

Mini-Curso de Minizinc, (baseado em exemplos)

Claudio Cesar de Sá

Departamento de Ciência da Computação
Centro de Ciências e Tecnologias
Universidade do Estado de Santa Catarina

24 de agosto de 2016

Sumário (1)

Contextualização

- Problemas e Otimização
- Otimização
- Programação por Restrições
- Histórico
- Propósitos
- Motivação
- Paradigma Declarativo
- Características do MiniZinc
- Instalação e uso
- Estrutura de um Modelo
- Exemplo Inicial

Elementos da Linguagem

- Parâmetros e Variáveis
- Alguns Operadores Lógicos

Sumário (2)

Outras Características

Exemplos Introdutórios

Um Clássico da PO

Teoria dos Conjuntos

Construindo Funções e Predicados

Função Binária

Uso na Lógica Proposicional

Uso na Lógica de Primeira-Ordem

Vetores 1D

Problema da soma de subconjuntos

Régua de Golomb

Função: $y = \text{soma}(\text{vetor } 1D)$

Vetores 2D

Quadrado Mágico

Problema de Atribuição

Sumário (3)

Os Nadadores Americanos
Job-Shop-Schedulling
Coloração de Mapas

Melhorando as Buscas

Parâmetro search
Restrições Globais

Tópicos Gerais

String e Fix
Ainda Conjuntos

Conclusões

Referências Bibliográficas

Notas

- Todos os códigos apresentados se encontram em:
`https://github.com/claudiosa/CCS/tree/master/minizinc`, com o prefixo `sbpo_`
- Sempre atualizado este tutorial em `https://github.com/claudiosa/CCS/tree/master/minizinc/curso_minizinc`
- Metodologia: ensino da linguagem via exemplos apresentando os principais recursos e técnicas
- Agradecimentos a todos da SBPO-2016 pela oportunidade em organizar este material e estudantes da UDESC por testarem parte dele ao longo dos anos.

Problemas × Otimização



Trocar esta figura futuramente

Problemas e Otimização

Complexidade \Leftrightarrow Encontrar soluções:

- Problemas complexos de interesse prático (e teórico): NPs \uparrow
- Tentativas de soluções: diversas direções (teoria) e muitos paradigmas computacionais (práticas)
- Seguem desde um modelo matemático existente a um modelo empírico a ser descoberto. Exemplificando:

Problemas e Otimização

Complexidade \Leftrightarrow Encontrar soluções:

- Problemas complexos de interesse prático (e teórico): NPs \uparrow
- Tentativas de soluções: diversas direções (teoria) e muitos paradigmas computacionais (práticas)
- Seguem desde um modelo matemático existente a um modelo empírico a ser descoberto. Exemplificando:
 - Uma equação de regressão linear: $y = ax^2 + b$
 - ... até ...
 - Programação genética (evolução de um modelo)
- Problemas apresentam características comuns como: variáveis, domínios, restrições, espaços de estados (finitos e infinitos, contínuos e discretos) ...

Otimização

Complexidade \Leftrightarrow Otimização:

- A área de **Otimização** tem uma divisão: Discreta ou Combinatória e Contínua ou Numérica (funções deriváveis)

Otimização

Complexidade \Leftrightarrow Otimização:

- A área de **Otimização** tem uma divisão: Discreta ou Combinatória e Contínua ou Numérica (funções deriváveis)
 - Combinatória:** Problemas definidos em um espaço de estados finitos (ou infinito mas enumerável)
 - Numérica:** Definidos em subespaços infinitos e não enumeráveis, como os números reais e complexos
- Difícil: problemas que tenham uma ordem maior ou igual a $2^{O(n)}$ são **exponenciais**, consequentemente, **difíceis**!

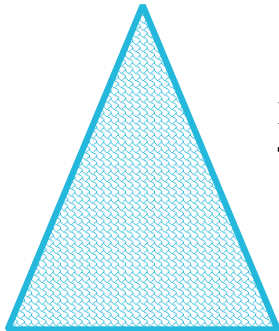
Como atacar estes problemas?

Técnicas:

- Combinatória:**
- Busca Local
 - Métodos Gulosos: busca tipo subida a encosta (*hill-climbing*), recozimento simulado (*simulated annealing*), busca tabu, etc.
 - Programação Dinâmica
 - **Programação por Restrições (PR)**
 - Redes de Fluxo
 -
- Numérica:**
- Descida do Gradiente
 - Gauss-Newton
 - Lavemberg-Marquardt
 -

Programação por Restrições (PR)

Problema = muitos estados ...



PR
→

Reduzindo os estados ...

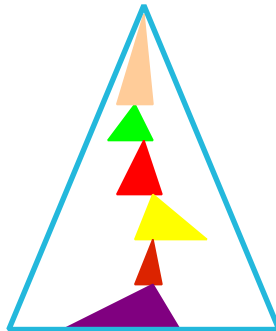


Figura: O *mar de estados* e a filtragem da PR

Onde o objetivo é:

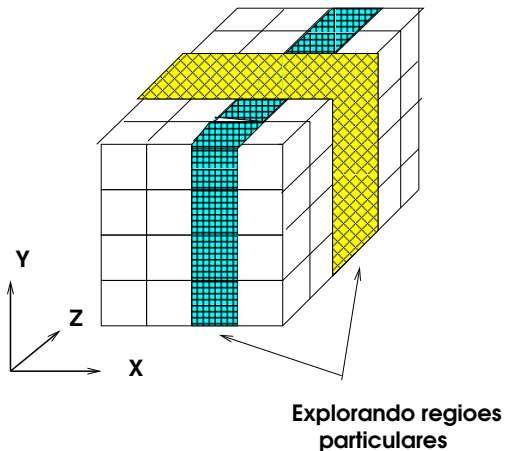


Figura: Operando com regiões específicas ou reduzidas

Redução em sub-problemas:

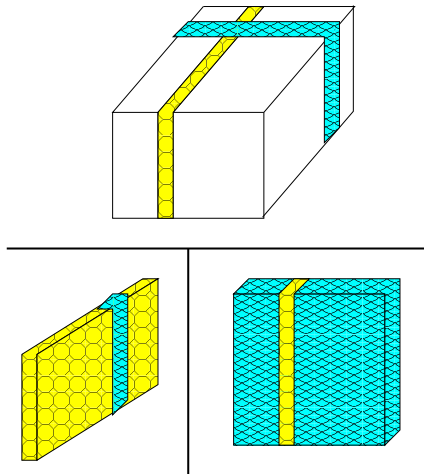
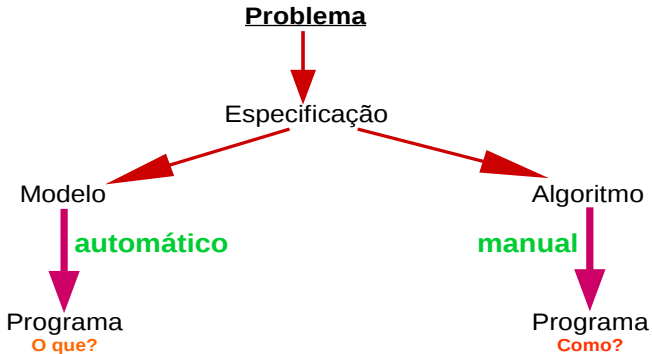


Figura: Redução de P em outros sub-problemas equivalentes

Construção de modelos e implementações:

Modelagem

Programação



Ferramentas: linguagens, tradutores e *solvers*:

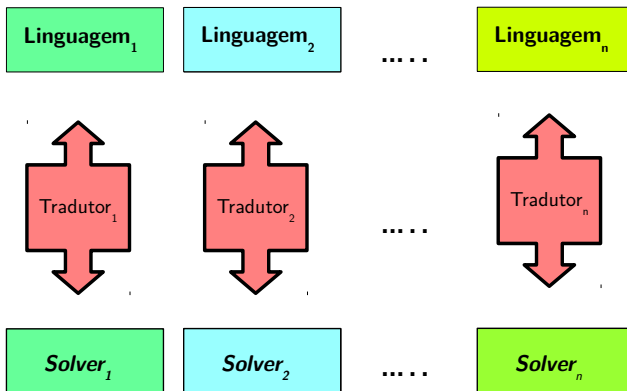


Figura: Linguagens, bibliotecas e *solvers* de propósitos diversos

Minizinc, tradutores e os *solvers*:

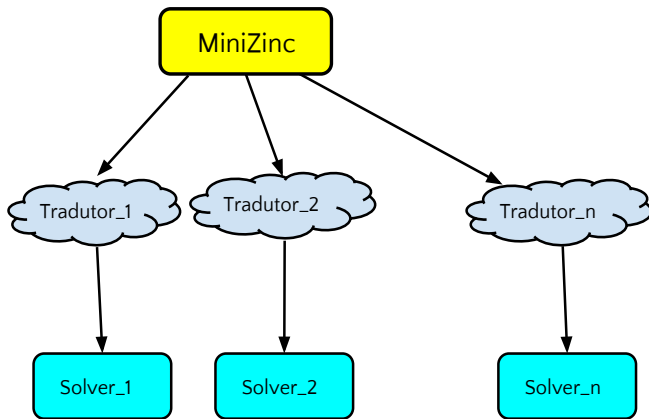


Figura: Há muitos conversores do MiniZinc para vários *solvers* \Rightarrow uma proposta unificada

Histórico

➔ Em 2006 a comunidade de CP *Constraint Programming* discutiu a necessidade de uma linguagem unificada para seus modelos e pesquisas

Histórico

- ➔ Em 2006 a comunidade de CP *Constraint Programming* discutiu a necessidade de uma linguagem unificada para seus modelos e pesquisas
- ➔ Inicialmente a linguagem ZINC foi criada pelo NICTA, Universidade de Melbourne e Universidade de Monash. Tudo na Austrália!

Histórico

- Em 2006 a comunidade de CP *Constraint Programming* discutiu a necessidade de uma linguagem unificada para seus modelos e pesquisas
- Inicialmente a linguagem ZINC foi criada pelo NICTA, Universidade de Melbourne e Universidade de Monash. Tudo na Austrália!
- O MINZINC é um sub-conjunto do ZINC

Histórico

- Em 2006 a comunidade de CP *Constraint Programming* discutiu a necessidade de uma linguagem unificada para seus modelos e pesquisas
- Inicialmente a linguagem ZINC foi criada pelo NICTA, Universidade de Melbourne e Universidade de Monash. Tudo na Austrália!
- O MINZINC é um sub-conjunto do ZINC
- Linguagem de modelagem \Rightarrow paradigma lógico de programação

Histórico

- Em 2006 a comunidade de CP *Constraint Programming* discutiu a necessidade de uma linguagem unificada para seus modelos e pesquisas
- Inicialmente a linguagem ZINC foi criada pelo NICTA, Universidade de Melbourne e Universidade de Monash. Tudo na Austrália!
- O MINZINC é um sub-conjunto do ZINC
- Linguagem de modelagem \Rightarrow paradigma lógico de programação
- Minizinc é compilado para o FlatZinc – cujo código é traduzido há vários outros *solvers*

Propósitos

- ➔ Objetivo: resolver problemas de otimização combinatória e PSR (Problemas de Satisfação de Restrições)
- ➔ O objetivo é descrever o problema: **declarar** no lugar de especificar o que o programa deve fazer
- ➔ Paradigma de programação imperativo: **como** deve ser calculado !
- ➔ Paradigma de programação declarativo: **o que** deve ser calculado!

Motivação

O que é um problema combinatório?

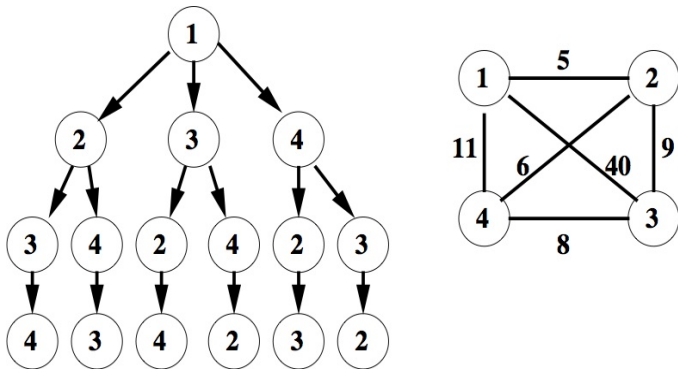


Figura: Problema da sequência de visitas

Complexidade \times Combinatória

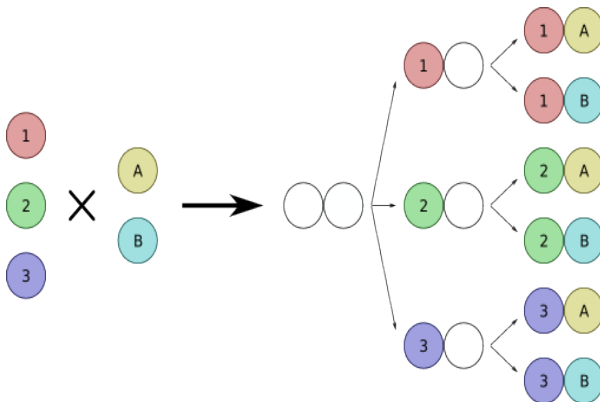


Figura: Contando combinações das variáveis: X e Y

Um paradigma computacional:

$$Modelo + Dados = A_1 \wedge A_2 \wedge A_3 \wedge \dots \wedge A_n$$

➡ A_i : são assertivas declaradas (declarações de restrições) sobre o problema

Um paradigma computacional:

$$Modelo + Dados = A_1 \wedge A_2 \wedge A_3 \wedge \dots \wedge A_n$$

- ➡ A_i : são assertivas declaradas (declarações de restrições) sobre o problema
- ➡ Linguagem \Rightarrow construir modelos \Rightarrow problemas reais

Um paradigma computacional:

$$Modelo + Dados = A_1 \wedge A_2 \wedge A_3 \wedge \dots \wedge A_n$$

- A_i : são assertivas declaradas (declarações de restrições) sobre o problema
- Linguagem \Rightarrow construir modelos \Rightarrow problemas reais
- **Modelos** \Leftrightarrow computáveis!

Um paradigma computacional:

$$Modelo + Dados = A_1 \wedge A_2 \wedge A_3 \wedge \dots \wedge A_n$$

- A_i : são assertivas declaradas (declarações de restrições) sobre o problema
- Linguagem \Rightarrow construir modelos \Rightarrow problemas reais
- **Modelos** \Leftrightarrow computáveis!
- Visão lógica: insatisfatível (sem respostas) ou consistente

Resumindo alguns livros e *solvers*

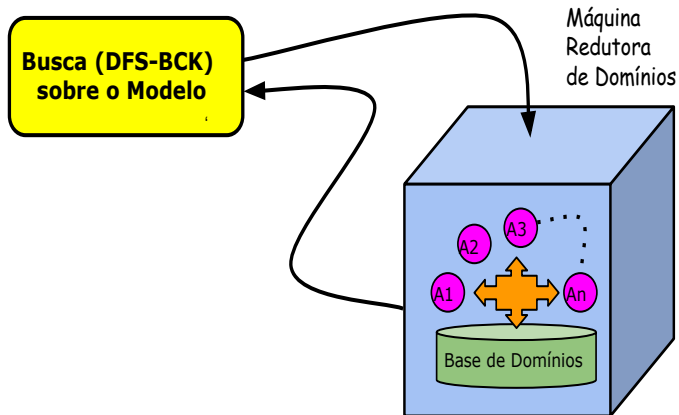


Figura: Ciclo entre a efetiva busca e a poda, na propagação das restrições

Características do MiniZinc

- Modelagem: imediata à abordagem matemática existente

Características do MiniZinc

- Modelagem: imediata à abordagem matemática existente
- Para isto, *MUITOS* recursos: operadores booleanos, aritméticos, constantes, variáveis, etc

Características do MiniZinc

- Modelagem: imediata à abordagem matemática existente
- Para isto, *MUITOS* recursos: operadores booleanos, aritméticos, constantes, variáveis, etc
- **Fortemente tipada**

Características do MiniZinc

- Modelagem: imediata à abordagem matemática existente
- Para isto, *MUITOS* recursos: operadores booleanos, aritméticos, constantes, variáveis, etc
- **Fortemente tipada**
- *Dois* tipos de dados: constantes e variáveis

Continuando as características:

- Constantes: são valores fixos – são conhecidos como **parâmetros**

Continuando as características:

- Constantes: são valores fixos – são conhecidos como **parâmetros**
- Variáveis: assumem valores sobre um domínio (aqui é o ponto)

Continuando as características:

- Constantes: são valores fixos – são conhecidos como **parâmetros**
- Variáveis: assumem valores sobre um domínio (aqui é o ponto)
- Logo: restringir estes domínios apenas para valores admissíveis

Continuando as características:

- Constantes: são valores fixos – são conhecidos como **parâmetros**
- Variáveis: assumem valores sobre um domínio (aqui é o ponto)
- Logo: restringir estes domínios apenas para valores admissíveis
- Mas há muitos tipos de dados: *int*, *bool*, *real*, *arrays*, *sets*, etc

Continuando as características:

- Constantes: são valores fixos – são conhecidos como **parâmetros**
- Variáveis: assumem valores sobre um domínio (aqui é o ponto)
- Logo: restringir estes domínios apenas para valores admissíveis
- Mas há muitos tipos de dados: *int*, *bool*, *real*, *arrays*, *sets*, etc
- Diferentemente da tipagem dinâmica aqui não existe!

Instalação e uso

Tem evoluído muito nestes últimos anos:

1. **Tudo tem sido simplificado**
2. **Download e detalhes: <http://www.minizinc.org/>**
3. Basicamente: baixar o arquivo da arquitetura desejada, instalar, acertar variáveis de ambiente, *path*, e usar como:

Instalação e uso

Tem evoluído muito nestes últimos anos:

1. **Tudo tem sido simplificado**
2. **Download e detalhes: <http://www.minizinc.org/>**
3. Basicamente: baixar o arquivo da arquitetura desejada, instalar, acertar variáveis de ambiente, *path*, e usar como:
 - Modo console (ou linha de comando) ou
 - Interface IDE

Resumindo

➔ Modo console: mzn2doc, mzn2fzn, mzn-g12fd, mzn-g12lazy, mzn-g12mip, mzn-gecode, ...

1. Edite o programa em um editor ASCII
2. Para compilar e executar:
mzn-xxxx nome-do-programa.mzn ou escolher um outro *solver*
3. Exemplo como todas soluções:
mzn-g12fd -all _solutions nome-do-programa.mzn
4. Detalhes e opções: **mzn-g12fd -help**

Resumindo

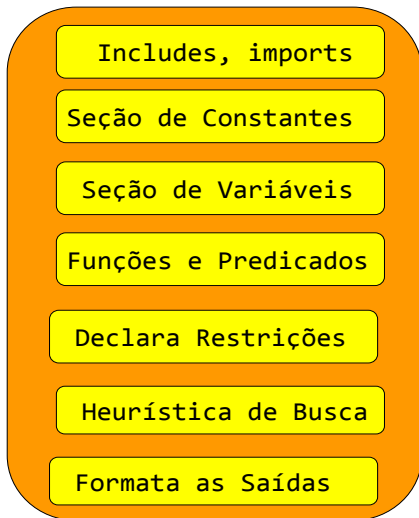
➔ Modo console: `mzn2doc`, `mzn2fzn`, `mzn-g12fd`, `mzn-g12lazy`, `mzn-g12mip`, `mzn-gecode`, ...

1. Edite o programa em um editor ASCII
2. Para compilar e executar:
`mzn-xxxx nome-do-programa.mzn` ou escolher um outro *solver*
3. Exemplo como todas soluções:
`mzn-g12fd -all _solutions nome-do-programa.mzn`
4. Detalhes e opções: `mzn-g12fd -help`

➔ Modo IDE: `minizinc_IDE` ou `minizincIDE`

➔ Na IDE dá para editar e alterar configurações

Estrutura de um Modelo



Exemplo \times Espaço de Estado (EE)

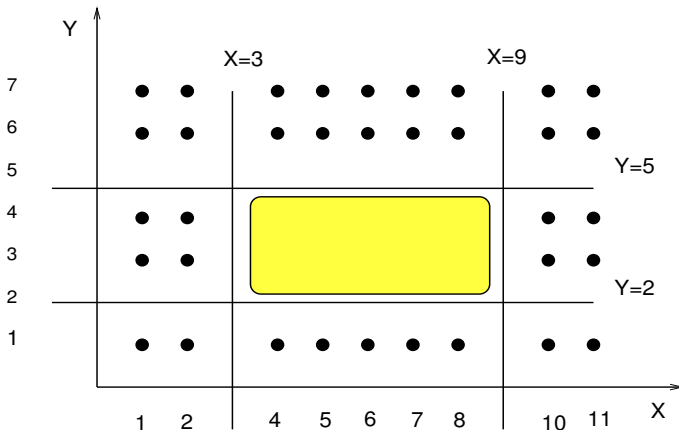


Figura: Obter os pontos do interior do retângulo

Exemplo (1)

```

1 %% Declara constantes
2 int: UM = 1; int: DOIS = 2; int: CINCO = 5;
3 %% Declara variaveis
4 var UM .. 11 : X; %% segue o dominio 1..11
5 var UM .. 7 : Y;
6
7 %% As restricoes
8 constraint
9     Y > DOIS /\ Y < CINCO ;
10
11 constraint
12     X > 3 /\ X < 9 ;
13
14 %%% A busca : MUITAS OPCOES ....
15 solve::int_search([X,Y],input_order,indomain_min,complete) satisfy;
16 %% SAIDAS
17 output [" X: ", show(X), " Y: ", show(Y), "\n"];

```

Saída

```
$ mzn-g12fd -a sbpo_xerek-ygor.mzn
```

```
  X: 4      Y: 3
```

```
-----
  X: 4      Y: 4
```

```
-----
  X: 5      Y: 3
```

```
-----
  X: 5      Y: 4
```

```
-----
  X: 7      Y: 4
```

```
-----
  X: 8      Y: 3
```

```
-----
  X: 8      Y: 4
```

```
=====
```

A ordem dos valores de saída!!!!

Parâmetros e Variáveis

Existem basicamente dois tipos de variáveis em Minizinc:

Parâmetros e Variáveis

Existem basicamente dois tipos de variáveis em Minizinc:

Parâmetros: *quase igual* às variáveis de linguagens de programação comuns. Entretanto, só é permitido atribuir um valor a um parâmetro uma única vez.

Parâmetros e Variáveis

Existem basicamente dois tipos de variáveis em Minizinc:

Parâmetros: *quase igual* às variáveis de linguagens de programação comuns. Entretanto, só é permitido atribuir um valor a um parâmetro uma única vez.

Variáveis de Decisão: mais próximo ao conceito de incógnitas da matemática. O valor de uma variável de decisão é escolhido pelo Minizinc para atender todas as restrições estabelecidas.

Parâmetros e Variáveis

Existem basicamente dois tipos de variáveis em Minizinc:

Parâmetros: *quase igual* às variáveis de linguagens de programação comuns. Entretanto, só é permitido atribuir um valor a um parâmetro uma única vez.

Variáveis de Decisão: mais próximo ao conceito de incógnitas da matemática. O valor de uma variável de decisão é escolhido pelo Minizinc para atender todas as restrições estabelecidas.

Variáveis de Restrição: similar a anterior, exceto que o domínio é específico as respostas desejadas do problema

Parâmetros e Variáveis

Existem basicamente dois tipos de variáveis em Minizinc:

Parâmetros: *quase igual* às variáveis de linguagens de programação comuns. Entretanto, só é permitido atribuir um valor a um parâmetro uma única vez.

Variáveis de Decisão: mais próximo ao conceito de incógnitas da matemática. O valor de uma variável de decisão é escolhido pelo Minizinc para atender todas as restrições estabelecidas.

Variáveis de Restrição: similar a anterior, exceto que o domínio é específico as respostas desejadas do problema

Variáveis de Restrição: estas são *descobertas* dentro de um domínio de valores sob um conjunto de restrições que é o **modelo a ser computado!**

Exemplos de Variáveis

Exemplo de Parâmetro (*variável fixa*) em MINIZINC

```
1 int: parametro = 5;
```

Exemplo de Variável em MINIZINC

```
1 var 1..15: variavel;
```

Constraints (Restrições)

Restrições podem ser equações ou desigualdades sobre as variáveis de decisão, de forma a restringir os possíveis valores que estas podem receber.

Constraints (Restrições)

Restrições podem ser equações ou desigualdades sobre as variáveis de decisão, de forma a restringir os possíveis valores que estas podem receber.

Exemplos de Restrições

```
1 constraint x > 2;  
2  
3 constraint 3*y - x <= 17;  
4  
5 constraint x != y;  
6  
7 constraint x = 2*z;
```

Alguns Operadores Lógicos

Operadores Lógicos

- Os operadores lógicos (*and*, *or*, *not*), que existem na maioria das linguagens de programação, também podem ser utilizados em MINIZINC nas restrições.
- Outros importantes conectivos: \rightarrow (implicação ou *if-then*) e \leftrightarrow (equivalência ou bi-implicação)

Alguns Operadores Lógicos

Operadores Lógicos

- Os operadores lógicos (*and*, *or*, *not*), que existem na maioria das linguagens de programação, também podem ser utilizados em MINIZINC nas restrições.
- Outros importantes conectivos: \rightarrow (implicação ou *if-then*) e \leftrightarrow (equivalência ou bi-implicação)

Exemplo de Utilização (and)

```

1 var bool : p;
2 var bool : q;
3 constraint
4     (p /\ q) == true;
5
6 solve satisfy;
7
8 output [show(p), "      ", show(q)];

```

Exemplo de Utilização (*or*)

```
1 constraint (p \/ q) = false;
```

Exemplo de Utilização (*not*)

```
1 constraint (not)p = true;
```

Outras Características

Diversos

- Muitas funções, predicados, e restrições prontos
- Boa aderência a *list comprehensions*
- Suporte integral a vetores

Quermesse da Nossa Escola

Exemplo

A escola local fará uma festa e esta precisa que façamos bolos para vender. Sabemos como fazer dois tipos de bolos. Eis a receita de cada um deles:

Quermesse da Nossa Escola

Exemplo

A escola local fará uma festa e esta precisa que façamos bolos para vender. Sabemos como fazer dois tipos de bolos. Eis a receita de cada um deles:

Bolo de Banana	Bolo de Chocolate
- 250g de farinha	- 200g de farinha
- 2 bananas	- 75g de cacau
- 75g de açúcar	- 150g de açúcar
- 100g de manteiga	- 150g de manteiga

Tabela: Insumos de cada bolo

Continuando o enunciado ...

O preço de venda de um Bolo de Chocolate é de R\$4,50 e de um Bolo de Banana é de R\$4,00. Temos 4kg de farinha, 6 bananas, 2kg de açúcar, 500g de manteiga e 500g de cacau. Qual a quantidade de cada bolo que deve ser feita para maximizar o lucro das vendas para a escola?

Comentários Gerais da Solução

1. Basicamente temos que encontrar: N_1 (número de bolos de Chocolate) e N_2 (número de bolos de Banana);

Comentários Gerais da Solução

1. Basicamente temos que encontrar: N_1 (número de bolos de Chocolate) e N_2 (número de bolos de Banana);
2. Tendo os valores N_1 e N_2 , sabemos o nosso lucro máximo, dado o valor por bolo vendido;

Comentários Gerais da Solução

1. Basicamente temos que encontrar: N_1 (número de bolos de Chocolate) e N_2 (número de bolos de Banana);
2. Tendo os valores N_1 e N_2 , sabemos o nosso lucro máximo, dado o valor por bolo vendido;
3. Assim a equação a ser maximizada é:
$$4500.N_1 + 4000.N_2 = \textit{Lucro}$$

Comentários Gerais da Solução

1. Basicamente temos que encontrar: N_1 (número de bolos de Chocolate) e N_2 (número de bolos de Banana);
2. Tendo os valores N_1 e N_2 , sabemos o nosso lucro máximo, dado o valor por bolo vendido;
3. Assim a equação a ser maximizada é:

$$4500.N_1 + 4000.N_2 = \textit{Lucro}$$
4. Sabe-se que **UM** bolo necessita de quantidades de insumos dado na tabela 1

Comentários Gerais da Solução

1. Basicamente temos que encontrar: N_1 (número de bolos de Chocolate) e N_2 (número de bolos de Banana);
2. Tendo os valores N_1 e N_2 , sabemos o nosso lucro máximo, dado o valor por bolo vendido;
3. Assim a equação a ser maximizada é:

$$4500.N_1 + 4000.N_2 = \textit{Lucro}$$
4. Sabe-se que **UM** bolo necessita de quantidades de insumos dado na tabela 1
5. Logo, se são N bolos por insumos e respeitando a disponibilidade de cada um, as restrições para ambos os bolos são do tipo:

$$N_1 \cdot qt_{manteiga_{chocolate}} + N_2 \cdot qt_{manteiga_{banana}} \leq Manteiga_{disponivel}$$

Comentários Gerais da Solução

1. Basicamente temos que encontrar: N_1 (número de bolos de Chocolate) e N_2 (número de bolos de Banana);
2. Tendo os valores N_1 e N_2 , sabemos o nosso lucro máximo, dado o valor por bolo vendido;
3. Assim a equação a ser maximizada é:

$$4500.N_1 + 4000.N_2 = \textit{Lucro}$$
4. Sabe-se que **UM** bolo necessita de quantidades de insumos dado na tabela 1
5. Logo, se são N bolos por insumos e respeitando a disponibilidade de cada um, as restrições para ambos os bolos são do tipo:

$$N_1 \cdot qt_{manteiga_{chocolate}} + N_2 \cdot qt_{manteiga_{banana}} \leq Manteiga_{disponivel}$$
6. E estes valores são tomados da tabela 1.

Uma tabela *conhecida* tipo:

	farinha	cacau	bananas	açucar	manteiga	
N_1 (Choco)	200	75	–	150	150	
N_2 (Banana)	250	–	2	75	100	
Disponível	4000	500	6	2000	500	

Código desta solução:

```

1 var 0..100: bc; %Bolo de chocolate: N1
2 var 0..100: bb; %Bolo de banana: N2
3
4 constraint
5     250*bb + 200*bc <= 4000;
6
7 constraint
8     2*bb <= 6;
9
10 constraint
11     75*bb + 150*bc <= 2000;
12
13 constraint
14     100*bb + 150*bc <= 500;
15
16 constraint
17     75*bc <= 500;
18
19 solve maximize (4500*bc + 4000*bb);
20
21 output[" Choc = ", show(bc), "\t Ban = ", show(bb)];

```

Saída

```
$ mzn-g12fd -a sbpo_bolos.mzn
  Choc = 0  Ban = 0
-----
  Choc = 1  Ban = 0
-----
  Choc = 2  Ban = 0
-----
  Choc = 3  Ban = 0
-----
  Choc = 2  Ban = 2
-----
=====
Finished in 36msec
```

Teoria dos Conjuntos (1)

Há várias funções prontas:

$A \cup B$, $A \cap B$, \overline{A} , etc

```

1 set of int: A = {4,5,7,13};
2 set of int: B = 1 .. 3 union {7} union {13}; %% DOM descontinuos
3 %set of int: B = {1,2,3,7,13};
4
5 var set of 1 .. 100 : Var_Uniao;
6 var set of 1 .. 100 : Var_Inters ;
7 var bool : Var_Bool;
8 var int : X_Var;
9
10 constraint Var_Uniao = B union A;
11
12 constraint Var_Inters = B intersect A;
13
14 constraint Var_Bool <-> (7 in A) /\ (7 in B); %% = ou <->
15
16 constraint true <-> (X_Var in A) /\ (X_Var in B);
17

```

Teoria dos Conjuntos (2)

```

18 solve satisfy;
19
20 %% Na saida tudo eh convertido para String: show_int, show_float ...
21 output
22 ["VAR_Uniao = " , show(Var_Uniao), "\n",
23  "VAR_Inters = " , show(Var_Inters), "\n",
24  "Var_Bool = " , show(Var_Bool ), "\n",
25  "X_Var = " , show_int(fix(X_Var), X_Var)];
26
27 %% fix: mais adiante ... mas aqui "fixa" a variavel para imprimir

```

Saída:

```
$ mzn-g12fd -a sbpo_teorias_conjuntos.mzn
VAR_Uniao = {1,2,3,4,5,7,13}
VAR_Inters = {7,13}
Var_Bol = true
X_Var =      7
-----
VAR_Uniao = {1,2,3,4,5,7,13}
VAR_Inters = {7,13}
Var_Bol = true
X_Var =      13
-----
=====
```

Funções e Predicados (1)

Exemplo: $y = x^3$

```

1  int: n = 3;
2  var int: z1;
3  var int: z2;
4
5  function var int: pot_3_F(var int: n) = n*n*n ;
6
7  predicate pot_3_P(int: n, var int: res) =
8      res = n*n*n ;
9
10 constraint
11     z1 = pot_3_F(n) ;
12
13 constraint
14     n 'pot_3_P' z2 ; %% ou como na LP0 pot_3_P(n,z2) ;
15
16 solve satisfy;
17
18 output ["n: ", show(n), "\n", "z1: ", show(z1), "\n",
19         "z2: ", show(z2), "\n"];

```

Saída:

```
$ mzn-g12fd -a minizinc/sbpo_funcao_01.mzn  
n: 3  
z1: 27  
z2: 27  
-----  
=====
```

Teste de paridade (1)

Exemplo: $f_{\text{paridade}}(5) = \text{false}$, $f_{\text{paridade}}(6) = \text{true}$

```
1 int : X = 5 ; %% constantes
2 int : Y = 6;
3 var bool : var_bool_01;
4 var bool : var_bool_02;
5
6 %% Temos if-then-else-endif e NENHUMA VARIÁVEL LOCAL
7 function var bool : testa_paridade(int : N) =
8     if( (N mod 2) == 0)
9         then
10             true
11         else
12             false
13         endif;
14
15 constraint
16     var_bool_01 == testa_paridade(X);
17
18 constraint
19     var_bool_02 == testa_paridade(Y);
20
21 /* OR var_bool_01 == ((x mod 2) == 0);
```


Teste de paridade (2)

Exemplo: $f_{\text{paridade}}(5) = \text{false}$, $f_{\text{paridade}}(6) = \text{true}$

```

22     var_bool_02 == ((y mod 2) == 0); */
23
24 solve satisfy;
25
26 output
27 ["   CTE_X = ", show(X), "   CTE_Y = ", show(Y), "\n",
28  "   VAR_B01 = ", show( var_bool_01 ), "\n",
29  "   VAR_B02 = ", show( var_bool_02 ) ] ;

```

Saída:

```
$ mzn-g12fd -a minizinc/sbpo_funcao_02.mzn
  CTE_X = 5   CTE_Y = 6
  VAR_B01 = false
  VAR_B02 = true
-----
=====
```

Uso na Lógica Proposicional (1)

Exemplos:

- Modus-Ponens: $x \wedge x \rightarrow y \vdash y$
- Modus-Tollens: $\sim y \wedge x \rightarrow y \vdash \sim x$

```

1 var bool : x;
2 var bool : y;
3 var bool : Phi01;
4 var bool : Phi02;
5
6 constraint                                %% MODUS PONENS
7     ((x /\
8      (x -> y)) -> y)
9     <-> Phi01 ;
10
11 constraint                                %% MODUS TOLLENS
12     ((not y /\
13      (x -> y)) -> not x)
14     <-> Phi02 ;

```

Uso na Lógica Proposicional (2)

```

15
16 solve satisfy;
17
18 output
19 [" X: "++show(x)++"      Y: "++ show(y) ++
20  "  MP:Phi01: "++ show(Phi01)++ "\n" ]++
21 [" X: "++show(x)++"      Y: "++ show(y) ++
22  "  MT:Phi02: "++ show(Phi02)++ "\n" ];

```

Saída:

```
$ mzn-g12fd -a minizinc/sbpo_interp_MP_MT.mzn
X: false   Y: false   MP:Phi01: true
X: false   Y: false   MT:Phi02: true
-----
X: true    Y: false   MP:Phi01: true
X: true    Y: false   MT:Phi02: true
-----
X: false   Y: true    MP:Phi01: true
X: false   Y: true    MT:Phi02: true
-----
X: true    Y: true    MP:Phi01: true
X: true    Y: true    MT:Phi02: true
-----
=====
```

Interpretação na Lógica de Primeira-Ordem

Sejam as FPO abaixo:

- Exemplo 01: $\forall x \exists y (y < x)$
- Exemplo 02: $\exists x \forall y (x < y)$
- Exemplo 03: $\forall x \exists y (x^2 == y)$
- Exemplo 04: $\exists x \forall y (x^2! = y)$
- Avalie a validade para os domínios: $D_x = \{2, 3, 4\}$ e $D_y = \{3, 4, 5\}$

Interpretação na Lógica de Primeira-Ordem (1)

```

1 %%Declarando dominio das variaveis
2 set of int: A = {2, 3, 4};    %% X eh um conjunto do tipo int
3 set of int: B = {3, 4, 5};    %% inicializado com {2, 3, 4}
4
5 function bool: exemplo_01(set of int: x, set of int: y) =
6     forall (i in x)
7         (exists (j in y)
8             (j < i));
9
10 function bool: exemplo_02(set of int: x, set of int: y) =
11     exists (i in x)
12         (forall (j in y)
13             (i < j));
14
15 function bool: exemplo_03(set of int: x, set of int: y) =
16     forall (i in x)
17         (exists (j in y)
18             pow(i,2) == j));
19
20 function bool: exemplo_04(set of int: x, set of int: y) =
21     exists (i in x)
22         (forall (j in y)

```

Interpretação na Lógica de Primeira-Ordem (2)

```

23         (pow(i,2) != j));
24
25 solve satisfy;
26
27 output["\n Exemplo 01: "++ show(exemplo_01(A,B))++
28       "\n Exemplo 02: "++ show(exemplo_02(A,B))++
29       "\n Exemplo 03: "++ show(exemplo_03(A,B))++
30       "\n Exemplo 04: "++ show(exemplo_04(A,B))];

```


Saída:

```
$ mzn-g12fd -a minizinc/sbpo_interp_F0L.mzn
```

```
Exemplo 01: false
```

```
Exemplo 02: true
```

```
Exemplo 03: false
```

```
Exemplo 04: true
```

```
-----
```

```
=====
```

Vetores (ou *Arrays*) Unidimensional ou 1D

Vetores 1D

- Seja `int : n = 7;`
- `array[1..n] of int : vetor01; (constante)`
- `array[1..n] of {0,1,2,3} : vetor02; (constante)`
- `array[1..n] of var { 0,1 } : vetor03; (variável)`

Soma dos Subconjuntos

Vetor 1D: (*Subset Sum Problem*)

Seja o conjunto de números $\{2, 3, 5, 7\}$. Encontre um subconjunto que satisfaça uma soma para um dado valor. Exemplo: $k = 9$

Subconjunto	Soma
$\{\}$	0
$\{2\}$	2
$\{2, 3\}$	5
$\{2, 3, 5\}$	10
.....
$\{2, 3, 5, 7\}$	17

Complexidade: $2^n = 16$, onde n é o número de elementos do conjunto

Soma dos subconjuntos (*Subset Sum Problem*) (1)

```
1 int: n = 7; % total de elementos do vetor
2 int: K = 15; %% Soma do sub-set
3
4 array[1..n] of var 0..1 : X_DEC;
5
6 array[1..n] of int : V_Valores;
7 V_Valores = [3, 4, 5, 7, 9, 10, 1];
8
9 var int: total_VALOR;
10
11 constraint
12     total_VALOR = sum([X_DEC[i] * V_Valores[i] | i in 1..n ]) ;
13     % OU          sum( i in 1..n ) (X_DEC[i]* V_Valores[i]);
14
15 constraint
16     total_VALOR == K;
17
18 % minimize ou maximize ALGO
19 solve satisfy;
20
21 output ["Total_VALOR: " ++ show(total_VALOR) ++ "\n" ++
22        "Seleciona:" ++ show( X_DEC ) ++ "\n\t " ,
```

Soma dos subconjuntos (*Subset Sum Problem*) (2)

23 `show(V_Valores)];`

```
$ mzn-g12fd -a minizinc/sbpo_sub_set_sum.mzn
```

```
Total_VALOR: 15
```

```
Seleciona:[1, 0, 1, 1, 0, 0, 0]
           [3, 4, 5, 7, 9, 10, 1]
```

```
-----
```

```
Total_VALOR: 15
```

```
Seleciona:[0, 0, 1, 0, 0, 1, 0]
           [3, 4, 5, 7, 9, 10, 1]
```

```
-----
```

```
Total_VALOR: 15
```

```
Seleciona:[1, 1, 0, 1, 0, 0, 1]
           [3, 4, 5, 7, 9, 10, 1]
```

```
-----
```

```
Total_VALOR: 15
```

```
Seleciona:[0, 0, 1, 0, 1, 0, 1]
           [3, 4, 5, 7, 9, 10, 1]
```

```
-----
```

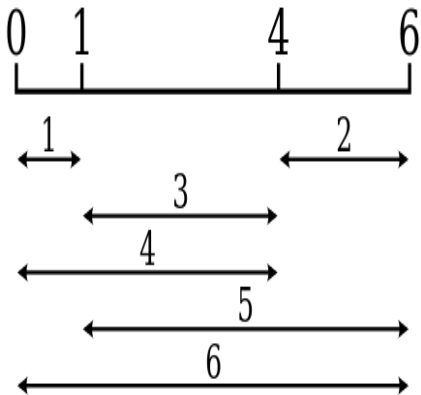
```
Total_VALOR: 15
```

```
Seleciona:[0, 1, 0, 0, 0, 1, 1]
           [3, 4, 5, 7, 9, 10, 1]
```

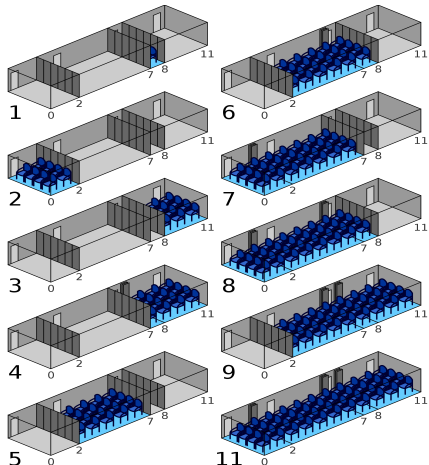
```
-----
```

```
=====
```

Régua de Golomb



(a) 4 marcas (ordem-4) e a maior distância entre duas marcas (comprimento: 6)



(b) Exemplo de aplicação

Vetor 1D: Régua de Golomb (1)

```

1 include "globals.mzn";
2 %% GOLOMB mas n itens a serem escolhidos e repetidos
3 int: n = 3; %% NUM de PEDACOS
4 int: m = 6; %% TAMANHO 0 .. 6
5
6 array[1..n] of var 0..m : regua; %% TAMANHO dos PEDACOS
7 array[1..(n+1)] of var 0..m : regua_SAIDA; %% APENAS para OUT
8
9 constraint %% pedacos maior que 0
10     forall(i in 1 .. n) ( regua[i] > 0 );
11
12 %% Diferentes e decrescente ... PEDACOS/medidas
13 constraint
14     alldifferent ( regua ); %% /\ decreasing( regua );
15
16 %% Diferentes medidas/PEDACOS entre TODOS os CORTES
17 constraint
18     forall(i in 1 .. n-2) (
19         forall(j in i+1 .. n) (regua[ i ] != regua[ j ] ) );
20
21 constraint %% CRITERIO DE REGUA OTIMA
22     sum([regua[i] | i in 1..n] ) == m;

```


Vetor 1D: Régua de Golomb (2)

```

23
24 constraint %% formatando uma saida
25     regua_SAIDA[1] == 0 /\
26     forall(i in 1 .. n) (
27         regua_SAIDA[i+1] == regua_SAIDA[i] + regua[i]
28     );
29
30 % minimize ou maximize ALGO
31 solve satisfy;
32
33 output
34 [" Tamanho dos cortes: "][show(regua[i])++] " | " | i in 1 .. n][++
35 [" A REGUA: "][show(regua_SAIDA[j] ) ++" | " | j in 1 .. n+1 ]];

```

Saída:

```
$ mzn-g12fd -a sbpo_golomb_ruler.mzn
```

```
Tamanho dos cortes: 1 | 3 | 2 | A REGUA: 0 | 1 | 4 | 6 |
```

```
-----
```

```
Tamanho dos cortes: 1 | 2 | 3 | A REGUA: 0 | 1 | 3 | 6 |
```

```
-----
```

```
Tamanho dos cortes: 2 | 3 | 1 | A REGUA: 0 | 2 | 5 | 6 |
```

```
-----
```

```
Tamanho dos cortes: 2 | 1 | 3 | A REGUA: 0 | 2 | 3 | 6 |
```

```
-----
```

```
Tamanho dos cortes: 3 | 2 | 1 | A REGUA: 0 | 3 | 5 | 6 |
```

```
-----
```

```
Tamanho dos cortes: 3 | 1 | 2 | A REGUA: 0 | 3 | 4 | 6 |
```

```
-----
```

```
=====
```

Criando funções, variáveis locais e escopo (1)

Exemplo: `y=soma(vetor 1D)`

```

1 var int: y;
2 %% EQUIVALE ao sum( i in 1..n ) (vetor_1d[i]);
3 function var int: sum_array_1D(array[int] of var int: x_1D) =
4   let{
5     int : n = length(x_1D);
6     array[1..n] of var int : temp;
7     constraint                                     %%% C_1
8     temp[1] == x_1D[1];
9     constraint                                     %%% C_2
10    forall(i in 2..n)
11      ( temp[i] == temp[i-1] + x_1D[i] );
12    } in temp[n] ; %% Valor acumulado e RETORNO
13
14 constraint
15   y = sum_array_1D([3,4,-7,17,13,0]);
16
17 solve satisfy;
18 output [" SOMA: " ++ show(y), "\n",
19   " Lim Inf: ", show(lb_array([3,4,-7,17,13,0])), "\n",
20   " Lim Sup: ", show(ub_array([3,4,-7,17,13,0]))];

```

Saída:

```
$ mzn-g12fd -a sbpo_my_sum_vetor_1D.mzn
SOMA: 30
Lim Inf: -7
Lim Sup: 17
-----
=====
```

Vetores Bi-dimensionais ou 2D

Motivação

- As matrizes são essenciais em alguns problemas. Exemplo: *job-shop problem*
- Bi-dimensional (tem nomes especiais)
- A rigor MiniZinc estende a idéia para vetores n-ários (n-dimensões)

Vetores Bi-dimensionais ou 2D (1)

Representação

```

1 array[1..3, 1..2] of int : A;
2 A = [| 4, 5
3      | 0, 9
4      | 5, 8 |];
5
6 array[1..2, 1..3] of int : B;
7 B = array2d(1..2, 1..3,
8             [9,8,-3, 5,-5,7]);
9
10 array[1..2, 1..3, 1..2] of int : C;
11 C = array3d(1..2, 1..3, 1..2,
12             [9, -5, 3, 5, 6, 8,
13             19, 12, -13, 17, -15, 18]);
14
15 solve satisfy;
16
17 output[show2d(A), "\n", show2d(B), "\n", show3d(C)];

```

Saída:

```
$ mzn-g12fd -a minizinc/sbpo_ilustra-2D.mzn
```

```
[| 4, 5 |  
  0, 9 |  
  5, 8 |]
```

```
[| 9, 8, -3 |  
  5, -5, 7 |]
```

```
[| | 9, -5 |  
    3, 5 |  
    6, 8 |,
```

```
| 19, 12 |  
-13, 17 |  
-15, 18 | |]
```

```
-----  
=====
```

Vetores 2D

Quadrado Mágico

- Um quadrado mágico é uma matriz $N \times N$ onde os somatórios das linhas, colunas e diagonais (principal e secundária) são todos iguais a um valor K . Além disso, os elementos da matriz devem ser diferentes entre si, com valores de 1 a $N^2 - 1$.
- Um quadrado mágico de ordem 4 ($N = 4$) é dado por:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

- Onde $K = \frac{N(N^2+1)}{2}$ (valor mágico), para $N = 4$ tem-se $K = 34$

Quadrado Mágico (1)

```

1 int: N = 4;
2 float: Kte = ceil(N*(N*N + 1)/2); %% Coercao float -> int ou floor
3
4 set of int : Index = 1..N;
5
6 array[Index, Index] of var 1 .. (N*N)-1: mat;
7
8 constraint forall(i in Index) %% LINHAS
9     (mat[i,1] + mat[i,2] + mat[i,3] + mat[i,4] = Kte);
10
11 constraint forall(j in Index) %% COLUNAS
12     (mat[1,j] + mat[2,j] + mat[3,j] + mat[4,j] = Kte);
13
14 constraint % Diagonal 1...
15     mat[1,1] + mat[2,2] + mat[3,3] + mat[4,4] = Kte;
16
17 constraint % Diagonal 2 ...
18     mat[4,1] + mat[3,2] + mat[2,3] + mat[1,4] = Kte;
19
20 constraint %% alldifferent diferente
21     forall(i in Index, j in Index, k in i..N, l in j..N)
22         (if (i != k /\ j != l)

```

Quadrado Mágico (2)

```

23     then mat[i,j] != mat[k,l]
24     else
25     true
26     endif);
27
28 solve satisfy;
29
30 %%% obs na formatacao
31 output[show_int(5,mat[i,j]) ++
32     if j==N then "\n" else " " endif |
33     i in 1..N, j in 1..N];

```

Uma Saída:

```
$ mzn-g12mip sbpo_quadrado_magico.mzn
```

```

  9      7      4      14
  4      14     5      11
 11      5     10      8
 10      8     15      1
```

```
-----
```

```
PS: solver mzn-g12mip
```

```
$ mzn-gecode sbpo_quadrado_magico.mzn
```

```

  4      5     11     14
  7     15      3      9
  8      2     14     10
 15     12      6      1
```

```
-----
```

```
PS: solver mzn-gecode
```

Vetores 2D

Problema de Atribuição

Seja uma matriz de peso:

<div>Mulheres</div> <div>Homens</div>	M_1	M_2	M_3	M_4	M_5
H_1	1	11	13	7	3
H_2	6	5	2	8	10
H_3	6	3	9	4	12
H_4	32	17	6	18	11
H_5	1	3	4	1	5

- Como obter pares (H_i, M_j) tal que cada mulher/homem tenham **um único** companheira(o) apenas!
- Estende-se a idéia para: máquinas \times trabalhadores, processos \times tarefas, tradutores \times linguagens, etc.

Vetores 2D: Problema de Atribuição

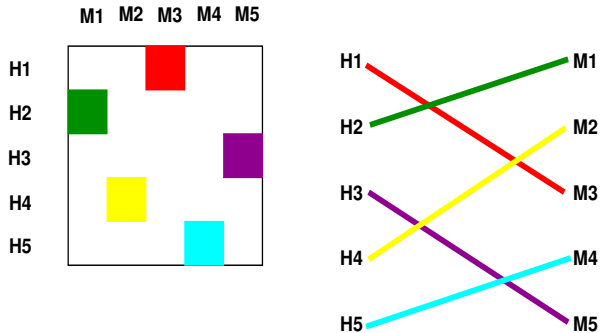


Figura: Matriz e o Grafo Bi-partido

Problema de Atribuição (1)

```

1 int: linhas = 4;      int: cols = 5;
2
3 %%%% m_PESO =  MATRIZ DEFINIDA  NO FINAL do arquivo
4
5 array[1..linhas, 1..cols] of int: m_PESO;
6 array[1..linhas, 1..cols] of var 0..1: x; %% x: MATRIZ DE DECISAO
7 var int: f_CUSTO;
8
9 constraint %% exatamente UMA escolha por linha
10 forall(i in 1..linhas) (
11     sum(j in 1..cols) (x[i,j]) == 1 );
12
13 constraint %% exatamente 0 ou UMA escolha por coluna
14 forall(j in 1..cols) (
15     sum(i in 1..linhas) (x[i,j]) <= 1 );
16
17 constraint %% Uma funcao CUSTO ou objetivo
18 f_CUSTO = sum(i in 1..linhas, j in 1..cols) (x[i,j]*m_PESO[i,j]);
19
20 solve maximize f_CUSTO;
21
22 output ["f_custo: ", show(f_CUSTO ), "\n", show2d(x)];

```

Problema de Atribuição (2)

```

23 /*****
24
25 m_PESO =
26 [ | 14, 5, 8, 7, 15,
27   | 2, 12, 6, 5, 3,
28   | 7, 8, 3, 9, 7,
29   | 2, 4, 6, 10, 1 | ];

```

Saída:

```
.....
MUITAS SAIDAS ATE ESTA COTA
-----
f_custo: 43
[| 1, 0, 0, 0, 0 |
  0, 1, 0, 0, 0 |
  0, 0, 0, 0, 1 |
  0, 0, 0, 1, 0 |]
-----
f_custo: 44
[| 0, 0, 0, 0, 1 |
  0, 1, 0, 0, 0 |
  1, 0, 0, 0, 0 |
  0, 0, 0, 1, 0 |]
-----
=====
```


Vetores 2D

Os Nadadores Americanos

Swimmer	Time (seconds)			
	Free	Breast	Fly	Back
Gary Hall	54	54	51	53
Mark Spitz	51	57	52	52
Jim Montgomery	50	53	54	56
Chet Jastremski	56	54	55	53

Figura: Do livro do *Operations Research: Applications and Algorithms* – Wayne L. Winston

Problema de Atribuição (1)

Nadadores Americanos

```

1 include "alldifferent.mzn";
2 int : N = 4;
3
4 array[1..N] of var 1..N: vetDecisao; %% VETOR Decisao 1D
5 var int: T_min;
6
7 array[1..N,1..N] of int: tempo_NADADORES;
8
9 tempo_NADADORES = array2d(1..N, 1..N,
10                             [54,54,51,53,
11                              51,57,52,52,
12                              50,53,54,56,
13                              56,54,55,53]);
14
15 constraint alldifferent(vetDecisao);
16
17 constraint
18     T_min = sum(i in 1..N)(tempo_NADADORES[i, vetDecisao[i]]);
19     %% Melhorar aqui
20
21 solve minimize T_min;

```

Problema de Atribuição (2)

Nadadores Americanos

```

22
23 output[" Menor tempo: ", show(T_min) , "\n",
24        " Atribuicao (vetDecisao): ", show(vetDecisao), "\n "] ++
25        [show(i) ++ ":" ++ show(vetDecisao[i]) ++ "-> " ++
26         show(tempo_NADADORES[ i, vetDecisao[i] ]) ++
27         "\n " | i in 1..N] ;

```

Saída:

```
$ mzn-g12fd -a minizinc/sbpo_nadadores.mzn
```

```
Menor tempo: 212
```

```
Atribuicao (vetDecisao): [3, 1, 4, 2]
```

```
1:3-> 51
```

```
2:1-> 51
```

```
3:4-> 56
```

```
4:2-> 54
```

```
-----
```

```
Menor tempo: 207
```

```
Atribuicao (vetDecisao): [3, 4, 1, 2]
```

```
1:3-> 51
```

```
2:4-> 52
```

```
3:1-> 50
```

```
4:2-> 54
```

```
-----
```

```
=====
```

Vetores 2D

Problema de *Job-Shop-Schedulling*

Tarefas (J_i)	Sequência			Tempo em M_j		
1	M_1	M_2	M_3	3	3	3
2	M_1	M_3	M_2	2	3	4
3	M_2	M_1	M_3	3	2	1

Job-Shop-Schedulling

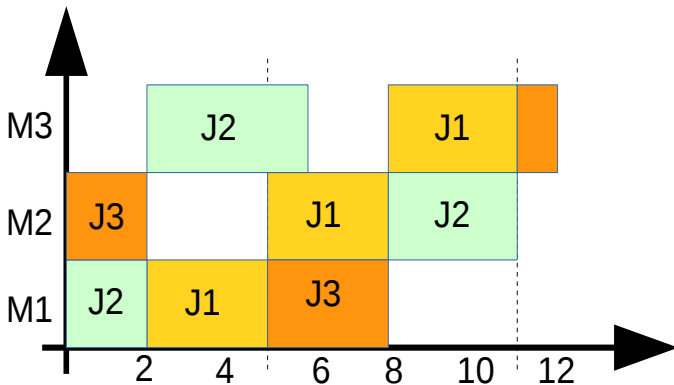


Figura: Uma solução !

Job-Shop-Scheduling

Ver código completo em:

https://github.com/claudiosa/CCS/minizinc/sbpo_job_shop.mzn

Partes do Código 1/3:

```

%% um limite para todos as tarefas tenham terminado
int: END_TIME =
    sum([job_time[j,k] | j in jobs, k in machines])+100;

%% A menor duracao eh o maior tempo de uma tarefa
var 0..END_TIME:
    min_duration =
        max([job_end[j, k] | j in jobs, k in machines]);

%% Evita inicializacoes com valores negativos
constraint
    forall(j in jobs, k in machines )
        (job_start[j,k] >= 0 );

%% Final de uma tarefa e o seu inicio + duracao
constraint
    forall(j in jobs, k in machines )
        (job_end[j,k] = job_start[j,k] + job_time[j,k]);
  
```


Partes do Código 2/3:

```

constraint          %% PRECEDENCIA -- MAIS DIFICIL
forall( j in  jobs)
  (forall( k1, k2  in  machines where k1 < k2)
    (if( job_sequence[j,k1] < job_sequence[j,k2] )
      then
        job_end[j,k1] <= job_start[j, k2]
      %%Tempo final de J1 eh menor ou igual o tempo de inicio de J2
      else
        job_end[j,k2] <= job_start[j, k1]
      endif
    ));
  
```

Partes do Código 3/3:

```

%% Disjuncao entre todas as tarefas sobre uma maquina
constraint
%% para cada MAQUINA cada JOB sera DISJUNTIVO
forall( k in machines )
    (disjunctive([job_start[j,k] | j in jobs ] ,
                 [job_time[j,k] | j in jobs ])) );

```

```
$ mzn-g12fd -a sbpo_job_shop.mzn
.....
job_start =
[|  2,  5,  8 |
  0,  8,  2 |
  5,  0, 11 |]
job_end =
[|  5,  8, 11 |
  2, 11,  6 |
  8,  2, 12 |]
.....
t_end = 12      MUITAS RESPOSTAS COM ESTA COTA
.....
SAIDA DETALHADA:

JOB 1 : 2..5  5..8  8..11
JOB 2 : 0..2  8..11 2..6
JOB 3 : 5..8  0..2 11..12
=====
```

Vetor 2D – Grafos



Figura: Coloração de Mapas – Regiões da Itália

Coloração de Mapas (1)

```

1  int: n=8; %% REGIOES
2  int: c=4; %% CORES
3
4  array [1..n, 1..n] of int : Adj; %% Matriz Adjacencia
5  array [1..n] of var 1..c : Col; %% Saida
6
7  constraint
8      forall (i in 1..n, j in i+1..n)
9          (if Adj[i,j] == 1 then Col[i] != Col[j] else true endif);
10
11 solve satisfy;
12
13 output [show(Col)];
14
15 Adj = [|0,1,0,0,0,0,0,0
16         |1,0,1,1,1,0,0,0
17         |0,1,0,1,0,0,0,0
18         |0,1,1,0,1,1,0,0
19         |0,1,0,1,0,1,1,0
20         |0,0,0,1,1,0,1,1
21         |0,0,0,0,1,1,0,0
22         |0,0,0,0,0,1,0,0|];

```

Coloração de Mapas (2)

```

23
24 %% Regioes da Italia
25 % 1 Friuli Venezia Giulia
26 % 2 Veneto
27 % 3 Trentino Alto Adige
28 % 4 Lombardy
29 % 5 Emilia-Romagna
30 % 6 Piedmont
31 % 7 Liguria
32 % 8 Aosta Valley
33 %%% CONEXOES
    
```

Saída:

```
$ mzn-gecode sbpo_coloracao_mapas.mzn  
[2, 1, 2, 3, 2, 1, 3, 2]  
UMA SAIDA
```

Melhorando as Buscas

Práticas nos experimentos

- Usar restrições globais \Rightarrow tem muitas!
- Restrições complexas. Exemplo: restrições reifadas, restrições *entubadas* ($y = f(x) \Leftrightarrow x = g(x)$)
- Comece com domínios reduzidos e vá aumentando gradativamente ao testar seus modelos \Rightarrow *comece pequeno*
- Variar as estratégias de buscas (eis a PR!) \Rightarrow ponto de exploração

Variando as Buscas

Parâmetro *search*

```
solve :: int_search(Var_Exp, Sel_VAR, Sel_DOM, estrategia)
        satisfy (ou minimize ou maximize);
```

Formato Geral:

```
int_search(Var_Exp, Sel_VAR, Sel_DOM, estrategia)
bool_search(Var_Exp, Sel_VAR, Sel_DOM, estrategia)
set_search(Var_Exp, Sel_VAR, Sel_DOM, estrategia)
```

Escolha da variável: input_order, first_fail, smallest, largest, dom_w_deg

- input_order: a ordem que vão aparecendo
- first_fail: variável com o menor tamanho de domínio
- anti_first_fail: oposto da anterior
- smallest: variável com o menor valor no domínio
- largest: variável com o maior valor no domínio

Variando as Buscas

Parâmetro *search*

Escolha do valor no domínio:

`indomain_min`, `indomain_max`, `indomain_median`, `indomain_random`,
`indomain_split`, `indomain_reverse_split`

Exemplificando, considere o domínio: $\{1, 3, 4, 18\}$

- `indomain_min`: 1, 3, ...
- `indomain_max`: 18, 4, ...
- `indomain_median`: 3, 4, ...
- `indomain_split`: $x \leq (1+18)/2$; $x > (1+18)/2$
- `indomain_reverse_split`: $x > (1+18)/2$; $x \leq (1+18)/2$

Exemplo *Search* (1)

```

1 include "globals.mzn";
2 int : N = 4;
3 array[1..N] of var 1..7 : x;
4
5 constraint
6     alldifferent(x) /\ increasing (x);
7 constraint
8     sum(x) == 13;
9
10 ann: Selec_VAR;    %% CRIANDO TIPOS - ann
11 ann: Selec_DOM;
12
13 solve :: int_search(x, Selec_VAR, Selec_DOM, complete)
14         satisfy;
15
16 %%% Vah modificando AQUI
17 Selec_VAR = dom_w_deg; %% dom_w_deg, first_fail, largest
18 Selec_DOM = indomain_min; %%

```

- ann: cria um tipo de anotação no código
- **Atenção:** cuidar das compatibilidades entre o parâmetro *search* e o *backend* utilizado.

Saída:

```
$ time(mzn-g12fd -a sbpo_var_val_choice.mzn)
x = array1d(1..4 ,[1, 2, 3, 4]);
-----
x = array1d(1..4 ,[1, 2, 3, 5]);
x = array1d(1..4 ,[1, 2, 3, 6]);
x = array1d(1..4 ,[1, 2, 3, 7]);
x = array1d(1..4 ,[1, 2, 4, 5]);
x = array1d(1..4 ,[1, 2, 4, 6]);
x = array1d(1..4 ,[1, 3, 4, 5]);
----- %%% REMOVIDOS ENTRE AS RESPOSTAS ACIMA
=====
real 0m0.158s
user 0m0.076s
sys 0m0.012s
```

Restrições Globais

`include "globals.mzn";` muito úteis:

- Restrições de escalonamento: `disjunctive`, `cumulative`, `alternative`;
- Restrições de ordenamento: `decreasing`, `increasing`, `sort`, etc
- Restrições extensionais: `regular`, `regular_nfa`, `table`;
- Restrições de empacotamento: `bin_packing`, `bin_packing_capa`,
- Restrições de *entubamento* (*channeling*): `int_set_channel`, `inverse`, `link_set_to_booleans`, etc
- Restrições de *genéricas-I*: `all_different`, `all_disjoint` (uso em conjuntos), `all_equal`, `nvalue`, etc
- Restrições de *genéricas-II*: `arg_max`, `arg_min`, `circuit`, `disjoint`, `maximum`, `member`, `minimum`, `network_flow`, `network_flow_cost`, `range`, `partition_set`, `sliding_sum`, etc

String e Fix (1)

```

1 set of int : Index = 1 .. 4;
2
3 array[Index] of string:
4     Estacoes = ["Verao", "Outono", "Inverno", "Primavera"];
5 var Index : x;
6
7 constraint
8     x >= 2;
9
10 solve satisfy;
11 output [ Estacoes[ fix (x) ], "\n", show(Estacoes) ];

```

Nota:

- Verifica se a variável está *fixada* e faz uma coerção de tipos
- Coerções possíveis: boo2int, int2float, set2array

Saída

```
$ mzn-gecode -a sbpo_string_fix.mzn
Outono
["Verao", "Outono", "Inverno", "Primavera"]
-----
Inverno
["Verao", "Outono", "Inverno", "Primavera"]
-----
Primavera
["Verao", "Outono", "Inverno", "Primavera"]
-----
=====
```

Voltando aos Conjuntos (1)

```

1 include "globals.mzn";
2 int : K = 7;
3 int: num_sets = 4; %%
4 set of int : U = 0..10; %% UNIVERSO
5 %% N conjuntos em y_sub[1] ate y_sub[N]
6 array[1 .. num_sets] of var set of U : y_sub;
7
8 %% BY HAKAN
9 predicate set_sum(var set of int: s, var int: the_sum) =
10   the_sum = sum(i in ub(s)) (bool2int(i in s) * i);
11
12 function var int: sum_SET(var set of int: SET) =
13   %sum(i in ub(s)) ( bool2int(i in s) * i);   %%% OU
14   sum([bool2int(i in SET) * i | i in ub(SET)]); % 1 * i
15
16 constraint
17   forall(i in 2..num_sets) (
18     card(y_sub[i]) == card(y_sub[i-1]) /\ %%card(y[i]) <= 4 /\
19     K == sum_SET(y_sub[i-1]) /\
20     set_sum(y_sub[i], K) );
21
22 constraint

```


Voltando aos Conjuntos (2)

```

23     alldifferent(y_sub); %% POLIMORFISMO
24
25 solve :: set_search(y_sub, Selec_VAR, Selec_DOM, complete)
26         satisfy;
27
28 ann: Selec_VAR;    %% CRIANDO TIPOS - ann
29 ann: Selec_DOM;
30
31 %%%% Modificando AQUI
32 Selec_VAR = first_fail;
33 Selec_DOM = indomain_min;

```

Nota:

- Nem todos os *solvers* estão preparados para os recursos de *set*
- Poderosos e trabalhosos

Saída

```
$ mzn-g12fd -a sbpo_mais_conjuntos.mzn
y_sub = array1d(1..4 ,[{0,1,6}, {0,2,5}, {0,3,4}, {1,2,4}]);
-----
y_sub = array1d(1..4 ,[{0,1,6}, {0,2,5}, {1,2,4}, {0,3,4}]);
-----
.....
ALGUMAS MUITAS SAIDAS
.....
y_sub = array1d(1..4 ,[3..4, {2,5}, {0,7}, {1,6}]);
-----
y_sub = array1d(1..4 ,[3..4, {2,5}, {1,6}, {0,7}]);
-----
=====
```

Conclusões

1. Formulação matemática \approx código MiniZinc
2. Declarativo \Rightarrow escrever “**o que**” e muito direto
- 3.
- 4.
5. Exemplos de mais códigos:
 - <https://github.com/hakank/hakank/tree/master/minizinc>
 -
 - <https://github.com/MiniZinc/> (**cuidado**)

Referências Bibliográficas

ALGUM BIBTEX aqui....