XPCE - grafika w Prologu

1. Co to jest grafika XPCE
2. Tworzenie okien dialogowych z wykorzystaniem XPCE
3. Obiekty graficzne
4. Coś prostszego, czyli diagramy i grafy w programie Graphviz napisane w języku DOT

Co to jest grafika XPCE

XPCE jest zestawem narzędzi do tworzenia graficznego interfejsu użytkownika w Prologu i innych językach programowania. XPCE w odmienny sposób podchodzi do tworzenia GUI.

Jądro XPCE jest obiektowo zorientowane. Pozwala na definicję klas i metod w różnych językach. Wbudowane procedury graficzne zdefiniowane są w C, dzięki czemu prędkość renderowania grafiki jest duża. Ponadto umożliwia to kompilację biblioteki XPCE na różnych platformach. Metody każdej klasy należącej do biblioteki XPCE zdefiniowane są przy użyciu Prologa, dzięki temu mogą one posiadać argumenty będące danymi Prologa i odwrotnie: dane Prologa mogą być powiązane ze zmiennymi XPCE.

Warstwa graficzna XPCE zapewnia wysoki stopień abstrakcji poprzez ukrywanie przed programistą szczegółów, takich jak np. obsługa zdarzeń, odświeżanie okienek, zarządzanie wyglądem okien. Jednocześnie zapewnia dostęp do tych szczegółów w razie potrzeby.

Tworzenie okien dialogowych z wykorzystaniem XPCE

- Tworzenie prostych obiektów

Tworzenie obiektów wykonywane jest z wykorzystaniem predykatu

new/2 (new(?Reference, +NewTerm))

pierwszy argument zwraca referencję do obiektu, a drugi jest rodzajem tworzonego obiektu (klasa obiektu).

```
?- new(Punkcik, point(10,20)).
Punkcik = @8308913/point.
?- new(@demo, dialog('Demo Window')).
true.
```

Utworzone obiekty możemy usunąć korzystając z predykatu, gdzie argumentem jest identyfikator obiektu

free/1 (free(+Reference))

```
?- free(Punkcik).
?- free(@demo).
```

Tworzenie okien dialogowych z wykorzystaniem XPCE

- Modyfikacja obiektów

Aby zmodyfikować aktualny stan obiektu wykorzystujemy predykat

send/2 (send(+Receiver, +Selector(...Args...))).

pierwszy argument to referencja do obiektu, drugi argument to metoda do wykonania.

```
?- send(@8308913, x(20)).
?- send(@demo, append(text_item(name))).
```

Okno pokazujemy w ten sposób:

```
?- send(@demo, open).
```

Okno w Windows 7 oraz Linux

Tworzenie okien dialogowych z wykorzystaniem XPCE

Predykat send można również dekomponować do postaci:

```
?- send(@demo, append, text_item(nazwa)).
```

Z tego zapisu wynika, że wszystkie argumenty od 3 będą argumentami append.

XPCE definiuje kilka podstawowych obiektów takich jak:

@display – referencja ekranu

@prolog – referencja maszyny wnioskującej

W nomenklaturze XPCE metodą jest każdy zdefiniowany predykat w Prologu:

```
?- send(@prolog, write('hello world')).
```

Tworzenie okien dialogowych z wykorzystaniem XPCE

- Odpytywanie obiektów

Kolejnym z fundamentalnych predykatów jest:

get/3 (get(+Receiver, +Selector(+Argument...), -Result))

Pierwsze dwa argumenty to identycznie jak w przypadku send referencja do obiektu i metoda do wykonania, natomiast 3 argument to zwracana wartość.

```
?- get(@8308913, y, Y).

Y = 20

?- get(@830913, distance(point(100,100)), Distance).

Distance = 113

?- get(@demo, display, D).

D = @display/display
```

Ostatni przykład zwraca obiekt poprzez jego referencję.

Tworzenie okien dialogowych z wykorzystaniem XPCE

Sprawdźmy rozdzielczość:

```
?- get(@display, size, Size),
     get(Size, width, W),
     get(Size, height, H).
Size = @7919987/size, W = 1280, H = 800.
```

Sprawdźmy co nam użytkownik wpisał w polu tekstowym okna dialogowego:

```
?- get(@demo, member(name), TextItem),
      get(TextItem, selection, Text).
TextItem = @573481/text_item, Text = hello
```

Tworzenie okien dialogowych z wykorzystaniem XPCE

- Prosta akcja

```
?- new(@okno,dialog(dialogowe)),
   new(@but,button(nacisnij,message(@prolog,write,'au! nie naciskaj!\n'))),
   send(@okno,append,@but),
   send(@okno,open).
```

Tworzenie okien dialogowych z wykorzystaniem XPCE

- Wbudowane elementy okna dialogowego

button	Zwykły przycisk. Realizuje metodę message na wciśnięcie.
text_item	Pole wprowadzania tekstu. Może być edytowalny lub nie, posiada wbudowany tryb konwersji (np. gdy chcemy wprowadzić liczbę), można używać spacji przy wprowadzaniu wartości.
int_item	Podobne do text_item. Posiada ograniczony rozmiar pola, guzik do zwiększenia/zmniejszenia wartości, kontrola typu i zakresu.
slider	Pole wyboru liczb w danym zakresie. Obsługuje zarówno liczby całkowite, jak i zmiennoprzecinkowe.
menu	Realizuje różne style menu wyboru: przyciski radio, przyciski z fajką, pola kombi itd.
menu_bar	Pasek menu.
label	Etykieta z tekstem lub z obrazkiem. Nie podatny na działania użytkownika.
list_browser	Pokazuje listę elementów, obiektów.
editor	Edytor tekstu. Obsługuje wiele czcionek i innych atrybutów tekstowych.
dialog_group	Grupa elementów w oknie dialogowym. Posiada granicę i etykietę.

Obiekty graficzne

XPCE oprócz tworzenia standardowych okien dialogowych, do których każdy użytkownik komputera jest przyzwyczajony, jesteśmy w stanie tworzyć obiekty graficzne.

```
?- new(@p, picture('Demo Picture')),
     send(@p, open).
```

Powyższy program przygotowuje obiekt graficzny o nazwie ‘Demo Picture’. Obrazek, który w ten sposób stworzymy będzie dwuwymiarowy. Ciekawostką może być fakt, że przygotowany obszar roboczy jest niekończenie duży.

Narysujmy kwadrat:

```
?- send(@p, display,
          new(@bo, box(100,100))).
```

Obiekty graficzne

Czas na kółko :

```
?- send(@p, display,
          new(@ci, circle(50)), point(25,25)).
```

Wstawianie grafiki:

```
?- send(@p, display,
          new(@bm, bitmap('32x32/books.xpm')), point(100,100)).
```

Przywitajmy się:

```
?- send(@p, display,
          new(@tx, text('Hello')), point(120, 50)).
```

Obiekty graficzne

Rysujemy krzywą linie:

```
?- send(@p, display,
          new(@bz, bezier_curve(point(50,100),
                                point(120,132),
                                point(50, 160),
                                point(120, 200)))).
```

XPCE umożliwia edycję już narysowanych obiektów graficznych. Na początek zaokrąglijmy rogi naszego kwadratu:

```
?- send(@bo, radius, 10).
```

Obiekty graficzne

Pomalujmy kółko:

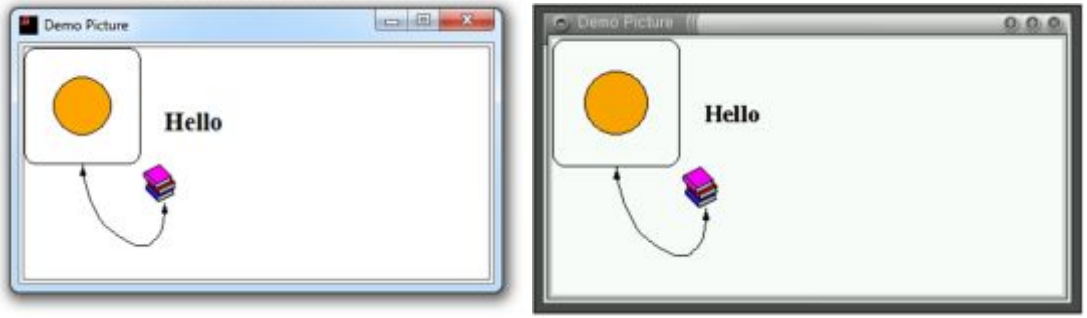
```
?- send(@ci, fill_pattern, colour(orange)).
```

Zmienimy czcionkę powitania:

```
?- send(@tx, font, font(times, bold, 18)).
```

Zrobimy groty:

```
?- send(@bz, arrows, both).
```



Wynik uzyskany w Windows 7 i Linux

Obiekty graficzne

- Podstawowe obiekty graficzne:

arrow	Grot. Zazwyczaj wykorzystywany z klasą line.
bezier	Krzywa Béziera. Obsługiwane są krzywe wielomianowe drugiego i trzeciego stopnia.
bitmap	Wizualizacja obrazu. Obsługiwane są monochromatyczne i kolorowe obrazy. Obrazy mogą mieć kształt złożony.
pixmap	Podklasa bitmap, obsługuje tylko kolorowe obrazy.
box	Prostokąt. Może mieć zaokrąglone rogi i być wypełniony kolorem.
circle	Specjalny przypadek elipsy.
ellipse	Kształt eliptyczny, możliwe wypełnienie.
arc	Część elipsy. Może mieć groty. Wypełniony wygląda jak kawałek ciasta.
line	Linia prosta, może mieć groty.
path	Linia łamana przechodząca przez punkty. Może mieć groty i być zaokrąglona na zgięciach.
text	Wizualizacja stringu w kilku czcionkach. Może mieć wiele atrybutów tekstowych.

Obiekty graficzne

- Interakcja myszy z grafiką

Wykorzystanie obiektu recogniser do obsługi myszy.

```
?- send(@ci, recogniser, move_gesture(left)).


?- send(@tx, recogniser,
           click_gesture(left, '', double,
                         message(@prolog,write,'Hello\n'))).
```

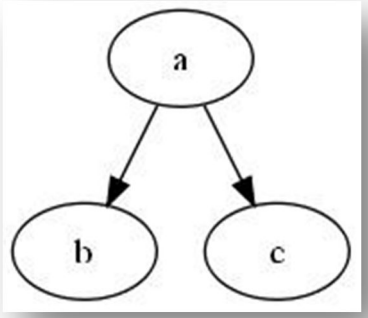
The DOT Language

Język DOT służy do opisywania schematów za pomocą tekstu.

Przy pomocy programu Graphviz możemy zaprojektować grafy i schematy definiując tylko relacje między poszczególnymi węzłami.

W języku DOT można utworzyć dwa rodzaje wykresów:
- grafy zorientowane

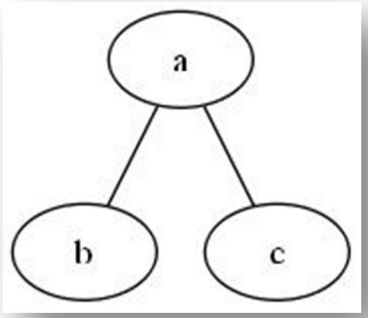
```
digraph przyklad
{
                        a->b;
                        a->c;
}
```



The DOT Language

- grafy niezorientowane

```
graph przyklad
{
      a--b;
      a--c;
}
```



The DOT Language

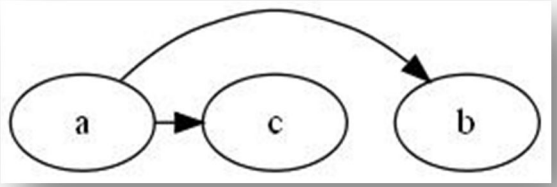
- Podgrafy

Podgrafy w języku DOT pełnią trzy podstawowe role:

- mogą być używane do reprezentowania struktury grafu, wskazując, że niektóre węzły i krawędzie powinny być zgrupowane razem

- może zapewnić kontekst dla ustawiania atrybutów. Na przykład, można określić, że podgraf niebieski to kolor domyślny dla wszystkich węzłów:

```
digraph przyklad
{
	a->b;
	a->c;
	subgraph
	{
		rank = same; a; b; c;
	}
}
```



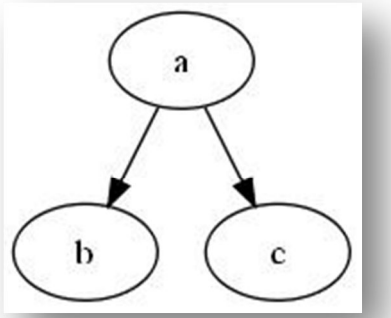
The DOT Language

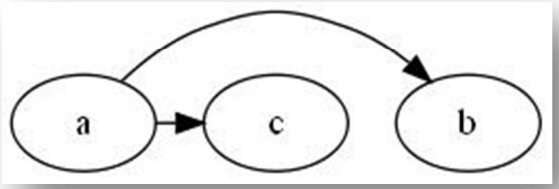
- trzecia rola wiąże się bezpośrednio z silnikiem graficznym Graphviz.

Jeżeli Graphiz napotka słowo kluczowe `subgraph`, to program wszystkie węzły należące do danego podgrafu są wymazywane i na nowo rysowane, zgodnie z instrukcjami zawartymi w podgrafie.

Ważny wniosek:

Podgrafy nie są częścią języka DOT a jedynie konwencją składniową przestrzeganą przez niektóre silniki graficzne.





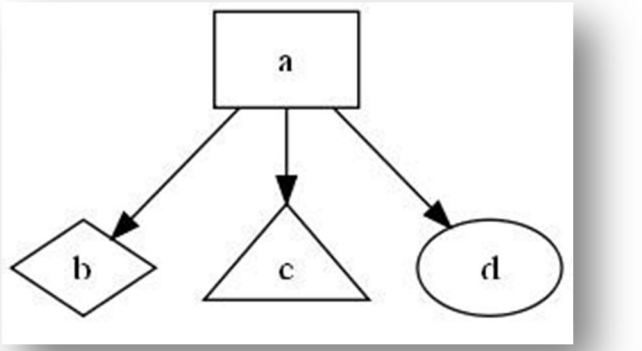
1. Odczyt węzłów metodą post-order i wrzucenie na stos
2. Odczyt węzłów i rysowanie.

The DOT Language

- Atrybuty węzłów

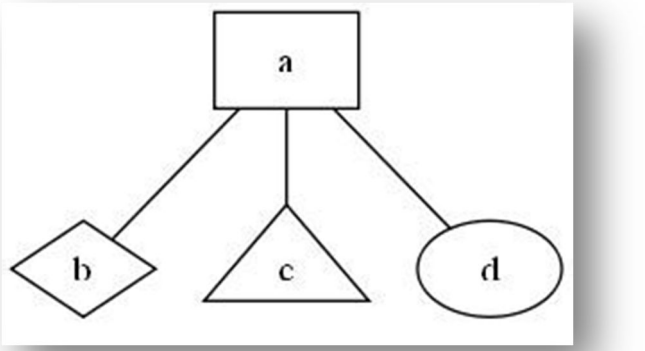
Język DOT umożliwia zmiany kształtu węzłów występujących w danym grafie:

```
digraph przyklad
{
      a->b;
      a [shape=box];
      b [shape=diamond];
      c [shape=triangle];
      a->c;
      a->d;
}
```



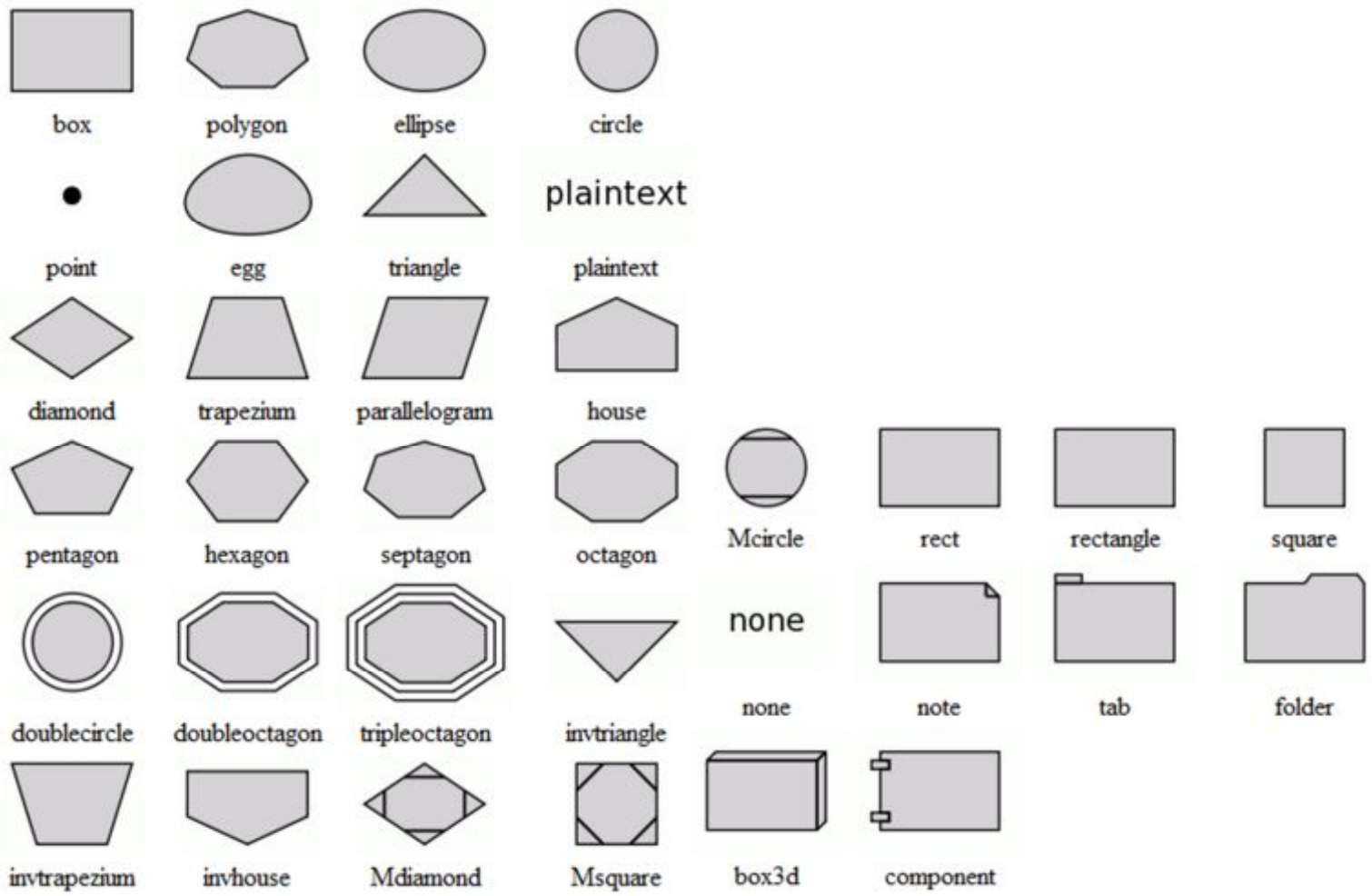
The DOT Language

```
graph przyklad
{
      a--b;
      a [shape=box];
      b [shape=diamond];
      c [shape=triangle];
      a--c;
      a--d;
}
```



The DOT Language

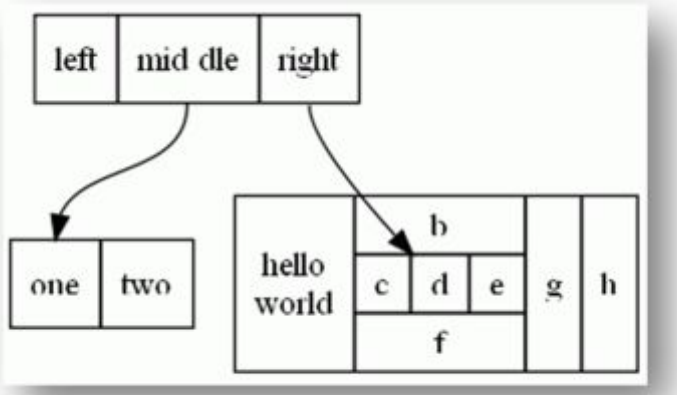
- Dostępne kształty:



The DOT Language

- Działanie na rekordach

```
digraph structs
{
        node [shape=record];
        struct1 [label="<f0> left|<f1> mid dle|<f2> right"];
        struct2 [label="<f0> one|<f1> two"];
        struct3 [label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"];
        struct1:f1 -> struct2:f0;
        struct1:f2 -> struct3:here;
}
```

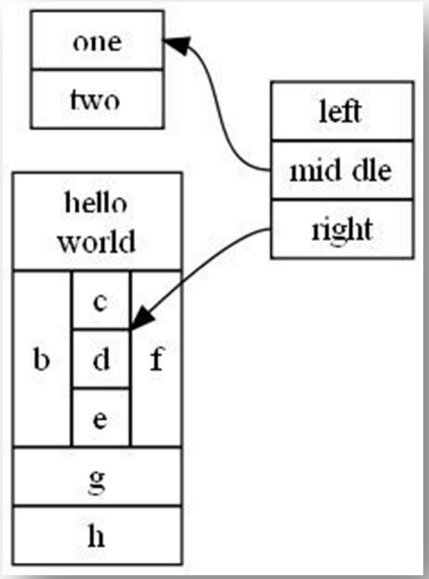


`node` - odwiłanie do wszystkich węzłów

`<etykieta>` - etykieta niewyświetlana w diagramie, wykorzystywana do określenia lokalizacji etykiety głównej

The DOT Language

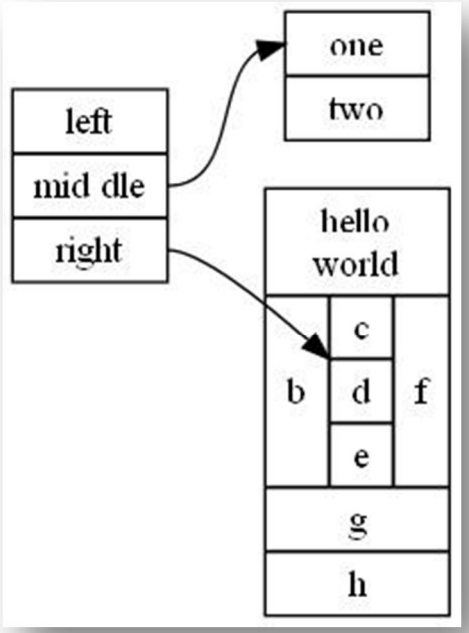
```
digraph structs
{
        node [shape=record];
        struct1 [label="<f0> left|<f1> mid dle|<f2> right"];
        struct2 [label="<f0> one|<f1> two"];
        struct3 [label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"];
        struct1:f1 -> struct2:f0;
        struct1:f2 -> struct3:here;
        rankdir=RL;
}
```



`rankdir` - zmiana orientacji wykresu, domyślnie `"TB"`

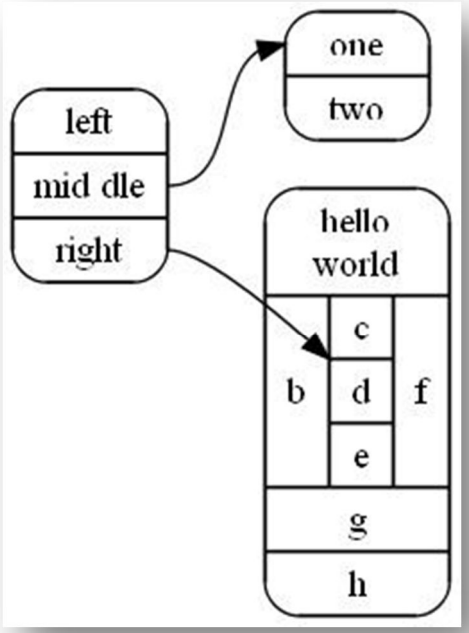
The DOT Language

```
digraph structs
{
        node [shape=record];
        struct1 [label="<f0> left|<f1> mid dle|<f2> right"];
        struct2 [label="<f0> one|<f1> two"];
        struct3 [label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"];
        struct1:f1 -> struct2:f0;
        struct1:f2 -> struct3:here;
        rankdir=LR;
}
```



The DOT Language

```
digraph structs
{
        node [shape=Mrecord];
        struct1 [label="<f0> left|<f1> mid dle|<f2> right"];
        struct2 [label="<f0> one|<f1> two"];
        struct3 [label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"];
        struct1:f1 -> struct2:f0;
        struct1:f2 -> struct3:here;
        rankdir=LR;
}
```



The DOT Language

- Kolorowanie węzłów

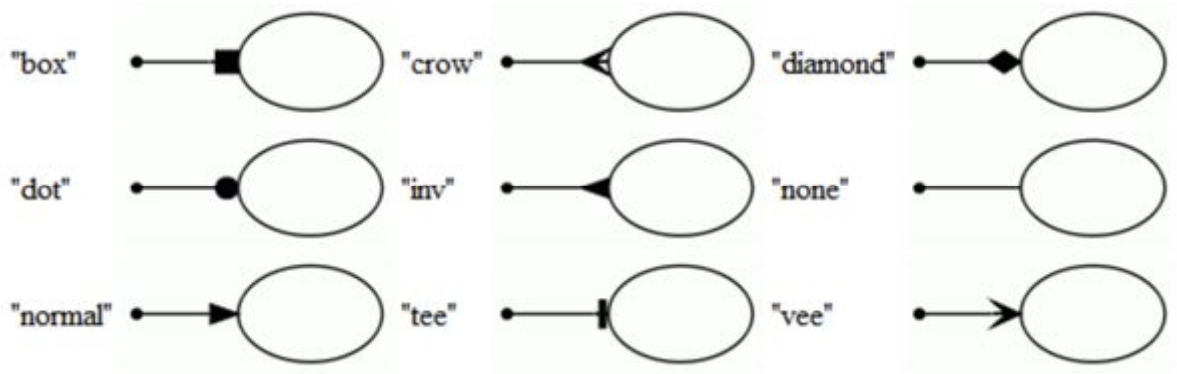
```
digraph G {
  rankdir=LR;
  node [shape=box, color=blue];
  node1 [style=filled];
  node2 [style=filled, fillcolor=red];
  node0 -> node1 -> node2;
}
```


```
digraph R {
  rankdir=LR;
  node [style=rounded];
  node1 [shape=box];
  node2 [fillcolor=yellow, style="rounded,filled", shape=diamond];
  node3 [shape=record, label="{ a | b | c }"];
  node1 -> node2 -> node3;
}
```

`fillcolor` - kolor wypełnienia

The DOT Language

- Rodzaje połączeń



Możliwe są dodatkowe modyfikacje związane grotami:

`l` - pozostawia odpowiedni grot tylko po lewej stronie połączenia

`r` - pozostawia odpowiedni grot tylko po prawej stronie połączenia

`o` - pozostawia niewypełniony grot

Uwaga:
Style strzałek działają tylko i wyłącznie w grafach zorientowanych.

The DOT Language

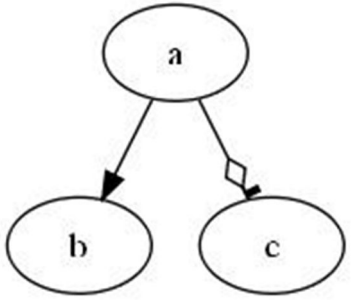
Nie wszystkie rodzaje grotów mają możliwość modyfikacji.

Tabela przedstawia groty i dostępne im modyfikacje:

Modifier	box	crow	diamond	dot	inv	none	normal	tee	vee
'l'/'r'	X	X	X		X		X	X	X
'o'	X		X	X	X		X		

Oprócz powyższych modyfikacji, dostępne jest również łączenie ze sobą kliku stylów grotów:

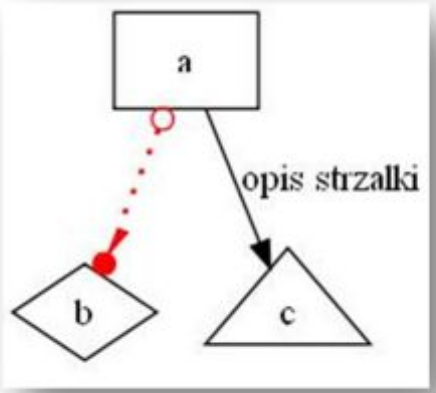
```
digraph R {
  a -> b;
  a -> c [arrowhead="lteeodiamond"];
}
```



The DOT Language

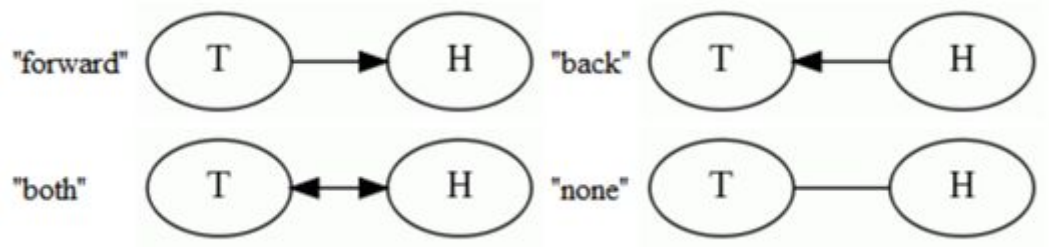
- Zmiana koloru, dodanie grotu na ogon i podpis strzałki

```
digraph przyklad{
      a->b [style="dotted,bold", dir=both, arrowhead=dotlnormal, arrowtail=odot, color=red];
      a [shape=box];
      b [shape=diamond];
      c [shape=triangle];
      a->c [label="opis strzalki"];
}
```



The DOT Language

Atrybut `dir dla T->H;` :



Wniosek:

Można za pomocą grafu zorientowanego można utworzyć graf niezorientowany i odwrotnie.

Atrybut `color` :

Możliwe jest definiowanie koloru na cztery sposoby:

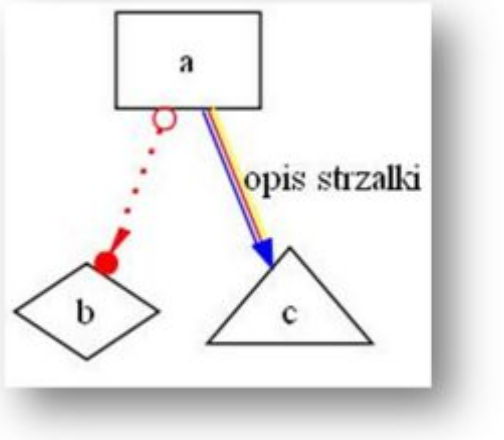
Color	RGB	HSV	String
White	"#ffffff"	"0.000 0.000 1.000"	"white"
Black	"#000000"	"0.000 0.000 0.000"	"black"
Red	"#ff0000"	"0.000 1.000 1.000"	"red"
Turquoise	"#40e0d0"	"0.482 0.714 0.878"	"turquoise"
Sienna	"#a0522d"	"0.051 0.718 0.627"	"sienna"

dodatkowo można zapisać kolor w systemie RGBA ("#%2x%2x%2x%2x")

The DOT Language

Język DOT umożliwia utworzenie listy kolorów:

```
digraph przyklad {
      a->b [style="dotted,bold",dir=both,arrowhead=dotlnormal,arrowtail=odot, color=red ];
      a [shape=box];
      b [shape=diamond];
      c [shape=triangle];
      a->c [label="opis strzalki" color="blue:red:yellow" ];
}
```



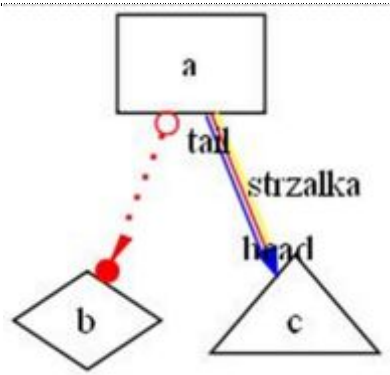
Podając w wybranej kolejności kolory oddzielone ‘:’ możemy utworzyć wielokolorowe obiekty.

The DOT Language

Atrybut `label:`

W przypadku węzłów atrybut `label` definiował tekst jaki ma być umieszczony wewnątrz danego węzła.
Dla połączeń jest to informacja jaka ma być wyświetlona w środku danego połączenia.
Do podpisywania końcowych fragmentów połączeń wykorzystujemy atrybuty:
`taillabel`-podpis jest umieszczany w ogonie danego połączenia
`headlabel`-popis jest umieszczany w głowie danego połączenia

```
digraph przyklad {
      a->b [style="dotted,bold",dir=both,arrowhead=dotlnormal,arrowtail=odot, color=red ];
      a [shape=box];
      b [shape=diamond];
      c [shape=triangle];
      a->c [headlabel="head", label="strzalka", taillabel="tail", color="blue:red:yellow" ];
}
```



Dziękuję za uwagę.

Literatura:

- Biblioteka XPCE w Prologu, Andrzej Mróz, Bartosz Bełcik, 13 stycznia 2003 rok
- Programing in XPCE/Prolog, Jan Wielemaker, Anjo Anjewierden, SWI, Uniwesity of Amsterdam
- http://www.swi-prolog.org/pldoc/refman/
- http://www.graphviz.org/Documentation.php