main program.LSC.

W pierwszej wersji programu symulującego ruch dwóch kół połączonych masą, zaistniała konieczność aktywacji obiektu (innego koła). Umożliwia to procedura resume (X), gdzie X jest referencją do obiektu, którego działanie ma być wznowione.

<u>Uwaga:</u> X musi być referencją do obiektu odłączonego.

Procedura resume (X), wywoływana w treści obiektu Y powoduje:

- wstrzymanie wykonywania programu obiektu Y,
- ustawienie punktu reaktywacji LSC obiektu Y na instrukcji występującej bezpośrednio po resume,
- przejście sterowania programu PSC do punktu reaktywacji LSC obiektu Y (rys. 64).



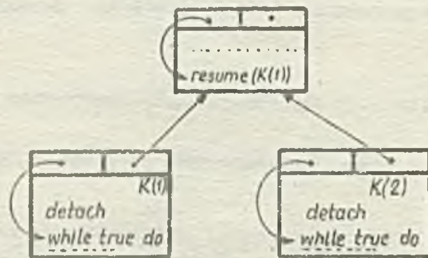
Rys. 64

W procedurze resume (X) nie można odwoływać się do obiektu będącego w stanie zakończonym, ponieważ nie ma on żadnego punktu reaktywacji (chociaż dostęp do atrybutów takiego obiektu istnieje).

Można teraz napisać pełny program symulujący ruch dwóch kół:

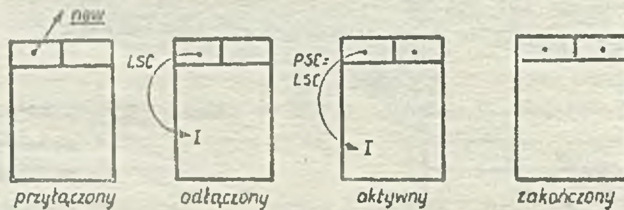
```
begin
  ref (koło) array K(1:2);
  integer i;
  class koło (k); integer k;
  begin
    ref (koło) następne;
    procedure obrót...;
    detach;

    while true do
    begin
      obrót;
      resume (następne);
    end;
  end;§ koło §
  for i:=1 step 1 until 2 do
  K (i):-new koło (i);
  K (1).następne :- K(2);
  K (2).następne :- K(1);
  resume(K(1));
end;
```

Układ quasi-równoległy obiektów występujących w powyższym programie ilustruje rys. 65.



Rys. 65

Podczas realizacji programu obiekt może znajdować się w 4 stanach: przyłączonym (po instrukcji new), odłączonym (po detach), aktywnym (po resume) oraz zakończonym (po końcowym end) (rys. 66).



Rys. 66

Obiekty znajdujące się w stanie odłączonym i aktywnym wraz z programem głównym tworzą układ quasi-równoległy, przy czym, jak wynika z rozważań, obiekt przechodzi w stan odłączony po pierwszym wywołaniu procedury detach w treści klasy, do której należy.

Przedstawiona tu koncepcja układu quasi-równoległego zawiera wiele uproszczeń w porównaniu z rzeczywistymi możliwościami Simuli 67. Obszerniejszy opis tej koncepcji można znaleźć w pracy [1] . Przedstawimy teraz wykorzystanie procedur detach i resume na dwóch przykładach.

• Dany jest zbiór znaków, przy czym ostatnim znakiem jest zadany znak końca zbioru # [3] . Zbiór ten ma strukturę rekordów o długości M znaków każdy. Ostatni rekord tego zbioru uzupełniony jest spacjami. Należy napisać program zmieniający ten zbiór na zbiór z rekordami o długości N każdy. Ostatni rekord także należy uzupełnić spacjami. Zakłada się, że dane są procedury czytaj˽rekord˽wejściowy oraz drukuj˽rekord˽wyjściowy, które odpowiednio wprowadzają bądź wyprowadzają rekord z bufora wejściowego i wyjściowego.

Program rozwiązujący powyższy problem, w pierwszej wersji, zawiera definicje dwóch klas wejście i wyjście. W klasach tych będzie opisany potem odpowiednio algorytm pobierania kolejnych znaków z bufora wejściowego oraz algorytm wypełniania i wyprowadzania zawartości bufora wyjściowego. Ponadto zadeklarowano tu dwie zmienne referencyjne czytnik i drukarka stanowiące nazwy obiektów należących do klas odpowiednio wejście i wyjście. Instrukcje programu głównego obejmują utworzenie obiektów czytnik i drukarka oraz aktywację obiektu czytnik. A zatem w pierwszej wersji program ma następującą postać:

```
begin
   class  wejście....;
   class  wyjście....;
   ref(wejście) czytnik;
   ref(wyjście) drukarka;
   czytnik :- new wejście;
   drukarka :- new wyjście;
   resume (czytnik);
end;
```

Program ten należy traktować jako pierwszy krok w opisie całego systemu przetwarzania rekordów. Ten sposób postępowania jest charakterystyczny dla analitycznej metody programowania (top-down). Należy zauważyć, że obiekty czytnik oraz drukarka będą wchodziły w skład układu quasi-równoległego, a zatem treść klas wejście i wyjście powinna zawierać wywołanie procedury detach, powodującej odłączenie obiektów. Ponadto wspólną częścią treści wspomnianych klas jest deklaracja bufora jako tablicy znaków, który w wypadku klasy wejście traktowany jest jako bufor wejściowy, zaś w wypadku klasy wyjście - jako bufor wyjściowy, a także zmiennej długość określającej długość rekordu. Widać stąd, że celowo jest zdefiniowanie nowej klasy o nazwie np. wewy, która obejmowałaby wyżej wymienione wspólne deklaracje i instrukcje, a jednocześnie stanowiła prefiks dla klas wejście i wyjście. Postać tej klasy jest następująca:

```
class wewy(długość_rekordu) ;
integer długość_rekordu;
begin
  character array bufor (1:długość_rekordu) ;
  detach;
end; $ wewy $
```

Możemy teraz szkicowo przedstawić treść klas wejście i wyjście. I tak w klasie wejście należy zdefiniować algorytm, który odczytywałby znak z bufora wejściowego, a następnie aktywował drukarkę, tj.

```
wewy class wejście;
while true do
begin
  czytaj znak C z bufora wejściowego ;
  resume (drukarka) ;
end;
```

Z kolei algorytm klasy wyjście powinien wprowadzać znak C do bufora wyjściowego. Gdy jest to znak końca zbioru - zawartość bufora wyjściowego jest wyprowadzana na zewnątrz, po uprzednim uzupełnieniu go spacjami. Program zostaje zakończony. Jeżeli nie jest to znak końca zbioru aktywowany jest czytnik, tj.

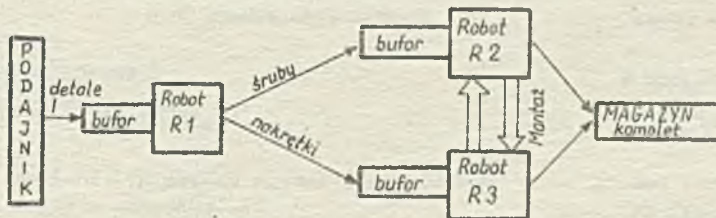
```
wewy class wyjście;
while true do
begin
  czytaj znak C do bufora wyjściowego;
  if not koniec_zbioru then resume (czytnik) else
  begin
        uzupełnij bufor spacjami;
        wyprowadź zawartość bufora;
        detach;
  end;
end;
```

Zwróćmy uwagę na to, że wywołanie procedury detach, w treści klasy wyjście, jest drugim wywołaniem tej procedury (pierwszy raz jest ona wywoływana w prefiksie wewy) , a zatem sterowanie przejdzie tym razem do programu głównego.
Każde utworzenie nowego obiektu klasy wejście lub wyjście (zob. pierwsza wersja programu) powoduje najpierw wykonanie treści klasy prefiksującej, tj. wewy, a zatem wywołanie procedury detach. W ten sposób nowo utworzone obiekty zostają odłączone od bloku i stają się składnikami układu quasi-równoległego.
Instrukcja resume (czytnik)(zob. pierwsza wersja programu) inicjuje proces przetwarzania rekordów. Poniżej przedstawiono pełny tekst programu z licznymi komentarzami:

```
begin
class wewy (długość␣rekordu);
integer długość␣rekordu;
begin
  character array bufor (1:długość␣rekordu);
  detach;

end; § wewy §
wewy class wejście ;
begin
  integer i ;
  procedure czytaj␣rekord␣wejściowy...; § deklaracja tej procedury jest znana z założe-
                                                  nia §
  while true do
  begin
    czytaj␣rekord␣wejściowy;  § wypełnienie bufora wejściowego bufor ciągiem M znaków §
    for i:=1 step 1 until długość␣rekordu do
    begin
      C := bufor (i);   § odczytanie i-tego znaku bufora wejściowego §
      resume (drukarka); § aktywacja drukarki §
    end;
  end;
end; § wejście §
wewy class wyjście;
begin
  integer j;
  procedure  drukuj␣rekord␣wyjściowy...; § deklaracja tej procedury znana jest z założe-
                                                   nia §
  while true do
  begin
    for j:=1 step 1 until długość␣rekordu do
    begin
      bufor(j):= C; § wprowadzenie odczytanego z czytnika znaku do bufora wyjściowego §
      if C / = koniec␣zbioru then resume (czytnik) else
       begin
         for j:=j+1 step 1 until długość␣rekordu do
         bufor (j):= "␣"; § jeśli znak C jest końcem zbioru wtedy pozostała część bufora
                            wyjściowego wypełniana jest spacjami §
         drukuj␣rekord␣wyjściowy; § wyprowadzenie zawartości bufora wyjściowego §
         detach;           § przejście do programu głównego równoważne zakończeniu progra-
                            mu §
       end;
    end;
    drukuj␣rekord␣wyjściowy  § wyprowadzenie bufora wyjściowego po utworzeniu rekordu
                               o zadanej długości i przejście do tworzenia nowego rekor-
                               du §
  end; § while §
end; § wyjście §
ref (wejście) czytnik;
ref (wyjście) drukarka;
character C, koniec␣zbioru;
          § program główny §
```

```
    konieo␣zbioru := inchar ; ‖ wozytanie znaku końca zbioru ‖
    ozytnik :- new wejście (inint) ; ‖ utworzenie obiektu ozytnik oraz wozytanie długości
                                       rekordu wejściowego ‖
    drukarka :- new wyjście (inint); ‖ utworzenie obiektu drukarka oraz wozytanie długości
                                       rekordu wyjściowego ‖
    resume (ozytnik) ;                 ‖ iniojaoja ozytania rekordu ‖
end; ‖ programu ‖
```

Kolejny przykład ilustruje współpracę trzech robotów przemysłowych pracujących w linii montażowej (rys. 67). Podajnik podaje losowo detale (śrubki i nakrętki) robotowi R1, przy czym podanie śruby jest tak samo prawdopodobne jak podanie nakrętki. Robot R1 rozpoznaje czy podany detal jest śrubą czy nakrętką. Jeżeli zidentyfikowana zostaje śruba, wtedy robot R1 przekazuje ją do bufora robota R2, jeżeli nakrętka - do bufora robota R3. Roboty R2 i R3 są robotami montażowymi i ich zadaniem jest kompletowanie par śrub i nakrętek, przy czym robot R2 wkręca śruby w



Rys. 67

nakrętki, zaś robot R3 nakręca nakrętki na śruby. Każdy z tych robotów jest aktywowany z chwilą wypełnienia jego bufora przez robot R1. Działanie robota montażowego polega wtedy na sprawdzeniu zawartości bufora robota sąsiedniego i w wypadku wykrycia w nim detalu do pary - na przesłaniu zawartości swojego bufora do magazynu komplet oraz aktywacji robota sąsiedniego. Jednocześnie robotowi sąsiedniemu przekazywana jest informacja o zestawieniu kompletu. Zadaniem sąsiedniego robota jest wtedy przeniesienie detalu ze swojego bufora do magazynu komplet i wykonanie czynności montażu. Jeżeli jeden z buforów robotów R2 i R3 jest pusty, wtedy uruchamiany jest ponownie podajnik, itd.
W programie zdefiniować należy zatem takie klasy jak detal, podajnik, robot␣1 oraz robot␣montażowy.
Obiekty klasy detal, prefiksowanej klasą LINK, mają jeden atrybut boolowski - śruba. Wartość true tego atrybutu wskazuje na to, że detal jest śrubą, zaś false - nakrętką. Wykorzystanie klasy LINK, a także procedur operujących na strukturach listowych (kolejkach) implikuje konieczność prefiksowania bloku programu klasą SIMSET.
Algorytm działania obiektu klasy robot␣1 sprowadza się do identyfikacji detalu i przesłania go do odpowiedniego robota montażowego. Klasą robot␣montażowy prefiksowano dwie klasy: robot␣2 i robot␣3. Należy zwrócić uwagę na to, że algorytmy działania i struktury danych obiektów należących do klas robot␣2 i robot␣3 różnią się jedynie sposobem realizacji montażu. Wystarczy zatem w klasie robot␣montażowy zdefiniować procedurę wirtualną montaż, zaś w klasach robot␣2 i robot␣3 podać odpowiednie treści tej procedury. Wtedy zgodnie z zasadą działania mechanizmu wirtualności, każde wywołanie procedury montaż dla obiektu należącego do klasy robot␣2 spowoduje realizację czynności wkręcania śruby w nakrętkę, zaś dla obiektu należącego do klasy robot␣3 - czynności nakręcania nakrętki na śrubę.
Pełny tekst programu wraz z licznymi komentarzami ułatwiającymi jego zrozumienie przedstawiono poniżej.

```
Simset  begin
          ref (head) komplet ;
          ref ( robot˽1 ) R1;
          ref(robot˽montażowy ) R2,R3;
          ref ( podajnik) podaj˽detal;
          integer U;
          link class detal (śruba);    boolean śruba;
          class podajnik;
          begin
            detach;                          § odłączenie obiektu §
            while true do
            begin
              new detal (draw(0.5,U))  .into (R1.bufor); § wygenerowanie detalu i wstawienie go
                                                            do bufora robota R1 §
              resume (R1);                   § aktywacja robota R1 §
            end;
          end; § podajnik §
          class robot˽1;
          begin
            ref (head) bufor;             § deklaracja bufora robota R1 §
            ref (detal) S;
            detach;                       § odłączenie obiektu §
            bufor :- new head;            § utworzenie bufora §
            while true do
            begin
              S :- bufor.first;
            if S.śruba then               § identyfikacja detalu §
            begin
              S.into (R2.bufor);          § wstawienie detalu do bufora R2 §
              resume (R2);                § aktywacja robota R2 §
            end   else
            begin
              S.into ( R3.bufor);         § wstawienie detalu do bufora R3 §
              resume (R3);                § aktywacja robota R3 §
            end;
            end; § while §
          end; § robot˽1 §
          class robot˽montażowy;  virtual: procedure montaż;
          begin
            ref (head) bufor;             § deklaracja bufora robota montażowego §
            ref ( robot˽montażowy) sąsiad;    § referencja do sąsiedniego robota montażowego §
            ref (detal)R;
            boolean sygnalizacja˽kompletu;
            detach;                        § odłączenie obiektu §
            bufor :- new head              § utworzenie bufora §
            while true do
            begin
              if sąsiad.sygnalizacja˽kompletu then § sprawdzenie czy sąsiedni robot zasygnalizował
                                                     zestawienie kompletu §
              begin
                bufor.first.into (komplet); § wstawienie detalu do magazynu komplet §
```

```
          montaż;
          sąsiad.sygnalizacja␣kompletu:=false; zgaszenie sygnalizacji kompletu u sąsiada $
          resume (sąsiad);                       aktywacja sąsiada $
        end else
        begin          $ w bloku tym robot sprawdza istnienie kompletu i fakt ten sygnali-
                         zuje sąsiadowi $
          R := bufor.first;
          if R =/= none and not sąsiad.bufor.empty then
          begin
            sygnalizacja␣kompletu:=true;
            R.into (komplet);
            resume (sąsiad);
          end else resume ( podaj␣detal); $ w przypadku nie zestawienia kompletu aktywowany
                                              jest podajnik $
        end;
      end; $ while $
    end; $ robot montażowy $
    robot␣montażowy class robot␣2;
    begin
      procedure montaż;
      wkręcenie śruby w nakrętkę...;
    end;
    robot␣montażowy class robot␣3;
    begin
      procedure montaż;
      nakręcenie nakrętki na śrubę...;
    end;
          $ program główny $
    komplet := new head;        $ utworzenie magazynu komplet $
    podaj␣detal := new podajnik;     $ utworzenie podajnika $
    R1 := new robot␣1;          $ utworzenie robota R1 $
    R2 := new robot␣2;          $ utworzenie robota R2 $
    R3 := new robot␣3;          $ utworzenie robota R3 $
    R2.sąsiad := R3;            $ wyznaczenie sąsiadów dla robotów montażowych $
    R3.sąsiad := R2;
    resume (podaj␣detal);       $ aktywacja podajnika $
  end; $ programu $
```

Klasa simulation

Systemowa klasa Simulation może być traktowana jako pakiet programowy ukierunkowany na problemy symulacyjne. Jest ona prefiksowana klasą Simset, a zatem są w niej dostępne wszystkie atrybuty zdefiniowane w tej ostatniej. Blok prefiksowany klasą Simulation bądź jej podklasą, stanowi program główny układu quasi-równoległego reprezentując zarazem model symulacyjny. Klasa Simulation może wystąpić jako prefiks na dowolnym poziomie zagnieżdżenia tekstowego.

Program symulacyjny składa się ze zbioru procesów (obiektów systemowej klasy process), które podlegają planowanym bądź nieplanowanym działaniom. Gdy proces jest planowany, ma on związany z nim atrybut czasu. Określa on chwilę, w której ma być wznowione działanie obiektu. Atrybut czasu nie oznacza czasu rzeczywistego lecz czas symulowany, którego "upływ" jest powodowany wykonaniem odpowiednich operacji, np. hold. Z chwilą zakończenia fazy aktywności obiektu może on być przeplanowany. Gdy obiekt nie zostaje przeplanowany, może on przejść w stan "zakończony",

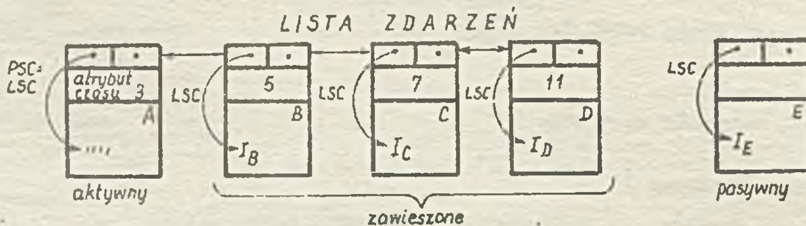
jeśli wykonane są wszystkie instrukcje obiektu bądź w stan pasywny, jeśli chwila wznowienia jego działania nie jest określona. A zatem aktualnie aktywny obiekt, do którego referencję daje systemowa procedura "current" zawsze ma związany z nim atrybut czasu o najmniejszej wartości. Oznacza on jednocześnie aktualny czas symulacji. Wartość czasu symulacji zwiększana jest w sposób dyskretny, w chwili, gdy nowy obiekt zaczyna być aktywny.

Istnieją 3 procedury planowania i przeplanowywania: hold, passivate i activate. Służą one do programowania zmian planu symulacji i sterowania wykonywaniem programów obiektów w ten sposób, aby zawsze aktywny był ten, który nosi nazwę current. Każdy obiekt(tylko klasy process lub jej podklasy)posiada atrybut czasu, wg którego może być za pośrednictwem tzw. zdarzenia procesu umieszczony na liście zdarzeń. Zmiany planu symulacji polegają na tworzeniu zdarzeń procesów, wprowadzaniu ich na listę zdarzeń bądź ich usuwaniu z tej listy. Dla uproszczenia będziemy przyjmowali, na tym etapie rozważań, że na liście zdarzeń znajdują się obiekty klasy process (procesy) chociaż w rzeczywistości są to zdarzenia odpowiadające tym obiektom.

Połączenie obiektów na liście zdarzeń dokonywane jest systemowo, niezależnie od link-własności klasy process (klasa process jest prefiksowana klasą link). A zatem obiekt klasy process może jednocześnie znajdować się na liście zdarzeń oraz być elementem dowolnego zbioru. Mechanizm ten będzie wyjaśniony dalej.

Obiekt klasy process, dalej nazywany krótko procesem, znajdujący się na pierwszym miejscu listy zdarzeń jest zawsze aktywny (current). Pozostałe elementy listy zdarzeń są w stanie zawieszonym. Procesy nie znajdujące się na liście zdarzeń są bądź pasywne bądź zakończone. Powyższe rozważania, jak również działanie procedur planowania ilustrują kolejne poniższe przykłady.

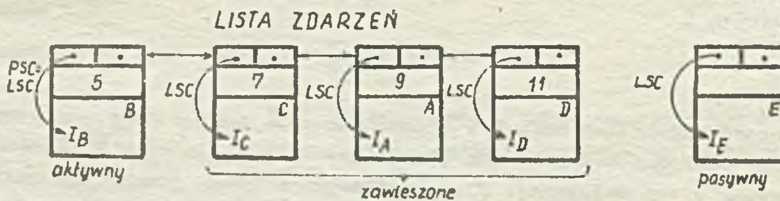
- Dane są 4 obiekty planowane A, B, C, D i jeden pasywny E (konfiguracja początkowa, rys.68)



Rys. 68

Obiekt aktywny A nosi nazwę current. Czas symulacji T związany jest z aktualnie aktywną fazą obiektu current, tzn. T=3.

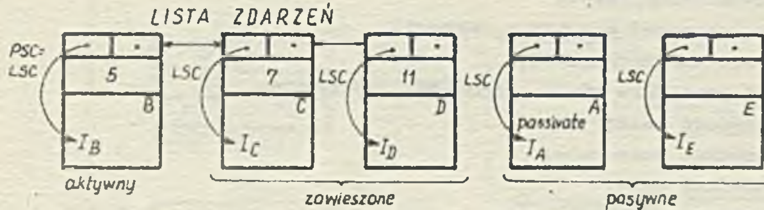
- Efekt działania procedury hold (6) na obiekcie A, z konfiguracji początkowej, ilustruje rys. 69.



Rys. 69

Procedura hold (6) przeplanowuje obiekt current (A) na chwilę time +6, gdzie time jest aktualnym czasem symulacji. Ponieważ atrybut czasu obiektu A wynosi teraz 9, zostaje on zawieszony, jego LSC wskazuje na pierwszą instrukcję po instrukcji hold (por.detach). Aktywny jest teraz obiekt B, który ma najmniejszy atrybut czasu (T=5). Zdarzenie obiektu A zostaje wstawione w odpowiednie miejsce listy zdarzeń.

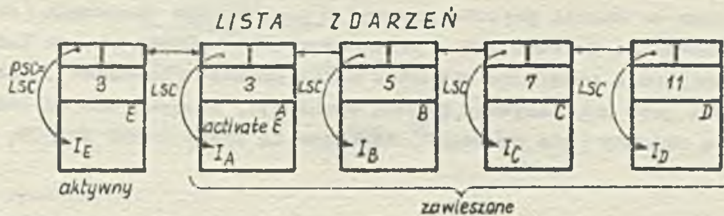
- Efekt działania procedury passivate na obiekcie A, z konfiguracji początkowej, ilustruje rys. 70.



Rys. 70

W wyniku wywołania procedury passivate działanie obiektu A nie zostaje zaplanowane, a więc nie może on pozostawać na liście zdarzeń. Obiekt A przechodzi w stan pasywny z punktem reaktywacji LSC na pierwszej instrukcji po instrukcji passivate (por. detach). Aktywny jest teraz obiekt B, który znajduje się na pierwszym miejscu listy zdarzeń.

- Efekt działania procedury activate E, wywołanej w treści obiektu A, z konfiguracji początkowej, ilustruje rys. 71.



Rys. 71

Procedura activate planuje działanie pasywnego obiektu E. Atrybut czasu obiektu E przyjmuje wartość aktualnego czasu symulacji (T=3). Obiektowi E nadana zostaje także nazwa current. Obiekt A przechodzi w stan zawieszony z LSC na pierwszej instrukcji po instrukcji activate. Należy dodać, że procedura activate daje efekt tylko wtedy, gdy dotyczy obiektów spasywowanych. Aktywacja obiektów znajdujących się na liście zdarzeń dokonuje się przez wykorzystanie procedury reactivate.

Zdefiniujemy teraz klasę Simulation, przy czym obowiązywać będzie zasada przedstawiania dużymi literami identyfikatorów niedostępnych dla programisty.

```
Simset class Simulation;
   begin
     link class EVENT NOTICE  (EVTIME,PROC) ;
     real EVTIME; ref (process) PROC;
     begin
       ref (EVENT NOTICE) procedure suc;
       suc:-if SUC is EVENT NOTICE then SUC else none;
       ref (EVENT NOTICE) procedure pred;
       pred:-PRED;
       ..........
     end;
     link class process;
     begin
       ref (EVENT NOTICE) EVENT;
       ...........
```

```
    end;
    ref (head) SQS;
    ref (EVENT NOTICE) procedure FIRSTEV;
    FIRSTEV:-SQS.first;
    ref (process) procedure current;
    current:-FIRSTEV.PROC;
    real procedure time;  time:=FIRSTEV.EVTIME;
    procedure hold;
    procedure passivate;
    procedure wait;
    procedure cancel;
    procedure activate;
    procedure reactivate;
    process class MAIN PROGRAM; ...
    ref (MAIN PROGRAM )main;
    SQS :- new head;
    main :- new MAIN PROGRAM;
    main.EVENT:-new EVENT NOTICE (0,main);
    main.EVENT.into (SQS);
  end; $ Simulation $
```

Klasa Simulation wprowadza do Simuli pojęcia procesu i planowanego zdarzenia (EVENT NOTICE) wraz z mechanizmami planowania. Zmienna SQS (sequencing set) odnosi się do kolejki, która steruje przebiegiem wykonywania programów obiektów klasy process (procesów). Jest to jednocześnie lista zdarzeń. Często jest ona nazywana planem symulacji. Stanowi ona oś czasu symulacji. Elementami zbioru SQS są obiekty (nie procesy!) należące do klasy EVENT NOTICE, zwane zdarzeniami procesów.

- Klasa EVENT NOTICE

Klasa EVENT NOTICE ma 2 atrybuty: PROC- kwalifikowany klasą process i EVTIME. Atrybut PROC jest nazwą procesu, zwanego procesem zdarzenia, a EVTIME określa chwilę tego zdarzenia. Zdarzenia (obiekty klasy EVENT NOTICE) występują na SQS w kolejności wzrastania atrybutu EVTIME */. A zatem obiekt klasy EVENT NOTICE odnosi się przez atrybut PROC do procesu i reprezentuje zdarzenie, które jest następną aktywną fazą tego procesu z punktem reaktywacji w chwili EVTIME. Obiekt klasy EVENT NOTICE będący pierwszym elementem SQS odnosi się do aktualnie aktywnego obiektu klasy process. Jego atrybut EVTIME określa aktualny czas symulacji.

Wynika stąd, że na SQS procesy reprezentowane są przez swoje zdarzenia. Same obiekty klasy process nie znajdują się więc na SQS. Dlatego można je umieszczać w dowolnych, zdefiniowanych przez programistę kolejkach (obiektach klasy head). Rys. 72 ilustruje listę zdarzeń SQS wraz ze znajdującymi się na niej zdarzeniami procesów.

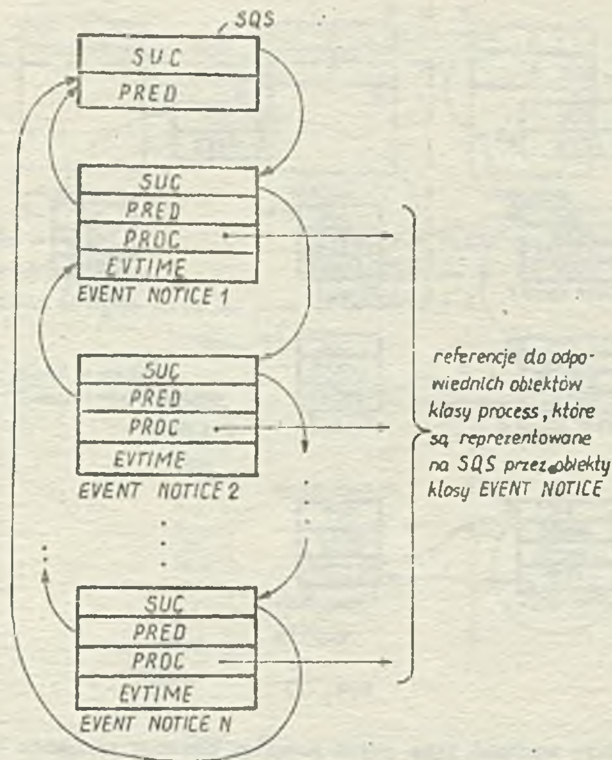
- Klasa process

Obiekty, których zdarzenia mogą być planowane w symulacji muszą należeć do klasy process lub do jej podklasy. Podamy teraz definicję klasy process:

```
link class process;
begin
  ref (EVENT NOTICE) EVENT;
```

---

*/ Kolejka SQS jest kolejką priorytetową (względem czasu).



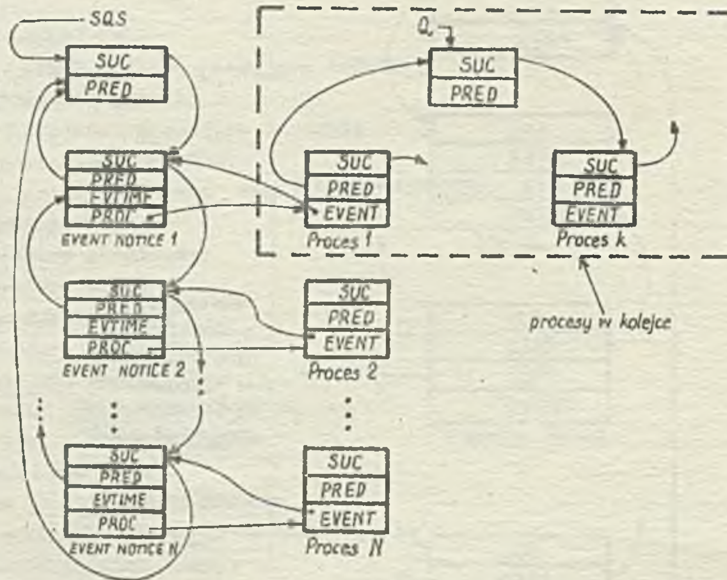
Rys. 72

```
boolean TERMINATED;
boolean procedure terminated;
terminated := TERMINATED;
boolean procedure idle;   idle:= EVENT == none;
real procedure evtime;
if idle then  BŁAD  else evtime:=EVENT.EVTIME;
ref (process) procedure nextev;
nextev := if idle then none else if EVENT.suc==none then none
          else EVENT.suc.PROC;
detach;
inner;
TERMINATED := true;
passivate;
end;
```

Obiekt klasy process ma wszystkie atrybuty klasy link, może więc być elementem zbioru. Atrybut EVENT procesu wskazuje na obiekt klasy EVENT NOTICE, który reprezentuje dany proces na liście zdarzeń SQS (rys. 73) . Rysunek ten ilustruje także możliwość umieszczania procesów w kolejkach (zbiorach).
Łączenie elementów listy zdarzeń SQS jest niezależne od link-własności klasy process. A zatem proces może znajdować się w dwóch zbiorach: jako element dowolnego zbioru oraz jako element SQS, poprzez swoje zdarzenie.

Omówimy teraz kolejno działanie procedur zdefiniowanych w klasie process:

- procedura idle daje wartość true jeśli proces nie jest reprezentowany na SQS, tzn. jest w stanie pasywnym lub zakończonym,
- procedura evtime wyznacza chwilę zdarzenia procesu,



Rys. 73

- procedura terminated daje wartość true jeśli program procesu osiągnie końcowy end.

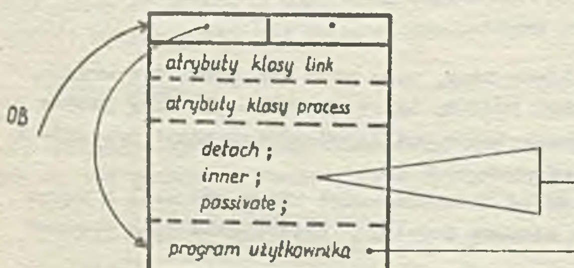
  Uwaga: Nie można wznowić zakończonego procesu.

- procedura nextev wyznacza referencję procesu, którego zdarzenie na liście zdarzeń występuje bezpośrednio po zdarzeniu danego procesu.

Rozważmy następujący przykład:

```
process class obiekt;
ref(obiekt)OB;
OB :- new obiekt;
```

Wygenerowanie obiektu klasy process lub jej podklasy powoduje jego odłączenie od bloku, ponieważ pierwszą instrukcją treści klasy process jest systemowe wywołanie procedury detach. Obiekt przechodzi w stan odłączony. Jego LSC ustawia się na pierwszej instrukcji po detach, którą jest instrukcja w programie użytkownika (działanie inner). W przykładzie będzie to instrukcja OB :- new obiekt.
Ilustruje to rys. 74 .

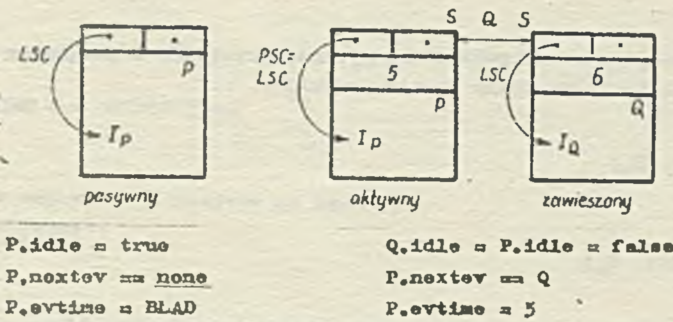


Rys. 74

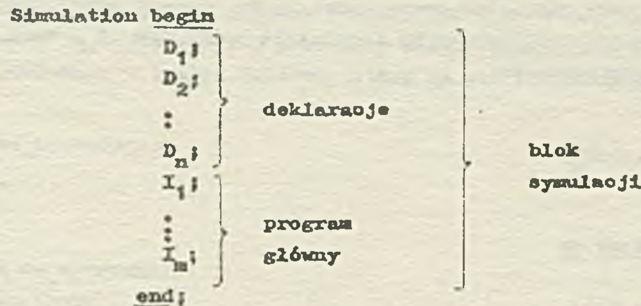
Kolejny przykład ilustruje działanie procedur idle, nextev i evtime (rys. 75) , w zależności od stanu w jakim znajduje się dany obiekt P i Q.

- Klasa MAIN PROGRAM

W poprzednim punkcie wspomniano, że w skład układu quasi-równoległego wchodzi, poza składnikami odłączonymi, program główny. Ponieważ podczas symulacji w układzie quasi-równoległym znajdują się procesy, wskazane jest, aby procesem był także blok symulacji, prefiksowany klasą Simulation i zawierający program główny (rys.76).



Rys. 75



Rys. 76

Nie można tego uczynić bezpośrednio, gdyż bloki nie mają referencji. W tym celu w klasie Simulation zdefiniowano niedostępną dla programisty klasę MAIN PROGRAM o następującej treści:

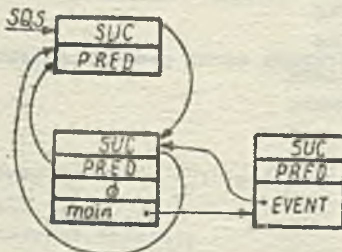
```
process class MAIN PROGRAM;
while true do    detach;
```

Uwaga: Instrukcje programu głównego są wykonywane po wywołaniu co najmniej drugi raz procedury detach, w dowolnym obiekcie układu quasi-równoległego (zob. działanie detach w wypadku obiektów odłączonych).

Działanie obiektu klasy MAIN PROGRAM "symulującego" program główny przedstawione będzie w następnym punkcie, ponieważ ściśle wiąże się ono z działaniem całego bloku symulacji.

- Działanie bloku symulacji

Wejście do bloku symulacji powoduje, zgodnie z algorytmem klasy Simulation, automatyczne wygenerowanie obiektu klasy head, o nazwie SQS, który reprezentuje listę zdarzeń, procesu main (main jest referencją do obiektu klasy MAIN PROGRAM), jego zdarzenia o chwili 0 oraz wstawienie tego ostatniego do listy zdarzeń. Następnie rozpoczyna się wykonywanie programu obiektu main. Tworzenie obiektu klasy MAIN PROGRAM ilustruje rys.77.



Rys. 77

Po instrukcji new MAIN PROGRAM (zob. opis ogólny klasy Simulation) obiekt main zostaje odłączony i pozostaje niezależnym składnikiem (pierwsze działanie detach). W każdej chwili, gdy obiekt main otrzymuje nazwę current, wchodzi on do bloku symulacji, a zatem wykonuje się program główny bloku symulacji. Jest to zgodne z działaniem procedury detach, gdyż wywołanie jej w wypadku obiektu odłączonego

powoduje przejście do wykonywania instrukcji wskazywanej przez LSC w programie głównym (punkt "Mechanizm quasi-równoległości"). Rozważmy proste przykłady.

```
• Simulation begin
          D1; . . . , Dn;
          I1;
          .....
          hold (1000); In;
          end;
```

W przykładzie tym działanie procesu main reprezentującego program główny zostało przeplanowane przez wywołanie procedury hold (1000). Może to oznaczać upływ czasu, po którym należy zakończyć symulację. Za 1000 jednostek czasu proces main otrzyma nazwę current i wykonują się instrukcje programu głównego występujące po instrukcji hold, tzn. I_n. Mogą to być np. instrukcje opracowania danych statystycznych.

```
• Simulation begin
          D1; .....,Dn;
          I1;
          ......
          while true do
          begin
            histo (.);
            hold (T);
          end;
          end;
```

Jeżeli chcemy rejestrować stan wszystkich lub wybranych obiektów układu quasi-równoległego, co zadany krok czasowy T, wygodnie jest wracać do programu głównego i tam np. za pomocą procedury histo zbierać wartości interesujących nas zmiennych losowych. W powyższym przykładzie obiekt main będzie aktywny (current) co T jednostek czasu.

• Procedury klasy Simulation

W klasie Simulation zdefiniowano liczne procedury ułatwiające budowę modeli symulacyjnych. Omówimy kolejno wszystkie z nich.

- ref (EVENT NOTICE) procedure FIRSTEV;
  FIRSTEV :- SQS.first;
  Procedura FIRSTEV, niedostępna dla programisty, wyznacza referencję do pierwszego zdarzenia procesu na liście zdarzeń SQS.

- ref (process) procedure current;
  current :- FIRSTEV.PROC;

  Procedura current daje referencję do obiektu klasy process lub jej podklasy, znajdującego się na pierwszym miejscu listy zdarzeń (aktywnego) , np. if current in X then goto A;

- real procedure time; time := FIRSTEV.EVTIME;
  Procedura time wyznacza aktualny czas symulacji (wartość atrybutu czasu procesu aktywnego), np. if time > 1000 then koniec symulacji;

- procedure hold (T); real T;

```
  begin
    inspect FIRSTEV do
    begin
      if T > 0 then EVTIME := EVTIME + T;
      if suc =/= none then
```

```
  begin
    if suo.EVTIME < = EVTIME then
    begin
      out;
      < wstaw w odpowiednie miejsce na SQS > ;
      resume (current);
    end;
  end;
 end;
end;
```

Procedura hold (T) przeplanowuje proces aktywny (obiekt current) w ten sposób, że zostaje on wznowiony w chwili time+T. Po wywołaniu tej procedury obiekt przechodzi w stan zawieszony. Procedura hold musi być wywoływana w treści obiektu, który ma być przeplanowany i odnosi się tylko do niego, np.

```
process class system␣operacyjny;
begin D1;...; Dn;
  while true do
  begin
    wybierz␣program␣do␣wykonania;
    hold (czas␣jednostkowy);
  end;
end;
```

- ```
  procedure passivate;
  begin
    inspect current do
    begin
      EVENT.out;
      EVENT :- none;
    end;
    if SQS.empty then BŁAD else resume (current);
  end;
  ```

Procedura passivate usuwa zdarzenie obiektu current z listy zdarzeń SQS. Obiekt przechodzi w stan pasywny. Nazwa current zostaje przypisana obiektowi, którego zdarzenie znajduje się na początku listy zdarzeń SQS. Obiekt ten rozpoczyna fazę aktywności (resume (current)). Podobnie do procedury hold, procedura passivate musi być wywoływana w treści obiektu, który ma być pasywowany, np.

```
process class samochód;
begin
  while paliwo do
  begin
    przejedź △ s ;
    hold (10);
  end;
  passivate;
end;
```

- ```
  procedure wait (S);  ref(head)S;
  begin
    current.into(S);
    passivate;
  end;
  ```

- Procedura wait umieszcza obiekt klasy process w zbiorze (kolejce) S i pasywuje go. Podobnie do dwóch poprzednich, procedura wait musi być wywoływana w treści obiektu, który ma być przesłany do zbioru S i spasywowany, np.

```
process class samochód;
if not paliwo then wait (stacja_benzynowa);
```

- procedure cancel (X); ref (process) X;

```
if X == current then passivate else
inspect X do
if EVENT =/= none then
begin
  EVENT.out;
  EVENT :- none;
end;
```

Procedura cancel (P) usuwa zdarzenie procesu P z listy zdarzeń SQS. Jeżeli proces P jest aktywny lub zawieszony to zostaje on spasywowany. Jeśli P jest referencją do obiektu spasywowanego lub zakończonego to procedura ta nie daje żadnego efektu. O ile procedurą passivate można spasowywać tylko obiekt, w treści którego jest ona wywołana, to procedura cancel (P) pasywuje dowolny obiekt P klasy process i może być wywołana z dowolnego miejsca w bloku symulacji (programu głównego bądź dowolnego obiektu).

Uwaga: passivate = cancel (current).

- procedure activate ;

Procedury aktywacji dotyczą procesów nie mających swoich zdarzeń na liście zdarzeń SQS. Wyróżnia się następujące instrukcje z wykorzystaniem activate:

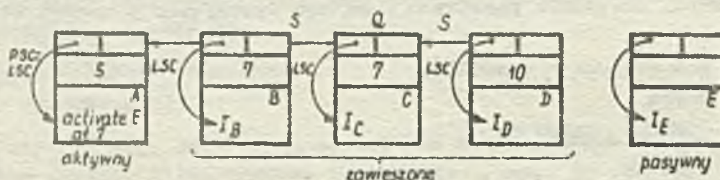
- activate X,
  gdzie X jest referencją do obiektu klasy process lub jej podklasy, powoduje utworzenie zdarzenia procesu X o chwili wznowienia równej time oraz umieszczenie go na początku listy zdarzeń SQS; proces X otrzymuje nazwę current i staje się aktywny (rys. 71).
- activate X $\left\{\begin{matrix}\text{at}\\ \text{delay}\end{matrix}\right\}$ T ;

  powoduje utworzenie zdarzenia procesu X o chwili wznowienia równej

  T'=max(T,time) w wypadku at i

  T''=max (time+T,time) w wypadku delay

  oraz umieszczenie go na liście zdarzeń SQS za wszystkimi występującymi tam zdarzeniami o chwilach nie przekraczających T' lub T'' (rys. 78 i 79);

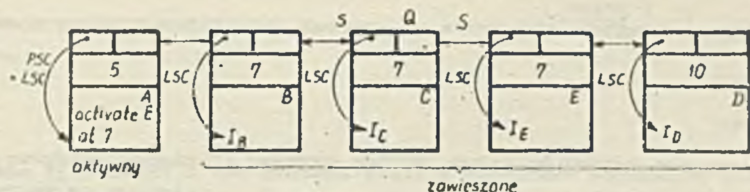


Rys. 78

Instrukcja activate E at 7 w treści obiektu A spowoduje przejście od konfiguracji początkowej (rys. 78) do konfiguracji przedstawionej na rys. 79;

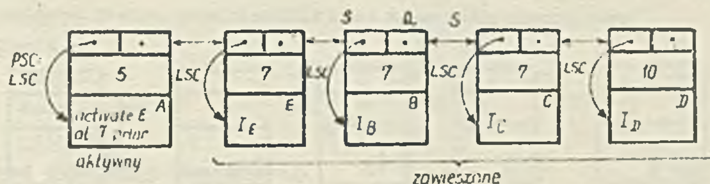
- activate X $\left\{\begin{matrix}\text{at}\\ \text{delay}\end{matrix}\right\}$ T prior ;

działa analogicznie do poprzedniej z tą różnicą, że utworzone zdarzenie procesu X umieszczone jest na SQS przed wszystkimi zdarzeniami o chwilach nie wcześniejszych od T' lub T'';



Rys. 79

instrukcja activate E at 7 prior spowoduje przejście od konfiguracji początkowej (rys.78) do konfiguracji na rys. 80.



Rys. 80

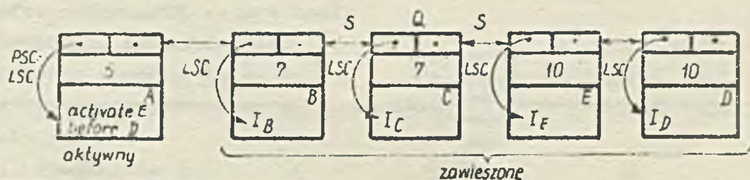
- activate X $\left\{ \begin{array}{l} \text{before} \\ \text{after} \end{array} \right\}$ Y ;

powoduje utworzenie zdarzenia procesu X o chwili równej chwili zdarzenia procesu Y i umieszczenie go na liście zdarzeń SQS bezpośrednio przed (before) lub za (after) zdarzeniem Y; jeżeli Y jest procesem pasywnym lub zakończonym to instrukcja ta jest równoważna wywołaniu procedury cancel (X); przyjmując za początkową konfigurację z rys. 78 instrukcje:

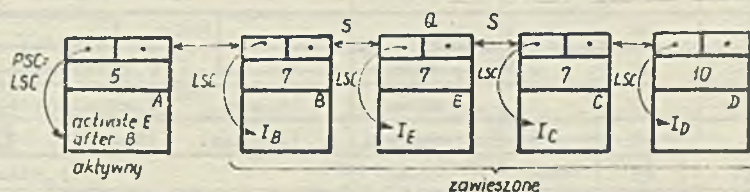
activate E before D;

activate E after B;

spowodują przejście do konfiguracji przedstawionych na rysunkach odpowiednio 81 i 82;



Rys. 81



Rys. 82

- procedure reactivate;

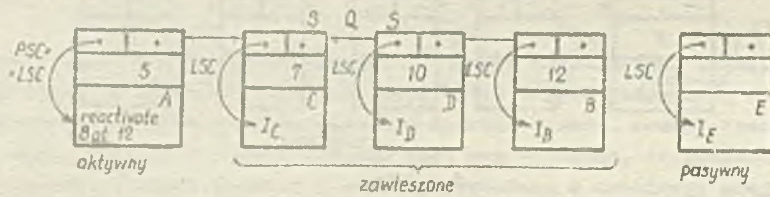
  Procedury reaktywacji dotyczą procesów aktywnych, zawieszonych lub pasywnych. W tym ostatnim wypadku są one równoważne procedurom aktywacji. Mamy więc:

  - reactivate X;
  - reactivate X $\left\{ \begin{array}{l} \text{at} \\ \text{delay} \end{array} \right\}$ T;

- reactivate X { at / delay } T prior ;

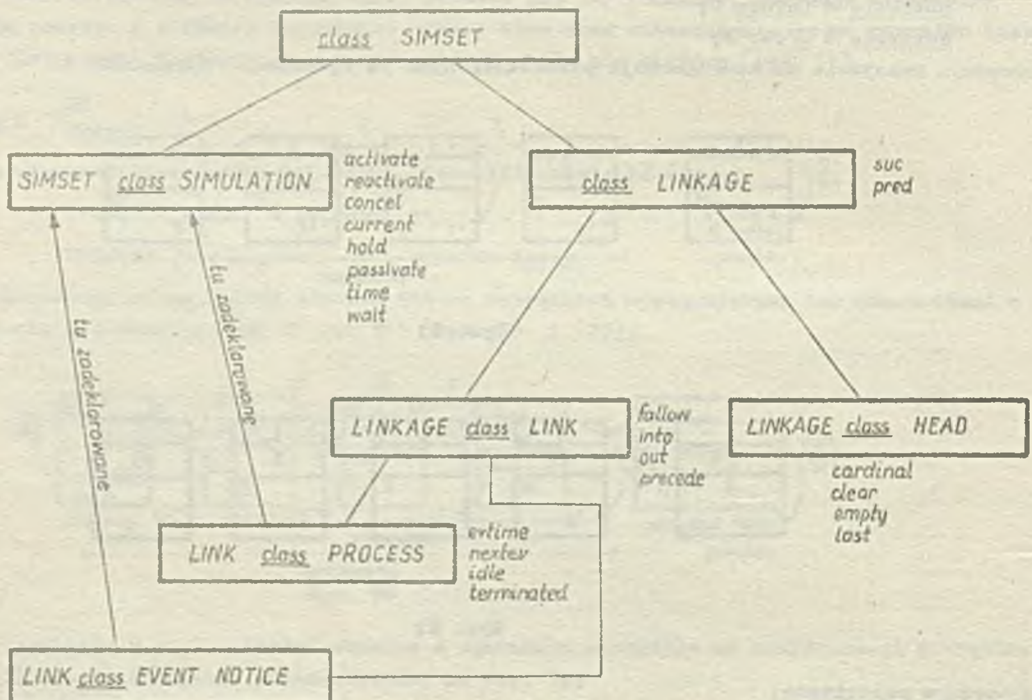
- reactivate X { before / after } Y;

Działanie tych procedur polega na usunięciu z listy zdarzeń SQS i likwidacji zdarzenia procesu X, a następnie wywołaniu procedury aktywacji w odniesieniu do tego procesu. Dla przykładu przypuśćmy, że mając daną konfigurację początkową z rys. 78, chcemy przeplanować chwilę wznowienia procesu B z 7 na 12. Ponieważ proces B znajduje się na liście zdarzeń, stąd użycie procedury aktywacji jest błędne. Należy wywołać procedurę reaktywacji, tj. reactivate B at 12. Otrzymana w wyniku tego konfiguracja ma postać (rys. 83).



Rys. 83

Ogólny schemat struktury klasy Simset i Simulation ilustruje rys. 84. Zwróćmy uwagę na to, że procedury activate, passivate, hold, itp. dotyczą tylko obiektów klasy process lub jej podklasy.



Rys. 84

Rysunek ten ilustruje ponadto możliwości dostępu do atrybutów zdefiniowanych na różnych poziomach prefiksowania.

- Przykłady

- Przeanalizujmy przykład symulacji procesu obróbki dwóch półfabrykatów przez jedną obrabiarkę. Program symulacyjny ma postać:

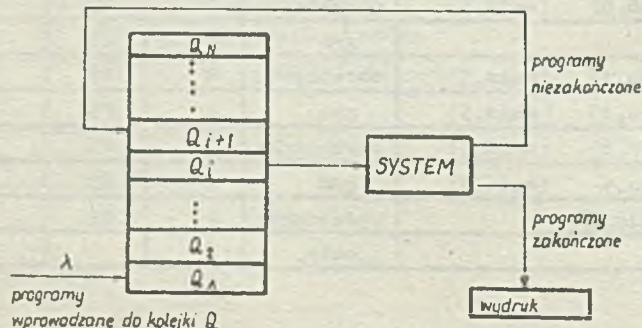
```
Simulation begin
              ref (obrabiarka) obróbka; ref (head) bufor␣obrabiarki;
              process class obrabiarka;
              begin
E1            A: inspect bufor␣obrabiarki.first when półfabrykat do
                 begin
                   out;
                   hold (2);
E2                 goto A;
                 end otherwise passivate;
E3               goto A;
               end; $ obrabiarka $
               link class półfabrykat;
               process class generator;
               begin   integer i;
G1               for i := 1 step 1 until 2 do
                 begin
                   new półfabrykat.into (bufor␣obrabiarki);
                   if obróbka.idle then activate obróbka delay 0;
                   hold (1);
                 end;
               end; $ generator $
               bufor␣obrabiarki :- new head;
               obróbka :- new obrabiarka;
F1             activate obróbka delay 0;
F2             activate new generator after obróbka;
               hold (5);
F3           end; $ Simulation $
```

tab. 1

S Q S	current	time	main	obrabiarka	gener.
(main,0)	main	0	F1		
(main,0) (obrabiarka,0)	main	0	F2	E1	
(main,0) (obrabiarka,0) (gen,0)	main	0	F2	E1	G1
(obrabiarka,0) (gen,0) (main,5)	obrabiarka	0	F3		G1
(gen,0) (main,5)	gen	0	F3	E3	
(obrabiarka,0) (gen,1) (main,5)	obrabiarka	0	F3		G1
(gen,1) (obrabiarka,2) (main,5)	gen	1	F3	E2	
(obrabiarka,2) (gen,2) (main,5)	obrabiarka	2	F3		G1
(gen,2) (obrabiarka,4) (main,5)	gen	2	F3	E2	
(obrabiarka,4) (main,5)	obrabiarka	4	F3		
(main,5)	main	5			

Przebieg programu ilustruje tab. 1. Wskutek wejścia do bloku symulacji generowany jest systemowo obiekt SQS , reprezentujący listę zdarzeń, proces main oraz zdarzenie procesu main z chwilą 0, które oznaczymy przez (main,0). Zdarzenie to wchodzi na listę zdarzeń SQS i rozpoczyna się wykonywanie programu procesu main. Tworzony jest zbiór bufor␣obrabiarki (kolejka) i obiekt obróbka, klasy process (reprezentujący obrabiarkę). Następnie (F1) generowany jest proces obrabiarka i jego zdarzenie w chwili 0, oznaczane dalej przez (obrabiarka,0). Podobnie tworzony jest proces generator, ze swoim zdarzeniem (gen,0) , które umieszczane jest na SQS za zdarzeniem (obrabiarka,0). Wskutek wykonania instrukcji hold(5) proces main zostaje przeplanowany, a jego nowe zdarzenie (main,5) umieszczane jest na ostatnim miejscu listy zdarzeń. Rozpoczyna się wykonywanie programu procesu obrabiarka od miejsca E1 wskazywanego od jego LSC. Ponieważ generator jeszcze nie działał więc kolejka bufor␣obrabiarki jest pusta i obrabiarka zostaje spasywowana (E3), a jej zdarzenie usunięte z listy zdarzeń. W wyniku działania generatora, który otrzymał teraz nazwę current, półfabrykat zostaje wprowadzony do kolejki bufor␣obrabiarki. Ponadto wygenerowane zostaje zdarzenie procesu obrabiarka (obrabiarka,0) i umieszczone za zdarzeniem aktualnie działającego generatora. Instrukcja hold (1) powoduje usunięcie z SQS i likwidację zdarzenia (gen,0) oraz utworzenie nowego zdarzenia (gen,1) i wstawienie go na SQS na zdarzeniem (obrabiarka,0) . Działanie generatora zostaje zawieszone i nazwę current otrzymuje proces obrabiarka. Ponieważ w kolejce bufor␣obrabiarki znajduje się jeden półfabrykat, jest on usunięty z tej kolejki i wzięty do obsługi. Instrukcja hold (2) powoduje usunięcie i likwidację zdarzenia (obrabiarka,0) z SQS oraz utworzenie nowego zdarzenia (obrabiarka,2) i wstawienie go na SQS za zdarzeniem (gen,1) . Obrabiarka przechodzi w stan zawieszony na 2 jednostki czasu symulując obróbkę półfabrykatu. Po wygenerowaniu dwóch półfabrykatów i ich obróbce przez obrabiarkę, procesy obrabiarka i generator zostają spasywowane, a ich zdarzenia usunięte z listy zdarzeń. Nazwę current otrzymuje teraz proces main, którego LSC znajduje się w punkcie F3. Program jest zakończony.

• Przeanalizujmy teraz inny przykład (rys. 85) . Należy zbudować model symulacyjny pewnego prostego systemu operacyjnego maszyny cyfrowej. Zadaniem tego systemu jest przydział jednostkowego czasu liczenia kolejnym programom, o zadanym czasie liczenia, wchodzącym do systemu zgodnie z rozkładem wykładniczym o intensywności λ , i przechowywanych w kolejce programów Q. Kolejka ta ma N poziomów Q1, Q2,...,QN, gdzie Qi stanowi kolejkę na i-tym poziomie. Programy wchodzące do systemu wprowadzane są zawsze do kolejki Q1. Jeżeli czas liczenia programu jest większy od czasu jednostkowego, wtedy program umieszczany jest w kolejce na poziomie o 1 wyższym od tego, z którego został wzięty do liczenia. System zawsze obsługuje w pierwszej kolejności programy z niższych poziomów, np. jeśli liczony był program z poziomu Qp i w tym czasie został wprowadzony nowy program do kolejki Q1, wtedy po obsłużeniu programu z poziomu Qp system przejdzie do wykonywania programu z kolejki poziomu Q1. Programy, które będą obliczone całkowicie, przesyłane są do zbioru wydruk. Z przedstawionego wyżej opisu wynika, że system w pierwszej kolejności obsługuje całkowicie programy o krótkim czasie liczenia. Te, które wymagają dłuższych czasów liczenia osiągają w kolejce Q wysokie poziomy, którym jednostkowy czas liczenia przydzielany jest znacznie rzadziej.
W modelu symulacyjnym należy ponadto obliczać i wydrukować rzeczywiste czasy liczenia wszystkich programów.



Rys. 85

Budowę modelu symulacyjnego rozpocznijmy od określenia podstawowych obiektów występujących w symulacji. Są to:

- programy,
- generator programów,
- system,
- kolejka wydruk oraz tablica kolejek Q.

Ważnym elementem w budowie modelu jest określenie przynależności wymienionych obiektów do odpowiednich klas, a także podanie definicji tych klas. Inaczej mówiąc należy wyznaczyć te obiekty, które w czasie symulacji będą procesami, tj. będą należały do klas prefiksowanych klasą process oraz obiekty będące jedynie elementami kolejek, tj. należące do klas prefiksowanych klasą link. O przynależności obiektu do danej klasy decyduje algorytm jego zachowania się. Powyższe rozważania ilustruje pierwsza wersja programu:

```
          begin
            integer N;
            N := inint;
Simulation begin
              link class program;
              begin
                real czas_wykonania, pozostały_czas, czas_wygenerowania;
                boolean sygnał_końca_programu;
                .............
              end; $ program $
              process class system ....;
              process class generator...;
              ref (head) wydruk;
              ref (head) array Q (1:N);
              wydruk :- new head;
              for j:=1 step 1 until N do
                Q (j) :- new head;
              activate new generator;
              hold (czas_symulacji);
            end; $ Simulation $
          end; $ programu $
```

W bloku zewnętrznym zadeklarowano i wczytano liczbę N poziomów kolejki Q. Blok wewnętrzny prefiksowany jest klasą Simulation. Dostępne są w nim zatem wszystkie procedury klasy Simulation. Zadeklarowano w nim między innymi klasy system i generator prefiksowane klasą process, ponieważ obiekty należące do tych klas stanowią w symulacji procesy, a także klasę program prefiksowaną klasą link. Obiekty tej ostatniej symulują zachowanie się programów. Każdy program ma takie atrybuty, jak czas_wykonania, pozostały_czas oraz czas_wygenerowania, a także atrybut boolowski sygnał_końca_programu, który przyjmuje wartość true, gdy program został całkowicie wykonany. Programy wstawiane są w czasie symulacji do kolejek Qi oraz wydruk, stąd konieczność prefiksowania klasy program systemową klasą link.

Generator programów jest obiektem, który w losowych chwilach generuje obiekty należące do klasy program. Ponieważ obiekt ten symuluje upływ czasu pomiędzy kolejnymi generacjami programów, musi więc należeć do klasy prefiksowanej klasą process.

System jest obiektem, który realizuje podstawowy algorytm obsługi programów sformułowany w opisie problemu. Należy on do klasy prefiksowanej klasą process ponieważ będzie on symulował upływ jednostkowego czasu przetwarzania pojedynczego programu.

Program główny zawiera generację zbioru wydruk oraz Q, aktywację generatora, a także wywołanie procedury hold (czas_symulacji), która zawiesza program główny main na czas określony w argumencie. Po upływie tego czasu symulacja jest zakończona.

Kolejne wersje programu powinny zawierać dalsze uszczegółowienie definicji poszczególnych klas oraz programu głównego. Poniżej przedstawiono końcową postać programu. Wzbogacono ją o liczne komentarze, które powinny ułatwić zrozumienie pełnego modelu symulacyjnego.

```
        begin
          integer N;
          N := inint;
Simulation begin
            real czas␣jednostkowy, czas␣symulacji, lambda;
            integer U,j;
            ref (system) komputer;
            ref (head) wydruk;
            ref (head) array Q (1:N);
            link class program;
            begin
              real czas␣wykonania, pozostały␣czas, czas␣wygenerowania;
              boolean sygnał␣końca␣programu;
              czas␣wygenerowania := time; $ wyznaczenie chwili generacji $
              pozostały␣czas := inreal; $ wczytanie czasu liczenia programu $
              if komputer.idle then activate komputer delay 0;
            $ wprowadzony program, w przypadku bezczynności systemu, aktywuje go $
            end; $ program $
            process class generator;
            begin
              while true do
              begin
                new program.into (Q(1)) ; $ generacja programu i wstawienie go na pierwszy
                                              poziom kolejki Q $
                hold (negexp (lambda,U)) ;
              end;
            end; $ generator $
            process class system;
            begin
              integer i; ref (program) prog; real czas␣obsługi;
              while true do
              begin
                for i:=1 step 1 until N do
                if not Q (i) .empty then
                begin                     $ blok obliczany jeśli i-ta kolejka nie jest pusta $
                  prog :- Q(i). first;
                  inspect prog do
                  begin
                    if pozostały␣czas≥czas␣jednostkowy then
                    begin              $ gdy pozostały czas jest większy od czasu jednostkowego $
                      czas␣obsługi := czas␣jednostkowy;
                      pozostały␣czas := pozostały␣czas-czas␣jednostkowy;
                    end else
                    begin
                      czas␣obsługi := pozostały␣czas;
                      sygnał␣końca␣programu := true;
                    end;
                    out;
```

```
                hold (czas␣obsługi);         $ symulacja czasu przetwarzania programu $
              . if sygnał␣końca␣programu then
                begin
                  czas␣wykonania:=time-czas␣wygenerowania; $ obliczenie rzeczywistego czasu
                                           liczenia $
                   into (wydruk) ;
                end  else
                if i<N  then into (Q(i+1)) else into  (Q(N));
                   $ wstawienie programu na wyższy poziom $
              end; $ inspect $
              goto styk; $ wyskok z pętli for w celu umożliwienia rozpoczęcia przeszukiwa-
                              nia kolejki Q od najniższego poziomu $
            end; $ for $
                passivate; $ w wypadku braku programów system pasywuje się $
     styk:  end; $ while $
         end; $ system $
                         $ program główny $
         lambda:=inreal;  czas␣jednostkowy:=inreal;  U:=inint;
         czas␣symulacji  := inreal;
         for j:=1 step 1 until N do
          Q (j) :- new head                $ utworzenie kolejek $
         wydruk :- new head;
         komputer :- new system;
         activate komputer;                $ aktywacja systemu $
         activate new generator;           $ aktywacja generatora $
         hold (czas␣symulacji);
           $ po tym czasie wszystkie wykonane całkowicie programy znajdują się w zbiorze
             wydruk $
         while not wydruk.empty do
         inspect wydruk.first when program do
         begin
           outfix (czas␣wykonania, 10,5) ; $ wydrukowanie wartości rzeczywistego czasu
                                               liczenia programu $
           outimage;
           out;                             $ usunięcie programu ze zbioru wydruk $
         end;
       end; $ Simulation $
     . end; $ programu $
```

- **Zakończenie**

Przedstawiony w tej oraz w poprzednich częściach język Simula 67 jest bardzo efektywnym narzędziem dla projektantów obiektowych systemów komputerowych. Sformułowanie problemu w tym języku stanowi w istocie gotowy program dla maszyny cyfrowej, która służy wtedy do przeprowadzenia eksperymentów symulacyjnych. Ważnym fragmentem języka jest jego bogactwo w zakresie operacji na tekstach, czego nie ma żaden inny język symulacyjny.

W procesie projektowania często wykorzystuje się Simulę 67 jedynie do opisu systemu nie przeprowadzając żadnych badań na komputerze. Posługiwanie się tym językiem zmusza wtedy projektanta-programistę do dokładnego określenia wszystkich atrybutów i funkcji systemu (procesu), co przyczynia się do precyzyjnego zrozumienia problemu.

Kierunki rozwijania Simuli 67 wiążą się ściśle z możliwościami rozszerzania tego języka o nowe klasy systemowe. W ten sposób powstała klasa CADSIM [5] do symulacji ciągłej, klasa OASIS [6] do symulacji baz danych i systemów operacyjnych, klasa ICGL [7] oraz GRAPHIC [8] do kon-

wersacyjnej grafiki komputerowej, itp. Prowadzone są także prace nad zastąpieniem klasy Simulation klasą Realtimesimulation [9], która umożliwiłaby symulację w czasie rzeczywistym. Simula była także inspiracją do opracowania pakietu programów o nazwie SIMLIB [10], który rozszerza język programowania mikroprocesorów PL/M na zagadnienia symulacyjne umożliwiając symulację bezpośrednio na mikroprocesorach.

Literatura

[1] Birthwistle G. i in.: Notes on Simula language. Publication No. S-7, NCC Forskningsweien 1B, Oslo, 1969

[2] Brithwistle G. i in.: Common base language. Simula information, Publication No.S-22, NCC, Forskningsweien 1B, Oslo, 1979

[3] Birthwistle G. i in.: Simula begin. Auerbach Publishers, INc. Philadelphia, Pa, 1973

[4] Iohbiach J.D., Morse S.: General concepts of Simula 67 programming language. Compagnie Internationale pour l'Informatique Les Clayes Sous Bois, France

[5] Cunningham J., Sim R.: Continous simulation in Simula: a brief description of class CADSIM. Simula Newsletter, 1976 t. 4, nr 2

[6] Unger B.W.: OASIS - a Simula extension for systems software and simulation. Simula Newsletter, 1977 t. 5 nr 2

[7] Alegria J.A.S.: ICGL - a Simula 67 extension for structured display programming in interactive computer graphic, Simula Newsletter, 1977 t. 6 nr 3

[8] Szpor L.: A class GRAPHIC for calcomp plotters. Simula Newsletter, 1978 t. 6 nr 4

[9] Philippot G.P.: Using the class realtimesystem. Simula Newsletter, 1978 t. 6 nr 3

[10] Poznański Z.: SIMLIB Simula-based simulation package for microcomputer system design. Simula Newsletter 1981 t. 9 nr 1.

dr Jacek OLSZEWSKI
Instytut Maszyn Matematycznych

O programowaniu i weryfikacji struktur systemów operacyjnych

Praca przedstawia skrócony opis opartego na Pascalu języka programowania systemów operacyjnych PHIL (Process Hierarchy Implementation Language) i wyjaśnia znaczenie struktur procesów, które można w tym języku programować. Bieg procesów w strukturach zaprogramowanych w Philu jest wyjaśniony za pomocą sieci Petri'ego. Znaczenie tych elementów języka, które stanowią uzupełnienie Pascala jest podane w postaci aksjomatycznych warunków ich weryfikacji.

Praca zawiera również przykład struktury procesów, która jest rozwiązaniem problemu pięciu filozofów, oraz próbę weryfikacji tej struktury.

1. Wstęp

Niniejsza praca jest uzupełnieniem książki pt. "Projektowanie struktur systemów operacyjnych" [6], która zawiera pewną propozycję metodologiczną dotyczącą projektowania i programowania systemów operacyjnych. Propozycja ta została po raz pierwszy opublikowana jako raport [5]. Ponieważ wspomnianą książkę napisałem z myślą o dość szerokim kręgu czytelników, nie umieściłam w niej rozważań dotyczących formalnych definicji wprowadzonych pojęć i prób weryfikacji przykładów struktur. Rozważania te są treścią niniejszej pracy.

Aby zrozumieć niniejszą pracę, niekoniecznie trzeba przeczytać wspomnianą książkę; wystarczy wiedza informatyczna w zakresie systemów operacyjnych na poziomie uniwersyteckim.

Propozycja zawarta w wymienionym opracowaniu zmierza ku takiemu podejściu i językowi programowania, w którym całość systemu operacyjnego jest ujęty w sposób jednolity, włączając w to także tzw. jądro. Jak wiadomo, jądrem systemu operacyjnego przyjęto nazywać tę jego część, która wymykała się dotychczasowym próbom pojęciowego usystematyzowania dziedziny systemów operacyjnych. Przez jądro rozumie się np. zbiór takich operacji wykonywanych w maszynie, które nie wchodzą w skład żadnego z procesów. Inne określenie jądra: "jądro jest to zbiór tzw. programów reakcji na przerwania i programów elementarnych operacji synchronizacyjnych". Innymi słowy, podstawą rozważań o strukturach systemów operacyjnych jest do tej pory nie sama maszyna, jak ją inżynierowie zaprojektowali, lecz maszyna z pewnym oprogramowaniem, które pozwala "nie wiedzieć" o systemie przerwań, o sposobie elementarnej synchronizacji i tp. Zaś działanie tak uzupełnionej maszyny przedstawia się jako pewien zbiór współbieżnych procesów. Samo pojęcie "proces" służy wyodrębnieniu oddzielnych akcji wykonywanych przez maszynę: a więc oddzielnych zbiorów danych oraz ciągów operacji, z których każdy odnosi się do "swojego" zbioru danych. Oczywiście nie byłoby powodu do rozważań o strukturach systemów operacyjnych, gdyby nie istniały dane wspólne dla wielu akcji, a także wspólne rejestry, urządzenia wejścia/wyjścia itp.

Proces - w powyższym rozumieniu tego terminu - jest wykonywaniem operacji, które odnoszą się do danych lokalnych (prywatnych), jak i do danych wspólnych z innymi procesami. Dane, które są wspólne dla dwóch lub więcej procesów, oraz operacje, które się do tych danych odnoszą, są z owych procesów wyodrębniane i określane mianem monitora ([1], [2] i in.). Bieg każdego z tych procesów może zatem być rozważany w następujących kategoriach: na zewnątrz monitora, wejście do monitora, wewnątrz monitora i wyjście z monitora. Proces może wejść do monitora pod warunkiem, że wewnątrz tego monitora nie ma żadnego innego procesu. W przeciwnym razie proces wchodzący do monitora będzie wstrzymany aż do chwili, gdy ten inny proces z monitora wyjdzie (szeregowanie krótkoterminowe). Ponadto, musi być możliwe zatrzymanie procesu także wewnątrz monitora, po to, aby umożliwić innemu procesowi dostęp do tych samych danych. Innymi słowy, nie możemy wykluczyć sytuacji, w której jeden proces musi poczekać na inny proces (szeregowanie średnioterminowe). Dlatego właśnie potrzebne są operacje synchronizacyjne. Ponieważ zaś wykonywanie tych operacji nie stanowi części żadnego z procesów gdyż są to działania rozpoczynające się wstrzymaniem procesu i kończące się wznowieniem procesu lub procesów - nie da się one ująć za pomocą pojęć "proces" i "monitor". Stąd potrzeba wyodrębnienia operacji, które nie wchodzą w skład żadnego z procesów, czyli utworzenia wspomnianego wyżej jądra.

Przedstawiona w opracowaniu [6] propozycja polega na tym, aby nieco inaczej rozumieć pojęcie proces. Proces w nowym rozumieniu będzie pojęciem węższym; podział tego, co się dzieje w maszynie na procesy będzie nieco drobniejszy. Natomiast liczba procesów i ich umiejscowienie w strukturze systemu operacyjnego będzie stało. Pojęcie proces będzie bowiem służyć wyodrębnieniu oddzielnych zbiorów danych oraz wyłącznie tych ciągów operacji, z których każdy odnosi się tylko do "swojego" zbioru danych. Tam, gdzie uprzednio mówiliśmy o operacjach dotyczących zbioru danych, który był wspólny dla wielu procesów, teraz powiemy, że istnieje oddzielny proces, dla którego ów zbiór jest jego własnym. Proces ten może być wywoływany przez inny proces, który przekazuje mu odpowiednie parametry oraz sam zatrzymuje się. Jeśli wywołanie procesu zdarzy się w czasie, gdy ów proces właśnie biegnie, wówczas wywołanie to będzie opóźnione aż do zakończenia trwającego przebiegu (szeregowanie krótkoterminowe). Wznowienie procesu wywołującego

nastąpi nie wcześniej niż po zakończeniu jednego przebiegu procesu wywołanego - tego przebiegu, który odpowiada wywołaniu.

Zaakceptowanie takiego rozumienia pojęcia "proces" sprawia, że to, co stanowi zazwyczaj działanie jądra, daje się potraktować jako wiele różnych procesów, które w strukturze systemu operacyjnego zajmują bardzo różne miejsca. Np. reakcja na przerwanie któregoś z kanałów transmisji danych może być potraktowana jako proces, który jest nadrzędny w stosunku do procesu szeregowania zgłoszeń do tego kanału; na równi z procesem czytania kolejnych znaków i umieszczania ich na odpowiednim buforze [6]. Innymi słowy, proces reagowania na przerwanie oraz proces czytania znaków wywołują proces szeregowania zgłoszeń jako im podrzędny. Przy dotychczasowym rozumieniu procesu czytanie znaków było traktowane jako proces, szeregowanie zgłoszeń - jako monitor, zaś reakcja na przerwanie - jako część działań jądra.

Na różnicę między dotychczasowym i nowym pojmowaniem procesu można również spojrzeć inaczej; od strony języka programowania. Język programowania systemów operacyjnych, u podstaw którego leży dotychczasowe określenie pojęcia"proces", z natury rzeczy zawiera pewne standardowe mechanizmy zapewniające wspomniane wyżej szeregowanie krótkoterminowe oraz średnioterminowe. Oznacza to, że chcąc korzystać z danego języka programowania, oprócz jego kompilatora musimy dysponować odpowiadającym mu systemem obsługi wykonania (ang. run-time system), który zawiera wspomniane mechanizmy. Ponieważ zaś mechanizmów tych nie można wyrazić za pomocą pojęć, na których język ten jest oparty, system obsługi wykonania musi być zaprogramowany w innym języku; praktycznie w kodzie maszynowym. Nietrudno zorientować się, że w wypadku języków programowania systemów operacyjnych ów system obsługi wykonania odpowiada temu, co wyżej określaliśmy jako jądro. Jeśli przyjmiemy nowe znaczenie pojęcia"proces", to okaże się, że system obsługi wykonania nie jest bezwzględnie konieczny. Język programowania systemów operacyjnych, nawet wysokiego poziomu, można sformułować tak, aby to, co ma być w maszynie wykonywane, było w całości wygenerowane przez jego kompilator. Jedynym standardowym mechanizmem koniecznym w takim języku jest bowiem mechanizm szeregowania krótkoterminowego, czyli mechanizm opóźnienia kolejnego rozpoczęcia lub wznowienia procesu aż do zatrzymania lub zakończenia jego bieżącego przebiegu. Ponieważ mechanizm ten polega na tzw. aktywnym czekaniu, do jego realizacji wystarczy, aby wygenerowany przez kompilator kod każdego z procesów zaopatrzony był w odpowiednie pętle aktywnego czekania. Nie trzeba więc do tego systemu obsługi wykonania. Zauważmy, że w wypadku maszyny jednoprocesorowej mechanizm ten jest w ogóle niepotrzebny.

Może teraz powstać pytanie, czy jednak system obsługi wykonania nie jest konieczny dla innych powodów niż standardowy mechanizm szeregowania krótkoterminowego. Poprzednio był on niezbędny ze względu na standardowy mechanizm szeregowania średnioterminowego oraz w wypadku języków wysokiego poziomu jest on uważany za niezbędny ze względu na standardowe operacje wejścia/wyjścia i dynamiczny przydział pamięci. Z tego, co powiedzieliśmy wyżej o propozycji nowego rozumienia pojęcia"proces"wynika, że szeregowanie średnioterminowe nie jest już kwestią żadnego standardowego mechanizmu w języku programowania. Ponieważ chodzi nam o język programowania systemów operacyjnych, a więc programowania m.in. procesów wejścia/wyjścia, proces dynamicznego przydziału pamięci itp., nie będziemy zakładać, że są to standardowe mechanizmy tego języka.

Istota zaproponowanego podejścia do projektowania i programowania systemów operacyjnych i odpowiadający temu podejściu język programowania zostały szczegółowo przedstawione i zilustrowane w opracowaniu [6]. Książkę tę można traktować jako wprowadzenie do zagadnień projektowania i programowania systemów operacyjnych dokonane w sposób nieformalny, odwołujący się do intuicji czytelnika. Powyższe odnosi się zwłaszcza do semantyki zaproponowanego języka. Język ten zyskał sobie nazwę Phil (od słów Process Hierarchy Implementation Language), toteż dla wygody będziemy tej nazwy tutaj używać. Składnia Phila jest przedstawiona w pracy [6] za pomocą diagramów syntetycznych zamieszczonych na końcu książki w Dodatku. Natomiast semantyka jest omówiona tylko na przykładach i wyjaśniona poprzez porównanie z Pascalem. Celem niniejszej pracy jest przedstawienie semantyki Phila w sposób nieco bardziej formalny i rozważenie możliwości weryfikacji struktur procesów zaprogramowanych w tym języku.

W następnym paragrafie Czytelnik znajdzie skrót tego, co w opracowaniu [6] stanowi opis

Phila, w paragrafie 3 - sposób przedstawiania w postaci sieci Petri'ego struktur procesów zaprogramowanych w Philu, w paragrafie 4 - aksjomatyczne definicje pojęć stanowiących uzupełnienie Pascala w Philu, w paragrafie 5 - próbę weryfikacji struktury procesów, która odpowiada rozwiązaniu problemu pięciu filozofów, a w paragrafie 6 - podsumowanie.

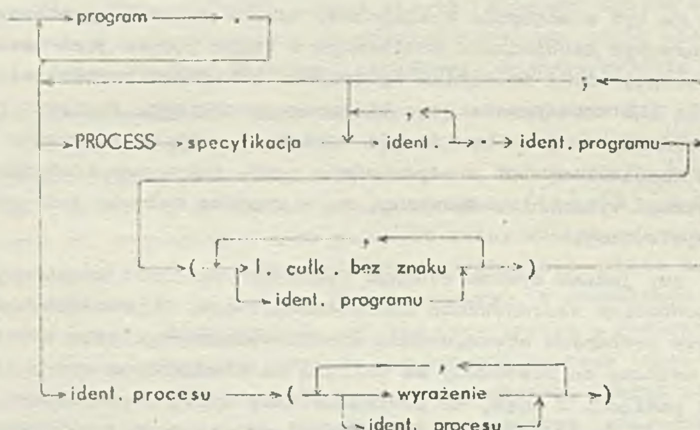
2. Phil

Phil można określić jako maszynowo ukierunkowany język programowania wysokiego poziomu. Jak wiele takich języków, jest on modyfikacją języka Pascal([1], [8] i in.). Modyfikacje Pascala z reguły polegają na usunięciu z niego elementów, które nie są w zgodzie z założoną filozofią konstruowania systemów operacyjnych, i na uzupełnieniu tak okrojonego Pascala o elementy odpowiadające pojęciom, na których opiera się owa filozofia.

W naszym języku usuwamy z Pascala:
- pliki (ang. files), ○ zbiory (sets), ○ zmienne dynamiczne, ○ procedury i funkcje rekurencyjne, czyli wszystko to, co wymagałoby systemu obsługi wykonania. Natomiast wprowadzamy pojęcia "proces" i "struktura procesów" oraz stowarzyszone z tymi pojęciami pewne standardowe struktury danych, procedury i zdania. Poniżej następuje bardzo skrócony opis Phila, przedstawiony z punktu widzenia struktur systemów operacyjnych, które mają w nim być programowane.

Struktura procesów jest określona przez pewien ciąg programów, ciąg deklaracji procesów oraz zdanie inicjujące. Diagram syntaktyczny, odpowiadający strukturze procesów, może wyglądać jak następuje*:

struktura procesów



To co w deklaracji procesu określamy symbolem "specyfikacja", jest zależne od organizacji maszyny i wskazuje, czy proces ma być np. procesem, który daje się przerywać, czy też takim, który nie daje się przerywać. Przy bardziej złożonej organizacji maszyny specyfikacja może zawierać np. numer poziomu przerwań (jeżeli maszyna ma wielopoziomowy system przerwań). Identyfikator programu wskazuje program, według którego proces ma być zadeklarowany. Program ten musi wystąpić w ciągu programów poprzedzającym deklaracje procesów. Samo pojęcie "program" jest więc traktowane jako pewien wzorzec procesu, tak jak typ danej jest pewnym wzorcem, według którego można zadeklarować zmienną. Jak widać, deklaracja procesu może jeszcze zawierać objęty nawiasami ciąg stałych liczbowych i identyfikatorów procesów. Jest to wyliczenie tych wartości, które

---

* Diagramy syntaktyczne, za pomocą których przedstawiamy definicje poszczególnych pojęć, zawierają symbole nieterminalne - oznaczone słowami złożonymi z małych liter, oraz symbole terminalne - oznaczone słowami złożonymi z dużych liter i innych znaków pisarskich.

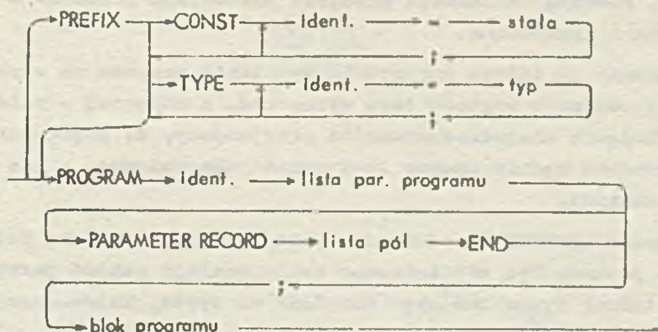
w programie procesu oznaczone są przez identyfikatory stałych parametrów programu, oraz tych procesów, które w deklarowanym procesie mogą być wywoływane, a które w programie procesu oznaczone są jako parametry rodzaju process (będzie to uwidocznione w następnych diagramach syntaktycznych odnoszących się do pojęcia program).

W odniesieniu do deklaracji procesów obowiązuje zasada, że proces musi być najpierw zadeklarowany, a dopiero potem może być wymieniony w deklaracji innego procesu na liście objętej nawiasami. Zatem ciąg deklaracji wszystkich procesów stanowi niejako porządek budowy całej struktury procesów. Najpierw muszą być zadeklarowane procesy, które nie wywołują żadnych innych, następnie procesy, które wywołują uprzednio zadeklarowane itd.

Ostatnim elementem struktury procesów opisanej powyższym diagramem jest zdanie oznaczające tzw. inicjujące wywołanie procesu. W zdaniu tym wskazany jest proces, od zainicjowania którego rozpoczyna się działanie całej struktury procesów *. Objęty nawiasami ciąg wyrażeń i identyfikatorów procesów w wywołaniu inicjującym ma zupełnie inne znaczenie niż ciąg stałych i identyfikatorów procesów w deklaracji procesu. Jest to tzw. lista parametrów wywołania procesu, której w programie tego procesu odpowiada tzw. rekord parametrów(zob. następny diagram), podobnie jak lista parametrów aktualnych procedury odpowiada liście parametrów formalnych tej procedury.

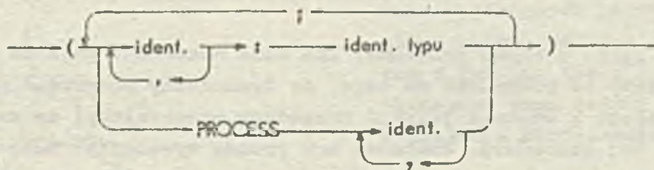
Symbol "program" użyty w powyższym diagramie można przedstawić diagramem:

program



Samo pojęcie "program" w Philu jest bardzo podobne do pojęcia "program" w Pascalu. Jedynie jego nagłówek jest nieco bardziej rozbudowany. Zawiera on bowiem, oprócz identyfikatora programu, tzw. listę parametrów programu oraz rekord parametrów. Lista parametrów programu określona diagramem:

lista par. programu



stanowi wyliczenie identyfikatorów stałych, których wartości zostaną określone dopiero w momencie deklaracji procesu wg tego programu, oraz identyfikatorów oznaczających procesy, które w tym programie będą wywoływane.

Rekord parametrów, składniowo różniący się od zwykłego rekordu tylko słowem PARAMETER, jest również listą parametrów, chociaż o zupełnie innym znaczeniu. Odpowiada on liście parametrów

---

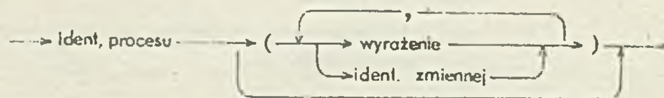
* W wypadku maszyny wieloprocesorowej inicjujących wywołań procesów powinno być tyle, ile jest w maszynie procesorów.

wywołania procesu, który zostanie zadeklarowany wg tego programu, podobnie jak lista parametrów formalnych procedury odpowiada liście parametrów aktualnych tej procedury. Odpowiedniość ta jest niezupełnie taka jak w procedurach; wyrażeniu na liście parametrów wywołania procesu odpowiada pole rekordu parametrów takiego typu, który obejmuje wartość tego wyrażenia, natomiast zmiennej - pole typu "wskaźnik" (ang. pointer) do typu tej zmiennej *. Jeśli trzymać się dalej analogii z procedurami, odpowiedniość ta może przypominać sposób realizacji listy parametrów procedury w kompilatorach. Przekazywanie parametrów przy wywołaniu jednego procesu przez drugi jest dlatego tak pomyślane, aby możliwa była niezależna kompilacja programów, wg których procesy te będą zadeklarowane. Wymaganie niezależnej kompilacji programów oznacza, że w trakcie kompilacji programu nic nie wiadomo o programach, wg których będą zadeklarowane procesy wywoływane w programie kompilowanym. Nie ma więc związku między listami parametrów wywołań procesów i rekordami parametrów występującymi w nagłówkach programów w trakcie ich kompilacji. Ów związek jest ustanawiany dopiero wtedy, gdy następują deklaracje procesów. Powyższy sposób przekazywania parametrów pozwala kompilatorowi dla każdego wywołania procesu utworzyć pewien rekord zawierający tzw. ślad procesu wywołującego (inaczej , miejsce i stan, do którego należy wrócić przy wznowieniu tego procesu) oraz wartości wyrażeń i wskaźniki do zmiennych, stosownie do tego, co zawiera lista parametrów wywołania procesu. Rekord ten nazwijmy aktualnym rekordem parametrów dla wywołanego procesu. Dla symetrii, rekord parametrów występujący w nagłówku programu nazwijmy formalnym. Utworzenie aktualnego rekordu parametrów jest więc niezależne od postaci odpowiedniego formalnego rekordu parametrów. Podczas deklaracji procesów postać obu rekordów - aktualnego i formalnego - może być porównana i sprawdzona.

Powiedzieliśmy, że wyrażeniu na liście parametrów wywołania procesu ma w rekordzie parametrów odpowiadać pole typu, który obejmuje wartość tego wyrażenia, a zmiennej - pole typu "wskaźnik" do typu tej zmiennej. Dla uniknięcia niejednoznaczności przyjmujemy, że pojedyncza zmienna na liście parametrów wywołania procesu będzie zawsze traktowana jako zmienna, a nie jako wyrażenie, na które składa się tylko ta zmienna.

Tekst zawarty między słowami PREFIX i PROGRAM jest przewidziany jako definicje tych stałych i typów danych, które powinny być zdefiniowane zanim nastąpi rekord parametrów. W rekordzie tym będą bowiem pola takich typów lub typu "wskaźnik" do typów, które muszą być zdefiniowane wcześniej.

Symbol blok programu jest określony diagramem, który różni się od diagramu symbolu blok w oryginalnym Pascalu tylko tym, że jest w nim przewidziany jeszcze jeden rodzaj zdania, tj. zdania stanowiącego wywołanie procesu:



Zdanie to może pojawić się tylko w bloku programu. Nie może natomiast pojawić się w bloku procedury ani funkcji, gdyż mogłoby to prowadzić do tego, że dynamiczny przydział pamięci znowu byłby niezbędny. Wywołanie procesu z wnętrza jakiejś procedury oznaczałoby, że należy zachować wszystkie dane lokalne dla tej procedury. Ponieważ zaś proces wywołujący może sam być ponownie wywołany zanim nastąpi jego wznowienie i, co więcej, w trakcie jego nowego przebiegu może nastąpić wywołanie rozważanej procedury, musiałoby nastąpić zarezerwowanie nowego obszaru pamięci dla jej danych lokalnych. Innymi słowy, dopuszczenie wywołania procesów wewnątrz procedur i funkcji byłoby tym samym, co dopuszczenie procedur i funkcji rekurencyjnych.

Dalsze szczegóły składni Phila oraz nieformalny opis jego semantyki można znaleźć w opracowaniu [6]. Tutaj przedstawimy jeszcze tylko znaczenie dwóch standardowych procedur: hold i resume.

Proces, w którym doszło do wywołania innego procesu, jest zatrzymany aż do wznowienia. Wznowienie to następuje w procesie wywołanym lub pośrednio wywołanym, tj. takim, który został wywo-

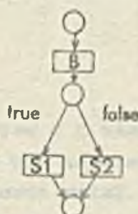
---

* Jak widać, chociaż zrezygnowaliśmy ze zmiennych dynamicznych, pozostały w Philu wskaźniki.

łany przez proces wywołany (głębokość takiego zagnieżdżenia wywołań jest dowolna). Wznowieniu procesu służą procedury hold(var l:link) i resume (l:link). Wykonanie procedury hold polega na przypisaniu śladu procesu wywołującego zmiennej stanowiącej parametr tej procedury (jest to więc przepisanie śladu z aktualnego rekordu parametrów do wskazanej zmiennej). Natomiast wykonanie procedury resume należy rozumieć jako zakończenie procesu wywołanego i wznowienie procesu, którego ślad jest wartością parametru tej procedury.

3. Struktury procesów jako sieci Petri'ego

Bieg działań w strukturze procesów opisanych w Philu najwygodniej będzie wyjaśnić za pomocą sieci Petri'ego [7]. Elementom tych sieci nadamy następujące interpretacje: przejścia (prostokąty) będą oznaczać zdania i wyrażenia, miejsca (kółka) będą stanowić warunki rozpoczęcia wykonywania zdań lub wyliczania wyrażeń, do których prowadzą strzałki zaczynające się w tych miejscach. Gdy przejście oznacza wyrażenie, wówczas jedno z miejsc, które dla tego przejścia są miejscami wyjściowymi, oznacza wybór deterministyczny. Np. zdanie if B then S1 else S2 przedstawiamy jako



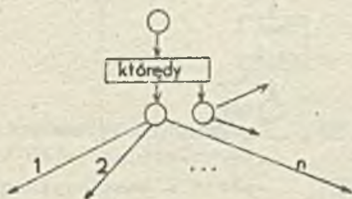
a zdanie

case A of
a1: S1;
a2: S2;
.
.
.
an: Sn end

jako

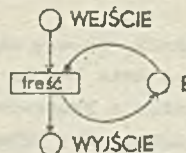
Przyjęta tutaj konwencja rysunkowa polega na oznaczaniu strzałek wychodzących z miejsca, które dla przejścia oznaczającego wyrażenie jest wyjściowe, wartościami tego wyrażenia.

Wybór niedeterministyczny jest oczywiście także brany pod uwagę, jak również kombinacja obu rodzajów wyboru, np.:



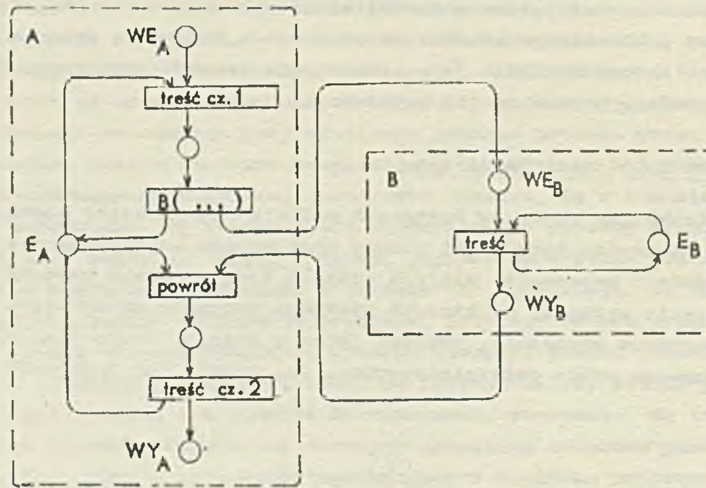
Odpalenie "którędy" spowoduje, że w obu miejscach wyjściowych pojawi się kropka, a dalej nastąpi wybór jednej z oznaczonych strzałek (1 lub 2 lub ... lub n) stosownie do wartości "którędy" oraz jednej z dwu nieoznaczonych strzałek - niedeterministycznie.

Proces możemy przedstawić w uproszczeniu jako



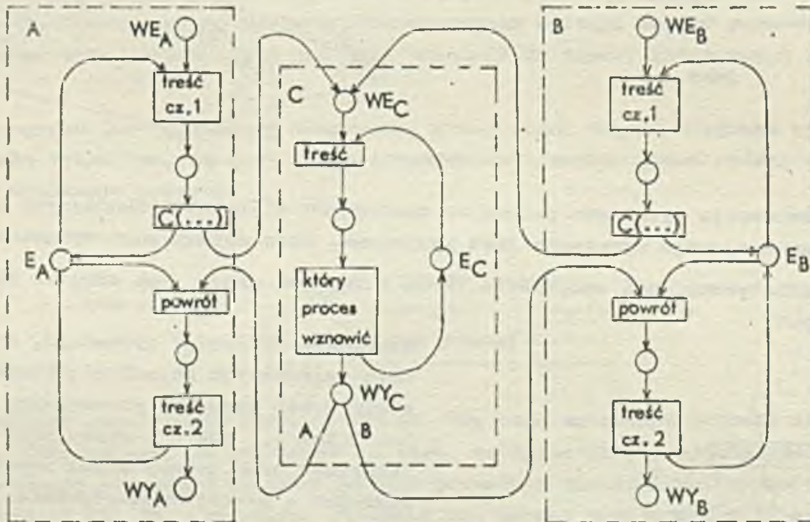
gdzie E oznacza miejsce służące wzajemnemu wykluczaniu się poszczególnych przebiegów procesu.

Rozważmy dwa procesy A i B takie, że A wywołuje B:



Z powyższego rysunku widać, że zarówno wywołanie B z wnętrza procesu A jak i zakończenie A powoduje umieszczenie kropki w E_A. Oznacza to, że może się rozpocząć następny przebieg A lub może nastąpić wznowienie poprzedniego przebiegu A. Zatem różne przebiegi jednego procesu spowodowane różnymi jego wywołaniami mogą być poprzeplatane z sobą w czasie, jeśli proces ten wywołuje inne procesy.

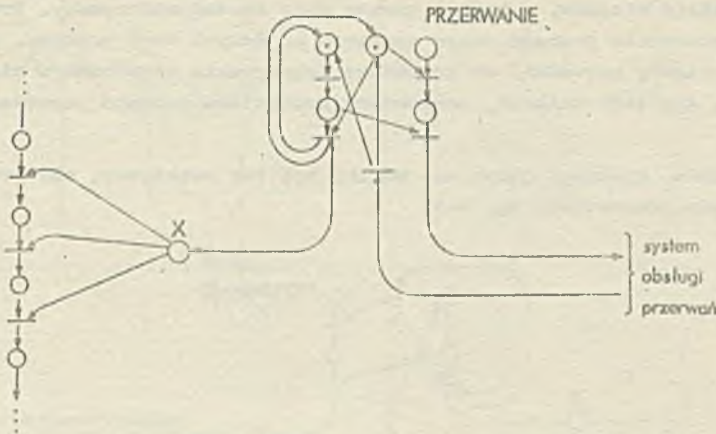
Rozważmy teraz trzy procesy A, B i C takie, że A i B wywołują C:



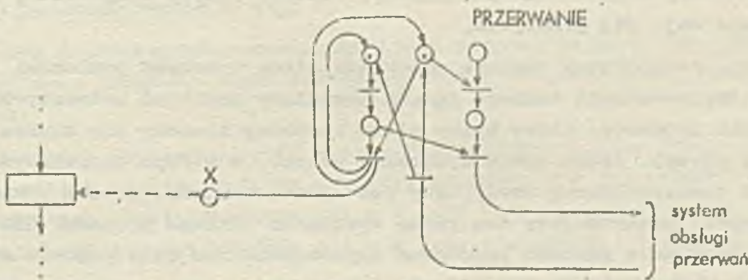
Rysunek ten wyjaśnia, co się dzieje, gdy w obu procesach, A i B, jednocześnie dochodzi do wywołania C. Wówczas dwie kropki pojawiają się w miejscu WE_C oraz po jednej w miejscach E_A i E_B. Oznacza to, że każdy z procesów A i B może być teraz zapoczątkowany lub wznowiony, oraz że nastąpią jeden po drugim dwa przebiegi procesu C. Nie wiadomo przy tym, które wywołanie C będzie najpierw wzięte pod uwagę.

Dotychczas przedstawione sieci Petri obrazują procesy, które nie dają się przerwać. Próba przedstawienia za pomocą tej metody procesów, które dają się przerywać sprawi, że sieci będą

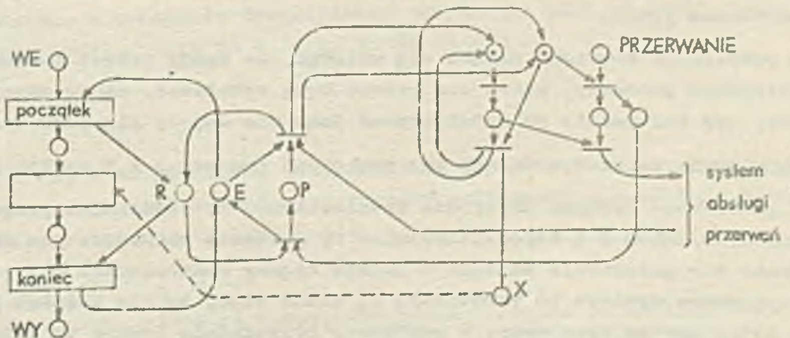
znacznie bardziej skomplikowane. W tym celu należy bowiem podzielić procesy na operacje, które są rzeczywiście niepodzielne (będziemy je oznaczać kreskami) i dla których istnieje pewne specjalne miejsce, powiedzmy X, stanowiące warunek wykonania każdej z niepodzielnych operacji procesu. Innymi słowy, obecność kropki w X przed wykonaniem kolejnej operacji będzie świadczyć, że od poprzedniej operacji przerwanie nie nastąpiło lub, że nastąpiło wznowienie tego procesu. Powiązanie miejsca X z systemem przerwań maszyny oraz z procesami obsługi przerwań można przedstawiać w różny sposób. A. Salwicki pokazał mi sposób, który przy założeniu tzw. semantyki MAX - - za każdym razem zostaje odpalona maksymalna liczba przejść w sieci [4] - pozwala obyć się bez wprowadzania przejść z różnymi priorytetami. Sposób ten wygląda tak:



Gdybyśmy w powyższy sposób mieli przedstawić w całości chociaż jeden proces, który daje się przerywać, a nie tylko jego trzy niepodzielne operacje, wówczas rysunek stałby się zbyt skomplikowany i mało czytelny. Dlatego też zastosujemy pewien skrót rysunkowy polegający na zastąpieniu całego ciągu niepodzielnych operacji jednym przejściem, które będzie połączone z miejscem X linią przerywaną. Sieć przedstawiona wyżej przyjmuje więc postać następującą:



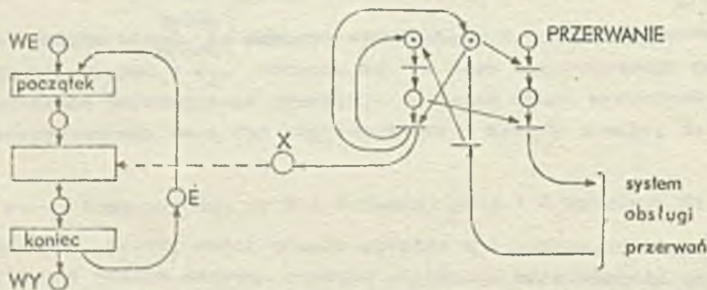
W takiej konwencji rysowania proces, który daje się przerywać, wraz z jego powiązaniami z systemem obsługi przerwań, możemy w całości przedstawić następująco:



Miejsce X jest tylko jedno dla całej struktury. Jest więc ono miejscem wspólnym dla wszystkich procesów, które dają się przerywać. Natomiast miejsca R, E i P są oddzielnymi, własnymi miejscami danego procesu. Miejsce E, jak uprzednio, służy wzajemnemu wykluczaniu się poszczególnych przebiegów tego procesu. Kropka w tym miejscu oznacza, że proces jest nieczynny i może być zapoczątkowany. Kropka w miejscu R oznacza, że proces jest czynny i ewentualne zapoczątkowanie go będzie wstrzymane aż do zakończenia jego bieżącego przebiegu lub do przerwania. Kropka w miejscu P oznacza, że proces został przerwany.

Rysunek powyższy wskazuje na pewne niebezpieczeństwo; gdy proces będzie przerwany, może on być ponownie zapoczątkowany. Ponieważ przerwanie następuje w momencie, który nie daje się z góry przewidzieć, nie możemy określić miejsca, w którym proces może zostać zatrzymany. Przypuśćmy, że miejsce to wypada przed zakończeniem pewnego ciągu operacji na danych tego procesu. Jeżeli ponowne zapoczątkowanie procesu miałoby prowadzić do ponownego wykonywania tego samego ciągu, to skutki byłyby katastrofalne. Aby tego uniknąć, powinniśmy proces ten uczynić procesem, który nie daje się przerwać.

Można teraz postawić pytanie, dlaczego język nie mógłby być tak określony, aby proces, który daje się przerywać, można było przedstawić np. tak:



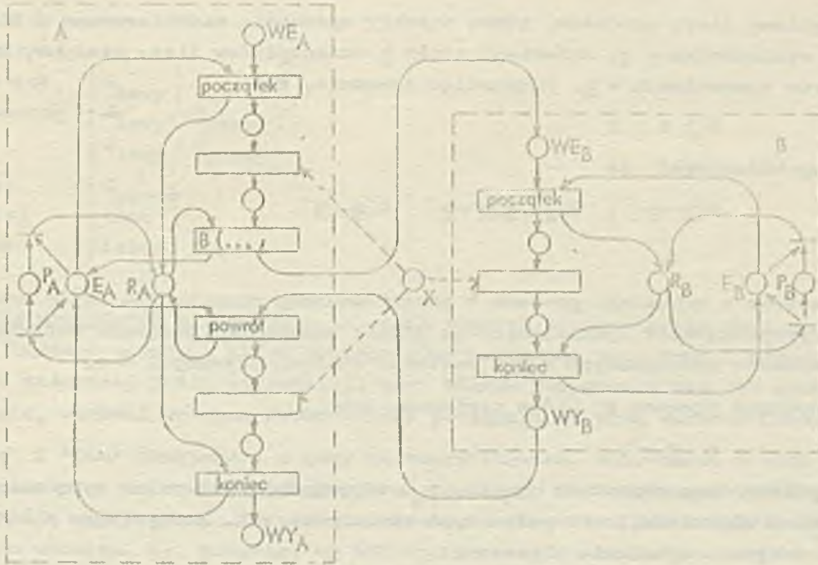
Oznaczałoby to, że proces, który jest przerwany, nie daje się zapoczątkować ponownie. A zatem niebezpieczeństwo, o którym mowa, wcale nie istniałoby. Procesy byłyby bezpieczne, niezależnie od tego, czy dają się czy nie dają się przerywać.

Nietrudno zorientować się, że powyższy rysunek odpowiada nieco wyższemu poziomowi języka programowania niż zamierzony. Implementacja takiego języka musiałaby zawierać automatyczny mechanizm opóźniania rozpoczynania procesów, który byłby nieco bardziej złożony niż zwykła pętla aktywnego czekania. Z drugiej strony, łatwo sobie wyobrazić proces, o którym chcielibyśmy z góry założyć, że będzie mógł być zapoczątkowany wtedy, gdy jest przerwany; np. proces wykonujący programy użytkowe - WPU. Zazwyczaj przewidujemy dwa różne wywołania takiego procesu: WPU (start,...) i WPU(stop,...). To drugie wywołanie powinno powodować zapoczątkowanie tego procesu również wtedy, gdy był on zapoczątkowany przez pierwsze wywołanie i następnie został przerwany. Gdyby to było niemożliwe, to nie moglibyśmy spowodować zakończenia programu użytkowego, w którym jest nieskończona pętla.

Z powyższych rozważań nasuwa się wniosek, że każdy proces powinniśmy szczegółowo zanalizować w kontekście procesów, które ten proces będą wywoływać, zanim zdecydujemy, czy może on być przerywany, czy też należy go zadeklarować jako nie dający się przerwać.

Rozpatrzmy teraz dwa dające się przerywać procesy A i B takie, że A wywołuje B:

Z poniższego rysunku nietrudno wywnioskować, że ewentualne przerwanie, które nastąpi w czasie między wywołaniem B i zapoczątkowaniem B, zostanie zaakceptowane dopiero po zapoczątkowaniu B. Podobnie gdy przerwanie nastąpi w czasie między zakończeniem B i powrotem do A, zostanie ono zaakceptowane dopiero po wznowieniu A. Widać więc, że dla każdego przerwania określony jest proces, który został przerwany. W powyższym przykładzie będzie to albo A albo B.



4. Phil w ujęciu aksjomatycznym

W dalszym ciągu niniejszej pracy pokażemy, na czym może polegać próba weryfikacji struktury procesów zaprogramowanej w Philu. W tym celu uzupełnimy aksjomatyczną definicję Pascala, podaną przez Hoare'a i Wirtha [3], o te elementy, które odpowiadają omówionym w poprzednich paragrafach pojęciom program, proces, wywołanie procesu oraz procedurom hold i resume. Wzorem Hoare'a i Wirtha będziemy stosować skrócony zapis tekstów stanowiących programy i deklaracje procesów oraz zdania wywołania procesów. Przykład programu zapiszemy jako

program R(A) **parameter** **record** X **end**; S.

W zapisie tym przez A oznaczyliśmy zawartość listy parametrów programu, przez X - listę pól rekordu parametrów, oraz przez S - blok stanowiący treść programu (zob. paragraf 2). Wprowadźmy dalsze oznaczenia:

- $\underline{a}$ - lista wszystkich identyfikatorów wyspecyfikowanych w A jako procesy,
- $\underline{b}$ - lista wszystkich identyfikatorów wyspecyfikowanych w A jako stałe,
- $\underline{x}$ - lista wszystkich identyfikatorów pól w X typu wskaźnik $(x_1,\dots,x_m)$,
- $\underline{o}$ - lista wszystkich identyfikatorów pól w X typu skalarnego $(o_1,\dots,o_n)$.

O treści programu S będziemy dla uproszczenia zakładać, że nie ma w niej podstawień na $x_i \in \underline{x}$ ani na $o_i \in o$. Założenie to pozostaje w mocy dla większości programów dowolnego systemu operacyjnego, aczkolwiek nie dla wszystkich. Np. w programie dynamicznego przydziału pamięci (zob. opracowanie [6] rodz. 2) mogą być właśnie podstawienia na $x_i \in \underline{x}$, a nie na $x_i\uparrow$.

Przyjmując, że

4.1 $\quad P\} \, S \, \{Q$

(w stosunku do oryginalnej notacji Hoare'a odwróciliśmy tutaj nawiasy klamrowe tak, aby predykaty P i Q, umieszczone w programach, miały postać komentarzy), możemy wnioskować o istnieniu takich funkcji $f_1,\dots,f_m$, które spełniają następującą implikację:

4.2 $\quad P \Rightarrow Q(f_1(\underline{x}\uparrow,\underline{o}) \to x_1\uparrow, \dots, f_m(\underline{x}\uparrow,\underline{o}) \to x_m\uparrow),$

gdzie przez $\underline{x}\uparrow$ oznaczyliśmy ciąg zmiennych, które są wskazywane przez wartości wskaźników tworzących listę $\underline{x}$. Funkcje $f_1,\dots,f_m$ można traktować jako odwzorowania wartości, jakie mają $\underline{x}\uparrow$ i $\underline{o}$ przed rozpoczęciem S, na wartości $\underline{x}\uparrow$ po zakończeniu S. Zapis $D(u \to v)$ oznacza, że wszystkie wolne wystąpienia v w D zastępujemy przez u.

Deklarację procesu r według programu R zapisujemy jako

$$\underline{process}\ r:R(\underline{g},\underline{h}),$$

gdzie przez $\underline{g}$ oznaczyliśmy listę procesów, które zostały uprzednio zadeklarowane i które odpowiadają identyfikatorom wymienionym w $\underline{a}$, natomiast przez $\underline{h}$ oznaczyliśmy listę stałych, które odpowiadają identyfikatorom wymienionym w $\underline{b}$. Przyjmując ponownie, że

$$P \} S \{ Q$$

dla programu R, możemy wnioskować, że

4.3 $$P(\underline{h}\rightarrow\underline{b}) \} \quad S(\underline{g}\rightarrow\underline{a},\underline{h}\rightarrow\underline{b}) \{ \quad Q(\underline{h}\rightarrow\underline{b})$$

dla procesu r.

Jak widać, wnioskowanie o przebiegu procesu r przeprowadzamy rozpatrując treść programu R, w której w miejsce identyfikatorów wymienionych na liście parametrów programu wstawiamy odpowiednio identyfikatory procesów wywoływanych w r i wartości stałych używanych w r.

Rozważmy teraz wywołanie procesu r, które zapiszemy jako

$$r\ (\underline{y},\underline{e}),$$

gdzie przez $\underline{y}$ oznaczyliśmy ciąg zmiennych $y_1,\ldots,y_m$ odpowiadających polom typu wskaźnika w X, a przez $\underline{e}$ - ciąg wyrażeń odpowiadających polom typu skalarnego w X. Korzystając z 4.1 i 4.2 formułujemy następujący aksjomat wywołania procesu r:

4.4 $$Q(f_1(\underline{y},\underline{e})\rightarrow y_1,\ldots,f_m(\underline{y},\underline{e})\rightarrow y_m)\} r(\underline{y},\underline{e})\{ Q \quad .$$

Dla procedury hold przyjmujemy:

4.5 $$P(l_o\rightarrow l)\} \text{hold}(l) \{ P \quad ,$$

gdzie przez l_o oznaczyliśmy wartość śladu procesu wywołującego. Natomiast dla procedury resume przyjmujemy:

$$P \wedge l=l_o \} \text{ resume}(l) \{ \text{ false} \quad .$$

Umieszczając $l=l_o$ w warunku wstępnym zaznaczyliśmy, że parametr procedury resume ma wartość i jest to wartość śladu jakiegoś procesu.

5. Próba weryfikacji struktury procesów na przykładzie pięciu filozofów

Przedstawienie problemu pięciu filozofów i jego rozwiązanie można znaleźć m.in. w opracowaniu [6]. Tutaj problem ten będziemy rozwiązywać od nowa tak, aby wraz z zapisem programów powstawał zapis warunków wstępnych i końcowych dla poszczególnych zdań. Systematycznie prowadzona weryfikacja struktury procesów zadeklarowanych wg tych programów powinna polegać na przepisaniu tekstu odpowiedniego programu dla każdej deklaracji procesu z odpowiednimi podstawieniami we wzorze 4.3 w poprzednim punkcie. W naszym przykładzie prowadziłoby to do pięciokrotnego przepisywania programów filozofa i widelca. Zastosujemy więc pewne skróty myślowe, które pozwolą skrócić zapis weryfikacji.

Rozwiązując problem pięciu filozofów i prowadząc weryfikację rozwiązania spróbujemy sprawdzić, czy weryfikacja ta pomoże nam wykryć ewentualny błąd i czy da wskazówki, co do jego usunięcia. Zatem celowo nie weźmiemy pod uwagę np. tego, że filozofom grozi blokada (deadlock).

W weryfikacji posłużymy się pewnymi zmiennymi boolowskimi G_i przypisanymi widelcom, którymi filozofowie posługują się przy jedzeniu. Wartość zmiennej G_i wskazuje, czy i-ty widelec jest podniesiony (true), czy też leży na stole (false).

Program, wg którego toczy się żywot każdego z filozofów, uzupełniony predykatami, może wyglądać jak następuje:

```
program FILOZOF(process LEWY,PRAWY);
type koodW= (podnoszę, kładę, inicW);
procedure myślę;
begin ...  end;
```

```
procedure jem;
begin ... end;
begin repeat          {true}
      myślę;          {true}
      LEWY(podnoszę);      {G_lewy}
      PRAWY(podnoszę);     {G_lewy ∧ G_prawy}
      jem;                 {G_lewy ∧ G_prawy}
      LEWY(kładę);         {G_prawy}
      PRAWY(kładę);        {true}
      until false          {false}
end.
```

Powyższy zapis predykatów w programie filozofa stanowi niejako wstęp do zamierzonej weryfikacji. Teraz dla każdego ze zdań, które opatrzyliśmy warunkami wstępnymi i końcowymi, powinniśmy wykazać, że po pierwsze, jeśli wykonywanie tego zdania rozpocznie się, to również i zakończy się, oraz po drugie, warunki wstępne rzeczywiście pociągają za sobą warunki końcowe.

Zdania "myślę" i "jem" przyjmiemy z góry za zweryfikowane, tzn. każde z nich rozpoczęte - zakończy się i spowoduje, że warunek wstępny true pociągnie za sobą warunek końcowy true. Do weryfikacji pozostają zatem zdania stanowiące wywołania procesów LEWY i PRAWY. W tym celu musimy rozpatrzyć program widelca, tj. program, wg którego procesy te zostaną zadeklarowane, a także deklaracje procesów odpowiadających poszczególnym filozofom. Przed deklaracje te następuje bowiem przydział poszczególnych widelców poszczególnym filozofom.

Program widelca, uzupełniony predykatami, zapiszemy jak następuje:

```
prefix type kodW= (podnoszę, kładę, inicW);
program WIDELEC(process S) parametr record op: kodW end;
type kodS=(wstrzymaj, kontynuuj, inicS);
var p,k:link;
    G: boolean;
begin case op of
podnoszę: begin  {true => p_j=p_j}
          hold(p);   {p=p_j}
          if G then   {G ∧ p=p_j}
          S(wstrzymaj,p);  {¬G ∧ p=p_j}
          G:=true;   {G ∧ p=p_j}
          resume(p)  {false}
          end;
kładę:    begin   {G ∧ p=p_j   =>   G ∧ p=p_j ∧ k_j=k_j}
          hold(k) {G ∧ p=p_j ∧ k=k_j}
          G:=false; {¬G ∧ p=p_j ∧ k=k_j}
          S(kontynuuj,p,k) {false}
          end;
inicW:    begin  {true  =>  p_o=p_o}
          hold(p);  {p=p_o}
          G:=false;  {p=p_o ∧ ¬G}
          resume(p) {false}
          end
end end.
```

Przez p_o oznaczyliśmy tutaj wartość śladu tego procesu, który zainicjuje proces zadeklarowany wg powyższego programu, czyli procesu, który wywoła wariant inicW; przez p_j - wartości śladów procesów, które będą wywoływać wariant podnoszę, k_j - procesów, które będą wywoływać wariant kładę.

W programie tym następują wywołania pewnego procesu szeregującego S. Zauważmy, że za pomocą predykatów stanowiących warunki wstępne i końcowe wywołań S zapisujemy niejako znaczenia tych wywołań. Np. przez

$$G \wedge p=p_j \} \quad S(wstrzymaj,p); \quad \{ \neg G \wedge p=p_j$$

wyrażamy to, że gdy widelec jest podniesiony (G) i ślad procesu-filozofa próbującego podnieść ten widelec jest zapisany w p, następuje wstrzymanie procesu, który toczy się wg powyższego programu, oraz to, że zostanie on wznowiony, gdy widelec leży (¬G) i ślad procesu-filozofa jest nadal zapisany w p. Ów proces-filozof jest więc tym, ze względu na który nastąpi wstrzymanie. A przez

$$\neg G \wedge p=p_j \wedge k=k_j \} \quad S(kontynuuj,p,k) \quad \{ false$$

wyrażamy to, że gdy widelec leży (¬G), w p jest zapisany ślad procesu-filozofa, który ostatni podnosił widelec i w k - ślad tego, który właśnie kładzie widelec; następuje przekazanie obu tych śladów procesowi S i na tym kończy się proces, którego program rozpatrujemy.

Widać zatem, że warunki wstępne i końcowe wywołań procesu S są zapisem tego, co ten program ma robić. Natomiast to, w jaki sposób będzie on to robić, będzie zapisem jego programu. Przyjmijmy, że program ten ma postać następującą:

```
prefix type kodS=(wstrzymaj, kontynuuj, inicS);
program SZER parameter record case op:kodS of
                          wstrzymaj: (a:↑link);
                          kontynuuj: (b,c:↑link);
                          inicS:(f0,f1,f2,f3,f4:↑link) end;
var KPTL: array [0..4] of link; {kolejka "kto pierwszy ten lepszy"}
    p,k:0..4;   {początek i koniec KPTL}
    CZ: array [0..3] of record l1,l2:link;
                              e: boolean end;
                  {tabelka śladów procesów czekających: l1 - ślad procesu, ze względu na który
                   nastąpiło wstrzymanie, l2 - ślad procesu wstrzymanego, e - czy miejsce jest
                   wolne}
    l:link; i:0..4; d:boolean; {zmienne pomocnicze}

procedure umieść (w:link); {na końcu kolejki KPTL}
begin KPTL[k]:=w;   k:=(k+1)mod5 end;

procedure pobierz(var w:link);   {z początku kolejki KPTL}
begin w:=KPTL[p];   p:=(p+1)mod5 end;

begin case op of

wstrzymaj: begin i:=0; while ¬CZ[i].e do i:=i+1;
                 {pierwsze wolne miejsce w CZ odnalezione}
                 CZ[i]. l1:=a↑; {ślad procesu, ze względu na który nastąpiło wstrzymanie, za-
                 pamiętany}
                 hold(CZ[i].l2 ; {ślad procesu wstrzymanego zapamiętany}
                 CZ[i].e:=false {zaznaczenie, że to miejsce jest zajęte}
           end;

kontynuuj: begin i:=0;
           repeat if ¬CZ[i].e & CZ[i]. l1=b↑  then {ślad procesu, ze względu na który było
                  wstrzymanie, odnaleziony}
           begin umieścić(CZ[i].l2); {w kolejce KPTL umieszczony ślad procesu, który był
                  wstrzymany}
                  d:=true end
           else begin d:=false;   i:=i+1 end
           until d | i=4;
           umieść (c↑)end;
```

```
inicS:    begin for i:=0 to 3 do CZ[i].e:=true;   {tabelka czekających zapoczątkowana jako
                        pusta - wszystkie miejsca wolne}
          KPTL[0]:= f0↑ ;  KPTL[1]:=f1↑;  KPTL[2]:=f2↑;
          KPTL[3]:=f3↑;   KPTL[4]:=f4↑; {kolejka KPTL zapełniona śladami wskazywanymi przez f0,
                        f1, f2, f3 i f4}
          end
          end;
pobierz(1);  resume(1) {wznowiony pierwszy z kolejki KPTL}
end.
```

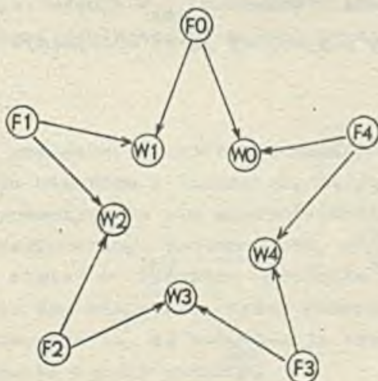
W programie tym nie zawarliśmy formalnego zapisu weryfikacji, gdyż jest on nieco bardziej złożony niż w programach filozofa i widelca. Zamiast tego mamy zwykłe komentarze, które mają pomóc w zrozumieniu treści programu. Nie chodzi bowiem o to, aby zweryfikować w sposób formalny całość, tylko pokazać jak weryfikację można prowadzić.

Załóżmy zatem, że powyższy program jest już zweryfikowany i powróćmy do programu filozofa. Zawarte w nim zdania LEWY (kładę) i PRAWY(kładę) dadzą się zweryfikować z łatwością, i to niezależnie od tego, jak zostaną zadeklarowane procesy filozofów i widelców. Nietrudno bowiem w programie widelca stwierdzić, że każde wywołanie wariantu kładę kończy się wywołaniem S i przekazaniem odpowiednich śladów procesów do wznowienia. Natomiast zdania LEWY(podnoszę) i PRAWY(podnoszę) pozostają nadal niezweryfikowane, ponieważ niezweryfikowane pozostało zdanie S(wstrzymaj, p) w programie widelca.

Na podstawie samych programów filozofa i widelca nic już więcej nie da się powiedzieć o zdaniach LEWY(podnoszę), PRAWY(podnoszę) i S(wstrzymaj, p). Pozostaje zatem rozważyć strukturę, którą będą tworzyć procesy filozofów i widelców, określoną przez ich deklaracje. Rozważmy więc strukturę powstałą np. przez następujące deklaracje (o procesie S zakładamy, że został już zadeklarowany):

```
proces W0,W1,W2,W3,W4: WIDELEC(S);
       F0:FILOZOF(W0,W1);
       F1:FILOZOF(W1,W2);
       F2:FILOZOF(W2,W3);
       F3:FILOZOF(W3,W4);
       F4:FILOZOF(W4,W0);
```

Deklaracje te oznaczają, że i-ty filozof posługuje się i-tym widelcem jako lewym oraz i⊕1-ym jako prawym (przez ⊕ oznaczyliśmy dodawanie modulo 5). Ilustrację takiej struktury procesów stanowi następujący rysunek (zob. konwencję rysowania struktur procesów w opracowaniu [6]):



Spróbujmy teraz sprawdzić, czy i-temu filozofowi może się zdarzyć, że nie będzie mógł podnieść lewego widelca, tj. czy zdanie W_i(podnoszę) w procesie F_i może się nie skończyć (przy deklaracji procesu F, dokonaliśmy zamiany identyfikatora LEWY na W_i) . Załóżmy, że istotnie; zdanie W_i(podnoszę) w procesie F_i nie kończy się. Wynika z tego, że G_i zachodziło już przy wywołaniu W_i(podnoszę) w procesie F_i i że zdanie S(wstrzymaj,p) w procesie W_i nie kończy się.

Wnioskujemy zatem, że w procesie $F_{i⊕1}$ zakończyło się zdanie W_i(podnoszę) – i⊕1-szy filozof podniósł swój prawy widelec. Dalej w procesie $F_{i⊕1}$ nastąpią zdania jem, $W_{i⊕1}$(kładę) i W_i (kładę). O każdym z tych zdań wiemy, że się zakończy. Wiemy również, że wywołanie W_i(kładę) doprowadzi w procesie W_i do spełnienia warunku $\neg G_i$ i do wywołania S(kontynuuj 2,p,k), a więc i do zakończenia zdania S(wstrzymaj,p). Skoro zakończy się zdanie S (wstrzymaj, p) w procesie W_i, to zakończy się również zdanie W_i(podnoszę) w procesie F_i, mimo iż próbowaliśmy dowieść coś wręcz przeciwnego.

Sprawdźmy teraz, czy i-temu filozofowi zawsze uda się podnieść jego prawy widelec, tzn, czy w procesie F_i rozpoczęte zdanie $W_{i⊕1}$(podnoszę) zawsze zakończy się. Załóżmy, że nie; zdanie $W_{i⊕1}$(podnoszę) w procesie F_i nie kończy się. Podobnie jak poprzednio oznacza to, że $G_{i⊕1}$ zachodziło już przy wywołaniu $W_{i⊕1}$(podnoszę) w procesie F_i, i że zdanie S(wstrzymaj,p) w procesie $W_{i⊕1}$ nie kończy się. Wniosek z tego, że w procesie $F_{i⊕1}$ zakończyło się zdanie $W_{i⊕1}$(podnoszę) – i⊕1-szy filozof podniósł swój lewy widelec. Dalej w procesie $F_{i⊕1}$ mają kolejno nastąpić zdania $W_{i⊕2}$(podnoszę), jem, $W_{i⊕1}$(kładę) i $W_{i⊕2}$(kładę). Powstaje więc pytanie, czy w procesie $F_{i⊕1}$ rozpoczęte zdanie $W_{i⊕2}$(podnoszę) może nie zakończyć się. Nietrudno spostrzec, że jest to pytanie analogiczne do postawionego na początku, w stosunku do procesu F_i. Innymi słowy, chcąc odpowiedzieć na pytanie, czy i-temu filozofowi zawsze uda się podnieść jego prawy widelec, musimy odpowiedzieć na pytanie, czy i⊕1-szemu filozofowi zawsze uda się podnieść jego prawy widelec. Czyniąc kolejno założenie, że to się nie uda filozofowi i⊕1, dochodzimy do tego samego pytania w stosunku do filozofa i⊕2, itd. A więc zakładając, że zdanie $W_{i⊕1}$(podnoszę) w procesie F_i nie kończy się dla kolejnych $i \in [0,4]$, wykazujemy słuszność tych założeń. Tym samym dochodzimy do wniosku, że możliwa jest wzajemna blokada wszystkich pięciu filozofów.

Jak zatem usunąć błąd, który wykryliśmy w strukturze procesów określonej powyższymi deklaracjami? Można wzorem Dijkstry [6] niedopuścić do stołu wszystkich naraz filozofów, tylko co najwyżej czterech. Można także niektórym filozofom zmienić zwyczaje, tzn. sprawić, żeby podnosili widelce w odwrotnym porządku w stosunku do reszty filozofów. Rozważmy np. strukturę procesów powstałą przez następujące deklaracje:

```
proces W0,W1,W2,W3,W4: WIDELEC (S);
       F0:FILOZOF(W0,W1);
       F1:FILOZOF(W1,W2);
       F2:FILOZOF(W3,W2);
       F3:FILOZOF(W3,W4);
       F4:FILOZOF(W4,W0);
```

Filozof nr 2 zachowuje się inaczej niż pozostali; najpierw podnosi widelec nr 3 traktując go jako swój lewy, potem widelec nr 2 jako swój prawy.

Dla uzmysłowienia sobie tej struktury narysujemy ją ze specjalnym uwzględnieniem "odwrotnego" umiejscowienia w niej procesu F2.

Dla tak określonej struktury procesów sprawdźmy, czy dla $i \in [0,4]$ zdania W_i(podnoszę) i $W_{i⊕1}$(podnoszę) w procesie F_i mogą nie zakończyć się.

Dla i=2 porządek wykonywania tych zdań jest odwrócony; najpierw W_3(podnoszę), potem W_2 (podnoszę). Nietrudno spostrzec, że dla i=0,1,4 zdanie W_i(podnoszę) w procesie F_i zawsze zakończy się. Rozumowanie prowadzące do tego wniosku jest takie samo jak dla struktury rozważanej poprzednio.

Nieco inaczej rzecz się ma dla i=2,3. Załóżmy zatem, że w procesie F2 zdanie W3(podnoszę) nie kończy się. Oznacza to, że G_3 , i że w procesie F3 zakończyło się zdanie W3(podnoszę). Dalej w procesie F3 następuje zdanie W4(podnoszę). Zakładając, że ono także nie kończy się, dochodzimy do wniosku, że G_4 , i że w procesie F4 zakończyło się zdanie W4(podnoszę). Powtarzając to samo rozumowanie dla F4, Fo i F1 dochodzimy do pytania, czy w procesie F1 zdanie W2(podnoszę) może nie zakończyć się. Otóż nie; ponieważ w procesie F2 nadal pozostaje nie zakończone zdanie W3(podnoszę), nie doszło jeszcze do rozpoczęcia zdania W2(podnoszę) i $\neg G_2$. Zatem zdanie W2(podnoszę) w procesie F1 zakończy się, a co za tym idzie, zostaną wykonane również zdania jem, W1(kładę) i W2(kładę). To pociągnie za sobą zakończenie zdania W1(podnoszę) w procesie F0. Prowadząc analogiczne rozumowanie dla F0, F4 i F3, dochodzimy do wniosku, że zdanie W3(podnoszę) w procesie F2 nie może się nie zakończyć.

Załóżmy teraz, że zdanie W2(podnoszę) w procesie F2 nie kończy się. Oznacza to, że G_2, i że w procesie F1 zakończyło się zdanie W2(podnoszę). Dalej w procesie F1 następują zdania jem, W1(kładę) i W2(kładę). To ostatnie zdanie doprowadzi do $\neg G_2$ i do zakończenia zdania W2(podnoszę) w procesie F2.

Załóżmy dalej, że w procesie F3 zdanie W3(podnoszę) nie kończy się. Oznacza to, że G_3, i że w procesie F2 zakończyło się zdanie W3(podnoszę). Nieco powyżej wykazaliśmy, że zdanie W2(podnoszę) w procesie F2 również zakończy się, a zatem nastąpią w nim zdania jem i W3(kładę). Wykonanie tego ostatniego zdania sprawi, że $\neg G_3$, i że zdanie W3(podnoszę) w procesie F3 zakończy się.

Załóżmy z kolei, że w procesie F3 zdanie W4(podnoszę) nie kończy się. Oznacza to, że G_4, i że zdanie W4 (podnoszę) w procesie F4 zakończyło się. Dalej w procesie F4 następuje zdanie W0(podnoszę). Zakładając o tym zdaniu, że się ono nie kończy, i prowadząc analogiczne rozumowanie dla procesów F0 i F1, dochodzimy do pytania, czy zdanie W2(podnoszę) w procesie F1 może się nie zakończyć. Otóż nie; niezakończone zdanie W2(podnoszę) w procesie F1 oznacza, że G_2, i że zdanie W2(podnoszę)zakończyło się w procesie F2. Dalej w procesie F2 następują zdania: jem, W3(kładę) i W2(kładę). Wykonanie tego ostatniego zdania sprawia, że $\neg G_2$ i że zdanie W2(podnoszę) w procesie F1 zakończy się. Skoro tak, to również w procesie F0 zakończy się zdanie W1 (podnoszę), ..., i w procesie F3 zakończy się zdanie W4(podnoszę).

Rozpatrując szczególne przypadki procesów F2 i F3, niejako przy okazji wykazaliśmy, że dla pozostałych procesów F_i (i=0,1,4) zdanie $W_{i\oplus 1}$(podnoszę) nie może nie zakończyć się. Zatem strukturę procesów określoną powyższymi deklaracjami możemy uważać za zweryfikowaną.

6. Podsumowanie

Próba weryfikacji struktury procesów, przeprowadzona w poprzednim paragrafie na przykładzie pięciu filozofów, pokazuje tylko niektóre z trudności, na jakie natrafi ten, kto zechce zweryfikować system operacyjny. Przypomnijmy, że nie przeprowadziliśmy weryfikacji procesu szeregowania S, uznając go z góry za zweryfikowany. Zauważmy też, że parametrami wywołań procesów w rozważonym przykładzie były tylko stałe, co istotnie uprościło weryfikację. W jakimkolwiek systemie operacyjnym będziemy mieć do czynienia z bardziej skomplikowanymi strukturami procesów niż w przedstawionym przykładzie. Oznacza to, że weryfikacja takiego systemu będzie trudniejsza, choć sam sposób jej przeprowadzenia będzie podobny.

Jednakże pozostaje jeszcze jeden nie poruszony aspekt weryfikacji. Otóż w poprzednim paragrafie milcząco przyjęliśmy założenie, że ewentualne przerwania nie mają wpływu na to, co się dzieje w rozważanej strukturze. Innymi słowy, założyliśmy, że jeśli któryś z procesów będzie przerwany, to zostanie on również wznowiony od miejsca przerwania, przy czym między przerwaniem i wznowieniem nie nastąpi wywołanie tego procesu. Dla pełnej weryfikacji rozważanej struktury należałoby wykazać, że założenie to jest spełnione. W tym celu konieczna jest znajomość całej

struktury danego systemu operacyjnego. W strukturze tej znajdą się procesy reakcji na przerwania oraz procesy, które są bezpośrednio lub pośrednio wywoływane przez procesy reakcji na przerwania. Jeśli wśród procesów w ten sposób wywoływanych są również te, o których uczyniliśmy powyższe założenie, to mamy błąd w rozważanej strukturze. Błąd ten może polegać na tym, że procesy, o których mowa, nie zostały zadeklarowane jako nie dające się przerwać, chociaż powinny być tak zadeklarowane. Może to być również błąd poważniejszy, tzn. taki, którego usunięcie będzie polegać na zmianie struktury procesów.

Podziękowanie

Wyrażam podziękowanie A. Salwickiemu i T. Müldnerowi za owocną dla mnie dyskusję nad treścią niniejszej pracy i uwagi, które pozwoliły poprawić przedstawiony tekst.

Literatura

[1] Brinch-Hansen P.: Concurrent Pascal report. Cal. Inst. of Technology, 1975

[2] Hoare C.A.R.: Monitor: an operating system structuring concept. CACM 1974 t. 17, nr 10, s. 554-557

[3] Hoare C.A.R., Wirth N.: An axiomatic definition of the programming language Pascal. Acta Inf. 1973, nr 2, s. 335-355

[4] Müldner T., Salwicki A.: On algorithmic properties of concurrent programs. W druku

[5] Olszewski J.: Machine-oriented version of Pascal: a proposal. Reading University report, RCS 1978, nr 100

[6] Olszewski J.: Projektowanie struktur systemów operacyjnych. Warszawa: WNT 1981

[7] Peterson J.L.: Petri nets. Computing Surveys 1977 t. 9, nr 3, s. 223-252

[8] Wirth N.: Modula: a language for modular multiprogramming. S-P&E 7, nr 1, s. 3-36.

Biuletyn Informacyjny OBIEKTOWE SYSTEMY KOMPUTEROWE Nr 5-6/1981

dr Andrzej ROWICKI
Instytut Maszyn Matematycznych

Pewne metody interpolacyjne i aproksymacyjne stosowane w systemach sterowania numerycznego

Wstęp

W latach sześćdziesiątych w związku z rozwojem techniki lotniczej i kosmicznej, w celu zapewnienia dużej dokładności i powtarzalności wyrobów zaczęto stosować obrabiarki sterowane numerycznie. Systemy te były kosztowne i wiele podzespołów było realizowanych na drodze analogowej. Ostatnio, w związku ze znacznym rozwojem techniki cyfrowej istnieje tendencja do zastępowania układów analogowych cyfrowymi, co pozwala zwiększyć dokładność i elastyczność systemu.

Obecnie w CNPTKiP wdrożony został do produkcji nowoczesny system MERA-CNC/NUCON-400 o budowie modularnej, przeznaczony do sterowania różnego typu obrabiarkami i urządzeniami wchodzącymi w skład systemów OSN. W odróżnieniu od klasycznych systemów sterowania numerycznego (NC) system MERA-CNC/NUCON-400 nie może być traktowany jako system o zamkniętej strukturze i w zależności od potrzeb może być on rozszerzany.

W związku z szeroko zakrojonym programem rozwoju postlicencyjnego */ systemu MERA-CNC/NUCON-400, którego celem jest zwiększenie możliwości systemu oraz modernizacja sprzętu, wydaje się celowe podjęcie pewnych prac podstawowych, w wyniku których uzyskana będzie możliwość prowadzenia porównawczej analizy różnych rozwiązań układowych.

W pierwszej części pracy dokonano ogólnego wprowadzenia w problematykę oraz podano podstawy matematyczne. Następnie dokonano przeglądu znanych metod interpolacji i aproksymacji stosowanych do systemów sterowania numerycznego. Przegląd ten nie wyczerpuje całości zagadnienia, m.in. pominięto metody przyrostowe, gdyż w sposób jawny nie występuje w nich zależność od wymiarów narzędzia skrawającego. Położono głównie nacisk na aspekty matematyczne funkcji realizowanych przez bloki odpowiedzialne za wyznaczenie toru narzędzia. Ze względu na duże znaczenie praktyczne oraz charakter obrabianych przedmiotów, szczególny nacisk położono na aproksymację liniową okręgu, oraz na kompensację wpływu średnicy narzędzia (frezu) na dokładność obrabianego przedmiotu. Omówiono również przybliżone rozwiązanie interpolatora kołowego, odbiegające od konwencjonalnych rozwiązań. Na zakończenie omówiono ważne zagadnienie analizy i wyznaczania błędów.

Wydaje się, że pracę można traktować jako zwięzłe wprowadzenie w problematykę związaną ze stosowaniem interpolacji i aproksymacji w systemach sterowania numerycznego.

• Interpolacja

Uwagi wstępne

Przy sterowaniu numerycznym obrabiarek występuje m.in. problem wyznaczania toru narzędzia dla uzyskania odpowiedniego kształtu przedmiotu obrabianego. Jeżeli kształt przedmiotu, który należy uzyskać w wyniku obróbki, jest zadany analitycznie to sprawa wyznaczania toru narzędzia się upraszcza. Jednakże w większości wypadków podane są tylko pewne charakterystyczne punkty oraz ogólna charakterystyka pożądanego kształtu lub jego fragmentów. W związku z tym istnieje konieczność stosowania metod analitycznych i interpolacji do wyznaczania toru narzędzia. Ogólnie rzecz biorąc, wyznaczanie toru narzędzia można podzielić na dwie fazy, tj. wyznaczenie współrzędnych dla punktów nieciągłości konturu oraz wyznaczanie współrzędnych dla fragmentów toru zawierających nieciągłości.

---

*/ Program rozwoju postlicencyjnego MERA-CNC/NUCON-400.
Ośrodek Badawczo-Rozwojowy Technik Komputerowych i Pomiarów, Warszawa, grudzień 1977.

Interpolację najczęściej stosujemy w następujących sytuacjach:

- gdy nie znamy postaci analitycznej funkcji opisującej wymagany kształt przedmiotu obrabianego oraz znamy wartość tej funkcji tylko dla skończonego zbioru argumentów,
- gdy analityczna postać funkcji opisującej wymagany kształt jest znana, jednakże wyliczanie jej wartości wymaga dużego nakładu pracy obliczeniowej ze względu na skomplikowaną postać tej funkcji.

Mówiąc nieco precyzyjniej, przez interpolację będziemy rozumieli postępowanie prowadzące do znalezienia wartości pewnej funkcji F dla dowolnego argumentu x leżącego w przedziale (x_o, x_n) jeżeli znane są jej wartości dla argumentów x_o x_1 .... x_n. Ogólnie rzecz biorąc, zagadnienie interpolacji wprowadza się do rozwiązania poniższego problemu.

Dane są wartości funkcji F dla skończonego podzbioru argumentów $\{x_o, x_1, .., x_n\}$. Należy skonstruować funkcję f (dostatecznie prostą dla celów obliczeniowych) spełniającą następujący warunek

$$f(x_i) = F(x_i) \qquad \text{dla} \quad i = 0,1,....,n \tag{1}$$

Funkcję f będziemy nazywali funkcją interpolacyjną, a argumenty $x_o, x_1 ... x_n$ węzłami interpolacji. Natomiast funkcję F będziemy nazywali funkcję interpolowaną.

Tak sformułowane zagadnienie interpolacji nie ma oczywiście jednoznacznego rozwiązania. Rozwiązanie zagadnienia staje się jednoznaczne, jeżeli dodatkowo założyć, że funkcja interpolująca f jest wielomianem co najwyżej stopnia n, tzn., że funkcja f jest opisana następującym wzorem:

$$f(x) = a_o + a_1 x + a_2 x^2 + ... + a_n x^n \tag{2}$$

Przyjęcie założeń (1) i (2) sprowadza rozwiązanie problemu do rozwiązania układu równań:

$$\begin{aligned} a_o + a_1 x_o + a_2 x_o^2 + ... + a_n x_n^n &= F(x_o) \\ a_o + a_1 x_1 + a_2 x_1^2 + ... + a_n x_1^n &= F(x_1) \\ \dots\dots\dots\dots\dots\dots&\dots\dots \\ a_o + a_1 x_n + a_2 x_n^2 + ... + a_n x_n^n &= F(x_n) \end{aligned} \tag{3}$$

z których wyznaczamy współczynnik a_i wielomianu (2).

Z założenia (1) wynika, że wyznacznik Van der Monde'a dla układu równań (3) jest różny od zera, a więc układ równań ma jednoznaczne rozwiązanie.

W rzeczywistości nie ma potrzeby bezpośredniego rozwiązywania układu równań (3) aby uzyskać wielomian interpolacyjny. Są znane inne bezpośrednie metody uzyskiwania wielomianów interpolacyjnych. Najbardziej są znane metody oparte na wielomianach Lagrange'a i różnicach Newtona.

- Wielomian interpolacyjny Lagrange'a

Wielomian interpolacyjny Lagrange'a jest określony następującym wzorem:

$$L_n(x) = F(x_o)\, h_o(x) + F(x_1)\, h_1(x) + ... + F(x_n)\, h_n(x) \tag{4}$$

gdzie $h_i(x)$ jest wielomianem stopnia n określonym w sposób następujący:

$$h_i(x) = \frac{(x-x_o)(x-x_1)...(x-x_{i-1})(x-x_{i+1})...(x-x_n)}{(x_i-x_o)(x_i-x_1)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)} \tag{5}$$

spełniającym następujący warunek

$$h_i(x_j) = \begin{cases} 0 & \text{dla } j \neq i \\ 1 & \text{dla } j = i \end{cases} \tag{6}$$

Z warunku (6) wynika, że wielomian (4) dla węzłów interpolacji przyjmuje wartości identyczne z wartościami funkcji interpolowanej, co oznacza, że jest spełnione założenie (1). Natomiast

w punktach różnych od węzłów interpolacja może przyjmować wartości różniące się od wartości funkcji interpolowanej. Wynika więc potrzeba oszacowania różnicy między funkcją interpolowaną F a jej wielomianem interpolującym L_n.

Można pokazać, że w przedziale dotkniętym (x_o, x_n) zachodzi następująca relacja

$$F(x) = L_n(x) + R_n(x) \tag{7}$$

gdzie reszta R_n jest określona wzorem

$$R_n(x) = \frac{F^{n+1}(z)}{(n+1)}(x-x_o)(x-x_1)\dots(x-x_n) \tag{8}$$

oraz z jest liczbą zawartą między najmniejszą i największą z liczb $x_o, x_1, \dots, x_n$.

Oczywiście, aby oszacować resztę należy przyjąć taką wartość zmiennej z aby (8) miało wartość maksymalną.

Reszta R_n określa oczywiście błąd, jaki popełniamy, zastępując funkcję F wielomianem interpolacyjnym L_n. Łatwo zauważyć, że wartość reszty zależy od wartości wielomianu

$$W_n(x) = (x-x_o)(x-x_1)\dots(x-x_n) \tag{9}$$

Wartość wielomianu (9) można zmieniać przez zmianę węzłów x_i w danym przedziale (a,b). Można postawić zadanie takiego doboru węzłów aby

$$\sup_{x\in(a,b)} |W_n(x)| = \min.$$

Postawione zadanie można rozwiązać wykorzystując wielomiany Czebyszewa.

Z przeprowadzonych tu rozważań wynika, że narzucenie pewnych warunków na dokładność interpolacji (reszta R_n) jest możliwe, jeżeli znamy wartość funkcji interpolowanej nie tylko w węzłach interpolacji, ogólnie rzecz biorąc, ale i wtedy gdy funkcja interpolowana jest całkowicie określona w pewnym przedziale skończonym. Jeżeli funkcja jest określona w pewnym przedziale, to stosując metody aproksymacyjne można narzucić silniejsze warunki na dokładność interpolacji. Stosując aproksymację wielomianami Czebyszewa do funkcji interpolowanej, uzyskujemy nie tylko najlepsze przybliżenia średniokwadratowe ale również dobre wyniki w sensie przybliżenia jednostajnego. Do rozwiązania tego zagadnienia można również stosować aproksymację metodą najmniejszych kwadratów.

Przeprowadzone rozważania dla interpolacji metodą wielomianów Lagrange'a mają charakter bardziej ogólny, odnoszą się również do innych metod interpolacji. Po tych ogólnych rozważaniach dotyczących interpolacji omówimy jeszcze interpolację metodą ilorazów różnicowych Newtona.

- Metoda ilorazów różnicowych Newtona

Niech F będzie funkcją interpolowaną, określoną w $n+1$ różnych węzłach $x_o, x_1, \dots, x_n$. Wprowadzimy dla $n \geqslant i \geqslant m$ operator różnicowy D^m zdefiniowany indukcyjnie w sposób następujący:

1° $$D^1 F(x_i) = \frac{F(x_i) - F(x_{i-1})}{x_i - x_{i-1}}$$

2° $$D^{m+1} F(x_i) = \frac{D^m F(x_i) - D^m F(x_{i-1})}{x_i - x_{i-m-1}}$$

Wartość operatora D^m będziemy nazywali ilorazem różnicowym m-tego rzędu.

Tablicę ilorazów różnicowych zapisuje się zazwyczaj w następującej postaci:

x	$F(x)$	D^1	D^2	D^3	D^4	D^5
x_0	$F(x_0)$					
		$D^1(F x_1)$				
			$D^2 F(x_2)$			
x_1	$F(x_1)$					
		$D^1 F(x_2)$		$D^3 F(x_3)$		
x_2	$F(x_2)$		$D^2 F(x_3)$		$D^4 F(x_4)$	
		$D^1 F(x_3)$		$D^3 F(x_4)$		$D^5 F(x_5)$
x_3	$F(x_3)$		$D^2 F(x_4)$		$D^4 F(x_5)$	
				$D^3 F(x_5)$		
x_4	$F(x_4)$		$D^2 F(x_5)$			
		$D^1 F(x_5)$				
x_5	$F(x_5)$					

Za pomocą ilorazów różnicowych można zapisać wielomian interpolacyjny w sposób następujący:

$$W_n(x) = F(x_0) + (x-x_0)\, D^1 F(x_1) + (x-x_0)(x-x1)\, D^2 F(x_2) + \dots + (x-x_0)(x-x_1)\dots$$
$$\dots (x-x_{n-1})\, D^n F(x_n) \qquad (10)$$

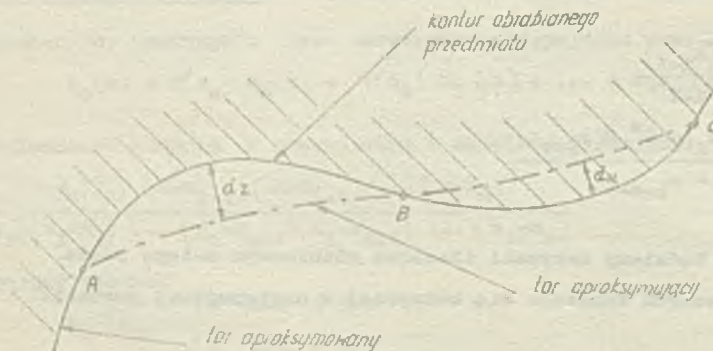
Wielomian (10) jest nazywany wielomianem interpolacyjnym Newtona z ilorazami różnicowymi.

Wprowadzając operatory różnicy zwykłej i różnicy wstecznej można uzyskać różne postacie wielomianu interpolacyjnego Newtona. Wprowadzając dalsze uogólnienia można pokazać, że wielomian interpolacyjny Lagrange'a jest pewną modyfikacją wielomianu interpolacyjnego Newtona.

Stosowanie metod interpolacyjnych i aproksymacyjnych nie zawsze zapewnia optymalny kształt i długość łuku łączącego węzły interpolacji. Ponieważ kształt i długość łuku ma istotne znaczenie dla wyznaczania toru narzędzia, celowe jest a nawet konieczne w niektórych wypadkach stosowanie metod wariacyjnych i innych metod matematycznych.

Interpolacja kołowa i paraboliczna

Ze względu na stosunkową prostotę obliczeniową najczęściej do interpolacji krzywych wyższego rzędu stosuje się interpolację paraboliczną i kołową, tzn. że łuki krzywej zastępujemy odpowiednio dobranymi fragmentami paraboli lub okręgu. Jednakże najczęściej stosuje się jednak interpolację kołową ze względu na charakter obrabianych przedmiotów i wymaganą dokładność. Zasadę działania interpolacji kołowej czy też parabolicznej można schematycznie przedstawić, jak to pokazano na rys.1.



Rys. 1. Ilustracja graficzna interpolacji

Fragment krzywej zawartej między punktami AC zastąpiono w wyniku interpolacji łukami okręgu (paraboli) AB i BC. Długość odcinków d_z i d_w określają maksymalne odchylenia torów, które nazywa się, w zależności od sytuacji, dokładnością, błędem aproksymacji lub też błędem interpolacji. Ogólnie rzecz biorąc, poszczególne fragmenty krzywej mogą być aproksymowane z różną dokładnością. Na ogół wyróżnia się dokładność wewnętrzną (d_w) i zewnętrzną (d_z). Oczywiście dokładność wewnętrzna i zewnętrzna, jest sprawą względną, zależną od naszej umowy. Przyjmiemy następującą konwencję: jeżeli tor aproksymujący leży na zewnątrz konturu przedmiotu obrabianego, to wówczas dokładność nazywamy zewnętrzną i odwrotnie.

Obecnie, w związku ze znacznym rozwojem techniki cyfrowej i układów cyfrowych oraz znacznym ich potanieniem, w systemach sterowania numerycznego interpolatory analogowe zastępuje się interpolatorami cyfrowymi. W związku z tym zasadniczego znaczenia nabiera interpolacja liniowa, gdyż interpolatory cyfrowe działają na zasadzie aproksymacji liniowej. Z powyższych powodów omówimy nieco dokładniej interpolację liniową.

- Interpolacja liniowa

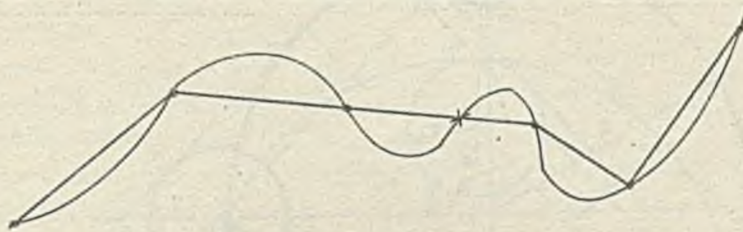
Mówimy, że funkcja F określona w przedziale domkniętym (a, b) jest interpolowana w sposób liniowy przez funkcję f jeżeli dla $0 \leq p \leq 1$ zachodzi następująca relacja:

$$f(a+pb) = F(a) + p\,(F(a+b) - F(a)) \tag{11}$$

Natomiast błąd interpolacji d można oszacować w sposób następujący:

$$d \leq b^2 \left| p/2 \right| \max \left| F''(x) \right|$$

Jeżeli przyjmiemy interpretację geometryczną to w zależności (11) wynika, że łuki zawarte między węzłami interpolacji, odpowiadające funkcji interpolowanej, zastępujemy odcinkami linii prostej. W wyniku interpolacji liniowej krzywa interpolowana zostaje zastąpiona prostą łamaną mającą punkty wspólne co najmniej w węzłach interpolacji z krzywą interpolowaną (rys. 2).



. - węzły interpolacji
x - punkt przecięcia

Rys. 2. Interpolacja liniowa krzywej

Podobnie jak poprzednio można wyróżnić dokładność wewnętrzną i zewnętrzną.

- Aproksymacja liniowa okręgu

  - Uwagi wstępne

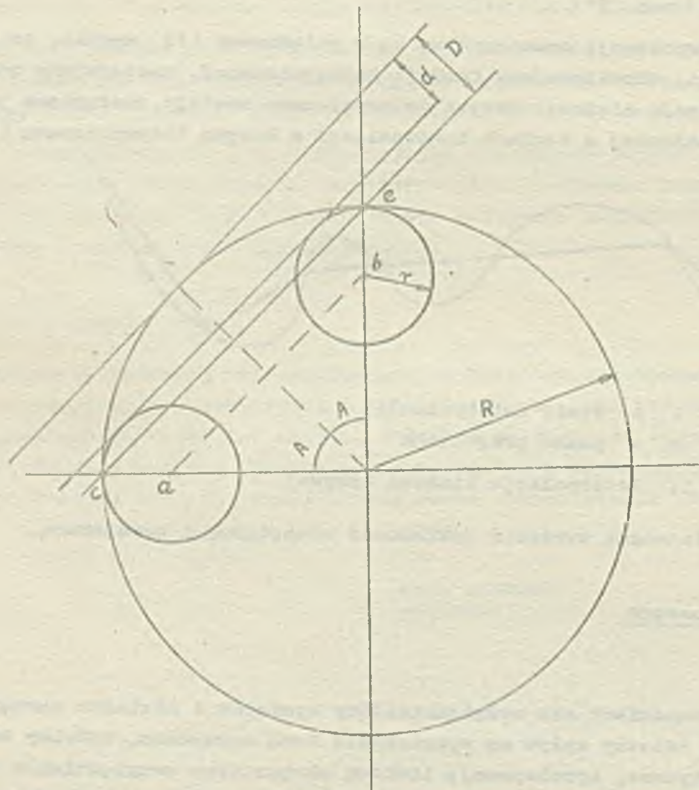
W dotychczasowych rozważaniach nie uwzględnialiśmy wymiarów i kształtu narzędzia. Oczywiście, że te parametry mają istotny wpływ na wyznaczanie toru narzędzia. Omówimy teraz, ze względu na duże znaczenie praktyczne, aproksymację liniową okręgu przy uwzględnieniu wpływu wymiarów narzędzia (średnicy frezu). Jednakże nie będziemy uwzględniali zmiany wymiarów spowodowanych zużyciem narzędzia. System realizujący aproksymację liniową okręgu nazywa się zazwyczaj interpolatorem kołowym. Ze względu na określenie dokładności można wyróżnić trzy podstawowe metody realizacji interpolatora a mianowicie: cięciw, stycznych i siecznych. Metody te schematycznie zilustrowano na rys. 3.

Rys. 3. Ilustracja graficzna metod liniowej aproksymacji okręgu

Przeanalizujemy teraz nieco dokładniej aproksymację liniową okręgu za pomocą cięciw, stycznych i siecznych. Najpierw rozpatrzymy realizację za pomocą cięciw.

- Metoda cięciw dla łuków wklęsłych

Aproksymację liniową okręgu za pomocą cięciw można przedstawić schematycznie(rys. 4).



Rys. 4. Ilustracja graficzna aproksymacji liniowej okręgu metodą cięciw

Jeżeli przesuniemy narzędzie o promieniu r po linii prostej z punktu a do b, jak to przedstawiono na rys. 4, to d określa dokładności aproksymacji. Natomiast droga kątowa dla tego przypadku jest wyznaczona przez kąt 2A. Natomiast D określa dokładność uzyskaną w wyniku przesunięcia narzędzia o średnicy równej zero po linii prostej z punktu c do punktu e. Droga

kątowa dla tego przypadku wynosi 2A.

Łatwo się przekonać, że zachodzą następujące związki:

$$
\begin{aligned}
D &= R - R \cos A \\
D - d &= r - r \cos A
\end{aligned}
\qquad (12)
$$

Rozwiązując powyższe równania uzyskujemy

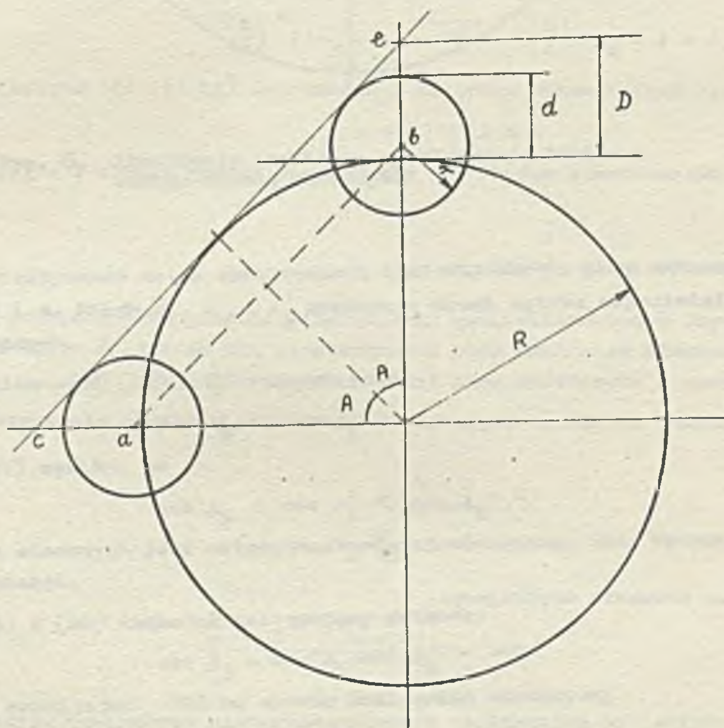
$$
\cos A = \frac{c - d}{c} = 1 - \frac{d}{c} \qquad (13)
$$

gdzie $c = R - r$

Z zależności (13) wynika, że zwiększając średnicę narzędzia uzyskujemy większą dokładność a przy zachowaniu tej samej dokładności i zwiększeniu średnicy narzędzia powiększa się droga kątowa, co jest równoznaczne ze zmniejszeniem liczby węzłów interpolacji. Powyższe wyniki są zgodne z intuicją i wynikają bezpośrednio z zasady aproksymacji okręgu metodą cięciw (rys. 4).

- **Metoda stycznych dla łuków wklęsłych**

Korzystając z rys. 5 przeanalizujemy aproksymację liniową okręgu opartą na metodzie stycznych. Przyjęte oznaczenia mają taką samą interpretację jak na rys. 4.



Rys. 5. Ilustracja graficzna aproksymacji liniowej okręgu metodą stycznych

Łatwo się przekonać, że dla aproksymacji liniowej okręgu metodą stycznych zachodzą następujące związki:

$$
\begin{aligned}
D &= \frac{R}{\cos A} - R \\
D - d &= \frac{r}{\cos A} - r
\end{aligned}
\qquad (14)
$$

Rozwiązując powyższe równania uzyskujemy:

$$\cos A = \frac{c}{c + d} = \left(1 + \frac{d}{c}\right)^{-1} \tag{15}$$

Podobnie jak poprzednio, w zależności (15) wynika, że zwiększając średnicę narzędzia uzyskujemy większą dokładność a przy zachowaniu tej samej dokładności i zwiększeniu średnicy narzędzia powiększa się droga kątowa, co jest równoważne ze zmniejszeniem liczby węzłów interpolacji.

Porównamy teraz efektywność metody cięciw i stycznych. Jako kryterium efektywności przyjmiemy liczbę węzłów interpolacji. Metodę będziemy nazywali bardziej efektywną jeżeli używa mniejszej liczby węzłów interpolacji. Wprowadzenie takiego kryterium wydaje się być dość naturalne.* W dalszych rozważaniach dotyczących porównywania efektywności metod interpolacji, ograniczymy się tylko do kryterium opartego na liczbie węzłów interpolacji.

Niech A_1 i A_2 oznaczają drogi kątowe odpowiednio dla metody cięciw i stycznych. Przy założeniu identycznej wartości parametrów d i c, z zależności (13) i (15) uzyskujemy, że

$$\cos A_1 < \cos A_2 \tag{16}$$

co oznacza, że metoda cięciw jest bardziej efektywna, gdyż zapewnia mniejszą liczbę węzłów interpolacji. Ponadto metoda cięciw zapewnia dokładne odwzorowanie toru w punktach zmiany krzywizny toru.

Jeżeli $\frac{d}{c} < 1$ to rozwijając (15) na szereg Maclaurina uzyskujemy

$$\cos A = 1 - \frac{d}{c} + \left(\frac{d}{c}\right)^2 - \left(\frac{d}{c}\right)^3 \ldots + (-1)^n \left(\frac{d}{c}\right)^n \tag{17}$$

Jeżeli $\frac{d}{c}$ jest dostatecznie małe, to z zależności (17) i (13) uzyskujemy ostatecznie, że

$$\cos A_1 \approx \cos A_2 \tag{18}$$

co oznacza, że dla dostatecznie małych $\frac{d}{c}$ efektywność metody cięciw i stycznych jest porównywalna.

- Metoda siecznych dla łuków wklęsłych

Korzystając z rys. 6 przeanalizujemy aproksymację liniową okręgu opartą na metodzie siecznych. Przyjęte oznaczenia mają taką samą interpretację jak na rys. 4 i 5. Ponadto zakładamy, że dokładności zewnętrzna i wewnętrzna mają takie same wartości (rys. 6).

Łatwo się przekonać, że dla proksymacji liniowej okręgu metodą siecznych zachodzą następujące związki:

$$D = \frac{R - d}{\cos A} - R \tag{19}$$

$$D - d = \frac{r}{\cos A} - r$$

Rozwiązując powyższe równania uzyskujemy:

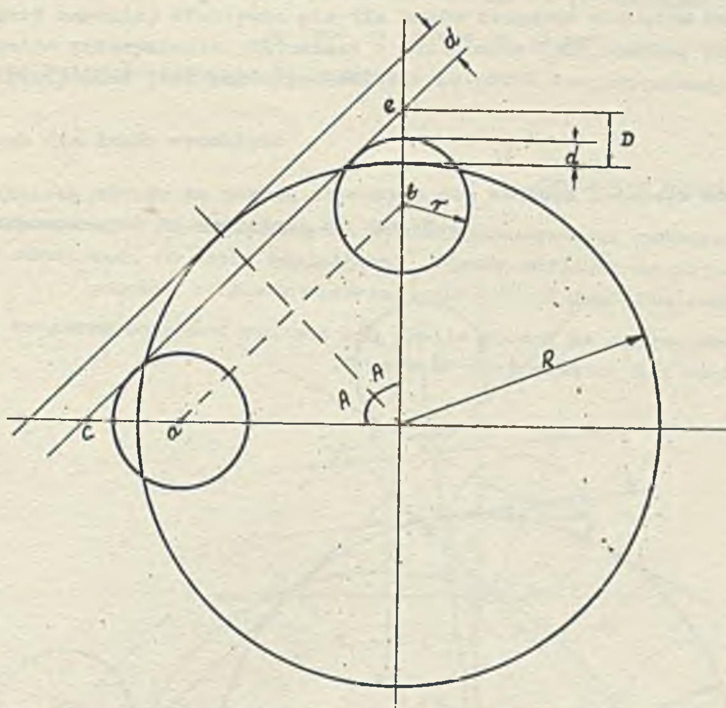
$$\cos A = \frac{c - d}{c + d} = \frac{1 - \frac{d}{c}}{1 + \frac{d}{c}} \tag{20}$$

Z zależności (20) wynika, że zwiększając średnicę narzędzia uzyskujemy większą dokładność a przy zachowaniu tej samej dokładności i zwiększeniu średnicy narzędzia powiększa się droga kątowa, co jest równoważne ze zmniejszeniem liczby węzłów interpolacji. Powyższą analizę przeprowadzono zakładając, że dokładność wewnętrzna i zewnętrzna są jednakowe (przy tych założeniach została wyprowadzona zależność (20)). Jeżeli założymy, że dokładność wewnętrzna jest stała, to korzystając z interpretacji geometrycznej metody siecznych przedstawionej na rys. 6, łatwo zauważyć, że zwiększenie średnicy narzędzia, przy zachowaniu tej samej drogi kątowej pociąga za sobą zwiększenie dokładności wewnętrznej. Natomiast uzyskanie jednakowych dokładności pociąga

---

* Przy założeniu jednakowego nakładu pracy (co prawie zawsze można przyjąć) wymaganego do wyznaczenia jednego węzła interpolacji, powyższe kryterium pozwala w łatwy sposób ocenić różne metody.

ze sobą powiększenie drogi kątowej.



Rys. 6. Ilustracja graficzna aproksymacji liniowej okręgu metodą siecznych

Porównamy teraz efektywność metod aproksymacji liniowej okręgu przy założeniu identycznej wartości parametrów d i c. Niech A_1, A_2 i A_3 oznaczają drogi kątowe odpowiednio dla metody cięciw, stycznych i siecznych.

Bezpośrednio z zależności (13) i (20) uzyskujemy

$$\cos A_3 < \cos A_1 \tag{21}$$

Natomiast z (16) i (21) wynika, że

$$\cos A_3 < \cos A_1 < \cos A_2 \tag{22}$$

co oznacza, że metoda siecznych jest metodą najbardziej efektywną, tzn. wymaga najmniejszej liczby węzłów interpolacji.

Na mocy (13), (15) i (20) zachodzi następujący związek:

$$\cos A_3 = \cos A_1 \cos A_2 \tag{23}$$

Jeżeli $\left|\frac{d}{c}\right| < 1$ to rozwijając (20) na szereg Maclaurina uzyskujemy

$$\cos A = 1 - 2\frac{d}{c} + 2\left(\frac{d}{c}\right)^2 - 2\left(\frac{d}{c}\right)^3 + \ldots 2(-1)^n \left(\frac{d}{c}\right)^n \tag{24}$$

Ponieważ

$$\left(1 - \frac{d}{c}\right)^2 = 1 - 2\frac{d}{c} + \left(\frac{d}{c}\right)^2 \tag{25}$$

to dla dostatecznie małych $\frac{d}{c}$ z (25), (24) i (13) uzyskujemy ostatecznie

$$\cos A_3 \approx \cos^2 A_1 \tag{16}$$

Powyższy wynik można również uzyskać na podstawie zależności (18) i (23).

Na mocy (26) i (18) uzyskujemy, że dla dostatecznie małych $\frac{d}{c}$ zachodzi relacja

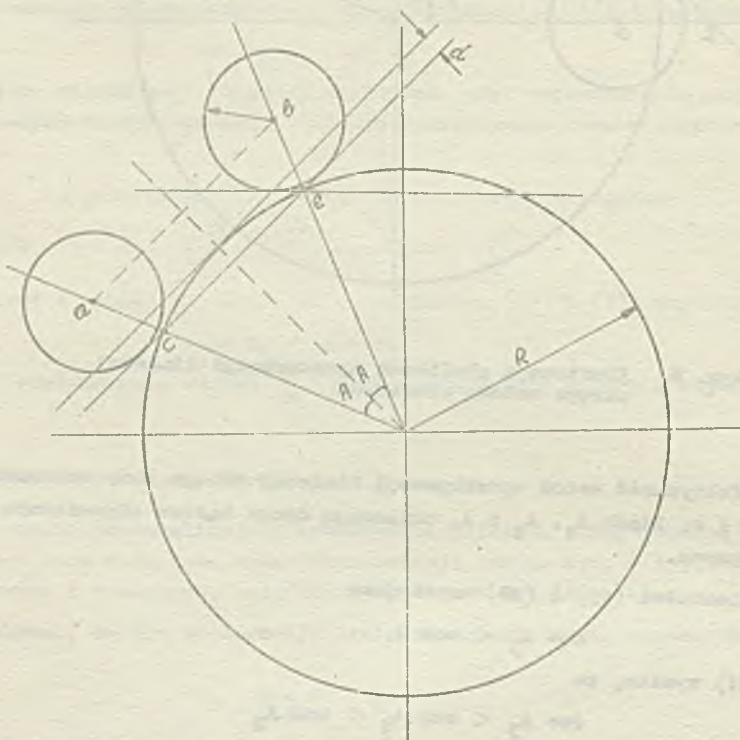
$$\cos A_2 \approx \cos^2 A_1 \approx \cos^2 A_2 \tag{27}$$

co oznacza, że metoda siecznych jest bardziej efektywna od pozostałych metod aproksymacji liniowej okręgu.

• Metoda cięciw dla łuków wypukłych

Dotychczas analizowaliśmy aproksymację liniową okręgu za pomocą cięciw, stycznych i siecznych dla konturów leżących na zewnątrz okręgu. Przejdziemy teraz do rozważania aproksymacji liniowej okręgu tymi samymi metodami dla kontutu leżącego wewnątrz okręgu.

Aproksymację liniową okręgu za pomocą cięciw dla konturu leżącego wewnątrz okręgu można przedstawić schematycznie jak to uczyniono na rys. 7.



Rys. 7. Ilustracja graficzna aproksymacji liniowej okręgu metodą cięciw

Bezpośrednio z rys. 7 wynika następująca zależność

$$\cos A = \frac{R - d}{R} = 1 - \frac{d}{R} \tag{28}$$

Z zależności (28) wynika, że droga kątowa zależy tylko od dokładności, a nie zależy od średnicy narzędzia, jak to miało miejsce dla aproksymacji liniowej okręgu cięciwami dla konturów leżących na zewnątrz okręgu. Niech A^w i A^z oznaczają odpowiednio drogi kątowe dla interpolacji liniowej okręgu leżącego wewnątrz i na zewnątrz konturu.

Na mocy (13) i (28) przy założeniu jednakowej dokładności i jednakowej średnicy okręgu uzyskujemy

$$\cos A^z < \cos A^w \tag{29}$$

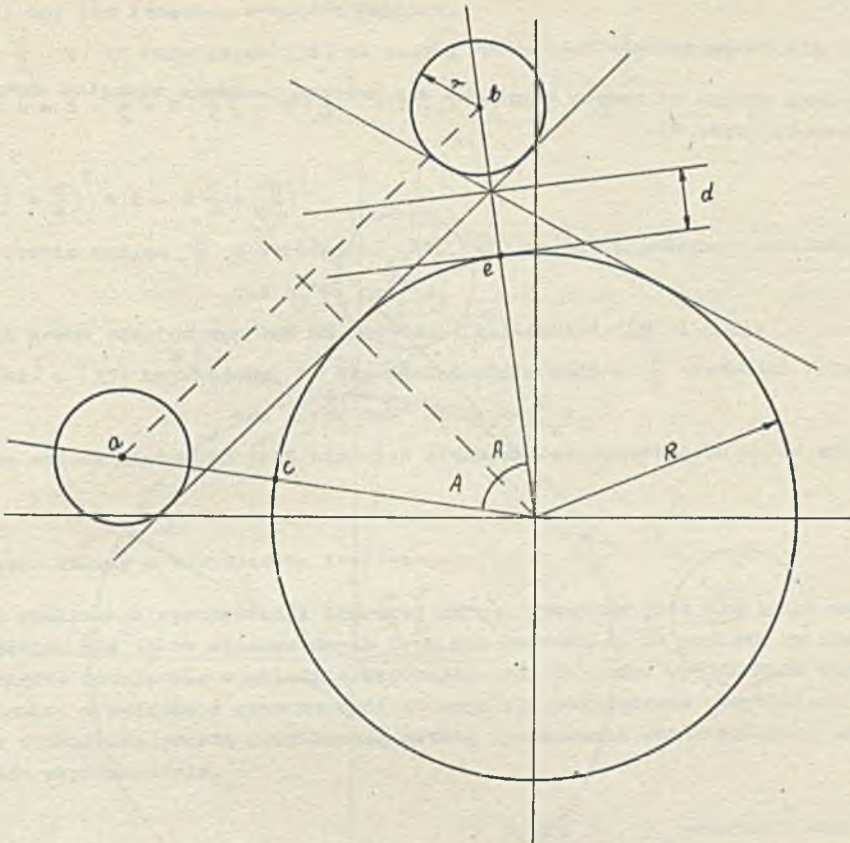
a przy dostatecznie małej wartości $\frac{r}{R}$, że

$$\cos A^z \approx \cos A^w \tag{30}$$

Z zależności (29) wynika, że aproksymacja liniowa okręgu cięciwami dla łuków leżących na zewnątrz konturu jest bardziej efektywna niż dla łuków leżących wewnątrz konturu gdyż zapewnia mniejszą liczbę węzłów interpolacji. Natomiast z zależności (30) wynika, że dla dostatecznie małej wartości $\frac{r}{R}$ efektywność jest porównywalna.

- Metoda stycznych dla łuków wypukłych

Aproksymację liniową okręgu za pomocą stycznych dla konturu leżącego wewnątrz okręgu można przedstawić schematycznie jak to uczyniono na rys. 8.



Rys. 8. Ilustracja graficzna aproksymacji liniowej okręgu metodą stycznych

Bezpośrednio z rys. 8 wynika następująca zależność:

$$\cos A = \frac{R}{R + d} = \left(1 + \frac{d}{R}\right)^{-1} \tag{31}$$

Podobnie jak poprzednio, z zależności (31) wynika, że droga kątowa zależy tylko od dokładności, a nie zależy od średnicy narzędzia, jak to miało miejsce dla aproksymacji liniowej okręgu stycznymi dla konturów leżących na zewnątrz okręgu. Na mocy zależności (15) i (31) przy założeniu jednakowej dokładności i jednakowej średnicy okręgu uzyskujemy:

$$\cos A^{z} < \cos A^{w} \tag{32}$$

a przy dokładności małej wartości $\frac{d}{R}$, że

$$\cos A^{z} \approx \cos A^{w} \tag{33}$$

co oznacza, że aproksymacja liniowa okręgu stycznymi dla łuków leżących na zewnątrz konturu jest bardziej efektywna niż dla łuków leżących wewnątrz konturu, gdyż zapewnia mniejszą liczbę węzłów interpolacji. Natomiast z zależności (33) wynika, że dla dostatecznie małej wartości $\frac{r}{R}$ efektywność jest porównywalna.

Ponieważ $\frac{d}{R} < 1$ to rozwijając zależność(31) na szereg Maclaurina uzyskujemy

$$\cos A = 1 - \frac{d}{R} + \left(\frac{d}{R}\right)^2 - \left(\frac{d}{R}\right)^3 \dots + \left(-1\right)^n \left(\frac{d}{R}\right)^n \tag{34}$$

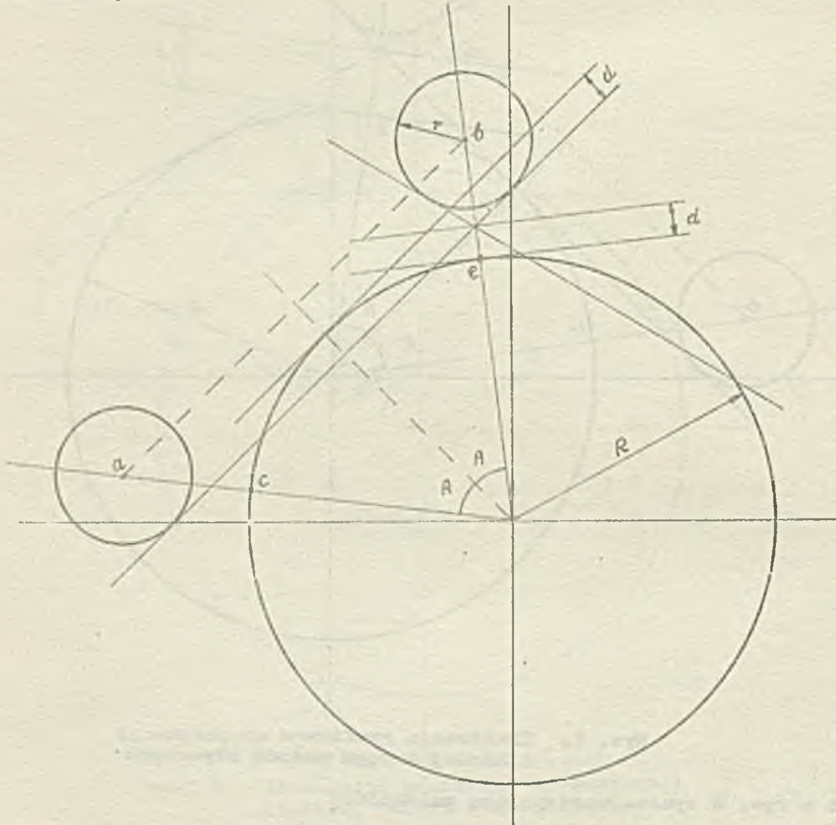
Jeżeli $\frac{d}{R}$ jest dostatecznie małe, to z zależności (28) i (34) uzyskujemy ostatecznie, że

$$\cos A_1 \approx \cos A_2 \tag{35}$$

co oznacza, że dla dostatecznie małych $\frac{d}{R}$ efektywność metody cięciw i stycznych jest porównywalna.

- Metoda siecznych dla łuków wypukłych

Aproksymację liniową okręgu za pomocą siecznych dla konturu leżącego wewnątrz okręgu można przedstawić schematycznie (rys. 9).



Rys. 9. Ilustracja graficzna aproksymacji liniowej okręgu metodą cięciw

Bezpośrednio z rys. 9 wynika następująca zależność

$$\cos A = \frac{R - d}{R + d} = \frac{1 - \frac{d}{R}}{1 + \frac{d}{R}} \tag{36}$$

co oznacza, że droga kątowa zależy tylko od dokładności, a nie zależy od średnicy narzędzia, jak to miało miejsce dla aproksymacji liniowej okręgu siecznymi dla konturów leżących na zewnątrz okręgu.

Na mocy zależności (20) i (36) przy założeniu jednakowej dokładności i średnicy okręgu uzyskujemy, że

$$\cos A^z < \cos A^w \tag{37}$$

a przy dostatecznie małej wartości $\frac{r}{R}$, że

$$\cos A^z \approx \cos A^w \tag{38}$$

co oznacza, że aproksymacja liniowa okręgu siecznymi dla łuków leżących na zewnątrz okręgu jest bardziej efektywna niż dla łuków leżących wewnątrz konturu, gdyż zapewnia mniejszą liczbę węzłów interpolacji, a dla dostatecznie małej wartości $\frac{r}{R}$ efektywność jest porównywalna.

Z zależności (28), (31) i (36) uzyskujemy następujący związek:

$$\cos A_3 = \cos A_1 \cos A_2 \tag{39}$$

co oznacza, że metoda siecznych jest metodą najbardziej efektywną.

Natomiast z zależności (23) i (39) wynika, że między metodą siecznych, stycznych i cięciw zachodzi ta sama relacja, niezależnie od tego, czy dokonujemy aproksymacji łuku leżącego na zewnątrz konturu, czy też leżącego wewnątrz konturu.

Ponieważ $\frac{d}{R} < 1$ to rozwijając (36) na szereg Maclaurina uzyskujemy

$$\cos A = 1 - \frac{d}{R} + 2\left(\frac{d}{R}\right)^2 - 2\left(\frac{d}{R}\right)^2 + \ldots\ldots 2(-1)^n \left(\frac{d}{R}\right)^n \tag{40}$$

Ponieważ

$$\left(1 - \frac{d}{R}\right)^2 = 1 - 2\frac{d}{R} + \left(\frac{d}{R}\right)^2 \tag{41}$$

to dla dostatecznie małych $\frac{d}{R}$ z zależności (41), (40) i (28) uzyskujemy ostatecznie

$$\cos A_3 \approx \cos^2 A_1 \tag{42}$$

Powyższy wynik można również uzyskać na podstawie zależności (39) i (35).

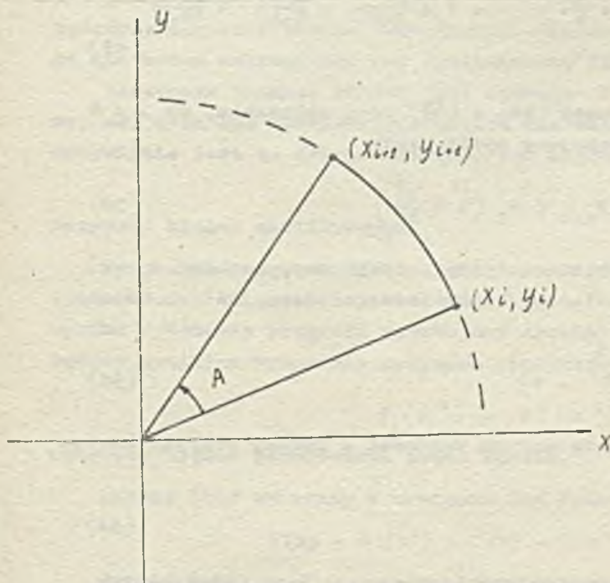
Na mocy (42) i (35) uzyskujemy, że dla dostatecznie małych $\frac{d}{R}$ zachodzi relacja

$$\cos A_3 \approx \cos^2 A_1 \approx \cos^2 A_2 \tag{43}$$

co oznacza, że metoda siecznych jest bardziej efektywna od pozostałych metod aproksymacji liniowej okręgu.

- Interpolator kołowy – przybliżona realizacja

Do pełnej realizacji aproksymacji liniowej okręgu wymagane jest nie tylko określenie drogi kątowej narzędzia, ale także współrzędnych toru osi narzędzia. Ze względu na realizację techniczną współrzędne podaje się w układzie kartezjańskim. Dokładne wyznaczenie współrzędnych jest dość czasochłonne, w związku z czym na ogół stosuje się rozwiązania przybliżone.
Omówimy teraz stosunkowo prostą przybliżoną metodę wyznaczania współrzędnych, opartą na transformacji układu współrzędnych.



Rys. 10. Interpolacja kołowa

Niech x_i, y_i oznaczają współrzędne punktu leżącego na okręgu, wyznaczonego w i-tym kroku działania interpolatora.
Niech x_{i+1}, y_{i+1} oznaczają odpowiednio współrzędne punktu leżącego na okręgu wyznaczonego w i+1-szym kroku działania interpolatora (rys.10).
Jeżeli jest znany kąt obrotu A, to dokonując transformacji układu współrzędnych opartej na obrocie układu współrzędnych o kąt A uzyskujemy następujące związki:

$$\begin{aligned} x_{i+1} &= x_i \cos A - y_i \sin A \\ y_{i+1} &= x_i \sin A + y_i \cos A \end{aligned} \tag{44}$$

Stosując rozwinięcie na szereg Maclaurina uzyskujemy

$$\begin{aligned} \sin A &= A - \frac{A^3}{3!} + \frac{A^5}{5!} - \frac{A^7}{7!} + \ldots \\ \cos A &= a - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!} \end{aligned} \tag{45}$$

Dla dostatecznie małych kątów A możemy przyjąć następujące przybliżenia:

$$\sin A = A$$
$$\cos A = 1 - \frac{A^2}{2!} \qquad (46)$$

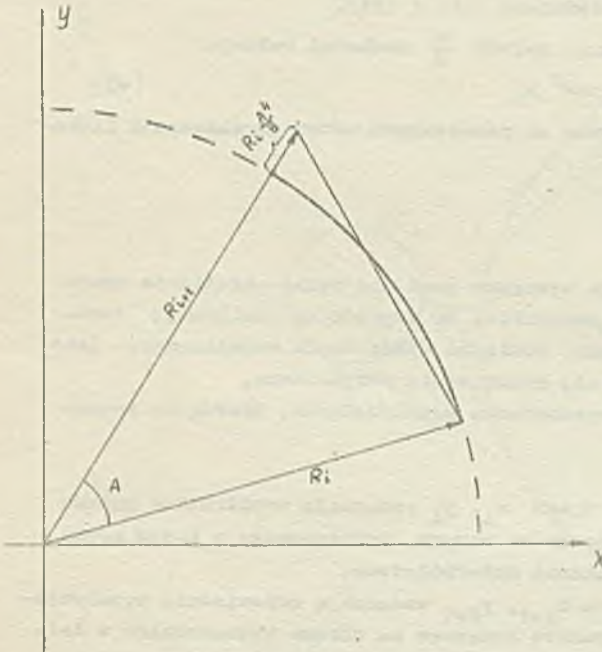
Przyjmując powyższe przybliżenia, popełniamy błąd mniejszy od $\frac{A^3}{3!}$ przy wyznaczaniu sin A, a przy wyznaczaniu cos A błąd mniejszy od $\frac{A^4}{4!}$.

Podstawiając (46) do (44) uzyskujemy następujące równania:

$$x_{i+1} - x_i = -x_i \frac{A^2}{2} - y_i A$$
$$y_{i+1} - y_i = x_i A - y_i \frac{A^2}{2} \qquad (47)$$

Z równań tych stosunkowo prosto można wyznaczyć współrzędne punktu x_{i+1}, y_{i+1}.

Zbadamy, jaki powstaje błąd wyznaczania promienia na podstawie założeń (47) tzn. błąd, który jest spowodowany przyjęciem przybliżeń określonych zależnością (46). Interpretacja geometryczna tego błędu została podana na rys. 11.



Rys. 11. Błąd związany z promieniem

Podnosząc równania (47) do kwadratu i rozwiązując je uzyskujemy ostatecznie, że

$$R_{i+1} = R_i \left(1 + \frac{A^4}{4}\right)^{\frac{1}{2}} \qquad (48)$$

gdzie

$$R_i = \left(x_i^2 + y_i^2\right)^{\frac{1}{2}} \quad \text{oraz}$$

$$R_{i+1} = \left(x_{i+1}^2 + y_{i+1}^2\right)^{\frac{1}{2}}$$

Jeżeli $\frac{A^4}{4} \leqslant 1$ to rozwiązując $\left(1 + \frac{A^4}{4}\right)^{\frac{1}{2}}$ na szereg Maclaurina uzyskujemy

$$\left(1 + \frac{A^4}{4}\right)^{\frac{1}{2}} = 1 + \frac{A^4}{2 \cdot 4} - \frac{A^8}{8 \cdot 4} + \frac{A^{12}}{16 \cdot 4} - \dots \qquad (49)$$

Na mocy (48) i (49) przy założeniu, że $A \neq 0$ uzyskujemy ostatecznie

$$R_{i+1} < R_i \left(1 + \frac{A^4}{8}\right) \qquad (50)$$

W zależności (50) uzyskujemy, że błąd $d_r(i)$ spowodowany przyrostem promienia w wyniku wykonania i-tego kroku interpolatora jest określony w sposób następujący:

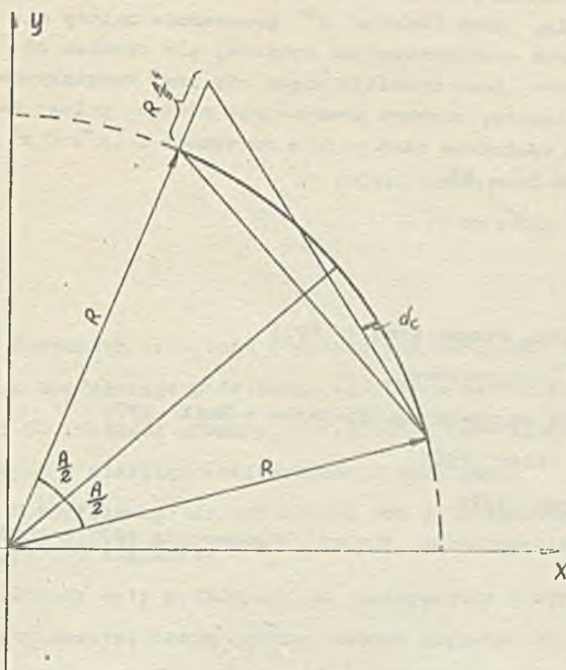
$$iR \frac{A^4}{8} < d_r(i) < R \left(\left(1 + \frac{A^4}{8}\right)^i - 1\right) \qquad (51)$$

Ponieważ $\left(1 + \frac{A^4}{8}\right)^i > 1 + i \frac{A^4}{8}$ to dla dostatecznie małych wartości A możemy przyjąć następujące przybliżenie

$$d_r(i) \approx i R \frac{A^4}{8} \qquad (52)$$

Błąd d_0 związany z poruszaniem się po cięciwie zamiast po łuku okręgu jest największy w pierwszym kroku algorytmu. Na podstawie rys. 12 uzyskujemy następujący związek:

$$\cos \frac{A}{2} < \frac{R - d_0}{R} \qquad (53)$$



Rys. 12. Błąd związany z cięciwą

Z (53) na mocy (46) dla dostatecznie małych wartości A uzyskujemy ostatecznie

$$d_o < R\frac{A^2}{8} \tag{54}$$

Na mocy (54) i (52) uzyskujemy ostatecznie, że:

$$d_r(i) \approx i\, A^2 d_o \tag{55}$$

co oznacza, że dla dostatecznie małych wartości A błąd d_o jest błędem dominującym.

- <u>Analiza błędów</u>

W dotychczasowych rozważaniach omówiliśmy aspekty matematyczne wyznaczania toru narzędzia obrabiarek sterowanych numerycznie. Nie omawialiśmy zagadnień natury technicznej oraz dokładnej realizacji interpolatora. Z zagadnień natury bardziej ogólnej zostało jeszcze do rozważenia zagadnienie błędów związanych z wyznaczaniem toru narzędzia.

Niech F będzie dowolną ustaloną funkcją. Rozważmy wyznaczanie wartości funkcji F dla argumentu x. Niech x^* oznacza wartość argumentu daną nam do dyspozycji (daną wyjściową) odbiegającą od rzeczywistej wartości x. Odchylenie wartości argumentu x^* od rzeczywistej wartości argumentu x, może być spowodowane ograniczoną długością reprezentacji danych, przyjętym systemem reprezentacji liczb w maszynie lub innymi przyczynami wpływającymi na dokładność danych wejściowych.

Różnicę wartości

$$F(x) - F(x^*)$$

będziemy nazywali błędem wewnętrznym. Dla ustalonej reprezentacji argumentów, błędu wewnętrznego nie można uniknąć ani też wyeliminować żadnymi metodami analitycznymi.

Następnym źródłem błędów jest funkcja f aproksymująca funkcję F. Funkcję f tak konstruujemy, aby była ona dostatecznie prosta dla celów obliczeniowych oraz łatwa do zaprogramowania. Oczywiście jest to źródłem dodatkowych błędów. Błąd ten określony w sposób następujący:

$$F(x^*) - f(x^*)$$

nazywamy błędem analitycznym.

Wykonanie programu realizującego funkcję f, jest dodatkowym źródłem błędów powstałych z ograniczonej długości reprezentacji liczb, obcięcia wyników pośrednich oraz zaokrągleń itp. W wyniku wykonania programu uzyskujemy aproksymację (f^*) funkcji aproksymującej funkcję F. Błąd będący wynikiem wykonania programu realizującego funkcję f

$$f(x^*) - f^*(x^*)$$

nazywamy błędem generowanym przez system.

Łączny błąd związany z wyznaczaniem funkcji F można określić w sposób następujący:

$$F(x) - f^*(x^*) = F(x) - F(x^*) + F(x^*) - f(x^*) + f(x^*) - f^*(x^*)$$

Jeżeli funkcja F jest różniczkowalna to błąd wewnętrzny można wyznaczyć z następującej zależności:

$$F(x) - F(x^*) = \partial/\partial x\,(F(x))\;(x - x^*)$$

Natomiast wyznaczanie błędu analitycznego jest typowym problemem występującym w teorii aproksymacji i istnieje wiele metod jego wyznaczania i szacowania.

Na pierwszy rzut oka mogłoby się wydawać, że wyznaczenie błędu generowanego można wyznaczać tymi samymi metodami jak błąd analityczny. Sytuacja jednak w tym wypadku nie jest taka prosta, bowiem są znaczne trudności z wyznaczeniem postaci analitycznej funkcji f^*. Sprawa wyznaczania postaci analitycznej funkcji f^* komplikuje się, gdyż funkcja f^* przeważnie zależy nie tylko od przyjętego systemu reprezentacji danych oraz charakterystyki systemu, ale również od kolejności wykonywania operacji i wielkości argumentu. Duże nadzieje wiąże się przy rozwiązywaniu tego problemu z tzw. analizą wsteczną, której istota, niezbyt precyzyjnie mówiąc, polega na wyznaczaniu takiej wartości argumentu x^* aby była spełniona następująca zależność $f(x^*) = f(x^*)$ oraz na uzyskaniu oszacowań na bezwzględną wartości $\bar{x}^* - \bar{x}^*$.

Literatura

[1] Simon W.: The numerical control of machine tools. Edward Arnold 1973

[2] Olesten N.O.: Numerical control. John Wiley - Interscience

[3] Stanton R.G.: Numerical methods for science and engineering. Prentice - Hall 1971

[4] Rall L.B.: Error in digital computation. John Wiley 1965

[5] Rice J.R.: Mathematical software. Academic Press 1971

[6] Bergren C.: A simple algorithm for circular interpolation. Control Engineering 1971 vol 18, nr 9 s. 57-39.

Sprawozdania z konferencji

23 Polska Konferencja Mechaniki Ciała Stałego PAN
3-11 września 1981 r.,Mrągowo

Staraniem Instytutu Podstawowych Problemów Techniki odbyła się w dniach 3-11.IX.1981 r. doroczna konferencja poświęcona problemom mechaniki ciał stałych. W ciągu ostatnich dwóch dni, z okazji 70 rocznicy urodzin prof.Witolda Nowackiego, odbyło się osobne sympozjum na temat pól sprzężonych w nieklasycznej mechanice kontinuum.

W konferencji wzięło udział stu kilkudziesięciu uczestników z kraju (wśród nich niżej podpisany) i z zagranicy.

Obrady były podzielone na następujące grupy tematyczne: konstrukcje, ciała niesprężyste,elementy maszyn, sesja ogólna, metody komputerowe, ośrodki ze strukturą komórkową, metody eksperymentalne, metody matematyczne, ośrodki ciągłe. Wymienione grupy tematów były omawiane kolejno,co pozwalało na wysłuchanie wszystkich wybranych referatów.

Szczególnie zajmująca z punktu widzenia Pracowni Oprogramowania Zastosowań IMM była sesja dotycząca metod komputerowych. Omawiano na niej przede wszystkim zastosowania metody elementów skończonych, ale także metody różnic skończonych do rozwiązywania nieliniowych geometrycznie i materiałowo konstrukcji. Rozważane były również zastosowania mes w mechanice pęknięć i interesujące obliczenia ilustrujące kierunek i szybkość propagacji szczelin w płytach. Wiele uwagi poświęcono analizie sprężysto-plastycznej belek i ram. Do uzyskania przedstawionych wyników numerycznych na ogół używano programów specjalistycznych, w niektórych tylko wypadkach wykorzystując duże uniwersalne systemy obliczeniowe mes.

W czasie sesji ogólnej prof.H.Frąckiewicz z IPPTPAN przedstawił stan badań w problemie węzłowym N°05-12 "Wytrzymałość i optymalizacja konstrukcji inżynierskich" wymieniając tematy szczegółowe, zespoły, ich kierowników oraz sposób i stan ich realizacji.

Sam sposób organizacji 23 konferencji był typowy, w dobrym tego słowa znaczeniu. Większość 20-minutowych referatów wygłoszono po angielsku, ilustrując je przeźroczami. Prezentację każdej grupy kolejnych trzech prac zamykała dyskusja prowadzona przez przewodniczącego sesji. Obrady odbywały się w hotelu Mrongovia w Mrągowie. Materiały z konferencji znajdują się w Bibliotece Instytutu Maszyn Matematycznych.

dr Adam LUTOBORSKI

Szkoła Metodologii Projektowania
16-19 września 1981 r., Karpacz-Bierutowice

W dniach 16-19 września 1981 r. w Karpaczu-Bierutowicach odbyła się Szkoła Metodologii Konstruowania Maszyn, Metody Rozwiązywania Problemów Projektowo-Konstrukcyjnych. Szkoła ta zorganizowana została po raz pierwszy. Organizatorem był Instytut Konstrukcji i Eksploatacji Maszyn Politechniki Wrocławskiej, a Kierownikiem Szkoły - doc.dr hab. inż. Ryszard Rohatyński.

Uczestnikami Szkoły byli nauczyciele akademiccy, pracownicy naukowi ze środowisk zajmujących się zagadnieniami metodologii projektowania oraz projektanci z biur projektowych i konstruktorzy z zakładów przemysłowych. Łącznie w Szkole wzięło udział ok.40 osób; zaznaczyła się liczbowa przewaga środowiska wrocławskiego, co należy uznać za naturalne ze względu na to, że ośrodek ten organizował tę Szkołę. W pracach Szkoły wziął też udział niżej podpisany, jako reprezentant zainteresowanego problemami metodologii projektowania Zespołu Pracowni BP-1 Instytutu Maszyn Matematycznych.

Program Szkoły obejmował cztery cykle wykładów, które prowadzili czterej wykładowcy. Doc. dr inż. Wojciech Tarnowski z Instytutu Automatyki Politechniki Śląskiej w Gliwicach miał do dyspozycji 6 godzin zajęć. Przeprowadził w tym czasie interesujący wykład na temat struktury procesu projektowania omawiając zagadnienia:

- makrostruktura (struktura pionowa) procesu projektowania
- mikrostruktura (struktura pozioma) procesu projektowania
- dekompozycja w projektowaniu
- optymalizacja w procesie projektowania
- konstrukcja globalnego kryterium oceny rozwiązań projektowych w warunkach istnienia wielu kryteriów cząstkowych.

Prezentowany przez wykładowcę materiał wynikał z jego własnych prac. Wykład był prowadzony bardzo żywo i barwnie, spotkał się więc z bardzo dobrym przyjęciem słuchaczy i wywołał wiele ciekawych głosów w dyskusji. Moim zadniem do najciekawszych fragmentów wykładu należy zaliczyć - niezauważone chyba przez innych słuchaczy - oryginalne sformułowanie zadania optymalizacji w projektowaniu. Autor w jednym zadaniu optymalizacyjnym połączył zagadnienie optymalizacji właściwości konstrukcyjnych projektowanego wyrobu oraz wynikających z jego produkcji efektów ekonomicznych, które osiąga producent. Natomiast innym ciekawym i żywo dyskutowanym problemem był sposób konstrukcji tzw. globalnego kryterium oceny rozwiązań projektowych w warunkach kompromisu. Sposób konstrukcji owej funkcji globalnej oparł autor na pojęciu użyteczności sformułowanym w latach pięćdziesiątych przez Von Neumana i Morgensterna.

Drugi cykl wykładów prowadził doc.dr inż. Bogusław Machowski z Instytutu Podstaw Budowy Maszyn Akademii Górniczo-Hutniczej z Krakowa, który przez kolejne 6 godzin zajmował się metodami poszukiwania różnych koncepcji rozwiązań zadań projektowych. Między innymi przedstawił metodę LEMACH-3 oraz dość szczegółowo omówił metodę morfologiczną generacji rozwiązań projektowych. Metodę tę zilustrował ciekawym praktycznym przykładem zaczerpniętym z dziedziny projektowania konstrukcji mechanicznych.

W cyklu trzecim dr inż. Rudolf Kołodziej z Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej (zastępujący nieobecnego prof. A.Sielickiego) omawiał głównie zagadnienia selekcji rozwiązań projektowych. Wykładowca założył, że słuchacze z przedstawianą problematyką zetknęli się po raz pierwszy, materiał przedstawił więc w sposób ogólnikowy i szkolny, poświęcając wiele uwagi wyjaśnieniu problemów podstawowych. W konsekwencji zabrakło czasu na dokładniejsze omówienie poszczególnych zasygnalizowanych zagadnień.

W cyklu czwartym (5-godzinnym) dr Andrzej Strzelecki - psycholog z Zakładu Prakseologii i Naukoznawstwa Instytutu Filozofii i Socjologii PAN w Warszawie przedstawił psychologiczne mechanizmy wpływające na wyniki twórczej pracy projektantów. W wykładzie pokazał jak można badać cechy osobowości człowieka i jaki wpływ mają poszczególne cechy na wyniki jego pracy twórczej. Zaprezentował słuchaczom wyniki prac własnych związanych z badaniem studentów Wydziału Architektury Politechniki Warszawskiej. Celem tych badań było ustalenie jakie cechy osobowości studentów mają wpływ na wyniki w nauce, czy są one stałe w czasie studiów czy też są zależne od czasu studiowania oraz czy egzaminy wstępne badają te cechy, które powinny badać. Jako ciekawostkę można podać, że pod wpływem tych prac zmieniono zakres egzaminu z rysunku, ponieważ dotychczasowy nie badał tych cech kandydatów, które miały wpływ na wyniki w nauce. Na zakończenie odbyła się ożywiona dyskusja wykazująca, że wykład był bardzo potrzebny i ciekawie przeprowadzony.

Odbyła się także dyskusja na temat założeń programowych następnych szkół metodologii konstruowania maszyn. Do głównych postulatów zgłoszonych przez uczestników należy zaliczyć:

- włączenie do tematyki szkoły zagadnień komputeryzącji procesu projektowania
- prezentowanie oryginalnych prac własnych uczestników i wykładowców z zakresu praktycznego projektowania
- kontynuowanie cyklu wykładów z zakresu roli psychologii czy socjologii w procesie projektowania
- wydawanie materiałów szkoleniowych uczestnikom.

Brak materiałów stanowił poważny mankament Szkoły. Jedynie doc. W.Tarnowski do jednego z fragmentów swego wykładu rozdał uczestnikom materiały, stanowiące rozszerzenie omawianego problemu. Podanie literatury przez wykładowców nie może zastąpić materiałów będących niezbędną pomocą dla słuchaczy.

Na zakończenie należy dodać, że organizatorzy urządzili skromną wystawę literatury wiążącej się z tematyką szkoły. Należy to kontynuować w następnych latach.

Szkołę zakończono miłym spotkaniem towarzyskim, co pozwoliło uczestnikom rozstać się z życzeniami spotkania na następnej konferencji za rok lub dwa lata.

dr inż. Ryszard PAWLIK

"Zastosowanie komputerów w przemyśle"
Szczecin, 17-18 września 1981 r.

W dniach 17 i 18 września 1981 r. w Szczecinie odbyła się druga konferencja na temat zastosowania komputerów w przemyśle. Obrady odbywały się w pięciu sekcjach:

1 - Zastosowanie komputerów w układach czasu rzeczwistego
2 - Układy mikroprocesorowe i wyspecjalizowane
3 - Zastosowanie komputerów w projektowaniu
4 - Zastosowanie komputerów w sterowaniu przedsiębiorstwem
5 - Modele, symulacja komputerów

Wszystkie referaty były ograniczone w czasie do 10 minut. Po każdych trzech referatach odbywała się dyskusja.

Zdecydowana większość referatów dotyczyła konkretnych zastosowań w przemyśle, które są bądź wdrożone bądź w trakcie przygotowań do wdrożenia. Oczywiście podział na sekcje jest w dużej mierze czysto formalny, np. sekcja 1 i 2 dotyczą zastosowań w układach czasu rzeczywistego z tym,że w sekcji 1 omawiane były układy wykorzystujące minikomputery, natomiast w części 2 minikomputery i mikroprocesory wyspecjalizowane. Oczywiście ścisłe podzielenie tematu konferencji na podtematy jest niemożliwe.

Należy podkreślić starania organizatorów (zresztą uwieńczone sukcesem) aby referaty zbliżone tematycznie nie odbywały się w tym samym czasie, co umożliwiło wysłuchanie i wzięcie udziału w większości dyskusji interesujących uczestnika konferencji.

Spośród wygłoszonych referatów na szczególną uwagę zasługuje moim zdaniem "Komputerowy system przygotowania produkcji tkanin żakardowych - MERA-ŻAKARD", wygłoszony przez W.Tarnowskiego (sekcja 4). Jest to system działający w przemyśle.

Natomiast spośród referatów sekcji 3 uczestników konferencji zainteresował referat "Narzędzia programistyczne dla komputerowo wspomaganego programowania" wygłoszony przez dr inż. Stanisławę Bonkowicz-Sittauer z Instytutu Maszyn Matematycznych.

Konferencję zakończyła dyskusja plenarna, na której podano wnioski dotyczące dalszego rozwoju zastosowań komputerów w przemyśle. Najczęściej powtarzany był wniosek o ujednolicenie urządzeń cyfrowych (wprowadzenie modularności). Ogólny ton dyskusji na temat dalszego rozwoju komputeryzacji w Polsce był jednak pesymistyczny.

Duże niezadowolenie budziły wśród uczestników: bardzo słaba baza materialna oraz brak informacji.Również niewielkie zainteresowanie zakładów przemysłowych urządzeniami komputerowymi skłania do niezbyt optymistycznych wniosków.

Ustalono, że trzecia konferencja na temat zastosowania komputerów w przemyśle odbędzie się za dwa lata.

mgr Marek POKULNIEWICZ

Automatyzacja prac w projektowaniu. Komputerowe wspomaganie projektowania

Sprawozdanie z konferencji w Poznaniu w dn.28-29 października 1981 r.

W dniach 28-29 października 1981 r. odbyła się w Poznaniu IV Krajowa Konferencja Naukowo-Techniczna "Automatyzacja prac w projektowaniu" na temat komputerowego wspomagania projektowania (KWP). Konferencja ta została zorganizowana przez Stowarzyszenie Inżynierów i Techników Mechaników Polskich oddział w Poznaniu, Polskie Towarzystwo Cybernetyczne oddział w Poznaniu oraz Akademię Ekonomiczną w Poznaniu. Z punktu widzenia poruszonych tematów stanowi ona kontynuację poprzednio zorganizowanych konferencji w latach 1972, 1973 oraz 1978.

Na konferencję zgłoszono 48 referatów (w tym kilka z NRD i Czechosłowacji), z których część została wygłoszona na sesji plenarnej, natomiast pozostałe podczas obrad w następujących sekcjach:

sekcja I - "Projektowanie systemów KWP"
sekcja II - "Problemy praktyki KWP"
sekcja III - "Programowanie rozwoju i wdrażania metod informatyki"
sekcja IV - "Bank danych w skomputeryzowanych systemach zarządzania"

Na uwagę zasługuje wygłoszony podczas obrad plenarnych referat Olgi Bortkiewicz-Stulińskiej "Wpływ komputeryzacji na twórczość projektancką", przekazujący refleksje autorki na temat wpływu stosowania komputerów w procesie projektowania na oryginalność i nowatorstwo projektów".

Podczas obrad sekcji I poruszane były zarówno ogólne problemy dotyczące wykorzystania maszyn cyfrowych do wspomagania projektowania jak i konkretne, funkcjonujące systemy wspomagające projektowanie. Na uwagę zasługuje tu system generujący automatycznie dokumentację projektową na podstawie informacji wprowadzonych do maszyny cyfrowej podczas projektowania systemu informatycznego za pomocą pakietu programowanego PSL/PSA. Referat ten był prezentowany przez przedstawiciela Uczelnianego Ośrodka Przetwarzania Informacji Akademii Ekonomicznej w Poznaniu.

Dużo dyskusji wzbudził referat dotyczący oprogramowania minikomputera MERA 400, przy czym pytania odnosiły się w zasadzie nie do właściwej treści referatu, ale dotyczyły samego minikomputera MERA 400. W związku z dużym zainteresowaniem tym tematem na następną konferencję postanowiono zaprosić przedstawiciela producenta minikomputera MERA 400, który mógłby udzielić kompetentnych odpowiedzi na pytania stawiane przez uczestników konferencji.

Podczas obrad sekcji II omawiane były głównie istniejące systemy do wspomagania projektowania. Poruszane były problemy dotyczące projektowania modelowego, projektowania wielopoziomowych systemów informatycznych oraz adaptacyjne formy współpracy projektanta z systemem komputerowym wspomagającym projektowanie.

Na obradach sekcji III zaprezentowano istniejące systemy wspomagające projektowanie w różnych dziedzinach, np. projektowanie słupów linii elektroenergetycznych, geometrii tras komunikacyjnych, procesów obróbki części maszyn, obwodów drukowanych i in.

Po raz pierwszy włączono do tematyki obrad konferencji problemy banku danych sekcja IV jako kontynuację cyklu narad naukowo-technicznych organizowanych dotychczas, a dotyczących tej tematyki.

Warto na koniec zauważyć, że trzy referaty wygłosili pracownicy Instytutu Maszyn Matematycznych.

mgr inż. Halina GUTOWSKA

Zastosowanie elektronicznych maszyn cyfrowych w pracach inżynierskich

Sprawozdanie z konferencji

W dniach 23-24 listopada 1981 r. odbyła się w Katowicach konferencja naukowo-techniczna "Zastosowanie elektronicznych maszyn cyfrowych w pracach inżynierskich" zorganizowana przez Radę Oddziału Wojewódzkiego NOT w Katowicach.

W konferencji tej uczestniczył i wygłosił referat niżej podpisany.

Pierwszego dnia odbyły się obrady plenarne, na których wygłoszono 4 referaty oraz obrady sekcji I "Wykorzystanie ETO w zagadnieniach mechaniki konstrukcji", na których wygłoszono 3 referaty.

Drugiego dnia obradowały równolegle: sekcja II "Komputerowe wpomaganie projektowania urządzeń i sieci energetycznych oraz cieplnych", na której wygłoszono 6 referatów oraz sekcja III "Technika komputerowa w zagadnieniach chemii i ochronie środowiska", na której wygłoszono 4 referaty.

Pełne teksty wszystkich referatów opublikowano w materiałach konferencyjnych[x/]. Warto zwrócić uwagę na referaty wygłoszone na posiedzeniu plenarnym. Bardzo ciekawy referat o charakterze informacyjnym wygłosił mgr inż. M.Rustanowicz na temat możliwości zastosowań i perspektyw rozwojowych komputerów biurkowych, do których zaliczył minikomputery rodziny IBM seria 5100, Hewlet-Packard seria 1000 oraz minikomputery takich firm jak DATA GENERAL, DEC, INTEL, WANG i in. W referacie autor przedstawią typową konfigurację komputera biurkowego, jego oprogramowanie oraz krótki przegląd zastosowań. Mgr inż. G.Wyrzykowski przedstawił możliwości zastosowań pakietu procedur graficznych PARYS opracowanych w Centrum Projektowania i Zastosowań Informatyki w Warszawie /dawne ZETO Warszawa połączone z OBRI w Warszawie . Pakiet ten umożliwia m.in. rysowanie wykresów warstwicowych zadanych funkcji, rysowanie konstrukcji przestrzennych z pominięciem linii niewidocznych itp. Użytkownik ma do wyboru 10 typów linii, np. prosta, łamana, krzywa, pogrubiana, przerywana itp. Pakiet zapewnia wyjście na urządzenia kreślące typu CALCOMP, BENSON, KINGMATIC oraz DIGIGRAF. Pakiet może być eksploatowany na maszynie srii RIAD lub IBM o minimalnej pojemności pamięci operacyjnej 256 k bajtów i systemie operacyjnym OS.

Dr inż. A.Berndt wygłosił referat, w którym omówił cele, zadania oraz możliwe do osiągnięcia korzyści ekonomiczne związane ze stosowaniem systemu CHEMOPRONET. Autorzy przedstawili 4 przykłady możliwego zastosowania bark dokładnej informacji czy przykłady te były do końca zrealizowane . Korzyści ekonomiczne możliwe do osiągnięcia w wyniku zastosowania pakietu określa się na ok. kilkuset milionów złotych.

Niżej podpisany przedstawił projekt realizacji zintegrowanego minikomputerowego systemu wspomagania prac inżynierskich.

Na zakończenie należy dodać, że w materiałach umieszczono dodatkowo 13 komunikatów o zrealizowanych systemach i programach komputerowych przeznaczonych do szerszego rozpowszechniania. Nie zadbano jednak o podanie nazw i adresów instytucji, które dysponują tymi programami. Brak jest także w materiałach informacji o miejscach pracy autorów, co utrudnia kontakt z nimi.

dr inż. Ryszard PAWLIK

BIBLIOTEKA GŁÓWNA
Politechniki Śląskiej
P 3057/81

WARUNKI PRENUMERATY

Prenumeratę na kraj przyjmują Oddziały RSW "Prasa-Książka-Ruch" oraz urzędy pocztowe i doręczyciele w terminie do dnia 25 listopada na rok następny.

Cena prenumeraty rocznej zł 840.

Jednostki gospodarki uspołecznionej, instytucje, organizacje i wszelkiego rodzaju zakłady pracy zamawiają prenumeratę w miejscowych Oddziałach RSW "Prasa-Książka-Ruch", w miejscowościach zaś, w których nie ma Oddziałów RSW - w urzędach pocztowych.

Czytelnicy indywidualni opłacają prenumeratę wyłącznie w urzędach pocztowych i u doręczycieli.

Prenumeratę ze zleceniem wysyłki za granicę przyjmuje RSW "Prasa-Książka-Ruch". Centrala Kolportażu Prasy i Wydawnictw, ul. Towarowa 28, 00-958 Warszawa, konto PKO Nr 1153-201045.

Prenumerata ze zleceniem wysyłki za granicę jest droższa od prenumeraty krajowej o 50% dla zleceniodawców indywidualnych i o 100% dla zlecających instytucji i zakładów pracy.

INDEKS 35362