Łukasz Mazur*

# Programowanie z ograniczeniami przy wykorzystaniu ILOG OPL

## 1. Wprowadzenie

### 1.1. Zagadnienia wstępne

Programowanie w logice z ograniczeniami CLP (*Constraint Logic Programing*) łączy w sobie dwa paradygmaty programowania:

1) programowanie logiczne LP (*Logic Programing*),
2) programowanie z ograniczeniami CP (*Constraint Programing*).

W przeciwieństwie do imperatywnych języków programowania, w których podaje się sekwencje poleceń do wykonania, w językach programowania logicznego specyfikuje się zestaw zależności, na podstawie których system dedukcyjny próbuje udowodnić zadane twierdzenie [11].

Programowanie z ograniczeniami z kolei polega na podaniu pewnych ograniczeń, które określają właściwości poszukiwanego rozwiązania, wyrażają one relacje pomiędzy zmiennymi. Połączenie dwóch omawianych metod programowania zaowocowało stworzeniem narzędzia, które w większości zastosowań jest bardziej wydajne od zwykłego programowania logicznego.

### 1.2. Obszary zastosowań

W ostatnich latach programowanie z ograniczeniami jest popularnym sposobem modelowania i rozwiązywania wielu problemów z następujących dziedzin takich jak [3]:

- sztucznej inteligencji,
- problemów kombinatorycznych,
- biologii molekularnej,
- harmonogramowania,
- inżynierii elektronicznej (*lokalizacja błędów*),
- konstruowania efektywnych parserów językowych,
- systemów baz danych (*zapewnienie spójności danych*),
- projektowania obwodów drukowanych.

* Doktorant Wydziału EAIiE, Akademia Górniczo-Hutnicza w Krakowie

Do rozwiązywania wyżej przedstawionych problemów mamy do wyboru dwie metodyki matematyczne programowanie oraz programowanie z ograniczeniami. Która z metod jest lepsza? Odpowiedzi na to pytanie są następujące:

- to zależy od danych,
- jak również od strategii poszukiwania,
- jak i od kombinatorycznej natury samego problemu.

Generalnie potrzebujemy narzędzi wspomagających obie metodologie rozwiązywania zadań. W trakcie stosowania ich do konkretnego problemu porównujemy ich przydatność i wybieramy lepsze. Liniowe oraz całkowite programowanie zawsze posiada funkcję celu, natomiast programowanie z ograniczeniami, a w szczególności problem spełnienia ograniczeń może mieć wiele możliwych rozwiązań. Specjaliści od programowania z ograniczeniami powiedzą iż mają „rozwiązanie", co w świecie nomenklatury programowania matematycznego będzie brzmieć „rzeczywiste rozwiązanie".

Słownikowe odpowiedniki dla badanych metodologii programowania są następujące [9]:

| **Programowanie matematyczne** | **Programowanie z ograniczeniami** |
|---|---|
| Rozwiązanie dopuszczalne | Istnienie rozwiązania lub nie |
| Rozwiązanie optymalne | Zoptymalizowane rozwiązanie |
| Zmienna decyzyjna | Zmienna ograniczeniowa |
| Zmienna ustalona | Zmienna związana |
| Wzmocnienie granicy | Redukcja dziedziny |
| Iteracyjne rozwiązywanie | Propagacja ograniczeń |

Główną zaletą programowania z ograniczeniami jest jego deklaratywność, czyli sformułowanie zadania zawiera jednocześnie deklarację wybranej standardowej metody rozwiązywania problemu. Programowanie to bazuje na modelowaniu zadania jako problemu spełnienia ograniczeń CSP (*Constraint Satisfacton Problem*). Ograniczenia są zależne od dziedzin zmiennych, których dotyczą. Najpopularniejszą i pierwszą dziedziną zmiennych była skończona dziedzina liczb naturalnych. Innymi dziedzinami są: skończone zbiory, drzewa, rekordy oraz przedziały rzeczywiste. Najistotniejszą cechą i największą zaletą programowania z ograniczeniami jest ich propagacja, jako element wyróżniający tę konwencję spośród innych, co w dalszej części artykułu zostało wyjaśnione.

## 2. Typy ograniczeń

Patrząc ogólnie wyróżnić można następujące typy ograniczeń, możliwych do wykorzystania podczas formułowania problemu (*zadania*) w omawianym środowisku programistycznym ILOG OPL [10]:

– Ograniczenia logiczne:

  Jeżeli x jest równe 4, to y jest równe 5.
  Czynność a poprzedza czynność b **lub** czynność b poprzedza czynność a.

– Ograniczenia globalne:

  Wszystkie elementy w tablicy **x** są różne.

– Metaograniczenia:

  Liczba momentów, kiedy dowolny element tablicy x ma wartość 5 jest dokładnie równa 3.

– Ograniczenia na poziomie elementu:

  Koszt przyporządkowania osoby i do zadania j to **cost[job[i]]**, kiedy **job[i]** jest w stanie j.

Przykłady rożnych rodzajów ograniczeń występujących w ILOG OPL, które zostały wcześniej wypisane można następująco zaimplementować w kodzie programu [10]:

– Ograniczenia logiczne:

```
(x = 4) => (y = 5)
(a.end <= b.start) \/ (b.end <= a.start)
```

– Ograniczenia globalne:

```
alldiffrent(x)
```

– Metaograniczenia:

```
sum(i In S) (x[i]) < 5 ) = 3
```

– Ograniczenia na poziomie elementu:

```
Z = y[x[i]]
```

Słowa kluczowe wykorzystywane w ograniczeniach logicznych, jakie są dopuszczalne do zastosowania są zarówno klasycznymi operatorami logicznymi, jak i funkcjami agregującymi:

- **and, or**
- **not**
- **if … then …**
- **min, max**
- **absolute value, piecewise linear**

Operatory logiczne, jakie mamy do dyspozycji w omawianym środowisku, umożliwiają bardziej naturalne formułowanie kryterium dla zapisu problemu w zadaniu.

### 2.1. Problemy, jakie możemy rozwiązywać

Typy problemów, które możemy rozwiązywać za pomocą matematycznego programowania możliwego do wykorzystania w ILOG OPL, zostały wyszczególnione poniżej. Są to m.in. [9]:

- programowanie liniowe,
- mieszane programowanie całkowitoliczbowe liniowe,
- programowanie kwadratowe,
- mieszane programowanie całkowitoliczbowe kwadratowe,
- kwadratowe programowanie z ograniczeniami,
- mieszane całkowitoliczbowe kwadratowe programowanie z ograniczeniami.

Dalej zostały krótko opisane niektóre modele matematycznego programowania, jakie zostało nam udostępnione poprzez wykorzystanie omawianego pakietu.

Problem programowania liniowego możemy zdefiniować następująco:

$$\textit{Minimize} \quad c^T x$$

$$\textit{Subject to} \quad Ax = b$$

$$l \le x \le u$$

gdzie:

$c^T x$ – funkcja celu,
$Ax = b$ – ograniczenia,
$l$ – dolna granica,
$x$ – zmienna decyzyjna,
$u$ – górna granica.

W problemach typu mieszanego istnieją zmienne $\boldsymbol{x}$, które są typu całkowitego:

$$l \le x \le u$$

**Przykład 1**

$$0 \le x1 \le 40$$

gdzie $x1$ – wartość zmiennej jest całkowita.

$$x2, x3 \ge 0.$$

Programowanie kwadratowe z ograniczeniami przedstawiamy jako równanie, w którym dochodzą dodatkowe ograniczenia funkcji kwadratowej:

$$\frac{1}{2} x^T Q_k x + a_k x \le b_k.$$

**Przykład 2**
*Subject to:*

$$-x1 + x2 + x3 \leq 20,$$

$$x1 - 3x2 + x3 \leq 30,$$

$$x1 * x1 + x2 * x2 + x3 * x3 \leq 1.$$

Charakterystyka wyszczególnionych modeli ma na celu uzmysłowienie czytelnikowi, iż autor programu nie jest zdeterminowany do wykorzystywania jednego podejścia podczas formułowania oraz rozwiązywania zadania, lecz ma on dostęp do wielu metodologii, które powinien dobierać w zależności od zaistniałego problemu.

## 3. Wprowadzenie do środowiska ILOG OPL

### 3.1. Krótki przegląd

Od ponad 10 lat nowatorskie rozwiązania firmy ILOG, oferującej oprogramowanie i usługi korporacyjne, pozwala firmom zwiększać elastyczność oraz podnosić wydajność działania. Ponad 1000 globalnych korporacji i ponad 300 wiodących dostawców oprogramowania wykorzystuje reguły biznesowe, technologie optymalizacji i wizualizacji, oferowane przez firmę ILOG oraz jej produkty, w celu zwiększenia stopy zwrotu z inwestycji, tworzenia udanych produktów i usług oraz powiększania przewagi konkurencyjnej.

### 3.2. Narzędzie programistyczne

Wersja *Light* środowiska ILOG OPL została specjalnie przygotowana do testowania oraz celów edukacyjnych. Zawiera pełną dokumentację oraz możliwość formułowania problemów w postaci projektów jak i ich wykonywanie [7].

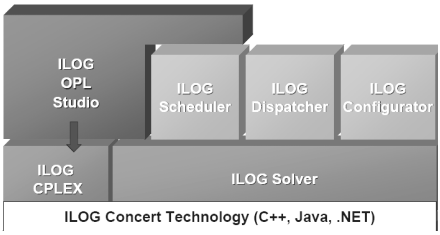
Pomiędzy wersją środowiska oznaczoną jako *Light* a pełną wersją występuje kilka różnic, które w oczywisty sposób ograniczają jego możliwości. Są to m.in.:

- brak interfejsów programistycznych C++,
- ograniczono rozmiar (*złożoność*) formułowanych problemów,
- możliwość uruchamiania tylko na platformie Windows.

Główne cechy środowiska ILOG OPL to [4]:

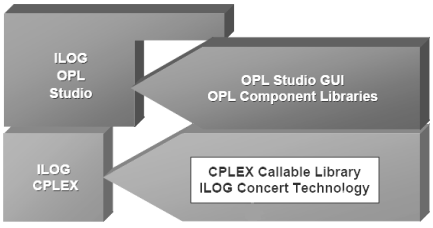
- język służący do modelowania i optymalizacji OPL (*Optimization Programming Language*),
- język wykorzystywany podczas implementacji strategii poszukiwania,
- system programistyczny zawierający potężne predefiniowane algorytmy do rozwiązywania zadań metodą programowania z ograniczeniami.

Architektura komponentowa platformy ILOG OPL została przedstawiona na rysunku 1, w celu zobrazowania, z jakimi modułami obliczeniowymi możemy współpracować w trakcie tworzenia programu [5].




**Rys. 1.** Wspólne API dla bibliotek programistycznych CPLEX oraz modułu ILOG Solver

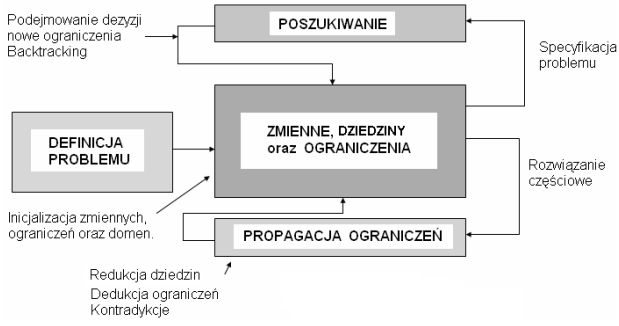
Biblioteki należące do omawianego pakietu programistycznego, zawierają podział na moduły, jak zobrazowano na rysunku 2. Kolejne z prezentowanych modułów zapewniają charakterystyczne swoiste cechy odpowiadające konkretnemu typowi zadania. Stanowią one integralną część całego środowiska.




**Rys. 2.** Przepływ danych w aplikacji zbudowanej w oparciu o bibliotekę CPLEX [5]

## 3.3. Mechanizmy działające w środowisku

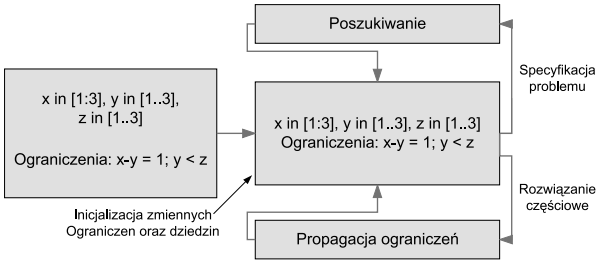
Mechanizmy, które są wykorzystywane podczas rozwiązywania problemu z ograniczeniami poprzez wewnętrznie wykonywany proces propagacji ograniczeń oraz dystrybucji zmiennych dla pakietu ILOG OPL, zostały zademonstrowane na rysunku 3.




**Rys. 3.** Zestaw modułów funkcjonalnych mechanizmu środowiska

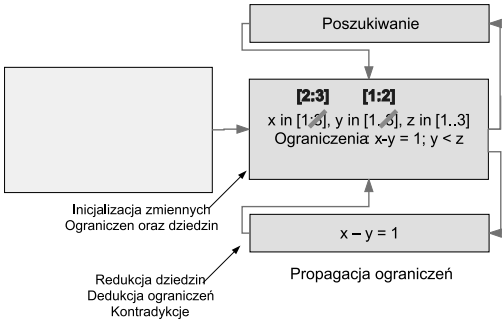
W dalszym etapie pracy zostanie przedstawiony przykład odwzorowujący krokowo działanie poszczególnych mechanizmów w trakcie rozwiązywania problemu, w którym mamy zdefiniowane zadanie klasy programowania z ograniczeniami:

**Krok 1.** Wstępnie następuje inicjalizacja zmiennych, ograniczeń oraz dziedzin (rys. 4).



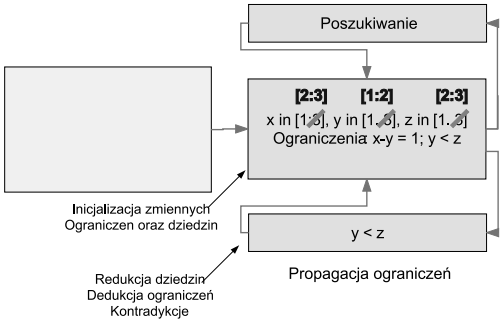

**Rys. 4.** Stan mechanizmu, krok 1

**Krok 2.** Na skutek zadziałania pierwszego z wymienionych ograniczeń oraz jego propagacji następuje częściowa redukcja dziedzin dla zmiennych X i Y (rys. 5).



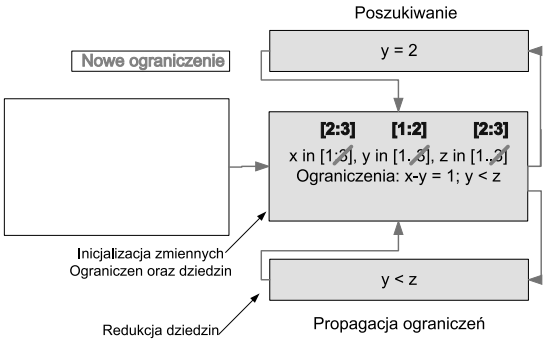

**Rys. 5.** Stan mechanizmu, krok 2

**Krok 3.** Dodając kolejne ograniczenie, zostaje zredukowana dziedzina zmiennej Z (rys. 6).



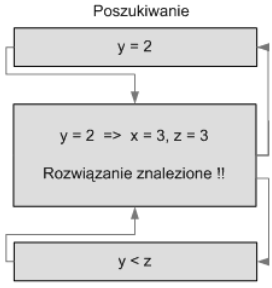

**Rys. 6.** Stan mechanizmu, krok 3

**Krok 4.** Dla zawężonego obszaru poszukiwania wybieramy wartość dla zmiennej Y i kontynuujemy poszukiwania wartości dla reszty zmiennych spełniających ograniczenia (rys. 7).




**Rys. 7.** Stan mechanizmu, krok 4

**Krok 5.** Satysfakcjonujące rozwiązanie, spełniające zadane ograniczenia zostało odnalezione w postaci wartości: X = 3, Y = 2 oraz Z = 3 (rys. 8).




**Rys. 8.** Stan mechanizmu, krok 5

## 4. Algorytmy i poszukiwanie rozwiązania

### 4.1. Algorytmy

W omawianej metodologii programowania można wyróżnić bardzo wiele metod algorytmicznych oraz podejść do wyznaczania rozwiązania. Jednak ze względu na charakter raczej przeglądowy artykułu wprowadzający do omawianych technik zostaną przedstawione najbardziej charakterystyczne metody. Do głównych algorytmów wykorzystywanych w programowaniu z ograniczeniami zaliczyć można następujące podejścia [1]:

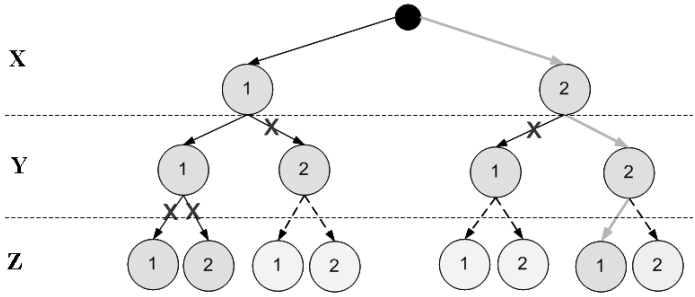
Mamy trzy zmienne X, Y, Z, które mogą przyjmować wartości z zakresu przypisanych do nich dziedzin oraz ograniczenia, które warunkują problem spełnialności (CSP) (rys. 9).

$$X \in \{1,2\}, \quad Y \in \{1,2\}, \quad Z \in \{1,2\}$$
$$X = Y, \quad X \neq Z, \quad Y > Z$$

**Rys. 9.** Definicja zmiennych, dziedzin oraz ograniczeń

**Wycofywanie się** (*Backtracking*)

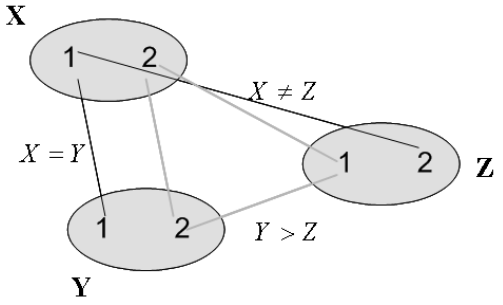
Algorytm dokonuje ukonkretniania kolejnych zmiennych dopóty, dopóki ograniczenia są spełniane, w przeciwnym wypadku cofa się, podstawiając kolejną (*inną*) wartość do zmiennej i kontynuuje poszukiwanie dla następnych zmiennych, odpowiednio podstawiając do nich możliwe wartości z zakresu ich dziedzin, sprawdzając ograniczenia (rys. 10).




**Rys. 10.** Drzewo przeszukiwanych wartości dla zmiennych podczas działania algorytmu

**Spójność łuków** (*Arc Consystency*)

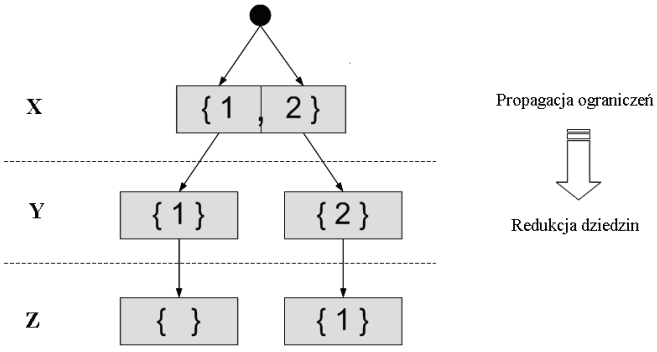
Spójność łukowa sprawdza spełnialność więzów lokalnie pomiędzy nieprzypisanymi zmiennymi. Łuki są spójne, gdy dla wartości każdej zmiennej danego węzła (rys. 11), istnieje przy zadanym ograniczeniu odpowiednia wartość zmiennej z węzła, do którego została ona przypisana [1].




**Rys. 11.** Graf wartości zmiennych oraz relacje ograniczeń pomiędzy nimi

**Sprawdzanie do przodu** (*Forward Checking*)

Procedura ta po aktualizacji nowej zmiennej poprzez spełnienie zadanego ograniczenia, z dziedzin pozostałych zmiennych (rys. 12) usuwa te wartości, które są sprzeczne z już ukonkretnionymi (*zaktualizowanymi*) zmiennymi. Jest to metoda wynikająca z połączenia przeszukiwania z nawracaniem ze sprawdzaniem spójności ograniczeń. Dla tego podejścia można zaznaczyć, iż w praktycznie każdym przypadku otrzymujemy efekty lepsze aniżeli wcześniej przedstawiony algorytm wycofywania się.




**Rys. 12.** Propagacja ograniczeń wraz z aktualizacją dziedzin

## 4.2. Propagacja ograniczeń

Zasadą działania propagacji jest usuwanie wartości niespełniających ograniczeń z dziedzin zmiennych. Języki do programowania z ograniczeniami mają możliwość wyrażania zmiennych z zakresu liczb naturalnych (*najczęściej stosowane*), przedziałów liczb rzeczywistych oraz zbiorów.

## 4.3. Dystrybucja zmiennych

W większości przypadków propagacja ograniczeń nie prowadzi do rozwiązania. Dlatego programowanie z ograniczeniami jest ściśle związane z dystrybucją połączoną z poszukiwaniem. Dystrybucja polega na wprowadzeniu dodatkowego ograniczenia (*często jest to przyporządkowanie jednej wartości do zmiennej, a zadaniem dystrybucji jest odpowiednie wybranie zmiennej i wartości*). Kiedy to nastąpi, sprawdzana jest spójność poprzez propagację ograniczeń.

Istnieją trzy możliwości:

1) znalezione zostanie rozwiązanie (*wszystkie zmienne mają po jednej wartości w swojej dziedzinie*);
2) dziedziny niektórych zmiennych zostaną zawężone, ale jednoznaczne rozwiązanie nie jest wciąż wyznaczone, więc dystrybucja jest dokonywana na kolejnej zmiennej;
3) dodatkowe ograniczenie jest niespójne z pozostałymi ograniczeniami, więc proces nawrotu jest dokonywany, a wybrana wartość z dziedziny wybranej zmiennej jest usuwana.

### 4.4. Poszukiwanie rozwiązania

Ten proces jest dokonywany iteracyjnie i jest nazywany poszukiwaniem. Poszukiwanie jest odpowiedzialne za zatrzymanie; po znalezieniu pierwszego rozwiązania lub pewnej liczby rozwiązań lub wszystkich rozwiązań.

Mowa tu o wielu rozwiązaniach, które mogą, lecz nie muszą występować w trakcie poszukiwania. Gdy wystąpi taka sytuacja, języki programowania z ograniczeniami mają możliwość optymalizacji pewnego zbioru rozwiązań dopuszczalnych, potrzebna jest wtedy funkcja kryterialna, która wyznaczy, w jakim kierunku ma być optymalizowane rozwiązanie aby wynik nas satysfakcjonował.

## 5. Zapis programu w ILOG OPL

Zapis programu w postaci problemu spełnienia ograniczeń (CSP), w którym szukamy dopuszczalnego rozwiązania obejmuje następujące etapy [6]:

1. deklaracje zmiennych, w tym zmiennych decyzyjnych;
2. zdefiniowanie ograniczeń.

### 5.1. Zapis zmiennych

Deklaracja zmiennych:

```
int n; // deklaruje zmienną całkowitą
int+ n; // deklaruje zmienną całkowitą dodatnią
int n in 0..9; // deklaruje zmienną całkowitą o podanym zakresie
```

Inne możliwe typy to: float, string, oraz własne zdefinowane typy.
Można zdefiniować przedziały zmienności:

```
range zakres = 1..20; // zakres liczb całkowitych
range float zakres = 1.5..2.5; // zakres liczb rzeczywistych
range zakres = 1..ilosc(dane); // górny zakres może być zmienny
```

Można stosować typ enumeracyjny:

```
{string} kolory = {"bialy", "czerwony", "czarny"};
// zapisywać dalej jako: mapa["bialy"]
```

Deklaracja zmiennych decyzyjnych może być wykonana następująco [6]:

```
dvar int x;
// prosta deklaracja bez podania dziedziny (przedziału zmienności) x
dvar int+ x in 0..99;
// deklaracja zmiennej x z podaniem jej dziedziny
dvar float w[zakres];
```

```
// szukamy wektora liczb rzeczywistych w, którego indeks podany jest //
poprzez zakres (np. zakres = 0..5)
dvar int y[1..5,1..8] in 0..1;
// szukamy elementów macierzy y o wymiarach 5×8
// mogą być one zerem lub jedynką
```

## 5.2. Zapis ograniczeń

Ograniczenia zapisujemy po słowie kluczowym:

```
Subject to {
                definicja_ograniczeń;
           };
```

Najczęściej stosowane słowa kluczowe przy zapisie ograniczeń globalnych to:

```
forall(i in zakres)
// dla wszystkich wartości z podanego (lub zdefiniowanego wcześniej) //
zakresu np. forall(i in zakres) x[i]>=0;
forall(ordered i in 1..5, j in 1..6)
// dla wszystkich i, j z podanego zakresu takich, że i<j
sum(i in zakres) x[i]<k
// suma wszystkich elementów wektora x, np.: suma ta ma być mniejsza //
od k. Podobnie stosować można funkcje: max()oraz min()
alldifferent(x);
// wszystkie wartości z wektora x muszą być od siebie różne
```

### Ograniczenia logiczne

W omawianym języku mamy do dyspozycji różnego rodzaju konstrukcje logicznych ograniczeń, które są bardzo użytecznym składnikiem formułowanego problemu. Poniżej wyszczególniono zapis kilku z nich, wraz z podaniem odpowiedników w trakcie implementacji [10]:

- Logiczne OR:
  ```
  (job[”Bob”] = Carpenter) \/ (job[”Joe”] = Plumber);
  ```
- Logiczne IF-THEN:
  ```
  (jobs[”Bob”] = Carpenter) => (jobs[”Joe”] = Plumber);
  ```
- Logiczne IF-AND-ONLY-IF:
  ```
  (jobs[”Bob”] = Carpenter) <=> (jobs[”Joe”] = Plumber);
  ```
- Logiczne AND (użyte wraz z IF-THEN):
  ```
  ((jobs[”Bob”] = Carpenter) & (jobs[”Joe”] = Plumber))=>
  (jobs[”Bill”] <> Mechanic);
  ```
- Logiczne NOT:
  ```
  not(j in someset) (jobs[j.person] = j.job)) =>(x = 4);
  ```

### 5.3. Problemy przydziału

Problemy przydziału zadań (*operacji*) do poszczególnych wykonawców (*maszyn, ludzi*) możliwe są do formułowania w różnych konwencjach programistycznych. Podano przykład deklaracji zadania oraz jego zapis jako zarówno programowanie matematyczne, jak i programowanie z ograniczeniami [9]:

Przykładowe zmienne zawierające dane oraz zmienna decyzyjna mogą być zadeklarowane:

```
enum People ...; // literały określające osoby
enum Jobs ...;   // literały określające czynności
float+ cost[People,Jobs] = ...; // koszt przydziału
dvar float+ costperson[People];
```

gdzie konkretne wartości dla danej zmiennej przypisujemy w osobnym pliku (**.dat*).

Programowanie matematyczne umożliwia formułowanie problemu w następujący sposób:

```
dvar x[People, Jobs] in 0..1;
forall (p in People)
sum (j in Jobs) x[p,j] = 1;
costperson[p) = sum (j in Jobs) cost[p,j]*x[p,j];
```

Programowanie z ograniczeniami natomiast dla analogicznego przykładu będzie miało podany niżej charakter:

```
dvar Jobs assign[People];
forall (p in People)
costperson[p] = cost [p, assign[p]];
```

Komentarze w środowisku ILOG OPL stosujemy podobnie jak w języku C++ (//, /* */).

## 6. Przykłady implementacji programów w ILOG OPL

W rozdziale tym zostały przedstawione na klasycznych przykładach problemów z ograniczeniami, sposoby implementacji programów zapisanych w postaci kodu w ILOG OPL, wraz z otrzymanymi wynikami po wykonaniu programu.

### 6.1. Kolorowanie mapy

Kolorowanie mapy Australii trzema kolorami, w taki sposób, aby sąsiadujące pola (*stany*) miały różne kolory, jest to klasyczny problem trójkolorowalności w grafie:

**Listing 1**

Przykład programu wyznaczającego kolorowanie mapy.

```
{string} Country = {„a", „b", „c", „d", „e", „f", „g"}; // sąsiadują-
ce pola, zmienne
string Colors[1..3] = [„red", „green", „blue"]; // tablica kolorów,
dziedzina zmiennych
dvar int color[Country] in 1..3; // zmienna decyzyjna zadania
// deklaracja ograniczeń
constraint c1; constraint c2; constraint c3; constraint c4; constra-
int c5;
constraint c6; constraint c7; constraint c8; constraint c9; constra-
int c10;
minimize color[„a"]; // funkcja kryterium
subject to // zadanie do wykonania
   {
     c1 = color[„a"]!=color[„b"]; // definicje ograniczeń
     c2 = color[„a"]!=color[„c"];
     c3 = color[„b"]!=color[„c"];
     c4 = color[„b"]!=color[„d"];
     c5 = color[„c"]!=color[„e"];
     c6 = color[„c"]!=color[„f"];
     c7 = color[„e"]!=color[„d"];
     c8 = color[„e"]!=color[„f"];
     c9 = color[„b"]!=color[„c"];
    c10 = color[„b"]!=color[„d"];
   }
Execute DISPLAY {
   for(p in Country)
    writeln(„Region [„, p,"] = „, Colors[color[p]]); // wyświetlenie
wyniku zadania
}
```

Problem możemy zaimplementować w podany powyżej sposób. Dopuszczalna jest także inna konwencja programistyczna, stwarza to większą dowolność implementacji. Po wykonaniu programu otrzymujemy rozwiązanie, które jest przyporządkowaniem do odpowiedniego regionu na mapie jednego z trzech wymienionych w dziedzinie kolorów:

```
Region[a] = red, Region[b] = green, Region[c] = blue,
Region[d] = red, Region[e] = green, Region[f] = red, Region[g] = red.
```

## 6.2. Problem plecakowy

Klasyczny problem plecakowy przedstawia się jako problem upakowania jak największej liczby przedmiotów o zadanej wartości właściwości (*objętości*) do określonego mak-

symalną wartością (*pojemnością*) plecaka (*pojemnika*). Należy wybrać taki podzbiór przedmiotów by suma wartości była możliwie jak największa, a suma wag była nie większa od danej pojemności plecaka. Poniżej przedstawiono w celu demonstracji dla przykładowych danych omawiany problem.

**Listing 2**

Przykład programu implementującego problem plecakowy.

```
range items = 1..10; // ilość przedmoitów
int MaxCapacity = 1000;  // maksymalna pojemność
int value[items] = [29,30,25,27,28,21,23,19,18,17];// wartość przedmiotu
int weight[items] = [30,32,33,31,34,40,45,44,39,35];// waga przedmiotu
dvar int take[items] in 0..MaxCapacity;  // zmienna decyzyjna
maximize   // funkcja kryterium optymalizacji
         sum(i in items) value[i]*take[i];
subject to // zadanie do wykonania, ograniczenia
        {
         sum(i in items) weight[i] * take[i] <= MaxCapacity;
        }
```

Po wykonaniu programu dla wyżej sformułowanych danych oraz uwzględnieniu kryterium optymalizacji jak również ograniczeń otrzymujemy wynik w postaci kolejno:

Pierwszego częściowego rozwiązania jako:

```
Partial solution with objective = 960:
take = [30 3 0 0 0 0 0 0 0 0];
```

Drugiego częściowego rozwiązania jako:

```
Partial solution with objective = 962:
take = [28 5 0 0 0 0 0 0 0 0];
```

Finalne rozwiązanie ostateczne jako:

```
Final solution with objective = 962:
take = [28 5 0 0 0 0 0 0 0 0];
```

## 7. Wnioski

Omówiony w niniejszym artykule paradygmat programowania z ograniczeniami w środowisku ILOG OPL doskonale nadaje się do rozwiązywania trudnych kombinatorycznie problemów. Jest on bardzo użyteczny między innymi do rozwiązywania zagadnień alokacji, gdzie skutecznie można go wykorzystywać w problemach przydziału stanowisk do lądowania na lotniskach oraz przydziale miejsc postoju dla statków w portach. Bardzo

dobrze nadaje się także do problemów harmonogramowania zadań w różnego rodzaju gałęziach przemysłowych, takich jak np: środków transportu, naftowym, tworzyw sztucznych oraz przemyśle hutniczym.

Z powodzeniem można modelować i rozwiązywać rzeczywiste zagadnienia o klasie problemu NP, do której zalicza się m.in.: przydział personelu, rozdział dyżurów w szpitalach, przydział załogi do lotów w liniach lotniczych oraz przedsiębiorstwach kolejowych.

Duża liczba udanych zastosowań jak i rozpoczętych projektów w systemach typu automatyki przemysłowej świadczy o trafnym umiejscowieniu omawianej metodyki programowania.

Ze względu na przeglądową formę artykułu o paradygmacie programowania, nie opisano problemów na rzeczywistych przykładach zagadnień, które występują w świecie, ponieważ są one obszerną klasą zagadnień do rozpatrzenia jako kolejny temat.

## Literatura

[1] Bartak R., *http://kti.mff.cuni.cz/~bartak/html/presentations.html.* Materiały z prac badawczych oraz prezentacje. Charles University, Prague, 1999.
[2] Belaid M., *Examples for Discrete Constraint Programming.* Prezentacja referatu problemowego. University of Victoria, 2005.
[3] Bielecki T., Marcinkowski J.: *Constraint Logic Programming = Programowanie logiczne z użyciem ograniczeń*. Referat problemowy. UMK Toruń Wydział Matematyki i Informatyki, 2003.
[4] Clarke L., *Embedding CPLEX using ILOG Concert Technology.* Prezentacja. ILOG Inc, 2004.
[5] Clarke L., *Introduction to ILOG CPLEX.* Prezentacja. ILOG Inc, 2004.
[6] Duda J.: *http://www.zarz.agh.edu.pl/jduda/dydaktyka/iw.html.* Zbiór opracowań w formie elektronicznej, materiałów dydaktycznych. AGH Kraków, 2004.
[7] ILOG Inc.: *www.ilog.com*. Oficjalna strona producenta.
[8] Lustig I., *Introducing NEW ILOG CPLEX 10.0 And NEW ILOG OPL Development Studio 4.2*. Prezentacja. ILOG Inc, 2004.
[9] Lustig I., *Introduction to Constraint Programming and its Relationship to Mathematical Programming: A Constraint Programming Overview.* Prezentacja. ILOG Inc, 2004.
[10] Lustig I., *Modeling and Solving Combinatorial Problems with Constraint Programming using ILOG OPL Studio*. Prezentacja. ILOG Inc, 2004.
[11] Kawulak R., Szczęsny K., *Constraint Logic Programing*. Projekt z przedmiotu „Metody Inżynierii Wiedzy". AGH Kraków AiR, 2006.
[12] Kobylański P.: *http://www.im.pwr.wroc.pl/~przemko/dyd/tw/.* Materiały dydaktyczne z prowadzonych kursów. PW IMiF Wrocław, 2005.