Beata Jankowska,
Instytut Automatyk, Robotyki i Inżynierii Informatycznej

IX. Podstawowe pojęcia z zakresu programowania. Style i języki programowania

1. Pojęcie problemu

1.1. Przez **problem** rozumie się ogólne, najczęściej sparametryzowane pytanie, na które należy udzielić odpowiedzi. Problem jest zadany, gdy zostaną przedstawione:
- opis parametrów problemu,
- pytanie,
- charakterystyka postaci odpowiedzi (poszukiwanego rozwiązania).

Przypadek szczególny problemu (możliwy przykład) uzyskuje się po ustaleniu wartości wszystkich parametrów problemu.

1.2. Przykład problemu: problem osiągalności w grafie (ang. reachability).
Graf G = (V, E), gdzie V oznacza skończony zbiór wierzchołków, a E – skończony zbiór krawędzi (par wierzchołków).
Czy dla zadanego grafu G i danych dwóch wierzchołków i, j $\in$ V istnieje ścieżka prowadząca od wierzchołka i do wierzchołka j?
Problem ten ma nieskończenie wiele możliwych przykładów. Problem ma charaketr decyzyjny, oczekiwaną odpowiedzią jest odpowiedź dychotomiczna tak/nie.

2. Pojęcie programowania

2.1. Programowanie to jednoznaczne formułowanie zadań (problemów) i sposobów ich rozwiązania, z przeznaczeniem do wykonania przez automat (komputer, system informatyczny).
Proces programowania polega na odwzorowaniu pewnego fragmentu rzeczywistości w jego opis w języku programowania.

2.2. Powyższe odwzorowanie odbywa się zwykle w kilku krokach, polegających na:
- precyzyjnym wskazaniu dziedziny (fragmentu odwzorowywanej rzeczywistości),

- poddaniu dziedziny procesowi abstrahowania (wybór tych zjawisk, które są ważne ze względu na sformułowany problem),
- budowie abstrakcyjnego modelu dziedziny – modelu konceptualnego,
- analiza skonstruowanego modelu (odkryciu i sformułowaniu właściwości modelu, czyli praw obowiązujących w wybranym fragmencie rzeczywistości),
- implementacji, czyli zapisie odkrytych praw w postaci specyfikacji systemu informatycznego (zbiór precyzyjnych informacji o systemie i jego funkcjonowaniu),
- wykonaniu obliczeń w systemie.

2.3. Język specyfikacji systemu musi być precyzyjny i mieć ustaloną zarówno składnię, jak i semantykę, czyli znaczenia poprawnie skonstruowanych napisów języka. Stosuje się dwie podstawowe metody definiowania znaczeń poszczególnych napisów języka:

- operacyjną – z każdym napisem wiąże się proces jego interpretacji, reprezentowany skończonym ciągiem konfiguracji obliczeniowych w maszynie abstrakcyjnej;
- deklaratywny – interpretacja napisów jest niezależna od przyjętych w modelu rozstrzygnięć operacyjnych, czyli założonych metod przetwarzania informacji:
  - denotacyjny – definiuje się funkcje semantyczne dla napisów elementarnych, a semantykę napisów złożonych komponuje ze znaczeń przypisywanych elementom składowym tych napisów;
  - aksjomatyczny – znaczenie programu definiuje się charakteryzując wpływ poszczególnych elementów tego programu na jego niezmienniki.

2.4. W skrócie, proces programowania obejmuje następujące etapy:

- sformułowanie problemu,
- zbudowanie modelu logiczno-matematycznego,
- określenie metody rozwiązania problemu (zbudowanie algorytmu),

– zakodowanie algorytmu w języku programowania (implementacja algorytmu),
– wykonanie obliczeń w systemie informatycznym.

3. Pojęcie algorytmu

3.1. Algorytm to jednoznaczny, dobrze określony przepis rozwiązania zadanego problemu. Podstawowe właściwości algorytmu:
– działa na danych wejściowych,
– wytwarza dane wyjściowe (wyniki),
– składa się z kroków (akcji),
– jest dobrze określony,
– jest skończony lub cykliczny,
– jest wykonywalny.

3.2. Istnieją problemy, których nie da się rozwiązać (nie istnieje algorytm ich rozwiązania). Jeśli dla zadanego problemu można zdefiniować algorytm, to jest to równoznaczne z możliwością zdefiniowania nieskończenie wielu algorytmów rozwiązania tego problemu.

3.3. Ocena jakości algorytmu odnosi się zwykle do jego złożoności obliczeniowej, czyli wielkości zasobów potrzebnych do jego wykonania (w systemach informatycznych najczęściej interesują nas czas i pamięć). Złożoność zależy od rozmiaru zadania, czyli wielkości zbioru danych wejściowych dla zadania.

3.4. W algorytmie należy zadbać o odpowiednią jego modularyzację. Modularyzacja algorytmu polega na wyodrębnieniu możliwie niezależnych części (modułów) algorytmu, w których wykonuje się czynności prowadzące do realizacji pośrednich celów, z jednoczesnym ustaleniem zasad łączenia tych części w całość. Modularyzacja algorytmów służy do ukrywania szczegółów realizacji pośrednich celów w poszczególnych modułach.
Dobrze wyróżnione moduły mają następujące właściwości:

- są niezależne semantycznie (znaczenie modułu może być zinterpretowane niezależnie od pozostałych części algorytmu),
- są niezależne składniowo (poszczególne moduły mogą być zdefiniowane w różnych językach),
- spełniają zasadę abstrakcji danych (interpretacja informacji przekazywanych pomiędzy modułami nie zależy od interpretacji obowiązujących wewnątrz modułów).

Współcześnie w powszechnym użyciu są moduły w postaci **obiektów** (modelowanie i programowanie zorientowane obiektowo).

4. Pojęcie języka programowania

4.1. Języki do zapisu algorytmów przeznaczonych do wykonywania przez maszynę nazywa się **językami programowania**, a algorytmy zapisane w tych językach – **programami**.

4.2. Języki programowania można klasyfikować na wiele sposobów, na przykład:

- ze względu na sposób definiowania semantyki języka,
- ze względu na poziom abstrakcji zapisów – na języki niskiego i wysokiego poziomu,
- ze względu na obszar zastosowań – na języki ogólnego i specjalnego przeznaczenia,
- ze względu na sposób interpretowania zapisów – na języki kompilowane i interpretowane.

5. Style i języki programowania

5.1. Powstanie i rozwój stylów i języków programowania ma swój początek w pracach matematyczno-logicznych z pierwszej połowy XX wieku. Wprowadzono wtedy intuicyjne pojęcie **funkcji obliczalnej**, oznaczające funkcję, której wartość dla dowolnych wskazanych argumentów można obliczyć w sposób efektywny (w skończonym czasie).

5.2. Na bazie hipotez Churcha i Turinga skonstruowano liczne systemy formalne, na gruncie których definiuje się pojęcie funkcji

obliczalnej. Systemy te wyrastają z trzech następujących nurtów badań nad obliczalnością:
– algebraicznego,
– modelowego lub
– logicznego.

5.3. Logiczny i algebraiczny nurty obliczeń stały się inspiracją dla **deklaratywnego stylu programowania**. Program zapisany w języku deklaratywnym ma zwykle zwięzłą postać. Metoda obliczeń jest bowiem ustalona i zaszyta w mechanizmie wnioskującym, charakterystycznym dla logiki lub rachunku stanowiącego teoretyczną podstawę języka.

5.4. Przykład 1. Dana jest lista elementów. Przetwórz ją w taki sposób, by otrzymać nową listę z unikalnymi wystąpieniami wszystkich elementów występujących na liście zadanej.
Dokonaj tego poprzez zachowanie pierwszych wystąpień elementów z zadanej listy, np.:

```
[a,d,f,f,t,s,a,f,d,x,s] ->
[a,d,f,t,s,x]
```

Oto rozwiązanie problemu skonstruowane w Prologu – jednym z najpoopularniejszych jezyków programowania deklaratywnego.

```
uni(A,B):-
    unik(A,[],C),    % pierwszy argument ”unik” (A)
    reverse(C,B).    % reprezentuje liste wejsciowa,
                     % skracajaca się w trakcie
                     % obliczen;
                     % drugi, pomocniczy ([])
                     % – konstruowana, stopniowo
                     % wydluzajaca sie liste unikatow;
                     % trzeci argument (C) jest
                     % wykorzystywany na koncu
                     % obliczenia
                     % ”reverse” odpowiada za
                     % odwrocenie listy C i umieszcze-
                     % nie jej w pozycji wynikowej B

unik([H1|T1],L,X):-
    not(member(H1,L)),
    unik(T1,[H1|L],X).

unik([H1|T1],L,X):-
    member(H1,L),
    unik(T1,L,X).

unik([],X,X).        %tutaj ma miejsce wykorzystanie
                     % trzeciego argumentu predykatu;
                     % gdy lista poczatkowa jest już
                     % pusta ([]), a na drugiej
                     % pozycji znajduje się pelna
                     % lista unikatow (X),
                     % to przepisujemy te liste
                     % unikatow na pozycje wynikowa –
                     % trzeci argument ”unik”

?-  uni([a,d,f,f,t,s,a,f,d,x,s],X).
X = [a,d,f,t,s,x] .
```

5.5. Przykład 2 (Prolog):

```
perfect(X) :- number(X),
              X > 1,
              Y is X - 1,
              dividers(X,Y,L),
              sum(L,X).

dividers(X,0,[]) :- !.

dividers(X,Y,[Y|T]) :-    M is X mod Y,
                          M == 0,!,
                          Y2 is Y-1,
                          dividers(X,Y2,T).

dividers(X,Y,L) :-    M is X mod Y,
                      M =\= 0,
                      Y2 is Y-1,
                      dividers(X,Y2,L).

sum([X],X):- !.

sum([X|T],S1):-  sum(T,S2),
                 S1 is X + S2.

?- perfect(28).
true.
```

5.6. Przykład 3. W pewnym zbiorze mężczyzn pochodzących z tej samej rodziny zadano relacje pokrewieństwa typu „ojcostwo” i „braterstwo”. Znajdź wszystkie możliwe do odkrycia relacje „stryjostwa” pomiędzy rozważanymi osobami.
Oto rozwiązanie problemu w języku deklaratywnym CLIPS.

```
(deffacts rodzina
  (ojciec piotrek piotr)
  (ojciec michas michal)
  (ojciec wojtus wojciech)
  (brat michal piotr)
  (brat michal lukasz)
  (brat wojciech zygmunt)
  (brat wojtus szymek)
)
(defrule stryj
(ojciec ?a ?b)
(or (brat ?e ?b)
    (brat ?b ?e))
=>
(assert (stryj ?a ?e))
)

CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0    (initial-fact)
f-1    (ojciec piotrek piotr)
f-2    (ojciec michas michal)
f-3    (ojciec wojtus wojciech)
f-4    (brat michal piotr)
f-5    (brat michal lukasz)
f-6    (brat wojciech zygmunt)
f-7    (brat wojtus szymek)
f-8    (stryj wojtus zygmunt)
f-9    (stryj michas lukasz)
f-10   (stryj michas piotr)
f-11   (stryj piotrek michal)
```

5.6. Z modelowego nurtu obliczeń wyrasta **proceduralny (imperatywny) styl programowania**. Program w języku proceduralnym jest zestawem instrukcji definiujących algorytm działania. Stopień ogólności tych instrukcji zależy od poziomu użytego języka programowania: im wyższy poziom języka, tym bardziej ogólne są instrukcje pochodzące z jego repertuaru.

5.7. Przedstaw rozwiązanie problemu z punktu 5.4 w jezyku imperatywnym C. Oto sugerowany początek rozwiązania:

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
void dodaj(int liczba, int **lista);
int _tmain(int argc, _TCHAR* argv[])

{    int l;
     int *wskl;

     while (!feof(stdin))
          {scanf("%d", &l);
           dodaj(l, &wskl);}
     . . . . .
```

5.8. Do grupy języków proceduralnych zaliczają się:
- assemblery.
- większość języków pośrednich, np. P-code dla Pascala,
- niektóre języki wysokiego poziomu, między innymi: Fortran, Basic, Pascal, C, Ada i – w pewnym zakresie – C++, C#, Java.

5.9. Każdy algorytm składa się z opisu podmiotów oraz opisu czynności, które mają być na tych podmiotach wykonane. W programie sformułowanym w języku proceduralnym opisom tym odpowiadają, odpowiednio: dane i instrukcje.