Politechnika Poznańska
Instytut Informatyki

Adam Meissner

Adam.Meissner@put.poznan.pl

http://www.man.poznan.pl/~ameis

# SZTUCZNA INTELIGANCJA

## Podstawy programowania z ograniczeniami

## Literatura

[1] Apt K.R., *Principles of Constraint Programming*, Cambridge Univ. Press, 2003.

[2] Niederliński A., *Programowanie w logice z ograniczeniami. Łagodne wprowadzenie dla platformy EC-LiPSe*, Wyd. Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 2014.

[3] Van Roy P., Haridi S., *Concepts, Techniques, and Models of Computer Programming*, The MIT Press, Cambridge, USA, 2004.

[4] Mozart Consortium, materiały dotyczące programowania w języku Oz, 2008.

# Wprowadzenie (1)

- w paradygmacie programowania z ograniczeniami (ang. *constraint proramming*, CP) program jest opisem rozpatrywanego problemu (tzw. problemu CSP, ang. *constraint satisfaction problem*), mającym postać zbioru formuł logicznych (tzw. *ograniczeń* lub *więzów*) zawierających zmienne
- rozwiązaniem problemu CSP jest dowolne wartościowanie zmiennych spełniające wszystkie ograniczenia

## Przykład 1

Dany jest zbiór ograniczeń *P*, składający się z następujących elementów:

(a) `X::2#8`,

(b) `Y::7#9`,

(c) `X*Y <: 29`,

(d) `2 * X + Y >: 13`.

Przykładowym rozwiązaniem problemu opisanego przez zbiór *P* jest wartościowanie zmiennych: `X = 3, Y = 8`.

# Wprowadzenie (2)

## Metody rozwiązywania problemu CSP

- metody specyficzne dziedzinowo, np.:
  - algorytmy rozwiązywania układu równań liniowych
  - algebry binarnych diagramów decyzyjnych
- metody ogólnego przeznaczenia, wykorzystujące:
  - *propagację* i *dystrybucję* ograniczeń,
  - algorytmy konstruowania *drzewa poszukiwań.*

## Obszary zastosowań paradygmatu CP

- projektowanie układów elektronicznych
- badania operacyjne (problemy optymalizacji)
- ekonometria
- biologia obliczeniowa (np. partycjonowanie DNA)
- przetwarzanie języka naturalnego.

## Dziedziny standardowe w systemach CP(.)

- FD - skończone zbiory nieujemnych liczb całkowitych
- FS - skończone zbiory, których elementami są dziedziny FD
- B - zbiór {0, 1}
- R - skończone zbiory liczb zmiennoprzecinkowych.

# Rozwiązywanie problemu CSP (1)

## Propagacja i dystrybucja ograniczeń

- rozwiązywanie problemu CSP tą metodą polega na naprzemiennym wykonywaniu na zbiorze ograniczeń dwóch działań, tj. *propagacji* oraz *dystrybucji*
- wykonywanie ww. operacji kończy się w przypadku znalezienia rozwiązania problemu albo uzyskania sprzeczności

## Własności propagacji

- operacja ta polega na sukcesywnym zawężaniu dziedzin zmiennych zgodnie z semantyką więzów w celu usunięcia elementów, z których nie można zbudować żadnego rozwiązania
- porządek kroków propagacji nie ma wpływu na wynik końcowy
- propagacja nie jest operacją wystarczającą do rozwiązania problemu CSP, zwiększa ona jednak efektywność procesu poszukiwania rozwiązania.

## Przykład 2

Na zbiorze ograniczeń z przykładu 1 można wykonać następujące kroki propagacji.

1. Ograniczyć dziedzinę zmiennej `X` do zbioru `2#4`, gdyż liczby większe od `4` nie spełniają ograniczenia (c), nawet gdy zmienna `Y` przyjmuje możliwie najmniejszą wartość (tj. `7`)
2. Ograniczyć dziedzinę zmiennej `X` do zbioru `3#4`, gdyż liczba `2` nie spełnia ograniczenia (d), nawet gdy zmienna `Y` przyjmuje możliwie największą wartość (tj. `9`)

Żadne dalsze zawężanie dziedzin zmiennych nie jest możliwe.

# Rozwiązywanie problemu CSP (2)

## Dystrybucja

- operacja ta polega na utworzeniu na podstawie danego zbioru więzów $P$ nowych zbiorów więzów; najczęściej są to zbiory postaci $P \cup \{F\}$ oraz $P \cup \{\neg F\}$
- formuły $F$ oraz $\neg F$ dzielą dziedzinę pewnej zmiennej na (dwa) rozłączne podzbiory, np. dla zmiennej `X::2#8` rozpatrywane formuły mogą mieć postać `X >:4` oraz `X =<:4`
- *strategia dystrybucji* jest zbiorem reguł określających:
  - którą ze zmiennych *niezdeterminowanych* (tj. zmiennych o dziedzinie liczniejszej niż 1) wybrać do dystrybucji
  - w jaki sposób podzielić dziedzinę tej zmiennej na rozłączne podzbiory
- dystrybucja jest działaniem wystarczającym do rozwiązania problemu CSP, gdyż dziedzinę każdej zmiennej można stopniowo dzielić na coraz mniejsze podzbiory, aż w końcu zostaną wyznaczone wszystkie wartościowania zmiennych

# Przestrzenie obliczeń w programowaniu z ograniczeniami (1)

## Pojęcia podstawowe

- *ograniczenie podstawowe* (ang. *basic constraint*) – zmienna wraz z jej aktualną dziedziną (np. `X::1#5`), inne ograniczenia (o postaci formuł) to *ograniczenia niepodstawowe*
- *skład więzów* (ang. *constraint store*) – zbiór ograniczeń podstawowych
- *propagator* – agent odpowiadający danemu ograniczeniu niepodstawowemu, zadanie tego agenta polega na obserwowaniu składu więzów i wykonywaniu w odpowiednich okolicznościach propagacji
- stany propagatora:
  - *nadmiarowy* (ang. *entailed*), gdy odpowiadające mu ograniczenie jest spełnione dla każdego wartościowania zmiennych z danego składu więzów, np. propagator odpowiadający ograniczeniu `X <: 10` jest nadmiarowy, gdy skład więzów zawiera ograniczenie `X::1#5`
  - *sprzeczny* (ang. *failed*), gdy odpowiadające mu ograniczenie nie jest spełnione dla żadnego wartościowania zmiennych z danego składu więzów, np. propagator od-powiadający ograniczeniu `X <: 10` jest sprzeczny, gdy skład więzów zawiera ograniczenie `X::50#100`
  - *stabilny* (ang. *stable*), gdy propagator jest sprzeczny lub gdy nie może ograniczyć dziedziny żadnej zmiennej, np. propagatory odpowiadające ograniczeniom (c) i (d) w przykładzie 2 stają się stabilne po wykonaniu drugiego kroku propagacji

# Przestrzenie obliczeń w programowaniu z ograniczeniami (2)

## Pojęcia podstawowe

- *przestrzeń obliczeń* (ang. *computation space*) – skład więzów wraz z przyporządkowanymi mu propagatorami
- *dystrybutor* (ang. *distributor*) – agent, którego zadanie polega na obserwowaniu przestrzeni obliczeń i wykonywaniu w odpowiednich okolicznościach dystrybucji, wg. zadanej strategii
- stany przestrzeni obliczeń:
  - *rozwiązana* (ang. *solved*), gdy każdy jej propagator jest nadmiarowy; wówczas dowolne wartościowanie zmiennych nad ich dziedzinami jest rozwiązaniem rozpatrywanego problemu
  - *sprzeczna*, gdy co najmniej jeden z jej propagatorów jest sprzeczny
  - *stabilna*, gdy jest niesprzeczna i gdy każdy jej propagator jest stabilny
- *drzewo poszukiwań* (ang. *search tree*) – drzewo, którego wierzchołkami są przestrzenie obliczeń a dowolne dwa wierzchołki są połączone krawędzią wtedy i tylko wtedy, gdy wierzchołek potomny powstaje w wyniku dystrybucji wykonanej w wierzchołku rodzicielskim.

# Przestrzenie obliczeń w programowaniu z ograniczeniami (3)

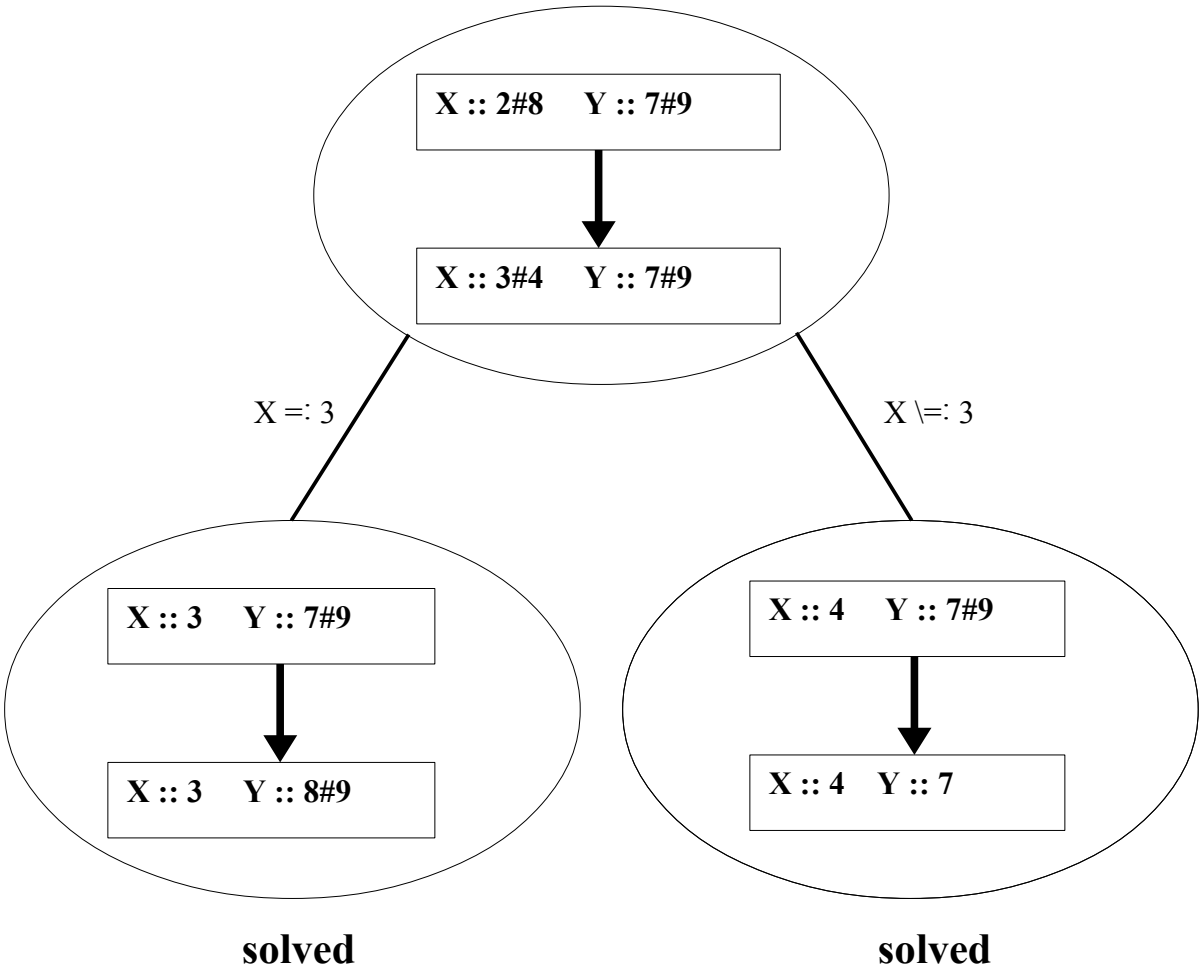
## Drzewo poszukiwań, strategie dystrybucji

- drzewo poszukiwań jest modelem obliczeń prowadzonych w toku rozwiązywania problemu CSP metodą propagacji i dystrybucji
- struktura drzewa poszukiwań jest określona przez strategię dystrybucji
- w systemach programowania CP (np. w systemie Mozart) definiuje się standardowe strategie dystrybucji, takie jak:
  - *naive* – dziedzinę dowolnie wybranej zmiennej `X` dzieli się na podzbiory za pomocą formuł `X =: l` oraz `X \=: l`, gdzie `l` jest najmniejszym elementem aktualnej dziedziny zmiennej `X`
  - *ff* (ang. *first fail*) – do dystrybucji wybiera się zmienną o najmniejszej dziedzinie w rozpatrywanym składzie więzów, zasada podziału dziedziny jest taka sama jak w przypadku strategii *naive*
  - *split* – do dystrybucji wybiera się zmienną `X` o najmniejszej dziedzinie w rozpatrywanym składzie więzów a podziału dziedziny dokonuje się za pomocą formuł `X >: m` oraz `X =<: m`, gdzie `m` jest środkowym elementem dziedziny aktualnej

# Przestrzenie obliczeń w programowaniu z ograniczeniami (4)

## Przykład 3

Drzewo poszukiwań dla problemu CSP z przykładu 1; symbol `X::n` odpowiada ograniczeniu `X::n#n`.




Ograniczenia "niepodstawowe" z przykładu 1:

- `X * Y <: 29`
- `2 * X + Y >: 13`

# Programowanie z ograniczeniami w systemie Mozart (1)

## Ogólna postać programu

```
proc {Script Sol}
    % deklaracje zmiennych
in
    % specyfikacja więzów podstawowych
    % specyfikacja więzów "niepodstawowych"
    % specyfikacja strategii dystrybucji
end
```

### Przykład 4 (wg [3, 4])

Program będący opisem problemu, polegającego na zna-lezieniu takiego przypisania cyfr literom występującym w słowach SEND + MORE = MONEY, aby uzyskać poprawne równanie.

```
proc {Money Sol}
  S E N D M O R Y
in
  Sol = sol(s:S e:E n:N d:D m:M o:O r:R y:Y)
  Sol ::: 0#9
  {FD.distinct Sol}
  S \=: 0
  M \=: 0
        1000 * S + 100 * E + 10 * N + D
  +     1000 * M + 100 * O + 10 * R + E
  =: 10000 * M + 1000 * O + 100 * N + 10 * E + Y
  {FD.distribute naive Sol}
end
```

# Programowanie z ograniczeniami w systemie Mozart (2)

## Wykonywanie programu

- wykonanie programu CP w środowisku Mozart polega na skonstruowaniu odpowiedniego drzewa poszukiwań
- strategia konstruowania drzewa poszukiwań (tj. strategia wykonania programu) nie ma wpływu na strukturę drzewa określoną przez strategię dystrybucji; oznacza to, że ten sam program CP może zostać poprawnie wykonany na wiele sposobów
- rodzaje strategii wykonywania programu:
  - s. poszukujące jednego rozwiązania
  - s. poszukujące wszystkich rozwiązań
  - s. poszukujące rozwiązania najlepszego.

## Klasa `Search`

- do wykonywania programów CP służą *silniki* lub inaczej *maszyny przeszukujące* (ang. *search engines*); należą do nich obiekty klasy `Search`, które realizują różnorakie strategie konstruowania drzewa poszukiwań
- obiekty klasy `Search` tworzy się i uaktywnia za pomocą następujących instrukcji:
  - `{Search.base.one +Script ?Xs}`
  - `{Search.base.all +Script ?Xs}`
  - `{Search.base.one +Script +Order ?Xs}`

  gdzie parametry `Script`, `Xs` oraz `Order` oznaczają odpowiednio rozpatrywany program, poszukiwane rozwiązanie oraz kryterium wyznaczania rozwiązania najlepszego.

# Programowanie z ograniczeniami w systemie Mozart (3)

## Klasa `Search`

- inny sposób wykonania programu polega na utworzeniu silnika ogólnego przeznaczenia (1) po czym wskazuje się akcję, którą obiekt ten ma wykonać (2)

  (1) `E = {New Search.object script(+Script)},`

  (2) `{E [next(?Xs) | last(?Xs) | stop]},`

  metawyrażenie [`next(?Xs) | last(?Xs) | stop`] należy zastąpić jednym z jego elementów rozdzielonych znakiem |; akcje `next`, `last` oraz `stop` oznaczają odpowiednio poszukiwanie rozwiązania następnego, poszukiwanie rozwiązania ostatniego oraz zatrzymanie silnika,

- drzewo poszukiwań można konstruować współbieżnie; w tym celu należy utworzyć odpowiedni silnik (3) po czym wskazać rodzaj strategi poszukiwania rozwiązania, którą ma on realizować (4)

  (3) `E = {New Search.parallel init(w1:1#ssh w2:2#rsh)},`

  (4) `{E [one(PFun ?Xs) |all(PFun ?Xs) |best(PFun ?Xs)]},`

  wyrażenie `init(w1:1#ssh w2:2#rsh)` specyfikuje rozproszone środowisko obliczeniowe (patrz [4]) a symbol `PFun` oznacza *funktor* (patrz [3]), w którym należy umieścić program wg ogólnego schematu:

```
functor PFun
import
  % moduły biblioteczne, zawierające
  % wykorzystywane propagatory
export +Script
define
  % deklaracja procedury +Script
end
```

# Programowanie z ograniczeniami w środowisku SWI-Prolog 7.6.X (1)

## Wstęp

- środowisko SWI-Prolog 7.6.X zawiera 2 biblioteki służące do programowania z ograniczeniami; są to: **`clpfd`** (obliczenia nad dziedziną FD) oraz **`clpqr`** (obliczenia nad dziedziną R)
- program wykonuje się wg strategii DFS, zaimplementowanej w maszynie prologowej - nie ma możliwości wyboru strategii poprzez wskazanie maszyny przeszukującej (tak jak w środowisku Mozart)
- w tym samym programie można łączyć elementy programowania w paradygmacie CP oraz LP - paradygmat "wypadkowy" nosi nazwę *Constraint Logic Programming* (CLP)
- ogólny schemat programu:

```
:- use_module(library(clpfd)).
main(Sol) :-
%  specyfikacja więzów podstawowych
%  specyfikacja więzów "niepodstawowych"
%  specyfikacja strategii dystrybucji
```

# Programowanie z ograniczeniami w środowisku SWI-Prolog 7.6.X (2)

## Przykład 5

Poniższy program reprezentuje ograniczenia z przykładu 1; dystrybucję realizuje się wg strategii *leftmost* (odpowiednik strategii *naive*).

```
:- use_module(library(clpfd)).

przyk5(Sol) :-
   X in 2..8,
   Y in 7..9,
   Sol = [X,Y],
   X * Y #< 29,
   2 * X + Y #> 13,
   labeling([leftmost],Sol).
```

## Przykład 6

Odpowiednik programu z przykładu 4

```
:- use_module(library(clpfd)).
money(Sol) :-
   Vars = [S,E,N,D,M,O,R,Y],
   Vars ins 0..9,
   Sol=sol(s:S,e:E,n:N,d:D,m:M,o:O,r:R,y:Y),
   all_different(Vars),
   M #\= 0, S #\= 0,
   S*1000 + E*100 + N*10 + D +
   M*1000 + O*100 + R*10 + E #=
   M*10000 + O*1000 + N*100 + E*10 + Y,
   labeling([leftmost],Vars).
```