ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1988

Seria: TRANSPORT z. 6 Nr kol. 908

Roman KONIECZNY

O PEWNYCH KONWENCJACH NOTACJI ALGORYTMÓW PROGRAMÓW KOMPUTEROWYCH

Streszczenie. Artykuł traktuje o zagadnieniach zapisu algorytmów programów komputerowych. Wyszczególnione zostały cechy dobrej notacji algorytmu. Dokonano również zestawienia stosowanych konwencji notacyjnych. Wybór właściwej konwencji notacyjnej w poważnej mierze rzutuje na jakość projektu oprogramowania komputera. Stosowane konwencje notacyjne zestawiono w artykule w następujących grupach:

- konwencja werbalistyczna (oparta na opisie zbliżonym do języka naturalnego),
- konwencja graficzna (oparta na symbolice rysunkowej),
- konwencja symboliczna (oparta na abstrakcyjnych sposobach opisu),
- konwencja metaprogramistyczna (oparta na zapisie zbliżonym do programu komputerowego).

Przedstawione w artykule zagadnienia winny być również - zdaniem autora - szerzej eksponowane w dydaktyce przedmiotów informatycznych.

1. Wprowadzenie

Problem formalizacji opisu świata i zjawisk w nim zachodzących pojawił się około pięć tysięcy lat temu, kiedy to człowiek po raz pierwszy zaczął posługiwać się pismem. Następnym krokiem milowym w rozwoju formalizacji opisu świata było opracowanie przez al-Chorezmiego (787-847) pewnej procedury rachunkowej - nazwanej później algorytmem 10 . Z chwilą pojawienia się komputerów powstał problem notacji algorytmów programów komputerowych. Zagadnienie to obecnie staje się szczególnie ważne - na etapie projektowania oprogramowania - kiedy przed informatyką spiętrzyły się takie zadania, jak: przetwarzanie wiedzy, modelowanie systemów wielkich czy też przetwarzanie języka naturalnego. Zadania te zmuszają do bardziej energicznych poszukiwań w zakresie ulepszenia notacji algorytmów. Ma to również odniesienie do projektowania systemów komputerowych dla potrzeb transportu, a także do dydaktyki przedmiotów informatycznych w szkołach wyższych na kierunku Transport.

2. Cechy dobrej notacji

Dobrą notację algorytmu powinny charakteryzować następujące cechy:

- wysoki szczebel abstrakcji (połączony z widzeniem szczegółów oraz wyjątków),
- uniwersalizm (ta sama notacja służyć może do opisu różnych zjawisk),
- zwięzłość opisu (notacja umożliwia zminimalizowanie liczby stron dokumentacji projektu oprogramowania),
- jednoznaczność (notacja wyklucza różne interpretacje tego samego zapisu),
- czytelność (zapis algorytmu powinien umożliwiać czytanie go "jak z książki", tzn. od lewej do prawej... z góry na dół... bez konieczności wędrówki po odsyłaczach),
- widzialność programu (czytelnik powinien "widzieć" program w jednym ze znanych języków programowania lub w języku łatwym do zdefiniowania i zaimplementowania na bazie istniejącego sprzętu).

Przyjmijmy dla przykładu, że chcemy oprogramować model systemu kolei europejskich. W systemie tym mogą nas interesować różne poziomy abstrakcji - - od podsystemu danego kraju, aż po silnik lokomotywy wybranego pociągu... Przy tak postawionym zadaniu wskazana byłaby notacja, która połączyłaby formalizm teorii systemów, np. [5, 11] z pokazaniem programowalności w jednym z języków wysokiego poziomu, np. ADA, SIMULA czy LOGLAN.

3. Stosowane konwencje notacyjne

Stosowane konwencje notacji algorytmów programów komputerowych można umownie podzielić na następujące grupy:

- konwencja werbalistyczna,
- konwencja graficzna,
- konwencja symboliczna,
- konwencja metaprogramistyczna.

3.1. Konwencja werbalistyczna

Konwencja ta polega na słownym opisie lagorytmu. Wyróżnić tutaj można następujące sposoby opisu:

- opis zarysowy,
- opis typu przepis,
- opis typu HIPO,
- opis lingwistyczny.

Opis zarysowy naświetla w pewnym sensie "literacką" wizję algorytmu. Na przykład: "...Znana jest struktura systemu kolejowej trakcji elektrycznej. Mając określony rozkład jazdy pociągów na poszczególnych liniach, obliczyć

zużycie energii na wykonanie pracy przewozowej w ciągu jednej doby. Wyniki przedstawić w formie kolorowych wykresów przestrzennych..."

Innym sposobem jest opis o charakterze przepisu, zawierającego wyspecyfikowany ciąg działań (kroków) do wykonania. Na przykład:

KROK 1: Wprowadzenie danych opisujących strukturę systemu kolejowej trakcji elektrycznej.
KROK 2: Określenie zadań przewozowych.
KROK 3: Generowanie rozkładu jazdy pociągów.
KROK 4: Symulacja pracy systemu.
KROK 5: Ocena wyników symulacji.
Przejście do KROKU 3, gdy wyniki są niezadowalające.
KROK 6: Redakcja i wyprowadzanie wyników końcowych.

Opis typu HIPO (od ang. Hierarchy plus Input-Process-Output, tj. hierarchia plus wejście-proces-wyjście) jest dalszym rozwinięciem sposobu typu przepis. Opis HIPO zaproponowany został przez firmę IBM [6, 17]. Przykładem takiego opisu może być:

SYSTEM PRZEWOZÓW KOLEJOWYCH

1. WPROWADŹ STRUKTURĘ SIECI KOLEJOWEJ.
2. WPROWADŹ STRUKTURĘ OTOCZENIA.
3. WPROWADŹ ZADANIA PRZEWOZOWE.
4. GENERUJ ROZKŁAD JAZDY.
5. SYMULUJ PRACĘ SYSTEMU.
6. WYPROWADŹ WYNIKI POŚREDNIE.

.....

2.1. WPROWADŹ STRUKTURĘ OTOCZENIA ENERGETYCZNEGO.
2.2. WPROWADŹ STRUKTURĘ ROZMIESZCZENIA STACJI ŁADUNKOWYCH.
.....

2.1.1. WPROWADŹ DANE O UKŁADZIE ELEKTROENERGETYCZNYCH LINII ZASILAJĄCYCH.
2.1.2. WPROWADŹ DANE OKREŚLAJĄCE ROZMIESZCZENIE PODSTACJI TRAKCYJNYCH.
2.1.3. WPROWADŹ DANE O POSZCZEGÓLNYCH PODSTACJACH.
.....

Dalsze rozwijanie opisu zbliża coraz bardziej do rzeczywistej specyfikacji zadań dla programu komputerowego.

Inaczej wygląda lingwistyczny sposób opisu algorytmu. Opis taki jest stosowany w przypadku braku pełnej informacji o modelowanym obiekcie. Ogólny schemat takiego opisu jest następujący [16]:

JEŻELI X jest w stanie X_1, TO Y jest w stanie Y_1 ,

RÓWNIEŻ

JEŻELI X jest w stanie X_2, TO Y jest w stanie Y_2,

RÓWNIEŻ

.....

.....

.....

RÓWNIEŻ

JEŻELI X jest w stanie X_n, TO Y jest w stanie Y_n,

gdzie: $X_1 \ldots X_n$, $Y_1 \ldots Y_n$ charakteryzują stany wejścia i wyjścia obiektu.

Przykładowo: dla zmiennych lingwistycznych V (prędkość pociągu) oraz W (opory ruchu), traktując je jako zbiory rozmyte, można napisać:

JEŻELI V = NULL TO W = SUPERDUŻE ,

RÓWNIEŻ

JEŻELI V = ZERO TO W = DUŻE ,

RÓWNIEŻ

JEŻELI V = BARDZO MAŁE TO W = MAŁE ,

RÓWNIEŻ

JEŻELI V = MAŁE TO W = ŚREDNIE ,

RÓWNIEŻ

JEŻELI V = ŚREDNIE TO W = DUŻE ,

RÓWNIEŻ

JEŻELI V = DUŻE TO W = BARDZO DUŻE ,

RÓWNIEŻ

JEŻELI V = SUPERDUŻE TO W = SUPERDUŻE .

Konwencja werbalistyczna notacji algorytmów charakteryzuje się wysokim szczeblem abstrakcji, dużą uniwersalnością, łatwą czytelnością, stosunkowo dużą zwięzłością przy jednoczesnej dużej niejednoznaczności i słabej widzialności programu. Gdy zwiększa się szczegółowość opisu, wówczas zmniejsza się jego niejednoznaczność oraz zwięzłość, natomiast lepsza staje się widzialność programu.

3.2. Konwencja graficzna

Konwencja ta polega na rysunkowym przedstawieniu algorytmu. Jest to do chwili obecnej najczęściej spotykana konwencja. Ujęcie algorytmu zbliżone

jest do konwencji werbalistycznej, wzbogaconej o elementy graficzne. Wyróżnić tutaj można następujące sposoby rysowania:

- klasyczny schemat blokowy, np [7, 14],
- schemat typu diagram, np. [2],
- schemat typu drzewo, np. [1],
- schemat typu graf, np. [3],
- schemat uproszczony, np. [18].

Konwencja graficzna nie zawsze jest jednak zadowalająca dla zaawansowanych programistów czy projektantów oprogramowania, którzy potrafią "widzieć" program bez schematu rysunkowego lub nie lubią rysować "pudełek".

3.3. Konwencja symboliczna

Konwencja ta bazuje na stosowaniu różnego rodzaju symboliki w celu zwiększenia jednoznaczności zapisów. Wyróżnić tutaj można następujące sposoby notacji: za pomocą operatorów blokowych (np. Buslenko), za pomocą gramatyk (np. Chomsky, Mc Carthy), a także indywidualistyczne notacje (np. Wymore).

Operatory blokowe wprowadzone jeszcze w latach 50 przez Lapunowa [9], a następnie szeroko stosowane przez Buslenkę [4] w modelowaniu systemów złożonych - odpowiadają w zasadzie elementom schematu blokowego w konwencji graficznej. Przykładowy zapis może mieć następującą postać:

$$^{17}A_1 W_{2\downarrow 18}\ A_3^{6,16} W_{4\downarrow 7}\ L_5\ G_6^4\ ^4L_7\ \ F_8\ M_9\ M_{10}\ A_{11}\ W_{12\downarrow 14}$$

$$F_{13}^{15}\ ^{12}F_{14}\ ^{13,14}L_{15}\ W_{16}^{\downarrow 4}\ A_{17}^{1}\ ^{2}Y_{18}$$

gdzie poszczególne symbole oznaczają:

A_i - blok obliczeń arytmetycznych,

W_i - blok warunkowy,

L_i - licznik,

G_i - generator funkcji stochastycznej,

F_i - generator funkcji deterministycznej,

M_i - blok obliczający wartość maksymalną lub minimalną funkcji,

Y_i - blok wyjściowy - redakcja wyników.

Ponadto: zapis G_6^4 oznacza, że z bloku G_6 sterowanie zostanie przekazane do bloku nr 4; zapis $^{13,14}L_{15}$ oznacza, że operatorowi L_{15} sterowanie może być przekazane z bloku nr 13 lub nr 14; zapis $W_{2\downarrow 18}$ oznacza przekazanie sterowania do bloku nr 18, gdy warunek badany w bloku W_2 nie jest spełniony, strzałka do góry oznaczałaby sytuację odwrotną.

Szczegółowy opis wyspecyfikowanych bloków może być podany w konwencji werbalistycznej (słownie) lub symbolicznej (ciąg wzorów matematycznych), lub w obu tych konwencjach.

Wśród innych notacji na uwagę zasługuje formalizm Mc Carthy'ego. Istotną cechą tego formalizmu jest strategia rekursji, dzięki której zapis algorytmu sprowadzony może być do ciągu definicji rekurencyjnych [3]. Na przykład:

.....

silnia(x) = (x=0 → 1, x*silnia(x-1))
potęga(x,y) = (y=0 → 1, true → x*potęga(x,y-1))

.....

Algorytm zapisany na bazie tej notacji staje się łatwo programowalny w językach typu PASCAL.

Inny przykład to notacja Wymore'a [11] - przeznaczona do opisu systemów złożonych. Stanowi ona połączenie wysokiej abstrakcji matematycznej ze stosunkowo prostą symboliką o zwiększonej opisowości. Poniżej podano uproszczony fragment zapisu w tej notacji.

Przyjmijmy, że system Z będzie szóstką:

$$Z = (S, W, F, Y, T, r)$$

gdzie poszczególne elementy są zdefiniowane następująco:

STANY(Z) = S;
WEJŚCIA(Z) = W ;
BRAMYWEJŚCIOWE(Z) = I, Jeśli W = $\times$ I dla pewnego zbioru niepustego I zbiorów niepustych i $\overline{I} \geq 2$,
$= \{W\}$ w przeciwnym przypadku ;
FUNKCJEWEJŚCIA(Z) = F ;
ZACHOWANIE(Z) = Y;
SKALACZASU(Z) = T ;
ruch(Z) = r .

Na bazie powyższego zapisu można zdefiniować podsystem:

Niech Z i Z^* będą systemami. Wówczas Z jest podsystemem systemu Z^* wtedy i tylko wtedy, gdy:

STANY(Z) $\subset$ STANY(Z^*);
WEJŚCIA(Z) $\subset$ WEJŚCIA(Z^*);
FUNKCJAWEJŚCIA(Z) $\subset$ FUNKCJAWEJŚCIA(Z^*);
SKALACZASU(Z) $\subset$ SKALACZASU(Z^*);

oraz dla każdej funkcji f $\in$ FUNKCJAWEJŚCIA(Z) i każdego t $\in$ SKALACZASU(Z)
(ruch(Z))(f,t) = ograniczenie ((ruch(Z^*))(f,t), STANY(Z)) .

W latach 70 dr T. Oren opracował kompilator, dla którego językiem źródłowym jest język splecionej teorii systemów Wymore'a [13]. Językowi temu na-

dał nazwę GEST, która jest akronimem nazwy GEneral System Theory implementor. Instrukcje przyjmują w tym kompilatorze następującą postać: niech Z będzie systemem, którego STANY(Z) są takie... , WEJŚCIA(Z) takie... itd. Można w ten sposób określić zespół systemów z przyporządkowaniem bram wejściowych i określeniem funkcji wyjścia oraz zażądać odpowiedzi na pytania dotyczące konkretnej trajektorii czasowej lub trajektorii wyjściowych dla zdefiniowanego w ten sposób systemu wynikowego.

Konwencja symboliczna notacji algorytmów - stosowana przede wszystkim do opisu systemów złożonych - charakteryzuje się: dużą uniwersalnością, dużą zwięzłością i jednoznacznością, natomiast gorszą czytelnością. W konwencji tej możliwe jest najbardziej zwięzłe tworzenie zapisów na różnych poziomach abstrakcji, przez co jest ona wygodna do formalizacji takich zagadnień informatycznych, jak np. "samooprogramowywanie się" komputerów, tj. generowanie procedur, macierzy algorytmów, a nawet rekurencyjne generowanie języków programowania. Istnieje również możliwość takiego zdefiniowania symboliki, że zapisy na wysokim szczeblu abstrakcji matematycznej widziane będą wprost jako instrukcje języka programowania.

3.4. Konwencja metaprogramistyczna

Idea wykorzystania języka programowania do notacji algorytmów sięga lat 50. W tym celu opracowano szereg języków, z których najbardziej popularny był ALGOL-60 (ALGOrythmic Language). W literaturze często spotykana była odmiana tego języka, tzw. ALGOL publikacyjny. Chodziło o wykorzystanie instrukcji ALGOLu-60 tylko do notacji algorytmu, a nie do pisania programu. (W tym miejscu należy także przypomnieć interesującą polską próbę w tym zakresie - jaką był język SAKO [12].)

Z chwilą gdy ALGOL-60 stał się niewystarczający do notacji złożonych algorytmów, zaczęto tworzyć inne specjalnie do tego celu przeznaczone metajęzyki. Jako przykład podać tu można następujące opracowania: Pidgin ALGOL [1] do opisu procedur, język MIL do programowania wielkiego, język PSL do opisu problemów [17], czy też język CLAN do opisu procesów współbieżnych [8].

Poniżej podano przykład zapisu w Pidgin ALGOLu [1]:

comment zapis ilustruje algorytm Dijkstry wyznaczania najkrótszych dróg o ustalonym porządku.

WEJŚCIE: Graf niezorientowany $G = (V,E)$, wierzchołek początkowy $v_0 \in V$ i funkcja l przyporządkowująca każdej krawędzi liczbę nieujemną, zwaną długością krawędzi. Przyjmujemy, że $l(v_i,v_j)$ jest równe $+\infty$, jeżeli $i \neq j$ i nie prowadzi krawędź do v_i do v_j, oraz że $l(v,v) = 0$.

WYJŚCIE: Dla każdego $v \in V$ minimalna wartość sumy wartości krawędzi drogi P prowadzącej od v_0 do v.

METODA: Konstruujemy zbiór $S \subseteq V$ taki, że najkrótsza droga od v_0 do każdego wierzchołka zbioru S prowadzi przez wierzchołki zbioru S. Tablica D [v] zawiera aktualne długości najkrótszych dróg od v_0 do v prowadzących przez wierzchołki zbioru S.

```
begin
1.   S←{v0};
2.   D[v0] ← 0 ;
3.   for each v ∈ V - {v0} do D[v]←l(v0,v);
4.   while S ≠ V do
        begin
5.          wybierz ze zbioru V - S wierzchołek w, dla którego D [w]
            jest najmniejsze;
6.          dodaj  w do S ;
7.          for each v ∈ V - S do
8.              D[v]←MIN(D [v], D[w] + l(w,v))
        end
end
```

Z innych zasługujących na odnotowanie języków wyróżnić należy: LOGLAN oraz MIL. Języki te - obok Pidgin ALGOLu - stanowić mogą wiodące przykłady dla konwencji metaprogramistycznej.

W LOGLANie [15] łatwo można opisywać złożone typy i struktury danych, a także procesy równoległe. Dzięki mechanizmowi prefiksowania (tj. składania modułów) możliwe jest opisywanie systemów hierarchicznych o różnych poziomach abstrakcji. Dzięki wbudowanej klasie SIMULATION język ten może być również użyty do modelowania zagadnień transportowych.

Opracowanie języka MIL (Module Interconnection Language) miało za zadanie ułatwić osiąganie następujących celów [17]:

1. Zachęcenie do określenia struktury systemu przed rozpoczęciem programowania szczegółów.
2. Zachęcenie do programowania modułów przy założeniu poprawnego środowiska (tj. otoczenia programowego), lecz bez znajomości jego nieistotnych szczegółów.
3. Zachęcenie do tworzenia hierarchii systemu przy równoczesnym zezwoleniu na swobodne, lecz zdyscyplinowane połączenia między modułami.
4. Zachęcenie do ukrywania informacji i budowy maszyn wirtualnych, tj. podsystemów, których wewnętrzna struktura jest ukryta, ale które dostarczają pożądanych zasobów.
5. Zachęcenie do opisu wzajemnych powiązań modułów (tj. części programu) niezależnie od opisu samych modułów.

Wszystkie powyższe zabiegi wprowadzone w języku MIL umożliwiają płynne przejście od abstrakcyjnego opisu formalnego systemu - wyrażanego często w konwencji symbolicznej - do konkretnego projektu oprogramowania dla dowolnego typu komputera.

4. Uwagi końcowe

W tabeli 1 zestawiono omówione w artykule konwencje notacyjne. Z zaprezentowanych sposobów notacji algorytmów konwencja metaprogramistyczna, z uwagi na najlepszą widzialność programu - wydaje się być najbardziej predystynowania do tworzenia projektów oprogramowania złożonych systemów.

Tabela 1

Stosowane konwencje notacji algorytmów programów komputerowych

1.	KONWENCJA WERBALISTYCZNA			
	1a) Opis zarysowy	1b) Opis typu przepis	1c) Opis typu HIPO	1d) Opis lingwistyczny
2.	KONWENCJA GRAFICZNA			
	Stosowany najczęściej standard - rysunkowego odwzorowania schematu blokowego programu.			
3.	KONWENCJA SYMBOLICZNA			
	3a) Operatory blokowe Lapunowa	3b) Gramatyki np. Chomsky, Mc Carthy	3c) Notacja Wymore'a	3d) Indywidualistyczne notacje
4.	KONWENCJA METAPROGRAMISTYCZNA			
	4a) Język programowania wykorzystywany do notacji algorytmu (np. ALGOL 60)			
	4b) Specjalny metajęzyk przeznaczony do notacji algorytmów, np. Pidgin ALGOL publikacyjny, język PSL do opisu problemów.			

Przedstawiony artykuł sygnalizuje jedynie problematykę, której - zdaniem autora - poświęca się stanowczo zbyt mało uwagi w publikacjach informatycznych. Również w dydaktyce często zapomina się o tym - ucząc za wszelką cenę programowania w jakimś języku (najczęściej uzależnionym od posiadanego sprzętu komputerowego), a nie zwracając uwagi na zagadnienia szersze i bardziej uniwersalne - związane z opisem formalnym rozpatrywanych kwestii.

Oczywiście, wybór konwencji notacyjnej jest sprawą trochę filozoficzną, podobnie jak wybór metodologii opisu świata... Opis ten może być dokonany na płaszczyźnie matematycznej lub artystycznej; za pomocą równań różniczkowych lub zdarzeń, topologii lub rekurencji... Pamiętać jednak należy, jak pisał prof. W.M. Turski w książce "Propedeutyka informatyki", że „Dobór właściwych konwencji notacyjnych stanowi często nie tylko o czytelności przeka-

zywanych treści, ale także o klasie wiadomości, które chcemy przekazać. Konwencje notacyjne wpływają także na styl, w jakim są wyrażone wiadomości". Rozszerzając tę myśl, można postawić tezę, że w przyjętej notacji tkwi lepsze lub gorsze rozwiązanie problemu... Tkwi sukces lub porażka projektu.

LITERATURA

[1] Aho A.V., Hopcroft J.E., Ullman J.D.: Projektowanie i analiza algorytmów komputerowych. PWN, Warszawa 1983.

[2] Bauer F.L., Goos G.: Informatyka. WNT, Warszawa 1977.

[3] Brady J.M.: Informatyka teoretyczna w ujęciu programistycznym. WNT, Warszawa 1983.

[4] Buslenko N.P.: Modielirowanije złożnych sistiem. Wyd. "Nauka", Moskwa 1978.

[5] Buslenko N.P., Kałacznikow W.W., Kowalenko O.N.: Teoria systemów złożonych. PWN, Warszawa 1979.

[6] HIPO - A Design Aid and Documentation Technique. Form GC20 - 1951, IBM.

[7] Hughes J.K., Michtom J.I.: A structured Approach to Programming. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1977.

[8] Iszkowskie W., Maniecki M.: Programowanie współbieżne. WNT, Warszawa 1982.

[9] Jerszow A.P.: Wprowadzenie do teorii programowania. WNT, Warszawa 1981.

[10] Kaufmann H.: Dzieje komputerów. PWN, Warszawa 1980.

[11] Klir G.J. (praca zbiorowa): Ogólna teoria systemów. WNT, Warszawa 1976.

[12] Łukaszewicz L., Mazurkiewicz A.: System automatycznego programowania SAKO. Wyd. PAN, Warszawa 1966.

[13] Oren T.I.: GEST: General System Theory Implementor. A Combined Digital Simulation Language. Doctoral Dissertation, University of Arizona, Tucson 1971.

[14] Polski Komitet Normalizacji i Miar: PN-75/E-01226 - Przetwarzanie danych. Symbole graficzne. Wyd. Normalizacyjne, Warszawa 1976.

[15] Salwicki A.: Metodologia programowania w LOGLANIE. Materiały Jesiennej Szkoły PTI, Serock 1985.

[16] Stachowicz M.S., Kochańska M.E.: Identyfikacja obiektów o niepełnej informacji. Materiały Konferencji Naukowej "Aktualne problemy transportu szynowego", Politechnika Krakowska, grudzień 1985.

[17] Turski M.W.: Metodologia programowania. WNT, Warszawa 1978.

[18] Wawiłow A.A. (praca zbiorowa): Imitacjonnoje modielirowanje proizvodstviennych sistiem, "Maszinostrojenije", Moskwa, "Technika", Berlin 1983.

Recenzent:
Doc. dr Jan Wiesner

Wpłynęło do Redakcji 29.09.1986

О НЕКОТОРОЙ КОНВЕНЦИИ НОТАЦИИ АЛГОРИТМОВ КОМПЬЮТЕРНЫХ ПРОГРАММ

Р е з ю м е

В статье затрагиваются вопросы записи алгоритмов компьютерных программ. Выделены свойства хорошей нотации алгоритма. Сопоставлены также применяемые конвенции нотаций. Выбор нужной нотационной конвенции в большой степени определяет качество проекта программного обеспечения компьютера. Применяемые конвенции сопоставлены в статье.

Представленные в работе проблемы должны быть также, по мнению автора, экспонированы в дидактике предметов по информатике.

ON SOME CONVENTIONS OF ALGORITHMS NOTATION FOR COMPUTER PROGRAMS

S u m m a r y

Problems of program algorithms notations are discussed. Properties of good notations are presented. Notation conventions used in practice are set up. The proper choice determines the quality of computer programming design. The notation conventions are gathered in the following groups:

- Word - descriptive convention (based on the description near to the natural one)
- Graphical convention (based on the drawing symbols)
- Symbolic convention (based on the abstractive ways of description)
- Metaprogrammistic convention (based on the description close to the computer program)

Problems presented in the paper should be more extensively presented in the computer science education.