MiniZinc Handbook

发行版本 *2.8.5*

Peter J. Stuckey, Kim Marriott, Guido Tack

2024 年 05 月 17 日

Contents

指南

索引

节节节节

CHAPTER 1.1

Introduction

1.1.1 Structure

1.1.2 How to Read This

列表dummy.mzn

```
% Just an example
var int: x;
solve satisfy;
```

CHAPTER 1.2

Installation

1.2.1 Microsoft Windows

.mzn.dzn.fzn

C:\>setx PATH "%PATH%;C:\Program Files\MiniZinc 2.8.5 (bundled)\"

1.2.2 Linux

1.2.2.1 Snap

minizincminizinc.ide

```
$ snap install minizinc --classic
```

1.2.2.2 AppImage

```
$ chmod +x MiniZincIDE-2.8.5-x86_64.AppImage
$ ./MiniZincIDE-2.8.5-x86_64.AppImage
```

```
~/Applications//opt/Applications/minizincMiniZincIDE/usr/local/bin$HOME/.local/share/{applicatio
```

```
$ mv ./MiniZincIDE-2.8.5-x86_64.AppImage /opt/Applications/
$ /opt/Applications/MiniZincIDE-2.8.5-x86_64.AppImage install
```

```
$BIN_LOCATION$DESKTOP_LOCATION
```

```
$ BIN_LOCATION=/usr/local/bin DESKTOP_LOCATION=$HOME/.local/share ./MiniZincIDE-2.8.
↪5-x86_64.AppImage install
```

1.2.2.3 Archive

,

```
$ tar xf MiniZincIDE-2.8.5-bundle-linux-x86_64.tgz
```

```
.tgzMiniZincIDE.sh
```

```
$ export PATH=MiniZincIDE-2.8.5-bundle-linux-x86_64/bin:$PATH
$ export LD_LIBRARY_PATH=MiniZincIDE-2.8.5-bundle-linux-x86_64/lib:$LD_LIBRARY_PATH
$ export QT_PLUGIN_PATH=MiniZincIDE-2.8.5-bundle-linux-x86_64/plugins:$QT_PLUGIN_PATH
```

1.2.3 Apple macOS

.dmg'

```
$ export PATH=/Applications/MiniZincIDE.app/Contents/Resources:$PATH
```

1.2.4 Adding Third-party Solvers

节节

CHAPTER 1.3

First steps with MiniZinc

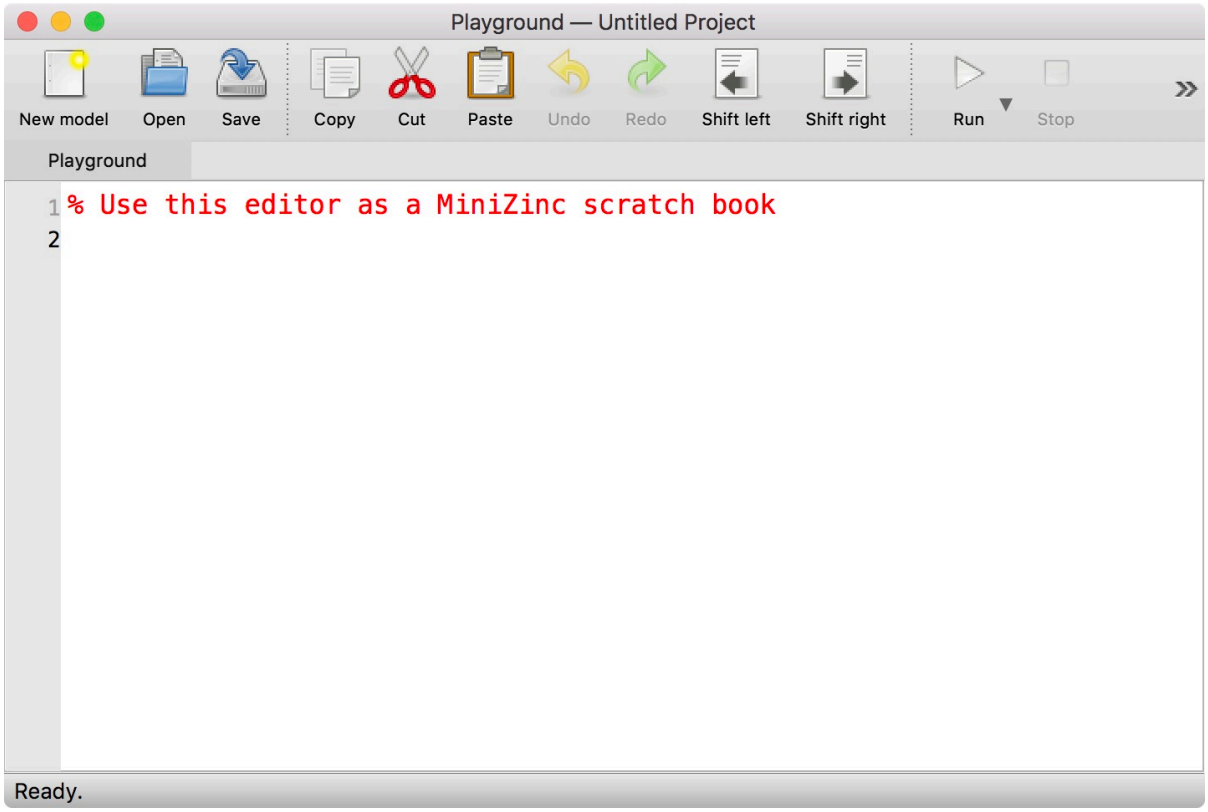
节

minizinc节minizinc

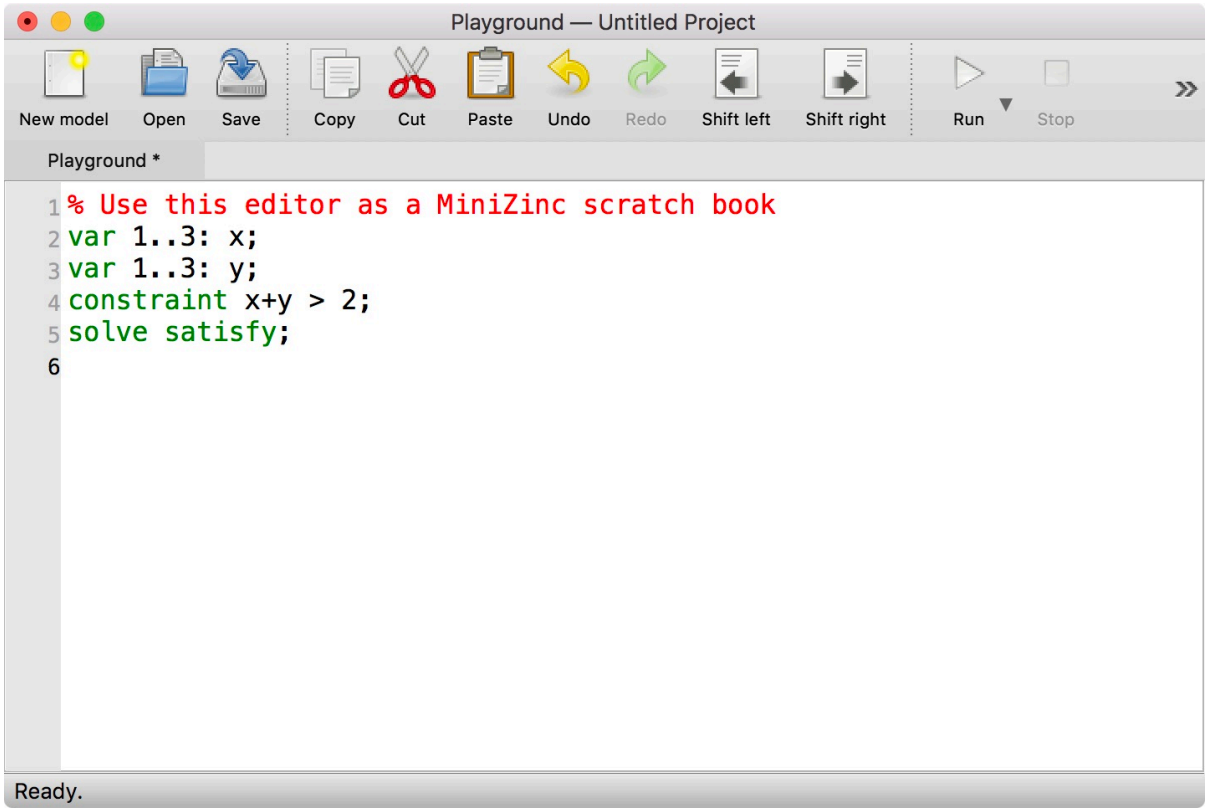
1.3.1 The MiniZinc IDE

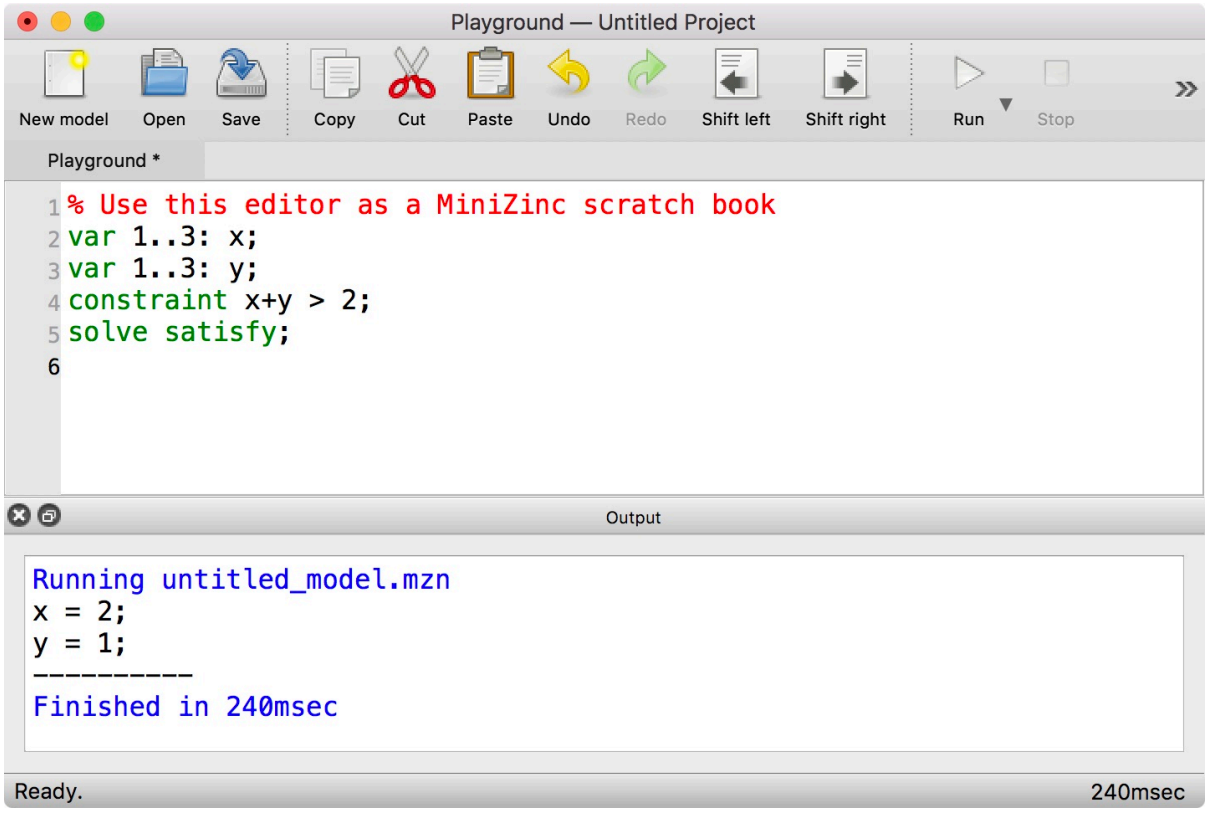
，

节

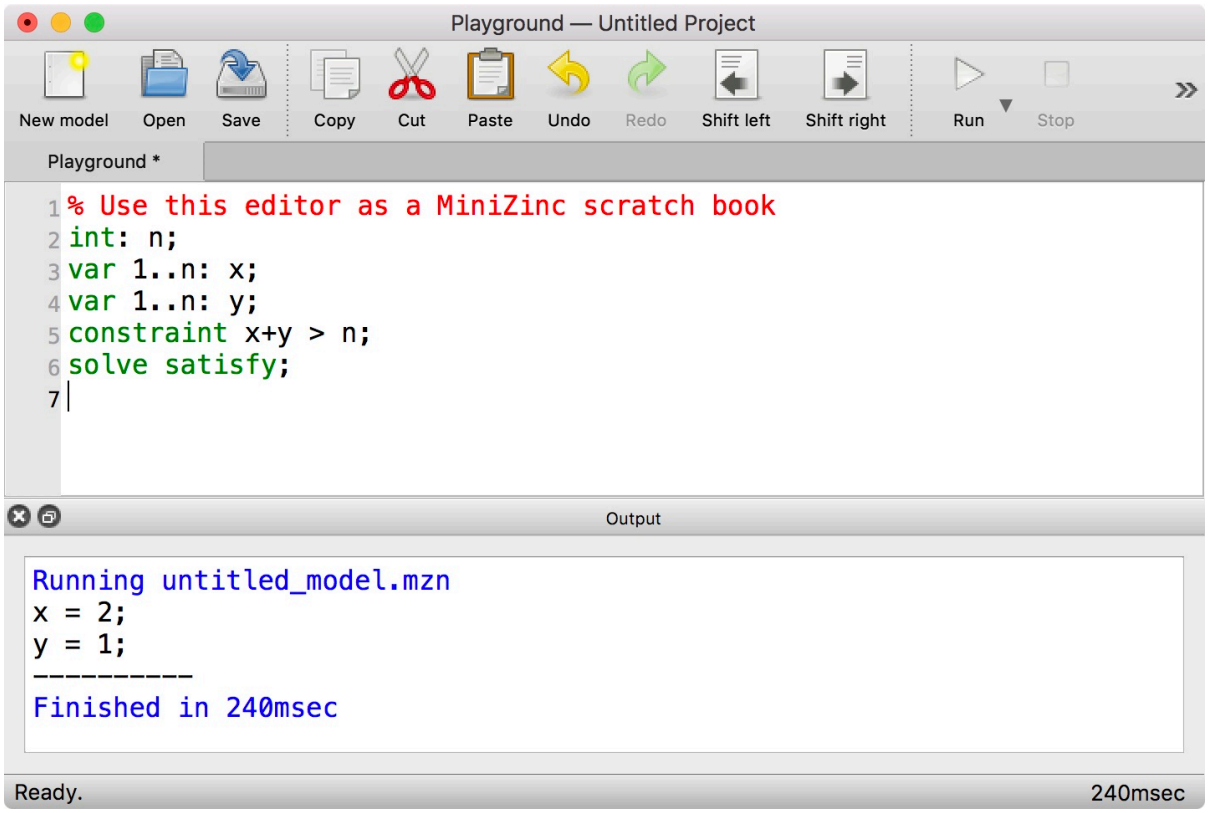


,

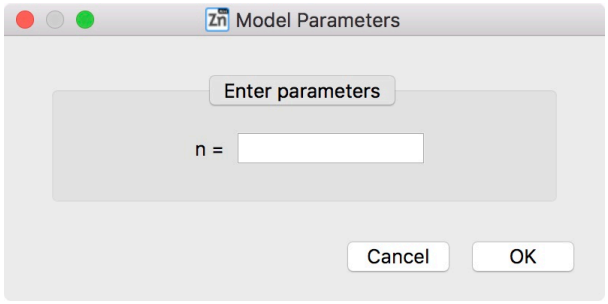

Playground — Untitled Project
New model
Open
Save
Copy
Cut
Paste
Undo
Redo
Shift left
Shift right
Run
Stop
Playground *
1 % Use this editor as a MiniZinc scratch book
2 var 1..3: x;
3 var 1..3: y;
4 constraint x+y > 2;
5 solve satisfy;
6
Ready.

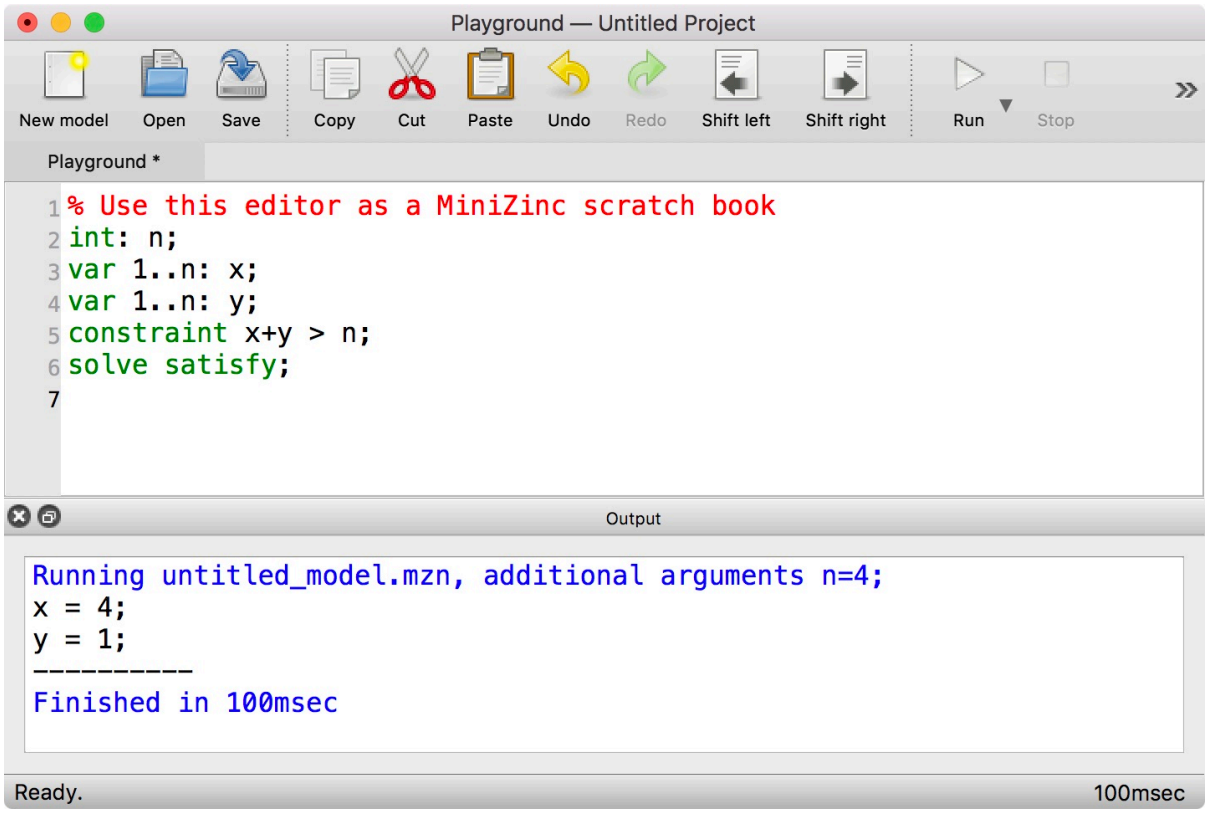

Playground — Untitled Project
New model
Open
Save
Copy
Cut
Paste
Undo
Redo
Shift left
Shift right
Run
Stop
Playground *
1 % Use this editor as a MiniZinc scratch book
2 var 1..3: x;
3 var 1..3: y;
4 constraint x+y > 2;
5 solve satisfy;
6
Output
Running untitled_model.mzn
x = 2;
y = 1;

Finished in 240msec
Ready.
240msec


Playground — Untitled Project
New model
Open
Save
Copy
Cut
Paste
Undo
Redo
Shift left
Shift right
Run
Stop
Playground *
1 % Use this editor as a MiniZinc scratch book
2 int: n;
3 var 1..n: x;
4 var 1..n: y;
5 constraint x+y > n;
6 solve satisfy;
7
Output
Running untitled_model.mzn
x = 2;
y = 1;

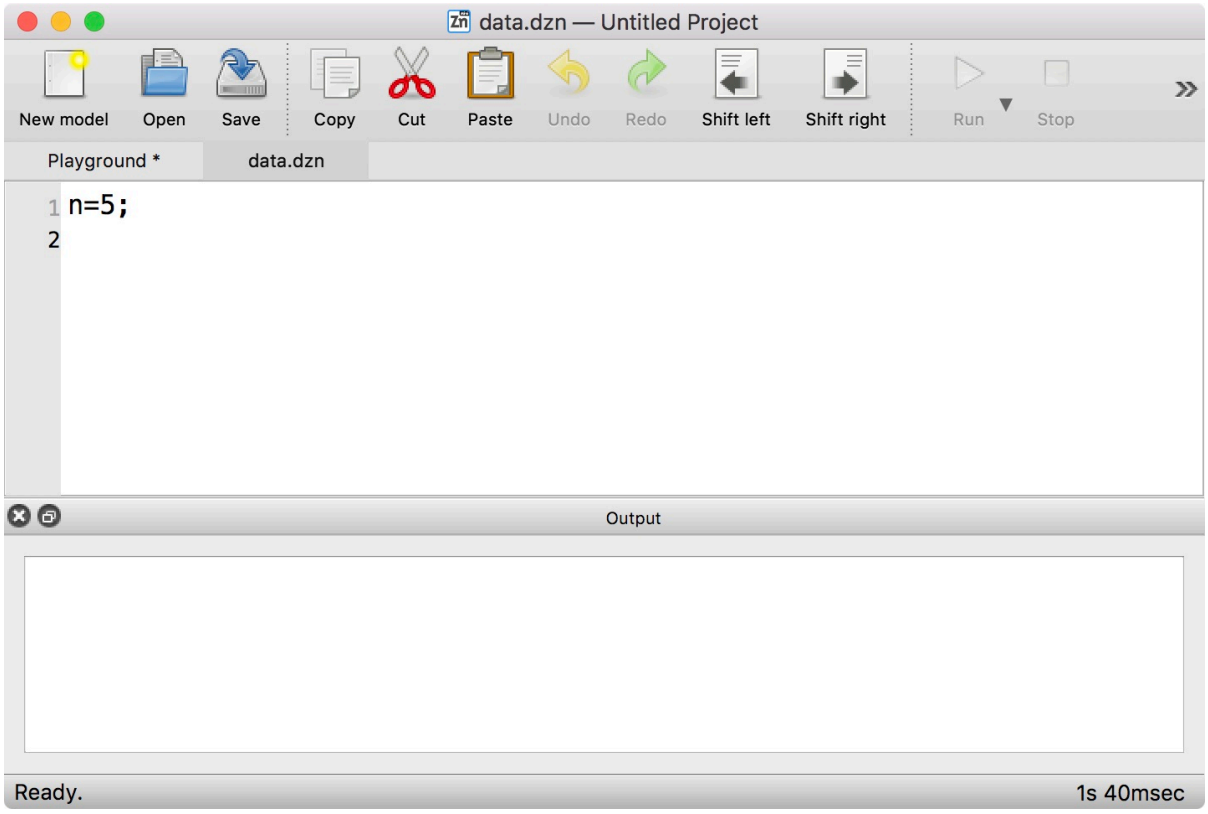
Finished in 240msec
Ready.
240msec

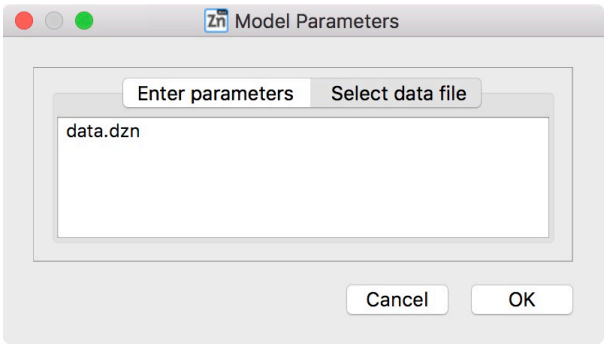

Model Parameters
Enter parameters
n =
Cancel
OK


Playground — Untitled Project
New model
Open
Save
Copy
Cut
Paste
Undo
Redo
Shift left
Shift right
Run
Stop
Playground *
1 % Use this editor as a MiniZinc scratch book
2 int: n;
3 var 1..n: x;
4 var 1..n: y;
5 constraint x+y > n;
6 solve satisfy;
7
Output
Running untitled_model.mzn, additional arguments n=4;
x = 4;
y = 1;

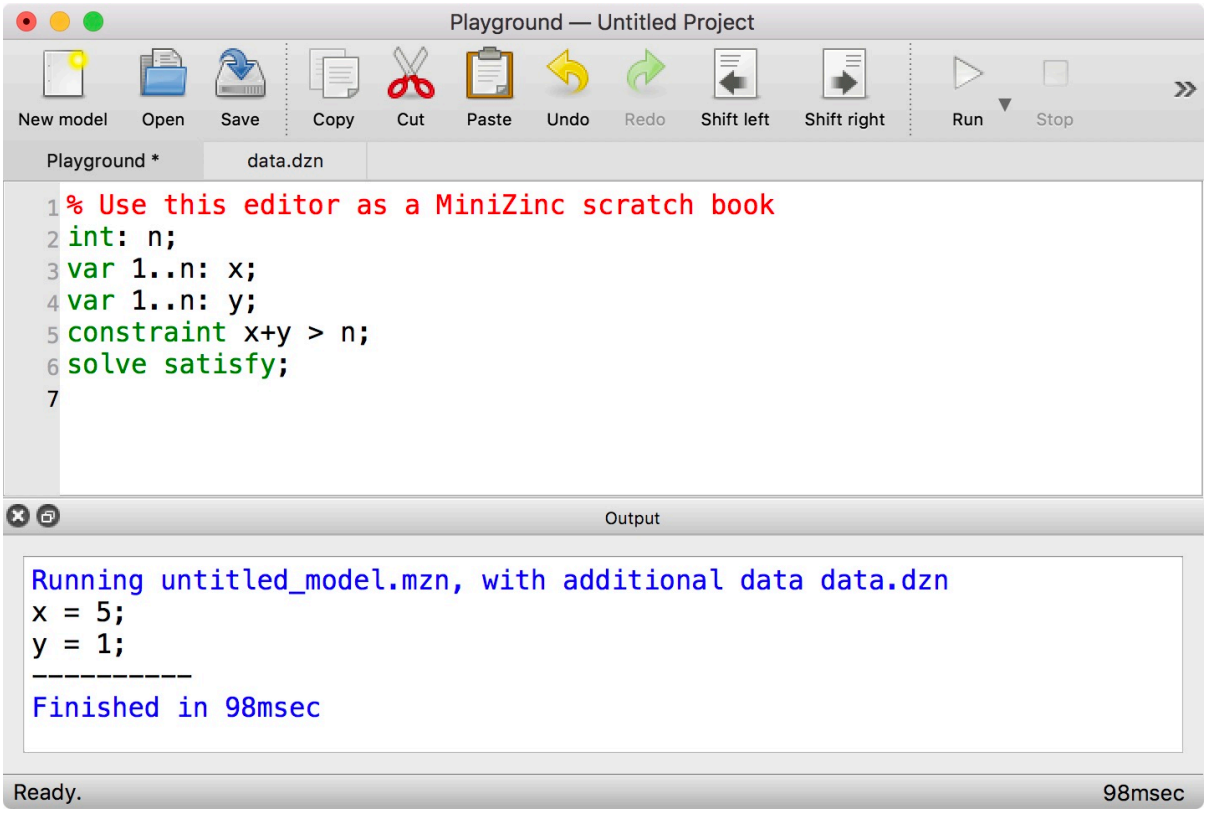
Finished in 100msec
Ready.
100msec

’ data.dzn


data.dzn — Untitled Project
New model
Open
Save
Copy
Cut
Paste
Undo
Redo
Shift left
Shift right
Run
Stop
Playground *
data.dzn
1 n=5;
2
Output
Ready.
1s 40msec



data.dzn



节

1.3.2 The MiniZinc command line tool

minizincPATH节

’ model.mzn

```
var 1..3: x;
var 1..3: y;
constraint x+y > 3;
solve satisfy;
```

minizinc

```
$ minizinc model.mzn
x = 3;
y = 1;
----------
$
```

minizinc

```
$ minizinc model.mzn data.dzn
x = 5;
y = 1;
----------
$
```

minizinc-a

```
$ minizinc -a model.mzn
x = 3;
y = 1;
----------
x = 2;
y = 2;
----------
x = 3;
y = 2;
----------
x = 1;
y = 3;
----------
x = 2;
y = 3;
----------
x = 3;
y = 3;
----------
==========
$
```

minizincminizinc --help节

指南

CHAPTER 2.1

MiniZinc 基本模型

在此节中，我们利用两个简单的例子来介绍一个模型的基本结构。

2.1.1 第一个实例

图澳大利亚各州

作为我们的第一个例子，假设我们要去给图中的澳大利亚地图涂色。它包含了七个不同的州和地区，而每一块都要被涂一个颜色来保证相邻的区域有不同的颜色。

列表一个用来给澳大利亚的州和地区涂色的模型aust.mzn

```
% 用nc个颜色来涂澳大利亚
int: nc = 3;

var 1..nc: wa;   var 1..nc: nt;  var 1..nc: sa;   var 1..nc: q;
var 1..nc: nsw;  var 1..nc: v;   var 1..nc: t;

constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;
solve satisfy;

output ["wa=\(wa)\t nt=\(nt)\t sa=\(sa)\n",
        "q=\(q)\t nsw=\(nsw)\t v=\(v)\n",
         "t=", show(t),  "\n"];
```

我们可以很容易的用给此问题建模。此模型在列表中给出。

模型中的第一行是注释。注释开始于%来表明此行剩下的部分是注释。同时也含有语言风格的由/*开始和*/结束的块注释。

模型中的下一部分声明了模型中的变量。此行

```
int: nc = 3;
```

定义了一个问题的参数来代表可用的颜色个数。在很多编程语言中，参数和变量是类似的。它们必须被声明并且指定一个类型。在此例中，类型是int。通过赋值，它们被设了值。允许变量声明时被赋值（就像上面那一行）或者单独给出一个赋值语句。因此下面的表示跟上面的只一行表示是相等的

```
int: nc;
nc = 3;
```

和很多编程语言中的变量不一样的是，这里的参数只可以被赋唯一的值。如果一个参数出现在了多于一个的赋值中，就会出现错误。

基本的参数类型包括整型int，浮点型float，布尔型bool以及字符串型string。同时也支持数组和集合。

模型同时也可能包含另一种类型的变量决策变量。决策变量是数学的或者逻辑的变量。和一般的编程语言中的参数和变量不同，建模者不需要给决策变量一个值。而是在开始时，一个决策变量的值是不知道的。只有当模型被执行的时候，求解系统才来决定决策变量是否可以被赋值从而满足模型中的约束。若满足，则被赋值。

在我们的模型例子中，我们给每一个区域一个决策变量wantsaqnswv和t。它们代表了会被用来填充区域的（未知）颜色。

对于每一个决策变量，我们需要给出变量可能的取值集合。这个被称为变量的定义域。定义域部分可以在变量声明的时候同时给出，这时决策变量的类型就会从定义域中的数值的类型推断出。

中的决策变量的类型可以为布尔型，整型，浮点型或者集合。同时也可以是元素为决策变量的数组。在我们的模型例子中，我们使用整型去给不用的颜色建模。通过使用var声明，我们的每一个决策变量被声明为定义域为一个整数类型的范围表示1..nc，来表明集合$\{1, 2, \ldots, nc\}$。所有数值的类型为整型，所以模型中的所有的变量是整型决策变量。

标识符

用来命名参数和变量的标识符是一列由大小写字母，数字以及下划线_字符组成的字符串。它们必须开始于一个字母字符。因此myName_2是一个有效的标识符。（和）的关键字不允许被用为标识符名字。它们在中被列出。所有的操作符都不能被用做标识符名字。它们在中被列出。

仔细地区别了以下两种模型变量：参数和决策变量。利用决策变量创建的表达式类型比利用参数可以创建的表达式类型更局限。但是，在任何可以用决策变量的地方，同类型的参数变量也可以被应用。

整型变量声明

一个整型参数变量可以被声明为以下两种方式：

```
int : <变量名>
<l> .. <u> : <变量名>
```

<l>和<u>是固定的整型表达式。
一个整型决策变量被声明为以下两种方式：

```
var int : <变量名>
var <l>..<u> : <变量名>
```

<l>和<u>是固定的整型表达式。

参数和决策变量形式上的区别在于对变量的实例化。变量的实例化和类型的结合被叫为类型实例化。既然你已经开始使用，毫无疑问的你会看到很多类型实例化的错误例子。

模型的下一部分是约束。它们详细说明了决策变量想要组成一个模型的有效解必须要满足的布尔型表达式。在这个例子中我们有一些决策变量之间的不等式。它们规定如果两个区域是相邻的，则它们必须有不同的颜色。

关系操作符

提供了以下关系操作符关系操作符：
相等===不等!=小于<大于>小于等于<=和大于等于>=

模型中的下一行：

```
solve satisfy;
```

表明了它是什么类型的问题。在这个例子中，它是一个满足问题：我们希望给决策变量找到一个值使得约束被满足，但具体是哪一个值却没有所谓。

模型的最后一个部分是输出语句。它告诉当模型被运行并且找到一个解解的时候，要输出什么。

输出和字符串

一个输出语句跟着一串字符。它们通常或者是写在双引号之间的字符串常量字符串常量并且对特殊字符用类似语言的标记法，或者是show(e)格式的表达式，其中e是表达式。例子中的\n代表换行符，\t代表制表符。
数字的show有各种不同方式的表示：show_int(n,X)在至少个字符里输出整型X的值，若$n > 0$则右对齐，否则则左对齐；show_float(n,d,X)在至少$|n|$个字符里输出浮点型X的值，若$n > 0$则右对齐，否则则左对齐，并且小数点后有d个字符。
字符串常量必须在同一行中。长的字符串常量可以利用字符串连接符++来分成几行。例如，字符串常量

```
"Invalid datafile: Amount of flour is non-negative"
```

和字符串常量表达式

```
"Invalid datafile: " ++
"Amount of flour is non-negative"
```

是相等的。
支持内插字符串内插字符串。表达式可以直接插入字符串常量中。"\(e)"形式的子字符串会被替代为show(e)。例如，"t=\(t)\n"产生和"t=" ++ show(t) ++ "\n"一样的字符串。
一个模型可以包含多个输出语句。在这种情况下所有输出会根据它们在模型中出现的顺序连接。

我们可以通过点击中的按钮或者输入

```
$ minizinc --solver Gecode aust.mzn
```

来评估我们的模型。其中aust.mzn是包含我们的模型的文件名字。我们必须使用文件扩展名.mzn来表明一个模型。带有--solver Gecode选项的命令minizinc使用有限域求解器去评估我们的模型。如果你使用的是二进制发布这个求解器实际上是预设的所以你也可以运行minizinc aust.mzn。

当我们运行上面的命令后，我们得到如下的结果：

```
wa=2   nt=3  sa=1
q=2  nsw=3   v=2
t=1
----------
```

个破折号----------这行是自动被输出的，用来表明一个解已经被找到。

2.1.2 算术优化实例

我们的第二个例子来自于要为了本地的校园游乐会烤一些蛋糕的需求。我们知道如何制作两种蛋糕。警告请不要在家里使用这些配方制作一个香蕉蛋糕的制作需要克自发酵的面粉，个捣碎的香蕉，克糖和克黄油。一个巧克力蛋糕的制作需要克自发酵的面粉，克可可粉，克糖和克黄油。一个巧克力蛋糕可以卖，一个香蕉蛋糕可以卖。我们一共有千克的自发酵面粉，个香蕉，千克的糖，克的黄油和克的可可粉。问题是对每一种类型的蛋糕，我们需要烤多少给游乐会来得到最大的利润。一个可能的模型在列表中给出。

列表决定为了校园游乐会要烤多少香蕉和巧克力蛋糕的模型。cakes.mzn

```
% 为校园游乐会做蛋糕

var 0..100: b; % 香蕉蛋糕的个数
var 0..100: c; % 巧克力蛋糕的个数

% 面粉
constraint 250*b + 200*c <= 4000;
% 香蕉
constraint 2*b  <= 6;
% 糖
constraint 75*b + 150*c <= 2000;
% 黄油
constraint 100*b + 150*c <= 500;
% 可可粉
constraint 75*c <= 500;

% 最大化我们的利润
solve maximize 400*b + 450*c;

output ["no. of banana cakes = \(b)\n",
        "no. of chocolate cakes = \(c)\n"];
```

第一个新特征是的使用。

整数算术操作符

提供了标准的整数算术操作符。加+减-乘*整数除div和整数模mod同时也提供了一元操作符+和-。
整数模被定义为输出和被除数a一样正负的$a \bmod b$值。整数除被定义为使得$a = b * (a \operatorname{div} b) + (a \bmod b)$值。
提供了标准的整数函数用来取绝对值abs和幂函数pow例如abs(-4)和pow(2,5)分别求得数值4和32。
算术常量的语法是相当标准的。整数常量可以是十进制，十六进制或者八进制。例如05 1230x1b70o777。

例子中的第二个新特征是优化。这行

```
solve maximize 400 * b + 450 * c;
```

指出我们想找一个可以使语句中的表达式（我们叫做目标）最大化的解。这个目标可以为任何类型的算术表达式。我们可以把关键字maximize换为minimize来表明一个最小化问题。

当我们运行上面这个模型时，我们得到以下的结果：

```
no. of banana cakes = 2
no. of chocolate cakes = 2
----------
==========
```

一旦系统证明了一个解是最优解，这一行==========在最优化问题中会自动被输出。

2.1.3 数据文件和谓词

此模型的一个缺点是如果下一次我们希望解决一个相似的问题，即我们需要为学校烤蛋糕（这是经常发生的），我们需要改变模型中的约束来表明食品间拥有的原料数量。如果我们想重新利用此模型，我们最好使得每个原料的数量作为模型的参数，然后在模型的最上层设置它们的值。

更好的办法是在一个单独的数据文件中设置这些参数的值。（就像很多其他的建模语言一样）允许使用数据文件来设置在原始模型中声明的参数的值。通过运行不同的数据文件，使得同样的模型可以很容易地和不同的数据一起使用。

数据文件的文件扩展名必须是.dzn，来表明它是一个数据文件。一个模型可以被任何多个数据文件运行（但是每个变量参数在每个文件中只能被赋一个值）

列表独立于数据的用来决定为了校园游乐会要烤多少香蕉和巧克力蛋糕的模型。
cakes2.mzn

```
% 为校园游乐会做蛋糕（和数据文件一起）

int: flour;  %拥有的面粉克数
int: banana; %拥有的香蕉个数
int: sugar;  %拥有的糖克数
int: butter; %拥有的黄油克数
int: cocoa;  %拥有的可可粉克数

constraint assert(flour >= 0,"Invalid datafile: " ++
                  "Amount of flour should be non-negative");
constraint assert(banana >= 0,"Invalid datafile: " ++
                  "Amount of banana should be non-negative");
constraint assert(sugar >= 0,"Invalid datafile: " ++
                  "Amount of sugar should be non-negative");
constraint assert(butter >= 0,"Invalid datafile: " ++
                  "Amount of butter should be non-negative");
constraint assert(cocoa >= 0,"Invalid datafile: " ++
                  "Amount of cocoa should be non-negative");

var 0..100: b; % 香蕉蛋糕的个数
var 0..100: c; % 巧克力蛋糕的个数

% 面粉
constraint 250*b + 200*c <= flour;
% 香蕉
constraint 2*b  <= banana;
% 糖
constraint 75*b + 150*c <= sugar;
% 黄油
constraint 100*b + 150*c <= butter;
% 可可粉
constraint 75*c <= cocoa;
```

```
% 最大化我们的利润
solve maximize 400*b + 450*c;

output ["no. of banana cakes = \(b)\n",
        "no. of chocolate cakes = \(c)\n"];
```

我们的新模型在列表中给出。我们可以用下面的命令来运行

数据文件pantry.dzn在列表中给出。我们得到和cakes.mzn同样的结果。运行下面的命令

```
$ minizinc cakes2.mzn pantry2.dzn
```

利用另外一个列表中定义的数据集，我们得到如下结果

```
no. of banana cakes = 3
no. of chocolate cakes = 8
----------
==========
```

如果我们从cakes.mzn中去掉输出语句，会使用默认的输出。这种情况下得到的输出是

```
b = 3;
c = 8;
----------
==========
```

默认输出

一个没有输出语句的模型会给每一个决策变量以及它的值一个输出行，除非决策变量已经在声明的时候被赋了一个表达式。注意观察此输出是如何呈现一个正确的数据文件格式的。

列表cakes2.mzn的数据文件例子pantry.dzn

```
flour = 4000;
banana = 6;
sugar = 2000;
butter = 500;
cocoa = 500;
```

列表cakes2.mzn的数据文件例子pantry2.dzn

```
flour = 8000;
banana = 11;
sugar = 3000;
butter = 1500;
cocoa = 800;
```

通过使用命令行标示-D，小的数据文件可以被直接输入而不是必须要创建一个.dzn文件，其中是数据文件里面的内容。

```
$ minizinc cakes2.mzn -D \
    "flour=4000;banana=6;sugar=2000;butter=500;cocoa=500;"
```

会给出和

```
$ minizinc cakes2.mzn pantry.dzn
```

一模一样的结果。

数据文件只能包含给模型中的决策变量和参数赋值的语句。

防御性编程建议我们应该检查数据文件中的数值是否合理。在我们的例子中，检查所有原料的份量是否是非负的并且若不正确则产生一个运行错误，这是明智的。提供了一个内置的布尔型操作符断言用来检查参数值。格式是`assert(B,S)`。布尔型表达式`B`被检测。若它是假的，运行中断。此时字符串表达式`S`作为错误信息被输出。如果我们想当面粉的份量是负值的时候去检测出并且产生合适的错误信息，我们可以直接加入下面的一行

```
constraint assert(flour >= 0,"Amount of flour is non-negative");
```

到我们的模型中。注意断言表达式是一个布尔型表达式，所以它被看做是一种类型的约束。我们可以加入类似的行来检测其他原料的份量是否是非负值。

2.1.4 实数求解

“” 列表

通过使用浮点数求解，也支持“实数”约束求解。考虑一个要在季度分期偿还的一年短期贷款问题。此问题的一个模型在列表中给出。它使用了一个简单的计算每季度结束后所欠款的利息计算方式。

列表确定一年借款每季度还款关系的模型loan.mzn

```
% 变量
var float: R;        % 季度还款
var float: P;        % 初始借贷本金
var 0.0 .. 10.0: I;  % 利率

% 中间变量
var float: B1; % 一个季度后的欠款
var float: B2; % 两个季度后的欠款
var float: B3; % 三个季度后的欠款
var float: B4; % 最后欠款

constraint B1 = P * (1.0 + I) - R;
constraint B2 = B1 * (1.0 + I) - R;
constraint B3 = B2 * (1.0 + I) - R;
constraint B4 = B3 * (1.0 + I) - R;

solve satisfy;

output [
 "Borrowing ", show_float(0, 2, P), " at ", show(I*100.0),
 "% interest, and repaying ", show_float(0, 2, R),
  "\nper quarter for 1 year leaves ", show_float(0, 2, B4), " owing\n"
```

```
];
```

注意我们声明了一个浮点型变量f，它和整型变量很类似。只是这里我们使用关键字float而不是int。

我们可以用同样的模型来回答一系列不同的问题。第一个问题是：如果我以利息借款并且每季度还款，我最终还欠款多少？这个问题在数据文件loan1.dzn中被编码。

由于我们希望用实数求解，我们需要使用一个可以支持这种问题类型的求解器。捆绑二进制发布预设的求解器支持浮点型变量一个混合整数线性求解器可能更加适合这种类型的问题。发布包含了这样的一个求解器。我们可以通过从求解器菜单按钮下面的三角形选择COIN-BC来使用或者在命令行中运行命令minizinc --solver cbc

```
$ minizinc --solver cbc loan.mzn loan1.dzn
```

输出是

```
Borrowing 1000.00 at 4.0% interest, and repaying 260.00
per quarter for 1 year leaves 65.78 owing
----------
```

第二个问题是如果我希望用的利息来借款并且在最后的时候一点都不欠款，我需要在每季度还款多少？这个问题在数据文件loan2.dzn中被编码。运行命令

```
$ minizinc --solver cbc loan.mzn loan2.dzn
```

后的输出是

```
Borrowing 1000.00 at 4.0% interest, and repaying 275.49
per quarter for 1 year leaves 0.00 owing
----------
```

第三个问题是如果我可以每个季度返还我可以用的利息来借款多少并且在最后的时候一点都不欠款？这个问题在数据文件loan3.dzn中被编码。运行命令

```
$ minizinc --solver cbc loan.mzn loan3.dzn
```

后的输出是

```
Borrowing 907.47 at 4.0% interest, and repaying 250.00
per quarter for 1 year leaves 0.00 owing
----------
```

列表loan.mzn的数据文件例子loan1.dzn

```
I = 0.04;
P = 1000.0;
R = 260.0;
```

列表loan.mzn的数据文件例子loan2.dzn

```
I = 0.04;
P = 1000.0;
B4 = 0.0;
```

列表loan.mzn的数据文件例子loan3.dzn

```
I = 0.04;
R = 250.0;
B4 = 0.0;
```

浮点算术操作符

提供了标准的浮点算术操作符加+减-乘*和浮点除/。同时也提供了一元操作符+和-。
不会自动地强制转换整数为浮点数。内建函数int2float被用来达到此目的。注意强制转换的一个后果是表达式a / b总是被认为是一个浮点除。如果你需要一个整数除请确定使用div操作符。
同时也包含浮点型函数来计算绝对值abs平方根sqrt自然对数ln底数为的对数log2底数为的对数log10的幂exp正弦sin余弦cos正切tan反正弦asin反余弦acos反正切atan双曲正弦sinh双曲余弦cosh双曲正切tanh双曲反正弦asinh双曲反余弦acosh双曲反正切atanh和唯一的二元函数次方pow，其余的都是一元函数。
算术常量的语法是相当标准的。浮点数常量的例子有1.05，1.3e-5和1.3E+5。

2.1.5 模型的基本结构

我们现在可以去总结模型的基本结构了。它由多个项组成，每一个在其最后都有一个分号;。项可以按照任何顺序出现。例如，标识符在被使用之前不需要被声明。

有八种类型的项。

引用项允许另外一个文件的内容被插入模型中。它们有以下形式：

```
include <文件名>;
```

其中<文件名>是一个字符串常量。它们使得大的模型可以被分为小的子模型以及包含库文件中定义的约束。我们会在列表中看到一个例子。

变量声明声明新的变量。这种变量是全局变量，可以在模型中的任何地方被提到。变量有两种。在模型中被赋一个固定值的参数变量以及只有在模型被求解的时候才会被赋值的决策变量。我们称参数是固定的，决策变量是不固定的。变量可以选择性地被赋一个值来作为声明的一部分。形式是：

```
<类型-实例化 表达式>: <变量> [ = ] <表达式>;
```

<类型-实例化 表达式>给了变量的类型和实例化。这些是比较复杂的其中一面。用par来实例化声明参数，用var来实例化声明决策变量。如果没有明确的实例化声明，则变量是一个参数。类型可以为基类型，一个整数或者浮点数范围，或者一个数组或集合。基类型有floatintstringboolann。其中只有floatintbool可以被决策变量使用。基类型ann是一个注解–我们会在搜索中讨论注解。整数范围表达式可以被用来代替类型int类似的，浮点数范围表达式可以被用来代替类型float。这些通常被用来定义一个整型决策变量的定义域，

但也可以被用来限制一个整型参数的范围。变量声明的另外一个用处是定义枚举类型—我们会在枚举类型中讨论。

赋值项给一个变量赋一个值。它们有以下形式：

```
<变量> = <表达式>;
```

数值可以被赋给决策变量。在这种情况下，赋值相当于加入constraint <变量> = <表达式>

约束项是模型的核心。它们有以下形式：

```
constraint <布尔型表达式>;
```

我们已经看到了使用算术比较的简单约束以及内建函数assert操作符。在下一节我们会看到更加复杂的约束例子。

求解项详细说明了到底要找哪种类型的解。正如我们看到的，它们有以下三种形式：

```
solve satisfy;
solve maximize <算术表达式>;
solve minimize <算术表达式>;
```

一个模型必须有且只有一个求解项。

输出项用来恰当的呈现模型运行后的结果。它们有下面的形式：

```
output [ <字符串表达式>, ..., <字符串表达式> ];
```

如果没有输出项，会默认输出所有没有被以赋值项的形式赋值的决策变量值。

枚举类型声明我们会在数组和集合和枚举类型中讨论。

谓词函数和测试项被用来定义新的约束，函数和布尔测试。我们会在谓词和函数中讨论。

注解项用来定义一个新的注解。我们会在搜索中讨论。

CHAPTER 2.2

更多复杂模型

在上一节中，我们介绍了模型的基本结构。在这一节中，我们介绍数组和集合数据结构，枚举类型，以及更加复杂的约束。

2.2.1 数组和集合

在绝大多数情况下，我们都是有兴趣建一个约束和变量的个数依赖于输入数据的模型。为了达到此目的，我们通常会使用数组。

考虑一个关于金属矩形板温度的简单有限元素模型。通过把矩形板在维的矩阵上分成有限个的元素，我们近似计算矩形板上的温度。一个模型在列表中给出。它声明了有限元素模型的宽w和高h。

声明

```
set of int: HEIGHT = 0..h;
set of int: CHEIGHT = 1..h-1;
set of int: WIDTH = 0..w;
set of int: CWIDTH = 1..w-1;
array[HEIGHT,WIDTH] of var float: t; % 在点 (i,j) 处的温度
```

声明了四个固定的整型集合来描述有限元素模型的尺寸：HEIGHT是整个模型的整体高度，而CHEIGHT是省略了顶部和底部的中心高度，WIDTH是模型的整体宽度，而CWIDTH是省略了左侧和右侧的中心宽度。最后，声明了一个浮点型变量组成的行编号从0到w，列编号从0到h的两维数组t用来表示金属板上每一点的温度。我们可以用表达式t[i,j]来得到数组中第i^{th}行和第j^{th}列的元素。

拉普拉斯方程规定当金属板达到一个稳定状态时，每一个内部点的温度是它的正交相邻点的平均值。约束

```
constraint forall(i in CHEIGHT, j in CWIDTH)(
              4.0*t[i,j] = t[i-1,j] + t[i,j-1] + t[i+1,j] + t[i,j+1]);
```

保证了每一个内部点(i, j)是它的四个正交相邻点的平均值。约束

```
% 边约束
constraint forall(i in CHEIGHT)(t[i,0] = left);
constraint forall(i in CHEIGHT)(t[i,w] = right);
constraint forall(j in CWIDTH)(t[0,j] = top);
constraint forall(j in CWIDTH)(t[h,j] = bottom);
```

限制了每一个边的温度必须是相等的，并且给了这些温度名字：left，right，top和bottom。而约束

```
% 角约束
constraint t[0,0]=0.0;
constraint t[0,w]=0.0;
constraint t[h,0]=0.0;
constraint t[h,w]=0.0;
```

确保了角的温度（这些是不相干的）被设置为。我们可以用列表中给出的模型来决定一个被分成×个元素的金属板的温度。其中左右下侧的温度为，上侧的温度为。

列表决定稳定状态温度的有限元平板模型laplace.mzn

```
int: w = 4;
int: h = 4;

% arraydec
set of int: HEIGHT = 0..h;
set of int: CHEIGHT = 1..h-1;
set of int: WIDTH = 0..w;
set of int: CWIDTH = 1..w-1;
array[HEIGHT,WIDTH] of var float: t; % 在点 (i,j) 处的温度
var float: left;   % 左侧温度
var float: right;  % 右侧温度
var float: top;    % 顶部温度
var float: bottom; % 底部温度

% 拉普拉斯方程：每一个内部点温度是它相邻点的平均值
constraint forall(i in CHEIGHT, j in CWIDTH)(
              4.0*t[i,j] = t[i-1,j] + t[i,j-1] + t[i+1,j] + t[i,j+1]);

% sides
% 边约束
constraint forall(i in CHEIGHT)(t[i,0] = left);
constraint forall(i in CHEIGHT)(t[i,w] = right);
constraint forall(j in CWIDTH)(t[0,j] = top);
constraint forall(j in CWIDTH)(t[h,j] = bottom);

% 角约束
constraint t[0,0]=0.0;
constraint t[0,w]=0.0;
constraint t[h,0]=0.0;
constraint t[h,w]=0.0;
left = 0.0;
right = 0.0;
top = 100.0;
bottom = 0.0;
```

```
solve satisfy;

output [ show_float(6, 2, t[i,j]) ++
         if j == w then "\n" else " " endif |
         i in HEIGHT, j in WIDTH
];
```

运行命令

```
$ minizinc --solver cbc laplace.mzn
```

得到输出

```
  0.00 100.00 100.00 100.00   0.00
  0.00  42.86  52.68  42.86   0.00
  0.00  18.75  25.00  18.75   0.00
  0.00   7.14   9.82   7.14   0.00
  0.00   0.00   0.00   0.00   0.00
----------
```

集合

集合变量用以下方式声明

```
set of <类型-实例化> : <变量名> ;
```

整型，枚举型（参见后面），浮点型和布尔型集合都可以定义。决策变量集合只可以是类型为整型或者枚举型的变量集合。集合常量有以下形式

```
{ <表达式-1>, ..., <表达式-n> }
```

或者是以下形式的整型，枚举型或浮点型范围表达式

```
<表达式-1> .. <表达式-2>
```

标准的集合操作符有：元素属于in非严格的集合包含subset非严格的超集关系superset并集union交集intersect集合差运算diff集合对称差symdiff和集合元素的个数card
我们已经看到集合变量和集合常量（包含范围）可以被用来作为变量声明时的隐式类型。在这种情况下变量拥有集合元素中的类型并且被隐式地约束为集合中的一个成员。

我们的烤蛋糕问题是一个非常简单的批量生产计划问题例子。在这类问题中，我们希望去决定每种类型的产品要制造多少来最大化利润。同时制造一个产品会消耗不同数量固定的资源。我们可以扩展列表中的模型为一个不限制资源和产品类型的模型去处理这种类型的问题。这个模型在列表中给出。一个（烤蛋糕问题的）数据文件例子在列表中给出。

列表简单批量生产计划模型prod-planning.mzn

```
% Products to be produced
enum Products;
% profit per unit for each product
```

```
array[Products] of int: profit;
% Resources to be used
enum Resources;
% amount of each resource available
array[Resources] of int: capacity;

% units of each resource required to produce 1 unit of product
array[Products, Resources] of int: consumption;
constraint assert(forall (r in Resources, p in Products)
          (consumption[p,r] >= 0), "Error: negative consumption");

% bound on number of Products
int: mproducts = max (p in Products)
                     (min (r in Resources where consumption[p,r] > 0)
                          (capacity[r] div consumption[p,r]));

% Variables: how much should we make of each product
array[Products] of var 0..mproducts: produce;
array[Resources] of var 0..max(capacity): used;

% Production cannot use more than the available Resources:
constraint forall (r in Resources) (
      used[r] = sum (p in Products)(consumption[p, r] * produce[p])
);
constraint forall (r in Resources) (
      used[r] <= capacity[r]
);

% Maximize profit
solve maximize sum (p in Products) (profit[p]*produce[p]);

output [ "\(p) = \(produce[p]);\n" | p in Products ] ++
       [ "\(r) = \(used[r]);\n" | r in Resources ];
```

列表简单批量生产计划模型的数据文件例子prod-planning-data.dzn

```
% Data file for simple production planning model
Products = { BananaCake, ChocolateCake };
profit = [400, 450]; % in cents

Resources = { Flour, Banana, Sugar, Butter, Cocoa };
capacity = [4000, 6, 2000, 500, 500];

consumption= [| 250, 2, 75, 100, 0,
              | 200, 0, 150, 150, 75 |];
```

这个模型的新特征是只用枚举类型枚举类型。这使得我们可以把资源和产品的选择作为模型的参数。模型的第一个项

```
enum Products;
```

声明Products为未知的产品集合。

枚举类型

枚举类型，我们称为enums用以下方式声明

```
enum <变量名> ;
```

一个枚举类型用以下赋值的方式定义

```
enum <变量名> = { <变量名-1>, ..., <变量名-n> } ;
```

其中<变量名-1>…<变量名-n>是名为<变量名>的枚举类型中的元素。通过这个定义，枚举类型中的每个元素也被有效地声明为这个类型的一个新的常量。声明和定义可以像往常一样结合为一行。

第二个项声明了一个整型数组：

```
array[Products] of int: profit;
```

profit数组的下标集合是Products。理想情况下，这种声明方式表明只有集合Products中的元素才能被用来做数组的下标。

有n个元素组成的枚举类型中的元素的行为方式和整数$1 \ldots n$的行为方式很像。它们可以被比较，它们可以按照它们出现在枚举类型定义中的顺序被排序，它们可以遍历，它们可以作为数组的下标，实际上，它们可以出现在一个整数可以出现的任何地方。

在数据文件例子中，我们用一列整数来初始化数组

```
Products = { BananaCake, ChocolateCake };
profit = [400,450];
```

意思是香蕉蛋糕的利润是，而巧克力蛋糕的利润是。在内部，BananaCake会被看成是像整数一样，而ChocolateCake会被看成像整数一样。虽然不提供明确的列表类型，但用1..n为下标集合的一维数组表现起来就像列表。我们有时候也会称它们为列表。

根据同样的方法，接下来的两项中我们声明了一个资源集合Resources，一个表明每种资源可获得量的数组capacity。

更有趣的是项

```
array[Products, Resources] of int: consumption;
```

声明了一个两维数组consumption。consumption[p,r]的值是制造一单位的产品p所需要的资源r的数量。其中第一个下标是行下标，而第二个下标是列下标。

数据文件包含了一个两维数组的初始化例子

```
consumption= [| 250, 2, 75,  100, 0,
              | 200, 0, 150, 150, 75 |];
```

注意分隔符|是怎样被用来分隔行的。

数组

因此，提供一维和多维数组。它们用以下类型来声明：

```
array [ <下标集合-1>, ..., <下标集合-n> ] of <类型-实例化>
```

要求数组声明要给出每一维的下标集合。下标集合或者是一个整型范围，一个被初始化为整型范围的集合变量，或者是一个枚举类型。数组可以是所有的基类型：整型，枚举型，布尔型，浮点型或者字符串型。这些可以是固定的或者不固定的，除了字符串型，它只可以是参数。数组也可以作用于集合但是不可以作用于数组。
一维数组常量有以下格式

```
[ <表达式-1>, ..., <表达式-n> ]
```

而二维数组常量有以下格式

```
[| <表达式-1-1>, ..., <表达式-1-n> |
   ...                             |
   <表达式-m-1>, ..., <表达式-m-n> |]
```

其中这个数组有m行n列。
内建函数array1darray2d等家族可以被用来从一个列表（或者更准确的说是一个一维数组）去实例化任何维度的数组。调用

```
array<n>d(<下标集合-1>, ..., <下标集合-n>, <列表>)
```

返回一个n维的数组，它的下标集合在前n个参数给出，最后一个参数包含了数组的元素。例如array2d(1..3, 1..2, [1, 2, 3, 4, 5, 6])和[|1, 2 |3, 4 |5, 6|]是相等的。
数组元素按照通常的方式获取获取：a[i,j]给出第i^{th}行第j^{th}列的元素。
串联操作符++可以被用来串联两个一维的数组。结果得到一个列表，即一个元素从索引的一维数组。例如[4000, 6] ++ [2000, 500, 500]求得[4000, 6, 2000, 500, 500]。内建函数length返回一维数组的长度。

模型的下一项定义了参数mproducts。它被设为可以生产出的任何类型产品的数量上限。这个确实是一个复杂的内嵌数组推导式和聚合操作符例子。在我们试图理解这些项和剩下的模型之前，我们应该先介绍一下它们。

首先，提供了在很多函数式编程语言都提供的列表推导式。例如，列表推导式[i + j | i, j in 1..3 where j < i]算得[2 + 1, 3 + 1, 3 + 2]等同于[3, 4, 5]。[3, 4, 5]只是一个下标集合为1..3的数组。

同时也提供了集合推导式，它有类似的语法：例如{i + j | i, j in 1..3 where j < i}计算得到集合{3, 4, 5}。

列表和集合推导式

列表推导式的一般格式是

```
[ <表达式> | <生成元表达式> ]
```

<表达式>指明了如何从<生成元表达式>产生的元素输出列表中创建元素。生成元<generator-exp>由逗号分开的一列生成元表达式组成，选择性地跟着一个布尔型表达式。两种格式是

```
<生成元>, ..., <生成元>
<生成元>, ..., <生成元> where <布尔表达式>
```

第二种格式中的可选择的<布尔型表达式>被用作生成元表达式的过滤器：只有满足布尔型表达式的输出列表中的元素才被用来构建元素。生成元<generator>有以下格式

```
<标识符>, ..., <标识符> in <数组表达式>
```

每一个标识符是一个迭代器，轮流从数值表达式中取值，最后一个标识符变化的最迅速。列表推导式的生成元和<布尔型表达式>通常不会涉及决策变量。如果它们确实涉及了决策变量，那么产生的列表是一列var opt <T>，其中是<表达式>的类型。更多细节，请参考选项类型中有关选项类型的论述。
集合推导式几乎和列表推导式一样：唯一的不同是这里使用{和}括住表达式而不是[和]。集合推导式生成的元素必须是固定的，即不能是决策变量。类似的，集合推导式的生成元和可选择的<布尔型表达式>必须是固定的。

第二，提供了一系列的可以把一维数组的元素聚合起来的内建函数。它们中最有用的可能是forall。它接收一个布尔型表达式数组（即，约束），返回单个布尔型表达式，它是对数组中的布尔型表达式的逻辑合取。

例如，以下表达式

```
forall( [a[i] != a[j] | i,j in 1..3 where i < j])
```

其中a是一个下标集合为1..3的算术数组。它约束了a中的元素是互相不相同的。列表推导式计算得到[a[1] != a[2], a[1] != a[3], a[2] != a[3]]，所以forall函数返回逻辑合取a[1] != a[2] /\ a[1] != a[3] /\ a[2] != a[3]。

聚合函数

算术数组的聚合函数有sum把元素加起来，product把元素乘起来，和min跟max各自返回数组中的最小和最大元素。当作用于一个空的数组时，min和max返回一个运行错误，sum返回，product返回。
为数组提供了包含有布尔型表达式的四个聚合函数。正如我们看到的，它们中的第一个是forall，它返回一个等于多个约束的逻辑合取的单个约束。第二个函数，exists，返回多个约束的逻辑析取。因此forall强制数组中的所有约束都满足，而exists确保至少有一个约束满足。第三个函数，xorall确保奇数个约束满足。第四个函数，iffall确保偶数个约束满足。

第三个，也是难点的最后一个部分是当使用数组推导式时，允许使用一个特别的聚合函数的语法。建模者不仅仅可以用

```
forall( [a[i] != a[j] | i,j in 1..3 where i < j])
```

也可以用一个更加数学的表示

```
forall (i,j in 1..3 where i < j) (a[i] != a[j])
```

两种表达方式是完全相等的：建模者可以自由使用任何一个他们认为更自然的表达方式。

生成表达式

一个生成表达式有以下格式

```
<聚合函数> ( <生成元表达式> ) ( <表达式> )
```

圆括号内的<生成元表达式>以及构造表达式<表达式>是非选择性的：它们必须存在。它等同于

```
<聚合函数> ( [ <表达式> | <生成元表达式> ] )
```

<聚合函数>可以是的任何由单个数组作为其参数的函数。

接下来我们就来了解列表中的简单批量生产计划模型剩余的部分。现在请暂时忽略定义mproducts的这部分。接下来的项：

```
array[Products] of var 0..mproducts: produce;
```

定义了一个一维的决策变量数组produce。produce[p]的值代表了最优解中产品p的数量。下一项

```
array[Resources] of var 0..max(capacity): used;
```

定义了一个辅助变量集合来记录每一种资源的使用量。下面的两个约束

```
constraint forall (r in Resources)
           (used[r] = sum (p in Products) (consumption[p, r] * produce[p]));
constraint forall (r in Resources)(used[r] <= capacity[r] );
```

使用used[r]计算资源r的总体消耗以及保证它是少于可获得的资源r的量。最后，项

```
solve maximize sum (p in Products) (profit[p]*produce[p]);
```

表明这是一个最大化问题以及最大化的目标是全部利润。

现在我们回到mproducts的定义。对每个产品p，表达式

```
(min (r in Resources where consumption[p,r] > 0)
                                (capacity[r] div consumption[p,r])
```

决定了在考虑了每种资源r的数量以及制造产品p需要的r量的情况下，p可以生产的最大量。注意过滤器where consumption[p,r] > 0的使用保证了只有此产品需要的资源才会考虑，因此避免了出现除数为零的错误。所以，完整的表达式

```
int: mproducts = max (p in Products)
                     (min (r in Resources where consumption[p,r] > 0)
                                (capacity[r] div consumption[p,r]));
```

计算任何产品可以被制造的最大量，因此它可以被作为produce中的决策变量定义域的上限。

最后，注意输出项是比较复杂的，并且使用了列表推导式列表推导式去创建一个易于理解的输出。运行

```
$ minizinc --solver gecode prod-planning.mzn prod-planning-data.dzn
```

输出得到如下结果

```
BananaCake = 2;
ChocolateCake = 2;
Flour = 900;
Banana = 4;
Sugar = 450;
Butter = 500;
Cocoa = 150;
----------
==========
```

2.2.2 全局约束

包含了一个全局约束的库，这些全局约束也可以被用来定义模型。一个例子是alldifferent约束，它要求所有参数中的变量都必须是互相不相等的。

列表算式谜题模型send-more-money.mzn

```
include "alldifferent.mzn";

var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;

constraint           1000 * S + 100 * E + 10 * N + D
                   + 1000 * M + 100 * O + 10 * R + E
       = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;

constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;

output ["   \(S)\(E)\(N)\(D)\n",
```

```
        "+  \(M)\(O)\(R)\(E)\n",
        "= \(M)\(O)\(N)\(E)\(Y)\n"];
```

问题要求给每一个字母赋不同的数值使得此算术约束满足。列表中的模型使用alldifferent([S,E,N,D,M,O,R,Y])约束表达式来保证每个字母有不同的数字值。在使用了引用项

```
include "alldifferent.mzn";
```

后，此全局约束alldifferent可以在模型中使用。我们可以用以下代替此行

```
include "globals.mzn";
```

它包含了所有的全局约束。

一系列所有在中定义了的全局约束都被包含在了发布的文档中。对一些重要的全局约束的描述，请参见全局约束。

2.2.3 条件表达式

提供了一个条件表达式。它的一个使用例子如下

```
int: r = if y != 0 then x div y else 0 endif;
```

若y不是零，则r设为x除以y，否则则设为零。

条件表达式

条件表达式的格式是

```
if <布尔型表达式> then <表达式-1> else <表达式-2> endif
```

它是一个真表达式而不是一个控制流语句，所以它可以被用于其他表达式中。如果<布尔型表达式>是真，则它取值<表达式-1>，否则则是<表达式-2>。条件表达式的类型是<表达式-1>和<表达式-2>的类型，而它们俩必须有相同的类型。
如果<布尔型表达式>包含决策变量，则表达式的类型实例化是var <T>，其中<T>是<表达式-1>和<表达式-2>的类型，就算是在两个表达式都已经固定了的情况下也是如此。

列表广义数独问题的模型sudoku.mzn

```
include "alldifferent.mzn";

int: S;
int: N = S * S;
int: digs = ceil(log(10.0,int2float(N))); % 输出的数字

set of int: PuzzleRange = 1..N;
set of int: SubSquareRange = 1..S;

array[1..N,1..N] of 0..N: start; %% 板初始0 = 空
array[1..N,1..N] of var PuzzleRange: puzzle;
```

```
% 填充初始板
constraint forall(i,j in PuzzleRange)(
    if start[i,j] > 0 then puzzle[i,j] = start[i,j] else true endif );

% 每行中取值各不相同
constraint forall (i in PuzzleRange) (
                   alldifferent( [ puzzle[i,j] | j in PuzzleRange ]) );

% 每列中取值各不相同
constraint forall (j in PuzzleRange) (
                   alldifferent( [ puzzle[i,j] | i in PuzzleRange ]) );

% 每个子方格块中取值各不相同
constraint
        forall (a, o in SubSquareRange)(
                alldifferent( [ puzzle[(a-1) *S + a1, (o-1)*S + o1] |
                                        a1, o1 in SubSquareRange ] ) );

solve satisfy;

output  [ show_int(digs,puzzle[i,j]) ++ " " ++
          if j mod S == 0 then " " else "" endif ++
          if j == N then
              if i != N then
                 if i mod S == 0 then "\n\n" else "\n" endif
          else "" endif else "" endif
          | i,j in PuzzleRange ] ++ ["\n"];
```

列表广义数独问题的数据文件例子sudoku.dzn

```
S=3;
start=[|
0, 0, 0, 0, 0, 0, 0, 0, 0|
0, 6, 8, 4, 0, 1, 0, 7, 0|
0, 0, 0, 0, 8, 5, 0, 3, 0|
0, 2, 6, 8, 0, 9, 0, 4, 0|
0, 0, 7, 0, 0, 0, 9, 0, 0|
0, 5, 0, 1, 0, 6, 3, 2, 0|
0, 4, 0, 6, 1, 0, 0, 0, 0|
0, 3, 0, 2, 0, 7, 6, 9, 0|
0, 0, 0, 0, 0, 0, 0, 0, 0|];
```

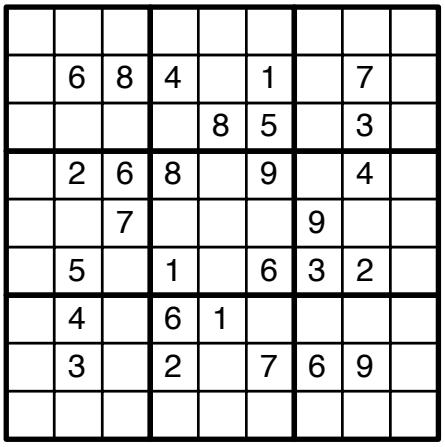
在创建复杂模型或者复杂输出时，条件表达式是非常有用的。我们来看下列表中的数独问题模型。板的初始位置在参数start中给出，其中代表了一个空的板位置。通过使用以下条件表达式

```
constraint forall(i,j in PuzzleRange)(
     if start[i,j] > 0 then puzzle[i,j] = start[i,j] else true endif );
```

它被转换为对决策变量puzzle的约束。

在定义复杂输出时，条件表达式也很有用。在数独模型列表中，表达式

```
if j mod S == 0 then " " else "" endif
```



图sudoku.dzn代表的问题。

在大小为S的组群之间插入了一个额外的空格。输出表达式同时也使用条件表达式来在每S行后面加入一个空白行。这样得到的输出有很高的可读性。

剩下的约束保证了每行中，每列中以及每$S \times S$子方格块中的值都是互相不相同的。

通过使用标示-a或--all-solutions，我们可以用求解得到一个满足问题solve satisfy的所有解。运行

```
$ minizinc --all-solutions sudoku.mzn sudoku.dzn
```

得到

```
 5 9 3  7 6 2  8 1 4
 2 6 8  4 3 1  5 7 9
 7 1 4  9 8 5  2 3 6

 3 2 6  8 5 9  1 4 7
 1 8 7  3 2 4  9 6 5
 4 5 9  1 7 6  3 2 8

 9 4 2  6 1 8  7 5 3
 8 3 5  2 4 7  6 9 1
 6 7 1  5 9 3  4 8 2
----------
==========
```

当系统输出完所有可能的解之后，此行==========被输出。在这里则表明了此问题只有一个解。

2.2.4 枚举类型

枚举类型允许我们根据一个或者是数据中的一部分，或者在模型中被命名的对象集合来创建模型。这样一来，模型就更容易被理解和调试。我们之前已经简单介绍了枚举类型或者。在这一小分段，我们会探索如何可以全面地使用它们，并且给出一些处理枚举类型的内建函数。

让我们重新回顾一下基本模型中的给澳大利亚涂色问题。

列表使用枚举类型的澳大利亚涂色模型aust-enum.mzn

```
enum Color;
var Color: wa;
var Color: nt;
var Color: sa;
var Color: q;
var Color: nsw;
var Color: v;
var Color: t;
constraint wa != nt /\ wa != sa /\ nt != sa /\ nt != q /\ sa != q;
constraint sa != nsw /\ sa != v /\ q != nsw /\ nsw != v;
solve satisfy;
```

列表中的模型声明了一个枚举类型Color，而它必须在数据文件中被定义。每一个州变量被声明为从此枚举类型中取一个值。使用以下方式运行这个程序

```
$ minizinc -D"Color = { red, yellow, blue };" aust-enum.mzn
```

可能会得到输出

```
wa = yellow;
nt = blue;
sa = red;
q = yellow;
nsw = blue;
v = yellow;
t = red;
```

枚举类型变量声明

一个枚举类型参数变量被声明为以下两种方式：

```
<枚举名> : <变量名>
<l>..<u> : <变量名>
```

其中<枚举名>是枚举类型的名字，<l>和<u>是此枚举类型的固定枚举类型表达式。

枚举类型一个重要的行为是，当它们出现的位置所期望的是整数时，它们会自动地强制转换为整数。这样一来，这就允许我们使用定义在整数上的全局变量，例如

```
global_cardinality_low_up([wa,nt,sa,q,nsw,v,t],
                          [red,yellow,blue],[2,2,2],[2,2,3]);
```

要求每种颜色至少有两个州涂上并且有三个州被涂了蓝色。

枚举类型操作符

有一系列关于枚举类型的内部操作符：

`enum_next(X,x)`返回枚举类型`X`中`x`后的下一个值。这是一个部份函数如果`x`是枚举类型`X`最后一个值则函数会返回⊥令包含这个表达式的布尔表达式返回`false`。

`enum_prev(X,x)``enum_prev(X,x)`返回枚举类型`X`中`x`的上一个值。`enum_prev`同样是一个部份函数。

`to_enum(X,i)`映射一个整型表达式`i`到一个在`X`的枚举类型值或者如果`i`是小于等于或大于中元素的个数则返回。

注意，一些标准函数也是可以应用于枚举类型上

`card(X)`返回枚举类型`X`的势。

`min(X)`返回枚举类型`X`中最小的元素。

`max(X)`返回枚举类型`X`中最大的元素。

2.2.5 复杂约束

约束是模型的核心。我们已经看到了简单关系表达式，但是约束其实是比这更加强大的。一个约束可以是任何布尔型表达式。想象一个包含两个时间上不能重叠的任务的调度问题。如果`s1`和`s2`是相对应的起始时间，`d1`和`d2`是相对应的持续时间，我们可以表达约束为：

```
constraint s1 + d1 <= s2  \/ s2 + d2 <= s1;
```

来保证任务之间互相不会重叠。

布尔型

中的布尔型表达式可以按照标准的数学语法来书写。布尔常量是`真`或`假`，布尔型操作符有合取，即，与`/\`，析取，即，或`\/`，必要条件蕴含`<-`，充分条件蕴含`->`，充分必要蕴含`<->`以及非`not`。内建函数`bool2int`强制转换布尔型为整型：如果参数为真，它返回，否则返回。

列表车间作业调度问题模型`jobshop.mzn`

```
enum JOB;
enum TASK;
TASK: last = max(TASK);
array [JOB,TASK] of int: d;                    % 任务持续时间
int: total = sum(i in JOB, j in TASK)(d[i,j]);% 总持续时间
int: digs = ceil(log(10.0,int2float(total))); % 输出的数值
array [JOB,TASK] of var 0..total: s;           % 起始时间
var 0..total: end;                             % 总结束时间


constraint %% 保证任务按照顺序出现
    forall(i in JOB) (
        forall(j in TASK where j < last)
            (s[i,j] + d[i,j] <= s[i,enum_next(TASK,j)]) /\
        s[i,last] + d[i,last] <= end
    );

constraint %% 保证任务之间没有重叠
    forall(j in TASK) (
        forall(i,k in JOB where i < k) (
            s[i,j] + d[i,j] <= s[k,j] \/
```

```
            s[k,j] + d[k,j] <= s[i,j]
        )
    );

solve minimize end;

output ["end = \(end)\n"] ++
       [ show_int(digs,s[i,j]) ++ " " ++
         if j == last then "\n" else "" endif |
         i in JOB, j in TASK ];
```

列表车间作业调度问题数据jdata.dzn

```
JOB = _(1..5);
TASK = _(1..5);
d = [| 1, 4, 5, 3, 6
     | 3, 2, 7, 1, 2
     | 4, 4, 4, 4, 4
     | 1, 1, 1, 6, 8
     | 7, 3, 2, 2, 1 |];
```

列表中的车间作业调度模型给出了一个使用析取建模功能的现实例子。车间作业调度问题中，我们有一个作业集合，每一个包含一系列的在不同机器上的任务：任务[i,j]是在第i^{th}个作业中运行在第j^{th}个机器上的任务。每列任务必须按照顺序完成，并且运行在同一个机器上的任何两个任务在时间上都不能重叠。就算是对这个问题的小的实例找最优解都会是很有挑战性的。

命令

```
$ minizinc --all-solutions jobshop.mzn jdata.dzn
```

求解了一个小的车间作业调度问题，并且显示了优化问题在all-solutions下的表现。在这里，求解器只有当找到一个更好的解时才会输出它，而不是输出所有的可能最优解。这个命令下的（部分）输出是：

```
end = 39
 5  9 13 22 30
 6 13 18 25 36
 0  4  8 12 16
 4  8 12 16 22
 9 16 25 27 38
----------
end = 37
 4  8 12 17 20
 5 13 18 26 34
 0  4  8 12 16
 8 12 17 20 26
 9 16 25 27 36
----------
end = 34
 0  1  5 10 13
 6 10 15 23 31
 2  6 11 19 27
 1  5 10 13 19
 9 16 22 24 33
```

```
----------
end = 30
 5  9 13 18 21
 6 13 18 25 27
 1  5  9 13 17
 0  1  2  3  9
 9 16 25 27 29
----------
==========
```

表明一个结束时间为的最优解终于被找到，并且被证明为是最优的。通过加一个约束`end = 30`，并且把求解项改为`solve satisfy`，然后运行

```
$ minizinc --all-solutions jobshop.mzn jobshop.dzn
```

我们可以得到所有的最优解。这个问题有个最优解。

列表稳定婚姻问题模型stable-marriage.mzn

```
int: n;

enum Men = _(1..n);
enum Women = _(1..n);

array[Women, Men] of int: rankWomen;
array[Men, Women] of int: rankMen;

array[Men] of var Women: wife;
array[Women] of var Men: husband;

% assignment
constraint forall (m in Men) (husband[wife[m]]=m);
constraint forall (w in Women) (wife[husband[w]]=w);
% ranking
constraint forall (m in Men, o in Women) (
     rankMen[m,o] < rankMen[m,wife[m]] ->
         rankWomen[o,husband[o]] < rankWomen[o,m] );

constraint forall (w in Women, o in Men) (
     rankWomen[w,o] < rankWomen[w,husband[w]] ->
         rankMen[o,wife[o]] < rankMen[o,w] );
solve satisfy;

output ["wives= \(wife)\nhusbands= \(husband)\n"];
```

列表稳定婚姻问题模型的数据文件例子。stable-marriage.dzn

```
n = 5;
rankWomen =
 [| 1, 2, 4, 3, 5,
  | 3, 5, 1, 2, 4,
  | 5, 4, 2, 1, 3,
  | 1, 3, 5, 4, 2,
  | 4, 2, 3, 5, 1 |];
```

```
rankMen =
 [| 5, 1, 2, 4, 3,
  | 4, 1, 3, 2, 5,
  | 5, 3, 2, 4, 1,
  | 1, 5, 4, 3, 2,
  | 4, 3, 2, 1, 5 |];
```

中的另外一个强大的建模特征是决策变量可以被用来访问数组。作为一个例子，考虑（老式的）稳定婚姻问题。我们有n个（直）女以及n个（直）男。每一个男士有一个女士排行榜，女士也是。我们想给每一个女士男士找一个丈夫妻子来使得所有的婚姻按以下意义上来说都是稳定的：

每当m喜欢另外一个女士o多过他的妻子w时，o喜欢她的丈夫多过m，以及

每当w喜欢另外一个男士o多过她的丈夫m时，o喜欢他的妻子多过w。

这个问题可以很优雅地在中建模。模型和数据例子在列表和列表中分别被给出。

模型中的前三项声明了男士女士的数量以及男士和女士的集合。在这里我们介绍匿名枚举类型的使用。Men和Women都是大小为n的集合，但是我们不希望把它们混合到一起，所以我们使用了一个匿名枚举类型。这就允许检测到使用Men为Women或者反之的建模错误。

矩阵rankWomen和rankMen分别给出了男士们的女士排行以及女士们的男士排行。因此，项rankWomen[w,m]给出了女士的关于男士的排行。在排行中的数目越小，此男士或者女士被选择的倾向越大。

有两个决策变量的数组：wife和husband。这两个分别代表了每个男士的妻子和每个女士的丈夫。

前两个约束

```
constraint forall (m in Men) (husband[wife[m]]=m);
constraint forall (w in Women) (wife[husband[w]]=w);
```

确保了丈夫和妻子的分配是一致的：w是m的妻子蕴含了m是w的丈夫，反之亦然。注意在husband[wife[m]]中，下标表达式wife[m]是一个决策变量，而不是一个参数。

接下来的两个约束是稳定条件的直接编码：

```
constraint forall (m in Men, o in Women) (
     rankMen[m,o] < rankMen[m,wife[m]] ->
         rankWomen[o,husband[o]] < rankWomen[o,m] );

constraint forall (w in Women, o in Men) (
     rankWomen[w,o] < rankWomen[w,husband[w]] ->
         rankMen[o,wife[o]] < rankMen[o,w] );
```

在有了用决策变量作为数组的下标和用标准的布尔型连接符构建约束的功能后，稳定婚姻问题的自然建模才变得可行。敏锐的读者可能会在这时产生疑问，如果数组下标变量取了一个超出数组下标集合的值，会产生什么情况。把这种情况看做失败：一个数组访问a[e]在其周围最近的布尔型语境中隐含地加入了约束e in index_set(a)，其中index_set(a)给出了a的下标集合。

匿名枚举类型

一个匿名枚举类型表达式有格式`anon_enum(<n>)`，其中`<n>`是一个固定的整型表达式，它定义了枚举类型的大小。
除了其中的元素没有名字，匿名枚举类型和其他的枚举类型一样。当被输出时，它们根据枚举类型的名字被给定独有的名字。

例如，如下的变量声明

```
array[1..2] of int: a= [2,3];
var 0..2: x;
var 2..3: y;
```

约束`a[x] = y`会在$x = 1 \wedge y = 2$和$x = 2 \wedge y = 3$时得到满足。约束`not a[x] = y`会在$x = 0 \wedge y = 2$，$x = 0 \wedge y = 3 x = 1 \wedge y = 3$和$x = 2 \wedge y = 2$时得到满足。

当参数无效访问数组时，正式的语义会把此情况看成失败来确保参数和决策变量的处理方式是一致的，但是会发出警告，因为这种情况下几乎总是会有错误出现。

列表魔术串问题模型`magic-series.mzn`

```
int: n;
array[0..n-1] of var 0..n: s;

constraint forall(i in 0..n-1) (
   s[i] = (sum(j in 0..n-1)(bool2int(s[j]=i))));

solve satisfy;

output [ "s = \(s);\n" ] ;
```

强制转换函数`bool2int`可以被任何布尔型表达式调用。这就使得建模者可以使用所谓的高价约束。举个简单的例子，请看魔术串问题：找到一列数字$s = [s_0, \ldots, s_{n-1}]$使得是数字$i$出现在$s$的次数。一个解的例子是$s = [1, 2, 1, 0]$。

这个问题的一个模型在列表中给出。列表的使用使得我们可以把函数`s[j]=i`满足的次数加起来。运行命令

```
$ minizinc --all-solutions magic-series.mzn -D "n=4;"
```

得到输出

```
s = [1, 2, 1, 0];
----------
s = [2, 0, 2, 0];
----------
==========
```

确切地显示出这个问题的两个解。

注意当有需要的时候，会自动地强制转换布尔型为整型以及整型为浮点型。我们可以把列表中的约束项替换为

```
constraint forall(i in 0..n-1) (
   s[i] = (sum(j in 0..n-1)(s[j]=i)));
```

由于系统实际上会自动地加入缺失的bool2int，布尔型表达式s[j] = i会被自动地强制转换为整型，所以会得到同样的结果。

强制转换

中，通过使用函数bool2int，我们可以把一个布尔型数值强制转换为一个整型数值。同样地，通过使用函数int2float，我们也可以把一个整型数值强制转换为一个浮点型数值。被强制转换的数值的实例化和原数值一样。例如，par bool被强制转换为par int，而var bool被强制转换为var int。
通过适当地在模型中加入bool2int和int2float，会自动地强制转换布尔型表达式为整型表达式，以及整型表达式为浮点型表达式。注意通过两步转换，它也会强制转换布尔型为浮点型。

2.2.6 集合约束

另外一个强大的建模特征是它允许包含整数的集合是决策变量：这表示当模型被评估时，求解器会查找哪些元素在集合中。

举个简单的例子，背包问题。这个问题是背包问题的局限版本，即我们或者选择把物品放入背包或者不放。每一个物品有一个重量和一个利润，在受限制于背包不能太满的条件下，我们想找到选取哪些物品会得到最大化利润。

很自然地，我们在中使用单个的决策变量来建模：var set of ITEM: knapsack其中ITEM是可放置的物品集合。如果数组weight[i]和profit[i]分别给出物品i的重量和利润，以及背包可以装载的最大重量是capacity，则一个自然的模型在列表中给出。

列表背包问题模型knapsack.mzn

```
enum ITEM;
int: capacity;

array[ITEM] of int: profits;
array[ITEM] of int: weights;

var set of ITEM: knapsack;

constraint sum (i in knapsack) (weights[i]) <= capacity;

solve maximize sum (i in knapsack) (profits[i]) ;

output ["knapsack = \(knapsack)\n"];
```

注意，关键字var出现在set声明之前，表明这个集合本身是决策变量。这就和一个var关键字描述其中元素而不是数组自身的数组形成对比，因为此时数组的基本结构，即它的下标集合，是固定了的。

列表高尔夫联谊问题模型social-golfers.mzn

```
include "partition_set.mzn";
int: weeks;    set of int: WEEK = 1..weeks;
int: groups;   set of int: GROUP = 1..groups;
```

```
int: size;     set of int: SIZE = 1..size;
int: ngolfers = groups*size;
set of int: GOLFER = 1..ngolfers;

array[WEEK,GROUP] of var set of GOLFER: Sched;

% constraints
constraint
    forall (i in WEEK, j in GROUP) (
           card(Sched[i,j]) = size
        /\ forall (k in j+1..groups) (
                Sched[i,j] intersect Sched[i,k] = {}
           )
    ) /\
    forall (i in WEEK) (
          partition_set([Sched[i,j] | j in GROUP], GOLFER)
    ) /\
    forall (i in 1..weeks-1, j in i+1..weeks) (
        forall (x,y in GROUP) (
            card(Sched[i,x] intersect Sched[j,y]) <= 1
        )
    );
% symmetry
  constraint
    % Fix the first week %
    forall (i in GROUP, j in SIZE) (
        ((i-1)*size + j) in Sched[1,i]
    ) /\
    % Fix first group of second week %
    forall (i in SIZE) (
        ((i-1)*size + 1) in Sched[2,1]
    ) /\
    % Fix first 'size' players
    forall (w in 2..weeks, p in SIZE) (
        p in Sched[w,p]
    );

solve satisfy;

output [ show(Sched[i,j]) ++ " " ++
         if j == groups then "\n" else "" endif |
         i in WEEK, j in GROUP ];
```

列表高尔夫联谊问题数据social-golfers.dzn

```
weeks = 4;
groups = 4;
size =3 ;
```

我们来看一个更复杂的关于集合约束的例子，列表中给出的高尔夫联谊问题。这个问题的目的是给groups×size个高尔夫手在weeks时间内安排一个高尔夫联赛。每一周我们需要安排groups个大小为size的不同的组。任何一对高尔夫手都不能一起出现于两个组中进行比赛。

模型中的变量是第i^{th}周第j^{th}组的高尔夫手组成的集合。

行中的约束首先对每一周的第一个集合进行一个排序来去除掉周之间可以互相调换的对称。然

后它对每一周内的集合进行了一个排序，同时使得每一个集合的势为size。接下来通过使用全局约束partition_set，确保了每一周都是对高尔夫手集合的一个划分。最后一个约束确保了任何两个高尔夫手都不会一起在两个组内比赛（因为任何两个组的交集的势最多都是）。

我们也有

在行中，我们也给出了去对称初始化约束：第一周被固定为所有的高尔夫手都按顺序排列；第二周的第一组被规定为是由第一周的前几组的第一个选手组成；最后，对于剩下的周，模型规定第一个size内的高尔夫手们出现在他们相对应的组数中。

运行命令

```
$ minizinc social-golfers.mzn social-golfers.dzn
```

其中数据文件定义了一个周数为，大小为，组数为的问题，得到如下结果

```
1..3 4..6 7..9 10..12
{ 1, 4, 7 } { 2, 5, 10 } { 3, 9, 11 } { 6, 8, 12 }
{ 1, 5, 8 } { 2, 6, 11 } { 3, 7, 12 } { 4, 9, 10 }
{ 1, 6, 9 } { 2, 4, 12 } { 3, 8, 10 } { 5, 7, 11 }
----------
```

注意范围集合是如何以范围格式输出的。

2.2.7 汇总

我们以一个可以阐释这一章介绍的大部分特征的复杂例子来结束这一节，包括枚举类型，复杂约束，全局约束以及复杂输出。

列表使用枚举类型规划婚礼座位wedding.mzn

```
enum Guests = { bride, groom, bestman, bridesmaid, bob, carol,
  ted, alice, ron, rona, ed, clara};
set of int: Seats = 1..12;
set of int: Hatreds = 1..5;
array[Hatreds] of Guests: h1 = [groom, carol, ed, bride, ted];
array[Hatreds] of Guests: h2 = [clara, bestman, ted, alice, ron];
set of Guests: Males = {groom, bestman, bob, ted, ron,ed};
set of Guests: Females = {bride,bridesmaid,carol,alice,rona,clara};

array[Guests] of var Seats: pos; % 客人的座位
array[Hatreds] of var Seats: p1;  % 互相憎恶的客人1的座位
array[Hatreds] of var Seats: p2;  % 互相憎恶的客人2的座位
array[Hatreds] of var 0..1: sameside;  % 互相憎恶的客人是否坐在同一边
array[Hatreds] of var Seats: cost;  % 互相憎恶的客人的距离

include "alldifferent.mzn";
constraint alldifferent(pos);
constraint forall(g in Males)( pos[g] mod 2 == 1 );
constraint forall(g in Females)( pos[g] mod 2 == 0 );
constraint not (pos[ed] in {1,6,7,12});
constraint abs(pos[bride] - pos[groom]) <= 1 /\
           (pos[bride] <= 6 <-> pos[groom] <= 6);
constraint forall(h in Hatreds)(
               p1[h] = pos[h1[h]] /\
               p2[h] = pos[h2[h]] /\
```

```
                sameside[h] = bool2int(p1[h] <= 6 <-> p2[h] <= 6) /\
                cost[h] = sameside[h] * abs(p1[h] - p2[h]) +
                   (1 - sameside[h]) * (abs(13 - p1[h] - p2[h]) + 1) );

solve maximize sum(h in Hatreds)(cost[h]);

output [ show(g)++" " | s in Seats,g in Guests where fix(pos[g]) == s]
       ++ ["\n"];
```

列表中的模型安排婚礼桌上的座位。这个桌子有个编码的顺序排列的座位，每边有个。男士必须坐奇数号码的座位，女士坐偶数。由于恐惧症不能坐在桌子的边缘，新郎和新娘必须坐在彼此旁边。我们的目的是最大化已知的互相憎恶的人之间的距离。如果在同一边，座位之间的距离是座位号码之间的差，否则则是和其对面座位的距离。

注意在输出语句中我们观察每个座位s来找一个客人g分配给此座位。我们利用内建函数fix，它检查一个决策变量是否是固定的以及输出它的固定值，否则的话中断。在输出语句中使用此函数总是安全的，因为当输出语句被运行的时候，所有的决策变量都应该是固定了的。

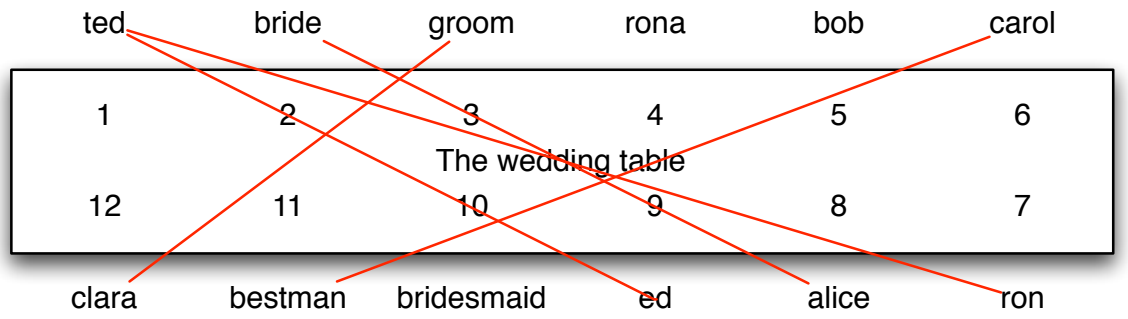
运行

```
$ minizinc wedding.mzn
```

得到输出

```
ted bride groom rona ed carol ron alice bob bridesmaid bestman clara
----------
==========
```

最终得到的座位安排在图中给出。其中连线表示互相憎恶，总的距离是



图婚礼桌上座位的安排

固定

输出项中，内建函数fix检查一个决策变量的值是否固定，然后把决策变量的实例化强制转换为参数。

CHAPTER 2.3

谓词和函数

中的谓词允许我们用简洁的方法来表达模型中的复杂约束。中的谓词利用预先定义好的全局约束建模，同时也让建模者获取以及定义新的复杂约束。中的函数用来捕捉模型中的共同结构。实际上，一个谓词就是一个输出类型为`var bool`的函数。

2.3.1 全局约束

中定义了很多可以在建模中使用的全局约束。由于全局约束的列表一直在慢慢增加，最终确定的列表可以在发布的文档中找到。下面我们讨论一些最重要的全局约束。

2.3.1.1 Alldifferent

约束`alldifferent`的输入为一个变量数组，它约束了这些变量取不同的值。

`alldifferent`的使用有以下格式

```
alldifferent(array[int] of var int: x)
```

即，参数是一个整型变量数组。

`alldifferent`是约束规划中被最多研究以及使用的全局约束之一。它被用来定义分配子问题，人们也给出了`alldifferent`的高效全局传播器。`send-more-money.mzn`列表和`sudoku.mzn`列表是使用`alldifferent`的模型例子。

2.3.1.2 Cumulative

约束`cumulative`被用来描述资源累积使用情况。

```
cumulative(array[int] of var int: s, array[int] of var int: d,
           array[int] of var int: r, var int: b)
```

规定对于一个起始时间为`s`，持续时间为`d`以及资源需求量为`r`的任务集合，在任何时间对资源的需求量都不能超过一个全局资源量界限`b`。

列表使用cumulative来建模搬运家具问题的模型moving.mzn

```
include "cumulative.mzn";

enum OBJECTS;
array[OBJECTS] of int: duration; % duration to move
array[OBJECTS] of int: handlers; % number of handlers required
array[OBJECTS] of int: trolleys; % number of trolleys required

int: available_handlers;
int: available_trolleys;
int: available_time;

array[OBJECTS] of var 0..available_time: start;
var 0..available_time: end;

constraint cumulative(start, duration, handlers, available_handlers);
constraint cumulative(start, duration, trolleys, available_trolleys);

constraint forall(o in OBJECTS)(start[o] +duration[o] <= end);

solve minimize end;

output [ "start = \(start)\nend = \(end)\n"];
```

列表使用cumulative来建模搬运家具问题的数据moving.dzn

```
OBJECTS = { piano, fridge, doublebed, singlebed,
            wardrobe, chair1, chair2, table };

duration = [60, 45, 30, 30, 20, 15, 15, 15];
handlers = [3, 2, 2, 1, 2, 1, 1, 2];
trolleys = [2, 1, 2, 2, 2, 0, 0, 1];

available_time = 180;
available_handlers = 4;
available_trolleys = 3;
```

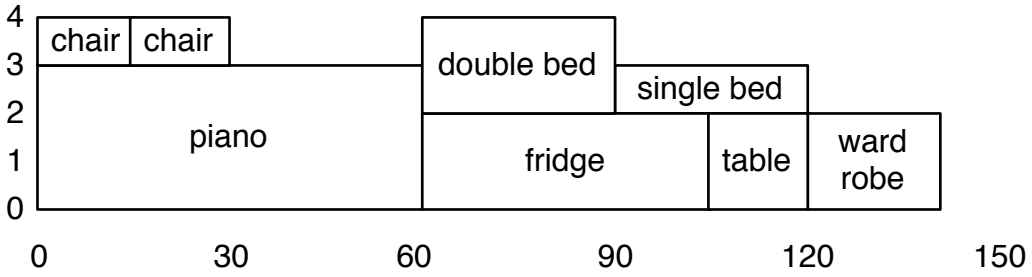
列表中的模型为搬运家具规划一个行程表使得每一份家具在搬运的过程中都有足够的搬用工和足够的手推车可以使用。允许的时间，可以使用的搬运工以及手推车被给出，每个物体的搬运持续时间，需要的搬运工和手推车的数量等数据也被给出。使用列表中的数据，命令

```
$ minizinc moving.mzn moving.dzn
```

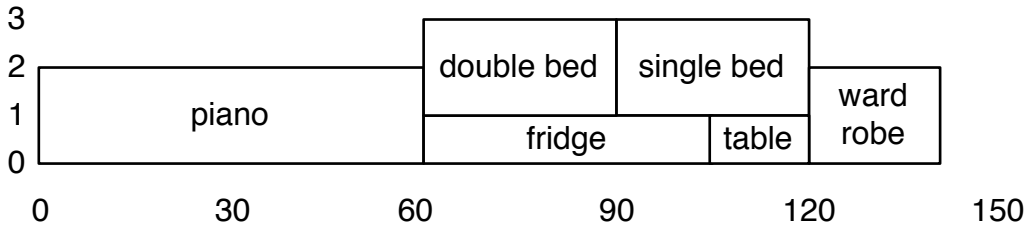
可能会得到如下输出

```
start = [0, 60, 60, 90, 120, 0, 15, 105]
end = 140
----------
==========
```

图图给出了这个解中搬运时每个时间点所需要的搬运工和手推车。



图搬运时搬运工使用量直方图



图搬运时手推车使用量直方图

2.3.1.3 Table

约束table强制变量元组从一个元组集合中取值。由于中没有元组，我们用数组来描述它。根据元组是布尔型还是整型，table的使用有以下两种格式

```
table(array[int] of var bool: x, array[int, int] of bool: t)
table(array[int] of var int:  x, array[int, int] of int:  t)
```

强制约束了$x \in t$，其中x和t中的每一行是元组，t是一个元组集合。

列表使用table约束来建模食物规划问题的模型meal.mzn

```
% 规划均衡的膳食
include "table.mzn";
int: min_energy;
int: min_protein;
int: max_salt;
int: max_fat;
set of FOOD: desserts;
set of FOOD: mains;
set of FOOD: sides;
enum FEATURE = { name, energy, protein, salt, fat, cost};
enum FOOD;
array[FOOD,FEATURE] of int: dd; % 食物数据库

array[FEATURE] of var int: main;
array[FEATURE] of var int: side;
array[FEATURE] of var int: dessert;
var int: budget;

constraint main[name] in mains;
```

```
constraint side[name] in sides;
constraint dessert[name] in desserts;
constraint table(main, dd);
constraint table(side, dd);
constraint table(dessert, dd);
constraint main[energy] + side[energy] + dessert[energy] >=min_energy;
constraint main[protein]+side[protein]+dessert[protein] >=min_protein;
constraint main[salt] + side[salt] + dessert[salt] <= max_salt;
constraint main[fat] + side[fat] + dessert[fat] <= max_fat;
constraint budget = main[cost] + side[cost] + dessert[cost];

solve minimize budget;

output ["main = ",show(to_enum(FOOD,main[name])),
        ", side = ",show(to_enum(FOOD,side[name])),
        ", dessert = ",show(to_enum(FOOD,dessert[name])),
        ", cost = ",show(budget), "\n"];
```

列表定义table的食物规划的数据meal.dzn

```
FOOD = { icecream, banana, chocolatecake, lasagna,
         steak, rice, chips, brocolli, beans} ;

dd = [| icecream,      1200,  50,  10, 120,  400     % 冰淇淋
      | banana,         800, 120,   5,  20,  120     % 香蕉
      | chocolatecake, 2500, 400,  20, 100,  600     % 巧克力蛋糕
      | lasagna,       3000, 200, 100, 250,  450     % 千层面
      | steak,         1800, 800,  50, 100, 1200     % 牛排
      | rice,          1200,  50,   5,  20,  100     % 米饭
      | chips,         2000,  50, 200, 200,  250     % 薯条
      | brocolli,       700, 100,  10,  10,  125     % 花椰菜
      | beans,         1900, 250,  60,  90,  150 |]; % 黄豆

min_energy = 3300;
min_protein = 500;
max_salt = 180;
max_fat = 320;
desserts = { icecream, banana, chocolatecake };
mains = { lasagna, steak, rice };
sides = { chips, brocolli, beans };
```

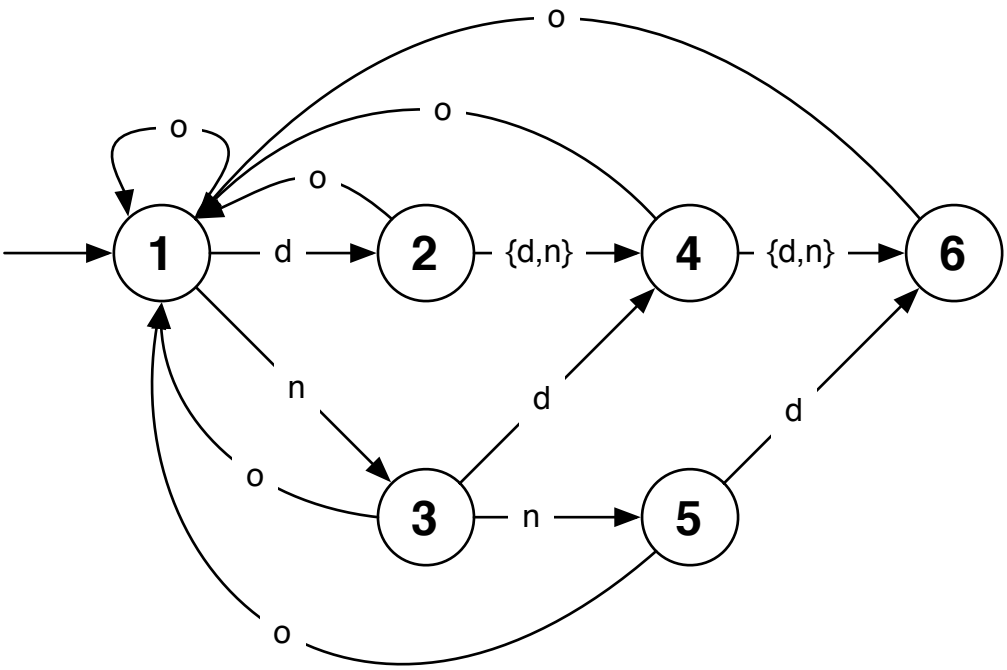
列表中的模型寻找均衡的膳食。每一个食物项都有一个名字（用整数表示），卡路里数，蛋白质克数，盐毫克数，脂肪克数以及单位为分的价钱。这些个项之间的关系用一个table约束来描述。模型寻找拥有最小花费，最少卡路里数min_energy，最少蛋白质量min_protein，最大盐分max_salt以及脂肪max_fat的膳食。

2.3.1.4 Regular

约束regular用来约束一系列的变量取有限自动机定义的值。regular的使用有以下方式

```
regular(array[int] of var int: x, int: Q, int: S,
        array[int,int] of int: d, int: q0, set of int: F)
```

它约束了x中的一列值（它们必须是在范围1..S内）被一个有Q个状态，输入为1..S，转换函数为d（<1..Q, 1..S>映射到0..Q），初始状态为q0（必须在1..Q中）和接受状态为F`（必须在 :mzn:`1..Q中）的接受。状态被保留为总是会失败的状态。



图判定正确排班的。

我们来看下护士排班问题。每一个护士每一天被安排为以下其中一种：白班夜班或者休息。每四天，护士必须有至少一天的休息。每个护士都不可以被安排为连续三天夜班。这个问题可以使用图中的不完全来表示。我们可以把这个表示为初始状态是1，结束状态是1..6，转换函数为

	d	n	o

注意状态表中的状态代表一个错误状态。列表中给出的模型为num_nurses个护士num_days天寻找

一个排班，其中我们要求白天有req_day个护士值班，晚上有req_night个护士值班，以及每个护士至少有min_night个夜班。

列表使用regular约束来建模的护士排班问题模型nurse.mzn

```
% Simple nurse rostering
include "regular.mzn";
enum NURSE;
enum DAY;
int: req_day;
int: req_night;
int: min_night;

enum SHIFT = { d, n, o };

enum STATE = { S1, S2, S3, S4, S5, S6 };

array[STATE,SHIFT] of opt STATE: t =
     [|      d:  n:  o:
      | S1:  S2, S3, S1
      | S2:  S4, S4, S1
      | S3:  S4, S5, S1
      | S4:  S6, S6, S1
      | S5:  S6, <>, S1
      | S6:  <>, <>, S1|];

array[NURSE,DAY] of var SHIFT: roster;

constraint forall(j in DAY)(
             sum(i in NURSE)(roster[i,j] == d) == req_day /\
             sum(i in NURSE)(roster[i,j] == n) == req_night
           );
constraint forall(i in NURSE)(
             regular([roster[i,j] | j in DAY], t, S1, STATE) /\
             sum(j in DAY)(roster[i,j] == n) >= min_night
           );

solve satisfy;
```

运行命令

```
$ minizinc nurse.mzn nurse.dzn
```

找到一个给个护士天的排班，要求白天有个人值班，夜晚有个人值班，以及每个护士最少有个夜班。一个可能的结果是

```
o d n n o n n d o o
d o n d o d n n o n
o d d o d o d n n o
d d d o n n d o n n
d o d n n o d o d d
n n o d d d o d d d
n n o d d d o d d d
----------
```

另外一种约束是regular_nfa。它使用（没有\epsilon弧）来定义表达式。此约束有以下格式

```
regular_nfa(array[int] of var int: x, int: Q, int: S,
        array[int,int] of set of int: d, int: q0, set of int: F)
```

它约束了数组x中的数值序列（必须在范围1..S中）被含有Q个状态，输入为1..S，转换函数为d（映射<1..Q, 1..S>到1..Q的子集），初始状态为q0（必须在范围1..Q中）以及接受状态为F（必须在范围1..Q中）的接受。在这里，我们没必要再给出失败状态，因为转换函数可以映射到一个状态的空集。

2.3.2 定义谓词

的其中一个最强大的建模特征是建模者可以定义他们自己的高级约束。这就使得他们可以对模型进行抽象化和模块化。也允许了在不同的模型之间重新利用约束以及促使了用来定义标准约束和类型的特殊库应用的发展。

列表使用谓词的车间作业调度问题模型jobshop2.mzn

```
int: jobs;                                      % 作业的数量
set of int: JOB = 1..jobs;
int: tasks;                                     % 每个作业的任务数量
set of int: TASK = 1..tasks;
array [JOB,TASK] of int: d;                     % 任务持续时间
int: total = sum(i in JOB, j in TASK)(d[i,j]);% 总持续时间
int: digs = ceil(log(10.0,total));              % 输出的数值
array [JOB,TASK] of var 0..total: s;            % 起始时间
var 0..total: end;                              % 总结束时间

% nooverlap
predicate no_overlap(var int:s1, int:d1, var int:s2, int:d2) =
    s1 + d1 <= s2 \/ s2 + d2 <= s1;

constraint %% 保证任务按照顺序出现
    forall(i in JOB) (
        forall(j in 1..tasks-1)
            (s[i,j] + d[i,j] <= s[i,j+1]) /\
        s[i,tasks] + d[i,tasks] <= end
    );

constraint %% 保证任务之间没有重叠
    forall(j in TASK) (
        forall(i,k in JOB where i < k) (
            no_overlap(s[i,j], d[i,j], s[k,j], d[k,j])
        )
    );

solve minimize end;

output ["end = \(end)\n"] ++
       [ show_int(digs,s[i,j]) ++ " " ++
         if j == tasks then "\n" else "" endif |
         i in JOB, j in TASK ];
```

我们用一个简单的例子开始，回顾下前面章节中的车间作业调度问题。这个模型在列表中给出。我们感兴趣的项是谓词项：

```
predicate no_overlap(var int:s1, int:d1, var int:s2, int:d2) =
    s1 + d1 <= s2 \/ s2 + d2 <= s1;
```

它定义了一个新的约束用来约束起始时间为s1，持续时间为d1的任务不能和起始时间为s2，持续时间为d2的任务重叠。它可以在模型的任何（包含决策变量的）布尔型表达式可以出现的地方使用。

和谓词一样，建模者也可以定义只涉及到参数的新的约束。和谓词不一样的是，它们可以被用在条件表达式的测试中。它们被关键字test定义。例如

```
test even(int:x) = x mod 2 = 0;
```

谓词定义

使用以下形式的语句，我们可以定义谓词

```
predicate <谓词名> ( <参数定义>, ..., <参数定义> ) = <布尔表达式>
```

<谓词名>必须是一个合法的标识符，每个<参数定义>都是一个合法的类型声明。
参数定义的一个松弛是数组的索引类型可以是没有限制地写为int。
类似的，使用以下形式的语句，我们定义测试

```
test <谓词名> ( <参数定义>, ..., <参数定义> ) = <布尔表达式>
```

其中的<布尔表达式>必须是固定的。
另外我们介绍一个谓词中使用到的assert命令的新形式。

```
assert ( <布尔表达式>, <字符串表达式>, <表达式> )
```

assert表达式的类型和最后一个参数的类型一样。assert表达式检测第一个参数是否为假，如果是则输出第二个参数字符串。如果第一个参数是真，则输出第三个参数。

注意表达式中的第三个参数是延迟的，即如果第一个参数是假，它就不会被评估。所以它可以被用来检查

```
predicate lookup(array[int] of var int:x, int: i, var int: y) =
    assert(i in index_set(x), "index out of range in lookup"
           y = x[i]
    );
```

此代码在i超出数组x的范围时不会计算x[i]。

2.3.3 定义函数

中的函数和谓词一样定义，但是它有一个更一般的返回类型。

下面的函数定义了一个数独矩阵中的第a^{th}个子方块的第$a1^{th}$行。

```
function int: posn(int: a, int: a1) = (a-1) * S + a1;
```

有了这个定义之后，我们可以把列表中的数独问题的最后一个约束替换为

```
constraint forall(a, o in SubSquareRange)(
                  alldifferent([ puzzle [ posn(a,a0), posn(o,o1) ] |
                                       a1,o1 in SubSquareRange ] ) );
```

函数对于描述模型中经常用到的复杂表达式非常有用。例如，想象下在$n \times n$的方格的不同位置上放置数字到n使得任何两个数字i和j之间的曼哈顿距离比这两个数字其中最大的值减一还要大。我们的目的是最小化数组对之间的总的曼哈顿距离。曼哈顿距离函数可以表达为：

```
function var int: manhattan(var int: x1, var int: y1,
                            var int: x2, var int: y2) =
         abs(x1 - x2) + abs(y1 - y2);
```

完整的模型在列表中给出。

列表阐释如何使用函数的数字放置问题模型manhattan.mzn

```
int: n;
set of int: NUM = 1..n;

array[NUM] of var NUM: x;
array[NUM] of var NUM: y;
array[NUM,NUM] of var 0..2*n-2: dist =
     array2d(NUM,NUM,[
     if i < j then manhattan(x[i],y[i],x[j],y[j]) else 0 endif
                     | i,j in NUM ]);

% manf
function var int: manhattan(var int: x1, var int: y1,
                            var int: x2, var int: y2) =
         abs(x1 - x2) + abs(y1 - y2);

constraint forall(i,j in NUM where i < j)
                 (dist[i,j] >= max(i,j)-1);

var int: obj = sum(i,j in NUM where i < j)(dist[i,j]);
solve minimize obj;

% simply to display result
include "alldifferent_except_0.mzn";
array[NUM,NUM] of var 0..n: grid;
constraint forall(i in NUM)(grid[x[i],y[i]] = i);
constraint alldifferent_except_0([grid[i,j] | i,j in NUM]);

output ["obj = \(obj);\n"] ++
```

```
      [ if fix(grid[i,j]) > 0 then show(grid[i,j]) else "." endif
        ++ if j = n  then "\n" else "" endif
      | i,j in NUM ];
```

函数定义

函数用以下格式的语句定义

```
function <返回类型> : <函数名> ( <参数定义>, ..., <参数定义> ) = <表达式>
```

<函数名>必须是一个合法的标识符。每一个<参数定义>是一个合法的类型声明。<返回类型>是函数的返回类型，它必须是<表达式>的类型。参数和谓词定义中的参数有一样的限制。

中的函数可以有任何返回类型，而不只是固定的返回类型。在定义和记录多次出现在模型中的复杂表达式时，函数是非常有用的。

2.3.4 反射函数

为了方便写出一般性的测试和谓词，各种反射函数会返回数组的下标集合，集合的定义域以及决策变量范围的信息。关于下标集合的有以下反射函数index_set(<1-D array>)index_set_1of2(<2-D array>)index_set_2of2(<2-D array>)以及关于更高维数组的反射函数。

车间作业问题的一个更好的模型是把所有的对于同一个机器上的不重叠约束结合为一个单个的析取约束。这个方法的一个优点是虽然我们只是初始地把它建模成一个non-overlap约束的连接，但是如果下层的求解器对于解决析取约束有一个更好的方法，在对我们的模型最小改变的情况下，我们可以直接使用它。这个模型在列表中给出。

列表使用disjunctive谓词的车间作业调度问题模型jobshop3.mzn

```
include "disjunctive.mzn";

int: jobs;                                      % 作业的数量
set of int: JOB = 1..jobs;
int: tasks;                                     % 每个作业的任务数量
set of int: TASK = 1..tasks;
array [JOB,TASK] of int: d;                     % 任务持续时间
int: total = sum(i in JOB, j in TASK)(d[i,j]);% 总持续时间
int: digs = ceil(log(10.0,total));              % 输出的数值
array [JOB,TASK] of var 0..total: s;            % 起始时间
var 0..total: end;                              % 总结束时间

constraint %% 保证任务按照顺序出现
    forall(i in JOB) (
        forall(j in 1..tasks-1)
            (s[i,j] + d[i,j] <= s[i,j+1]) /\
        s[i,tasks] + d[i,tasks] <= end
    );

constraint %% 保证任务之间没有重叠
    forall(j in TASK) (
        disjunctive([s[i,j] | i in JOB], [d[i,j] | i in JOB])
    );

solve minimize end;
```

```
output ["end = \(end)\n"] ++
       [ show_int(digs,s[i,j]) ++ " " ++
         if j == tasks then "\n" else "" endif |
         i in JOB, j in TASK ];
```

约束disjunctive获取每个任务的开始时间数组以及它们的持续时间数组，确保每次只有一个任务是被激活的。我们定义析取约束为一个有以下特征的谓词

```
predicate disjunctive(array[int] of var int:s, array[int] of int:d);
```

在列表中，我们可以用这个析取约束定义任务之间不重叠。我们假设disjunctive谓词的定义已经在模型中引用的文件disjunctive.mzn中给出。

如果下层的系统直接支持disjunctive，则会在它的全局目录下包含一个disjunctive.mzn文件(拥有上述特征定义内容)。如果我们使用的系统不直接支持析取，通过创建文件disjunctive.mzn，我们可以给出我们自己的定义。最简单的实现是单单使用上面定义的no_overlap谓词。一个更好的实现是利用全局约束cumulative，假如下层求解器支持它的话。列表给出了一个disjunctive的实现。注意我们使用index_set反射函数来 () 检查disjunctive的参数是有意义的，以及 () 构建cumulative的合适大小的资源利用数组。另外注意这里我们使用了assert的三元组版本。

列表使用cumulative来定义一个disjunctive谓词disjunctive.mzn

```
include "cumulative.mzn";

predicate disjunctive(array[int] of var int:s, array[int] of int:d) =
          assert(index_set(s) == index_set(d), "disjunctive: " ++
          "first and second arguments must have the same index set",
            cumulative(s, d, [ 1 | i in index_set(s) ], 1)
          );
```

2.3.5 局部变量

在谓词，函数或者测试中，引进局部变量总是非常有用的。表达式let允许你去这样做。它可以被用来引进决策变量决策变量和参数，但是参数必须被初始化。例如：

```
var s..e: x;
let {int: l = s div 2; int: u = e div 2; var l .. u: y;} in x = 2*y
```

引进了参数l和u以及变量y。let表达式虽然在谓词，函数和测试定义中最有用，它也可以被用在其他的表达式中。例如，来消除共同的子表达式：

```
constraint let { var int: s = x1 + x2 + x3 + x4 } in
           l <= s /\ s <= u;
```

局部变量可以被用在任何地方，在简化复杂表达式时也很有用。通过使用局部变量来定义目标函数而不是显式地加入很多个变量，列表给出了稳定婚姻模型的一个改进版本。

列表使用局部变量来定义一个复杂的目标函数wedding2.mzn

```
enum Guests = { bride, groom, bestman, bridesmaid, bob, carol,
  ted, alice, ron, rona, ed, clara};
set of int: Seats = 1..12;
set of int: Hatreds = 1..5;
array[Hatreds] of Guests: h1 = [groom, carol, ed, bride, ted];
array[Hatreds] of Guests: h2 = [clara, bestman, ted, alice, ron];
set of Guests: Males = {groom, bestman, bob, ted, ron,ed};
set of Guests: Females = {bride,bridesmaid,carol,alice,rona,clara};

array[Guests] of var Seats: pos; % 客人的座位

include "alldifferent.mzn";
constraint alldifferent(pos);

constraint forall(g in Males)( pos[g] mod 2 == 1 );
constraint forall(g in Females)( pos[g] mod 2 == 0 );

constraint not (pos[ed] in {1,6,7,12});
constraint abs(pos[bride] - pos[groom]) <= 1 /\
           (pos[bride] <= 6 <-> pos[groom] <= 6);

solve maximize sum(h in Hatreds)(
          let {  var Seats: p1 = pos[h1[h]];
                 var Seats: p2 = pos[h2[h]];
                 var 0..1: same = bool2int(p1 <= 6 <-> p2 <= 6); } in
          same * abs(p1 - p2) + (1-same) * (abs(13 - p1 - p2) + 1));

output [ show(g)++" " | s in Seats,g in Guests where fix(pos[g]) == s]
       ++ ["\n"];
```

2.3.6 语境

有一个局限，即含有决策变量并且在声明时没有初始化的谓词和函数不可以被用在一个否定语境下。下面例子是非法的

```
predicate even(var int:x) =
          let { var int: y } in x = 2 * y;

constraint not even(z);
```

原因是求解器只解决存在约束的问题。如果我们在否定语境下引入了一个局部变量，则此变量是普遍地量化了，因此超出下层求解器的解决范围。例如，$\neg(z)$等价于$\neg\exists y.z = 2y$，然后等价于$\forall y.z \neq 2y$。

如果局部变量被赋了值，则它们可以被用在否定语境中。下面的例子是合法的

```
predicate even(var int:x) =
          let { var int: y = x div 2; } in x = 2 * y;

constraint not even(z);
```

注意，现在even的意思是正确的，因为如果x是偶数，则$x = 2 * (x2)$。由于被z功能性定义了，$\neg(z)$等价于$\neg\exists y.y = z2 \wedge z = 2y$，同时等价于$\exists y.y = z2 \wedge \neg z \neq 2y$。

中的任意表达式都出现在以下四种语境中的一种中：根，肯定，否定，或者混合。非布尔型表达式的语境直接地为包含其最近的布尔型表达式的语境。唯一的例外是目标表达式出现在一个根语境下（由于它没有包含其的布尔型表达式）。

为了方便定义语境，我们把蕴含表达式`e1 -> e2`等价地写为`not e1 \/ e2`，`e1 <- e2`等价地写为`e1 \/ not e2`。

一个布尔型表达式的语境可能有：

根

根语境是任何作为constraint的参数或者作为一个赋值项出现的表达式e的语境，或者作为一个出现在根语境中的`e1 /\ e2`的子表达式e1或e2的语境。

根语境下的布尔型表达式必须在问题的任何模型中都满足。

肯定

肯定语境是任何作为一个出现在根语境或者肯定语境中的`e1 \/ e2`的子表达式e1或e2的语境，或者是作为一个出现在肯定语境中的`e1 /\ e2`的子表达式e1或e2`的语境，或者是作为一个出现在否定语境中的 :mzn:`not e`的子表达式e的语境。

肯定语境下的布尔型表达式不是必须要在模型中满足，但是满足它们会增加包含其的约束被满足的可能性。对于一个肯定语境下的表达式，从包含其的根语境到此表达式有偶数个否定。

否定

否定语境是任何作为一个出现在根语境或者否定语境中的`e1 \/ e2`或`e1 /\ e2`的子表达式e1或e2，或者是作为一个出现在肯定语境中的`not e`的子表达式e的语境。

否定语境下的布尔型表达式不是必须要满足，但是让它们成假会增加包含其的约束被满足的可能性。对于一个否定语境下的表达式，从包含其的根语境到此表达式有奇数个否定。

混合

混合语境是任何作为一个出现在`e1 <-> e2`e1 = e2或者bool2int(e)中的子表达式e1或e2的语境。

混合语境下的表达式实际上既是肯定也是否定的。通过以下可以看出：`e1 <-> e2`等价于`(e1 /\ e2) \/ (not e1 /\ not e2)`以及`x = bool2int(e)`等价于`(e /\ x=1) \/ (not e /\ x=0)`。

观察以下代码段

```
constraint x > 0 /\ (i <= 4 -> x + bool2int(x > i) = 5);
```

其中`x > 0`在根语境中，`i <= 4`}在否定语境中，`x + bool2int(x > i) = 5`在肯定语境中，`x > i`在混合语境中。

2.3.7 局部约束

表达式也可以被用来引入局部约束，通常用来约束局部变量的行为。例如，考虑只利用乘法来定义开根号函数：

```
function var float: mysqrt(var float:x) =
         let { var float: y;
               constraint y >= 0;
               constraint x = y * y; } in y;
```

局部约束确保了y取正确的值；而此值则会被函数返回。

局部约束可以在表达式中使用，尽管最普遍的应用是在定义函数时。

表达式

局部变量可以在任何以下格式的表达式中引入：

```
let { <声明>; ... <声明> ; } in <表达式>
```

声明<dec>可以是决策变量或者参数（此时必须被初始化）或者约束项的声明。任何声明都不能在一个新的声明变量还没有引进时使用它。
注意局部变量和约束不可以出现在测试中。局部变量不可以出现在否定或者混合语境下的谓词和函数中，除非这个变量是用表达式定义的。

2.3.8 定义域反射函数

其他重要的反射函数有允许我们对变量定义域进行访问的函数。表达式lb(x)返回一个小于等于x在一个问题的解中可能取的值的数值。通常它会是x声明的下限。如果x被声明为一个非有限类型，例如，只是var int，则它是错误的。类似地，表达式dom(x)返回一个在问题的任何解中的可能值的（非严格）超集。再次，它通常是声明的值，如果它不是被声明为有限则会出现错误。

列表使用反射谓词reflection.mzn

```
var -10..10: x;
constraint x in 0..4;
int: y = lb(x);
set of int: D = dom(x);
solve satisfy;
output ["y = ", show(y), "\nD = ", show(D), "\n"];
```

例如，列表中的模型或者输出

```
y = -10
D = -10..10
----------
```

或

```
y = 0
D = {0, 1, 2, 3, 4}
----------
```

或任何满足$-10 \leq y \leq 0$和$\{0,\ldots,4\} \subseteq D \subseteq \{-10,\ldots,10\}$的答案。

变量定义域反射表达式应该以在任何安全近似下都正确的的方式使用。但是注意这个是没有被检查的！例如加入额外的代码

```
var -10..10: z;
constraint z <= y;
```

不是一个定义域信息的正确应用。因为使用更紧密（正确的）近似会比使用更弱的初始近似产生更多的解。

定义域反射

我们有查询包含变量的表达式的可能值的反射函数：

dom(<表达式>)返回<表达式>所有可能值的安全近似。

lb(<表达式>)返回<表达式>下限值的安全近似。

ub(<表达式>)返回<表达式>上限值的安全近似。

lb和ub的表达式必须是int，bool，float或者set of int类型。dom中表达式的类型不能是float。如果<表达式>中的一个变量有一个非有限声明类型（例如，var int或var float类型），则会出现一个错误。

我们也有直接作用于表达式数组的版本（有类似的限制）：

dom_array(<数组表达式>)返回数组中出现的表达式的所有可能值的并集的一个安全近似。

lb_array(<数组表达式>)返回数组中出现的所有表达式的下限的安全近似。

ub_array(<数组表达式>)返回数组中出现的所有表达式的下限的安全近似。

谓词，局部变量和定义域反射的结合使得复杂全局约束通过分解定义变为可能。利用列表中的代码，我们可以定义cumulative约束的根据时间的分解。

列表利用分解来定义一个谓词cumulative.mzn

```
%-----------------------------------------------------------------------%
% 需要给出一个任务集合，其中起始时间为's',
% 持续时间'd'以及资源需求量'r',
% 任何时候需求量都不能超过
% 一个全局资源界限'b'。
% 假设：
% - forall i, d[i] >= 0 and r[i] >= 0
%-----------------------------------------------------------------------%
predicate cumulative(array[int] of var int: s,
                     array[int] of var int: d,
                     array[int] of var int: r, var int: b) =
   assert(index_set(s) == index_set(d) /\
          index_set(s) == index_set(r),
     "cumulative: the array arguments must have identical index sets",
   assert(lb_array(d) >= 0 /\ lb_array(r) >= 0,
     "cumulative: durations and resource usages must be non-negative",
          let {
              set of int: times =
                lb_array(s) ..
                max([ ub(s[i]) + ub(d[i]) | i in index_set(s) ])
              }
          in
             forall( t in times ) (
```

```
                b >= sum( i in index_set(s) ) (
                   bool2int( s[i] <= t /\ t < s[i] + d[i] ) * r[i]
                )
             )
       )
    );
```

这个分解利用lb和ub来决定任务可以执行的时间范围集合。接下来，它对times中的每个时间times都断言在此时间t激活的所有任务所需要的资源量总和小于界限b。

2.3.9 作用域

中声明的作用域值得我们简单地介绍下。只有一个作用域，所以出现在声明中的所有变量都可以在模型中的每个表达式中可见。用以下几个方式，引进局部作用域变量：

推导式表达式中的迭代器

使用let表达式

谓词和函数中的参数

任何局部作用域变量都会覆盖同名称的外部作用域变量。

列表阐述变量作用域的模型scope.mzn

```
int: x = 3;
int: y = 4;
predicate smallx(var int:y) = -x <= y /\ y <= x;
predicate p(int: u, var bool: y) =
      exists(x in 1..u)(y \/ smallx(x));
constraint p(x,false);
solve satisfy;
```

例如，在列表中给出的模型中，-x <= y中的x是全局x，smallx(x)中的x是迭代器x in 1..u，而析取中的y是谓词的第二个参数。

CHAPTER 2.4

选项类型

选项类型是一个强大的抽象，使得简洁建模成为可能。一个选项类型决策变量代表了一个有其他可能⊤的变量，在中表达为<>，代表了这个变量是缺失的。选项类型变量在建模一个包含在其他变量没做决定之前不会有意义的变量的问题时是很有用的。

2.4.1 声明和使用选项类型

选项类型变量

一个选项类型变量被声明为：

```
var opt <类型> : <变量名>:
```

其中<类型>是int，float或bool中的一个，或者是一个固定范围的表达式。选项类型变量可以是参数，但是这个不常用。
一个选项类型变量可以有附加值<>表明它是缺失的。
三个内建函数被提供给选项类型变量：absent(v)只有在选项类型变量v取值<>时返回true，occurs(v)只有在选项类型变量v不取值<>时返回true，以及<>返回v的正常值或者当它取值<>时返回失败。

选项类型最常被用到的地方是调度中的可选择任务。在灵活的车间作业调度问题中，我们有n个在k个机器上执行的任务，其中完成每个机器上每个任务的时间可能是不一样的。我们的目标是最小化所有任务的总完成时间。一个使用选项类型来描述问题的模型在列表中给出。在建模这个问题的时候，我们使用$n \times k$个可选择的任务来代表每个机器上每个任务的可能性。我们使用alternative全局约束来要求任务的起始时间和它的持续时间跨越了组成它的可选择任务的时间，同时要求只有一个会实际运行。我们使用disjunctive全局变量在每个机器上最多有一个任务在运行，这里我们延伸到可选择的任务。最后我们约束任何时候最多有个任务在运行，利用一个在实际（不是可选择的）任务上作用的冗余约束。

列表使用选项类型的灵活车间作业调度模型flexible-js.mzn

```
include "globals.mzn";

int: horizon;                  % 时间范围
set of int: Time = 0..horizon;
enum Task;
enum Machine;

array[Task,Machine] of int: d; % 每个机器上的持续时间
int: maxd = max([ d[t,m] | t in Task, m in Machine ]);
int: mind = min([ d[t,m] | t in Task, m in Machine ]);

array[Task] of var Time: S;             % 起始时间
array[Task] of var mind..maxd: D;       % 持续时间
array[Task,Machine] of var opt Time: O; % 可选择的任务起始

constraint forall(t in Task)(alternative(S[t],D[t],
                [O[t,m]|m in Machine],[d[t,m]|m in Machine]));
constraint forall(m in Machine)
                (disjunctive([O[t,m]|t in Task],[d[t,m]|t in Task]));
constraint cumulative(S,D,[1|i in Task],card(Machine));

solve minimize max(t in Task)(S[t] + D[t]);
```

2.4.2 隐藏选项类型

当列表推导式是从在变量集合迭代上创建而来，或者`where`从句中的表达式还没有固定时，选项类型变量会隐式地出现。

例如，模型片段

```
var set of 1..n: x;
constraint sum(i in x)(i) <= limit;
```

是以下的语法糖

```
var set of 1..n: x;
constraint sum(i in 1..n)(if i in x then i else <> endif) <= limit;
```

内建函数`sum`实际上在一列类型实例化`var opt int`上操作。由于`<>`在中表现为标识，我们会得到期望的结果。

类似地，模型片段

```
array[1..n] of var int: x;
constraint forall(i in 1..n where x[i] >= 0)(x[i] <= limit);
```

是以下的语法糖

```
array[1..n] of var int: x;
constraint forall(i in 1..n)(if x[i] >= 0 then x[i] <= limit else <> endif);
```

同样地，函数forall实际上在一列类型实例化var opt bool上操作。由于<>在/\ 上表现为标识true，我们可以得到期望的结果。

尽管我们已经很小心了，隐式的使用可能会导致意外的行为。观察

```
var set of 1..9: x;
constraint card(x) <= 4;
constraint length([ i | i in x]) > 5;
solve satisfy;
```

它本应该是一个不可满足的问题。它返回x = {1,2,3,4}作为一个解例子。这个是正确的因为第二个约束等于

```
constraint length([ if i in x then i else <> endif | i in 1..9 ]) > 5;
```

而可选择整数列表的长度总会是，所以这个约束总是会满足。

我们可以通过不在变量集合上创建迭代或者使用不固定的where从句来避免隐式的选项类型。例如，上面的两个例子可以不使用选项类型重写为

```
var set of 1..n: x;
constraint sum(i in 1..n)(bool2int(i in x)*i) <= limit;
```

和

```
array[1..n] of var int: x;
constraint forall(i in 1..n)(x[i] >= 0 -> x[i] <= limit);
```

CHAPTER 2.5

Tuple and record types

“”

“” ’

2.5.1 Declaring and using tuples and records

```
<var-par> tuple(<ti-expr>, ...): <var-name>
```

<ti-expr>
varvar tuple(int, bool)tuple(var int, var bool)

```
(<expr>, [ <expr>, ... ])
```

(1, true)tuple(int,bool)

any: x = (1, true, 2.0)1x.12.0x.3

```
<var-par> record(<ti-expr-and-id>, ...): <var-name>
```

<ti-expr-and-id>
varvar record(int: i, bool: b)record(var int: i, var bool: b)

```
(<ident>: <expr>[, ... ])
```

(i: 1, b: true)record(int: i, bool: b)

any: x = (i: 1, b: true)1ix.i

2.5.2 Using type-inst synonyms

```
type <ident> <annotations> = <ti-expr>;
```

<ident><ti-expr>

CoordNumber

```
type Coord = var record(int: x, int: y, int: z);
type Number = int;
```

array[1..10] of Coord: placement;function Number:add(Number: x, Number: y) = x + y;

vartype OnOff = bool;var OnOff: choice;choicevarboolpartype Choice = varbool;par Choice: check = fix(choi
checkbool

2.5.3 Types with both var and par members

```
enum EmpId;
type Employee = record(
  string: name,
  array[Timespan] of bool: available,
  set of Capability: capacities,
  array[Timespan] of var Shift: shifts,
  var 0..infinity: hours,
);

array[EmpId] of Employee: employee;
```

employees

```
enum EmpId;
type EmployeeData = record(
  string: name,
  array[Timespan] of bool: available,
  set of Capability: capacities,
);
type EmployeeVar = record(
  array[Timespan] of var Shift: shifts,
  var 0..infinity: hours,
);

array[EmpId] of EmployeeData: employee_data;
array[EmpId] of EmployeeVar: employee_var;
```

employee_data++

```
type Employee = EmployeeData ++ EmployeeVar;

array[EmpId] of Employee: employee = [employee_data[id] ++ employee_var[id] | id in␣
↪EmpId];
```

```
<expr> ++ <expr>
<ti-expr> ++ <ti-expr>
```

2.5.4 Example: rectangle packing using records

,

“”

```
type Dimensions = record(int: width, int: height);
```

```
type Coordinates = record(var 0..infinity: x, var 0..infinity: y);
```

```
type Rectangle = Dimensions ++ Coordinates;

% No overlap predicate
predicate no_overlap(Rectangle: rectA, Rectangle: rectB) =
  rectA.x + rectA.width <= rectB.x
  \/ rectB.x + rectB.width <= rectA.x
  \/ rectA.y + rectA.height <= rectB.y
  \/ rectB.y + rectB.height <= rectA.y;
```

列表rect_packing.mznrect_packing.json

```
% Instance data
array[_] of Dimensions: rectDim;
Dimensions: area;

% Decision variables
array[index_set(rectDim)] of Coordinates: rectCoord ::no_output;

array[_] of Rectangle: rectangles ::output = [ rectDim[i] ++ rectCoord[i] | i in␣
↪index_set(rectDim)];

% Constraint: rectangles must be placed within the area
constraint forall(rect in rectangles) (
  rect.x + rect.width <= area.width
  /\ rect.y + rect.height <= area.height
);

% Constraint: no rectangles can overlap
constraint forall(i, j in index_set(rectangles) where i < j) (
  no_overlap(rectangles[i], rectangles[j])
);
```

CHAPTER 2.6

搜索

默认没有我们想如何搜索解的声明。这就把搜索全部都留给下层的求解器了。但是有些时候，尤其是对组合整数问题，我们或许想规定搜索应该如何去进行。这就需要我们和求解器沟通出一个搜索策略。注意，搜索策略不真的是模型的一部分。实际上，我们不要求每个求解器把所有可能的求解策略都实现了。通过使用来用一个稳定的方法跟约束求解器沟通额外的信息。

2.6.1 有限域搜索

利用有限域求解器搜索涉及到检查变量剩余的可能值以及选择进一步地约束一些变量。搜索则会加一个新的约束来限制这个变量的剩余值（实际上猜测解可能存在于哪里），然后使用传播来确定其他的值是否可能存在于解中。为了确保完全性，搜索会保留另外一个选择，而它是新约束的否定。搜索会当有限域求解器发现所有的约束都被满足，此时一个解已经被找到，或者有约束不被满足时停止。当不可满足出现的时候，搜索必须换另外一个不同的选择集合继续下去。通常有限域求解器使用深度优先搜索，它会撤销最后一个做的选择然后尝试做一个新的选择。

列表皇后问题模型nqueens.mzn

```
int: n;
array [1..n] of var 1..n: q; % i列的皇后在行q[i]

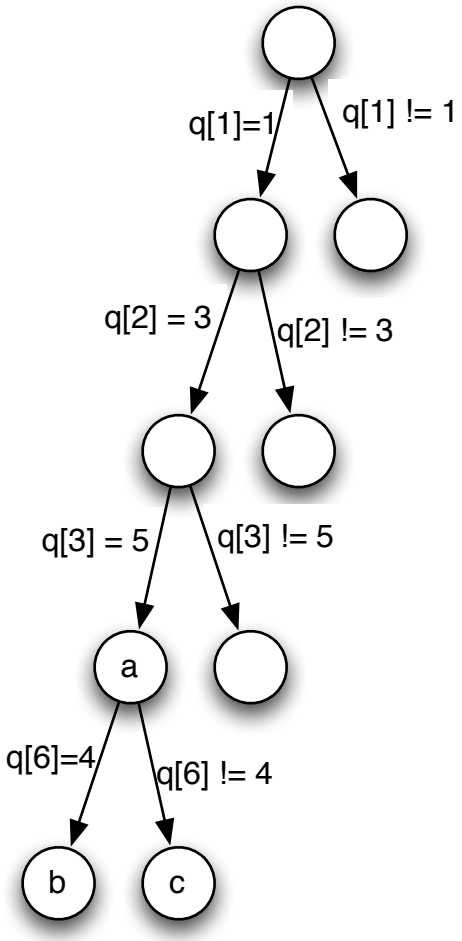
include "alldifferent.mzn";

constraint alldifferent(q);                        % 不同行
constraint alldifferent([ q[i] + i | i in 1..n]); % 不同对角线
constraint alldifferent([ q[i] - i | i in 1..n]); % 上+下

% 搜索
solve :: int_search(q, first_fail, indomain_min, complete)
      satisfy;
output [ if fix(q[j]) == i then "Q" else "." endif ++
         if j == n then "\n" else "" endif | i,j in 1..n]
```

有限域问题的一个简单例子是n皇后问题，它要求我们放置n个皇后在$n \times n$棋盘上使得任何两个都不会互相攻击。变量q[i]记录了在i列的皇后放置在哪一行上。alldifferent约束确保了任何两个皇后都不会在同一行或者对角线上。图的左边给出了n = 9的典型（部分）搜索树。我们首选

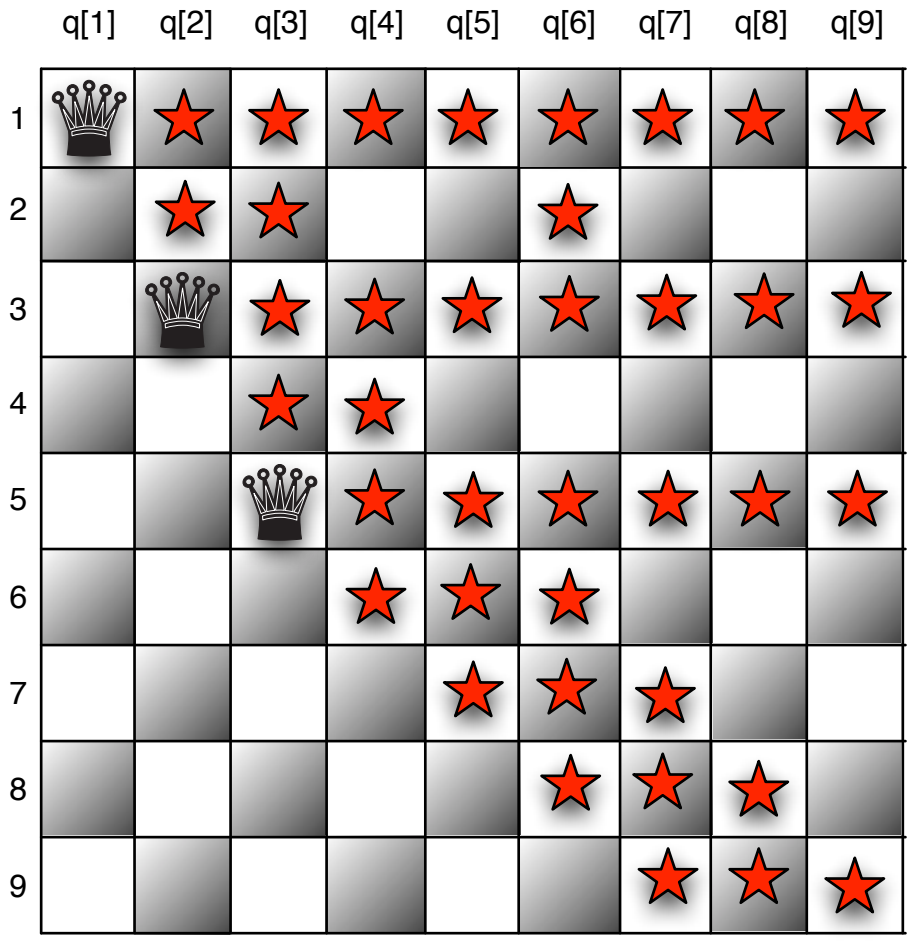
设置q[1] = 1，这样就可以从其他变量的定义域里面移除一些数值，例如q[2]不能取值或者我们接下来设置q[2] = 3，然后进一步地从其他变量的定义域里面移除一些数值。我们设置q[3] = 5（它最早的可能值）。在这三个决策后，棋盘的状态显示为图。其中皇后表示已经固定的皇后位置。星星表示此处放置的皇后会攻击到已经放置的皇后，所以我们不能在此处放置皇后。



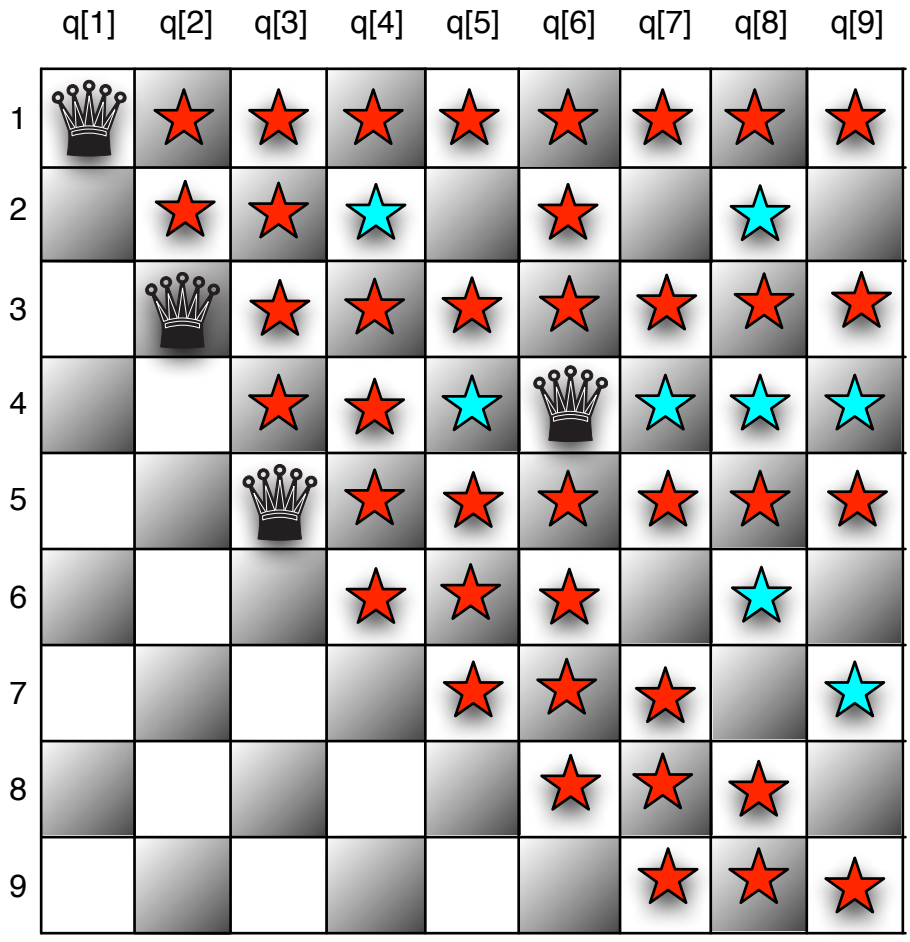
图皇后问题的部分搜索树

一个搜索策略决定要做哪一个选择。我们目前所做的决定都按照一个简单的策略：选择第一个还没有固定的变量，尝试设置它为它的最小可能值。按照这个策略，下一个决策应该是q[4] = 7。变量选择的另外一个策略是选择现在可能值集合定义域最小的变量。按照这个所谓最先失败变量选择策略，下一个决策应该是q[6] = 4。如果我们做了这个决策，则初始的传播会去除掉图中显示的额外的值。但是它使得q[8]只剩余有一个值。所以q[8] = 7被执行。但是这又使得q[7]和q[9]也只剩余一个值。因此这个时候有个约束一定会被违反。我们检测到了不满足性，求解器必须回溯取消最后一个决策q[6] = 4并且加入它的否定q[6] != 4（引导我们到了图中树的状态），即强制使q[6] = 8。这使得有些值从定义域中被去除。我们接下来继续重新启用搜索策略来决定该怎么做。

很多有限域搜索被定义为这种方式：选择一个变量来进一步约束，然后选择如何进一步地约束它。



图在加入q[1] = 1q[2] = 4q[3] = 5的状态



图在进一步加入q[6] = 4后的初始传播

2.6.2 搜索注解

中的搜索注解注明了为了找到一个问题的解应如何去搜索。注解附在求解项，在关键字solve之后。搜索注解

```
solve :: int_search(q, first_fail, indomain_min, complete)
      satisfy;
```

出现在求解项中。注解使用连接符::附为模型的一部分。这个搜索注解意思是我们应该按照从整型变量数组q中选择拥有最小现行定义域的变量（这个是first_fail规则），然后尝试设置其为它的最小可能值（indomain_min值选择），纵观整个搜索树来搜索（complete搜索）。

基本搜索注解

我们有三个基本搜索注解，相对应于不同的基本搜索类型：

int_search(<变量>, <变量选择>, <约束选择>, <策略>)其中<变量>是一个var int类型的一维数组，<变量选择>是一个接下来会讨论的变量选择注解，<约束选择>是一个接下来会讨论的如何约束一个变量的选择，<策略>是一个搜索策略，我们暂时假设为complete

bool_search(<变量>, <变量选择>, <约束选择>, <策略>)其中<变量>是一个var bool类型的一维数组，剩余的和上面一样。

set_search(<变量>, <变量选择>, <约束选择>, <策略>)其中<变量>是一个var set of int类型的一维数组，剩余的和上面一样。

float_search(<变量>, <精度>, <变量选择>, <约束选择>, <策略>)其中<变量>是一个一维var float数组<precision>是一个固定的用于表示ϵ浮点数其中两个数之差低于这个浮点数时被认为相等。剩余的和上面一样。

变量选择注解的例子有：

input_order从数组中按照顺序选择

first_fail选择拥有最小定义域大小的变量，以及

smallest选择拥有最小值的变量。

约束一个变量的方式有：

indomain_min赋最小的定义域内的值给变量，

indomain_median赋定义域内的中间值给变量，

indomain_random从定义域中取一个随机的值赋给变量，以及

indomain_split把变量定义域一分为二然后去除掉上半部分。

对于完全搜素，<策略>基本都是complete。关于一份完整的变量和约束选择注解，请参看参考文档中的说明书。

利用搜索构造注解，我们可以创建更加复杂的搜索策略。目前我们只有一个这样的注解。

```
seq_search([ <搜素注解>, ..., <搜素注解> ])
```

顺序搜索构造首先执行对列表中的第一个注解所指定的变量的搜索，当这个注解中的所有的变量都固定后，它执行第二个搜索注解，等等。直到所有的搜索注解都完成。

我们来看一下列表中给出的车间作业调度模型。我们可以替换求解项为

```
solve :: seq_search([
            int_search(s, smallest, indomain_min, complete),
            int_search([end], input_order, indomain_min, complete)])
      minimize end
```

通过选择可以最早开始的作业并设置其为s，起始时间被设置为s。当所有的起始时间都设置完后，终止时间end或许还没有固定。因此我们设置其为它的最小可能取值。

2.6.3 注解

在中，注解是第一类对象。我们可以在模型中声明新的注解，以及声明和赋值给注解变量。

注解

注解有一个类型ann。你可以声明一个注解参数（拥有可选择的赋值）

```
ann : <标识符>;
ann : <标识符> = <注解表达式> ;
```

对注解变量赋值和对其他参数赋值一样操作。
表达式，变量声明，和solve项都可以通过使用::操作符来成为注解。
使用注解项注解项，我们可以声明一个新的注解annotation项

```
annotation <注解名> ( <参数定义>, ..., <参数定义> ) ;
```

列表皇后问题的注解模型nqueens-ann.mzn

```
annotation bitdomain(int:nwords);

include "alldifferent.mzn";

int: n;
array [1..n] of var 1..n: q :: bitdomain(n div 32 + 1);

constraint alldifferent(q) :: domain_propagation;
constraint alldifferent([ q[i] + i | i in 1..n]) :: domain_propagation;
constraint alldifferent([ q[i] - i | i in 1..n]) :: domain_propagation;

ann: search_ann;

solve :: search_ann satisfy;

output [ if fix(q[j]) == i then "Q" else "." endif ++
         if j == n then "\n" else "" endif | i,j in 1..n];
```

列表中的程序阐述了注解声明，注解和注解变量的使用。我们声明一个新的注解bitdomain，意思是来告诉求解器变量定义域应该通过大小为nwords的比特数组来表示。注解附注在变量q的声明之后。每一个alldifferent约束都被注解为内部注解domain，而它指导求解器去使用alldifferent的定义域传播版本（如果有的话）。一个注解变量search_ann被声明和使用来定义搜索策略。我们可以在一个单独的数据文件中来给出搜素策略的值。

搜索注解的例子或许有以下几种（我们假设每一行都在一个单独的数据文件中）

```
search_ann = int_search(q, input_order, indomain_min, complete);
search_ann = int_search(q, input_order, indomain_median, complete);
search_ann = int_search(q, first_fail, indomain_min, complete);
search_ann = int_search(q, first_fail, indomain_median, complete);
```

第一个只是按顺序来选择皇后然后设置其为最小值。第二个按顺序来选择皇后，但是设置中间值给它。第三个选择定义域大小最小的皇后，然后设置最小值给它。最后一个策略选择定义域大小最小的皇后，设置中间值给它。

不同的搜索策略对于能多容易找到解有显著的差异。下面的表格给出了一个简单的关于使用种不同的搜索策略找到皇后问题的第一个解所要做的决策个数（其中—表示超过个决策）。很明显地看到，合适的搜索策略会产生显著的提高。

n	input-min	input-median	ff-min	ff-median
	—			
	—		—	
	—	—	—	

CHAPTER 2.7

MiniZinc 中的有效建模实践

对同一个问题，几乎总是存在多种方式来建模。其中一些产生的模型可以很有效地求解，另外一些则不是。通常情况下，我们很难提前判断哪个模型是对解决一个特定的问题最有效的。事实上，这或许十分依赖于我们使用的底层求解器。在这一章中，我们专注于建模实践，来避免产生模型的过程和产生的模型低效。

2.7.1 变量界限

有限域传播器，是所针对的求解器中的核心类型。在当其所涉及到的变量的界限越紧凑时，此传播器越有效。它也会当问题含有会取很大整型数值的子表达式时表现得很差，因为它们可能会隐式地限制整型变量的大小。

列表没有无界整数的模型grocery.mzn

```
var int: item1;
var int: item2;
var int: item3;
var int: item4;

constraint item1 + item2 + item3 + item4 == 711;
constraint item1 * item2 * item3 * item4 == 711 * 100 * 100 * 100;

constraint         0 < item1 /\ item1 <= item2
           /\ item2 <= item3 /\ item3 <= item4;

solve satisfy;

output ["{", show(item1), ",", show(item2), ",", show(item3), ",",
        show(item4),"}\n"];
```

在列表中的中的杂货店问题要找寻个物品使得它们的价格加起来有元并且乘起来也有元。变量被声明为无界限。运行

```
$ minizinc --solver g12fd grocery.mzn
```

得到

```
=====UNSATISFIABLE=====
% /tmp/mznfile85EWzj.fzn:11: warning: model inconsistency detected before search.
```

这是因为乘法中的中间表达式的类型也会是`var int`，也会被求解器给一个默认的界限 $-1,000,000 \ldots 1,000,000$。但是这个范围太小了以至于不能承载住中间表达式所可能取的值。

更改模型使得初始变量都被声明为拥有更紧致的界限

```
var 1..711: item1;
var 1..711: item2;
var 1..711: item3;
var 1..711: item4;
```

我们得到一个更好的模型，因为现在可以推断出中间表达式的界限，并且使用此界限而不是默认的界限。在做此更改后，求解模型我们得到

```
{120,125,150,316}
----------
```

注意，就算是改善的模型也可能对于某些求解器来说会很难解决。运行

```
$ minizinc --solver g12lazy grocery.mzn
```

不能得到任何结果，因为求解器给中间产生的变量创建了巨大的表示。

给变量加界限

在模型中要尽量使用有界限的变量。当使用`let`声明来引进新的变量时，始终尽量给它们定义正确的和紧凑的界限。这会使得你的模型更有效率，避免出现意外溢出的可能性。一个例外是当你引进一个新的变量然后立刻定义它等于一个表达式，通常都可以从此表达式推断出此变量有效的界限。

2.7.2 有效的生成元

想象下我们想要计算在一个图中出现的三角形的个数（K_3子图）。假设此图由一个邻接矩阵定义：如果点和邻接，则`adj[i,j]`为真。我们或许可以写成

```
int: count = sum ([ 1 | i,j,k in NODES where i < j  /\ j < k
                      /\ adj[i,j] /\ adj[i,k] /\ adj[j,k]]);
```

这当然是对的，但是它检查了所有点可能组成的三元组。如果此图是稀疏的，在意识到一旦我们选择了`i`和`j`，就可以进行一些测试之后，我们可以做得更好。

```
int: count = sum ([ 1 | i,j in NODES where i < j  /\ adj[i,j],
                        k in NODES where j < k /\ adj[i,k] /\ adj[j,k]]);
```

你可以使用内建`trace`函数来帮助决定在生成元内发生了什么。

追踪

函数trace(s,e)在对表达式e求值并返回它的值之前就输出字符串s。它可以在任何情境下使用。

例如，我们可以查看在两种计算方式下的内部循环中分别进行了多少次测试。

```
int:count=sum([ 1 | i,j,k in NODES where
          trace("+", i<j /\j<k /\ adj[i,j] /\ adj[i,k] /\ adj[j,k]) ]);
adj = [| false, true,  true,  false
       | true,  false, true,  false
       | true,  true,  false, true
       | false, false, true,  false |];
constraint trace("\n",true);
solve satisfy;
```

得到输出：

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
----------
```

表示内部循环进行了次，而

```
  int: count = sum ([ 1 | i,j in NODES where i < j  /\ adj[i,j],
                          k in NODES where trace("+", j < k /\ adj[i,k] /\ adj[j,
↪k])]);
```

得到输出

```
++++++++++++++++
----------
```

表示内部循环进行了次。

注意你可以在trace中使用单独的字符串来帮助你理解模型创建过程中发生了什么。

```
int: count = sum( i,j in NODES where i < j /\ adj[i,j])(
       sum([trace("("++show(i)++","++show(j)++","++show(k)++")",1) |
             k in NODES where  j < k /\ adj[i,k] /\ adj[j,k]]));
```

会输出在计算过程中找到的每个三角形。得到输出

```
(1,2,3)
----------
```

我们要承认这里我们有一点点作弊在某些情况下编译器实际上会把where从句中的参数自动重新排序所以他们会尽快地被计算。在这种情况下加入trace函数实际上阻止了这种优化一般来说通过分离where从句把它们摆到尽量接近生成元这其实是一个很好的主意来帮助编译器运作正常。

2.7.3 冗余约束

模型的形式会影响约束求解器求解它的效率。在很多情况下加入冗余约束，即被现有的模型逻辑上隐含的约束，可能会让求解器在更早时候产生更多可用的信息从而提高找寻解的搜索效率。

回顾下第复杂约束节中的魔术串问题。

运行n = 16时的模型：

```
$ minizinc --all-solutions --statistics magic-series.mzn -D "n=16;"
```

```
n = 16
```

```
s = [12, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0];
----------
==========
```

统计数据显示需要有个失败。

我们可以在模型中加入冗余约束。由于序列中的每个数字是用来计算某一个数字出现的次数，我们知道它们的和肯定是n。类似地，由于这个序列是魔术的，我们知道s[i] * i的和肯定也是n。使用如下方式把这些约束加入我们的模型列表

列表使用冗余约束求解魔术串问题模型magic-series2.mzn

```
int: n;
array[0..n-1] of var 0..n: s;

constraint forall(i in 0..n-1) (
   s[i] = (sum(j in 0..n-1)(bool2int(s[j]=i))));

constraint redundant_constraint(sum(i in 0..n-1)(s[i]) = n);
constraint redundant_constraint(sum(i in 0..n-1)(s[i] * i) = n);

solve satisfy;

output [ "s = ", show(s), ";\n" ] ;
```

像之前那样求解同一个问题

```
$ minizinc --all-solutions --statistics magic-series2.mzn -D "n=16;"
```

产生了同样的输出。但是统计显示只搜索了个决策点。这些冗余约束使得求解器更早地去剪枝。

2.7.4 模型选择

在中有很多方式去给同一个问题建模，尽管其中有些模型或许会比另外的一些模型更自然。不同的模型或许会产生不同的求解效率。更糟糕的是，在不同的求解后端中，不同的模型或许会更好或更差。但是，我们还是可以给出一些关于普遍情况下产生更好的模型的指导

模型之间的选择

一个好的模型倾向于有以下特征

- 更少量的变量，或者至少是更少量的没有被其他变量功能上定义的变量。
- 更小的变量定义域范围
- 模型的约束定义更简洁或者直接
- 尽可能地使用全局约束

实际情况中，所有这些都需要通过检查这个模型的搜索到底多有效率来断定模型好坏。通常除了用实验之外，我们很难判断搜索是否高效。

观察如下问题，我们要找寻到n这n个数字的排列，使得相邻数字的差值也形成一个到n的排列。列表中给出了一个用直观的方式来建模此问题的模型。注意变量u被变量x功能性定义。所以最差情况下的搜索空间是n^n。

列表对于prob007所有间隔系列问题的模型allinterval.mzn

```
include "alldifferent.mzn";

int: n;

array[1..n] of var 1..n: x;      % 数字的序列
array[1..n-1] of var 1..n-1: u;  % 差的序列

constraint alldifferent(x);
constraint alldifferent(u);
constraint forall(i in 1..n-1)(u[i] = abs(x[i+1] - x[i]));

solve :: int_search(x, first_fail, indomain_min, complete)
      satisfy;
output ["x = ",show(x),"\n"];
```

在这个模型中，数组x代表n个数字的排序。约束自然地可用alldifferent来表示。

求解模型

```
$ minizinc --solver g12fd --all-solutions --statistics allinterval.mzn -D "n=10;"
```

在个决策点和秒的时间内找到了所有的解。

另外一个模型是使用数组y，其中y[i]代表数字i在序列中的位置。我们同时也使用变量v来建模表示差的位置。v[i]表示了绝对值差i在序列出现的位置。如果y[i]和y[j]差别为一，其中j > i，则代表了它们的位置是相邻的。所以v[j-i]被约束为两个位置中最早的那个。我们可以给这个模型加入两个冗余约束：由于我们知道差值n-1肯定会产生，我们就可以推断出和n的位置必须是相邻的abs(y[1] - y[n]) = 1。同时也告诉我们差值n-1的位置就是在y[1]和y[n]中的最早的那个位置，即v[n-1] = min(y[1], y[n])。有了这些之后，我们可以建模此问题为。输出语句从位置数组y里重现了原本的序列x。

列表中全区间序列问题prob007的一个逆向模型。allinterval2.mzn

```
include "alldifferent.mzn";

int: n;

array[1..n] of var 1..n: y;  % 每个数值的位置
```

```
array[1..n-1] of var 1..n-1: v; % 差值i的位置

constraint alldifferent(y);
constraint alldifferent(v);
constraint forall(i,j in 1..n where i < j)(
                    (y[i] - y[j] = 1 -> v[j-i] = y[j]) /\
                (y[j] - y[i] = 1 -> v[j-i] = y[i])
           );

constraint abs(y[1] - y[n]) = 1 /\ v[n-1] = min(y[1], y[n]);

solve :: int_search(y, first_fail, indomain_min, complete)
      satisfy;

output [ "x = [",] ++
       [ show(i) ++ if j == n then "]\n;" else ", " endif
         | j in 1..n, i in 1..n where j == fix(y[i]) ];
```

逆向模型跟初始模型有同样的变量和定义域大小。但是相对于给变量x和u的关系建模，逆向模型使用了一个更加非直接的方式来给变量y和v的关系建模。所以我们或许期望初始模型更好些。

命令

```
$ minizinc --solver g12fd --all-solutions --statistics allinterval2.mzn -D "n=10;"
```

在个决策点和秒内找到了所有的解。有趣的是，尽管这个模型不是简洁的，在变量y上搜索比在变量x上搜索更加有效率。简洁的缺乏意味着尽管搜索需要更少的决策点，但是在时间上实质上会更慢。

2.7.5 多重建模和连通

当我们对同一个问题有两个模型时，由于每个模型可以给求解器不同的信息，通过把两个模型中的变量系到一起从而同时使用两个模型或许对我们是有帮助的。

列表中全区间序列问题prob007的一个双重模型。allinterval3.mzn

```
include "inverse.mzn";

int: n;

array[1..n] of var 1..n: x;  % 数值的序列
array[1..n-1] of var 1..n-1: u;  % 差值的序列

constraint forall(i in 1..n-1)(u[i] = abs(x[i+1] - x[i]));

array[1..n] of var 1..n: y;  % 每个数值的位置
array[1..n-1] of var 1..n-1: v; % 差值的位置

constraint inverse(x,y);
constraint inverse(u,v);

constraint abs(y[1] - y[n]) = 1 /\ v[n-1] = min(y[1], y[n]);

solve :: int_search(y, first_fail, indomain_min, complete)
      satisfy;
```

```
output ["x = ",show(x),"\n"];
```

列表给出了一个结合allinterval.mzn和allinterval2.mzn特征的双重模型。模型的开始来自于allinterval.mzn。我们接着介绍了来自于allinterval2.mzn中的变量y和v。我们使用全局约束inverse来把变量绑到一起：inverse(x,y)约束y为x的逆向函数（反之亦然），即，x[i] = j <-> y[j] = i。列表中给出了它的一个定义。这个模型没有包含把变量y和v关联起来的约束，它们是冗余的（实际上是传播冗余）。所以它们不会给基于传播的求解器多余的信息。alldifferent也不见了。原因是它们被逆向约束变得冗余了（传播冗余）。唯一的约束是关于变量x和u和关系的约束以及y和v的冗余约束。

列表全局约束inverse的一个定义inverse.mzn

```
predicate inverse(array[int] of var int: f,
                  array[int] of var int: invf) =
    forall(j in index_set(invf))(invf[j] in index_set(f)) /\
    forall(i in index_set(f))(
        f[i] in index_set(invf) /\
        forall(j in index_set(invf))(j == f[i] <-> i == invf[j])
    );
```

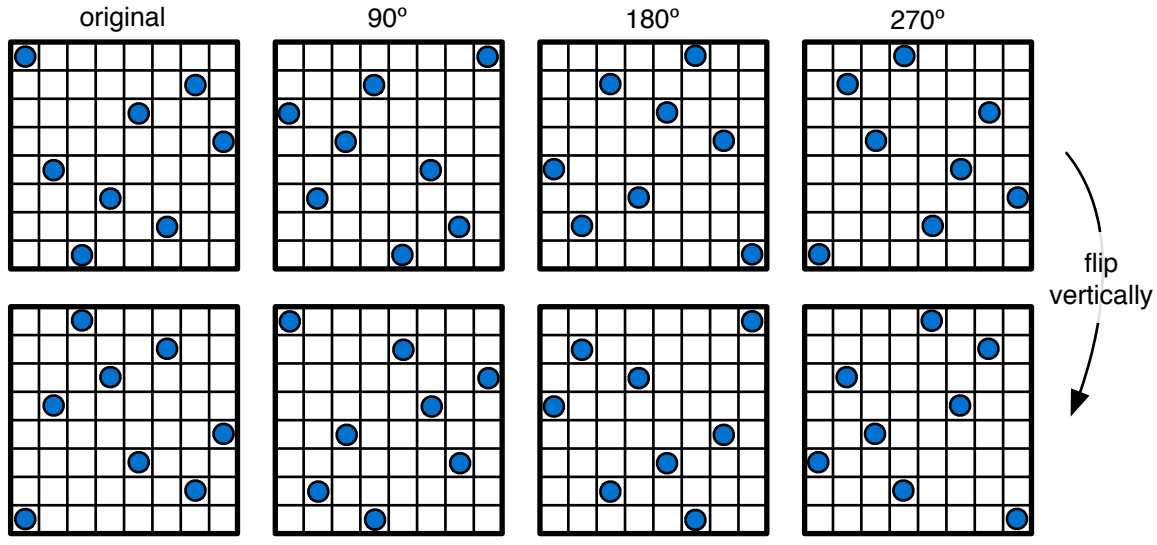
双重模型的一个优点是我们可以有更多的定义不同搜索策略的视角。运行双重模型，

```
$ minizinc --solver g12fd --all-solutions --statistics allinterval3.mzn -D "n=10;"
```

注意它使用逆向模型的搜索策略，标记变量y，在决策点和秒内找到了所有的解。注意标记变量x来运行同样的模型，需要个决策点和秒。

2.7.6 对称

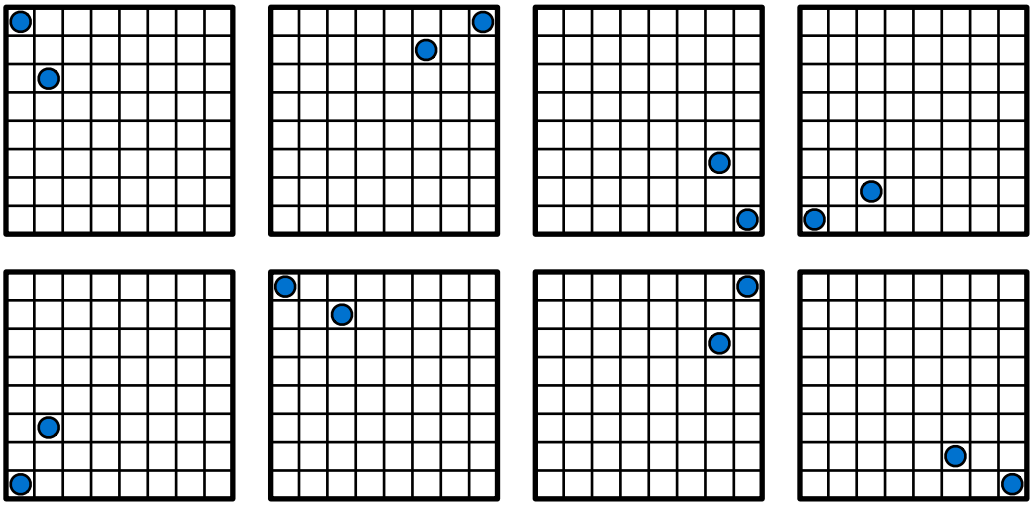
对称在约束满足和优化问题中是常见的。我们可以再次通过列表来看一下这个问题。在图棋盘的左上方展示了一个皇后问题的解标记为“”。剩下的棋盘展示了个解的对称版本旋转度度和度还有垂直翻转。



图皇后问题解的对称变化

如果我们想要穷举皇后问题的所有解很明显我们需要通过穷举彼此之间不对称的解为求解器省下一些工作然后生成这些的对称版本。这是我们想要在约束模型中摆脱对称的一个理由。另外一个更重要的理由是我们的求解器也可能会**探索非解状态的对称版本**

举个例子一个典型的约束求解器可能会尝试把第列皇后放在第行上这是可以的然后尝试把第列的皇后放到第行上。这在第一眼看是没有违反任何约束的。然而这种设置不能被完成成为一个解求解器会在一些搜索之后发现这一点图在左上方的棋盘展示了这种设置。现在没有任何东西会阻止求解器去尝试比如在图最低一行的左边第二种设置。其中第列的皇后仍然在第行而第列的皇后放在第行。于是即便是搜索一个解求解器可能需要探索很多它已经看到并证明不可满足状态的对称状态



图皇后问题不可满足约束的部分解的对称版本

2.7.6.1 静态的对称性破缺

解决对称问题的建模技巧叫做对称性破缺在它的最简单形式里需要在模型中加入约束使一个不完整的赋值的所有对称变换去除掉而保留一个。这些约束称作静态的对称性破缺约束。

对称性破缺背后基本的想法是加入顺序。举个例子我们可以通过简单地加入约束使第一列的皇后必须在棋盘的上半部分从而去除掉所有棋盘垂直翻转的。

```
constraint q[1] <= n div 2;
```

请相信以上约束会去除掉在图中所有对称变换的一半。为了去除所有对称我们需要更多工作。

当我们把所有对称都表示成数组变量的排列一组字典顺序约束可以用于破坏所有对称。举个例子如果数组变量名为x而翻转数组是这个问题的一种对称那么以下约束可以破坏那种对称

```
constraint lex_lesseq(x, reverse(x));
```

那么二维数组又怎么样呢字典顺序同样适用我们只需要把数组转换成一维的举个例子下面的约束破坏了沿着其中一个对角线翻转数组的对称性注意到第二个生成式里对换的数组下标

```
array[1..n,1..n] of var int: x;
constraint lex_lesseq([ x[i,j] | i,j in 1..n ], [ x[j,i] | i,j in 1..n ]);
```

字典排序约束的好处在于我们可以加入多个同时破坏几个对称性而不需要它们互相干扰只要我们保持第一个参数中的顺序一致即可。

对于皇后问题很不幸的是这个技巧不能马上适用因为又一些对称不能被描述成数组q的排列。克服这个问题的技巧是把皇后问题表示成布尔变量。对于每个棋盘的每个格子布尔变量表示是否有一个皇后在上面。现在所有的对称性都可以表示成这个数组的排列。因为主要的皇后问题的主要约束在整型数组q里面更加容易表达我们只需要两个模型都用起来然后在它们之间加入连通约束。正如多重建模和连通中解释的一样。

加入布尔变量连通约束和对称性破缺约束的完整模型展示在列表里面。我们可以做一些小实验来检查它是否成功的破坏所有对称性。尝试用不断增加的n运行模型比如从到数一下解的个数比如使用求解器的-s标志或者选择中”“和”“。你应该可以获得以下数列的解。你可以搜索来校验这个序列。

列表皇后问题对称性破缺的部分模型nqueens_sym.mzn

```
% 映射每一个位置 i,j到一个布尔变量上来表示在i,j上是否有一个皇后
array[1..n,1..n] of var bool: qb;

% 连通约束
constraint forall (i,j in 1..n) ( qb[i,j] <-> (q[i]=j) );

% 字典排序对称性破缺
constraint
    lex_lesseq(array1d(qb), [ qb[j,i] | i,j in 1..n ])
/\  lex_lesseq(array1d(qb), [ qb[i,j] | i in reverse(1..n), j in 1..n ])
/\  lex_lesseq(array1d(qb), [ qb[j,i] | i in 1..n, j in reverse(1..n) ])
/\  lex_lesseq(array1d(qb), [ qb[i,j] | i in 1..n, j in reverse(1..n) ])
/\  lex_lesseq(array1d(qb), [ qb[j,i] | i in reverse(1..n), j in 1..n ])
/\  lex_lesseq(array1d(qb), [ qb[i,j] | i,j in reverse(1..n) ])
/\  lex_lesseq(array1d(qb), [ qb[j,i] | i,j in reverse(1..n) ])
;
```

2.7.6.2 其他对称的例子

许多其他问题都有内在的对称性破坏这些对称性常常会令求解表现不一样。以下是一些常见的例子

装箱问题当尝试把物品装入箱子时任意两个有相同容量的箱子都是对称的

涂色问题当尝试为一个图涂色使得所有相邻的节点都有不同的颜色时我们通常用整型变量对颜色建模。但是对颜色的任意排列都是一种合法的涂色方案。

车辆路线问题如果任务是给顾客分配一些车辆任何两辆有相同容量的车可能是对称的这跟装箱问题是相似的

排班时间表问题两个有相同能力的职员可能是可以相互交换的就像两个有相同容量或者设备的的房间一样

CHAPTER 2.8

在 MiniZinc 中对布尔可满足性问题建模

可以被用来给布尔可满足性问题建模，这种问题的变量被限制为是布尔型bool。可以使用有效率的布尔可满足性求解器来有效地解决求得的模型。

2.8.1 整型建模

很多时候，尽管我们想要使用一个布尔可满足性求解器，我们可能也需要给问题的整数部分建模。有三种通用的方式使用布尔型变量对定义域为范围$0 \ldots m$内的整型变量I建模，其中$m = 2^k - 1$。

二元：I被表示为k个二元变量$i_0, \ldots, i_{k-1}$，其中$I = 2^{k-1}i_{k-1} + 2^{k-2}i_{k-2} + \cdots + 2i_1 + i_0$。在中，这可表示为

```
array[0..k-1]  of var bool: i;
var 0..pow(2,k)-1: I = sum(j in 0..k-1)(bool2int(i[j])*pow(2,j));
```

一元I被表示为m个二元变量$i_1, \ldots, i_m$且$i = \sum_{j=1}^{m} \texttt{bool2int}(i_j)$。由于在一元表示中有大量的冗余表示，我们通常要求$i_j \rightarrow i_{j-1}, 1 < j \leq m$。在中，这可表示为

```
array[1..m]  of var bool: i;
constraint forall(j in 2..m)(i[j] -> i[j-1]);
var 0..m: I = sum(j in 1..m)(bool2int(i[j]);
```

值其中I被表示为$m + 1$个二元变量$i_0, \ldots, i_m$，其中$i = k \Leftrightarrow i_k$并且$i_0, \ldots, i_m$中最多有一个为真。在中，这可表示为

```
array[0..m]  of var bool: i;
constraint sum(j in 0..m)(bool2int(i[j]) == 1;
var 0..m: I;
constraint foall(j in 0..m)(I == j <-> i[j]);
```

每种表示都有其优点和缺点。这取决于模型中需要对整数做什么样的操作，而这些操作在哪一种表示上更为方便。

2.8.2 非等式建模

接下来，让我们考虑如何为一个拉丁方问题建模。一个拉丁方问题是在$n \times n$个网格上放置$1..n$之间的数值使得每个数在每行每列都仅出现一次。图列表中给出了拉丁方问题的的一个整数模型。

列表拉丁方问题的整数模型latin.mzn

```
int: n; % 拉丁方的大小
array[1..n,1..n] of var 1..n: a;

include "alldifferent.mzn";
constraint forall(i in 1..n)(
             alldifferent(j in 1..n)(a[i,j]) /\
             alldifferent(j in 1..n)(a[j,i])
           );
solve satisfy;
output [ show(a[i,j]) ++ if j == n then "\n" else " " endif |
         i in 1..n, j in 1..n ];
```

整型变量直接的唯一的约束实际上是非等式，而它在约束alldifferent中被编码。数值表示是表达非等式的最佳方式。图列表给出了一个关于拉丁方问题的只含有布尔型变量的模型。注意每个整型数组元素a[i,j]被替换为一个布尔型数组。我们使用谓词exactlyone来约束每个数值在每行每列都仅出现一次，也用来约束有且仅有一个布尔型变量对应于整型数组元素a[i,j]为真。

列表拉丁方问题的布尔型模型latinbool.mzn

```
int: n; % 拉丁方的大小
array[1..n,1..n,1..n] of var bool: a;

predicate atmostone(array[int] of var bool:x) =
          forall(i,j in index_set(x) where i < j)(
            (not x[i] \/ not x[j]));
predicate exactlyone(array[int] of var bool:x) =
          atmostone(x) /\ exists(x);

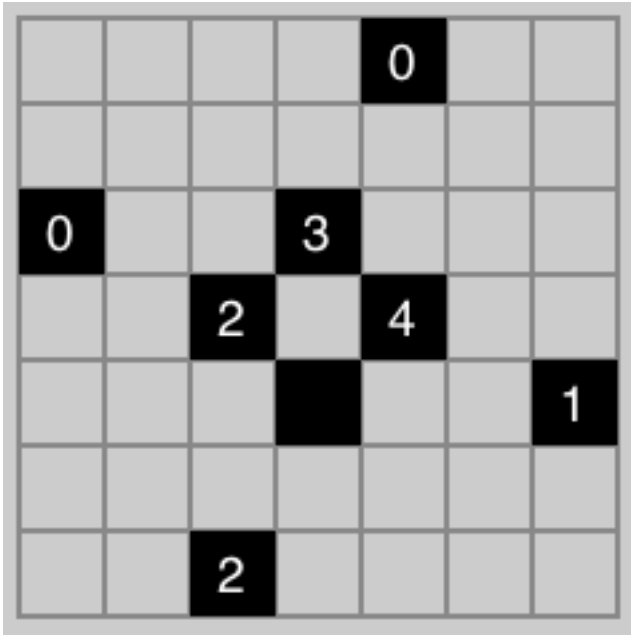
constraint forall(i,j in 1..n)(
             exactlyone(k in 1..n)(a[i,j,k]) /\
             exactlyone(k in 1..n)(a[i,k,j]) /\
             exactlyone(k in 1..n)(a[k,i,j])
           );
solve satisfy;
output [ if fix(a[i,j,k]) then
            show(k) ++ if j == n then "\n" else " " endif
         else "" endif | i,j,k in 1..n ];
```

2.8.3 势约束建模

让我们来看下如何对点灯游戏建模。这个游戏由一个矩形网格组成，每个网格为空白或者被填充。每个被填充的方格可能包含到之间的数字，或者没有数字。我们的目标是放置灯泡在空白网格使得

> 每个空白的网格是“被照亮的”，也就是说它可以透过一行没有被打断的空白网格看到光亮。
>
> 任何两个灯泡都不能看到彼此。
>
> 一个有数值的填充的网格相邻的灯泡个数必须等于这个网格中的数值。

图给出了点灯游戏的一个例子以及图给出了它的解。



图点灯游戏的一个例子展示

这个问题很自然地可以使用布尔型变量建模。布尔型变量用来决定哪一个网格包含有一个点灯以及哪一个没有。同时我们也有一些作用于填充的网格上的整数算术运算要考虑。

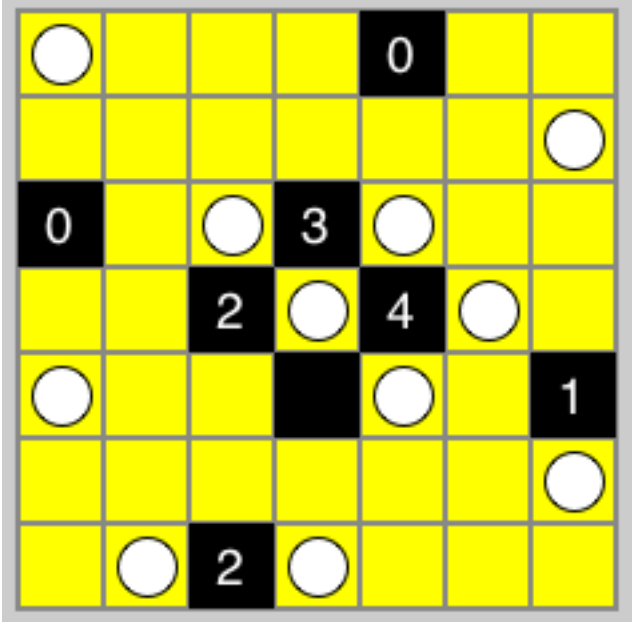
列表点灯游戏的模型lightup.mzn

```
int: h; set of int: H = 1..h; % 板高度
int: w; set of int: W = 1..w; % 板宽度
array[H,W] of -1..5: b;       % 板
int: E = -1;                  % 空白格
set of int: N = 0..4;         % 填充并含有数字的网格
int: F =  5;                  % 填充但不含有数字的网格

% 位置 (i1,j1) 对位置 (i2,j2) 可见
test visible(int: i1, int: j1, int: i2, int: j2) =
    ((i1 == i2) /\ forall(j in min(j1,j2)..max(j1,j2))(b[i1,j] == E))
 \/ ((j1 == j2) /\ forall(i in min(i1,i2)..max(i1,i2))(b[i,j1] == E));

array[H,W] of var bool: l; % is there a light

% 填充的网格没有灯泡
```



图点灯游戏的完整的解

```
constraint forall(i in H, j in W where b[i,j] != E)(l[i,j] == false);
% lights next to filled numbered square agree

constraint forall(i in H, j in W where b[i,j] in N)(
    bool_sum_eq([ l[i1,j1] | i1 in i-1..i+1, j1 in j-1..j+1 where
                             abs(i1 - i) + abs(j1 - j) == 1 /\
                             i1 in H /\ j1 in W ], b[i,j]));
% 每个空白网格是被照亮的
constraint forall(i in H, j in W where b[i,j] == E)(
            exists(j1 in W where visible(i,j,i,j1))(l[i,j1]) \/
            exists(i1 in H where visible(i,j,i1,j))(l[i1,j])
          );
% 任何两个灯泡看不到彼此
constraint forall(i1,i2 in H, j1,j2 in W where
                  (i1 != i2 \/ j1 != j2) /\ b[i1,j1] == E
                  /\ b[i2,j2] == E /\ visible(i1,j1,i2,j2))(
            not l[i1,j1] \/ not l[i2,j2]
          );

solve satisfy;
output [ if b[i,j] != E then show(b[i,j])
         else if fix(l[i,j]) then "L" else "." endif
         endif ++ if j == w then "\n" else " " endif |
         i in H, j in W];
```

图列表中给出了这个问题的一个模型。图图中给出的问题的数据文件在图列表中给出。

列表点灯游戏的图中实例的数据文件

```
h = 7;
w = 7;
b = [| -1,-1,-1,-1, 0,-1,-1
     | -1,-1,-1,-1,-1,-1,-1
```

```
      |  0,-1,-1, 3,-1,-1,-1
      | -1,-1, 2,-1, 4,-1,-1
      | -1,-1,-1, 5,-1,-1, 1
      | -1,-1,-1,-1,-1,-1,-1
      |  1,-1, 2,-1,-1,-1,-1 |];
```

模型利用了一个布尔型求和谓词

```
predicate bool_sum_eq(array[int] of var bool:x, int:s);
```

使得一个布尔型数组的和等于一个固定的整数。多种方式都能使用布尔型变量给约束建模。

加法器网络：我们可以使用包含加法器的一个网络给布尔型总和建立一个二元布尔型表达式

排序网络：我们可以通过使用一个排序网络去分类布尔型数组来创建一个布尔型总和的一元表达式

二元决策图：我们可以创建一个二维决策图（）来编码势约束。

列表使用二元加法器网络表示势约束bboolsum.mzn

```
% 布尔型变量x的总和 = s
predicate bool_sum_eq(array[int] of var bool:x, int:s) =
     let { int: c = length(x) } in
     if s < 0 then false
     elseif s == 0 then
          forall(i in 1..c)(x[i] == false)
     elseif s < c then
          let { % cp = number of bits required for representing 0..c
                int: cp = floor(log2(int2float(c))),
                % z is sum of x in binary
                array[0..cp] of var bool:z  } in
          binary_sum(x, z) /\
          % z == s
          forall(i in 0..cp)(z[i] == ((s div pow(2,i)) mod 2 == 1))
     elseif s == c then
          forall(i in 1..c)(x[i] == true)
     else false endif;

include "binarysum.mzn";
```

列表创建二元求和网络的代码binarysum.mzn

```
% 位 x 的的总和 = 二元表示的 s。
%           s[0], s[1], ..., s[k] 其中 2^k >= length(x) > 2^(k-1)
predicate binary_sum(array[int] of var bool:x,
                     array[int] of var bool:s)=
     let { int: l = length(x) } in
     if l == 1 then s[0] = x[1]
     elseif l == 2 then
            s[0] = (x[1] xor x[2]) /\ s[1] = (x[1] /\ x[2])
     else let { int: ll = (l div 2),
                array[1..ll] of var bool: f = [ x[i] | i in 1..ll ],
                array[1..ll] of var bool: t = [x[i]| i in ll+1..2*ll],
```

```
                var bool: b = if ll*2 == l then false else x[l] endif,
                int: cp = floor(log2(int2float(ll))),
                array[0..cp] of var bool: fs,
                array[0..cp] of var bool: ts } in
                binary_sum(f, fs) /\ binary_sum(t, ts) /\
                binary_add(fs, ts, b, s)
     endif;

% 把两个二元数值 x 和 y 加起来，位 ci 用来表示进位，来得到二元 s
predicate binary_add(array[int] of var bool: x,
                     array[int] of var bool: y,
                     var bool: ci,
                     array[int] of var bool: s) =
    let { int:l = length(x),
          int:n = length(s), } in
    assert(l == length(y),
          "length of binary_add input args must be same",
    assert(n == l \/ n == l+1, "length of binary_add output " ++
                          "must be equal or one more than inputs",
    let { array[0..l] of var bool: c } in
    full_adder(x[0], y[0], ci, s[0], c[0]) /\
    forall(i in 1..l)(full_adder(x[i], y[i], c[i-1], s[i], c[i])) /\
    if n > l then s[n] = c[l] else c[l] == false endif ));

predicate full_adder(var bool: x, var bool: y, var bool: ci,
                     var bool: s, var bool: co) =
          let { var bool: xy = x xor y } in
          s = (xy xor ci) /\ co = ((x /\ y) \/ (ci /\ xy));
```

我们可以使用图列表给出的二元加法器网络代码实现bool_sum_eq。图列表中定义的谓词binary_sum创建了一个x总和的二维表示法。它把列表分为两部分，把每一部分分别加起来得到它们的一个二元表示，然后用binary_add把这两个二元数值加起来。如果x列大小是奇数，则最后一位被保存起来作为二元加法时的进位来使用。

列表使用二元加法器网络表示势约束uboolsum.mzn

```
% 布尔型变量 x的总和 = s
predicate bool_sum_eq(array[int] of var bool:x, int:s) =
     let { int: c = length(x) } in
     if s < 0 then false
     elseif s == 0 then forall(i in 1..c)(x[i] == false)
     elseif s < c then
          let { % cp = nearest power of 2 >= c
                int: cp = pow(2,ceil(log2(int2float(c)))),
                array[1..cp] of var bool:y, % y is padded version of x
                array[1..cp] of var bool:z  } in
          forall(i in 1..c)(y[i] == x[i]) /\
          forall(i in c+1..cp)(y[i] == false) /\
          oesort(y, z) /\ z[s] == true /\ z[s+1] == false
     elseif s == c then forall(i in 1..c)(x[i] == true)
     else false endif;

include "oesort.mzn";
```

列表奇偶归并排序网络oesort.mzn

```
%% odd-even sort
%% y is the sorted version of x, all trues before falses
predicate oesort(array[int] of var bool:x, array[int] of var bool:y)=
  let { int: c = card(index_set(x)) } in
  if c == 1 then x[1] == y[1]
  elseif c == 2 then comparator(x[1],x[2],y[1],y[2])
  else
    let {
      array[1..c div 2] of var bool:xf = [x[i] | i in 1..c div 2],
      array[1..c div 2] of var bool:xl = [x[i] | i in c div 2 +1..c],
      array[1..c div 2] of var bool:tf,
      array[1..c div 2] of var bool:tl } in
    oesort(xf,tf) /\ oesort(xl,tl) /\ oemerge(tf ++ tl, y)
  endif;

%% odd-even merge
%% y is the sorted version of x, all trues before falses
%% assumes first half of x is sorted, and second half of x
predicate oemerge(array[int] of var bool:x, array[int] of var bool:y)=
  let { int: c = card(index_set(x)) } in
  if c == 1 then x[1] == y[1]
  elseif c == 2 then comparator(x[1],x[2],y[1],y[2])
  else
    let { array[1..c div 2] of var bool:xo =
            [ x[i] | i in 1..c where i mod 2 == 1],
          array[1..c div 2] of var bool:xe =
            [ x[i] | i in 1..c where i mod 2 == 0],
          array[1..c div 2] of var bool:to,
          array[1..c div 2] of var bool:te } in
      oemerge(xo,to) /\ oemerge(xe,te) /\
      y[1] = to[1] /\
      forall(i in 1..c div 2 -1)(
        comparator(te[i],to[i+1],y[2*i],y[2*i+1])) /\
      y[c] = te[c div 2]
  endif;

% comparator o1 = max(i1,i2), o2 = min(i1,i2)
predicate comparator(var bool:i1,var bool:i2,var bool:o1,var bool:o2)=
          (o1 = (i1 \/ i2)) /\ (o2 = (i1 /\ i2));
```

我们可以使用图列表中给出的一元排序网络代码来实现bool_sum_eq。势约束通过扩展输入x长度为的次幂，然后使用奇偶归并排序网络给得到的位排序来实现。奇偶归并排序工作方式在图列表中给出，它递归地把输入列表拆为两部分，给每一部分排序，然后再把有序的两部分归并起来。

列表使用二元加法器网络表示势约束bddsum.mzn

```
% the sum of booleans x = s
predicate bool_sum_eq(array[int] of var bool:x, int:s) =
     let { int: c = length(x),
           array[1..c] of var bool: y = [x[i] | i in index_set(x)]
     } in
```

```
      rec_bool_sum_eq(y, 1, s);

predicate rec_bool_sum_eq(array[int] of var bool:x, int: f, int:s) =
      let { int: c = length(x) } in
      if s < 0 then false
      elseif s == 0 then
           forall(i in f..c)(x[i] == false)
      elseif s < c - f + 1 then
           (x[f] == true /\ rec_bool_sum_eq(x,f+1,s-1)) \/
           (x[f] == false /\ rec_bool_sum_eq(x,f+1,s))
      elseif s == c - f + 1 then
           forall(i in f..c)(x[i] == true)
      else false endif;
```

我们可以使用图列表中给出的二元决策图代码来实现bool_sum_eq。势约束被分为两种情况：或者第一个元素x[1]为true并且剩下位的总和是s-1，或者x[1]为false并且剩下位的总和是s。它的效率的提高依赖于去除共同子表达式来避免产生太多的相同的约束。

CHAPTER 2.9

FlatZinc 和展平

约束求解器不会直接支持模型为了运行一个模型它被翻译成一个的简单子集叫反映了大多数约束求解器只会求解具有$ex\bar{i}stsc_1 \wedge \cdots \wedge c_m$的满足问题或者有$zc_1 \wedge \cdots \wedge c_m$的优化问题其中$c_i$是基本的约束而$z$是一个具有某些限定形式的整型或者浮点表达式

minizinc工具包含了编译器它可以用一个模型和数据文件来创建一个展平后的模型它等价于给定数据的模型表达为之前提到的受限制的形式通常来说构建一个给求解器的模型是对用户隐藏的不过你也可以通过以下命令查看结合数据data.dzn来展平一个模型model.mzn的结果

```
minizinc -c model.mzn data.dzn
```

这会创建一个模型叫model.fzn

在这一章中我们探索把翻译成的过程

2.9.1 展平表达式

底层求解器的限制意味着复杂的表达式需要被展平为内部不具有更复杂结构项的基本约束的合取式

思考以下保证两个在长方形箱子的两个圆不会重叠的模型

列表两个不会重叠的圆的模型cnonoverlap.mzn

```
float: width;          % 包含圆的长方形的宽
float: height;         % 包含圆的长方形的高
float: r1;
var r1..width-r1:  x1; % (x1,y1) 是第一个圆的中心
var r1..height-r1: y1;
float: r2;
var r2..width-r2:  x2; % (x2,y2) 是第二个圆的中心
var r2..height-r2: y2;
                       % 中心之间至少有r1 + r2的距离
constraint (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) >= (r1+r2)*(r1+r2);
solve satisfy;
```

2.9.1.1 简化和求值

给定数据文件

```
width = 10.0;
height = 8.0;
r1 = 2.0;
r2 = 3.0;
```

转换到首先通过替换所有参数为它们的值来简化模型然后求出所有取值已经固定的表达式的值在这步简化后参数的值将不再需要一个例外是大型数组的参数值如果它们被适用多于一次那么为了避免重复大的表达式这个参数会被保留

在简化后列表的模型的变量和参数声明部分变为

```
var 2.0 .. 8.0: x1;
var 2.0 .. 6.0: y1;
var 3.0 .. 7.0: x2;
var 3.0 .. 5.0: y2;
```

2.9.1.2 定义子表达式

现在没有约束求解器可以直接处理像在列表里复杂的约束表达式作为替代我们可以命名表达式中的每一个子表达式我们创建约束来构建及代表子表达式的数值让我们来看看约束表达式的子表达式(x1 - x2)是一个子表达式如果我们将它命名为FLOAT01我们可以定义它为constraint FLOAT01 = x1 - x2;注意到这个表达式只在模型中出现两次我们只需要构建这个值一次然后我们可以重复使用它这就是共同子表达式消除子表达式(x1 - x2)*(x1 - x2)可以命名为FLOAT02而我们可以定义它为constraint FLOAT02 = FLOAT01 * FLOAT01;我们可以命名constraint FLOAT03 = y1 - y2;和constraint FLOAT04 = FLOAT03 * FLOAT03;最后constraint FLOAT05 = FLOAT02 * FLOAT04;不等约束本身则变成constraint FLOAT05 >= 25.0;因为(r1+r2)*(r1 + r2)计算出结果为25.0于是这个约束被展平为

```
constraint FLOAT01 = x1 - x2;
constraint FLOAT02 = FLOAT01 * FLOAT01;
constraint FLOAT03 = y1 - y2;
constraint FLOAT04 = FLOAT03 * FLOAT03;
constraint FLOAT05 = FLOAT02 * FLOAT04;
constraint FLOAT05 >= 25.0
```

2.9.1.3 FlatZinc 约束形式

展平的最后步骤是把约束的形式转换成标准的形式它总是以$p(a_1,\ldots,a_n)$的形式出现其中的p是某个基础约束$a_1,\ldots,a_n$是参数尝试使用最少的不同约束形式所以FLOAT01 = x1 - x2首先尝试重写为FLOAT01 + x2 = x1然后使用float_plus输出基本的约束得出的约束形式如下

```
constraint float_plus(FLOAT01, x2, x1);
constraint float_plus(FLOAT03, y2, y1);
constraint float_plus(FLOAT02, FLOAT04, FLOAT05);
constraint float_times(FLOAT01, FLOAT01, FLOAT02);
constraint float_times(FLOAT03, FLOAT03, FLOAT04);
```

2.9.1.4 边界分析

我们仍然缺少一项声明引入的变量FLOAT01…FLOAT05这些可以被声明为var float不过为了令求解器的任务更简单尝试通过简单的分析确定新引入变量的上界和下界比如因为FLOAT01 = x1 - x2和2.0 ≤x1≤ 8.0还有3.0 ≤x2≤ 7.0所以可以得出−5.0 ≤FLOAT01≤ 5.0然后我们可以看到−25.0 ≤FLOAT02≤ 25.0尽管注意到如果我们发现相乘实际上是平方我们可以给出更精确的边界0.0 ≤FLOAT02≤ 25.0

细心的读者会发现在定义子表达式和约束形式里约束的展平形式的一点不同在后面没有非等约束因为一元的不等式可以完全被一个变量的边界表示出来不等关系可以令FLOAT05的下界变为25.0然后这会变得冗余最后列表的展平后形式是

```
% 变量
var 2.0 .. 8.0: x1;
var 2.0 .. 6.0: y1;
var 3.0 .. 7.0: x2;
var 3.0 .. 5.0: y2;
%
var -5.0..5.0:   FLOAT01;
var -25.0..25.0: FLOAT02;
var -3.0..3.0:   FLOAT03;
var -9.0..9.0:   FLOAT04;
var 25.0..34.0:  FLOAT05;
% 约束
constraint float_plus(FLOAT01, x2, x1);
constraint float_plus(FLOAT03, y2, y1);
constraint float_plus(FLOAT02, FLOAT04, FLOAT05);
constraint float_times(FLOAT01, FLOAT01, FLOAT02);
constraint float_times(FLOAT03, FLOAT03, FLOAT04);
%
solve satisfy;
```

2.9.1.5 目标函数

就像展平约束一样展平最小化和最大化目标函数跟其他表达式一样目标表达式被展平时创建一个变量在输出求解项永远是单一变量看一下例子表达式

2.9.2 线性表达式

约束的一个最重要的而广泛用于建模的形式是线性约束

$$a_1x_1 + \cdots + a_nx_n \begin{matrix} = \\ \leq \\ < \end{matrix} a_0$$

其中a_i是整数或者浮点数约束而x_i是整数或者浮点数变量它们非常有表达能力同时也是整数线性规划约束求解器唯一支持的形式从到的翻译器尝试创建线性约束而不是把线性约束变成许多子表达式

列表说明线性约束展平的模型linear.mzn

```
int:       d = -1;
var 0..10: x;
var -3..6: y;
var 3..8:  z;
constraint 3*x - y + x * z <= 19 + d * (x + y + z) - 4*d;
solve satisfy;
```

考虑在列表中的模型这里并没有为所有子表达式$3 * x3 * x - yx * z3 * x - y + x * zx + y + z$ $d * (x + y + z)19 + d * (x + y + z)$和$19 + d * (x + y + z) - 4 * d$创建一个变量这里的翻译在创建一个约束时会尝试创建一个尽可能记录大部分原约束内容的线性约束

展平会创建线性表达式并将其视为一个单位而不视为每个字表达是构建中间变量这也使创建的表达式简单化从约束中抽取出线性表达式为

```
var 0..80: INT01;
constraint 4*x + z + INT01 <= 23;
constraint INT01 = x * z;
```

注意到非线性表达式$x \times z$是如何被抽取出来作为一个新的子表达式并赋予名字的与此同时剩下的项会被收集起来从而使每个变量只出现一次的确变量y的项被移除了

最后每个约束被写到形式从而得到

```
var 0..80: INT01;
constraint int_lin_le([1,4,1],[INT01,x,z],23);
constraint int_times(x,z,INT01);
```

2.9.3 展开表达式

大多数的模型需要创建一些基于输入数据的约束通过数组类型列表和列生成解析还有聚合函数来支持这些模型

考虑以下从生产调度例子列表中出现的聚合函数表达式

```
int: mproducts = max (p in Products)
                     (min (r in Resources where consumption[p,r] > 0)
                                  (capacity[r] div consumption[p,r]));
```

由于这用到生成器语法我们可以把它重写成可以被编译器处理的相等的形式

```
int: mproducts = max([ min [ capacity[r] div consumption[p,r]
                           | r in Resources where consumption[p,r] > 0])
                     | p in Products]);
```

给定数据

```
nproducts = 2;
nresources = 5;
capacity = [4000, 6, 2000, 500, 500];
consumption= [| 250, 2, 75, 100, 0,
              | 200, 0, 150, 150, 75 |];
```

这首先创建p = 1的数组

```
[ capacity[r] div consumption[p,r]
                           | r in 1..5 where consumption[p,r] > 0]
```

也就是[16, 3, 26, 5]然后计算最小值为它之后为p = 2建立相同的数组[20, 13, 3, 6]并计算最小的数值为然后创建数组[3, 3]并计算最大值为在里面没有mproducts的表示这种通过计算数值

的方法会被用来代替mproducts

在约束模型中最常见的聚合表达式是forall表达式会被展开成为多个约束

考虑以下在例子列表中使用预设的分解alldifferent出现的片段

```
array[1..8] of var 0..9: v = [S,E,N,D,M,O,R,Y];
constraint forall(i,j in 1..8 where i < j)(v[i] != v[j])
```

forall表达式为每一对需要满足$i < j$的i, j创建一个约束所以创建

```
constraint v[1] != v[2]; % S != E
constraint v[1] != v[3]; % S != N
...
constraint v[1] != v[8]; % S != Y
constraint v[2] != v[3]; % E != N
...
constraint v[7] != v[8]; % R != Y
```

在中形成

```
constraint int_neq(S,E);
constraint int_neq(S,N);
...
constraint int_neq(S,Y);
constraint int_neq(E,N);
...
constraint int_neq(R,Y);
```

注意到临时的数组变量v[i]是如何在的输出中被原来的变量替换的

2.9.4 数组

一维变量在中可以有任意的下标只要它们是相邻的整数在所有数组都被1..l标注其中l是数组的长度这意味着数组查询时需要被转换成下标的形式

考虑以下模型来使用m个砝码来平衡一个长为2 * l2的跷跷板其中上面有一个重为cw小孩

```
int: cw;                                % 小孩重量
int: l2;                                % 一半跷跷板长度
int: m;                                 % 1kg砝码的数量
array[-l2..l2] of var 0..max(m,cw): w; % 在每个点的重量
var -l2..l2: p;                         % 孩子的位置
constraint sum(i in -l2..l2)(i * w[i]) = 0; % 平衡
constraint sum(i in -l2..l2)(w[i]) = m + cw; % 所有使用的砝码
constraint w[p] = cw;                   % 孩子在位置p
solve satisfy;
```

给定cw = 2l2 = 2和m = 3展开可以产生约束

```
array[-2..2] of var 0..3: w;
var -2..2: p
constraint -2*w[-2] + -1*w[-1] + 0*w[0] + 1*w[1] + 2*w[2] = 0;
constraint w[-2] + w[-1] + w[0] + w[1] + w[2] = 5;
```

```
constraint w[p] = 2;
```

不过坚持w数组从下标开始这意味着我们需要重写所有数组获取来使用新的下标数值对于固定值数组查找这很简单对于变量值数组查找我们可能需要创建一个新的变量以上公式的结果为

```
array[1..5] of var 0..3: w;
var -2..2: p
var 1..5: INT01;
constraint -2*w[1] + -1*w[2] + 0*w[3] + 1*w[4] + 2*w[5] = 0;
constraint w[1] + w[2] + w[3] + w[4] + w[5] = 5;
constraint w[INT01] = 2;
constraint INT01 = p + 3;
```

最后我们重写约束成的形式注意到变量数组下标查找的形式是如何映射到array_var_int_element上的

```
array[1..5] of var 0..3: w;
var -2..2: p
var 1..5: INT01;
constraint int_lin_eq([2, 1, -1, -2], [w[1], w[2], w[4], w[5]], 0);
constraint int_lin_eq([1, 1, 1, 1, 1], [w[1],w[2],w[3],w[4],w[5]], 5);
constraint array_var_int_element(INT01, w, 2);
constraint int_lin_eq([-1, 1], [INT01, p], -3);
```

支持多维数组但是目前来说只支持单维度数组这意味着多维度数组必须映射到单维度数组上而且多维度数组访问必须映射到单维度数组访问

考虑在有限元平面模型列表的等式约束

```
set of int: HEIGHT = 0..h;
set of int: CHEIGHT = 1..h-1;
set of int: WIDTH = 0..w;
set of int: CWIDTH = 1..w-1;
array[HEIGHT,WIDTH] of var float: t; % 在点 (i,j) 处的温度
var float: left;   % 左侧温度
var float: right;  % 右侧温度
var float: top;    % 顶部温度
var float: bottom; % 底部温度

% 拉普拉斯方程: 每一个内部点温度是它相邻点的平均值
constraint forall(i in CHEIGHT, j in CWIDTH)(
              4.0*t[i,j] = t[i-1,j] + t[i,j-1] + t[i+1,j] + t[i,j+1]);
```

假设w = 4和h = 4这会创建约束

```
array[0..4,0..4] of var float: t; % temperature at point (i,j)
constraint 4.0*t[1,1] = t[0,1] + t[1,0] + t[2,1] + t[1,2];
constraint 4.0*t[1,2] = t[0,2] + t[1,1] + t[2,2] + t[1,3];
constraint 4.0*t[1,3] = t[0,3] + t[1,2] + t[2,3] + t[1,4];
constraint 4.0*t[2,1] = t[1,1] + t[2,0] + t[3,1] + t[2,2];
constraint 4.0*t[2,2] = t[1,2] + t[2,1] + t[3,2] + t[2,3];
constraint 4.0*t[2,3] = t[1,3] + t[2,2] + t[3,3] + t[2,4];
constraint 4.0*t[3,1] = t[2,1] + t[3,0] + t[4,1] + t[3,2];
```

```
constraint 4.0*t[3,2] = t[2,2] + t[3,1] + t[4,2] + t[3,3];
constraint 4.0*t[3,3] = t[2,3] + t[3,2] + t[4,3] + t[3,4];
```

个元素的维数组被转换成一维数组而且下标也会相应改变所以`[i,j]`下标会变成`[i * 5 + j + 1]`

```
array [1..25] of var float: t;
constraint 4.0*t[7] = t[2] + t[6] + t[12] + t[8];
constraint 4.0*t[8] = t[3] + t[7] + t[13] + t[9];
constraint 4.0*t[9] = t[4] + t[8] + t[14] + t[10];
constraint 4.0*t[12] = t[7] + t[11] + t[17] + t[13];
constraint 4.0*t[13] = t[8] + t[12] + t[18] + t[14];
constraint 4.0*t[14] = t[9] + t[13] + t[19] + t[15];
constraint 4.0*t[17] = t[12] + t[16] + t[22] + t[18];
constraint 4.0*t[18] = t[13] + t[17] + t[23] + t[19];
constraint 4.0*t[19] = t[14] + t[18] + t[24] + t[20];
```

2.9.5 具体化

模型包含了只有变量和参数声明和一系列原始的约束所以当我们在用布尔连接符而不是析取式来建模时需要进行一些处理处理使用连接符而不只是析取式来构建的复杂公式其核心的方法是具体化具体化一个约束c创建新的约束等价于$b \leftrightarrow c$即如果约束满足则布尔变量b的值是`true`否则为`false`

当我们有能力具体化约束对待复杂公式的方式跟数学表达式并无不同我们为子表达式创建了一个名称和一个展平的约束来约束子表达式的数值

考虑以下任务调度例子列表中出现在约束表达式

```
constraint %% 保证任务之间没有重叠
    forall(j in 1..tasks) (
        forall(i,k in 1..jobs where i < k) (
            s[i,j] + d[i,j] <= s[k,j] \/
            s[k,j] + d[k,j] <= s[i,j]
    ) );
```

对于数据文件

```
jobs = 2;
tasks = 3;
d = [| 5, 3, 4 | 2, 6, 3 |]
```

然后展开过程生成

```
constraint s[1,1] + 5 <= s[2,1] \/ s[2,1] + 2 <= s[1,1];
constraint s[1,2] + 3 <= s[2,2] \/ s[2,2] + 6 <= s[1,2];
constraint s[1,3] + 4 <= s[2,3] \/ s[2,3] + 3 <= s[1,3];
```

具体化在析取式中出现的约束创建新的布尔变量来定义每个表达式的数值

```
array[1..2,1..3] of var 0..23: s;
constraint BOOL01 <-> s[1,1] + 5 <= s[2,1];
constraint BOOL02 <-> s[2,1] + 2 <= s[1,1];
```

```
constraint BOOL03 <-> s[1,2] + 3 <= s[2,2];
constraint BOOL04 <-> s[2,2] + 6 <= s[1,2];
constraint BOOL05 <-> s[1,3] + 4 <= s[2,3];
constraint BOOL06 <-> s[2,3] + 3 <= s[1,3];
constraint BOOL01 \/ BOOL02;
constraint BOOL03 \/ BOOL04;
constraint BOOL05 \/ BOOL06;
```

每个基础的约束现在会映射到形式下注意到二维数组s是如何映射到一维数组里面的

```
array[1..6] of var 0..23: s;
constraint int_lin_le_reif([1, -1], [s[1], s[4]], -5, BOOL01);
constraint int_lin_le_reif([-1, 1], [s[1], s[4]], -2, BOOL02);
constraint int_lin_le_reif([1, -1], [s[2], s[5]], -3, BOOL03);
constraint int_lin_le_reif([-1, 1], [s[2], s[5]], -6, BOOL04);
constraint int_lin_le_reif([1, -1], [s[3], s[6]], -4, BOOL05);
constraint int_lin_le_reif([-1, 1], [s[3], s[6]], -3, BOOL06);
constraint array_bool_or([BOOL01, BOOL02], true);
constraint array_bool_or([BOOL03, BOOL04], true);
constraint array_bool_or([BOOL05, BOOL06], true);
```

`int_lin_le_reif`是线性约束`int_lin_le`的具体化形式

大多数基本约束$p(\bar{x})$有一个具体化形式$(\bar{x}, b)$它利用最后额外的参数b来定义一个约束$b \leftrightarrow p(\bar{x})$定义像`int_plus`和`int_plus`的函数式关系的基本约束不需要支持具体化反而有结果的函数的等式被具体化了

具体化的另外一个重要作用出现在当我们使用强制转换函数`bool2int`可能是显式地或者隐式地把布尔表达式使用成整数表达式使用平整过程将创建一个布尔变量来保存一个布尔表达式参数以及一个整型变量限制到`0..1`来保存这个数值

考虑列表中的魔术序列问题

```
int: n;
array[0..n-1] of var 0..n: s;

constraint forall(i in 0..n-1) (
   s[i] = (sum(j in 0..n-1)(bool2int(s[j]=i))));
```

给定`n = 2`展开创造了

```
constraint s[0] = bool2int(s[0] = 0) + bool2int(s[1] = 0);
constraint s[1] = bool2int(s[0] = 1) + bool2int(s[1] = 1);
```

和展平创造了

```
constraint BOOL01 <-> s[0] = 0;
constraint BOOL03 <-> s[1] = 0;
constraint BOOL05 <-> s[0] = 1;
constraint BOOL07 <-> s[1] = 1;
constraint INT02 = bool2int(BOOL01);
constraint INT04 = bool2int(BOOL03);
constraint INT06 = bool2int(BOOL05);
```

```
constraint INT08 = bool2int(BOOL07);
constraint s[0] = INT02 + INT04;
constraint s[1] = INT06 + INT08;
```

最后的形式是

```
var bool: BOOL01;
var bool: BOOL03;
var bool: BOOL05;
var bool: BOOL07;
var 0..1: INT02;
var 0..1: INT04;
var 0..1: INT06;
var 0..1: INT08;
array [1..2] of var 0..2: s;
constraint int_eq_reif(s[1], 0, BOOL01);
constraint int_eq_reif(s[2], 0, BOOL03);
constraint int_eq_reif(s[1], 1, BOOL05);
constraint int_eq_reif(s[2], 1, BOOL07);
constraint bool2int(BOOL01, INT02);
constraint bool2int(BOOL03, INT04);
constraint bool2int(BOOL05, INT06);
constraint bool2int(BOOL07, INT08);
constraint int_lin_eq([-1, -1, 1], [INT02, INT04, s[1]], 0);
constraint int_lin_eq([-1, -1, 1], [INT06, INT08, s[2]], 0);
solve satisfy;
```

2.9.6 谓词

支持许多不同求解器的的一个重要的因素是全局约束还有真正的约束可以根据不同的求解器专业化

每一个求解器生命一个谓词有时会但有时并不会提供具体的定义举个例子一个求解器有一个内建的全局`alldifferent`谓词会包含定义

```
predicate alldifferent(array[int] of var int:x);
```

在全局约束库中同时一个求解器预设的分解会有定义

```
predicate alldifferent(array[int] of var int:x) =
    forall(i,j in index_set(x) where i < j)(x[i] != x[j]);
```

谓词调用$p(\bar{t})$在平整时首先为每个参数项t_i创建对应变量v_i如果谓词没有定义我们只需要使用创建的参数$p(\bar{v})$来调用谓词如果一个谓词有一个定义$p(\bar{x}) = \phi(\bar{x})$然后我们将用谓词的定义来替换这个谓词调用$p(\bar{t})$当中形式参数被替换为对应的参数变量即$\phi(\bar{v})$注意到如果一个谓词调用$p(\bar{t})$出现在具体化位置而且它没有定义我们则检查我们适用这个谓词的具体化版本$(\bar{x}, b)$

考虑在例子列表中`alldifferent`约束的

```
constraint alldifferent([S,E,N,D,M,O,R,Y]);
```

如果这个求解器有一个内建的`alldifferent`我们只需要为这个参数创建一个新的变量然后在调用时替换它

```
array[1..8] of var 0..9: v = [S,E,N,D,M,O,R,Y];
constraint alldifferent(v);
```

注意到边界分析尝试在新的数组变量上找到一个紧的边界建造这个数组参数的理由就是如果我们中使用相同的数组两次求解器不会创建它两次在这种情况下因为它不是使用两次后面的转换会把v替换成它的定义

如果求解器使用预设的定义alldifferent呢然后变量v会正常地定义谓词调用会被替换为一个当中变量被重命名的版本其中v替换了形式参数x结果的程序是

```
array[1..8] of var 0..9: v = [S,E,N,D,M,O,R,Y];
constraint forall(i,j in 1..8 where i < j)(v[i] != v[j])
```

我们可以在展开表达式中看到

考虑到以下约束其中alldifferent在一个具体化位置出现

```
constraint alldifferent([A,B,C]) \/ alldifferent([B,C,D]);
```

如果求解器有alldifferent的具体化形式这将会被展平为

```
constraint alldifferent_reif([A,B,C],BOOL01);
constraint alldifferent_reif([B,C,D],BOOL02);
constraint array_bool_or([BOOL01,BOOL02],true);
```

适用这个预设的分解谓词替换会首先创建

```
array[1..3] of var int: v1 = [A,B,C];
array[1..3] of var int: v2 = [B,C,D];
constraint forall(i,j in 1..3 where i<j)(v1[i] != v1[j]) \/
           forall(i,j in 1..3 where i<j)(v2[i] != v2[j]);
```

它最终会展平成形式

```
constraint int_neq_reif(A,B,BOOL01);
constraint int_neq_reif(A,C,BOOL02);
constraint int_neq_reif(B,C,BOOL03);
constraint array_bool_and([BOOL01,BOOL02,BOOL03],BOOL04);
constraint int_neq_reif(B,D,BOOL05);
constraint int_neq_reif(C,D,BOOL06);
constraint array_bool_and([BOOL03,BOOL05,BOOL06],BOOL07);
constraint array_bool_or([BOOL04,BOOL07],true);
```

注意到共同子表达式消除是如何利用具体化不等式B != C的虽然有一个更好的转换把共同约束提升到最顶层的合取式中

2.9.7 Let 表达式

表达式是中可用于引入新的变量的非常强大的工具在展平时表达式被转换成变量和约束声明这个的关系语义意味着这些约束必须像在第一个包含的布尔表达式中出现

表达式的一个重要特征是每一次它们被使用时它们都创建新的变量

考虑一下展平的代码

```
constraint even(u) \/ even(v);
predicate even(var int: x) =
          let { var int: y } in x = 2 * y;
```

首先谓词调用被他们的定义取代

```
constraint (let { var int: y} in u = 2 * y) \/
           (let { var int: y} in v = 2 * y);
```

然后变量会另外被重命名

```
constraint (let { var int: y1} in u = 2 * y1) \/
           (let { var int: y2} in v = 2 * y2);
```

最后变量声明会被抽取到第一层

```
var int: y1;
var int: y2;
constraint u = 2 * y1 \/ v = 2 * y2;
```

一旦表达式被清除我们可以像之前那样展平

记住表达式可以定义新引入的变量对某些参数的确需要这样做这些隐式地定义了必须满足的约束

考虑婚礼座位问题列表的复杂的目标函数

```
solve maximize sum(h in Hatreds)(
      let {  var Seats: p1 = pos[h1[h]],
             var Seats: p2 = pos[h2[h]],
             var 0..1: same = bool2int(p1 <= 6 <-> p2 <= 6) } in
      same * abs(p1 - p2) + (1-same) * (abs(13 - p1 - p2) + 1));
```

为了简介我们假设只使用前两个相互敌视的人所以

```
set of int: Hatreds = 1..2;
array[Hatreds] of Guests: h1 = [groom, carol];
array[Hatreds] of Guests: h2 = [clara, bestman];
```

展平的第一步是展开sum表达式给定为了简洁我们保留客人名字和参数Seats在实际中他们会被他们的定义取代

```
solve maximize
      (let { var Seats: p1 = pos[groom],
             var Seats: p2 = pos[clara],
             var 0..1: same = bool2int(p1 <= 6 <-> p2 <= 6) } in
       same * abs(p1 - p2) + (1-same) * (abs(13 - p1 - p2) + 1))
      +
      (let { var Seats: p1 = pos[carol],
             var Seats: p2 = pos[bestman],
             var 0..1: same = bool2int(p1 <= 6 <-> p2 <= 6) } in
       same * abs(p1 - p2) + (1-same) * (abs(13 - p1 - p2) + 1));
```

然后每一个在表达式的新变量会被分别命名为

```
solve maximize
      (let { var Seats: p11 = pos[groom],
             var Seats: p21 = pos[clara],
             var 0..1: same1 = bool2int(p11 <= 6 <-> p21 <= 6) } in
       same1 * abs(p11 - p21) + (1-same1) * (abs(13 - p11 - p21) + 1))
      +
      (let { var Seats: p12 = pos[carol],
             var Seats: p22 = pos[bestman],
             var 0..1: same2 = bool2int(p12 <= 6 <-> p22 <= 6) } in
       same2 * abs(p12 - p22) + (1-same2) * (abs(13 - p12 - p22) + 1));
```

在表达式的变量会被抽取到第一层并且定义约束会被抽取到正确的层在这里是最顶层

```
var Seats: p11;
var Seats: p21;
var 0..1: same1;
constraint p12 = pos[clara];
constraint p11 = pos[groom];
constraint same1 = bool2int(p11 <= 6 <-> p21 <= 6);
var Seats p12;
var Seats p22;
var 0..1: same2;
constraint p12 = pos[carol];
constraint p22 = pos[bestman];
constraint same2 = bool2int(p12 <= 6 <-> p22 <= 6) } in
solve maximize
      same1 * abs(p11 - p21) + (1-same1) * (abs(13 - p11 - p21) + 1))
      +
      same2 * abs(p12 - p22) + (1-same2) * (abs(13 - p12 - p22) + 1));
```

现在我们已经构成不需要使用表达式的等价的代码和展平可以正常进行

为了说明没有出现在最顶层的表达式的情况看看以下模型

```
var 0..9: x;
constraint x >= 1 -> let { var 2..9: y = x - 1 } in
                     y + (let { var int: z = x * y } in z * z) < 14;
```

我们抽取变量定义到最顶层约束到第一个围住的布尔语境这里是蕴含的右手边

```
var 0..9: x;
var 2..9: y;
var int: z;
constraint x >= 1 -> (y = x - 1 /\ z = x * y /\ y + z * z < 14);
```

注意到如果我们知道定义一个变量的等式的真值不会为假我们可以抽取它到最顶层这通常可以是求解大幅加快

对于上面的例子因为y的值域对于x - 1并不够大所以约束y = x - 1可能失败不过约束z = x * y不可以实际上边界分析会给予z足够大的边界来包含x * y所有的可能值一个更好的展平可以给出

```
var 0..9: x;
var 2..9: y;
var int: z;
constraint z = x * y;
constraint x >= 1 -> (y = x - 1 /\ y + z * z < 14);
```

现在编译器通过总是使引入变量声明的边界足够大它应该可以包含所有它定义的表达式的值然后在正确的语境中为表达式加入边界约束在上面的例子中这个结果是

```
var 0..9: x;
var -1..8: y;
var -9..72: z;
constraint y = x - 1;
constraint z = x * y;
constraint x >= 1 -> (y >= 2 /\ y + z * z < 14);
```

这个转换可以使求解更加高效因为变量的所有可能的复杂计算并没有被具体化

这种方法的另外一个原因是在引入变量出现在取反语境的时候它也可以被使用只要它们有一个定义考虑一下与之前相似的这个例子

```
var 0..9: x;
constraint (let { var 2..9: y = x - 1 } in
            y + (let { var int: z = x * y } in z * z) > 14) -> x >= 5;
```

这个表达式出现在否定语境中不过每个引入变量都被定义了展平后的代码是

```
var 0..9: x;
var -1..8: y;
var -9..72: z;
constraint y = x - 1;
constraint z = x * y;
constraint (y >= 2 /\ y + z * z > 14) -> x >= 5;
```

注意到作为对比的消除方法不能给出一个正确的转换

```
var 0..9: x;
var 2..9: y;
var int: z;
constraint (y = x - 1 /\ z = x * y /\ y + z * z > 14) -> x >= 5;
```

以上转换对于所有x的可能值给出结果而原来的约束除掉了x = 4的可能性

对于在表达式中的处理跟对定义的变量的处理是相似的你可以认为一个约束等价于定义一个新的布尔变量新的布尔变量定义可以从最顶层中抽取出来而布尔保存在正确的语境下

```
constraint z > 1 -> let { var int: y,
                          constraint (x >= 0) -> y = x,
                          constraint (x < 0)  -> y = -x
                        } in y * (y - 2) >= z;
```

可以处理成

```
constraint z > 1 -> let { var int: y,
                          var bool: b1 = ((x >= 0) -> y = x),
                          var bool: b2 = ((x < 0)  -> y = -x),
                          constraint b1 /\ b2
                        } in y * (y - 2) >= z;
```

然后展平成

```
constraint b1 = ((x >= 0) -> y = x);
constraint b2 = ((x < 0)  -> y = -x);
constraint z > 1 -> (b1 /\ b2 /\ y * (y - 2) >= z);
```

CHAPTER 3.1

The MiniZinc Command Line Tool

minizinc

3.1.1 Basic Usage

minizincmodel.mzndata.dzn

```
int: n;
array[1..n] of var 1..2*n: x;
include "alldifferent.mzn";
constraint alldifferent(x);
solve maximize sum(x);
output ["The resulting values are \(x).\n"];
```

```
n = 5;
```

model.mzndata.dzn

```
$ minizinc --solver gecode model.mzn data.dzn
```

```
The resulting values are [10, 9, 8, 7, 6].
----------
==========
```

-c

```
$ minizinc -c --solver gecode model.mzn data.dzn
```

model.fznmodel.oznmodel.mznmodel.fznminizinc

```
$ minizinc --solver gecode model.fzn
```

```
x = array1d(1..5 ,[10, 9, 8, 7, 6]);
----------
==========
```

model.oznminizinc.oznminizinc

```
$ minizinc --solver gecode model.fzn | minizinc --ozn-file model.ozn
```

节

3.1.2 Adding Solvers

节

3.1.2.1 Configuration files

节.msc

minizinc/solvers//usr/share/minizinc/solversProgram Files\\MiniZinc IDE (bundled)

$HOME/.minizinc/solvers

MZN_SOLVER_PATH:;

mzn_solver_path节

节

minizinc --solvers

3.1.2.2 Configuration for MIP solvers

--cplex-dllcplexXXXX.dllcplexlibcplexXXX.solibcplexXXXX.jnilibXXXXXXX

--gurobi-dllgurobiXX.dllgurobilibgurobiXX.solib

节

3.1.3 Options

minizinc--help

3.1.3.1 General options

minizinc

--help, -h

--version

--solvers

--solver <id>, --solver <solver configuration file>.msc
    --solversmip--solver mip--solver Gecode@6.3.0’ --solver org.gecode.gecode
    节

--help <id>
    --solver

-v, -l, --verbose

--verbose-compilation

-s, --statistics

--compiler-statistics

-c, --compile

--config-dirs

--solvers-json

--param-file <file>
    节

--json-stream

3.1.3.2 Solving options

--help <id><id>

-a, --all-solutions

-n <i>, --num-solutions <i>
    i

-i, --intermediate

-n-i, --no-intermediate

--all-satisfaction

--disable-all-satisfaction

-f, --free-search
    “”

--solver-statistics

--verbose-solving

-p <i>, --parallel <i>
    i

-r <i>, --random-seed <i>
    i

3.1.3.3 Flattener input options

--ignore-stdlib

--instance-check-only

-e, --model-check-only

--model-interface-only

--model-types-only

--no-optimize

-m <file>, --model <file>

-d <file>, --data <file>

--checker <file>, --solution-checker <file>

-D <data>, --cmdline-data <data>

--cmdline-json-data <data>

--stdlib-dir <dir>

-G --globals-dir --mzn-globals-dir <dir>

- --input-from-stdin
    --input-is-flatzinc

-I --search-dir

-D "fMIPdomains=false"

--MIPDMaxIntvEE <n>

--MIPDMaxDensEE <n>

--only-range-domains

--allow-multiple-assignments

--compile-solution-checker <file>.mzc.mzn

Flattener two-pass options

--two-pass

--use-gecode
    _

--shave
    _

--sac
    _

--pre-passes <n>

-O<n>

    _____

Flattener output options

minizinc

--no-output-ozn, -O-

--output-base <name>

--fzn <file>, --output-fzn-to-file <file>

-O, --ozn, --output-ozn-to-file <file>

--keep-paths
    ,

--output-paths

--output-paths-to-file <file>

--output-to-stdout, --output-fzn-to-stdout

--output-ozn-to-stdout

--output-paths-to-stdout

--output-mode <item|dzn|json>

--output-objective

--only-sections <section1[,section2,...]>

--not-sections <section1[,section2,...]>

-Werror

3.1.3.4 Solution output options

minizinc

--ozn-file <file>

-o <file>, --output-to-file <file>

-i <n>, --ignore-lines <n>, --ignore-leading-lines <n>

--soln-sep <s>, --soln-separator <s>, --solution-separator <s>
“_____ “

--soln-comma <s>, --solution-comma <s>

--unsat-msg (--unsatisfiable-msg)
”=====UNSATISFIABLE=====”

--unbounded-msg
”=====UNBOUNDED=====”

--unsatorunbnd-msg
”=====UNSATorUNBOUNDED=====”

--unknown-msg
”=====UNKNOWN=====”

--error-msg
”=====ERROR=====”

--search-complete-msg <msg>
”==========”

--non-unique

-c, --canonicalize

--output-non-canonical <file>

--output-raw <file>
’

--no-output-comments

--output-time

--no-flush-output
’

3.1.4 User Configuration Files

minizincPreferences.json--config-dirs

```
$ minizinc --config-dirs
{
  "globalConfigFile" : "/Applications/MiniZincIDE.app/Contents/Resources/share/
↪minizinc/Preferences.json",
  "userConfigFile" : "/Users/Joe/.minizinc/Preferences.json",
```

```
  "userSolverConfigDir" : "/Users/Joe/.minizinc/solvers",
  "mznStdlibDir" : "/Applications/MiniZincIDE.app/Contents/Resources/share/minizinc"
}
```

mzn_solver_path

mzn_lib_dir

tagDefaults[["cp","org.chuffed.chuffed"],["mip","org.minizinc.gurobi"]]"cp""mip"
""minizinc--solver

solverDefaults[["org.minizinc.gurobi","--gurobi-dll", "/Library/gurobi752/mac64/lib/libguro
' [["org.minizinc.gurobi","--uniform-search",""]]

```
{
  "mzn_solver_path": ["/usr/share/choco"],
  "tagDefaults": [["cp","org.choco-solver.choco"],["mip","org.minizinc.mip.cplex"],[
↪"","org.gecode.gecode"]],
  "solverDefaults": [["org.minizinc.mip.cplex","--cplex-dll","/opt/CPLEX_Studio128/
↪cplex/bin/x86-64_sles10_4.1/libcplex128.so"]]
}
```

' share/minizinc/etc/Library/Preferences

3.1.5 Command-line Parameter Files

minizinc--param-file--param-file.mpc--param-file

minizinc

truefalse

config.mpc

```
{
  "solver": "gecode",
  "cmdline-data": ["x = 1", "y = 2", "z = 3"],
  "-p": 2,
  "output-paths": true
}
```

minizinc --verbose config.mpc -p 4 model.mznminizinc --verbose --param-file config.mpc
-p 4 model.mzn

```
$ minizinc --verbose \
    --solver gecode \
    --cmdline-data "x = 1" \
    --cmdline-data "y = 2" \
    --cmdline-data "z = 3" \
    -p 2 \
    --output-paths \
```

```
    -p 4 \
    model.mzn
```

-p 4-p 2-p 4--cmdline-data--cmdline-data

instance.mpc

```
{
  "model": "model.mzn",
  "data": "data.dzn"
}
```

gecode.mpc

```
{
  "solver": "gecode",
  "intermediate-solutions": true
}
```

minizinc gecode.mpc instance.mpcmodel.dzndata.dzn

CHAPTER 3.2

The MiniZinc IDE

节节

节

3.2.1 Editing files

`.mzn.dzn`

`.mzn.dzn`

3.2.1.1 Editing functions

3.2.1.2 Fonts and dark mode

“”

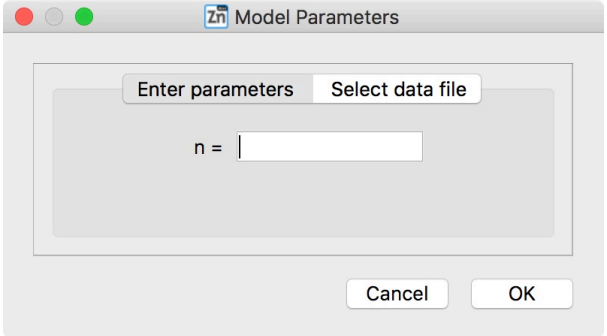
3.2.2 Configuring and Running a Solver

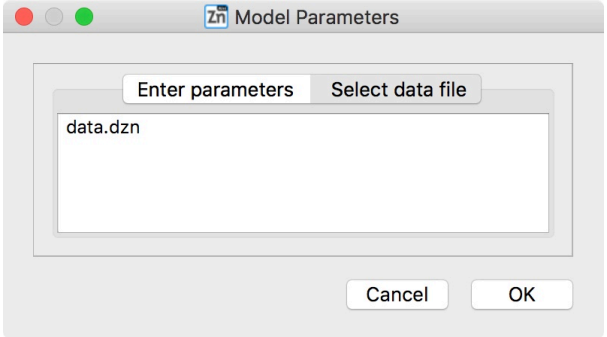
3.2.2.1 Running a model

Ctrl+RCmd+R

,

Ctrl+ECmd+E





3.2.2.2 Solver configurations

’ Ctrl+Shift+CCmd+Shift+C

图1.mpc

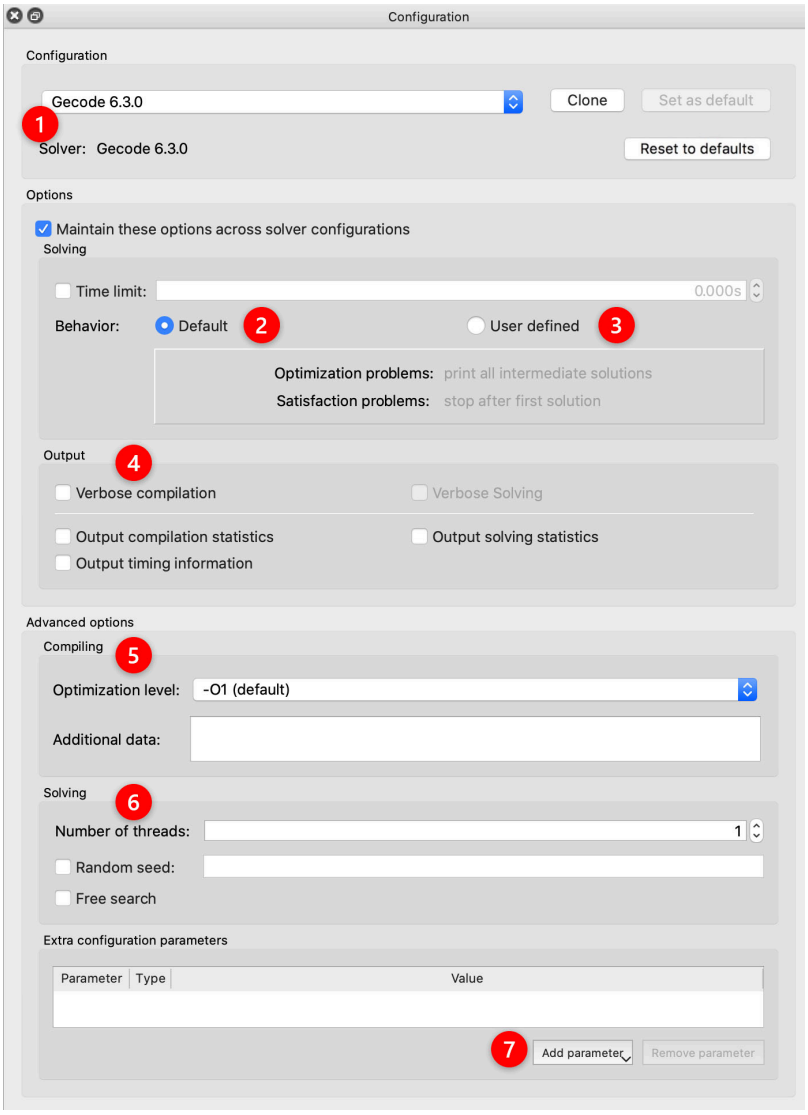
“” 23

4

5.dzn

6

minizinc节7


Configuration
Configuration
Gecode 6.3.0
Clone
Set as default
1
Solver: Gecode 6.3.0
Reset to defaults
Options
Maintain these options across solver configurations
Solving
Time limit:
0.000s
Behavior:
Default
2
User defined
3
Optimization problems: print all intermediate solutions
Satisfaction problems: stop after first solution
Output
4
Verbose compilation
Verbose Solving
Output compilation statistics
Output solving statistics
Output timing information
Advanced options
Compiling
5
Optimization level:
-O1 (default)
Additional data:
Solving
6
Number of threads:
1
Random seed:
Free search
Extra configuration parameters
Parameter
Type
Value
7
Add parameter
Remove parameter

图

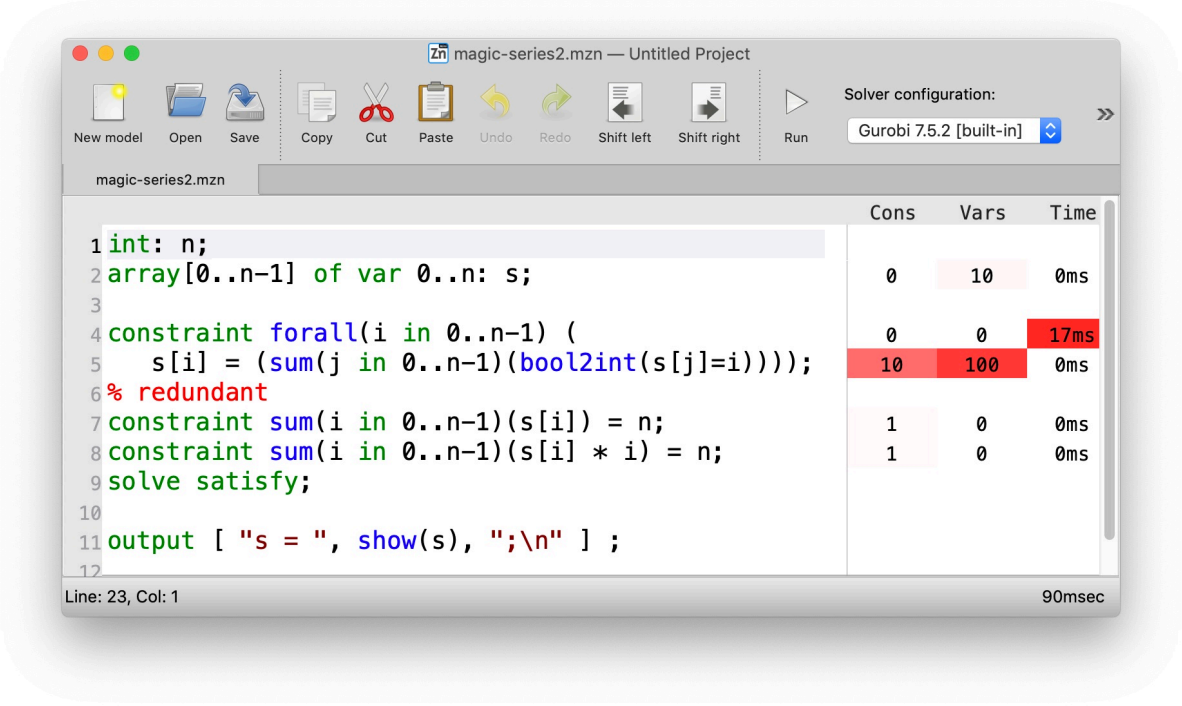
3.2.2.3 Automatic Solution Checking

abc.mznabc.mzcabc.mzc.mzn

3.2.2.4 Compiling a model

3.2.2.5 Profiling a model

列表图 “” “” “” n=10bool2int



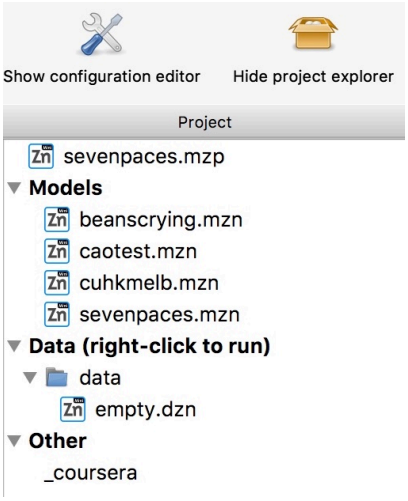
图n=10

节

3.2.3 Working With Projects

Ctrl+Shift+NCmd+Shift+N

图.dzn



图

3.2.4 Submitting Solutions to Online Courses

_mooc

图

3.2.5 Configuration Options

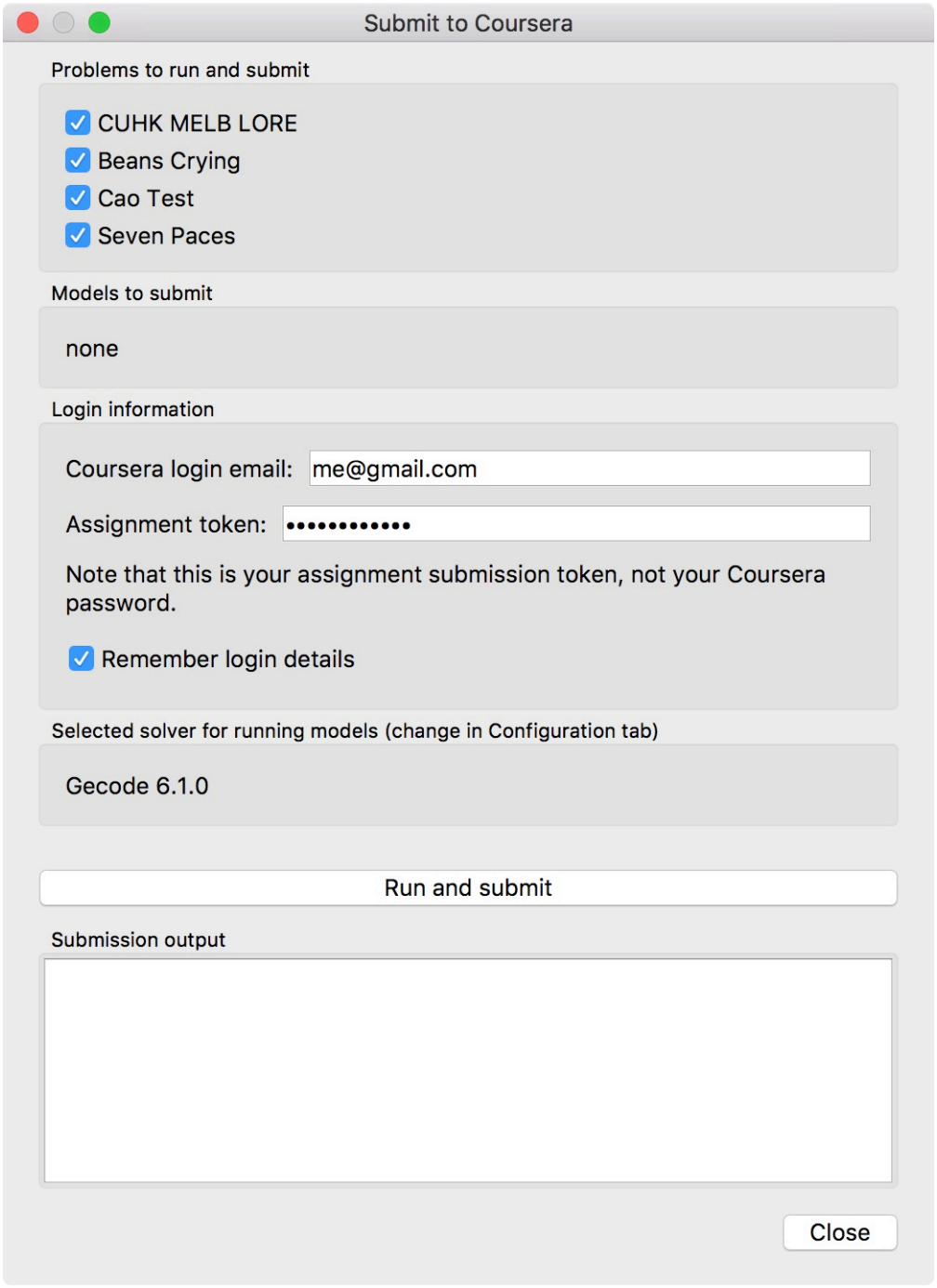
图

图

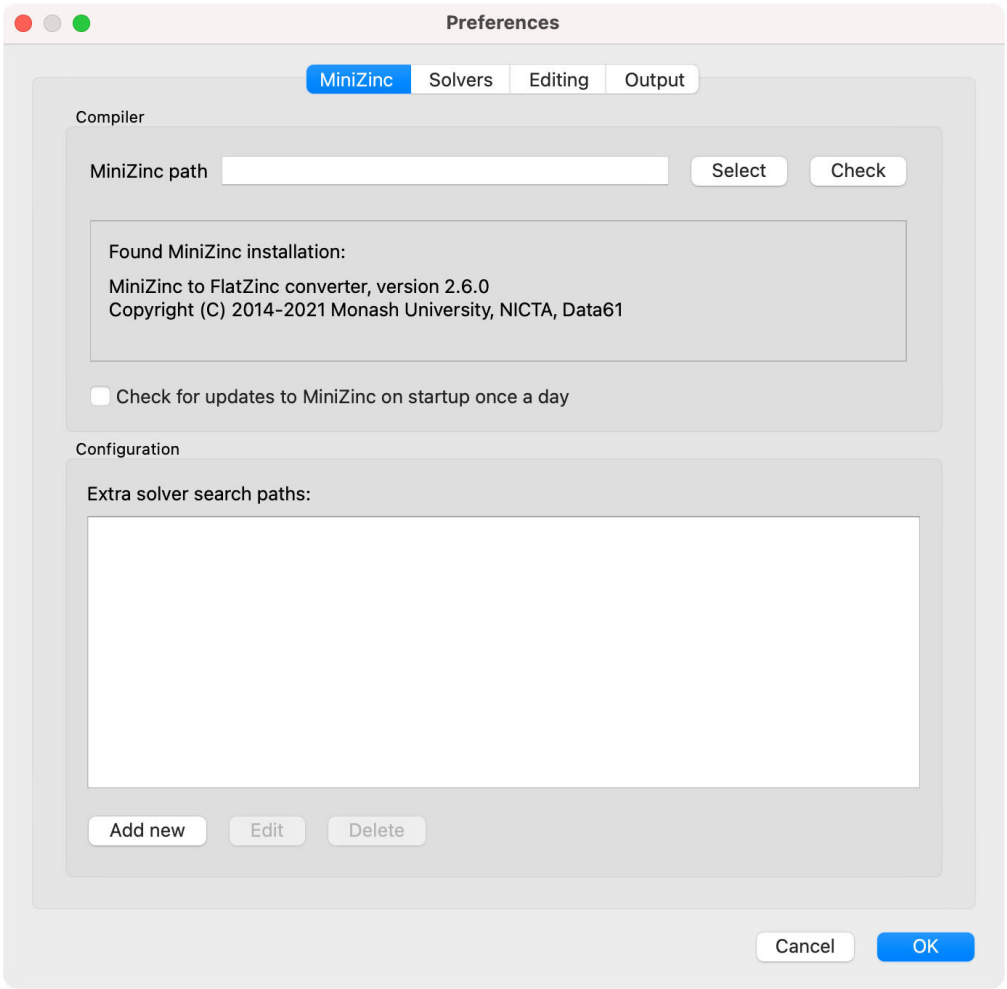
图

fig-ide-preferences-editing

fig-ide-preferences-output


Submit to Coursera
Problems to run and submit
CUHK MELB LORE
Beans Crying
Cao Test
Seven Paces
Models to submit
none
Login information
Coursera login email:
me@gmail.com
Assignment token:
Note that this is your assignment submission token, not your Coursera password.
Remember login details
Selected solver for running models (change in Configuration tab)
Gecode 6.1.0
Run and submit
Submission output
Close

图


Preferences
MiniZinc
Solvers
Editing
Output
Compiler
MiniZinc path
Select
Check
Found MiniZinc installation:
MiniZinc to FlatZinc converter, version 2.6.0
Copyright (C) 2014-2021 Monash University, NICTA, Data61
Check for updates to MiniZinc on startup once a day
Configuration
Extra solver search paths:
Add new
Edit
Delete
Cancel
OK

图

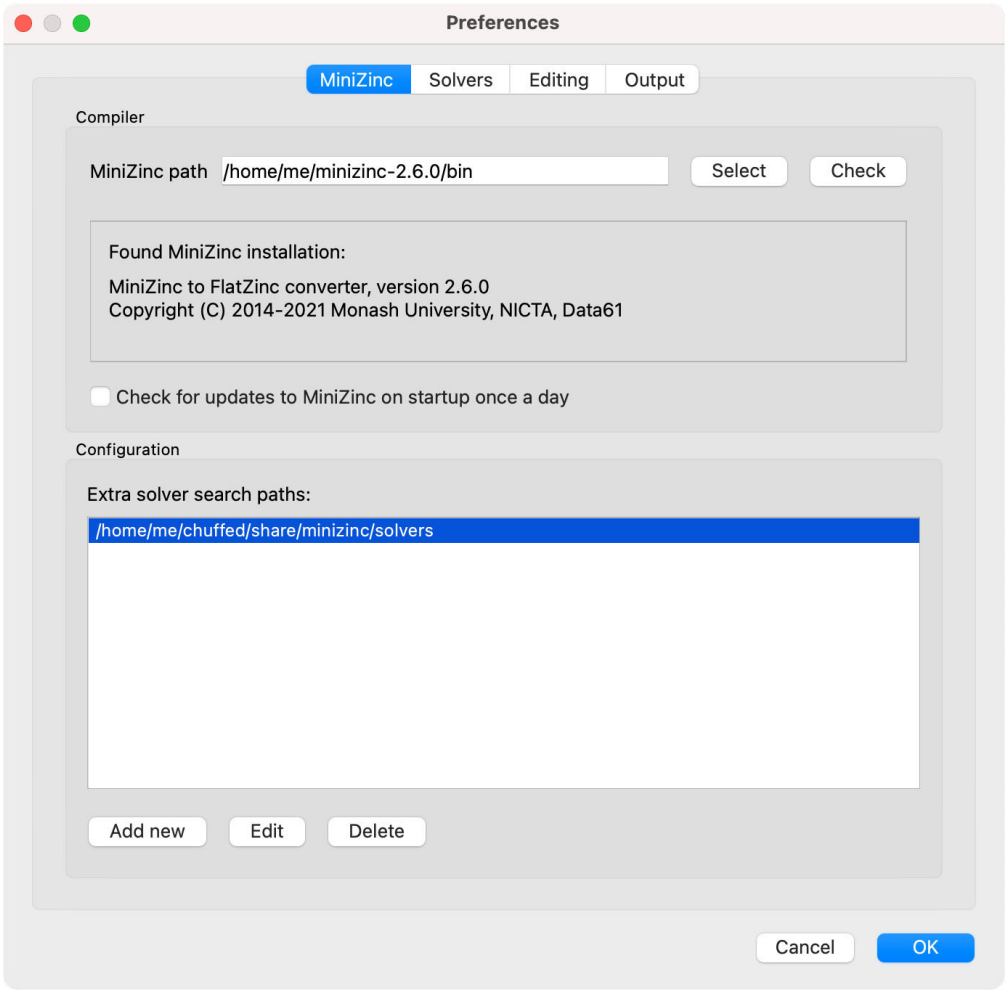
3.2.5.1 Locating the MiniZinc installation

minizinc图minizincPATH

minizincminizincminizinc图/home/me/minizinc-2.6.0/bin

3.2.5.2 Adding Third-Party Solvers

节图



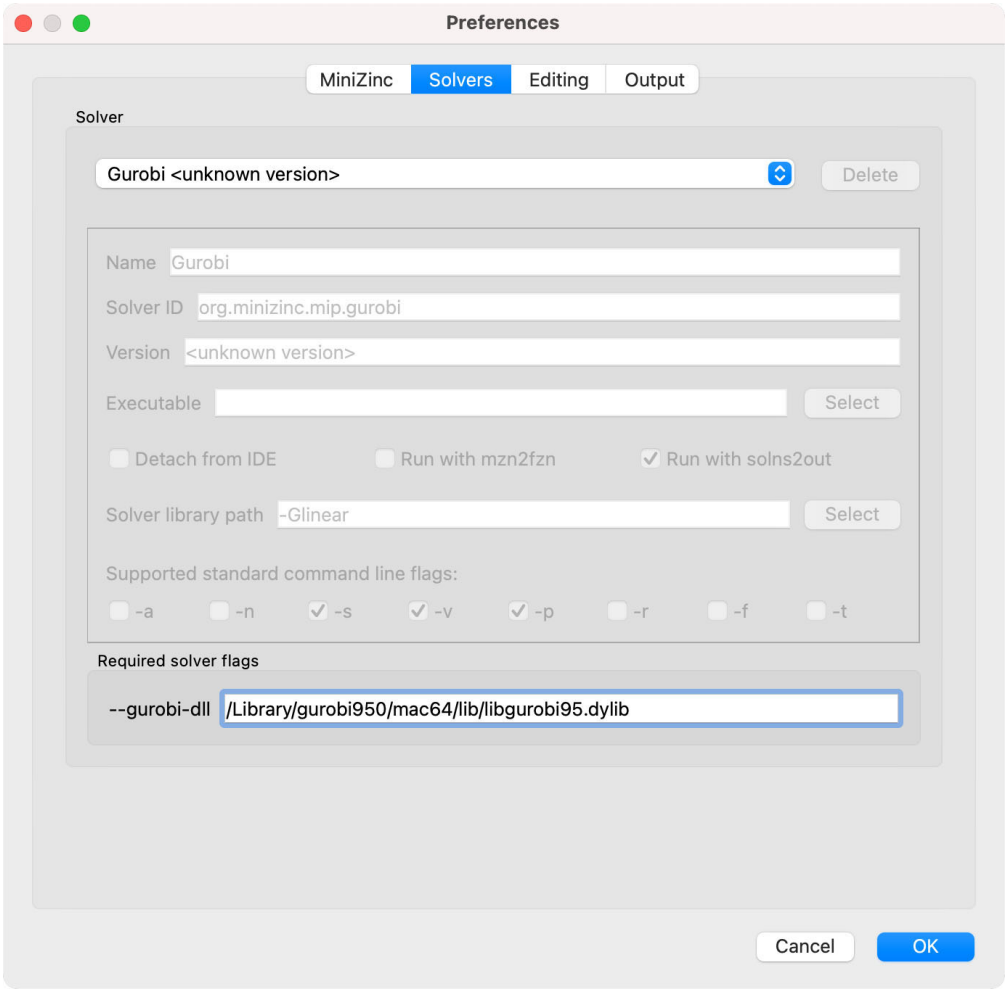
图

，

Configuring existing solvers

图gurobiXX.dllgurobilibgurobiXX.solibXX

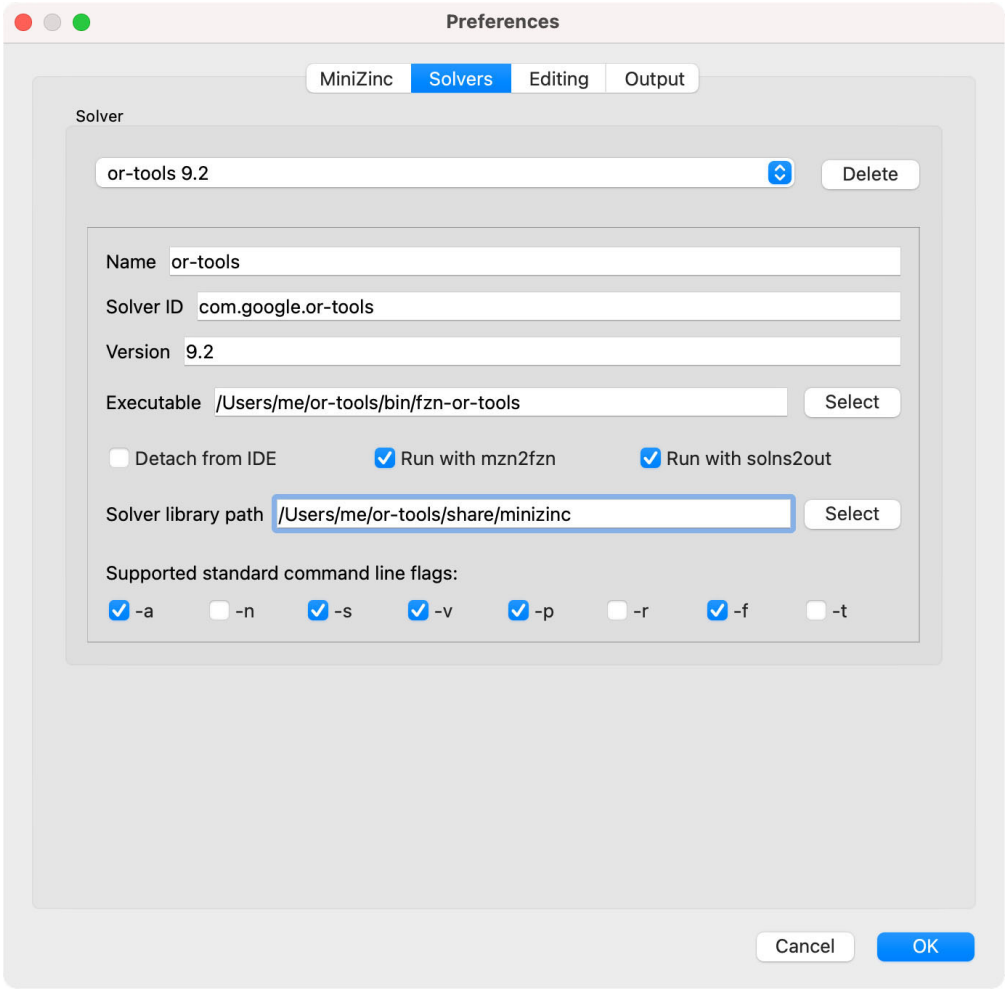
--cplex-dllcplexXXXX.dllcplexlibcplexXXX.solibcplexXXXX.jnilibXXXXXXX


Preferences
MiniZinc
Solvers
Editing
Output
Solver
Gurobi <unknown version>
Delete
Name
Gurobi
Solver ID
org.minizinc.mip.gurobi
Version
<unknown version>
Executable
Select
Detach from IDE
Run with mzn2fzn
Run with solns2out
Solver library path
-Glinear
Select
Supported standard command line flags:
-a
-n
-s
-v
-p
-r
-f
-t
Required solver flags
--gurobi-dll
/Library/gurobi950/mac64/lib/libgurobi95.dylib
Cancel
OK

图

Adding new solvers

图/Users/me/or-tools

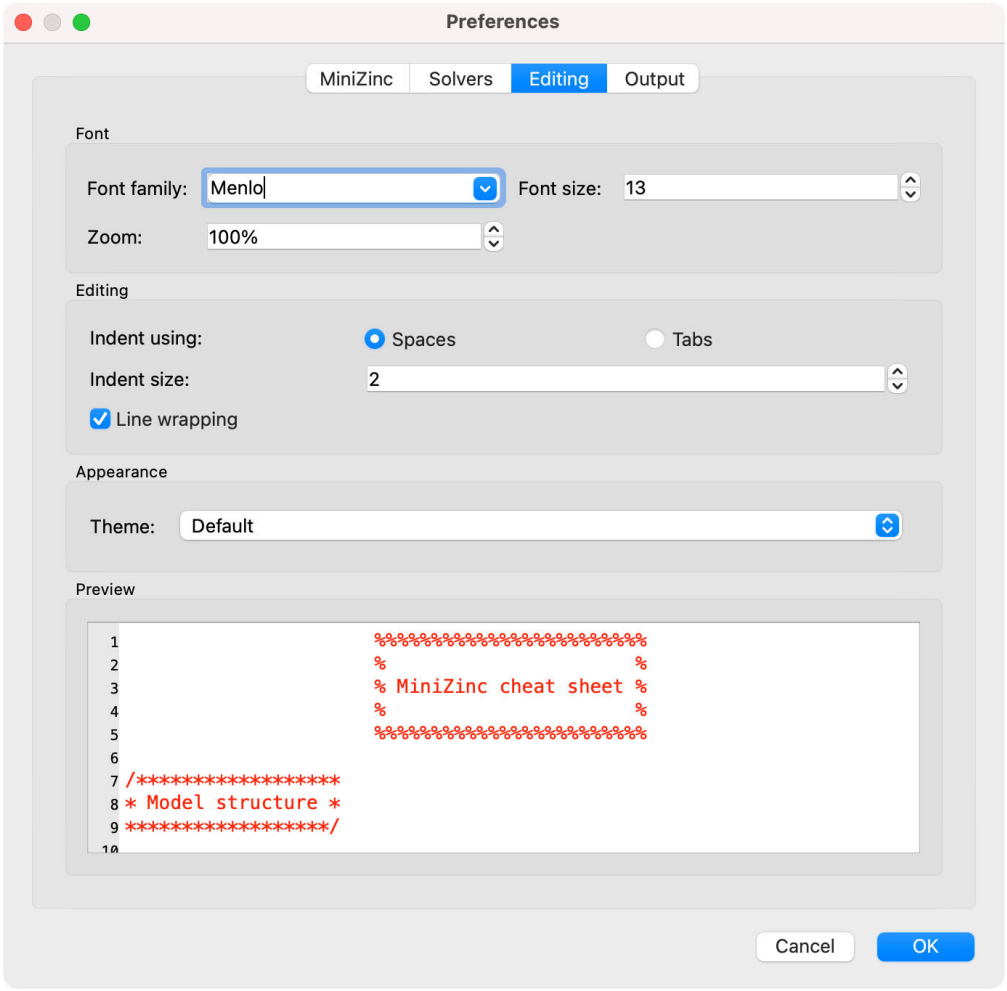


图

节-I

minizinc

3.2.5.3 Editing options

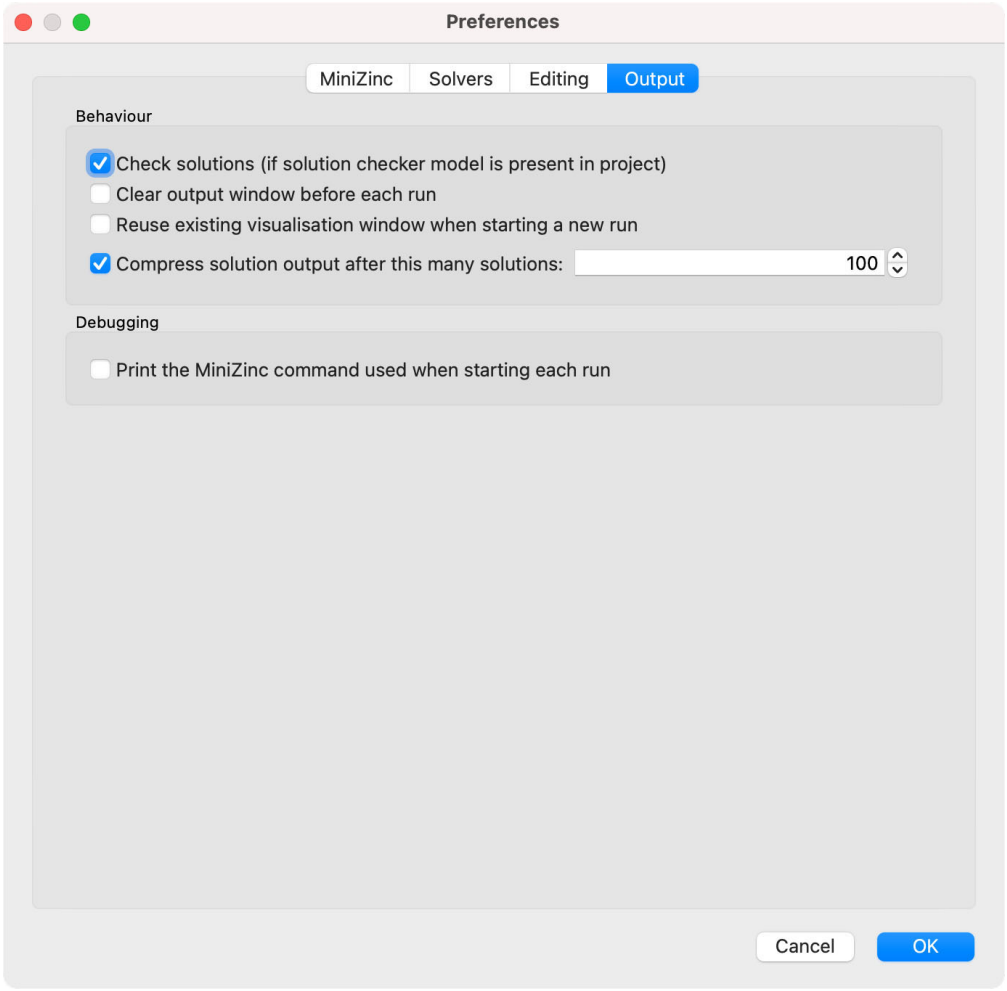


3.2.5.4 Output options

节

```
var 1..1000: x;
solve satisfy;
```

```
  [ 99 more solutions ]
x = 200;
----------
  [ 199 more solutions ]
x = 400;
----------
  [ 399 more solutions ]
x = 800;
----------
  [ 199 more solutions ]
x = 1000;
----------
```


Preferences
MiniZinc
Solvers
Editing
Output
Behaviour
Check solutions (if solution checker model is present in project)
Clear output window before each run
Reuse existing visualisation window when starting a new run
Compress solution output after this many solutions:
100
Debugging
Print the MiniZinc command used when starting each run
Cancel
OK

```
==========
```

```
... more solutions
```

CHAPTER 3.3

Visualising solutions in the MiniZinc IDE

3.3.1 Visualisation library

ide/vis.mzn

```
include "ide/vis.mzn";
```

,

3.3.2 Pre-defined visualisations

output

列表

列表vis_mst.mzn

```
include "globals.mzn";
include "ide/vis.mzn";

% Graph definition
enum NODE = {A, B, C, D, E, F, G, H, I, J};
array [int] of NODE: from = [A, A, A, B, B, B, B, C, C, D, D, E,  E, F, F, F, G,  G,
↪ H, H, I];
array [int] of NODE:   to = [B, C, E, C, D, E, G, D, F, F, G, G,  J, G, H, I, I,  J,
↪ I, J, J];
array [int] of int:     w = [3, 6, 9, 4, 2, 9, 9, 2, 9, 9, 8, 8, 18, 7, 4, 5, 9, 10,
↪ 1, 4, 3];

% Find the minimum spanning tree of the graph
var 0..sum(w): total_weight;
```

```
array [index_set(from)] of var bool: es;
constraint weighted_spanning_tree(
  card(NODE),   % Number of nodes
  length(from), % Number of edges
  from,         % Edge from node
  to,           % Edge to node
  w,            % Weight of edge
  es,           % Whether edge is in spanningtree
  total_weight  % Total weight of spanning tree
);
solve minimize total_weight;

% Graph visualisation
array [int] of string: edge_labels = [i: show(w[i]) | i in index_set(w)];
output vis_graph_highlight(
  from,                  % Edge from node
  to,                    % Edge to node
  edge_labels,           % Edges are labelled with their weights
  [i: true | i in NODE], % All nodes are present in a spanning tree
  es                     % Whether edge is in spanningtree
);

% Objective visualisation
output vis_line(total_weight, "Spanning tree weight");
```

3.3.3 Custom visualisations

outputvis_server

vis_serverinit_datasolution_data

/minizinc-ide.js

3.3.3.1 Example visualisation

基本模型

，

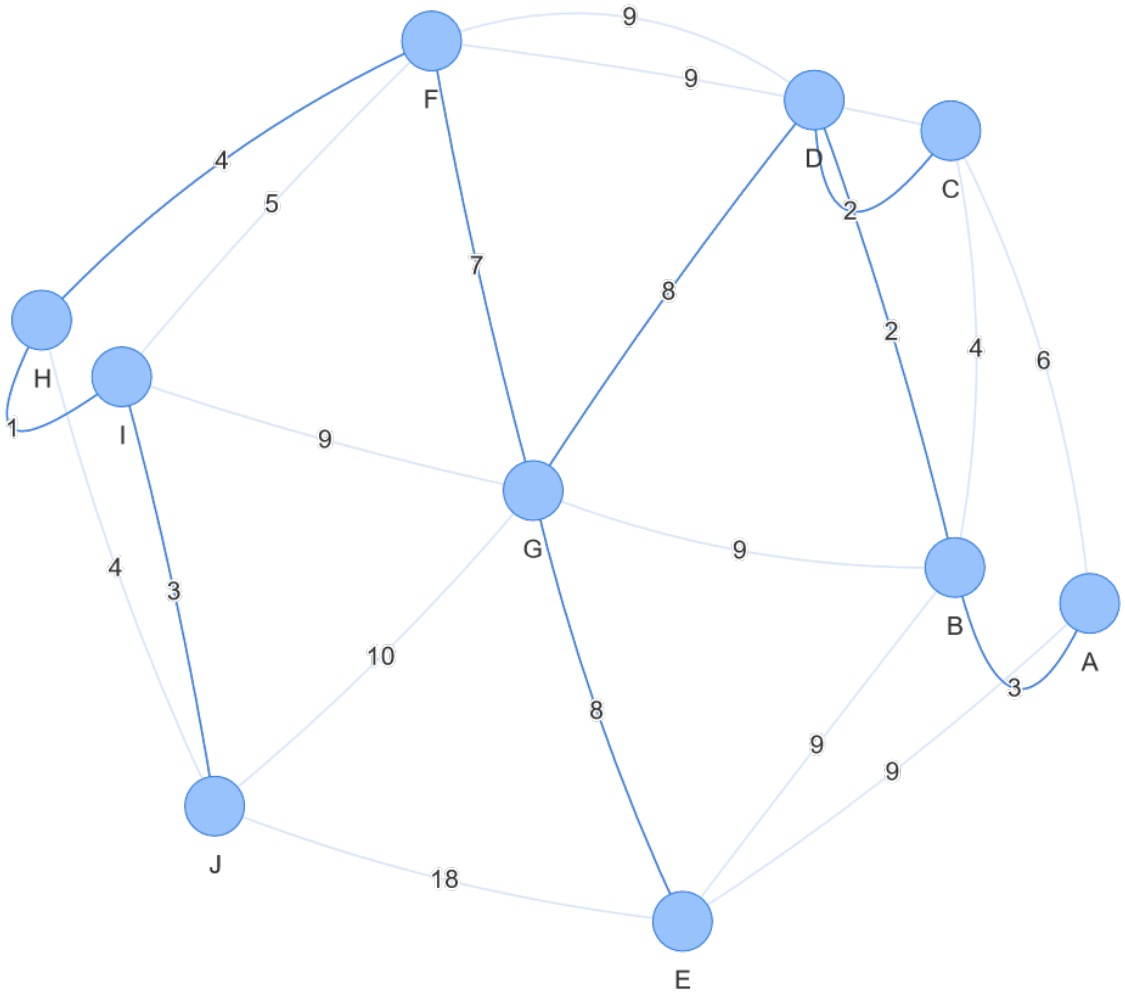
列表vis_aust.mzn

```
include "ide/vis.mzn";

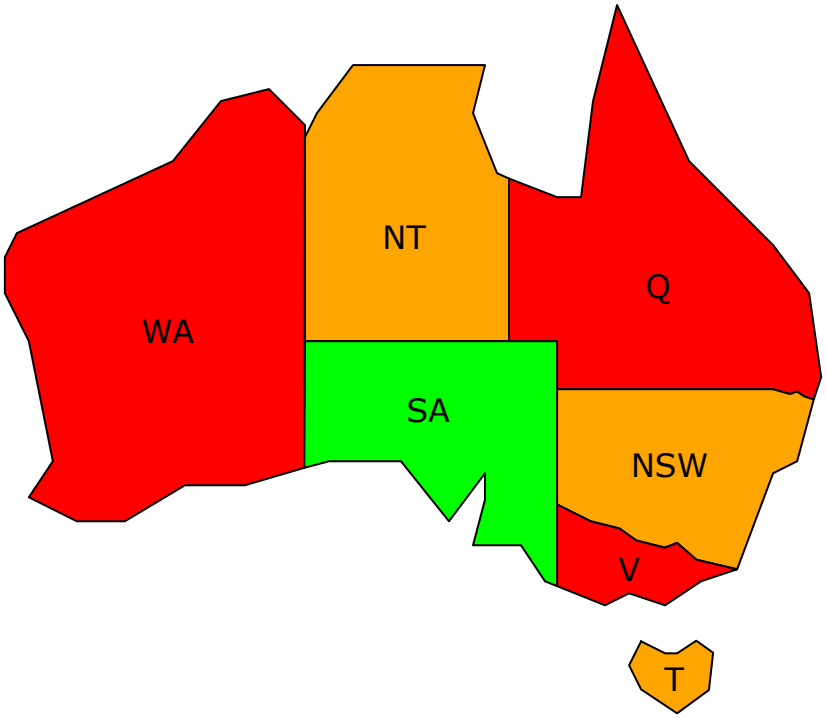
int: n_colors;

enum PALETTE = { red, lime, blue, cyan, magenta, yellow, orange };
set of PALETTE: COLOR = to_enum(PALETTE, 1..n_colors);

% Color for each region
var COLOR: wa;
var COLOR: nt;
```



图



图

```
var COLOR: sa;
var COLOR: q;
var COLOR: nsw;
var COLOR: v;
var COLOR: t;

% Neighboring regions have different colours
constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;

% Data to send during initialisation of the visualisation
any: initial_data = (n: n_colors);

% Data to send on each solution
% Note the use of :: output_only
any: solution_data :: output_only = (
  wa: show(wa), % Get colors as strings
  nt: show(nt),
  sa: show(sa),
  q: show(q),
  nsw: show(nsw),
  v: show(v),
  t: show(t)
);
```

```
% Launch the visualisation
output :: vis_server("vis_aust.html", initial_data) solution_data;
```

,

```
showJSON()

:: output_only
```

列表vis_aust.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Australia Visualisation</title>
    <script src="/minizinc-ide.js"></script>
    <style>
      html, body {
        margin: 0;
        padding: 0;
        overflow: hidden;
        font-family: Helvetica, sans-serif;
      }
      * {
        box-sizing: border-box;
      }
      main {
        width: 100vw;
        height: 100vh;
        padding: 50px;
      }
      svg {
        width: 100%;
        height: 100%;
      }
      .shapes {
        fill: none;
        stroke: #000;
        stroke-linejoin: round;
        stroke-linecap: round;
        stroke-width: 1;
      }
      .labels {
        font-family: sans-serif;
        font-size: 16px;
        text-anchor: middle;
        fill: #000;
      }
    </style>
  </head>
  <body>
    <main>
```

```
      <h3>Map coloring with <span id="count">n</span> colors</h3>
      <svg viewBox="0 0 412 357.33">
        <g class="shapes">
          <path id="wa" d="m152 67.333v-6l-18-18-24 6-24 30-78 36-6 12v18l12 24 12␣
↪60-12 18 24 12h24l30-18h30l29.68-8.72z" />
          <path id="nt" d="m152 67.333 6-12 18-24h66l-6 24 12 30 6 2.72v81.28h-102v-
↪102" />
          <path id="q" d="m254 88.053 24 9.28h12l6-48 12-48 36 78 42 42 18 24 6 42-
↪3.6 10.8h-1.2l-3.6-1.2-3.6-2.4-3.6 1.2-8.4-2.4h-108v-24h-24v-81.28" />
          <path id="nsw" d="m406.4 198.13-8.4 31.2-12 6-18 48-20.4-4.8-9.6-8.4-6 2.
↪4-14.4-3.6-8.4-6-14.4-3.6-16.8-8.4v-57.6h108l8.4 2.4 3.6-1.2 3.6 2.4 3.6 1.2z" />
          <path id="v" d="m368 283.33-20.4-4.8-9.6-8.4-6 2.4-14.4-3.6-8.4-6-14.4-3.
↪6-16.8-8.4v40.8l24 9.6 12-6 18 6 18-12z" />
          <path id="sa" d="m278 169.33h-126l-0.32 63.28 12.32-3.28h36l24 30 18-
↪24v13.2l-6 22.8h24l12 18 6 2.4v-122.4" />
          <path id="t" d="m320 319.33 12 6h6l9.6-6.32 8.4 6-2.08 18.64-15.92 11.68-
↪18-12-6-12 6-12" />
        </g>
        <g class="labels">
          <text x="83.533119" y="170.0192">WA</text>
          <text x="201.66409" y="123.36276">NT</text>
          <text x="328.74969" y="147.87715">Q</text>
          <text x="334.34839" y="237.23608">NSW</text>
          <text x="314.19815" y="289.03265">V</text>
          <text x="336.71103" y="344.3877">T</text>
          <text x="213.65285" y="209.55853">SA</text>
        </g>
      </svg>
    </main>
    <script src="vis_aust.js"></script>
  </body>
</html>
```

/minizinc-ide.jsMiniZincIDEvis_aust.js

列表vis_aust.js

```
const HTML_COLORS = ['red', 'yellow', 'blue', 'lime', 'magenta', 'cyan', 'orange'];

(async function() {
  // getUserData can be used to retrieve the JSON data passed to vis_server()
  const userData = await MiniZincIDE.getUserData();
  document.getElementById('count').textContent = userData.n;

  // Handler to set the colors for the solution
  function setSolution(data) {
    for (const r in data) {
      document.getElementById(r).setAttribute('fill', data[r]);
    }
  }

  // Visualise last solution on startup
  const numSols = await MiniZincIDE.getNumSolutions();
  if (numSols > 0) {
    const solution = await MiniZincIDE.getSolution(numSols - 1);
    setSolution(solution.data);
```

```
  }

  // Show new solutions if we're following the latest solution
  let followLatest = true;
  MiniZincIDE.on('solution', (solution) => {
    if (followLatest) {
      setSolution(solution.data);
    }
  });

  MiniZincIDE.on('goToSolution', async (index) => {
    // Requesting index -1 turns on following latest solution
    // Otherwise, we stop showing the latest solution and show the requested one
    followLatest = index === -1;
    const solution = await MiniZincIDE.getSolution(index);
    setSolution(solution.data);
  })
})();
```

MiniZincIDE.getUserData()vis_serversetSolution(data)MiniZincIDE.getSolution(idx)solution goToSolution

3.3.3.2 MiniZincIDE JavaScript API

```
MiniZincIDE.on(event, callback)
```

eventcallback

solutiontimedatavis_json

```
MiniZincIDE.on('solution', function(solution) {
  console.log(solution.time);
  console.log(solution.data);
});
```

goToSolution-1

```
MiniZincIDE.on('goToSolution', async function(index) {
  const solution = await MiniZincIDE.getSolution(index);
  visualiseSolution(solution);
});
```

status==========statustimestatus

ALL_SOLUTIONS

OPTIMAL_SOLUTION

UNSATISFIABLE

UNSAT_OR_UNBOUNDED

UNBOUNDED

UNKNOWN

ERROR

finish

```
MiniZincIDE.off(event, callback)
```

eventcallback

```
MiniZincIDE.getUserData()
```

Promiseide_launch_server()

```
MiniZincIDE.goToSolution(idx)
```

idxgoToSolution

-1

```
MiniZincIDE.solve(modelFile, dataFiles, options)
```

modelFiledataFilesoptionsmodelFiledataFilesnullmodelFile.mzndataFiles.dzn.jsonoptions

```
MiniZincIDE.getNumSolutions()
```

Promise

```
MiniZincIDE.getSolution(index)
```

Promisetimedata-1

```
MiniZincIDE.getAllSolutions()
```

Promisetimedata

```
MiniZincIDE.getStatus()
```

Promisetimestatusnull

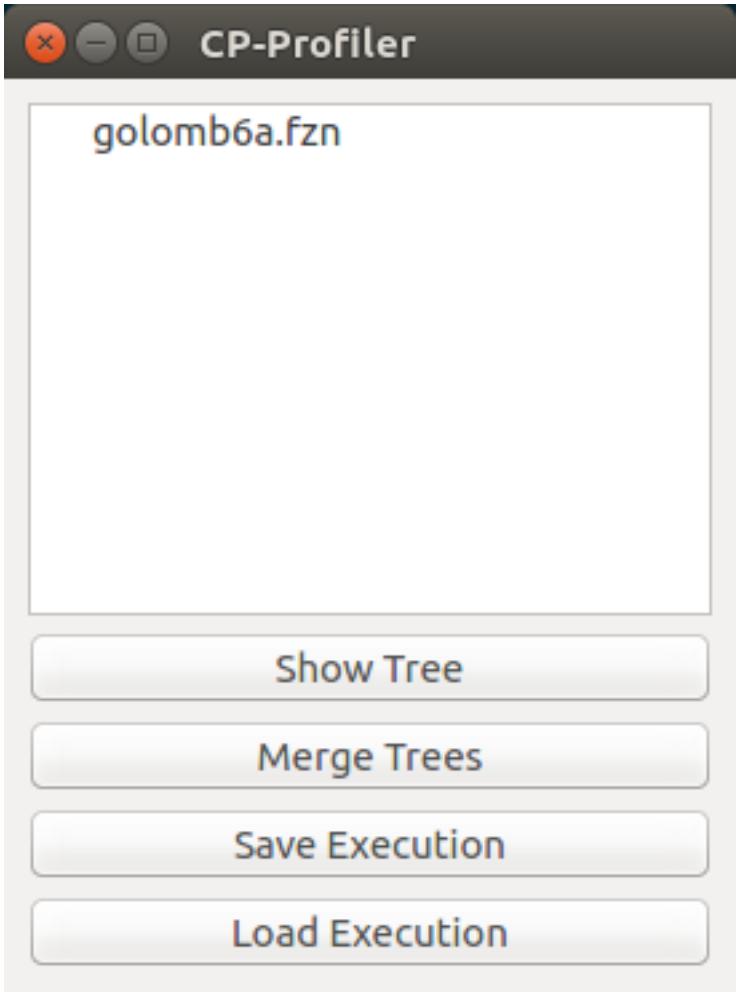
```
MiniZincIDE.getFinishTime()
```

Promisenull

CHAPTER 3.4

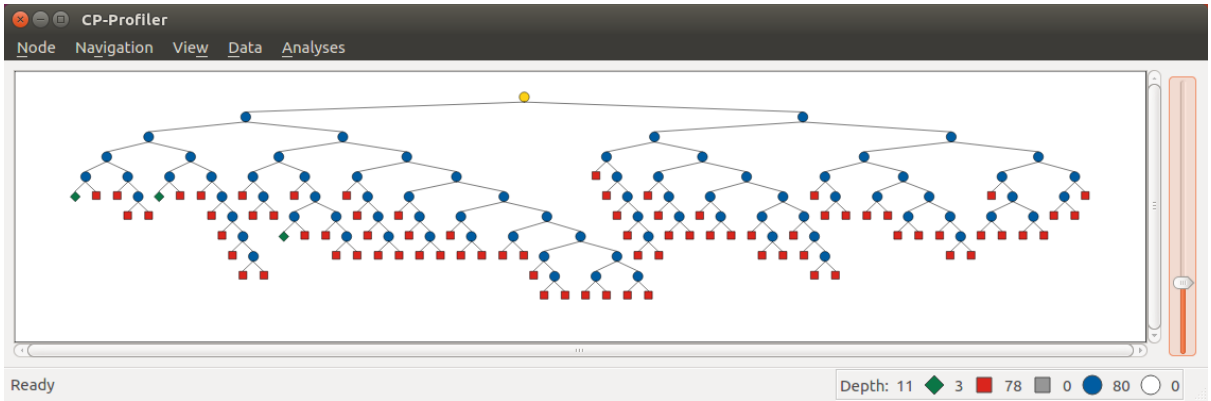
CP-Profiler

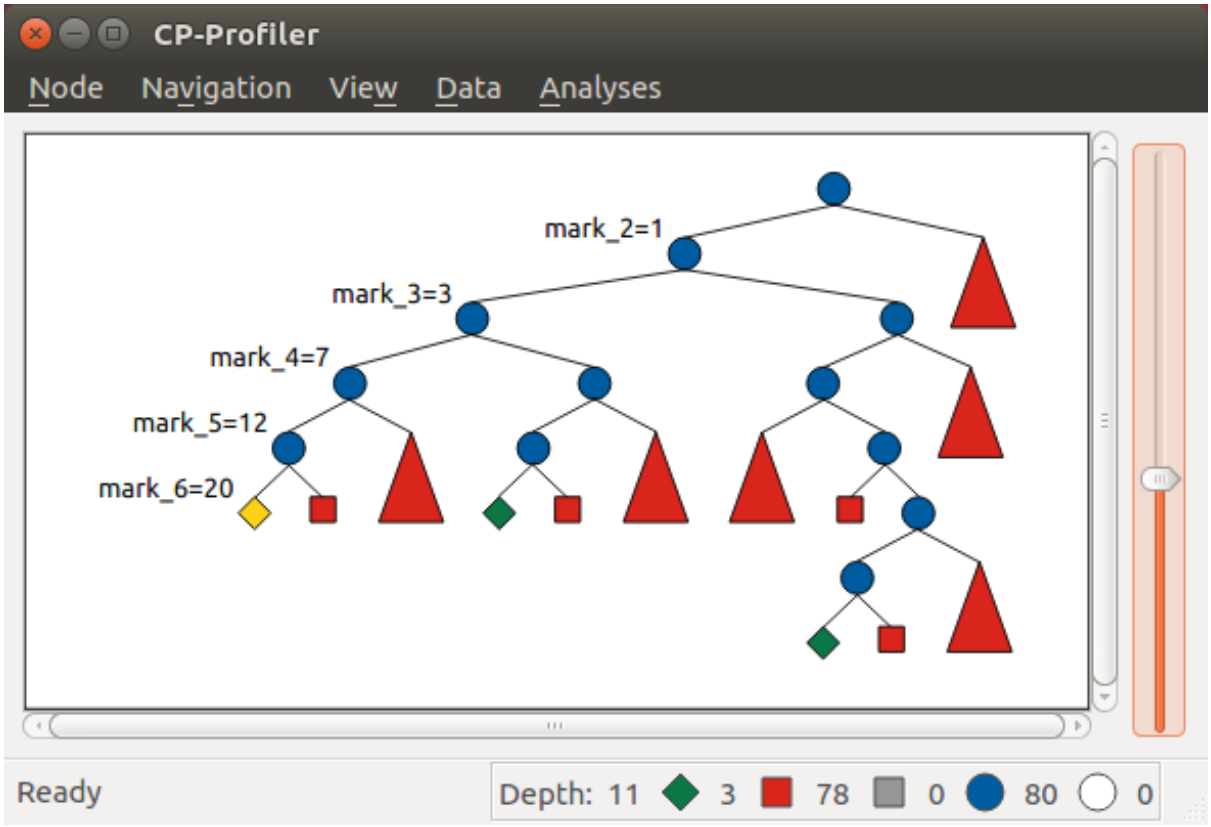
3.4.1 Using the profiler



3.4.2 Traditional Tree Visualisation

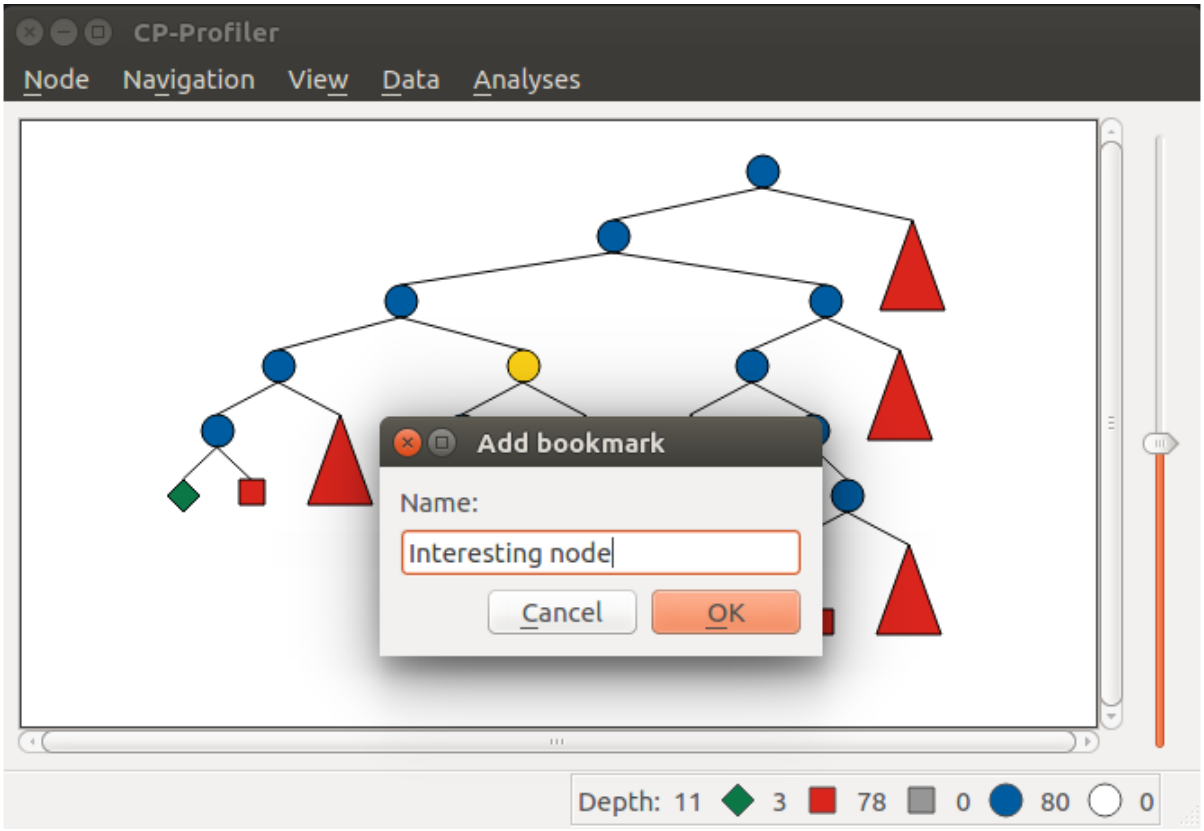
DownShift+Down–Up–Left–Right–R

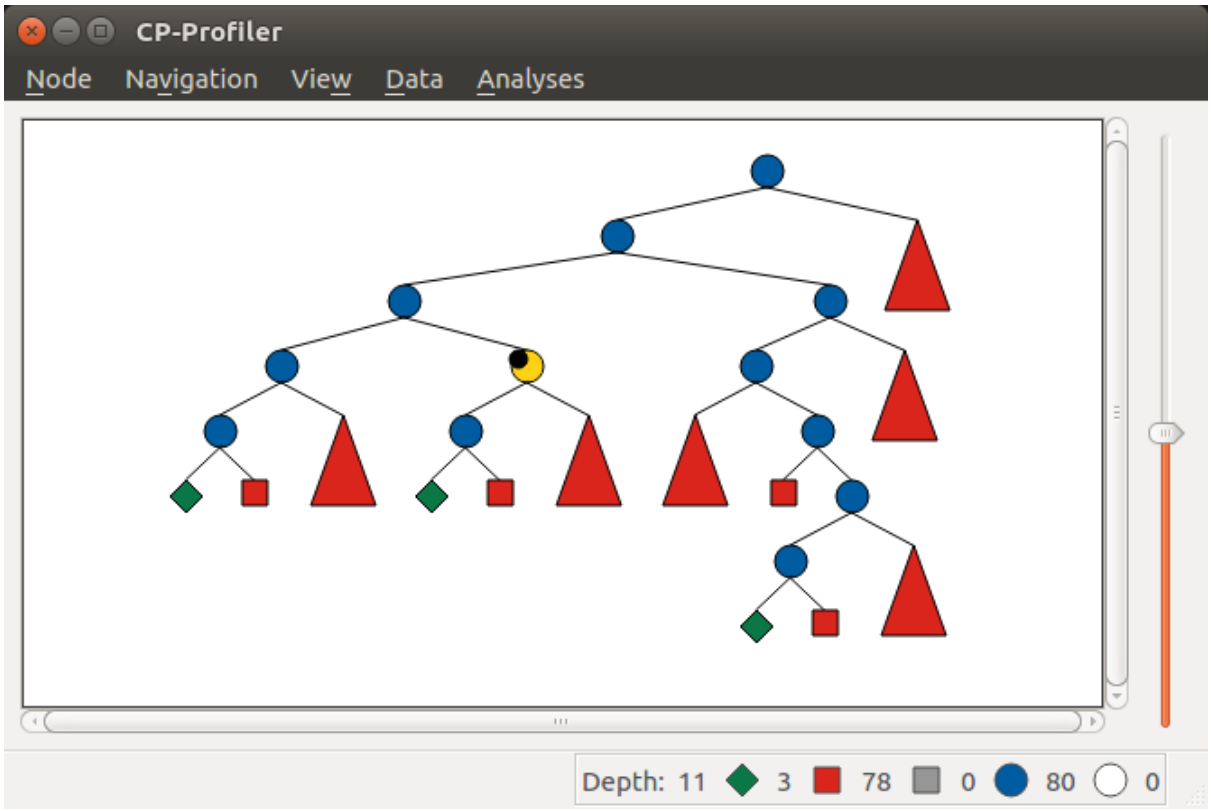


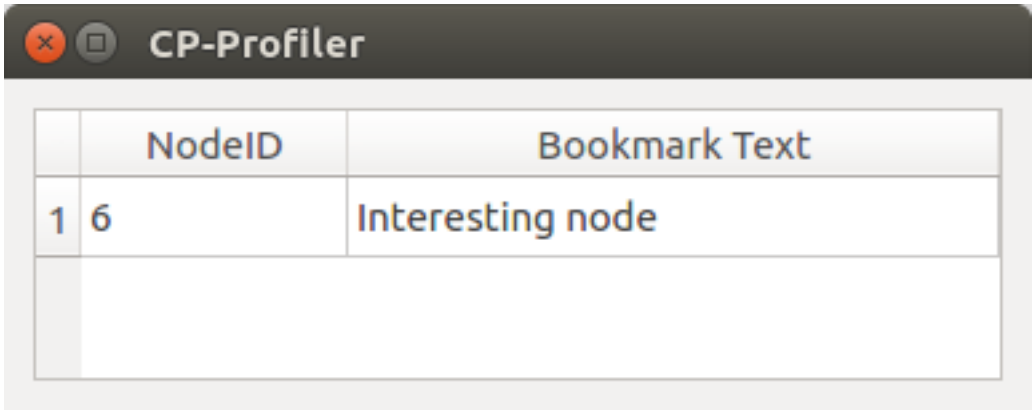


—LShift+L

Shift+B

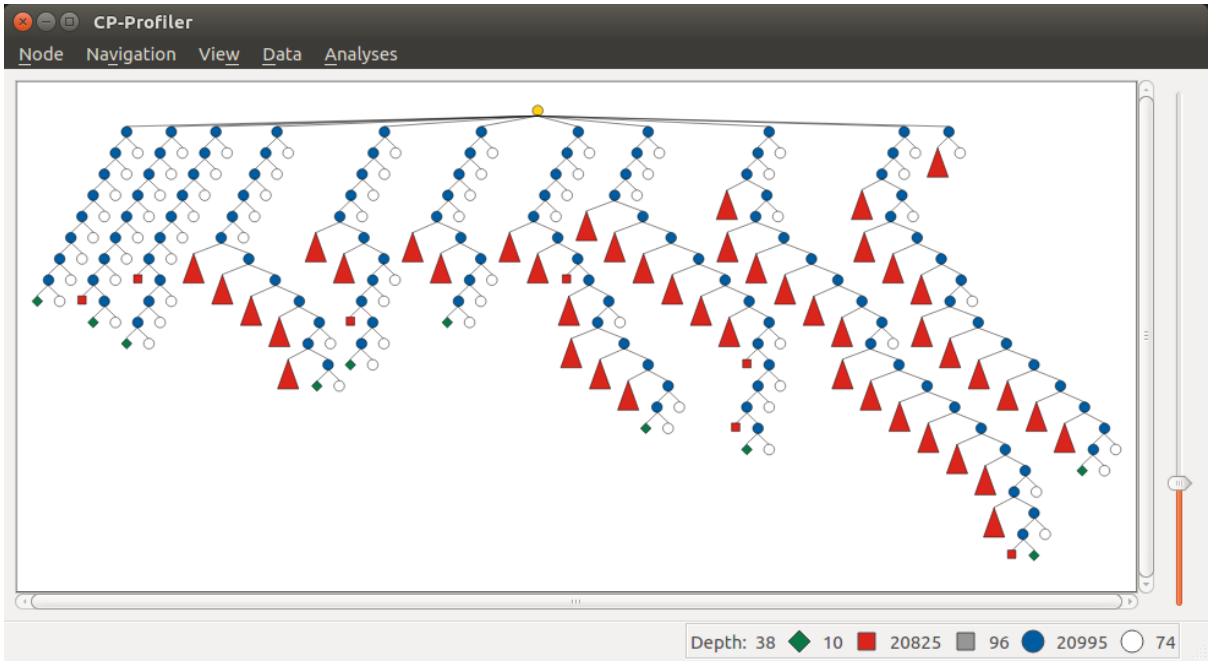

CP-Profiler
Node Navigation View Data Analyses
Add bookmark
Name:
Interesting node
Cancel
OK
Depth: 11 3 78 0 80 0


CP-Profiler
Node Navigation View Data Analyses
Depth: 11 3 78 0 80 0



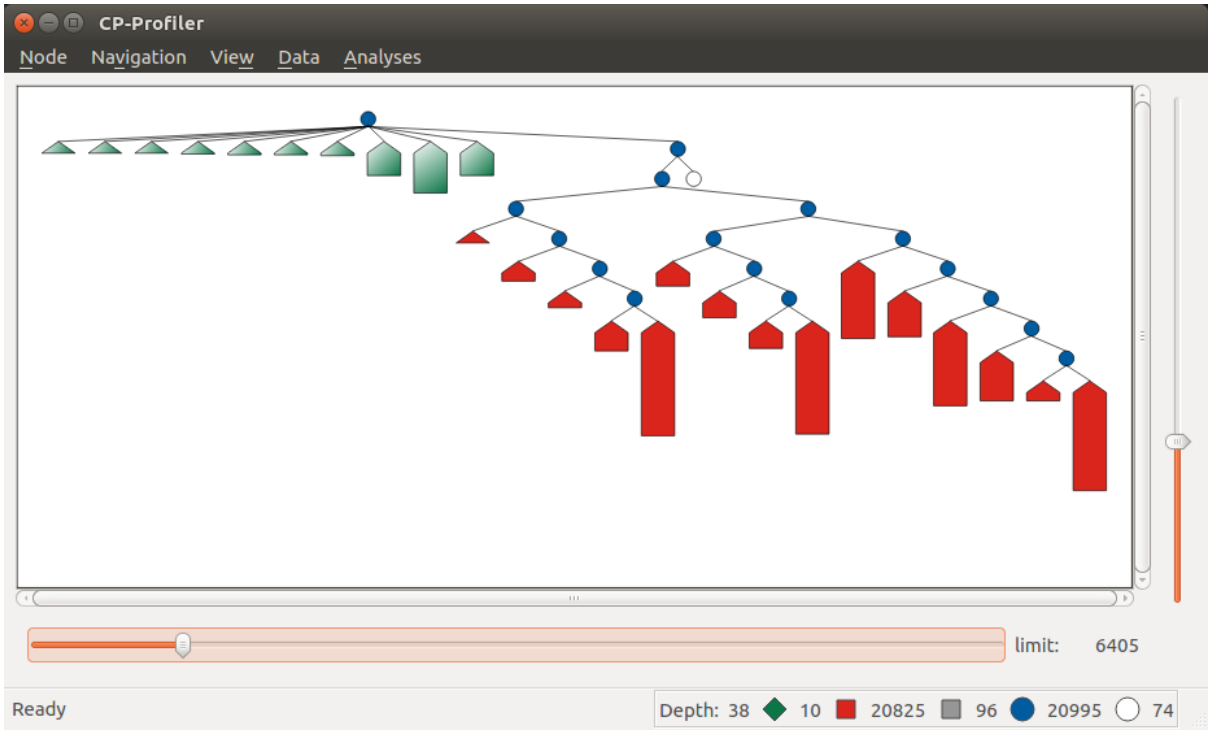
_’

3.4.3 Alternative Search Tree Visualisations



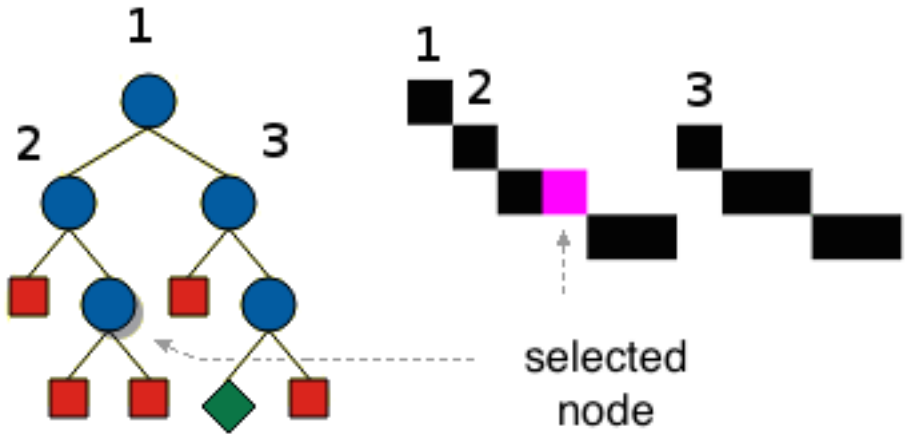
“” “”

“”



“”

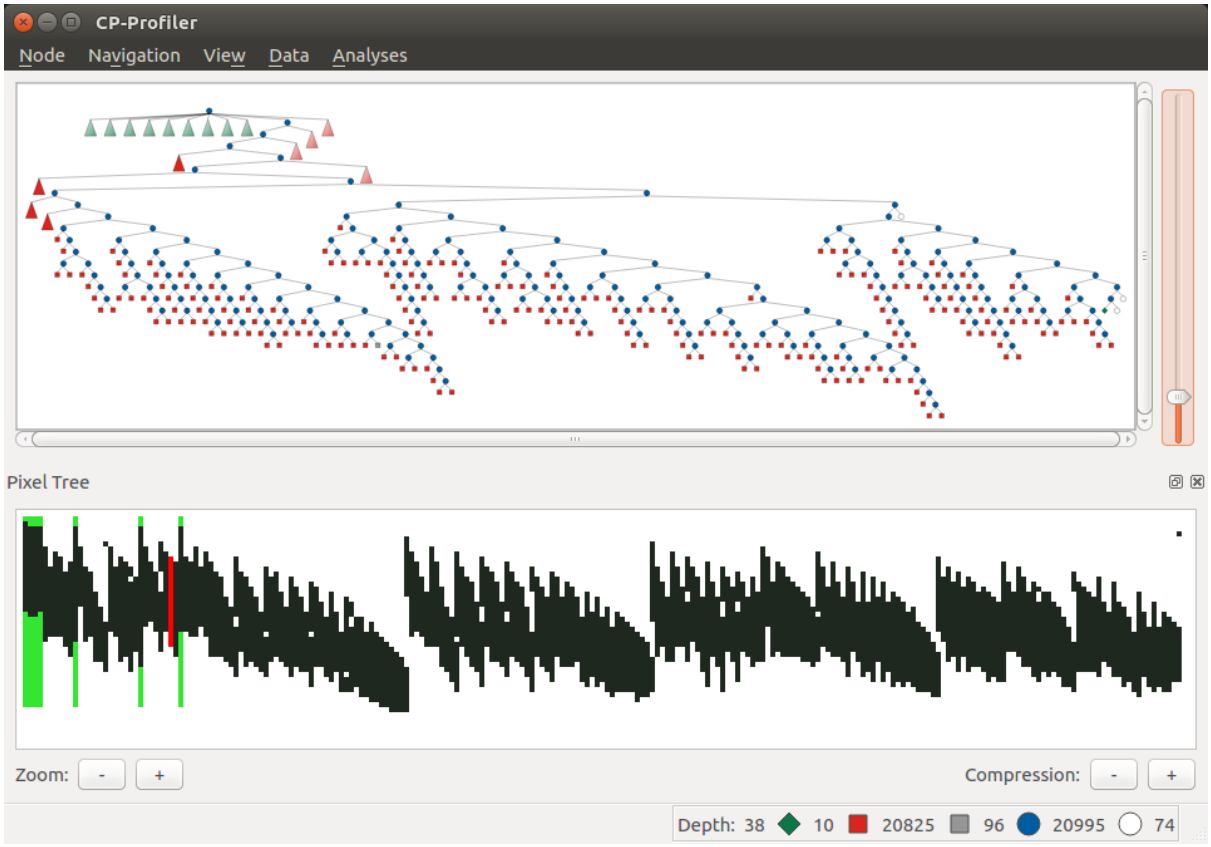
Ctrl+L



Shift+P

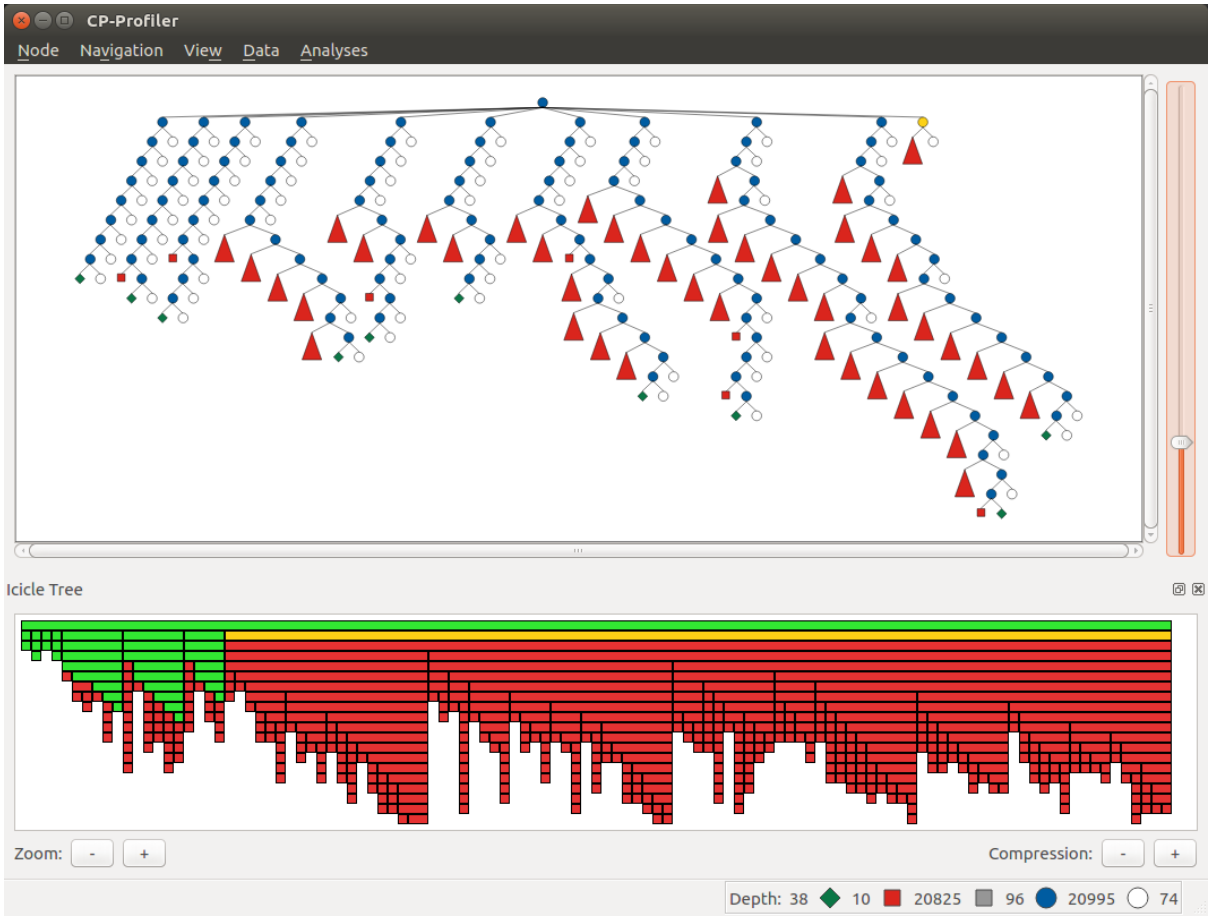
“”

“”


CP-Profiler
Node
Navigation
View
Data
Analyses
Pixel Tree
Zoom:
-
+
Compression:
-
+
Depth: 38
10
20825
96
20995
74

,

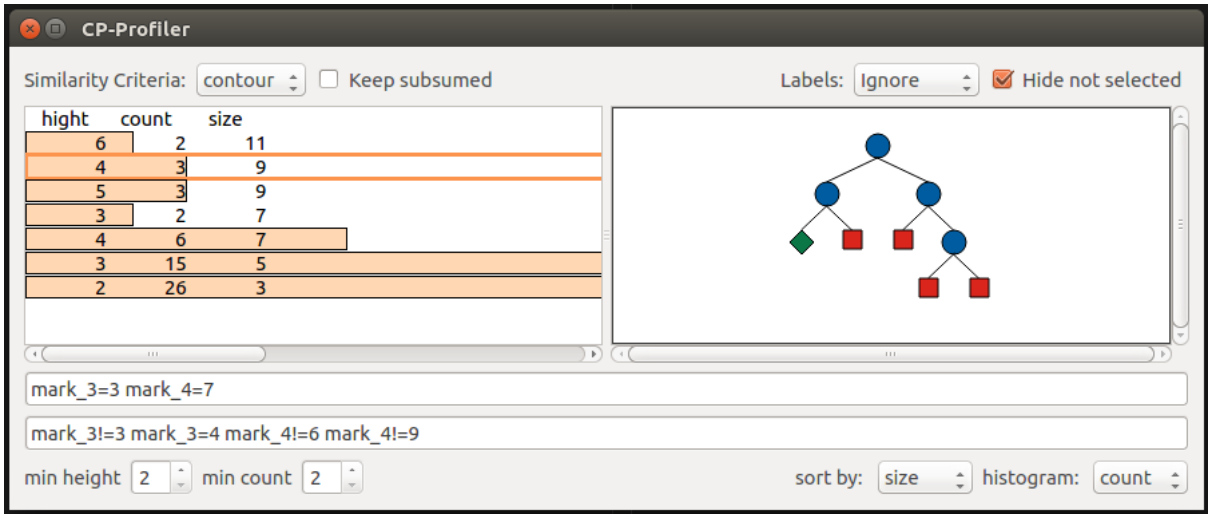
Shift+I

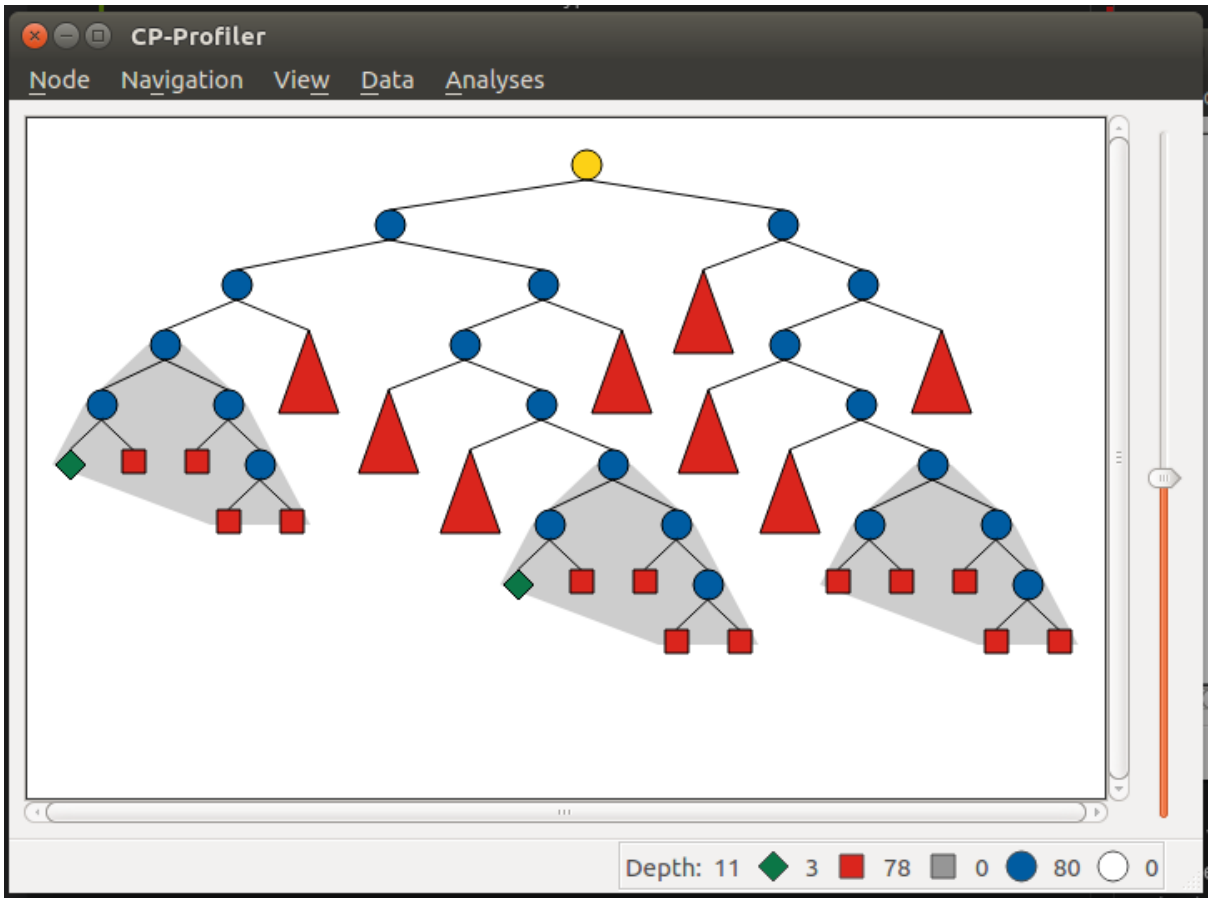


3.4.4 Similar Subtree Analysis

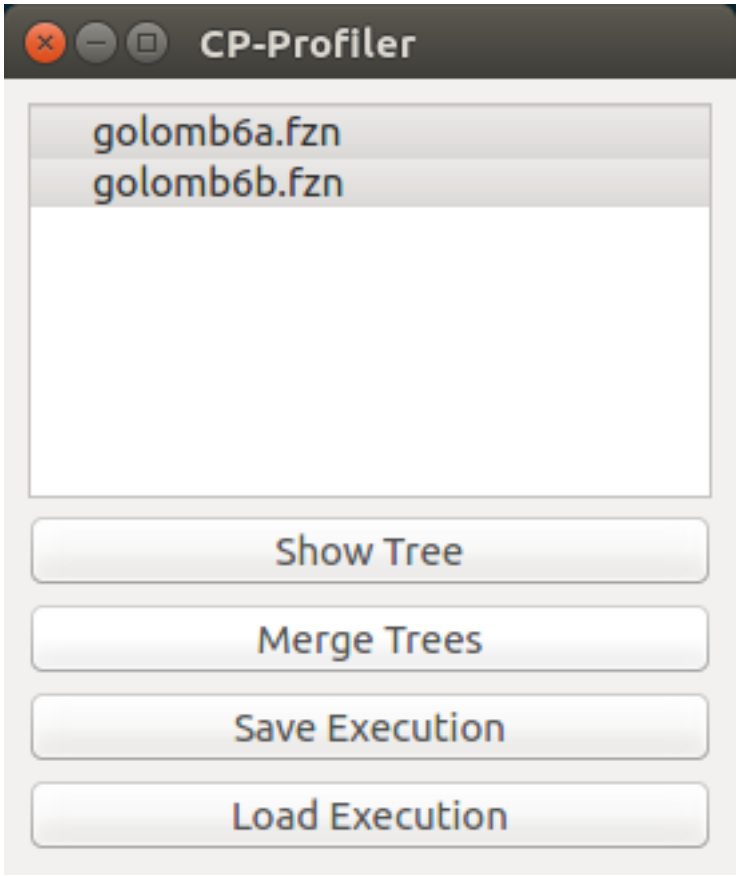
`Shift+S`

“”

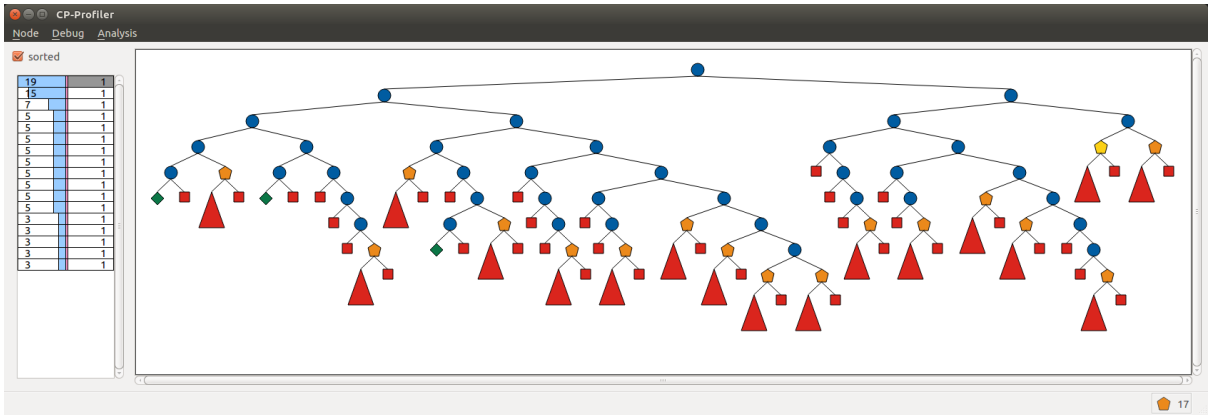




3.4.5 Execution Comparison by Merging

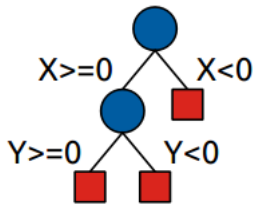


—



3.4.6 Replaying Search

, , ,



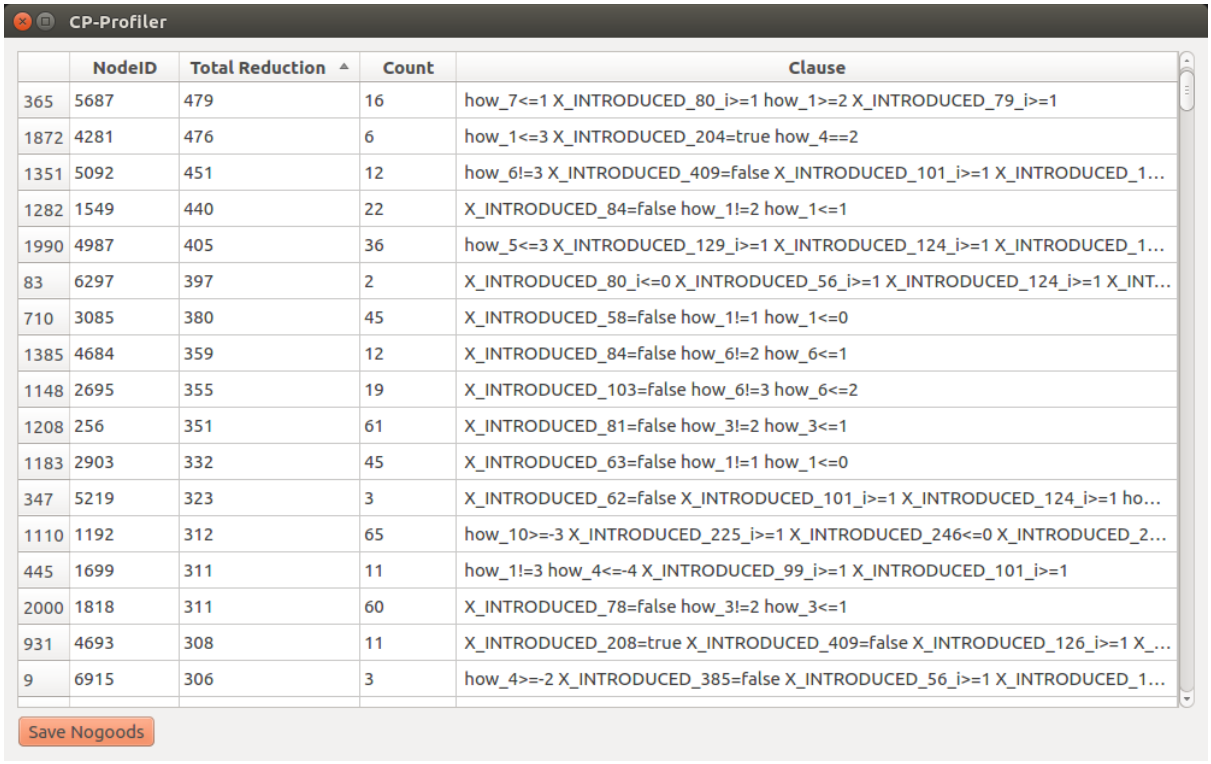

```
0 2 1 X>=0 2 X<0
1 2 3 Y>=0 4 Y<0
3 0
4 0
2 0
```

3.4.7 Nogood Analysis

_

```
chuffed pizza.fzn --cpprofiler
```

```
gecode --cpprofiler_replay pizza.search pizza.fzn
```



\t

3.4.8 The protocol (high level)

Name’ ’

Execution ID

Node ID

Parent IDNode ID

Alternative’ 01

Number of Children0

Status

Label	Node ID	Parent ID	Alternative	Number of Children	Status

3.4.9 The protocol (low level)

Node0Done1Start2Restart3

Nodeidpidaltchildrenstatus

idpid——altchildrenstatus’

label

nogood

info

field name	field id	size (bytes)
id		
pid		
alt		
children		
status		
label		
nogood		
info		

Row	Bytes	Interpretation
	00 00 00 21	
	02	
	02	
	00 00 00 1B	
	7b 22 6e 61 6d 65 22 3a 20 22 6d 69 6e 69 6d 61 6c 20 65 78 61 6c 70 6c 65 22 7d	‘ “” “” ’

Row	Bytes	Interpretation
	00 00 00 2B	
	00	
	00 00 00 00	
	FF FF FF FF	
	FF FF FF FF	
	FF FF FF FF	
	FF FF FF FF	
	FF FF FF FF	
	FF FF FF FF	
	00 00 00 02	
	02	
	00	
	00 00 00 04	
	52 6f 6f 74	‘’

Row	Bytes	Interpretation
	00 00 00 01	
	01	

CHAPTER 3.5

Solving Technologies and Solver Backends

minizincminizincminizinc “” minizinc

minizinc

```
$ minizinc --help <solver-id or tag>
```

3.5.1 Constraint Programming Solvers

‘’

3.5.1.1 Gecode

’

include "gecode.mzn";节

3.5.1.2 Chuffed

’ 搜索-f

节

3.5.1.3 OR-Tools

’ -p搜索-f

3.5.2 Mixed-Integer Programming Solvers

```
minizinc --solver mip -v -s -a model.mzn data.dzn
```

```
minizinc --solver gurobi -v -s -a model.mzn data.dzn
```

3.5.2.1 MIP-Aware Modeling (But Mostly Useful for All Backends)

‘’

```
constraint forall(s in TASKS)(exists([whentask[s]=0] ++
  [whentask[s]>= start[s]+(t*numslots) /\ whentask[s]<=stop[s]+(t*numslots) | t in␣
↪0..nummachines-1]));
```

```
constraint forall(s in TASKS)(whentask[s] in
  {0} union array_union([ start[s]+(t*numslots) .. stop[s]+(t*numslots) | t in 0..
↪nummachines-1]));
```

b=1 -> x<=0b=1 -> 0.001*x<=0.0—

<expr> <= 1000000*yvar 0..1: y ‘’ ’ <expr>y=0 -> <expr> <= 0y=0 -> 0.0001*<expr> <= 0.0

```
var 0..1: c;                                              %% Whether we construct␣
↪the road
var int: cost_road = 286*c + 1000000*(1-c);
var int: cost_final = min( [ cost_road, cost1, cost2 ] );
```

cost_road’ c=0.999999c=1cost_road=287

cost1cost2

```
int: cost_others_ub = 1+2*ub_array( [cost1, cost2] );    %% Multiply by 2 for a␣
↪stronger LP relaxation
var int: cost_road = 286*c + cost_others_ub*(1-c);
```

3.5.2.2 Useful Flattening Parameters

share/minizinc/linear/options.mzn

```
-D nSECcuts=0/1/2                           %% Subtour Elimination Constraints,␣
↪see below
-D fMIPdomains=true/false                   %% The unified domains feature, see␣
↪below
-D float_EPS=1e-6                           %% Epsilon for floats' strict␣
↪comparison,
                                            %% used e.g. for the following cases:
                                            %% x!=y, x<y, b -> x<y, b <-> x<=y
-DfIndConstr=true -DfMIPdomains=false       %% Use solver's indicator constraints,␣
↪see below
-DMinMaxGeneral=true                        %% Send min/max constraints to the␣
↪solver (Gurobi only)
-DQuadrFloat=false -DQuadrInt=false         %% Not forward float/integer␣
↪multiplications for MIQCP backends, see below
-DUseCumulative=false                       %% Not forward cumulative with fixed␣
↪durations/resources (SCIP only)
-DUseOrbisack=false                         %% Not forward lex_lesseq for binary/
↪bool vectors (SCIP only)
-DOrbisackAlwaysModelConstraint=true        %% lex_lesseq ignores being in␣
↪symmetry_breaking_constraint() (SCIP only)
                                            %% Required for SCIP 7.0.2, or use␣
↪patch: http://listserv.zib.de/pipermail/scip/2021-February/004213.html
-DUseOrbitope=false                         %% Not forward lex_chain_lesseq for␣
↪binary/bool matrices (SCIP only)
--no-half-reifications                      %% Turn off halfreification (full␣
↪reification was until v2.2.3)
```

3.5.2.3 Some Solver Options and Changed Default Values

, ,

```
-h <solver-tag>    full description of the backend options
--relGap <n>       relative gap |primal-dual|/<solver-dep> to stop. Default 1e-8,␣
↪set <0 to use backend's default
--feasTol <n>      primal feasibility tolerance (Gurobi). Default 1e-8
--intTol <n>       integrality tolerance for a variable. Default 1e-8
--solver-time-limit-feas <n>, --solver-tlf <n>
                   stop after <n> milliseconds after the first feasible solution␣
↪(some backends)
--writeModel <file>
                   write model to <file> (.lp, .mps, .sav, ...). All solvers␣
↪support the MPS format
                   which is industry standard. Most support the LP format. Some␣
↪solvers have own formats,
                   for example, the CIP format of SCIP ("constraint integer␣
↪programming").
--readParam <file>
                   read backend-specific parameters from file (some backends)
--writeParam <file>
                   write backend-specific parameters to file (some backends)
--readConcurrentParam <file>
                   each of these commands specifies a parameter file of one␣
↪concurrent solve (Gurobi only)
```

```
  --keep-paths       this standard flattening option annotates every item in FlatZinc␣
  ↪by its "flattening history".
                     For MIP solvers, it additionally assigns each constraint's name␣
  ↪as the first 255 symbols of that.
  --cbcArgs '-guess -cuts off -preprocess off -passc 1'
                     parameters for the COIN-OR CBC backend
```

-p--enable-cbc-parallel

3.5.2.4 Subtour Elimination Constraints

-DCOMPILE_BOOST_MINCUT=ON#definelib/algorithms/min_cut.cpp-DnSECcuts=<n><n>

3.5.2.5 Unified Domains (MIPdomains)

‘’

-D fMIPdomains=false--help

3.5.2.6 Indicator Constraints

’ -D fIndConstr=true -D fMIPdomains=false

3.5.2.7 Quadratic Constraints and Objectives (MIQCP)

-DQuadrFloat=false -DQuadrInt=false

3.5.2.8 Pools of User Cuts and Lazy Constraints

::MIP_cut::MIP_lazyshare/minizinc/linear/options.mzn

3.5.2.9 Warm Starts

3.5.3 Non-Linear Solvers via NL File Format

couenne.mscshare/minizinc/solvers

```
{
  "id" : "org.coin-or.couenne",
  "name" : "Couenne",
  "executable" : "/Users/tack/Downloads/couenne-osx/couenne",
  "version": "0.5.6",
  "supportsFzn":false,
  "supportsNL":true
}
```

version’

minizinc --solvers

minizinc --solver couenne model.mzn

-Glinear [-DQuadrFloat=true -DQuadrInt=true]

CHAPTER 3.6

Automatic Solution Checking, Model Validation, and Benchmarking

3.6.1 Autonomous Automatic Solution Checking and Benchmarking

--output-mode dzn_objective--output-objective.dzn--allow-multiple-assignments--output-objective

```
var int: a;
var float: b = a;
solve
  maximize a-b;
output
  [ "a+b is \(a+b)\n" ];
```

minizinc --solver gecode test_output_mode.mzn --output-mode dzn --output-objective

```
a = -2147483646;
_objective = 0.0;
----------
==========
```

.dzntests/benchmarking/mzn-test.py

```
mzn-test.py --solver gecode model.mzn data.dzn
```

mzn-test.py

3.6.2 Model Validation: Automatic Solution Checking with a Checker Model

```
minizinc model.mzn model.mzc.mzn data.dzn
```

```
minizinc --compile-solution-checker model.mzc.mzn
```

model.mzc

```
minizinc model.mzn model.mzc data.dzn
```

.mzc.mzc.mzn “” “”

3.6.2.1 Basic checker models

，

列表aust.mzn

```
% Colouring Australia using nc colours
int: nc = 3;

var 1..nc: wa;   var 1..nc: nt;  var 1..nc: sa;   var 1..nc: q;
var 1..nc: nsw;  var 1..nc: v;   var 1..nc: t;

constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;
solve satisfy;

output ["wa=\(wa)\t nt=\(nt)\t sa=\(sa)\n",
        "q=\(q)\t nsw=\(nsw)\t v=\(v)\n",
         "t=", show(t),  "\n"];
```

wantCORRECTINCORRECT’

列表aust.mzc.mzn

```
int: wa;
int: nt;
int: sa;
int: q;
int: nsw;
int: v;
int: t;

output [
  if wa!=nt /\ wa!=sa /\ nt!=sa /\ nt!=q /\ sa!=q /\
     sa!=nsw /\ sa!=v /\ q!=nsw /\ nsw!=v
  then "CORRECT\n"
  else "INCORRECT\n"
  endif
];
```

```
wa=3   nt=2    sa=1
q=3    nsw=2   v=3
t=3
% Solution checker report:
% CORRECT
----------
```

3.6.2.2 Detailed feedback

，

check’ INCORRECTCORRECT

列表aust-2.mzc.mzn

```
int: wa;
int: nt;
int: sa;
int: q;
int: nsw;
int: v;
int: t;

test check(bool: b,string: s) =
  if b then true else trace_stdout("ERROR: "++s++"\n",false) endif;

output [
  if check(wa!=nt, "wa and nt have the same colour")
  /\ check(wa!=sa, "wa and sa have the same colour")
  /\ check(nt!=sa, "nt and sa have the same colour")
  /\ check(nt!=q, "nt and q have the same colour")
  /\ check(sa!=q, "sa and q have the same colour")
  /\ check(sa!=nsw, "sa and nsw have the same colour")
```

```
  /\ check(sa!=v, "sa and v have the same colour")
  /\ check(q!=nsw, "q and nsw have the same colour")
  /\ check(nsw!=v, "nsw and v have the same colour")
  then "CORRECT: all constraints hold.\n"
  else "INCORRECT\n"
  endif
];
```

```
wa=3   nt=3    sa=3
q=3    nsw=3   v=3
t=3
% Solution checker report:
% ERROR: wa and nt have the same colour
% INCORRECT
----------
```

列表aust-3.mzc.mzn

```
int: wa;
int: nt;
int: sa;
int: q;
int: nsw;
int: v;
int: t;

test check(bool: b,string: s) =
  if b then true else trace_stdout("ERROR: "++s++"\n",false) endif;

array[int] of bool: checks = [
  check(wa!=nt, "wa and nt have the same colour"),
  check(wa!=sa, "wa and sa have the same colour"),
  check(nt!=sa, "nt and sa have the same colour"),
  check(nt!=q, "nt and q have the same colour"),
  check(sa!=q, "sa and q have the same colour"),
  check(sa!=nsw, "sa and nsw have the same colour"),
  check(sa!=v, "sa and v have the same colour"),
  check(q!=nsw, "q and nsw have the same colour"),
  check(nsw!=v, "nsw and v have the same colour")
];

output [
  if forall(checks)
  then "CORRECT: all constraints hold.\n"
  else "INCORRECT\n"
  endif
];
```

```
wa=3   nt=3    sa=3
q=3    nsw=3   v=3
t=3
% Solution checker report:
% ERROR: nsw and v have the same colour
% ERROR: q and nsw have the same colour
% ERROR: sa and v have the same colour
% ERROR: sa and nsw have the same colour
% ERROR: sa and q have the same colour
% ERROR: nt and q have the same colour
% ERROR: nt and sa have the same colour
% ERROR: wa and sa have the same colour
% ERROR: wa and nt have the same colour
% INCORRECT
----------
```

3.6.2.3 Instance data in checker models

列表nqueens.mzc.mzn

```
int: n; % number of queens
set of int: ROW = 1..n;
set of int: COL = 1..n;
array[int] of int: q; % col of queen in each row

test check(bool: b,string: s) =
  if b then true else trace_stdout("ERROR: "++s++"\n",false) endif;

output [
  if check(index_set(q)=1..n, "ERROR: array q should have index set 1..\(n)")
  /\ forall(i in 1..n)(check(q[i] in 1..n, "ERROR: q[\(i)] should have a value in 1.
↪.\(n)"))
  /\ forall(r1, r2 in 1..n where r1 < r2)
           (check(q[r1] != q[r2],
            "queens in rows \(r1) and \(r2) are on same column\n")
            /\
            check(q[r1]+r1 != q[r2]+r2,
            "queens in rows \(r1) and \(r2) are on same up diagonal\n")
            /\
            check(q[r1]-r1 != q[r2]-r2,
            "queens in rows \(r1) and \(r2) are on same down diagonal\n")
           )
  then "CORRECT: All constraints hold"
  else "INCORRECT" endif ];
```

3.6.2.4 Checking optimisation problems

_objectiveintfloat

3.6.2.5 Hidden variables

pos列表

列表photo.mzn

```
int: n;                           % number of people
set of int: PERSON = 1..n;        % set of people
enum GENDER = { M, F, O };        % set of genders
array[PERSON] of GENDER: g;       % the gender of each person
set of int: POSN = 1..n;          % set of positions

array[PERSON] of var POSN: pos;  % decisions: a position for each person

array[POSN] of var PERSON: who;  % view: a person for each position
include "inverse.mzn";
constraint inverse(pos,who);     % channel from decisions to view
constraint forall(i in 1..n-2)
                 (g[who[i]] != g[who[i+1]] \/ g[who[i+1]] != g[who[i+2]]);

solve minimize sum(i in 1..n-1)(abs(pos[i] - pos[i+1]));
```

poswhowhowho

who

列表photo.mznphoto.dzn

```
n = 9;
g = [M,M,M,M,F,F,F,M,M];
```

```
n = 9;
g = [M,M,M,M,F,F,F,M,M];
```

pos = [1,2,3,4,5,6,7,2,9]; _objective=25;inverse(pos,who)who

列表photo.mzc.mzn

```
int: n;
set of int: PERSON = 1..n;
enum GENDER = { M, F, O };
array[PERSON] of GENDER: g;
set of int: POSN = 1..n;
array[int] of int: pos;

array[POSN] of var PERSON: who;
int: _objective;
```

```
test alldifferent(array[int] of int: x) =
                forall(i,j in index_set(x) where i < j)(x[i] != x[j]);

include "inverse.mzn";
constraint if alldifferent(pos) then inverse(pos,who)
           else forall(i in 1..n)(who[i] = 1) endif;

output [if
  check_array(pos, n, POSN, "pos") /\
  check_alldifferent(pos,"pos") /\
  check_array(fix(pos), n, PERSON, "who") /\
  forall(i in 1..n-2)
        (check(g[fix(who[i])] != g[fix(who[i+1])] \/
               g[fix(who[i+1])] != g[fix(who[i+2])],
               "three people of the same gender \(g[fix(who[i])]) in positions \(i).
↪.\(i+2)\n")) /\
  let { int: obj = sum(i in 1..n-1)(abs(pos[i] - pos[i+1])); } in
    check(obj = _objective, "calculated objective \(obj) does not agree with␣
↪objective from the model (\(_objective))\n")
  then
    "CORRECT: All constraints hold"
  else
    "INCORRECT"
  endif];

test check(bool: b,string: s) =
  if b then true else trace_stdout("ERROR: "++s++"\n",false) endif;

test check_alldifferent(array[int] of int: x, string: name) =
  forall(i, j in index_set(x) where i < j)
        (check(x[i] != x[j], name ++ "[\(i)] = \(x[i]) = " ++
         name ++ "[\(j)] " ++
         "when they should be different\n"));

test check_int(int: x, set of int: vals, string: name) =
  check(x in vals, "integer \(name) is not in values \(vals)\n");

function bool: check_array(array[int] of int: x, int: length, set of int: vals,␣
↪string: name) =
  check(length(x) = length, "array \(name) is not of length \(length)\n") /\
  forall(i in index_set(x))
        (check(x[i] in vals, "element \(i) of array \(name), \(x[i]) is not in␣
↪values \(vals)\n"));
```

posalldifferenttestalldifferentinversewho

alldifferent

CHAPTER 3.7

Globalizer

,

3.7.1 Basic Usage

```
minizinc --solver globalizer model.mzn data-1.dzn data-2.dzn
```

cars.mzn

列表cars.mzn

```
% cars.mzn
include "globals.mzn";

int: n_cars; int: n_options; int: n_classes;

set of int: steps = 1..n_cars;
set of int: options = 1..n_options;
set of int: classes = 1..n_classes;

array [options] of int: max_per_block;
array [options] of int: block_size;
array [classes] of int: cars_in_class;
array [classes, options] of 0..1: need;

% The class of car being started at each step.
array [steps] of var classes: class;

% Which options are required by the car started at each step.
array [steps, options] of var 0..1: used;
```

“”

```
% Block p must be used at step s if the class of the car to be
% produced at step s needs it.
constraint forall (s in steps, p in options) (used[s, p]=need[class[s], p]);

% For each option p with block size b and maximum limit m, no consecutive
% sequence of b cars contains more than m that require option p.
constraint
    forall (p in options, i in 1..(n_cars - (block_size[p] - 1))) (
        sum (j in 0..(block_size[p] - 1)) (used[i + j, p])
            <= max_per_block[p]);

% Require that the right number of cars in each class are produced.
constraint forall (c in classes) (count(class, c, cars_in_class[c]));

solve satisfy; % Find any solution.
```

cars_data.dzn

列表cars_data.dzn

```
% cars_data.dzn
n_cars = 10;
n_options = 5;
n_classes = 6;
max_per_block = [1, 2, 1, 2, 1];
block_size =    [2, 3, 3, 5, 5];
cars_in_class = [1, 1, 2, 2, 2, 2];

need =  array2d(1..6, 1..5, [
    1, 0, 1, 1, 0,
    0, 0, 0, 1, 0,
    0, 1, 0, 0, 1,
    0, 1, 0, 1, 0,
    1, 0, 1, 0, 0,
    1, 1, 0, 0, 0
]);
```

```
% minizinc --solver globalizer cars.mzn cars_data.dzn
cars.mzn|33|12|33|69 [  ] gcc(class,cars_in_class)
cars.mzn|33|35|33|68 [  ] count(class,c,cars_in_class[c])
cars.mzn|28|27|28|65;cars.mzn|29|9|30|32 [  ] sliding_sum(0,max_per_block[p],block_
↪size[p],used[_, p])
```

‘’

‘’

gccsliding_sumsliding_sumcars.mzn|28|27|28|65i in 1..(n_cars - (block_size[p] - 1))
cars.mzn|29|27|28|65<=sum

```
constraint forall (p in options) (
    sliding_sum(0, max_per_block[p], block_size[p], used[..,p]));
```

3.7.2 Caveats

setenum_..used[_,p]used[..,p]

global_cardinalitygcc

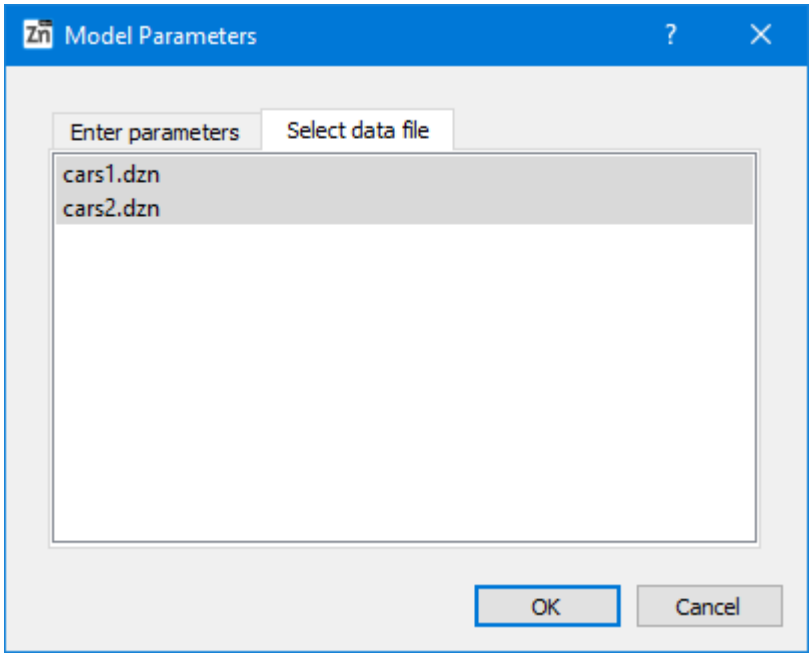
```
predicate gcc(array[int] of var int: x, array[int] of var int: counts) =
  global_cardinality(x,
                     [ i | i in index_set(counts) ],
                     array1d(counts));
```

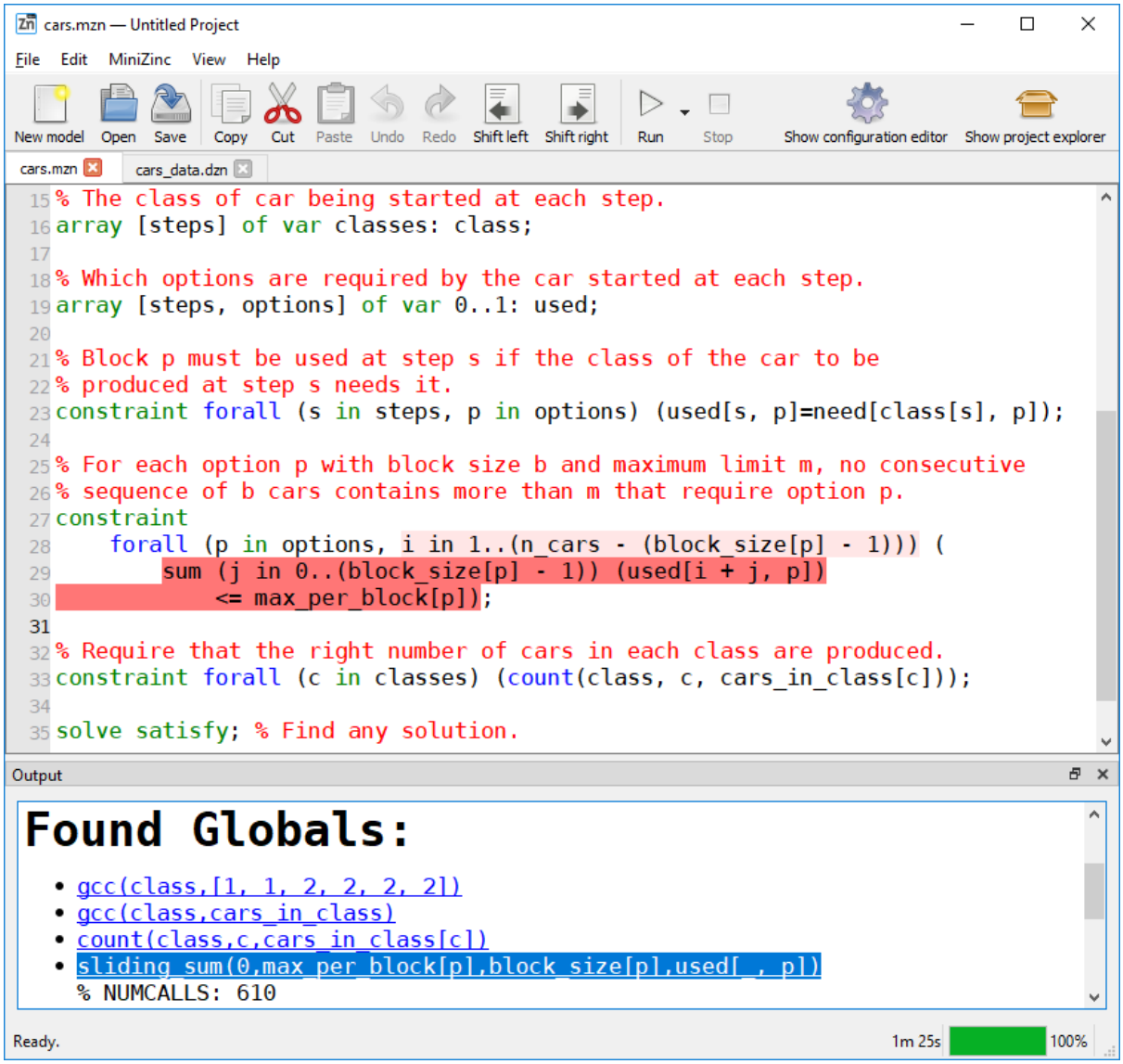
3.7.3 Supported Constraints

alldifferent	alldifferent_except_0	all_equal_int
bin_packing	bin_packing_capa	bin_packing_load
binaries_represent_int	binaries_represent_int_3A	binaries_represent_int_3B
binaries_represent_int_3C	channel	channelACB
count_geq	cumulative_assert	decreasing
gcc	global_cardinality	inverse
lex_lesseq_int_checking	lex2_checking	maximum_int_checking
minimum_int_checking	member	nvalue
strict_lex2_checking	subcircuit_checking	true
atleast	atmost	bin_packing_load_ub
circuit_checking	count	diffn
distribute	element	increasing
lex_less_int_checking	sliding_sum	sort_checking
unary	value_precede_checking	

3.7.4 Using Globalizer in the MiniZinc IDE

GlobalizerCtrlCmd


Model Parameters
Enter parameters
Select data file
cars1.dzn
cars2.dzn
OK
Cancel


cars.mzn — Untitled Project
File Edit MiniZinc View Help
New model Open Save Copy Cut Paste Undo Redo Shift left Shift right Run Stop Show configuration editor Show project explorer
cars.mzn
cars_data.dzn
% The class of car being started at each step.
array [steps] of var classes: class;
% Which options are required by the car started at each step.
array [steps, options] of var 0..1: used;
% Block p must be used at step s if the class of the car to be
% produced at step s needs it.
constraint forall (s in steps, p in options) (used[s, p]=need[class[s], p]);
% For each option p with block size b and maximum limit m, no consecutive
% sequence of b cars contains more than m that require option p.
constraint
forall (p in options, i in 1..(n_cars - (block_size[p] - 1))) (
sum (j in 0..(block_size[p] - 1)) (used[i + j, p])
<= max_per_block[p]);
% Require that the right number of cars in each class are produced.
constraint forall (c in classes) (count(class, c, cars_in_class[c]));
solve satisfy; % Find any solution.
Output
Found Globals:
gcc(class,[1, 1, 2, 2, 2, 2])
gcc(class,cars_in_class)
count(class,c,cars_in_class[c])
sliding_sum(0,max_per_block[p],block_size[p],used[_, p])
% NUMCALLS: 610
Ready.
1m 25s
100%

3.7.5 How it works

C1 /\ C2;C1; C2

forallforallforall(...) (C1 /\ C2);forall(...) (C1); forall(...) (C2)

--numConstraints

forall(i in 1..n) (c(i));c(1)c(n)

--randomSolutions

alldifferent

--sampleSolutions

3.7.6 Performance tips

--no-initial-pass

--constraintFilter-f

-t--free-search’

3.7.7 Limitations / Future work

alldifferent

CHAPTER 3.8

FindMUS

3.8.1 Basic Usage

```
findMUS model.mzn data-1.dzn
--paramset fzn-a-n <count>gecode_presolver--subsolver--solver cbc
```

3.8.1.1 Commandline arguments

```
,
-n &lt;count&gt;
-a-n 0
-t--timeout &lt;s&gt
--paramset hint,mzn,fzn
     mznfzn
     hint——
     mzn——
     fzn——
--no-leftover
     --no-leftover
```

“”

```
--frequent-stats
--no-progress%%%mzn-progress
--no-stats
--output-{html, json, brief, human}
     html
     json
     brief
     human

-g--domainsminizinc -c -g
--verbose-compile--verbose

' shrink
--shrink-alg lin,map_lin,qx,map_qx,qx2
          lin
          map_lin
          qxQuickXplain
          map_qxQuickXplain
          qx2map_qx
--depth mzn,fzn,&lt;n&gt
          mzn
          fzn
          &lt;n&gt
--restarts
--seed <n>map_qx

minizinc--solver--fzn-flags
--subsolver &lt;s&gt
     “”
--solver-flags &lt;f&gt
--solver-timelimit &lt;ms&gt
--adapt-timelimit
```

“”

```
--soft-defines
--hard-domains
    -g
--named-only
--filter-mode fg,ex
    fg
    ex
--filter-named &lt;names&gt;--filter-named-exclude &lt;names&gt;
--filter-path &lt;paths&gt;--filter-path-exclude &lt;paths&gt;
--filter-sep &lt;sep&gt;

,
--structure flat,gen,normal,mix
    gen
    flat
    gen
    normal
    mixgennormal
    idx
    idxmixidxnormal
--no-binarize



--verbose-{map,enum,subsolve} &lt;n&gtn
--verbose

--dump-dot &lt;dot&gt
```

3.8.1.2 Example

列表latin_squares.mzn

```
% latin_squares.mzn
include "globals.mzn";

int: n = 3;
set of int: N = 1..n;
```

```
array[N, N] of var N: X;

constraint :: "ADRows"
    forall (i in N)
        (alldifferent(X[i,..])           :: "AD(row \(i))"
        );
constraint :: "ADCols"
    forall (j in N)
        (alldifferent(X[..,j])           :: "AD(col \(j))"
        );

constraint :: "LLRows"
    forall (i in 1..n-1)
        (lex_less(X[i,..], X[i+1,..])    :: "LL(rows \(i) \(i+1))"
        );
constraint :: "LGCols"
    forall (j in 1..n-1)
        (lex_greater(X[..,j], X[..,j+1]) :: "LG(cols \(j) \(j+1))"
        );

solve satisfy;

output [ show2d(X) ];
```

ADRowsADColsalldifferentLLRowsLGColslex

findMUS -n 2 latin_squares.mzn-n 2

```
FznSubProblem:  hard cons: 0    soft cons: 10   leaves: 10      branches: 11   ␣
↪Built tree in 0.02854 seconds.
MUS: 7 3 6 4 8
Brief: gecode_all_different_int;@{ADCols@AD(col 1)}:(j=1)
gecode_all_different_int;@{ADCols@AD(col 2)}:(j=2)
gecode_array_int_lt;@{LGCols@LG(cols 1 2)}:(j=1)
gecode_array_int_lt;@{LLRows@LL(rows 1 2)}:(i=1)
gecode_array_int_lt;@{LLRows@LL(rows 2 3)}:(i=2)

Traces:
latin_squares.mzn|18|5|20|9|ca|forall;latin_squares.mzn|18|5|20|9|ac;latin_squares.
↪mzn|18|13|18|13|i=2;latin_squares.mzn|19|10|19|37|ca|lex_less
latin_squares.mzn|13|5|15|9|ca|forall;latin_squares.mzn|13|5|15|9|ac;latin_squares.
↪mzn|13|13|13|13|j=1;latin_squares.mzn|14|10|14|30|ca|alldifferent
latin_squares.mzn|18|5|20|9|ca|forall;latin_squares.mzn|18|5|20|9|ac;latin_squares.
↪mzn|18|13|18|13|i=1;latin_squares.mzn|19|10|19|37|ca|lex_less
latin_squares.mzn|13|5|15|9|ca|forall;latin_squares.mzn|13|5|15|9|ac;latin_squares.
↪mzn|13|13|13|13|j=2;latin_squares.mzn|14|10|14|30|ca|alldifferent
latin_squares.mzn|22|5|24|9|ca|forall;latin_squares.mzn|22|5|24|9|ac;latin_squares.
↪mzn|22|13|22|13|j=1;latin_squares.mzn|23|10|23|40|ca|lex_greater

MUS: 7 3 5 6 8 9
Brief: gecode_all_different_int;@{ADCols@AD(col 1)}:(j=1)
gecode_all_different_int;@{ADCols@AD(col 3)}:(j=3)
gecode_array_int_lt;@{LGCols@LG(cols 1 2)}:(j=1)
gecode_array_int_lt;@{LGCols@LG(cols 2 3)}:(j=2)
gecode_array_int_lt;@{LLRows@LL(rows 1 2)}:(i=1)
gecode_array_int_lt;@{LLRows@LL(rows 2 3)}:(i=2)
```

```
Traces:
latin_squares.mzn|18|5|20|9|ca|forall;latin_squares.mzn|18|5|20|9|ac;latin_squares.
↪mzn|18|13|18|13|i=2;latin_squares.mzn|19|10|19|37|ca|lex_less
latin_squares.mzn|13|5|15|9|ca|forall;latin_squares.mzn|13|5|15|9|ac;latin_squares.
↪mzn|13|13|13|13|j=1;latin_squares.mzn|14|10|14|30|ca|alldifferent
latin_squares.mzn|13|5|15|9|ca|forall;latin_squares.mzn|13|5|15|9|ac;latin_squares.
↪mzn|13|13|13|13|j=3;latin_squares.mzn|14|10|14|30|ca|alldifferent
latin_squares.mzn|18|5|20|9|ca|forall;latin_squares.mzn|18|5|20|9|ac;latin_squares.
↪mzn|18|13|18|13|i=1;latin_squares.mzn|19|10|19|37|ca|lex_less
latin_squares.mzn|22|5|24|9|ca|forall;latin_squares.mzn|22|5|24|9|ac;latin_squares.
↪mzn|22|13|22|13|j=1;latin_squares.mzn|23|10|23|40|ca|lex_greater
latin_squares.mzn|22|5|24|9|ca|forall;latin_squares.mzn|22|5|24|9|ac;latin_squares.
↪mzn|22|13|22|13|j=2;latin_squares.mzn|23|10|23|40|ca|lex_greater
Total Time: 0.39174     nmuses: 2       map:     40     sat:     21     total:    ␣
↪61
```

FznSubProblem:findMUS

MUS:

Brief:

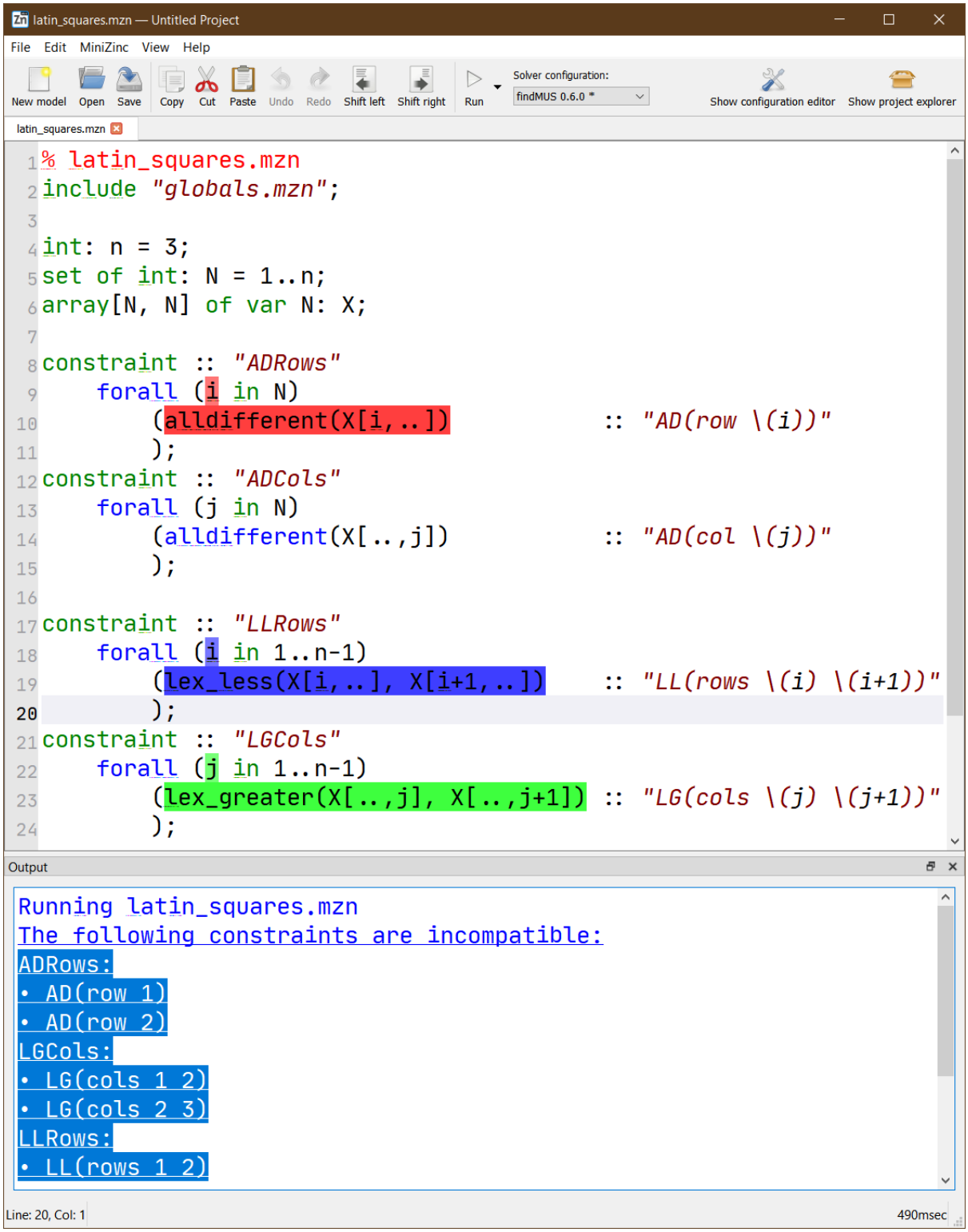
Traces:;ca|forall

Total Timemapsat

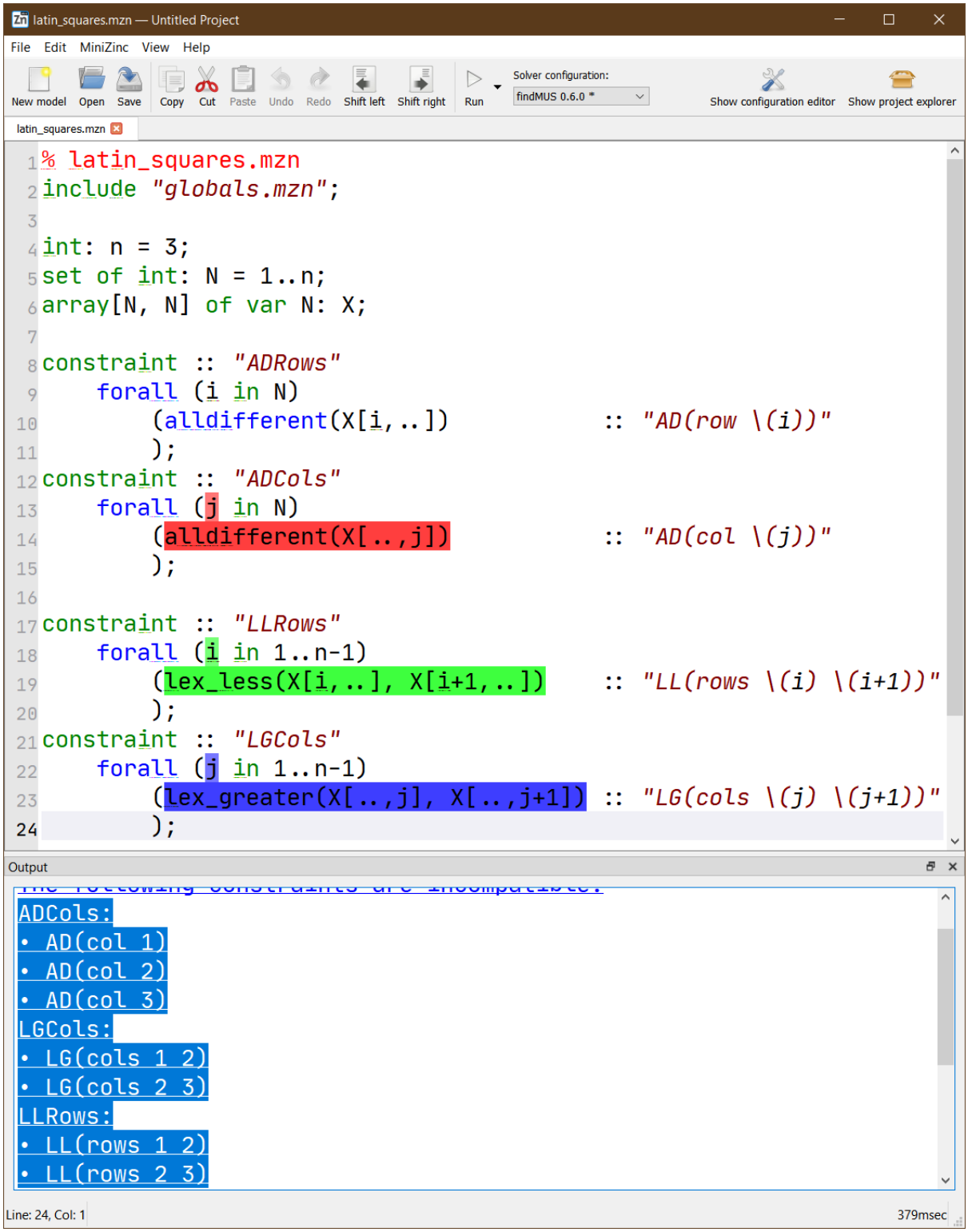
lexalldifferent

3.8.2 Using FindMUS in the MiniZinc IDE

FindMUS--paramset mzn

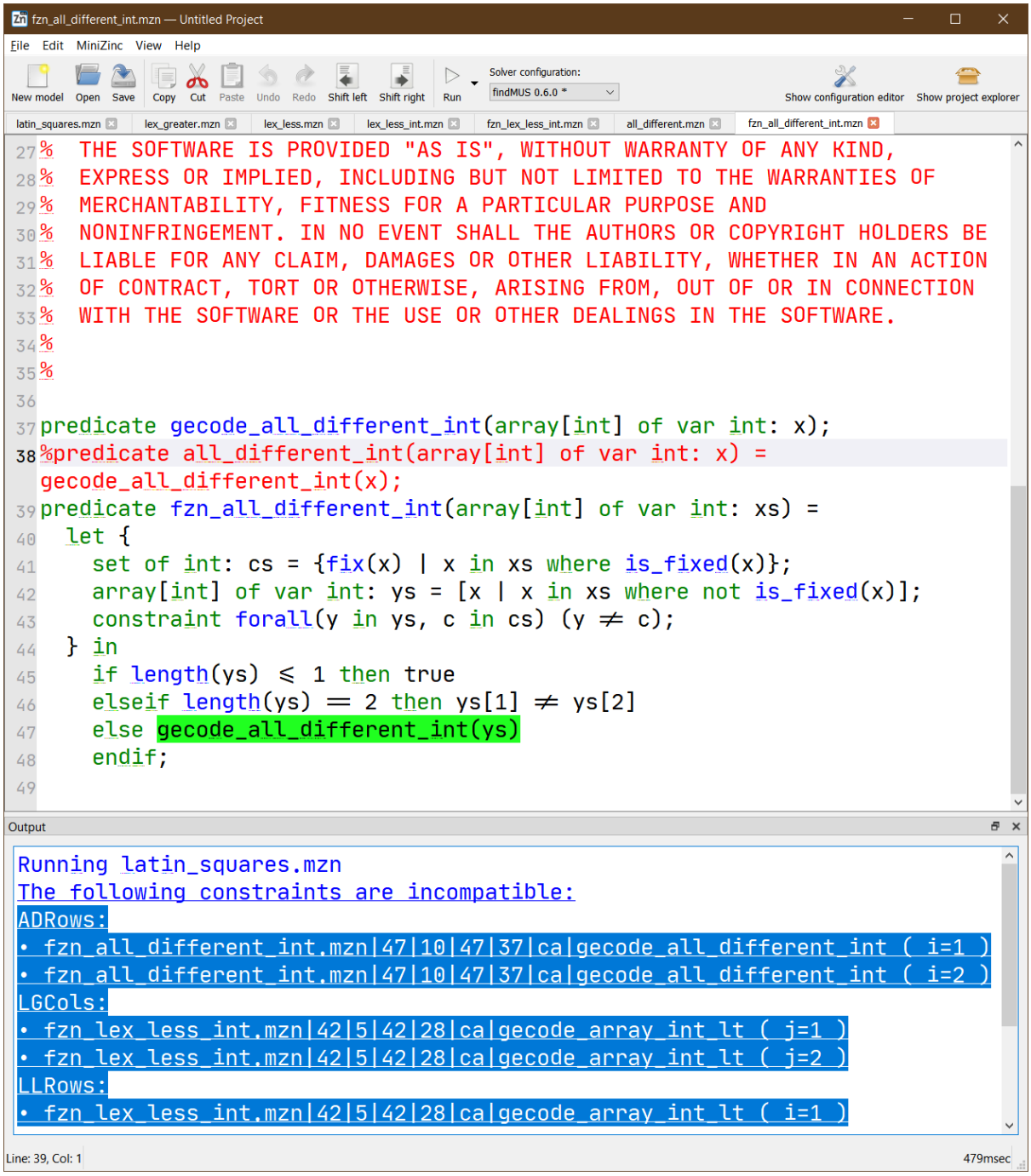



```
--paramset mzn--paramset hint
```



ADRowsLLRowsLGCols

--paramset fzn



’ fzn_all_different_int

3.8.3 How it works

’ ‘’

“”

3.8.4 Performance tips

--depth--filter-name--filter-path

```
findMUS -a --paramset fzn --output-brief ... > muses.log
findMUS -a --paramset fzn --output-brief --oracle-only muses.log ...--oracle-only
```

3.8.5 Limitations / Future work

CHAPTER 3.9

Mzn-Analyse

mzn-analyse “”

get-diversity-annsdiverse_solutions()

3.9.1 Basic Usage

mzn-analyse input.mzn [passes]

[passes]cmd:arg1,arg2,...

out:-out_fzn:-outout

json_outjson_clear

3.9.1.1 Passes: read/write models & json

in:in.mzn

out:out.mzn

out_fzn:out.fzn

no_out

json_out:out.json

json_clear

no_json

3.9.1.2 Passes: Manipulate Models

```
inline-includes
inline-all-includes
remove-anns:name1,[name2,...]
remove-includes:name1,[name2,...]
output-all ‘’
remove-stdlib
get-items:idx1,[idx2,...]…
filter-items:iid1,[iid2,...]
remove-items:iid1,[iid2,...]
filter-typeinst:{var|par}
replace-with-newvar:location1,location2 ‘’
repeat-model:kk
get-solve-anns
```

3.9.1.3 Passes: Special Passes

```
get-diversity-anns
annotate-data-deps
get-term-types:out.terms
get-data-deps:out.cons
get-exprs:location1,location2
get-ast:location1,location2
```

3.9.2 Examples

```
mzn-analyse
  “”
mzn-analyse in.mzn remove-items:solve,output out:solveless.mzn inline-includes
out_fzn


mzn-analyse in.fzn filter-items:constraint get-items:50 remove-anns


mzn-analyse rcpsp-wet-r0.annotated.fzn filter-items:constraint get-items:60,61,62
get-data-deps no_out
```

```
{"constraint_info": [
  [
    ["in", "i", "Tasks"],
    ["in", "j", "suc[i]"],
    ["assign", "j", "24"],
    ["assign", "i", "2"],
    ["eq", "suc[i]", "23..24"],
    ["eq", "Tasks", "1..32"]],
  [
    ["in", "i", "Tasks"],
    ["in", "j", "suc[i]"],
    ["assign", "j", "5"],
    ["assign", "i", "3"],
    ["eq", "suc[i]", "{5,6,17}"],
    ["eq", "Tasks", "1..32"]],
  [
    ["in", "i", "Tasks"],
    ["in", "j", "suc[i]"],
    ["assign", "j", "6"],
    ["assign", "i", "3"],
    ["eq", "suc[i]", "{5,6,17}"],
    ["eq", "Tasks", "1..32"]]]}
```


3.9.3 Limitations / Future work

`mzn-analyse`

CHAPTER 3.10

Using MiniZinc in Jupyter Notebooks

iminizinc “”

3.10.1 Installation

pip

```
pip install -U iminizinc
```

--user

minizincPATH “” PATH

3.10.2 Basic usage

%load_ext iminizinc%%minizinc

```
In[1]:  %load_ext iminizinc

In[2]:  n=8

In[3]:  %%minizinc

        include "globals.mzn";
        int: n;
        array[1..n] of var 1..n: queens;
        constraint all_different(queens);
        constraint all_different([queens[i]+i | i in 1..n]);
        constraint all_different([queens[i]-i | i in 1..n]);
        solve satisfy;
Out[3]: {u'queens': [4, 2, 7, 3, 6, 8, 5, 1]}
```

n

```
In[1]:  %load_ext iminizinc

In[2]:  n=8

In[3]:  %%minizinc -m bind

        include "globals.mzn";
        int: n;
        array[1..n] of var 1..n: queens;
        constraint all_different(queens);
        constraint all_different([queens[i]+i | i in 1..n]);
        constraint all_different([queens[i]-i | i in 1..n]);
        solve satisfy;

In[4]:  queens

Out[4]: [4, 2, 7, 3, 6, 8, 5, 1]
```

-a

```
In[1]:  %load_ext iminizinc

In[2]:  n=6

In[3]:  %%minizinc -a

        include "globals.mzn";
        int: n;
        array[1..n] of var 1..n: queens;
        constraint all_different(queens);
        constraint all_different([queens[i]+i | i in 1..n]);
        constraint all_different([queens[i]-i | i in 1..n]);
        solve satisfy;

Out[3]: [{u'queens': [5, 3, 1, 6, 4, 2]},
         {u'queens': [4, 1, 5, 2, 6, 3]},
         {u'queens': [3, 6, 2, 5, 1, 4]},
         {u'queens': [2, 4, 6, 1, 3, 5]}]
```

```
In[1]:  %%minizinc?
```

CHAPTER 3.11

Python Interface

警告

CHAPTER 3.12

JavaScript Interface

CHAPTER 3.13

Installation from Source Code

```
masterdevelop' README.md
'
```

CHAPTER 4.1

Specification of MiniZinc

4.1.1 Introduction

4.1.1.1 Original authors.

_

4.1.2 Notation

<item>

"constraint""""

["var"]

("," <ident>)*

(<msg>)+

```
<expr> "," ...
```

```
<expr> ( "," <expr> )* [ "," ]
```

[-+]?[0-9]+

,

\n

4.1.3 Overview of a Model

“”

4.1.3.1 Evaluation Phases

4.1.3.2 Run-time Outcomes

```
<output> ::= <no-solutions> [ <warnings> ] <free-text>
           | ( <solution> )* [ <complete> ] [ <warnings> ] <free-text>
```

4.1.3.3 Output

application/x-zinc-output

application/x-zinc-output

```
<solution> ::= <solution-text> [ \n ] "----------" \n
```

```
<no-solutions> ::= "=====UNSATISFIABLE=====" \n
```

```
<complete> ::= "==========" \n
```

```
<warnings> ::= ( <message> )+

<message>  ::= ( <line> )+
```

%

```
=====UNSATISFIABLE=====
% trentin.fzn:4: warning: model inconsistency detected before search.
```

4.1.4 Syntax Overview

4.1.4.1 Character Set

4.1.4.2 Comments

%/**/

4.1.4.3 Identifiers

```
<ident> ::= _?[A-Za-z][A-Za-z0-9_]*       % excluding keywords
          | "'" [^'\xa\xd\x0]+ "'"
```

```
my_name_2
MyName2
'An arbitrary identifier'
```

annannotationanyarraybool**case**constraintdiffdivelseelseifendifenumfalsefloatfunctionifininclude
intintersectletlistmaximizeminimizemodnotofopoptoutputparpredicaterecordsatisfysetsolvestring
subsetsupersetsymdifftestthen**true**tupletypeunionvarwherexor

4.1.5 High-level Model Structure

```
<model> ::= [ <item> ";" ... ]
```

```
<item>  ::= <include-item>
          | <var-decl-item>
          | <enum-item>
          | <type-inst-syn-item>
          | <assign-item>
          | <constraint-item>
          | <solve-item>
          | <output-item>
          | <predicate-item>
          | <test-item>
          | <function-item>
```

```
            | <annotation-item>

  <ti-expr-and-id> ::= <ti-expr> ":" <ident>
```

“”

ann

4.1.5.1 Model Instance Files

4.1.5.2 Namespaces

x

4.1.5.3 Scopes

4.1.6 Types and Type-insts

ann

4.1.6.1 Properties of Types

annspec-Setsspec-Arrays

ann><===!=<=>=

4.1.6.2 Instantiations

’

“”

ann

4.1.6.3 Type-insts

’ var int

par intvar intvar intpar int$\xrightarrow{c}$var int

par intvar intpar int$\xrightarrow{v}$var int

4.1.6.4 Type-inst expression overview

```
<ti-expr> ::= <base-ti-expr>
            | <array-ti-expr>

<base-ti-expr> ::= <var-par> <opt-ti> <set-ti> <base-ti-expr-tail> ["++" <base-ti-
↪expr>]
                 | "any" <ti-variable-expr-tail>

<var-par> ::= "var" | "par" |

<opt-ti> ::= "opt" |

<set-ti> ::= "set" "of" |

<base-type> ::= "bool"
```

```
                | "int"
                | "float"
                | "string"

  <base-ti-expr-tail> ::= <ident>
                        | <base-type>
                        | <ti-variable-expr-tail>
                        | <tuple-ti-expr-tail>
                        | <record-ti-expr-tail>
                        | "ann"
                        | "{" <expr> "," ... "}"
                        | <num-expr> ".." <num-expr>
```

<num-expr><expr>..<expr>

parvarpar

```
  par int
  int
```

var

varann

varvar

4.1.6.5 Built-in Scalar Types and Type-insts

Booleans

bool2int’

boolpar boolvar bool

false, true

par bool$\xrightarrow{v}$var boolvar bool$\xrightarrow{v}$var bool

falsetrue

par bool$\xrightarrow{c}$var bool

Integers

intpar intvar int

par int$\xrightarrow{v}$var intvar int$\xrightarrow{v}$var int

par int$\xrightarrow{c}$var intpar bool$\xrightarrow{c}$par intpar bool$\xrightarrow{c}$var intvar bool$\xrightarrow{c}$var int

Floats

NaN

floatpar floatvar float

par float$\xrightarrow{v}$var floatvar float$\xrightarrow{v}$var float

par int$\xrightarrow{c}$par floatpar int$\xrightarrow{c}$var floatvar int$\xrightarrow{c}$var floatpar float$\xrightarrow{c}$var float

Strings

stringpar string

show

4.1.6.6 Built-in Compound Types and Type-insts

Sets

,

```
<base-ti-expr> ::= <var-par> <opt-ti> "set" "of" <base-ti-expr-tail>
```

```
set of int
var set of bool
```

set of 1..2powerset(1..2){}, {1}, {1,2}, {2}

par set of TI$\xrightarrow{v}$var set of TIvar set of TI$\xrightarrow{v}$var set of TI

{1,2}{2,1}{} < {1,3} < {2}

par set of TI$\xrightarrow{c}$par set of UIpar set of TI$\xrightarrow{c}$var set of UIvar set of TI$\xrightarrow{c}$var set of UITI$\xrightarrow{c}$UI

Arrays

```
array[1..3] of int: a1;
array[0..3] of int: a2;
array[1..5, 1..10] of var float: a5;
```

,

```
a1 = [4,6,4,3,2];   % too many elements
a2 = [3,2,6,5];     % index set mismatch
a5 = [||];          % too few elements
```

a2[3,2,6,5]1..40..3array1d

```
a2 = array1d(0..3,[3,2,6,5]);     % correct
```

```
array[int,int] of int: a6;
```

```
enum X = {A,B,C};
enum Y = {D,E,F};
array[X] of int: x = array1d(X, [5,6,7]); % correct
array[Y] of int: y = x;                   % index set mismatch: Y != X
array[int] of int: z = x;                 % correct: assign X index set to int
array[X] of int: x2 = [10,11,12];         % correct: automatic coercion for array␣
↪literals
```

,

```
<array-ti-expr> ::= "array" [ <ti-expr> "," ... ] "of" <base-ti-expr>
                  | "list" "of" <base-ti-expr>
```

```
array[1..10] of int
list of var int
```

list of <T>array[int] of <T>

var

[1, 1][1, 2][1, 2, 3]array1d(2..4,[0, 0, 0])[1, 2, 3]

array[TI0] of TI$\xrightarrow{c}$array[UI0] of UITI0$\xrightarrow{c}$UI0TI$\xrightarrow{c}$UI

Option Types

optMaybe<>

boolintfloat

opt <T><T>

<><

optopt<>

TI$\xrightarrow{c}$opt UITI$\xrightarrow{c}$UI

Tuple Types

```
<tuple-ti-expr-tail> ::= "tuple" ( <ti-expr> "," ... )
```

```
tuple(int, var float)
```

tuple(1..2, {3,5}){(1,3), (1,5), (2,3), (2,5)}

tuple(TI1, ..., TIn)$\xrightarrow{c}$tuple(UI1, ..., UIn)TI1$\xrightarrow{c}$UI1$\cdots$TIn$\xrightarrow{c}$UIn

Record Types

,

,

```
record(var int: x, var int: y)
record(var int: y, var int: x)
```

```
<record-ti-expr-tail> ::= "record" ( <ti-expr-and-id> "," ... )
```

```
record(int: x, int: y)
```

record(1..2: x, {3,5}: y){(x: 1,y: 3), (x: 1,y: 5), (x: 2,y: 3), (x: 2,y: 5)}

record(TI1: x1, ..., TIn: xn)$\xrightarrow{c}$record(UI1: x1, ..., UIn: xn)TI1$\xrightarrow{c}$UI1$\cdots$TIn$\xrightarrow{c}$UIn

The Annotation Type

ann

annvar

ann

ann

4.1.6.7 User-defined Types and Type-insts

Enumerated Types

XXpar Xvar X

par X$\xrightarrow{v}$var Xvar X$\xrightarrow{v}$var X

par X$\xrightarrow{c}$par intvar X$\xrightarrow{c}$var int

Type-inst Synonyms

MyFixedIntpar intMyFixedIntpar int

varparpar

“”

4.1.6.8 Constrained Type-insts

Set Expression Type-insts

```
    1..3
    var {1,3,5}
```

’ var intvar 1..3

Float Range Type-insts

```
1.0 .. 3.01.0 .. 3.01 .. 3
```

4.1.7 Expressions

4.1.7.1 Expressions Overview

```
<expr> ::= <expr-atom> <expr-binop-tail>

<expr-atom> ::= <expr-atom-head> <expr-atom-tail> <annotations>

<expr-binop-tail> ::= [ <bin-op> <expr> ]

<expr-atom-head> ::= <builtin-un-op> <expr-atom>
                   | "(" <expr> ")"
                   | <ident-or-quoted-op>
                   | "_"
                   | <bool-literal>
                   | <int-literal>
                   | <float-literal>
                   | <string-literal>
                   | <set-literal>
                   | <set-comp>
                   | <array-literal>
                   | <array-literal-2d>
                   | <indexed-array-literal>
                   | <indexed-array-literal-2d>
                   | <tuple-literal>
                   | <record-literal>
                   | <array-comp>
                   | <indexed-array-comp>
                   | <ann-literal>
                   | <if-then-else-expr>
                   | <let-expr>
                   | <call-expr>
                   | <gen-call-expr>

<expr-atom-tail> ::=
                   | <array-access-tail> <expr-atom-tail>
```

axnot

```
not x::a;
not (x)::a;
not(x)::a;
'not'(x)::a;
```

<ident><gen-call-expr>

```
<num-expr> ::= <num-expr-atom> <num-expr-binop-tail>

<num-expr-atom> ::= <num-expr-atom-head> <expr-atom-tail> <annotations>

<num-expr-binop-tail> ::= [ <num-bin-op> <num-expr> ]

<num-expr-atom-head> ::= <builtin-num-un-op> <num-expr-atom>
                       | "(" <num-expr> ")"
                       | <ident-or-quoted-op>
                       | <int-literal>
                       | <float-literal>
                       | <if-then-else-expr>
                       | <let-expr>
                       | <call-expr>
                       | <gen-call-expr>
```

4.1.7.2 Operators

+

3 + 4not x

```
<builtin-op> ::= <builtin-bin-op> | <builtin-un-op>

<bin-op> ::= <builtin-bin-op> |  `<ident> `

<builtin-bin-op> ::= "<->" | "->" | "<-" | "\/" | "xor" | "/\"
                   | "<" | ">" | "<=" | ">=" | "==" | "=" | "!="
                   | "~=" | "~!="
                   | "in" | "subset" | "superset" | "union" | "diff" | "symdiff"
                   | ".." | "intersect" | "++" | "default" | <builtin-num-bin-op>

<builtin-un-op> ::= "not" | <builtin-num-un-op>
```

```
<num-bin-op> ::= <builtin-num-bin-op> |  `<ident> `

<builtin-num-bin-op> ::= "+" | "-" | "*" | "/" | "div" | "mod" | "^"
                       | "~+" | "~-" | "~*" | "~/" | "~div"

<builtin-num-un-op> ::= "+" | "-"
```

表

表

Operator	Unicode symbol	UTF-8 code
<->	↔	
->	→	
<-	←	
not	¬	
\/	∨	
/\	∧	
!=	≠	
<=	≤	
>=	≥	
in	∈	
subset	⊆	
superset	⊇	
union	∪	
intersect	∩	
^-1	$^{-1}$	

表1+2*31+(2*3)*+1+2+3(1+2)+3+a++b++ca++(b++c)++1<x<2<

^-1

表

Symbol(s)	Assoc.	Prec.
<->		
->		
<-		
\/		
xor		
/\		
<		
>		
<=		
>=		
==		
=		
!=		
in		
subset		
superset		
union		
diff		
symdiff		
..		
<..		
..<		
<..<		
+		
-		
*		
div		

续下页

表 1.2 – 接上页

Symbol(s)	Assoc.	Prec.
mod		
/		
intersect		
^		
++		
default		
`<ident>`		

```
A `min2` B
```

+-not

'+'(3, 4)3 + 4

4.1.7.3 Expression Atoms

Identifier Expressions and Quoted Operator Expressions

```
<ident-or-quoted-op> ::= <ident>
                       | ' <builtin-op>'
```

```
'+'
'union'
```

,

Anonymous Decision Variables

_

```
array[1..4] of var int: xs = [1, _, 3, _];
```

_

Boolean Literals

```
<bool-literal> ::= "false" | "true"
```

Integer and Float Literals

```
<int-literal> ::= [0-9]+
                | 0x[0-9A-Fa-f]+
                | 0o[0-7]+
```

```
00051230x1b70o777-1
```

```
<float-literal> ::= [0-9]+"."[0-9]+
                  | [0-9]+"."[0-9]+[Ee][-+]?[0-9]+
                  | [0-9]+[Ee][-+]?[0-9]+
                  | 0[xX]([0-9a-fA-F]*"."[0-9a-fA-F]+|[0-9a-fA-F]+".")([pP][+-]?[0-
↪9]+)
                  | (0[xX][0-9a-fA-F]+[pP][+-]?[0-9]+)
```

```
1.051.3e-51.3+e51..51.e5.1e5-1.0-1E05--
```

String Literals and String Interpolation

```
<string-contents> ::= ([^"\n\] |  \[0-7][0-7?][0-7]? | \x[0-9a-fA-F][0-9a-fA-F]? | \
↪n | \t | \" | \\)*

<string-literal> ::= """ <string-contents> """
                   | """ <string-contents> "\(" <string-interpolate-tail>

<string-interpolate-tail> ::= <expr> ")"<string-contents>"""
                            | <expr> ")"<string-contents>"\(" <string-interpolate-
↪tail>
```

```
    \"

    \\

    \n

    \t

    \x[0-9a-fA-F][0-9a-fA-F]?

    \x[0-7][0-7]?[0-7]?
```

```
"Hello, world!\n"
```

```
"\xe4\xbd\xa0\xe5\xa5\xbd means hello"
```

```
string: s = "This is a string literal "
         ++ "split across two lines.";
```

```
show
```

```
var set of 1..10: q;
solve satisfy;
output [show("The value of q is \(q), and it has \(card(q)) elements.")];
```

Set Literals

```
<set-literal> ::= "{" [ <expr> "," ... ] "}"
```

```
{ 1, 3, 5 }
{ }
{ 1, 2.0 }
```

```
1float
```

Set Comprehensions

```
<set-comp> ::= "{" <expr> "|" <comp-tail> "}"

<comp-tail> ::= <generator> [ "where" <expr> ] "," ...

<generator> ::= ( <ident> | "_" ) "," ... "in" <expr>
```

```
{ 2*i | i in 1..5 }     % { 2, 4, 6, 8, 10 }
{  1  | i in 1..5 }     % { 1 }   (no duplicates in sets)
```

```
|in
```

```
{ i | i in 1..10 where (i mod 2 = 0) }     % { 2, 4, 6, 8, 10 }
```

```
{ 3*i+j | i in 0..2, j in {0, 1} }    % { 0, 1, 3, 4, 6, 7 }
```

```
|
```

```
{ i+j | i in 1..3, j in 1..i }  % { 1+1, 2+1, 2+2, 3+1, 3+2, 3+3 }
```

```
[f(i, j) | i in A1 where p(i), j in A2 where q(i,j)]
```

Simple Array Literals

```
<array-literal> ::= "[" [ <expr> "," ... ] "]"
```

```
[1, 2, 3, 4]
[]
[1, _]
```

```
1var int
```

```
1..nn
```

Indexed Array Literals

```
<indexed-array-literal> ::= "[" [ ( <index-tuple> ":" <expr> ) "," ... ] "]"
                          | "[" [ <index-tuple> ":" <expr> "," <expr> "," ... ] "]"

<index-tuple> ::= <expr>
                | "(" [ <expr> "," ... ] ")"
```

```
[ 1: 1, 2: 2, 3: 3, 4: 4, 5: 5]
[ A: 0, B: 3, C: 5]
[ (1,2): 1, (1,3): 2, (2,2): 3, (2,3): 4]
[ 1: 1, 4: 2, 5: 3, 3: 4, 2: 5]
[ 0: A, B, C ]
```

1..5{A,B,C}1..22..3

“”

```
% not contiguous, index 2 missing:
[ 1: 1, 3:2, 4:, 3]

% not contiguous, index 2 missing in dimension 2:
[ (1,1): 0, (1,3): 1, (2,1): 0, (2,3): 2]

% not rectangular, second row has fewer entries than first row:
[ (1,1): 0, (1,2): 0, (1,3): 0, (2,1): 1, (2,2): 2]
```

2d Array Literals

```
<array-literal-2d> ::= "[|" [ [ <expr> "," ... ] "|" ... ] "|]"
```

```
[| 1, 2, 3
 | 4, 5, 6
 | 7, 8, 9 |]       % array[1..3, 1..3]
[| x, y, z |]       % array[1..1, 1..3]
[| 1 | _ | _ |]     % array[1..3, 1..1]
```

1var int

(1,1)..(m,n)mn

Indexed 2d Array Literals

```
<indexed-array-literal-2d> ::= "[|" [ <expr> ":" ... ] [ [ <expr> ":" ] <expr> "," .
↪.. ] "|]"
```

```
% only column index:
[| A: B: C:
 | 0, 0, 0
 | 1, 1, 1
 | 2, 2, 2 |];

% only row index:
[| A: 0, 0, 0
 | B: 1, 1, 1
 | C: 2, 2, 2 |];
```

```
% row and column index:
[|    A: B: C:
 | A: 0, 0, 0
 | B: 1, 1, 1
 | C: 2, 2, 2 |];
```

1..nn

Simple Array Comprehensions

```
<array-comp> ::= "[" <expr> "|" <comp-tail> "]"
```

in

```
[2*i | i in 1..5]         % [2, 4, 6, 8, 10]
[2*i | i in [3,5,7] ]     % [6, 10, 14]
[i   | i in [|1,2|3,4|] ] % [1, 2, 3, 4]
```

_n*n

```
[ 1 | _, _ in 1..n ]
```

```
var set of 1..5: x;
array[int] of var opt int: y = [ i * i | i in x ];
```

var bool

```
var int x;
array[int] of var opt int: y = [ i | i in 1..10 where i != x ];
```

1..nn

Indexed Array Comprehensions

```
<indexed-array-comp> ::= "[" <index-tuple> ":" <expr> "|" <comp-tail> "]"
```

```
% This equals [ 3:9, 4:12, 5:15 ]:
[ i: 3*i | i in 3..5 ]

% This generates a 2d array
% [|    1:  2:  3:
%  | 2:  7,  8,  9
%  | 3: 10, 11, 12
%  | 4: 13, 14, 15
%  |]:
[ (i,j): i*3+j | i in 2..4, j in 1..3]
```

Array Access Expressions

```
<array-access-tail> ::= "[" <expr> "," ... "]"
```

```
int: x = a1[1];
```

```
array[1..2] of int: a2 = [1, 2];
var int: i;
```

a2[i]var int

```
array[1..3,1..3] of int: a3;
int: y = a3[1, 2];
```

```
enum X = {A,B,C};
array[X] of int: a4 = [1,2,3];
int: y = a4[1];                  % wrong index type
int: z = a4[B];                  % correct
```

Array Slice Expressions

```
array[1..n,4..8] of int: x;
array[int] of int: row_2_of_x = x[2,4..8];
```

```
row_2_of_x4..8
```

```
array[1..n,4..8] of int: x;
array[int] of int: row_2_of_x = x[2,..];
```

```
array[1..n,4..8] of int: x;
array[int] of int: row_2_of_x = x[2,5..6];
```

```
row_2_of_x5..6
```

```
array[1..n,4..8] of int: x;
array[int] of int: row_2_of_x = x[2,..6];
```

```
4..66..6..8
```

..<....<<..<节

```
enum X = {A, B, C, D};
array[X, 1..3] of var int: x;
% x1 = x[A..C, 2..3]
array[_, _] of var int: x1 = x[A..<, <..];
% x2 = x[B..C, 3..3]
array[_, _] of var int: x2 = x[<..<, 2<..];
```

Annotation Literals

ann

```
<ann-literal> ::= <ident> [ "(" <expr> "," ... ")" ]
```

```
foo
cons(1, cons(2, cons(3, nil)))
```

If-then-else Expressions

```
<if-then-else-expr> ::= "if" <expr> "then" <expr> [ "elseif" <expr> "then" <expr>␣
↪]* "else" <expr> "endif"
```

```
if x <= y then x else y endif
if x < 0 then -1 elseif x > 0 then 1 else 0 endif
```

endif

ifpar boolvar boolthenelse

ifvar boolthenelse

ifpar boolthenelsevar bool

Let Expressions

```
<let-expr> ::= "let" "{" <let-item> ";" ... "}" "in" <expr>

<let-item> ::= <var-decl-item>
             | <constraint-item>
```

```
let { int: x = 3; int: y = 4; } in x + y;
let { var int: x;
      constraint x >= y /\ x >= -y /\ (x = y \/ x = -y); }
in x
```

,

in

```
let { int: x = 3; int: y = x; } in x + y;  % ok
let { int: y = x; int: x = 3; } in x + y;  % x not visible in y's defn.
let { int: x = x; } in x;                  % x not visible in x's defn.
let { int: x = 3; int: x = 4; } in x;      % x declared twice
```

```
let { var int: x; } in ...;    % ok
let {     int: x; } in ...;    % illegal
```

any

```
let { any: x = [5,6,7]; } in ...;    % x has type array[1..3] of int
```

in

```
not XX <-> Y}XX -> YY <- Xbool2int(X)
```

```
constraint b -> x + let { var 0..2: y; constraint y != -1;} in y >= 4;
```

```
var 0..2: y;
constraint b -> (x + y >= 4 /\ y != 1);
```

let

Call Expressions

```
<call-expr> ::= <ident-or-quoted-op> [ "(" <expr> "," ... ")" ]
```

```
x = min(3, 5);
```

/\\/-><-assert

A -> BAB

Generator Call Expressions

```
<gen-call-expr> ::= <ident-or-quoted-op> "(" <comp-tail> ")" "(" <expr> ")"
```

```
P(Gs)(E)P([E | Gs])
```

```
forall(i,j in Domain where i<j)
    (noattack(i, j, queens[i], queens[j]));
```

```
forall( [ noattack(i, j, queens[i], queens[j])
        | i,j in Domain where i<j ] );
```

4.1.8 Items

4.1.8.1 Include Items

```
<include-item> ::= "include" <string-literal>
```

```
include "foo.mzn";
```

```
foo.mzn
```

4.1.8.2 Variable Declaration Items

```
<var-decl-item> ::= <ti-expr-and-id> <annotations> [ "=" <expr> ]
                  | "any" ":" <ident> <annotations> [ "=" <expr> ]
```

```
int: A = 10;
```

```
int: A;
...
A = 10;
```

any

```
any: x = [1, 2, 3];
any: y = x[2];
```

xyany

4.1.8.3 Enum Items

```
<enum-item>  ::= "enum" <ident> <annotations> [ "=" <enum-cases-list> ]

<enum-cases-list> ::= <enum-cases>
                    | <enum-cases-list> "++" <enum-cases>

<enum-cases> ::= "{" <ident> "," ... "}"
               | "_" "(" <expr> ")"
               | <ident> "(" <ident> ")"
               | "anon_enum" "(" <expr> ")"
```

```
enum country = {Australia, Canada, China, England, USA};
```

Australia

_

```
enum Slot = _(1..n);
```

Slotenum Slot = anon_enum(n);

++

```
enum country_or_none = C(country) ++ {None};
```

country_or_nonecountryNonecountrycountry_or_nonecountrycountry_or_noneC(Canada)country_or_none

```
country_or_none: c_o_n = C(England);
country: c = C ¹(c_o_n);
```

C^-1(c_o_n)

```
enum Node = Left(1..n) ++ Right(1..n);
```

nn

```
enum Person = { Amy, Bert, Celeste, Doug, Emely };
enum Staff = S({Amy, Doug});
enum Customers = S(Person setminus S ¹(Staff));
```

```
enum Workers;
enum Shifts;
```

```
Workers = { welder, driller, stamper };
Shifts  = { idle, day, night };
```

```
enum Shifts;
Shifts: idle;            % Variable representing the idle constant.
```

```
enum Shifts = { idle_const, day, night };
idle = idle_const;      % Assignment to the variable.
```

idle_constidle

```
int: oz = Australia;  % oz = 1
```

T

```
% Return next greater enum value of x in enum type X
function T: enum_next(T: x);
```

```
function var T: enum_next(var T: x);

% Return next smaller enum value of x in enum type X
function T: enum_prev(T: x);
function var T: enum_prev(var T: x);

% Return enum base set for value x in enum type x
function set of T: enum_of(T: x);
function set of T: enum_of(var T: x);
function set of T: enum_of(set of T: x);
function set of T: enum_of(var set of T: x);

% Convert x to enum type X
function T: to_enum(set of T: X, int: x);
function var T: to_enum(set of T: X, var int: x);
```

4.1.8.4 Type-inst Synonym Items

```
<type-inst-syn-item> ::= "type" <ident> <annotations> "=" <ti-expr>
```

```
type MyInt     = int;
type Person    = record(string: name, int: height);
type Domain    = 1..n;
```

4.1.8.5 Assignment Items

```
<assign-item> ::= <ident> "=" <expr>
```

```
A = 10;
```

4.1.8.6 Constraint Items

```
<constraint-item> ::= "constraint" [ <string-annotation> ] <expr>
```

```
constraint a*x < b;
```

par boolvar bool

4.1.8.7 Solve Items

```
<solve-item> ::= "solve" <annotations> "satisfy"
               | "solve" <annotations> "minimize" <expr>
               | "solve" <annotations> "maximize" <expr>
```

```
solve satisfy;
solve maximize a*x + y - 3*z;
```

minimizemaximize

4.1.8.8 Output Items

```
<output-item> ::= "output" [ <string-annotation> ] <expr>
```

```
output ["The value of x is ", show(x), "!\n"];
```

array[int] of par string++showshow_intshow_float

--only-sections--not-sections

4.1.8.9 Annotation Items

ann

```
<annotation-item> ::= "annotation" <ident> <params>
```

```
annotation solver(int: kind);
```

4.1.8.10 User-defined Operations

```
<predicate-item> ::= "predicate" <operation-item-tail>

<test-item> ::= "test" <operation-item-tail>

<function-item> ::= "function" <ti-expr> ":" <operation-item-tail>

<operation-item-tail> ::= <ident> <params> <annotations> [ "=" <expr> ]

<params> ::= [ ( <ti-expr-and-id> "," ... ) ]
```

even

```
predicate even(var int: x) =
    x mod 2 = 0;
```

```
predicate alldifferent(array [int] of var int: xs);
```

Basic Properties

“” “” “”

par boolvar bool

Ad-hoc polymorphism

```
predicate p(1..5: x);
predicate p(1..5: x) = false;       % ok:     first definition
predicate p(1..5: x) = true;        % error:  repeated definition
```

pp(3)

```
predicate p(par int: x);
predicate p(var int: x);
```

```
predicate q(int:        x);
predicate q(set of int: x);
```

f

```
function int: f(int: x, float: y) = 0;
function int: f(float: x, int: y) = 1;
```

f(3,3)

p

g

```
function float: g(int: t1, float: t2) = t2;
function int  : g(float: t1, int: t2) = t1;
```

g(3,4)

```
float: s = g(3,4);
int: t = g(3,4);
```

st

$(S_1, ..., S_n)(T_1, ..., T_n)(R_1, ..., R_n)R_iS_iT_i$

$(S_1, ..., S_n)(T_1, ..., T_n)STS_i \preceq T_iiS \preceq T$

g(int,float)(float,int)(int,int)g(int,int)

```
function int: g(int: t1, int: t2) = t1;
```

gint

“” g(3,4)

Local Variables

have_common_divisor

```
predicate have_common_divisor(int: A, int: B) =
    let {
        var 2..min(A,B): C;
    } in
        A mod C = 0 /\
        B mod C = 0;
```

C

```
predicate have_common_divisor(int: A, int: B) =
    exists(C in 2..min(A,B))
        (A mod C = 0 /\ B mod C = 0);
```

4.1.9 Annotations

```
<annotations> ::= [ "::" <annotation> ]*

<annotation> ::= <expr-atom-head> <expr-atom-tail>

<string-annotation> ::= "::" <string-literal>
```

```
int: x::foo;
x = (3 + 4)::bar("a", 9)::baz("b");
solve :: blah(4)
    minimize x;
```

::

```
ann: '::'(var $T: e, ann: a);       % associative
ann: '::'(ann:    a, ann: b);       % associative + commutative
```

```
e :: a :: b
e :: b :: a
(e :: a) :: b
(e :: b) :: a
e :: (a :: b)
e :: (b :: a)
```

ann

```
constraint ::"first constraint" alldifferent(x);
constraint ::"second constraint" alldifferent(y);
constraint forall (i in 1..n) (my_constraint(x[i],y[i])::"constraint \(i)");
```

4.1.10 Partiality

4.1.10.1 Partial Assignments

```
1..5: x = 3;
```

```
int: x = 3;
constraint assert(x in 1..5,
                  "assignment to global parameter 'x' failed")
```

,

```
var 1..5: x = 3;
```

```
var int: x = 3;
constraint x in 1..5;
```

```
let { 1..5: x = 3; } in x+1
```

```
let { int: x = 3; } in
    assert(x in 1..5,
           "assignment to let parameter 'x' failed", x+1)
```

“” false

```
u = [ let { var 1..5: x = 6} in x, let { par 1..5: y = 6; } in y) ];
```

4.1.10.2 Partial Predicate/Function and Annotation Arguments

4.1.10.3 Partial Array Accesses

"" false

```
array[1..3] of int: a = [1,2,3];
var int: i;
constraint (a[i] + 3) > 10 \/ i = 99;
```

i ""

4.1.11 Built-in Operations

bool: '\/'(bool, bool)

TI: f(TI1,...,TIn)fTI,...,TInTI

4.1.11.1 Comparison Operations

><=>====!=

```
    bool: '<'(    $T,     $T)
var bool: '<'(var $T, var $T)
```

4.1.11.2 Arithmetic Operations

-*

```
    int:   '+'(    int,       int)
var int:   '+'(var int,   var int)
    float: '+'(    float,     float)
var float: '+'(var float, var float)
```

+

```
    int:   '-'(    int)
var int:   '-'(var int)
    float: '-'(    float)
var float: '-'(var float)
```

```
    int:   'div'(    int,       int)
var int:   'div'(var int,   var int)
    int:   'mod'(    int,       int)
var int:   'mod'(var int,   var int)
```

```
    float: '/'  (    float,     float)
var float: '/'  (var float, var float)
```

```
x = (x div y) * y + (x mod y)
```

```
 7 div  4 =  1        7 mod  4 =  3
-7 div  4 = -1       -7 mod  4 = -3
 7 div -4 = -1        7 mod -4 = 3
-7 div -4 =  1       -7 mod -4 = -3
```

product

```
    int:   sum(array[$T]  of     int  )
var int:   sum(array[$T]  of var int  )
    float: sum(array[$T]  of     float)
var float: sum(array[$T]  of var float)
```

max

```
$T:     min(    $T,     $T)
var $T: min(var $T, var $T)
```

max

```
$U:     min(array[$T]  of     $U)
var $U: min(array[$T]  of var $U)
```

max

```
$T:    min(set of $T)
```

```
    int:   abs(    int)
var int:   abs(var int)
    float: abs(    float)
var float: abs(var float)
```

```
    float: sqrt(    float)
var float: sqrt(var float)
```

pow(2, 5)32

```
  int: pow(int,       int)
float: pow(float,     float)
```

```
    float: exp(float)
var float: exp(var float)
```

log10log2

```
    float: ln(float)
var float: ln(var float)
```

```
float: log(float, float)
```

costanasinacosatansinhcoshtanhasinhacoshatanh

```
    float: sin(float)
var float: sin(var float)
```

4.1.11.3 Logical Operations

\/<--><->xornot

=><=<=><= “”

```
    bool: '/\'(    bool,     bool)
var bool: '/\'(var bool, var bool)
```

existsforalltrueexistsfalse

```
    bool: forall(array[$T]  of     bool)
var bool: forall(array[$T]  of var bool)
```

iffalltruefalse

```
    bool: xorall(array[$T]  of     bool: bs) = foldl('xor', false, bs)
var bool: xorall(array[$T]  of var bool: bs) = foldl('xor', false, bs)
```

4.1.11.4 Set Operations

```
    bool: 'in'(     $T,       set of $T )
var bool: 'in'(var $$E,   var set of $$E)
```

superset

```
    bool: 'subset'(    set of $T ,     set of $T )
var bool: 'subset'(var set of $$E, var set of $$E)
```

intersectdiffsymdiff

```
    set of  $T: 'union'(    set of  $T,     set of  $T )
var set of $$E: 'union'(var set of $$E, var set of $$E )
```

1..0l+1u-1enum_next(l)enum_prev(l)

```
% Return set {l, ..., u}
set of $$E: '..'($$E: l, $$E: u)
% Return set {l+1, ..., u}
set of $$E: '<..'($$E: l, $$E: u)
% Return set {l, ..., u-1}
set of $$E: '..<'($$E: l, $$E: u)
% Return set {l+1, ..., u-1}
set of $$E: '<..<'($$E: l, $$E: u)
```

```
% x = {l, ..., max(enum_of(l))}
set of X: x = l..;
% x = {min(enum_of(u)), ..., u}
set of X: x = ..u;
```

```
    int: card(    set of  $T)
var int: card(var set of $$E)
```

array_intersect

```
    set of  $U:    array_union(array[$T]  of     set of  $U)
var set of $$E:    array_union(array[$T]  of var set of $$E)
```

4.1.11.5 Array Operations

```
int: length(array[$T] of $U)
```

1..nn'++'

```
array[int] of $T: '++'(array[int] of $T, array[int] of $T)
```

1..nn

```
set of $T:  index_set     (array[$T]      of $V)
set of $T:  index_set_1of2(array[$T, $U]  of $V)
set of $U:  index_set_2of2(array[$T, $U]  of $V)
...
```

```
array[$T1] of $V: array1d(set of $T1, array[$U] of $V)
array[$T1,$T2] of $V:
    array2d(set of $T1, set of $T2, array[$U] of $V)
array[$T1,$T2,$T3] of $V:
    array3d(set of $T1, set of $T2, set of $T3, array[$U] of $V)
```

4.1.11.6 Coercion Operations

$+\infty -\infty$

```
int: ceil (float)
int: floor(float)
int: round(float)
```

```
    int:          bool2int(    bool)
var int:          bool2int(var bool)
    float:        int2float(    int)
var float:        int2float(var int)
array[int] of $T: set2array(set of $T)
```

xy

```
                    $T: 'default'(opt $T:x, $T: y);
                opt $T: 'default'(opt $T:x, opt $T: y);
                var $T: 'default'(var opt $T:x, var $T: y);
            var opt $T: 'default'(var opt $T:x, var opt $T: y);
array[$U] of        $T: 'default'(array[$U] of $T:x, array[$U] of $T: y);
array[$U] of opt    $T: 'default'(array[$U] of opt $T:x, array[$U] of opt $T: y);
array[$U] of var    $T: 'default'(array[$U] of var $T:x, array[$U] of var $T: y);
array[$U] of var opt $T: 'default'(array[$U] of var opt $T:x, array[$U] of var opt
↪$T: y);
```

4.1.11.7 String Operations

```
string: show($T)
```

```
string: show_int(int, var int);
```

```
string: show_float(int, int, var float)
```

'++'

```
string: '++'(string, string)
```

'++'

```
string: concat(array[$T] of string)
```

```
string: join(string, array[$T] of string)
```

4.1.11.8 Bound and Domain Operations

lbub

{}

```
int:   lb(var int)
float: lb(var float)
int:   ub(var int)
float: ub(var float)
```

```
set of int: lb(var set of int)
set of int: ub(var set of int)
```

```
int:        lb_array(array[$T] of var int)
float:      lb_array(array[$T] of var float)
set of int: lb_array(array[$T] of var set of int)
int:        ub_array(array[$T] of var int)
float:      ub_array(array[$T] of var float)
set of int: ub_array(array[$T] of var set of int)
```

```
set of int: dom(var int)
```

dom

```
set of int: dom_array(array[$T] of var int)
```

```
int: dom_size(var int)
```

dom_sizecard(dom(x))

,

,

4.1.11.9 Option Type Operations

⊤

```
opt $T:  '<>';
```

occursabsent

```
par bool: occurs(par opt $T);
var bool: occurs(var opt $T);
par bool: absent(par opt $T);
var bool: absent(var opt $T);
```

deopt

```
par $T: deopt{par opt $T);
var $T: deopt(var opt $T);
```

4.1.11.10 Other Operations

true

```
$T:   assert(bool, string, s$T)
par bool: assert(bool, string)
```

```
$T: abort(string)
```

```
bool: trace(string)
```

```
$T: trace(string, $T)
```

,

```
$T: fix(var $T)
```

false’

```
par bool: is_fixed(var $T)
```

4.1.12 Content-types

application/x-zinc-output

```
% Output
<output> ::= <no-solutions> [ <warnings> ] <free-text>
           | ( <solution> )* [ <complete> ] [ <warnings> ] <free-text>

% Solutions
<solution> ::= <solution-text> [ \n ] "----------" \n

% Unsatisfiable
<no-solutions> ::= "=====UNSATISFIABLE=====" \n

% Complete
<complete> ::= "==========" \n

% Messages
<warnings> ::= ( <message> )+

<message>  ::= ( <line> )+
<line>     ::= "%" [^\n]* \n
```

4.1.13 JSON support

truefalse

"set"

"e"

"c""e"

```
int: n;
enum Customers;
array[1..n,Customers] of int: distances;
array[1..n] of set of int: patterns;
```

```
{
  "n" : 2,
  "Customers" : [ {"e" : "Customer A"}, {"e" : "Customer B"}, {"e" : "Customer C"}␣
↪],
  "distances" : [ [1,2,3],
                  [4,5,6]],
  "patterns"  : [ {"set" : [1,3,5]}, {"set" : [2,4,6]} ]
}
```

nCustomersdistancespatterns

4.1.14 Full grammar

4.1.14.1 Items

```
% A MiniZinc model
<model> ::= [ <item> ";" ... ]

% Items
<item>  ::= <include-item>
          | <var-decl-item>
          | <enum-item>
          | <type-inst-syn-item>
          | <assign-item>
          | <constraint-item>
          | <solve-item>
          | <output-item>
          | <predicate-item>
          | <test-item>
          | <function-item>
          | <annotation-item>

<ti-expr-and-id> ::= <ti-expr> ":" <ident>

% Include items
<include-item> ::= "include" <string-literal>

% Variable declaration items
<var-decl-item> ::= <ti-expr-and-id> <annotations> [ "=" <expr> ]
                  | "any" ":" <ident> <annotations> [ "=" <expr> ]

% Enum items
<enum-item>  ::= "enum" <ident> <annotations> [ "=" <enum-cases-list> ]

<enum-cases-list> ::= <enum-cases>
                    | <enum-cases-list> "++" <enum-cases>

<enum-cases> ::= "{" <ident> "," ... "}"
               | "_" "(" <expr> ")"
               | <ident> "(" <ident> ")"
               | "anon_enum" "(" <expr> ")"

% Type-inst synonym items
<type-inst-syn-item> ::= "type" <ident> <annotations> "=" <ti-expr>

% Assign items
<assign-item> ::= <ident> "=" <expr>

% Constraint items
<constraint-item> ::= "constraint" [ <string-annotation> ] <expr>

% Solve item
<solve-item> ::= "solve" <annotations> "satisfy"
               | "solve" <annotations> "minimize" <expr>
               | "solve" <annotations> "maximize" <expr>

% Output items
<output-item> ::= "output" [ <string-annotation> ] <expr>
```

```
% Annotation items
<annotation-item> ::= "annotation" <ident> <params>

% Predicate, test and function items
<predicate-item> ::= "predicate" <operation-item-tail>

<test-item> ::= "test" <operation-item-tail>

<function-item> ::= "function" <ti-expr> ":" <operation-item-tail>

<operation-item-tail> ::= <ident> <params> <annotations> [ "=" <expr> ]

<params> ::= [ ( <ti-expr-and-id> "," ... ) ]
```

4.1.14.2 Type-Inst Expressions

```
<ti-expr> ::= <base-ti-expr>
            | <array-ti-expr>

<base-ti-expr> ::= <var-par> <opt-ti> <set-ti> <base-ti-expr-tail> ["++" <base-ti-
↪expr>]
                 | "any" <ti-variable-expr-tail>

<var-par> ::= "var" | "par" |

<opt-ti> ::= "opt" |

<set-ti> ::= "set" "of" |

<base-type> ::= "bool"
              | "int"
              | "float"
              | "string"

<base-ti-expr-tail> ::= <ident>
                      | <base-type>
                      | <ti-variable-expr-tail>
                      | <tuple-ti-expr-tail>
                      | <record-ti-expr-tail>
                      | "ann"
                      | "{" <expr> "," ... "}"
                      | <num-expr> ".." <num-expr>

% Type-inst variables
<ti-variable-expr-tail> ::= $[A-Za-z$][A-Za-z0-9_]*

% Array type-inst expressions
<array-ti-expr> ::= "array" [ <ti-expr> "," ... ] "of" <base-ti-expr>
                  | "list" "of" <base-ti-expr>

% Tuple type-inst expressions
<tuple-ti-expr-tail> ::= "tuple" ( <ti-expr> "," ... )

% Record type-inst expressions
<record-ti-expr-tail> ::= "record" ( <ti-expr-and-id> "," ... )
```

4.1.14.3 Expressions

```
<expr> ::= <expr-atom> <expr-binop-tail>

<expr-atom> ::= <expr-atom-head> <expr-atom-tail> <annotations>

<expr-binop-tail> ::= [ <bin-op> <expr> ]

<expr-atom-head> ::= <builtin-un-op> <expr-atom>
                   | "(" <expr> ")"
                   | <ident-or-quoted-op>
                   | "_"
                   | <bool-literal>
                   | <int-literal>
                   | <float-literal>
```

```
                    | <string-literal>
                    | <set-literal>
                    | <set-comp>
                    | <array-literal>
                    | <array-literal-2d>
                    | <indexed-array-literal>
                    | <indexed-array-literal-2d>
                    | <tuple-literal>
                    | <record-literal>
                    | <array-comp>
                    | <indexed-array-comp>
                    | <ann-literal>
                    | <if-then-else-expr>
                    | <let-expr>
                    | <call-expr>
                    | <gen-call-expr>

<expr-atom-tail> ::=
                    | <array-access-tail> <expr-atom-tail>

% Numeric expressions
<num-expr> ::= <num-expr-atom> <num-expr-binop-tail>

<num-expr-atom> ::= <num-expr-atom-head> <expr-atom-tail> <annotations>

<num-expr-binop-tail> ::= [ <num-bin-op> <num-expr> ]

<num-expr-atom-head> ::= <builtin-num-un-op> <num-expr-atom>
                       | "(" <num-expr> ")"
                       | <ident-or-quoted-op>
                       | <int-literal>
                       | <float-literal>
                       | <if-then-else-expr>
                       | <let-expr>
                       | <call-expr>
                       | <gen-call-expr>

% Built-in operators
<builtin-op> ::= <builtin-bin-op> | <builtin-un-op>

<bin-op> ::= <builtin-bin-op> |  `<ident> `

<builtin-bin-op> ::= "<->" | "->" | "<-" | "\/" | "xor" | "/\"
                    | "<" | ">" | "<=" | ">=" | "==" | "=" | "!="
                    | "~=" | "~!="
                    | "in" | "subset" | "superset" | "union" | "diff" | "symdiff"
                    | ".." | "intersect" | "++" | "default" | <builtin-num-bin-op>

<builtin-un-op> ::= "not" | <builtin-num-un-op>

% Built-in numeric operators
<num-bin-op> ::= <builtin-num-bin-op> |  `<ident> `

<builtin-num-bin-op> ::= "+" | "-" | "*" | "/" | "div" | "mod" | "^"
                       | "~+" | "~-" | "~*" | "~/" | "~div"

<builtin-num-un-op> ::= "+" | "-"
```

```
% Boolean literals
<bool-literal> ::= "false" | "true"

% Integer literals
<int-literal> ::= [0-9]+
                | 0x[0-9A-Fa-f]+
                | 0o[0-7]+

% Float literals
<float-literal> ::= [0-9]+"."[0-9]+
                  | [0-9]+"."[0-9]+[Ee][-+]?[0-9]+
                  | [0-9]+[Ee][-+]?[0-9]+
                  | 0[xX]([0-9a-fA-F]*"."[0-9a-fA-F]+|[0-9a-fA-F]+".")([pP][+-]?[0-
↪9]+)
                  | (0[xX][0-9a-fA-F]+[pP][+-]?[0-9]+)

% String literals
<string-contents> ::= ([^"\n\] |  \[0-7][0-7?][0-7]? | \x[0-9a-fA-F][0-9a-fA-F]? | \
↪n | \t | \" | \\)*

<string-literal> ::= """ <string-contents> """
                   | """ <string-contents> "\(" <string-interpolate-tail>

<string-interpolate-tail> ::= <expr> ")"<string-contents>"""
                            | <expr> ")"<string-contents>"\(" <string-interpolate-
↪tail>

% Set literals
<set-literal> ::= "{" [ <expr> "," ... ] "}"

% Set comprehensions
<set-comp> ::= "{" <expr> "|" <comp-tail> "}"

<comp-tail> ::= <generator> [ "where" <expr> ] "," ...

<generator> ::= ( <ident> | "_" ) "," ... "in" <expr>

% Array literals
<array-literal> ::= "[" [ <expr> "," ... ] "]"

% 2D Array literals
<array-literal-2d> ::= "[|" [ [ <expr> "," ... ] "|" ... ] "|]"

% Indexed 2D Array literals
<indexed-array-literal-2d> ::= "[|" [ <expr> ":" ... ] [ [ <expr> ":" ] <expr> "," .
↪.. ] "|]"

% Indexed array literals
<indexed-array-literal> ::= "[" [ ( <index-tuple> ":" <expr> ) "," ... ] "]"
                          | "[" [ <index-tuple> ":" <expr> "," <expr> "," ... ] "]"

<index-tuple> ::= <expr>
                | "(" [ <expr> "," ... ] ")"

% Tuple literals
<tuple-literal> ::= "(" <expr> "," [ <expr> "," ... ] ")"
```

```
% Record literals
<record-literal> ::= "(" <ident> ":" <expr> "," [ <ident> ":" <expr> "," ... ] ")"

% Array comprehensions
<array-comp> ::= "[" <expr> "|" <comp-tail> "]"

% Indexed array comprehensions
<indexed-array-comp> ::= "[" <index-tuple> ":" <expr> "|" <comp-tail> "]"

% Array access
<array-access-tail> ::= "[" <expr> "," ... "]"

% Annotation literals
<ann-literal> ::= <ident> [ "(" <expr> "," ... ")" ]

% If-then-else expressions
<if-then-else-expr> ::= "if" <expr> "then" <expr> [ "elseif" <expr> "then" <expr>␣
↪]* "else" <expr> "endif"

% Call expressions
<call-expr> ::= <ident-or-quoted-op> [ "(" <expr> "," ... ")" ]

% Let expressions
<let-expr> ::= "let" "{" <let-item> ";" ... "}" "in" <expr>

<let-item> ::= <var-decl-item>
             | <constraint-item>

% Generator call expressions
<gen-call-expr> ::= <ident-or-quoted-op> "(" <comp-tail> ")" "(" <expr> ")"
```

4.1.14.4 Miscellaneous Elements

```
% Identifiers
<ident> ::= _?[A-Za-z][A-Za-z0-9_]* | ' [^' \xa\xd\x0]+'

% Identifiers and quoted operators
<ident-or-quoted-op> ::= <ident>
                       | ' <builtin-op>'

% Annotations
<annotations> ::= [ "::" <annotation> ]*

<annotation> ::= <expr-atom-head> <expr-atom-tail>

<string-annotation> ::= "::" <string-literal>
```

4.2.1 Standard Library

4.2.1.1 Built-in functions and operators

Comparison Builtins

```
1.  test '!='($T: x, $T: y)
2.  test '!='(opt $T: x, opt $T: y)
3.  predicate '!='(any $T: x, any $T: y)
4.  test '!='(array [$U] of $T: x, array [$U] of $T: y)
5.  test '!='(array [$U] of opt $T: x, array [$U] of opt $T: y)
6.  predicate '!='(array [$U] of var $T: x, array [$U] of var $T: y)
7.  predicate '!='(array [$U] of var opt $T: x,
                   array [$U] of var opt $T: y)
```

xy

x != y

xy

x != y

```
test '<'($T: x, $T: y)
predicate '<'(var $T: x, var $T: y)
predicate '<'(var opt int: x, var opt int: y)
test '<'(opt int: x, opt int: y)
predicate '<'(var opt float: x, var opt float: y)
test '<'(opt float: x, opt float: y)
test '<'(array [$U] of $T: x, array [$U] of $T: y)
predicate '<'(array [$U] of var int: x, array [$U] of var int: y)
predicate '<'(array [$U] of var bool: x, array [$U] of var bool: y)
predicate '<'(array [$U] of var float: x, array [$U] of var float: y)
predicate '<'(array [$U] of var set of int: x,
              array [$U] of var set of int: y)
```

xy

x < y

xyxy

x < y

xy

x < y

```
test '<='($T: x, $T: y)
predicate '<='(var $T: x, var $T: y)
predicate '<='(var opt int: x, var opt int: y)
test '<='(opt int: x, opt int: y)
predicate '<='(var opt float: x, var opt float: y)
test '<='(opt float: x, opt float: y)
test '<='(array [$U] of $T: x, array [$U] of $T: y)
predicate '<='(array [$U] of var int: x, array [$U] of var int: y)
```

```
9.  predicate '<='(array [$U] of var bool: x, array [$U] of var bool: y)

10. predicate '<='(array [$U] of var float: x, array [$U] of var float: y)

11. predicate '<='(array [$U] of var set of int: x,
                   array [$U] of var set of int: y)
```

xy

x <= y

xyxy

x <= y

xy

x <= y

```
1.  test '='($T: x, $T: y)

2.  test '='(opt $T: x, opt $T: y)

3.  predicate '='(any $T: x, any $T: y)

4.  test '='(array [$U] of $T: x, array [$U] of $T: y)

5.  test '='(array [$U] of opt $T: x, array [$U] of opt $T: y)

6.  predicate '='(array [$U] of var $T: x, array [$U] of var $T: y)

7.  predicate '='(array [$U] of var opt $T: x, array [$U] of var opt $T: y)
```

xy

x = y

xy

x = y

```
1.  test '>'($T: x, $T: y)

2.  predicate '>'(var $T: x, var $T: y)

3.  predicate '>'(var opt int: x, var opt int: y)

4.  test '>'(opt int: x, opt int: y)
```

```
5.  predicate '>'(var opt float: x, var opt float: y)
6.  test '>'(opt float: x, opt float: y)
7.  test '>'(array [$U] of $T: x, array [$U] of $T: y)
8.  predicate '>'(array [$U] of var int: x, array [$U] of var int: y)
9.  predicate '>'(array [$U] of var bool: x, array [$U] of var bool: y)
10. predicate '>'(array [$U] of var float: x, array [$U] of var float: y)
11. predicate '>'(array [$U] of var set of int: x,
                  array [$U] of var set of int: y)
```

xy

x > y

xyxy

x > y

xy

x > y

```
1.  test '>='($T: x, $T: y)
2.  predicate '>='(var $T: x, var $T: y)
3.  predicate '>='(var opt int: x, var opt int: y)
4.  test '>='(opt int: x, opt int: y)
5.  predicate '>='(var opt float: x, var opt float: y)
6.  test '>='(opt float: x, opt float: y)
7.  test '>='(array [$U] of $T: x, array [$U] of $T: y)
8.  predicate '>='(array [$U] of var int: x, array [$U] of var int: y)
9.  predicate '>='(array [$U] of var bool: x, array [$U] of var bool: y)
10. predicate '>='(array [$U] of var float: x, array [$U] of var float: y)
11. predicate '>='(array [$U] of var set of int: x,
                   array [$U] of var set of int: y)
```

xy

x >= y

xyxy

x >= y

xy

x >= y

```
predicate '~!='(var opt bool: x, var opt bool: y)
predicate '~!='(var opt int: x, var opt int: y)
predicate '~!='(var opt float: x, var opt float: y)
```

xy

```
predicate '~='(var opt bool: x, var opt bool: y)
predicate '~='(var opt int: x, var opt int: y)
predicate '~='(var opt float: x, var opt float: y)
```

xy

Arithmetic Builtins

```
1. function int: '*'(int: x, int: y)
2. function var int: '*'(var int: x, var int: y)
3. function var int: '*'(var opt int: x, var opt int: y)
4. function int: '*'(opt int: x, opt int: y)
5. function var float: '*'(var opt float: x, var opt float: y)
6. function float: '*'(opt float: x, opt float: y)
7. function float: '*'(float: x, float: y)
8. function var float: '*'(var float: x, var float: y)
```

xy

x * y

xy

x * y

xy

x * y

```
1.  function int: '+'(int: x, int: y)
2.  function var int: '+'(var int: x, var int: y)
3.  function float: '+'(float: x, float: y)
4.  function var float: '+'(var float: x, var float: y)
5.  function var int: '+'(var opt int: x, var opt int: y)
6.  function int: '+'(opt int: x, opt int: y)
7.  function var float: '+'(var opt float: x, var opt float: y)
8.  function float: '+'(opt float: x, opt float: y)
```

xy

x + y

xy

x + y

```
1.  function int: '-'(int: x, int: y)
2.  function var int: '-'(var int: x, var int: y)
3.  function float: '-'(float: x, float: y)
4.  function var float: '-'(var float: x, var float: y)
5.  function var opt int: '-'(var opt int: x, var opt int: y)
6.  function var opt float: '-'(var opt float: x, var opt float: y)
7.  function opt int: '-'(opt int: x, opt int: y)
8.  function opt float: '-'(opt float: x, opt float: y)
9.  function int: '-'(int: x)
10. function var int: '-'(var int: x)
11. function float: '-'(float: x)
12. function var float: '-'(var float: x)
```

```
13.  function opt int: '-'(opt int: x)
14.  function var opt int: '-'(var opt int: x)
15.  function opt float: '-'(opt float: x)
16.  function var opt float: '-'(var opt float: x)
```

```
xy
x - y

xxyxy
x - y

x
- x

x
- x
```

```
1.  function float: '/'(float: x, float: y)
2.  function var float: '/'(var float: x, var float: y)
3.  function var opt float: '/'(var opt float: x, var opt float: y)
4.  function opt float: '/'(opt float: x, opt float: y)
```

```
xy
x / y

xxyxy
x / y
```

```
function int: '^'(int: x, int: y)
function var int: '^'(var int: x, var int: y)
function float: '^'(float: x, float: y)
function var float: '^'(var float: x, var float: y)
```

```
x ^ y
```

```
1.  function int: 'div'(int: x, int: y)
2.  function var int: 'div'(var int: x, var int: y)
3.  function var opt int: 'div'(var opt int: x, var opt int: y)
4.  function opt int: 'div'(opt int: x, opt int: y)
```

```
xy
x div y

xxyxy
x div y
```

```
1.  function int: 'mod'(int: x, int: y)
2.  function var int: 'mod'(var int: x, var int: y)
3.  function var opt int: 'mod'(var opt int: x, var opt int: y)
4.  function opt int: 'mod'(opt int: x, opt int: y)
```

```
xy
x mod y

xyxy
x mod y
```

```
1.  function int: abs(int: x)
2.  function var int: abs(var int: x)
3.  function opt int: abs(opt int: x)
4.  function var opt int: abs(var opt int: x)
5.  function float: abs(float: x)
6.  function var float: abs(var float: x)
7.  function opt float: abs(opt float: x)
```

```
8.  function var opt float: abs(var opt float: x)
```

x

x

x

x

```
function $$E: arg_max(array [$$E] of bool: x)
function $$E: arg_max(array [$$E] of int: x)
function $$E: arg_max(array [$$E] of float: x)
```

x

```
function $$E: arg_min(array [$$E] of bool: x)
function $$E: arg_min(array [$$E] of int: x)
function $$E: arg_min(array [$$E] of float: x)
```

x

```
function int: count(array [$T] of bool: x)
function var int: count(array [$T] of var bool: x)
function var int: count(array [$T] of var opt bool: x)
```

x

```
1.  function $T: max($T: x, $T: y)

2.  function var int: max(var int: x, var int: y)

3.  function var float: max(var float: x, var float: y)

4.  function $T: max(array [$U] of $T: x)

5.  function var int: max(array [$U] of var int: x)

6.  function var float: max(array [$U] of var float: x)

7.  function var int: max(array [$X] of var opt int: x)

8.  function var float: max(array [$X] of var opt float: x)
```

```
9.  function $$E: max(set of $$E: x)
```

xy

x

x

x

```
function var opt int: max_weak(array [$X] of var opt int: x)
function var opt float: max_weak(array [$X] of var opt float: x)
```

x

```
1.  function $T: min($T: x, $T: y)
2.  function var int: min(var int: x, var int: y)
3.  function var float: min(var float: x, var float: y)
4.  function $T: min(array [$U] of $T: x)
5.  function var int: min(array [$U] of var int: x)
6.  function var float: min(array [$U] of var float: x)
7.  function var int: min(array [$X] of var opt int: x)
8.  function var float: min(array [$X] of var opt float: x)
9.  function $$E: min(set of $$E: x)
```

xy

x

x

x

```
function var opt int: min_weak(array [$X] of var opt int: x)
function var opt float: min_weak(array [$X] of var opt float: x)
```

x

```
1.  function int: pow(int: x, int: y)
2.  function var int: pow(var int: x, int: y)
3.  function var int: pow(var int: x, var int: y)
4.  function float: pow(float: x, float: y)
5.  function var float: pow(var float: x, var float: y)
```

< 01 div pow(x, abs(y))

```
1.  function int: product(array [$T] of int: x)
2.  function var int: product(array [$T] of var int: x)
3.  function float: product(array [$T] of float: x)
4.  function var float: product(array [$T] of var float: x)
5.  function int: product(array [$T] of opt int: x)
6.  function var int: product(array [$T] of var opt int: x)
7.  function float: product(array [$T] of opt float: x)
8.  function var float: product(array [$T] of var opt float: x)
```

x

x

```
1.  function float: sqrt(float: x)
2.  function var float: sqrt(var float: x)
3.  function opt float: sqrt(opt float: x)
4.  function var opt float: sqrt(var opt float: x)
```

$\sqrt{}$

$\sqrt{x}$

```
1.  function int: sum(array [$T] of int: x)
2.  function var int: sum(array [$T] of var int: x)
3.  function float: sum(array [$T] of float: x)
4.  function var float: sum(array [$T] of var float: x)
5.  function var int: sum(array [int] of var opt int: x)
6.  function int: sum(array [int] of opt int: x)
7.  function var float: sum(array [int] of var opt float: x)
8.  function float: sum(array [int] of opt float: x)
```

x

x

```
function var opt int: '~*'(var opt int: x, var opt int: y)
function opt int: '~*'(opt int: x, opt int: y)
function var opt float: '~*'(var opt float: x, var opt float: y)
```

xy

```
function var opt int: '~+'(var opt int: x, var opt int: y)
function opt int: '~+'(opt int: x, opt int: y)
function var opt float: '~+'(var opt float: x, var opt float: y)
```

xy

```
function var opt int: '~-'(var opt int: x, var opt int: y)
function opt int: '~-'(opt int: x, opt int: y)
function var opt float: '~-'(var opt float: x, var opt float: y)
```

xy

```
function var opt float: '~/'(var opt float: x, var opt float: y)
function opt float: '~/'(opt float: x, opt float: y)
```

xyxy

```
function var opt int: '~div'(var opt int: x, var opt int: y)
function opt int: '~div'(opt int: x, opt int: y)
```

xyxy

Exponential and logarithmic builtins

```
1. function float: exp(float: x)
2. function var float: exp(var float: x)
3. function opt float: exp(opt float: x)
4. function var opt float: exp(var opt float: x)
```

e

ex

```
1. function float: ln(float: x)
2. function var float: ln(var float: x)
3. function opt float: ln(opt float: x)
4. function var opt float: ln(var opt float: x)
```

x

```
function float: log(float: x, float: y)
```

```
1. function float: log10(float: x)
2. function var float: log10(var float: x)
```

```
3.  function opt float: log10(opt float: x)
4.  function var opt float: log10(var opt float: x)
```

10

10 x

```
1.  function float: log2(float: x)
2.  function var float: log2(var float: x)
3.  function opt float: log2(opt float: x)
4.  function var opt float: log2(var opt float: x)
```

2

2 x

Trigonometric functions

```
1.  function float: acos(float: x)
2.  function var float: acos(var float: x)
3.  function opt float: acos(opt float: x)
4.  function var opt float: acos(var opt float: x)
```

x

```
1.  function float: acosh(float: x)
2.  function var float: acosh(var float: x)
3.  function opt float: acosh(opt float: x)
4.  function var opt float: acosh(var opt float: x)
```

x

```
1. function float: asin(float: x)
2. function var float: asin(var float: x)
3. function opt float: asin(opt float: x)
4. function var opt float: asin(var opt float: x)
```

x

```
1. function float: asinh(float: x)
2. function var float: asinh(var float: x)
3. function opt float: asinh(opt float: x)
4. function var opt float: asinh(var opt float: x)
```

x

```
1. function float: atan(float: x)
2. function var float: atan(var float: x)
3. function opt float: atan(opt float: x)
4. function var opt float: atan(var opt float: x)
```

x

```
1. function float: atanh(float: x)
2. function var float: atanh(var float: x)
```

```
3.  function opt float: atanh(opt float: x)
4.  function var opt float: atanh(var opt float: x)
```

x

```
1.  function float: cos(float: x)
2.  function var float: cos(var float: x)
3.  function opt float: cos(opt float: x)
4.  function var opt float: cos(var opt float: x)
```

x

```
1.  function float: cosh(float: x)
2.  function var float: cosh(var float: x)
3.  function opt float: cosh(opt float: x)
4.  function var opt float: cosh(var opt float: x)
```

x

```
1.  function float: sin(float: x)
2.  function var float: sin(var float: x)
3.  function opt float: sin(opt float: x)
4.  function var opt float: sin(var opt float: x)
```

x

```
1.  function float: sinh(float: x)
2.  function var float: sinh(var float: x)
3.  function opt float: sinh(opt float: x)
4.  function var opt float: sinh(var opt float: x)
```

x

```
1.  function float: tan(float: x)
2.  function var float: tan(var float: x)
3.  function opt float: tan(opt float: x)
4.  function var opt float: tan(var opt float: x)
```

x

```
1.  function float: tanh(float: x)
2.  function var float: tanh(var float: x)
3.  function opt float: tanh(opt float: x)
4.  function var opt float: tanh(var opt float: x)
```

x

Coercions

```
1.  function float: bool2float(bool: b)
2.  function opt float: bool2float(opt bool: b)
3.  function array [$T] of float: bool2float(array [$T] of bool: x)
4.  function array [$T] of var float: bool2float(array [$T] of var bool: x)
5.  function var float: bool2float(var bool: b)
6.  function var opt float: bool2float(var opt bool: x)
7.  function array [$T] of opt float: bool2float(array [$T] of opt bool: x)
8.  function array [$T] of var opt float: bool2float(array [$T] of var opt bool: x)
```

b

x
b
xx

x

```
1.  function int: bool2int(bool: b)
2.  function opt int: bool2int(opt bool: b)
3.  function var opt int: bool2int(var opt bool: x)
4.  function var int: bool2int(var bool: b)
5.  function array [$T] of var int: bool2int(array [$T] of var bool: b)
6.  function array [$T] of int: bool2int(array [$T] of bool: x)
7.  function array [$T] of set of int: bool2int(array [$T] of set of bool: x)
8.  function array [$T] of var int: bool2int(array [$T] of var bool: x)
9.  function array [$T] of opt int: bool2int(array [$T] of opt bool: x)
10.  function array [$T] of var opt int: bool2int(array [$T] of var opt bool: x)
```

b

xx

b

b

x

x

x

```
function int: ceil(float: x)
function var int: ceil(var float: x)
```

⌈⌉

```
function int: floor(float: x)
function var int: floor(var float: x)
```

⌊⌋

```
1.  function float: int2float(int: x)
2.  function opt float: int2float(opt int: x)
3.  function var float: int2float(var int: x)
4.  function var opt float: int2float(var opt int: x)
5.  function array [$T] of float: int2float(array [$T] of int: x)
6.  function array [$T] of opt float: int2float(array [$T] of opt int: x)
7.  function array [$T] of var float: int2float(array [$T] of var int: x)
8.  function array [$T] of var opt float: int2float(array [$T] of var opt int: x)
```

x

xx

x

x

```
function int: round(float: x)
function var int: round(var float: x)
```

x

```
function array [int] of $$E: set2array(set of $$E: x)
```

x

Array operations

Functions and Predicates

```
function array [int] of any $T: '++'(array [int] of any $T: x,
                                     array [int] of any $T: y)
```

xy

x ++ y

```
test 'in'($X: x, array [$T] of $X: y)
test 'in'(opt $X: x, array [$T] of opt $X: y)
predicate 'in'(var $X: x, array [$T] of var $X: y)
predicate 'in'(var opt $X: x, array [$T] of var opt $X: y)
test 'in'(set of $X: x, array [$T] of set of $X: y)
predicate 'in'(var set of $X: x, array [$T] of var set of $X: y)
```

yx

x in y

1. ```
   function array [int] of any $V: array1d(array [$U] of any $V: x)
   ```

2. ```
   function array [$$E] of any $V: array1d(set of $$E: S,
                                           array [$U] of any $V: x)
   ```

   xxx

   xSx

```
function array [$$E,$$F] of any $V: array2d(set of $$E: S1,
                                           set of $$F: S2,
                                           array [$U] of any $V: x)
```

xS1S2x

```
function var set of $$T: array2set(array [int] of var $$T: x)
function set of $$T: array2set(array [int] of $$T: x)
function set of $$T: array2set(array [int] of opt $$T: x)
function set of bool: array2set(array [int] of bool: x)
function set of float: array2set(array [int] of float: x)
```

x

```
function array [$$E,$$F,$$G] of any $V: array3d(set of $$E: S1,
                                                set of $$F: S2,
                                                set of $$G: S3,
                                                array [$U] of any $V: x)
```

xS1S2S3x

```
function array [$$E,$$F,$$G,$$H] of any $V: array4d(set of $$E: S1,
                                                    set of $$F: S2,
                                                    set of $$G: S3,
                                                    set of $$H: S4,
                                                    array [$U] of any $V: x)
```

xS1S2S3S4x

```
function array [$$E,$$F,$$G,$$H,$$I] of any $V: array5d(set of $$E: S1,
                                                        set of $$F: S2,
                                                        set of $$G: S3,
                                                        set of $$H: S4,
                                                        set of $$I: S5,
                                                        array [$U] of any $V: x)
```

xS1S2S3S4S5x

```
function array [$$E,$$F,$$G,$$H,$$I,$$J] of any $V: array6d(set of $$E: S1,
                                                            set of $$F: S2,
                                                            set of $$G: S3,
                                                            set of $$H: S4,
                                                            set of $$I: S5,
                                                            set of $$J: S6,
                                                            array [$U] of any $V: x)
```

xS1S2S3S4S5S6x

```
function array [$T] of any $V: arrayXd(array [$T] of any $X: x,
                                       array [$U] of any $V: y)
```

yxy

```
function array [$$E] of any $T: col(array [$$E,int] of any $T: x,
                                    int: c)
```

cx

```
test has_element($T: e, array [int] of $T: x)
test has_element($T: e, array [int] of opt $T: x)
predicate has_element($T: e, array [$$E] of any $T: x)
```

ex

```
test has_index(int: i, array [int] of any $T: x)
```

ix

```
function set of $$E: index_set(array [$$E] of any $U: x)
```

x

```
function set of $$E: index_set_1of2(array [$$E,int] of any $U: x)
```

x

```
function set of $$E: index_set_1of3(array [$$E,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_1of4(array [$$E,int,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_1of5(array [$$E,int,int,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_1of6(array [$$E,int,int,int,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_2of2(array [int,$$E] of any $U: x)
```

x

```
function set of $$E: index_set_2of3(array [int,$$E,int] of any $U: x)
```

x

```
function set of $$E: index_set_2of4(array [int,$$E,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_2of5(array [int,$$E,int,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_2of6(array [int,$$E,int,int,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_3of3(array [int,int,$$E] of any $U: x)
```

x

```
function set of $$E: index_set_3of4(array [int,int,$$E,int] of any $U: x)
```

x

```
function set of $$E: index_set_3of5(array [int,int,$$E,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_3of6(array [int,int,$$E,int,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_4of4(array [int,int,int,$$E] of any $U: x)
```

x

```
function set of $$E: index_set_4of5(array [int,int,int,$$E,int] of any $U: x)
```

x

```
function set of $$E: index_set_4of6(array [int,int,int,$$E,int,int] of any $U: x)
```

x

```
function set of $$E: index_set_5of5(array [int,int,int,int,$$E] of any $U: x)
```

x

```
function set of $$E: index_set_5of6(array [int,int,int,int,$$E,int] of any $U: x)
```

x

```
function set of $$E: index_set_6of6(array [int,int,int,int,int,$$E] of any $U: x)
```

x

```
test index_sets_agree(array [$T] of any $U: x, array [$T] of any $W: y)
```

xy

```
function int: length(array [$T] of any $U: x)
```

x

```
function array [$$E] of $T: reverse(array [$$E] of $T: x)
function array [$$E] of opt $T: reverse(array [$$E] of opt $T: x)
function array [$$E] of any $T: reverse(array [$$E] of any $T: x)
```

x

x

```
function array [$$E] of any $T: row(array [int,$$E] of any $T: x,
                                    int: r)
```

rx

```
function array [$$F] of any $T: slice_1d(array [$E] of any $T: x,
                                         array [int] of set of int: s,
                                         set of $$F: dims1)
```

xsdims1

```
function array [$$F,$$G] of any $T: slice_2d(array [$E] of any $T: x,
                                             array [int] of set of int: s,
                                             set of $$F: dims1,
                                             set of $$G: dims2)
```

xsdims1dims2

```
function array [$$F,$$G,$$H] of any $T: slice_3d(array [$E] of any $T: x,
                                                 array [int] of set of int: s,
                                                 set of $$F: dims1,
                                                 set of $$G: dims2,
                                                 set of $$H: dims3)
```

xsdims1dims2dims3

```
function array [$$F,$$G,$$H,$$I] of any $T: slice_4d(array [$E] of any $T: x,
                                                     array [int] of set of int: s,
                                                     set of $$F: dims1,
                                                     set of $$G: dims2,
                                                     set of $$H: dims3,
                                                     set of $$I: dims4)
```

xsdims1dims2dims3dims4

```
function array [$$F,$$G,$$H,$$I,$$J] of any $T: slice_5d(array [$E] of any $T: x,
                                                         array [int] of set of int:␣
↪s,
                                                         set of $$F: dims1,
                                                         set of $$G: dims2,
                                                         set of $$H: dims3,
                                                         set of $$I: dims4,
                                                         set of $$J: dims5)
```

xsdims1dims2dims3dims4dims5

```
function array [$$F,$$G,$$H,$$I,$$J,$$K] of any $T: slice_6d(array [$E] of any $T:␣
↪x,
                                                             array [int] of set of␣
↪int: s,
                                                             set of $$F: dims1,
                                                             set of $$G: dims2,
                                                             set of $$H: dims3,
                                                             set of $$I: dims4,
                                                             set of $$J: dims5,
                                                             set of $$K: dims6)
```

xsdims1dims2dims3dims4dims5dims6

Annotations

```
annotation array_check_form
```

Logical operations

```
test '->'(bool: x, bool: y)
test '->'(opt bool: x, opt bool: y)
predicate '->'(var bool: x, var bool: y)
predicate '->'(var opt bool: x, var opt bool: y)
```

xy

x -> y

```
1.  test '/\'(bool: x, bool: y)

2.  test '/\'(opt bool: x, opt bool: y)

3.  predicate '/\'(var bool: x, var bool: y)

4.  predicate '/\'(var opt bool: x, var opt bool: y)
```

xy

x /\ y

xy

x /\ y

xy

x /\ y

xy

x /\ y

```
test '<-'(bool: x, bool: y)
test '<-'(opt bool: x, opt bool: y)
predicate '<-'(var bool: x, var bool: y)
predicate '<-'(var opt bool: x, var opt bool: y)
```

yx

x <- y

```
test '<->'(bool: x, bool: y)
test '<->'(opt bool: x, opt bool: y)
predicate '<->'(var bool: x, var bool: y)
predicate '<->'(var opt bool: x, var opt bool: y)
```

xy

```
x <-> y
```

```
1. test '\/'(bool: x, bool: y)
2. test '\/'(opt bool: x, opt bool: y)
3. predicate '\/'(var bool: x, var bool: y)
4. predicate '\/'(var opt bool: x, var opt bool: y)
```

```
xy
x \/ y
xy
x \/ y
xy
x \/ y
xy
x \/ y
```

```
1. test 'not'(bool: x)
2. function opt bool: 'not'(opt bool: x)
3. predicate 'not'(var bool: x)
4. function var opt bool: 'not'(var opt bool: x)
```

```
x
not x
x
not x
x
not x
x
not x
```

```
test 'xor'(bool: x, bool: y)
test 'xor'(opt bool: x, opt bool: y)
predicate 'xor'(var bool: x, var bool: y)
predicate 'xor'(var opt bool: x, var opt bool: y)
```

xy

x xor y

```
predicate bool_not(var bool: b)
```

b

```
test clause(array [$T] of bool: x, array [$T] of bool: y)
test clause(array [$T] of opt bool: x, array [$T] of opt bool: y)
predicate clause(array [$T] of var bool: x, array [$T] of var bool: y)
predicate clause(array [$T] of var opt bool: x,
                 array [$T] of var opt bool: y)
```

$(\bigvee_i [i]) \vee (\bigvee_j \neg [j])$

```
1.  test exists(array [$T] of bool: x)

2.  predicate exists(array [$T] of var bool: x)

3.  predicate exists(array [int] of var opt bool: x)
```

$\bigvee_i [i]$

ix

```
1.  test forall(array [$T] of bool: x)

2.  predicate forall(array [$T] of var bool: x)

3.  predicate forall(array [int] of var opt bool: x)
```

$\bigwedge_i [i]$

ix

```
test iffall(array [$T] of bool: x)
test iffall(array [$T] of opt bool: x)
predicate iffall(array [$T] of var bool: x)
predicate iffall(array [$T] of var opt bool: x)
```

$\oplus (\oplus_i [i])$

```
test xorall(array [$T] of bool: x)
test xorall(array [$T] of opt bool: x)
predicate xorall(array [$T] of var bool: x)
predicate xorall(array [$T] of var opt bool: x)
```

$\oplus_i[i]$

Set operations

```
function set of $$E: '..'($$E: a, $$E: b)
function set of float: '..'(float: a, float: b)
function set of bool: '..'(bool: a, bool: b)
```

$\{,\ldots,\}$

a .. b

```
function set of $$E: '..<'($$E: a, $$E: b)
```

$\{,\ldots,-1\}$

a ..< b

```
function set of $$E: '<..'($$E: a, $$E: b)
```

$\{+1,\ldots,\}$

a <.. b

```
function set of $$E: '<..<'($$E: a, $$E: b)
```

$\{+1,\ldots,-1\}$

a <..< b

```
function set of $T: 'diff'(set of $T: x, set of $T: y)
function var set of $$T: 'diff'(var set of $$T: x, var set of $$T: y)
```

$\setminus$

x diff y

```
1.  test 'in'(int: x, set of int: y)
2.  predicate 'in'(var int: x, var set of int: y)
3.  predicate 'in'(var opt int: x, set of int: y)
4.  test 'in'(opt int: x, set of int: y)
5.  predicate 'in'(var opt int: x, var set of int: y)
6.  test 'in'(float: x, set of float: y)
7.  predicate 'in'(var float: x, set of float: y)
8.  test 'in'(opt float: x, set of float: y)
9.  predicate 'in'(var opt float: x, set of float: y)
```

```
xy
x in y
xy
x in y
xyx
x in y
xyx
x in y
xyx
x in y

xy
x in y
xyx
x in y
xyx
x in y
```

```
function set of $T: 'intersect'(set of $T: x, set of $T: y)
function var set of $$T: 'intersect'(var set of $$T: x,
                                     var set of $$T: y)
```

```
xy
x intersect y
```

```
1.  test 'subset'(set of $T: x, set of $T: y)

2.  predicate 'subset'(var set of int: x, var set of int: y)
```

xy

x subset y

xy

x subset y

```
1.  test 'superset'(set of $T: x, set of $T: y)

2.  predicate 'superset'(var set of int: x, var set of int: y)
```

xy

x superset y

xy

x superset y

```
function set of $T: 'symdiff'(set of $T: x, set of $T: y)
function var set of $$T: 'symdiff'(var set of $$T: x,
                                   var set of $$T: y)
```

xy

x symdiff y

```
function set of $T: 'union'(set of $T: x, set of $T: y)
function var set of $$T: 'union'(var set of $$T: x, var set of $$T: y)
```

xy

x union y

```
function set of $$E: '..<o'($$E: a)
```

$\{, \ldots, -1\}$

```
function set of $$E: '..o'($$E: a)
```

$\{, \ldots, -1\}$

```
function set of $$E: '<..<o'($$E: a)
```

$\{+1, \ldots, -1\}$

```
function set of $$E: '<..o'($$E: a)
```

$\{+1, \ldots, \}$

```
1.  function set of $U: array_intersect(array [$T] of set of $U: x)

2.  function var set of $$E: array_intersect(array [$T] of var set of $$E: x)
```

x

xx

```
function set of $U: array_union(array [$T] of set of $U: x)
function var set of $$E: array_union(array [$T] of var set of $$E: x)
```

x

```
function int: card(set of $T: x)
function var int: card(var set of int: x)
```

x

```
function var $$E: max(var set of $$E: s)
```

s

```
function var $$E: min(var set of $$E: s)
```

s

```
function set of $$E: 'o..'($$E: a)
```

$\{, \ldots, -1\}$

```
function set of $$E: 'o..<'($$E: a)
```

$\{, \ldots, -1\}$

```
function set of $$E: 'o<..'($$E: a)
```

$\{+1, \ldots, \}$

```
function set of $$E: 'o<..<'($$E: a)
```

$\{+1, \ldots, -1\}$

```
function array [int] of int: set_to_ranges(set of int: S)
function array [int] of float: set_to_ranges(set of float: S)
function array [int] of bool: set_to_ranges(set of bool: S)
```

S

String operations

```
function string: '++'(string: s1, string: s2)
```

s1s2

s1 ++ s2

```
function string: concat(array [$T] of string: s)
```

s

```
function string: file_path()
```

```
1.  function string: format(any $T: x)
2.  function string: format(array [$U] of any $T: x)
3.  function string: format(int: w, int: p, any $T: x)
4.  function string: format(int: w, int: p, array [$U] of any $T: x)
5.  function string: format(int: w, any $T: x)
6.  function string: format(int: w, array [$U] of any $T: x)
```

x

xww
xpp

xww
xpp

xww

```
1.  function string: format_justify_string(int: w, string: x)
2.  function string: format_justify_string(int: w, int: p, string: x)
```

xww

xww
xpp

```
function string: join(string: d, array [$T] of string: s)
```

sd

```
function string: json_array(array [int] of string: arr)
```

arr

```
function string: json_object(array [int,1..2] of string: obj)
```

obj

obji1iobji2

```
1.  function array [int] of string: outputJSON()

2.  function array [int] of string: outputJSON(bool: b)
```

b

```
function array [int] of string: outputJSONParameters()
```

```
test output_to_json_section(string: s, any $T: o)
test output_to_json_section(string: s, array [$U] of any $T: o)
```

os

```
test output_to_section(string: s, string: o)
```

os

```
function string: show(any $T: x)
function string: show(array [$U] of any $T: x)
```

x

```
function string: show2d(array [$$E,$$F] of any $T: x)
```

x

```
function string: show2d_indexed(array [int] of string: row_hdr,
                                array [int] of string: col_hdr,
                                array [int,int] of string: vs)
```

vsrow_hdrcol_hdr

```
function string: show3d(array [int,int,int] of any $T: x)
```

x

```
function string: showJSON(any $T: x)
function string: showJSON(array [$U] of any $T: x)
```

x

```
function string: show_float(int: w, int: p, var float: x)
```

xwwpp

```
function string: show_int(int: w, var int: x)
```

xww

```
function int: string_length(string: s)
```

s

Functions for enums

```
1.  function $$E: enum_next($$E: x)
2.  function opt $$E: enum_next(opt $$E: x)
3.  function var $$E: enum_next(var $$E: x)
4.  function var opt $$E: enum_next(var opt $$E: x)
5.  function $$E: enum_next(set of $$E: e, $$E: x)
```

```
6.  function opt $$E: enum_next(set of $$E: e, opt $$E: x)

7.  function var $$E: enum_next(set of $$E: e, var $$E: x)

8.  function var opt $$E: enum_next(set of $$E: e, var opt $$E: x)
```

x

xe

```
function set of $$E: enum_of(var opt $$E: x)
function set of $$E: enum_of(var set of $$E: x)
function set of $$E: enum_of(array [$T] of var opt $$E: x)
function set of $$E: enum_of(array [$T] of var set of $$E: x)
```

x

```
1.  function $$E: enum_prev($$E: x)

2.  function opt $$E: enum_prev(opt $$E: x)

3.  function var $$E: enum_prev(var $$E: x)

4.  function var opt $$E: enum_prev(var opt $$E: x)

5.  function $$E: enum_prev(set of $$E: e, $$E: x)

6.  function opt $$E: enum_prev(set of $$E: e, opt $$E: x)

7.  function var $$E: enum_prev(set of $$E: e, var $$E: x)

8.  function var opt $$E: enum_prev(set of $$E: e, var opt $$E: x)
```

x

xe

```
function $$E: to_enum(set of $$E: X, int: x)
function opt $$E: to_enum(set of $$E: X, opt int: x)
function var $$E: to_enum(set of $$E: X, var int: x)
function var opt $$E: to_enum(set of $$E: X, var opt int: x)
function array [$U] of $$E: to_enum(set of $$E: X,
                                    array [$U] of int: x)
function array [$U] of opt $$E: to_enum(set of $$E: X,
```

```
                                        array [$U] of opt int: x)
function array [$U] of var $$E: to_enum(set of $$E: X,
                                        array [$U] of var int: x)
function array [$U] of var opt $$E: to_enum(set of $$E: X,
                                            array [$U] of var opt int: x)
function set of $$E: to_enum(set of $$E: X, set of int: x)
function array [$U] of set of $$E: to_enum(set of $$E: X,
                                           array [$U] of set of int: x)
function array [$U] of var set of $$E: to_enum(set of $$E: X,
                                               array [$U] of var set of int: x)
```

xX

4.2.1.2 Annotations

General annotations

Functions and Predicates

```
annotation mzn_output_section(string: s)
```

soutput :: "my_section" ["hello, world\n"]output ["hello, world\n"] to "my_section

---

Annotations

```
annotation add_to_output
```

```
annotation annotated_expression
```

```
annotation cache_result
```

```
annotation computed_domain
```

```
annotation constraint_name(string: s)
```

s

```
1.  annotation defines_var(var opt $T: c)
2.  annotation defines_var(array [$U] of var opt $T: arr)
```

c

arr

```
annotation doc_comment(string: s)
```

s

```
annotation domain_change_constraint
```

```
annotation empty_annotation
```

```
annotation expression_name(string: s)
```

s

```
annotation is_defined_var
```

```
annotation is_reverse_map
```

```
annotation maybe_partial
```

```
annotation mzn_add_annotated_expression(int: idx)
```

```
annotation mzn_check_enum_var(array [int] of set of int: x)
```

x

```
annotation mzn_check_var
```

```
annotation mzn_constraint_name(string: n)
```

n

```
annotation mzn_deprecated(string: version, string: explanation)
```

versionexplanation

```
annotation mzn_expression_name(string: n)
```

n

```
annotation mzn_path(string: s)
```

s–

```
annotation mzn_rhs_from_assignment
```

```
annotation no_cse
```

```
annotation no_output
```

```
annotation 'output'(any $T: x)
annotation 'output'(array [$U] of any $T: x)
```

x

```
annotation output_array(array [$U] of set of int: a)
```

a

```
annotation output_only
```

```
annotation output_var
```

```
annotation promise_commutative
```

```
annotation promise_total
```

```
annotation var_is_introduced
```

Propagation strength annotations

```
annotation bounds
```

```
annotation bounds_propagation
```

```
annotation domain
```

```
annotation domain_propagation
```

```
annotation value_propagation
```

Search annotations

Variable selection annotations

```
annotation anti_first_fail
```

```
annotation dom_w_deg
```

```
annotation first_fail
```

```
annotation impact
```

```
annotation input_order
```

```
annotation largest
```

```
annotation max_regret
```

```
annotation most_constrained
```

```
annotation occurrence
```

```
annotation smallest
```

Value choice annotations

```
annotation indomain
```

```
annotation indomain_interval
```

```
annotation indomain_max
```

```
annotation indomain_median
```

```
annotation indomain_middle
```

```
annotation indomain_min
```

```
annotation indomain_random
```

```
annotation indomain_reverse_split
```

```
annotation indomain_split
```

```
annotation indomain_split_random
```

```
annotation outdomain_max
```

```
annotation outdomain_median
```

```
annotation outdomain_min
```

```
annotation outdomain_random
```

Exploration strategy annotations

```
annotation complete
```

Restart annotations

```
annotation restart_constant(int: scale)
```

scale

```
annotation restart_geometric(float: base, int: scale)
```

basescale

```
annotation restart_linear(int: scale)
```

scale

```
annotation restart_luby(int: scale)
```

scale

```
annotation restart_none
```

Other declarations

Functions and Predicates

```
1.  annotation bool_search(array [$X] of var bool: x,
                           ann: select,
                           ann: choice,
                           ann: explore)

2.  annotation bool_search(array [$X] of var bool: x,
                           ann: select,
                           ann: choice)

3.  annotation bool_search(array [$X] of var opt bool: x,
                           ann: select,
                           ann: choice,
                           ann: explore)

4.  annotation bool_search(array [$X] of var opt bool: x,
                           ann: select,
                           ann: choice)
```

xselectchoiceexplorex

xselectchoicex

xselectchoiceexplorex

xselectchoicex

```
1.  annotation float_search(array [$X] of var float: x,
                            float: prec,
```

```
                            ann: select,
                            ann: choice,
                            ann: explore)

2.  annotation float_search(array [$X] of var float: x,
                            float: prec,
                            ann: select,
                            ann: choice)

3.  annotation float_search(array [$X] of var opt float: x,
                            float: prec,
                            ann: select,
                            ann: choice,
                            ann: explore)

4.  annotation float_search(array [$X] of var opt float: x,
                            float: prec,
                            ann: select,
                            ann: choice)
```

xprecselectchoiceexplorex

xprecselectchoicex

xprecselectchoiceexplorex

xprecselectchoicex

```
1.  annotation int_search(array [$X] of var int: x,
                          ann: select,
                          ann: choice,
                          ann: explore)

2.  annotation int_search(array [$X] of var int: x,
                          ann: select,
                          ann: choice)

3.  annotation int_search(array [$X] of var opt int: x,
                          ann: select,
                          ann: choice,
                          ann: explore)

4.  annotation int_search(array [$X] of var opt int: x,
                          ann: select,
                          ann: choice)
```

xselectchoiceexplorex

xselectchoicex

xselectchoiceexplorex

xselectchoicex

```
1.  annotation set_search(array [$X] of var set of int: x,
                          ann: select,
                          ann: choice,
                          ann: explore)

2.  annotation set_search(array [$X] of var set of int: x,
                          ann: select,
                          ann: choice)
```

xselectchoiceexplorex

xselectchoicex

Annotations

```
annotation seq_search(array [int] of ann: s)
```

s

Warm start annotations

Warm start annotations with optional values

```
1.  annotation warm_start(array [int] of var bool: x,
                          array [int] of opt bool: v)

2.  annotation warm_start(array [int] of var int: x,
                          array [int] of opt int: v)

3.  annotation warm_start(array [int] of var float: x,
                          array [int] of opt float: v)

4.  annotation warm_start(array [int] of var set of int: x,
                          array [int] of opt set of int: v)
```

vx

vx

vx

vx

Other declarations

```
1.  annotation warm_start(array [int] of var bool: x,
                          array [int] of bool: v)

2.  annotation warm_start(array [int] of var int: x, array [int] of int: v)

3.  annotation warm_start(array [int] of var float: x,
                          array [int] of float: v)

4.  annotation warm_start(array [int] of var set of int: x,
                          array [int] of set of int: v)
```

vx

vx

vx

vx

```
annotation warm_start_array(array [int] of ann: w)
```

w

Large Neighbourhood Search annotations

```
annotation relax_and_reconstruct(array [int] of var int: x, int: p)
```

xp

xp

Context annotations

```
annotation ctx_mix
```

```
annotation ctx_neg
```

```
annotation ctx_pos
```

```
annotation ctx_root
```

Redundant and symmetry breaking constraints

```
predicate implied_constraint(var bool: b)
```

b

```
predicate redundant_constraint(var bool: b)
```

b

```
predicate symmetry_breaking_constraint(var bool: b)
```

b

4.2.1.3 Option type support

Option type support for Booleans

```
predicate absent(var opt bool: x)
```

x

```
predicate deopt(var opt bool: x)
```

xx

```
predicate occurs(var opt bool: x)
```

x

Option type support for integers

```
predicate absent(var opt int: x)
```

x

```
function var $$E: deopt(var opt $$E: x)
```

xx

```
test had_zero(var opt int: x)
test had_zero(opt int: x)
test had_zero(array [int] of var opt int: x)
```

x

$(x) \wedge (x) = 0$

```
predicate occurs(var opt int: x)
```

x

Option type support for floats

```
predicate absent(var opt float: x)
```

x

```
function var float: deopt(var opt float: x)
```

xx

```
predicate occurs(var opt float: x)
```

x

Other declarations

```
1.  test absent($T: x)
2.  test absent(var $T: x)
3.  test absent(set of $$T: x)
4.  test absent(var set of $$T: x)
5.  test absent(opt $T: x)
```

x

x

```
 1.  function $$T: deopt(opt $$T: x)
 2.  function $T: deopt(opt $T: x)
 3.  function array [$$U] of $$T: deopt(array [$$U] of opt $$T: x)
 4.  function array [$$U] of $T: deopt(array [$$U] of opt $T: x)
 5.  function var $T: deopt(var $T: x)
 6.  function $T: deopt($T: x)
 7.  function array [$U] of var $T: deopt(array [$U] of var $T: x)
 8.  function array [$$U] of var bool: deopt(array [$$U] of var opt bool: x)
 9.  function array [$$U] of var $$E: deopt(array [$$U] of var opt $$E: x)
10.  function array [$$U] of var float: deopt(array [$$U] of var opt float: x)
```

xx

xx

xx

xx

xx

```
1. test occurs($T: x)
2. test occurs(var $T: x)
3. test occurs(set of $$T: x)
4. test occurs(var set of $$T: x)
5. test occurs(opt $T: x)
```

x

x

4.2.1.4 Array sorting operations

```
function array [int] of $$E: arg_sort(array [$$E] of int: x)
function array [int] of $$E: arg_sort(array [$$E] of float: x)
```

pxxpixpi

xpixpi→pipi

```
function array [$$E] of int: sort(array [$$E] of int: x)
function array [$$E] of float: sort(array [$$E] of float: x)
function array [$$E] of bool: sort(array [$$E] of bool: x)
```

x

```
function array [$$E] of any $T: sort_by(array [$$E] of any $T: x,
                                        array [$$E] of int: y)
function array [$$E] of any $T: sort_by(array [$$E] of any $T: x,
                                        array [$$E] of float: y)
```

xy

yiyjijxixj

4.2.1.5 Language information

Constants

```
opt int: mzn_min_version_required
```

Functions and Predicates

```
function int: mzn_compiler_version()
```

```
function string: mzn_version_to_string(int: v)
```

v

4.2.1.6 Compiler options

Constants

```
opt bool: mzn_absent_zero
```

```
opt bool: mzn_half_reify_clause
```

```
opt bool: mzn_ignore_redundant_constraints
```

```
opt bool: mzn_ignore_symmetry_breaking_constraints
```

```
opt bool: mzn_opt_annotate_computed_domains
```

```
opt bool: mzn_opt_annotate_defines_var
```

```
opt bool: mzn_opt_only_range_domains
```

Functions and Predicates

```
test mzn_check_absent_zero()
```

```
test mzn_check_annotate_computed_domains()
```

```
test mzn_check_annotate_defines_var()
```

```
test mzn_check_half_reify_clause()
```

```
test mzn_check_ignore_redundant_constraints()
```

```
test mzn_check_ignore_symmetry_breaking_constraints()
```

```
test mzn_check_only_range_domains()
```

4.2.1.7 Conditionals

```
predicate if_then_else(array [int] of var bool: c,
                       array [int] of int: x,
                       var int: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of var int: x,
                       var int: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of opt int: x,
                       var opt int: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of var opt int: x,
                       var opt int: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of bool: x,
                       var bool: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of var bool: x,
                       var bool: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of opt bool: x,
                       var opt bool: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of var opt bool: x,
                       var opt bool: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of float: x,
                       var float: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of var float: x,
                       var float: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of opt float: x,
                       var opt float: y)
predicate if_then_else(array [int] of var bool: c,
                       array [int] of var opt float: x,
                       var opt float: y)
predicate if_then_else(array [int] of var bool: c,
```

```
                        array [int] of set of int: x,
                        var set of int: y)
predicate if_then_else(array [int] of var bool: c,
                        array [int] of var set of int: x,
                        var set of int: y)
```

$\{[i] \wedge \nexists [1..i-1] \rightarrow = [i]\}$

c

```
predicate if_then_else_partiality(array [int] of var bool: c,
                                  array [int] of var bool: d,
                                  var bool: b)
```

cdib

4.2.1.8 Reflection operations

```
test annotate(any $T: x, ann: a)
```

xa

```
function set of $$E: dom(var $$E: x)
function set of $$E: dom(var opt $$E: x)
```

x

```
function set of $$E: dom_array(array [$T] of var $$E: x)
function set of $$E: dom_array(array [$T] of var opt $$E: x)
```

x

```
function set of int: dom_array_occurring(array [$T] of var opt int: x)
```

x

```
function set of int: dom_bounds_array(array [$T] of var int: x)
function set of int: dom_bounds_array(array [$T] of var opt int: x)
```

x

```
function int: dom_size(var int: x)
```

x

```
1.  function $T: fix(var $T: x)
2.  function opt $T: fix(var opt $T: x)
3.  function array [$U] of $T: fix(array [$U] of var $T: x)
4.  function array [$U] of opt $T: fix(array [$U] of var opt $T: x)
```

x

x

```
test has_ann(any $T: x, ann: a)
```

xa

```
test has_bounds(var int: x)
test has_bounds(var float: x)
```

x

```
test has_ub_set(var set of int: x)
```

x

```
1.  test is_fixed(any $T: x)
2.  test is_fixed(array [$U] of any $T: x)
```

x

x

```
test is_same(any $T: x, any $U: y)
```

xy

```
1.  function $$E: lb(var $$E: x)
2.  function $$E: lb(var opt $$E: x)
3.  function float: lb(var float: x)
4.  function float: lb(var opt float: x)
5.  function set of $$E: lb(var set of $$E: x)
6.  function array [$U] of $$E: lb(array [$U] of var $$E: x)
7.  function array [$U] of float: lb(array [$U] of var float: x)
8.  function array [$U] of set of $$E: lb(array [$U] of var set of $$E: x)
```

x

x

```
1.  function $$E: lb_array(array [$U] of var opt $$E: x)
2.  function float: lb_array(array [$U] of var opt float: x)
3.  function set of $$E: lb_array(array [$U] of var set of $$E: x)
```

x

x

```
1.  function $$E: ub(var $$E: x)
2.  function $$E: ub(var opt $$E: x)
3.  function float: ub(var float: x)
4.  function float: ub(var opt float: x)
5.  function set of $$E: ub(var set of $$E: x)
6.  function array [$U] of $$E: ub(array [$U] of var $$E: x)
```

```
7.  function array [$U] of float: ub(array [$U] of var float: x)

8.  function array [$U] of set of $$E: ub(array [$U] of var set of $$E: x)
```

x

x

```
1.  function $$E: ub_array(array [$U] of var opt $$E: x)

2.  function float: ub_array(array [$U] of var opt float: x)

3.  function set of $$E: ub_array(array [$U] of var set of $$E: x)
```

x

x

4.2.1.9 Assertions and debugging functions

Constants

```
bool: debug_mode
```

--debug

Functions and Predicates

```
test abort(string: msg)
```

msg

```
1.  function any $T: assert(bool: b, string: msg, any $T: x)

2.  function array [$U] of any $T: assert(bool: b,
                                          string: msg,
                                          array [$U] of any $T: x)

3.  test assert(bool: b, string: msg)
```

bxmsg

bmsg

```
test assert_dbg(bool: b, string: msg)
```

--debugbmsg

```
1.  function $T: default(opt $T: x, $T: y)

2.  function opt $T: default(opt $T: x, opt $T: y)

3.  function var $T: default(var opt $T: x, var $T: y)

4.  function var opt $T: default(var opt $T: x, var opt $T: y)

5.  function array [$U] of $T: default(array [$U] of $T: x,
                                       array [$U] of $T: y)

6.  function array [$U] of opt $T: default(array [$U] of opt $T: x,
                                           array [$U] of opt $T: y)

7.  function array [$U] of var $T: default(array [$U] of var $T: x,
                                           array [$U] of var $T: y)

8.  function array [$U] of var opt $T: default(array [$U] of var opt $T: x,
                                               array [$U] of var opt $T: y)

9.  function set of $$E: default(set of $$E: x, set of $$E: y)

10. function var set of $$E: default(var set of $$E: x, var set of $$E: y)

11. function set of $T: default(opt set of $T: x, set of $T: y)

12. function opt set of $T: default(opt set of $T: x, opt set of $T: y)
```

xxy

xxy

xxy

```
annotation expression_name_dbg(string: s)
```

expression_names–

```
function string: logstream_to_string()
```

```
test mzn_internal_check_debug_mode()
```

--debug

```
1.  function any $T: trace(string: msg, any $T: x)

2.  function array [$U] of any $T: trace(string: msg,
                                         array [$U] of any $T: x)

3.  test trace(string: msg)
```

xmsg

msg

```
test trace_dbg(string: msg)
```

--debugmsg

```
function any $T: trace_exp(any $T: x)
function array [$U] of any $T: trace_exp(array [$U] of any $T: x)
```

x

```
1.  function any $T: trace_logstream(string: msg, any $T: x)

2.  function array [$U] of any $T: trace_logstream(string: msg,
                                                   array [$U] of any $T: x)

3.  test trace_logstream(string: msg)
```

xmsg

msg

```
1.  function any $T: trace_stdout(string: msg, any $T: x)

2.  function array [$U] of any $T: trace_stdout(string: msg,
                                                 array [$U] of any $T: x)
```

```
3.  test trace_stdout(string: msg)
```

xmsg

msg

```
test trace_to_json_section(string: section, opt $T: x)
test trace_to_json_section(string: section, array [$U] of opt $T: x)
```

xsection

```
1.  function any $T: trace_to_section(string: section,
                                      string: msg,
                                      any $T: x)

2.  function array [$U] of any $T: trace_to_section(string: section,
                                                    string: msg,
                                                    array [$U] of any $T: x)

3.  test trace_to_section(string: section, string: msg)
```

xmsgsection

msgsection

Annotations

```
annotation mzn_break_here
```

4.2.1.10 Random Number Generator builtins

```
test bernoulli(float: p)
```

```
function int: binomial(int: t, float: p)
```

tp

```
function float: cauchy(float: mean, float: scale)
function float: cauchy(int: mean, float: scale)
```

,

```
function float: chisquared(int: n)
function float: chisquared(float: n)
```

```
function int: discrete_distribution(array [int] of int: weights)
```

```
function float: exponential(int: lambda)
function float: exponential(float: lambda)
```

```
function float: fdistribution(float: d1, float: d2)
function float: fdistribution(int: d1, int: d2)
```

,

```
function float: gamma(float: alpha, float: beta)
function float: gamma(int: alpha, float: beta)
```

,

```
function float: lognormal(float: mean, float: std)
function float: lognormal(int: mean, float: std)
```

,

```
function float: normal(float: mean, float: std)
function float: normal(int: mean, float: std)
```

,

```
1.  function int: poisson(float: mean)

2.  function int: poisson(int: mean)
```

mean

mean

```
function float: tdistribution(float: n)
function float: tdistribution(int: n)
```

,

```
1.  function float: uniform(float: lowerbound, float: upperbound)

2.  function int: uniform(int: lowerbound, int: upperbound)

3.  function int: uniform(set of int: S)
```

,

S

```
function float: weibull(float: shape, float: scale)
function float: weibull(int: shape, float: scale)
```

,

4.2.1.11 Stuctured Output

```
function string: show_array2d_bool(array [int,int] of var bool: x)
```

x

```
1.  function string: show_gantt(array [$$T] of var int: start,
                                array [$$T] of var int: dur,
                                array [$$T] of string: name)

2.  function string: show_gantt(array [$$T] of var int: start,
```

```
                                array [$$T] of var int: dur)
```

startdurname

startdur

4.2.2 Global constraints

4.2.2.1 All-Different and related constraints

```
1. predicate all_different(array [$X] of var int: x)

2. predicate all_different(array [$X] of var set of int: x)

3. predicate all_different(array [$X] of var opt int: x)
```

x

x

```
predicate all_different_except(array [$X] of var int: vs,
                               set of int: S)
```

vsS

```
predicate all_different_except_0(array [$X] of var int: vs)
```

vs

```
predicate all_disjoint(array [$X] of var set of int: S)
```

S

```
predicate all_equal(array [$X] of var int: x)
predicate all_equal(array [$X] of var set of int: x)
```

x

```
1. predicate nvalue(var int: n, array [$X] of var int: x)
2. predicate nvalue(var int: n, array [$X] of var opt int: x)
3. function var int: nvalue(array [$X] of var int: x)
4. function var int: nvalue(array [$X] of var opt int: x)
```

xn

x

```
predicate symmetric_all_different(array [int] of var int: x)
```

xixij→xji

4.2.2.2 Lexicographic constraints

```
predicate lex2(array [int,int] of var int: x)
```

x

```
predicate lex2_strict(array [int,int] of var int: x)
```

x

```
predicate lex_chain(array [int,int] of var bool: a)
predicate lex_chain(array [int,int] of var int: a)
```

a

```
predicate lex_chain_greater(array [int,int] of var bool: a)
predicate lex_chain_greater(array [int,int] of var int: a)
```

a

```
predicate lex_chain_greatereq(array [int,int] of var bool: a)
predicate lex_chain_greatereq(array [int,int] of var int: a)
```

a

```
predicate lex_chain_greatereq_orbitope(array [int,int] of var int: a,
                                       int: kind)
```

akind

```
predicate lex_chain_less(array [int,int] of var bool: a)
predicate lex_chain_less(array [int,int] of var int: a)
```

a

```
predicate lex_chain_lesseq(array [int,int] of var bool: a)
predicate lex_chain_lesseq(array [int,int] of var int: a)
```

a

```
predicate lex_chain_lesseq_orbitope(array [int,int] of var int: a,
                                    int: kind)
```

akind

```
predicate lex_greater(array [int] of var bool: x,
                      array [int] of var bool: y)
predicate lex_greater(array [int] of var int: x,
                      array [int] of var int: y)
predicate lex_greater(array [int] of var float: x,
                      array [int] of var float: y)
predicate lex_greater(array [int] of var set of int: x,
                      array [int] of var set of int: y)
```

xy

```
predicate lex_greatereq(array [int] of var bool: x,
                        array [int] of var bool: y)
predicate lex_greatereq(array [int] of var int: x,
                        array [int] of var int: y)
predicate lex_greatereq(array [int] of var float: x,
                        array [int] of var float: y)
predicate lex_greatereq(array [int] of var set of int: x,
                        array [int] of var set of int: y)
```

xy

```
predicate lex_less(array [int] of var bool: x,
                   array [int] of var bool: y)
predicate lex_less(array [int] of var int: x,
                   array [int] of var int: y)
predicate lex_less(array [int] of var float: x,
                   array [int] of var float: y)
predicate lex_less(array [int] of var set of int: x,
                   array [int] of var set of int: y)
```

xy

```
predicate lex_lesseq(array [int] of var bool: x,
                     array [int] of var bool: y)
predicate lex_lesseq(array [int] of var float: x,
                     array [int] of var float: y)
predicate lex_lesseq(array [int] of var int: x,
                     array [int] of var int: y)
predicate lex_lesseq(array [int] of var set of int: x,
                     array [int] of var set of int: y)
```

xy

1. ```
   predicate seq_precede_chain(array [int] of var int: x)
   ```

2. ```
   predicate seq_precede_chain(array [int] of var set of int: x)
   ```

iixi

ixii

```
predicate strict_lex2(array [int,int] of var int: x)
```

x

1. ```
   predicate value_precede($$E: s, $$E: t, array [int] of var $$E: x)
   ```

2. ```
   predicate value_precede($$E: s, $$E: t, array [int] of var opt $$E: x)
   ```

3. ```
   predicate value_precede($$E: s,
                           $$E: t,
                           array [int] of var set of $$E: x)
   ```
   ```

stx

xtxs

stx

xtsxst

```
1. predicate value_precede_chain(array [int] of $$E: c,
                                 array [int] of var $$E: x)

2. predicate value_precede_chain(array [int] of int: c,
                                 array [int] of var opt int: x)

3. predicate value_precede_chain(array [int] of $$E: c,
                                 array [int] of var set of $$E: x)
```

cicix

xcixci

cicix

xcicixcici

```
predicate var_perm_sym(array [$$X] of var $$Y: x,
                       array [$$Z,$$X] of $$X: p)
```

xp

```
predicate var_sqr_sym(array [$$X,$$X] of var $$Y: x)
```

x

4.2.2.3 Sorting constraints

```
1. predicate arg_sort(array [$$E] of var int: x,
                      array [int] of var $$E: p)

2. predicate arg_sort(array [$$E] of var float: x,
                      array [int] of var $$E: p)

3. function array [int] of var $$E: arg_sort(array [$$E] of var int: x)

4. function array [int] of var $$E: arg_sort(array [$$E] of var float: x)
```

pxxpixpi

xpixpi→pipi

pxxpixpi

xpixpi→pipi

```
predicate decreasing(array [$X] of var bool: x)
predicate decreasing(array [$X] of var opt float: x)
predicate decreasing(array [$X] of var opt int: x)
predicate decreasing(array [$X] of var set of int: x)
```

x

```
predicate increasing(array [$X] of var bool: x)
predicate increasing(array [$X] of var float: x)
predicate increasing(array [$X] of var opt float: x)
predicate increasing(array [$X] of var int: x)
predicate increasing(array [$X] of var opt int: x)
predicate increasing(array [$X] of var set of int: x)
```

x

---

1. `predicate sort(array [int] of var int: x, array [int] of var int: y)`

2. `function array [int] of var int: sort(array [int] of var int: x)`

xyy

x

```
predicate strictly_decreasing(array [$X] of var opt int: x)
predicate strictly_decreasing(array [$X] of var opt float: x)
predicate strictly_decreasing(array [$X] of var set of int: x)
```

x

---

1. `predicate strictly_increasing(array [$X] of var int: x)`

2. `predicate strictly_increasing(array [$X] of var opt int: x)`

3. `predicate strictly_increasing(array [$X] of var float: x)`

```
4.  predicate strictly_increasing(array [$X] of var opt float: x)

5.  predicate strictly_increasing(array [$X] of var set of int: x)
```

x

x

4.2.2.4 Channeling constraints

```
predicate int_set_channel(array [int] of var int: x,
                          array [int] of var set of int: y)
```

xyxij↔iyj

```
1.  predicate inverse(array [$$X] of var $$Y: f,
                      array [$$Y] of var $$X: invf)

2.  function array [$$E] of var $$F: inverse(array [$$F] of var $$E: f)

3.  function array [$$E] of var opt $$F: inverse(array [$$F] of var opt $$E: f)

4.  function array [$$E] of $$F: inverse(array [$$F] of $$E: f)
```

finvf

f

```
predicate inverse_in_range(array [$$A] of var $$B: X,
                           array [$$B] of var $$A: Y)
```

iXjjYjYi

jYiiXiXj

```
predicate inverse_set(array [$$X] of var set of $$Y: f,
                      array [$$Y] of var set of $$X: invf)
```

finvfjiij'

```
predicate link_set_to_booleans(var set of $$E: s,
                               array [$$E] of var bool: b)
```

bsis↔bi

bs

4.2.2.5 Counting constraints

count(i in x)(i=c) <= ddistributeglobal_cardinality

```
1.  predicate among(var int: n, array [$X] of var $$E: x, set of $$E: v)

2.  function var int: among(array [$X] of var $$E: x, set of $$E: v)
```

nxv

xv

```
predicate at_least(int: n,
                   array [$X] of var set of $$E: x,
                   set of $$E: v)
```

nxv

```
predicate at_most(int: n,
                  array [$X] of var set of $$E: x,
                  set of $$E: v)
```

nxv

```
predicate at_most1(array [$X] of var set of int: s)
```

s

```
1.  function var int: count(array [$X] of var opt $$E: x, var $$E: y)

2.  predicate count(array [$X] of var opt $$E: x, var $$E: y, var int: c)
```

yx

cyx

```
1.  predicate count_eq(array [$X] of var $$E: x, var $$E: y, var int: c)

2.  predicate count_eq(array [$X] of var opt $$E: x,
                       var $$E: y,
```

```
                      var int: c)

3.  predicate count_eq(array [$X] of var $$E: x, $$E: y, int: c)

4.  function var int: count_eq(array [$X] of var opt $$E: x, var $$E: y)
```

cyx

yx

```
1.  predicate count_geq(array [$X] of var $$E: x, var $$E: y, var int: c)

2.  predicate count_geq(array [$X] of var opt $$E: x,
                        var $$E: y,
                        var int: c)

3.  predicate count_geq(array [$X] of var $$E: x, $$E: y, int: c)
```

cyx

cyx

cyx

```
predicate count_gt(array [$X] of var $$E: x, var $$E: y, var int: c)
predicate count_gt(array [$X] of var opt $$E: x,
                   var $$E: y,
                   var int: c)
predicate count_gt(array [$X] of var $$E: x, $$E: y, int: c)
```

cyx

```
predicate count_leq(array [$X] of var $$E: x, var $$E: y, var int: c)
predicate count_leq(array [$X] of var opt $$E: x,
                    var $$E: y,
                    var int: c)
predicate count_leq(array [$X] of var $$E: x, $$E: y, int: c)
```

cyx

```
predicate count_lt(array [$X] of var $$E: x, var $$E: y, var int: c)
predicate count_lt(array [$X] of var opt $$E: x,
                   var $$E: y,
                   var int: c)
predicate count_lt(array [$X] of var $$E: x, $$E: y, int: c)
```

cyx

```
predicate count_neq(array [$X] of var $$E: x, var $$E: y, var int: c)
predicate count_neq(array [$X] of var opt $$E: x,
                    var $$E: y,
                    var int: c)
predicate count_neq(array [$X] of var $$E: x, $$E: y, int: c)
```

cyx

```
1.  predicate distribute(array [$X] of var int: card,
                         array [$X] of var int: value,
                         array [$Y] of var int: base)

2.  function array [$X] of var int: distribute(array [$X] of var int: value,
                                               array [$Y] of var int: base)
```

cardivalueibasevalue

valueibasevalue

```
predicate exactly(int: n,
                  array [$X] of var set of $$E: x,
                  set of $$E: v)
```

nxv

```
1.  predicate global_cardinality(array [$X] of var $$E: x,
                                 array [$Y] of $$E: cover,
                                 array [$Y] of var int: counts)

2.  predicate global_cardinality(array [$X] of var opt $$E: x,
                                 array [$Y] of $$E: cover,
                                 array [$Y] of var int: counts)

3.  predicate global_cardinality(array [$X] of var $$E: x,
                                 array [$Y] of $$E: cover,
                                 array [$Y] of int: lbound,
                                 array [$Y] of int: ubound)

4.  predicate global_cardinality(array [$X] of var opt $$E: x,
                                 array [$Y] of $$E: cover,
                                 array [$Y] of int: lbound,
                                 array [$Y] of int: ubound)

5.  predicate global_cardinality(array [$X] of var set of $$E: x,
                                 array [$Y] of $$E: cover,
                                 array [$Y] of var int: counts)
```

```
6.  predicate global_cardinality(array [$X] of var set of $$E: x,
                                 array [$Y] of $$E: cover,
                                 array [$Y] of int: lbound,
                                 array [$Y] of int: ubound)

7.  function array [$Y] of var int: global_cardinality(array [$X] of var $$E: x,
                                                       array [$Y] of $$E: cover)

8.  function array [$Y] of var int: global_cardinality(array [$X] of var opt $$E: x,
                                                       array [$Y] of $$E: cover)
```

coverixcountsi

icoverilboundiuboundix

coverixcountsi

icoverilboundiuboundix

coverix

```
1.  predicate global_cardinality_closed(array [$X] of var $$E: x,
                                        array [$Y] of $$E: cover,
                                        array [$Y] of var int: counts)

2.  predicate global_cardinality_closed(array [$X] of var opt $$E: x,
                                        array [$Y] of $$E: cover,
                                        array [$Y] of var int: counts)

3.  predicate global_cardinality_closed(array [$X] of var $$E: x,
                                        array [$Y] of $$E: cover,
                                        array [$Y] of int: lbound,
                                        array [$Y] of int: ubound)

4.  predicate global_cardinality_closed(array [$X] of var opt $$E: x,
                                        array [$Y] of $$E: cover,
                                        array [$Y] of int: lbound,
                                        array [$Y] of int: ubound)

5.  predicate global_cardinality_closed(array [$X] of var set of $$E: x,
                                        array [$Y] of $$E: cover,
                                        array [$Y] of var int: counts)

6.  predicate global_cardinality_closed(array [$X] of var set of $$E: x,
                                        array [$Y] of $$E: cover,
                                        array [$Y] of int: lbound,
                                        array [$Y] of int: ubound)

7.  function array [$Y] of var int: global_cardinality_closed(array [$X] of var $
↪$E: x,
                                                              array [$Y] of $$E:␣
```

```
↪cover)

8.  function array [$Y] of var int: global_cardinality_closed(array [$X] of var opt
↪$$E: x,
                                                              array [$Y] of $$E:␣
↪cover)
```

coverixcountsi

xcover

coverixcountsi

xcover

icoverilboundiuboundix

xcover

icoverilboundiuboundix

xcover

coverixcountsi

xcover

icoverilboundiuboundix

xcover

coverix

xcover

coverix

xcover

4.2.2.6 Array-related constraints

```
predicate element(var $$E: i, array [$$E] of var bool: x, var bool: y)
predicate element(var $$E: i,
                  array [$$E] of var float: x,
                  var float: y)
predicate element(var $$E: i, array [$$E] of var $$T: x, var $$T: y)
predicate element(var $$E: i,
                  array [$$E] of var set of $$T: x,
                  var set of $$T: y)
```

iyx

```
1.  predicate member(array [int] of var bool: x, var bool: y)

2.  predicate member(array [int] of var float: x, var float: y)
```

```
3.  predicate member(array [int] of var $$E: x, var $$E: y)

4.  predicate member(array [int] of var set of $$E: x, var set of $$E: y)

5.  predicate member(var set of $$E: x, var $$E: y)
```

yx

yx

```
predicate write(array [$$E] of var int: I,
                var int: i,
                var int: v,
                array [$$E] of var int: O)
```

OIiv

IOIiIv

```
1.  predicate writes(array [$$X] of var int: I,
                     array [$$Y] of var int: P,
                     array [$$Y] of var int: V,
                     array [$$X] of var int: O)

2.  function array [$$X] of var int: writes(array [$$X] of var int: I,
                                            array [$$Y] of var int: P,
                                            array [$$Y] of var int: V)
```

OIPV

IOIPIV

IPV

IPIV

```
predicate writes_seq(array [$$X] of var int: I,
                     array [$$Y] of var int: P,
                     array [$$Y] of var int: V,
                     array [$$X] of var int: O)
```

OIPV

IOIPIV

4.2.2.7 Set-related constraints

```
predicate disjoint(var set of $$E: s1, var set of $$E: s2)
```

s1s2

```
predicate partition_set(array [int] of var set of $$E: S,
                        set of $$E: universe)
```

Suniverse

```
1.  predicate roots(array [$$X] of var $$Y: x,
                    var set of $$X: s,
                    var set of $$Y: t)

2.  function var set of $$X: roots(array [$$X] of var $$Y: x,
                                   var set of $$Y: t)
```

xitis

sxitis

4.2.2.8 Mathematical constraints

```
1.  function var $$E: arg_max(array [$$E] of var int: x)

2.  function var $$E: arg_max(array [$$E] of var bool: x)

3.  function var $$E: arg_max(array [$$E] of var float: x)

4.  function var $$E: arg_max(array [$$E] of var opt int: x)

5.  function var $$E: arg_max(array [$$E] of var opt bool: x)

6.  function var $$E: arg_max(array [$$E] of var opt float: x)
```

x

x

```
function var opt $$E: arg_max_weak(array [$$E] of var opt int: x)
function var opt $$E: arg_max_weak(array [$$E] of var opt bool: x)
function var opt $$E: arg_max_weak(array [$$E] of var opt float: x)
```

x

```
1.  function var $$E: arg_min(array [$$E] of var int: x)

2.  function var $$E: arg_min(array [$$E] of var bool: x)

3.  function var $$E: arg_min(array [$$E] of var float: x)

4.  function var $$E: arg_min(array [$$E] of var opt int: x)

5.  function var $$E: arg_min(array [$$E] of var opt bool: x)

6.  function var $$E: arg_min(array [$$E] of var opt float: x)
```

x

x

```
function var opt $$E: arg_min_weak(array [$$E] of var opt int: x)
function var opt $$E: arg_min_weak(array [$$E] of var opt bool: x)
function var opt $$E: arg_min_weak(array [$$E] of var opt float: x)
```

x

```
function var $$E: arg_val(array [$$E] of var bool: x, var bool: v)
function var $$E: arg_val(array [$$E] of var opt bool: x,
                          var opt bool: v)
function var $$E: arg_val(array [$$E] of var float: x, var float: v)
function var $$E: arg_val(array [$$E] of var opt float: x,
                          var float: v)
function var $$E: arg_val(array [$$E] of var $$V: x, var $$V: v)
function var $$E: arg_val(array [$$E] of var opt $$V: x,
                          var opt $$V: v)
```

vxvx

```
function var opt $$E: arg_val_weak(array [$$E] of var bool: x,
                                   var bool: v)
function var opt $$E: arg_val_weak(array [$$E] of var opt bool: x,
                                   var opt bool: v)
function var opt $$E: arg_val_weak(array [$$E] of var float: x,
                                   var float: v)
function var opt $$E: arg_val_weak(array [$$E] of var opt float: x,
                                   var opt float: v)
function var opt $$E: arg_val_weak(array [$$E] of var $$V: x,
                                   var $$V: v)
```

```
function var opt $$E: arg_val_weak(array [$$E] of var opt $$V: x,
                                   var opt $$V: v)
```

vxvvx

```
predicate maximum(var $$E: m, array [int] of var $$E: x)
predicate maximum(var float: m, array [int] of var float: x)
```

mx

x

```
1.  predicate maximum_arg(array [int] of var int: x, var int: i)

2.  predicate maximum_arg(array [$$E] of var bool: x, var $$E: i)

3.  predicate maximum_arg(array [$$E] of var float: x, var $$E: i)
```

ix

x

ix

x

```
predicate minimum(var float: m, array [int] of var float: x)
predicate minimum(var $$E: m, array [int] of var $$E: x)
```

mx

x

```
predicate minimum_arg(array [int] of var int: x, var int: i)
predicate minimum_arg(array [int] of var bool: x, var int: i)
predicate minimum_arg(array [int] of var float: x, var int: i)
```

ix

x

```
1.  function var float: piecewise_linear(var float: x,
                                         array [int] of float: xi,
                                         array [int] of float: vi)
```

```
2.  predicate piecewise_linear(var float: x,
                               var float: y,
                               array [int] of float: xi,
                               array [int] of float: vi)

3.  function var float: piecewise_linear(var float: x,
                                         array [int] of float: x_start,
                                         array [int] of float: x_end,
                                         array [int] of float: v_start,
                                         array [int] of float: v_end)

4.  predicate piecewise_linear(var float: x,
                               var float: y,
                               array [int] of float: x_start,
                               array [int] of float: x_end,
                               array [int] of float: v_start,
                               array [int] of float: v_end)
```

xxivi

yxxivi

xix_startiv_startix_endiv_endi

yxix_startiv_startix_endiv_endi

```
1.  predicate range(array [$$X] of var $$Y: x,
                    var set of $$X: s,
                    var set of $$Y: t)

2.  function var set of int: range(array [int] of var int: x,
                                   var set of int: s)
```

xstsx

xssx

```
predicate sliding_sum(int: low,
                      int: up,
                      int: seq,
                      array [int] of var int: vs)
```

vsi⋯vsiseqlowup

```
predicate sum_pred(var $$X: i,
                   array [$$X] of set of $$Y: sets,
                   array [$$Y] of int: cs,
                   var int: s)
```

csi1csiNsi1iNisets

‘’ ‘’

```
predicate sum_set(array [$$X] of $$Y: vs,
                  array [$$X] of int: ws,
                  var set of $$Y: x,
                  var int: s)
```

wsi1wsiNsvsi1vsiNx

4.2.2.9 Packing constraints

```
predicate bin_packing(int: c,
                      array [int] of var int: bin,
                      array [int] of int: w)
```

iwibinic

iwi

c

```
predicate bin_packing_capa(array [int] of int: c,
                           array [int] of var int: bin,
                           array [int] of int: w)
```

iwibinibcb

iwi

bcb

```
1. predicate bin_packing_load(array [int] of var int: load,
                              array [int] of var int: bin,
                              array [int] of int: w)

2. function array [int] of var int: bin_packing_load(array [int] of var int: bin,
                                                     array [int] of int: w)
```

iwibinibloadb

iwi

iwibini

iwi

```
predicate diffn(array [int] of var int: x,
                array [int] of var int: y,
                array [int] of var int: dx,
                array [int] of var int: dy)
```

ixiyidxidyi

```
predicate diffn_k(array [int,int] of var int: box_posn,
                  array [int,int] of var int: box_size)
```

kijbox_posnijjbox_sizeij

```
predicate diffn_nonstrict(array [int] of var int: x,
                          array [int] of var int: y,
                          array [int] of var int: dx,
                          array [int] of var int: dy)
```

ixiyidxidyi

```
predicate diffn_nonstrict_k(array [int,int] of var int: box_posn,
                            array [int,int] of var int: box_size)
```

kijbox_posnijjbox_sizeij

```
predicate geost(int: k,
                array [int,int] of int: rect_size,
                array [int,int] of int: rect_offset,
                array [int] of set of int: shape,
                array [int,int] of var int: x,
                array [int] of var int: kind)
```

k

k

rect_sizek

rect_offsetk

shapei

xxijij

kind

```
predicate geost_bb(int: k,
                   array [int,int] of int: rect_size,
                   array [int,int] of int: rect_offset,
                   array [int] of set of int: shape,
                   array [int,int] of var int: x,
                   array [int] of var int: kind,
                   array [int] of var int: l,
                   array [int] of var int: u)
```

kk

k

rect_sizek

rect_offsetk

shapei

xxijij

kind

llii

uuii

```
predicate geost_nonoverlap_k(array [int] of var int: x1,
                             array [int] of int: w1,
                             array [int] of var int: x2,
                             array [int] of int: w2)
```

k

x1

w1

x2

w2

```
predicate geost_smallest_bb(int: k,
                            array [int,int] of int: rect_size,
                            array [int,int] of int: rect_offset,
                            array [int] of set of int: shape,
                            array [int,int] of var int: x,
                            array [int] of var int: kind,
                            array [int] of var int: l,
```

```
                                array [int] of var int: u)
```

kk2k

k

rect_sizek

rect_offsetk

shapei

xxijij

kind

llii

uuii

```
predicate knapsack(array [$$I] of int: w,
                   array [$$I] of int: p,
                   array [$$I] of var int: x,
                   var int: W,
                   var int: P)
```

wp

wpx

w

p

x

W

P

4.2.2.10 Scheduling constraints

```
predicate alternative(var opt int: s0,
                      var int: d0,
                      array [int] of var opt int: s,
                      array [int] of var int: d)
```

s0d0sidi

```
1.  predicate cumulative(array [int] of var int: s,
                         array [int] of var int: d,
                         array [int] of var int: r,
                         var int: b)

2.  predicate cumulative(array [int] of var opt int: s,
                         array [int] of var int: d,
                         array [int] of var int: r,
                         var int: b)
```

sdrb

idiri

sdrb

idiri

```
predicate cumulatives(array [$$E] of var int: s,
                      array [$$E] of var int: d,
                      array [$$E] of var int: r,
                      array [$$E] of var $$M: m,
                      array [$$M] of var int: b,
                      bool: upper)
```

sdrmbupperimibjtj\tupperupperbj

idi

```
1.  predicate disjunctive(array [$$T] of var int: s,
                          array [$$T] of var int: d)

2.  predicate disjunctive(array [$$T] of var opt int: s,
                          array [$$T] of var int: d)
```

sd

idi

sd

idi

```
1.  predicate disjunctive_strict(array [$$T] of var int: s,
                                 array [$$T] of var int: d)

2.  predicate disjunctive_strict(array [$$T] of var opt int: s,
                                 array [$$T] of var int: d)
```

sd

idi

sd

idi

```
predicate span(var opt int: s0,
               var int: d0,
               array [$$E] of var opt int: s,
               array [$$E] of var int: d)
```

s0d0sidi

4.2.2.11 Graph constraints

fromto(from[i],to[i])from[i]to[i]

```
1.  predicate bounded_dpath(int: N,
                            int: E,
                            array [int] of int: from,
                            array [int] of int: to,
                            array [int] of int: w,
                            var int: s,
                            var int: t,
                            array [int] of var bool: ns,
                            array [int] of var bool: es,
                            var int: K)

2.  predicate bounded_dpath(array [int] of $$N: from,
                            array [int] of $$N: to,
                            array [int] of int: w,
```

```
                            var $$N: s,
                            var $$N: t,
                            array [$$N] of var bool: ns,
                            array [int] of var bool: es,
                            var int: K)
```

nsesstK

N

E

fromN

toN

w

s

t

ns

es

K

nsesstK

from

to

w

s

t

ns

es

K

```
1.  predicate bounded_path(int: N,
                           int: E,
                           array [int] of int: from,
                           array [int] of int: to,
                           array [int] of int: w,
                           var int: s,
                           var int: t,
                           array [int] of var bool: ns,
                           array [int] of var bool: es,
                           var int: K)
```

```
2.  predicate bounded_path(array [int] of $$N: from,
                           array [int] of $$N: to,
                           array [int] of int: w,
                           var $$N: s,
                           var $$N: t,
                           array [$$N] of var bool: ns,
                           array [int] of var bool: es,
                           var int: K)
```

nsesstK

N

E

fromN

toN

w

s

t

ns

es

K

nsesstK

from

to

w

s

t

ns

es

K

```
1.  predicate circuit(array [$$E] of var $$E: x)

2.  predicate circuit(array [$$E] of var opt $$E: x)
```

xxijji

xxijji

```
predicate connected(array [$$E] of $$N: from,
                    array [$$E] of $$N: to,
                    array [$$N] of var bool: ns,
                    array [$$E] of var bool: es)
```

nses

from

to

ns

es

```
predicate d_weighted_spanning_tree(int: N,
                                   int: E,
                                   array [int] of int: from,
                                   array [int] of int: to,
                                   array [int] of int: w,
                                   var int: r,
                                   array [int] of var bool: es,
                                   var int: K)
```

esrW

N

E

fromN

toN

w

r

es

K

```
predicate dag(array [$$E] of $$N: from,
              array [$$E] of $$N: to,
              array [$$N] of var bool: ns,
              array [$$E] of var bool: es)
```

nses

from

to

ns

es

```
predicate dconnected(array [$$E] of $$N: from,
                     array [$$E] of $$N: to,
                     array [$$N] of var bool: ns,
                     array [$$E] of var bool: es)
```

nses

from

to

ns

es

```
1.  predicate dpath(int: N,
                    int: E,
                    array [int] of int: from,
                    array [int] of int: to,
                    var int: s,
                    var int: t,
                    array [int] of var bool: ns,
                    array [int] of var bool: es)

2.  predicate dpath(array [int] of $$N: from,
                    array [int] of $$N: to,
                    var $$N: s,
                    var $$N: t,
                    array [$$N] of var bool: ns,
                    array [int] of var bool: es)
```

nsesst

N

E

fromN

toN

s

t

ns

es

nsesst

from

to

s

t

ns

es

```
1.  predicate dreachable(int: N,
                         int: E,
                         array [int] of int: from,
                         array [int] of int: to,
                         var int: r,
                         array [int] of var bool: ns,
                         array [int] of var bool: es)

2.  predicate dreachable(array [int] of $$N: from,
                         array [int] of $$N: to,
                         var $$N: r,
                         array [$$N] of var bool: ns,
                         array [int] of var bool: es)
```

nsesr

N

E

fromN

toN

r

ns

es

nsesr

from

to

r

ns

es

```
predicate dsteiner(int: N,
                   int: E,
                   array [int] of int: from,
                   array [int] of int: to,
                   array [int] of int: w,
                   var int: r,
                   array [int] of var bool: ns,
                   array [int] of var bool: es,
                   var int: K)
```

nsesrW

N

E

fromN

toN

w

r

ns

es

K

```
1.  predicate dtree(int: N,
                    int: E,
                    array [int] of int: from,
                    array [int] of int: to,
                    var int: r,
                    array [int] of var bool: ns,
                    array [int] of var bool: es)

2.  predicate dtree(array [$$E] of $$N: from,
                    array [$$E] of $$N: to,
                    var $$N: r,
                    array [$$N] of var bool: ns,
                    array [$$E] of var bool: es)
```

nsesr

N

E

fromN

toN

r

ns

es

nsesr

from

to

r

ns

es

```
predicate network_flow(array [$$E,1..2] of $$N: arc,
                       array [$$N] of int: balance,
                       array [$$E] of var int: flow)
```

arciarciarci

balancebalance = output - input

flow

```
predicate network_flow_cost(array [$$E,1..2] of $$N: arc,
                            array [$$N] of int: balance,
                            array [$$E] of int: weight,
                            array [$$E] of var int: flow,
                            var int: cost)
```

arciarciarci

balancebalance = output - input

weight

flow

cost

```
1.  predicate path(int: N,
                   int: E,
                   array [int] of int: from,
                   array [int] of int: to,
                   var int: s,
                   var int: t,
                   array [int] of var bool: ns,
                   array [int] of var bool: es)

2.  predicate path(array [int] of $$N: from,
                   array [int] of $$N: to,
                   var $$N: s,
                   var $$N: t,
                   array [$$N] of var bool: ns,
                   array [int] of var bool: es)
```

nsesst

N

E

fromN

toN

s

t

ns

es

nsesst

from

to

s

t

ns

es

```
1.  predicate reachable(int: N,
                        int: E,
                        array [int] of int: from,
                        array [int] of int: to,
                        var int: r,
                        array [int] of var bool: ns,
                        array [int] of var bool: es)
```

```
2.  predicate reachable(array [int] of $$N: from,
                        array [int] of $$N: to,
                        var $$N: r,
                        array [$$N] of var bool: ns,
                        array [int] of var bool: es)
```

nsesr

N

E

fromN

toN

r

ns

es

nsesr

from

to

r

ns

es

```
predicate steiner(int: N,
                  int: E,
                  array [int] of int: from,
                  array [int] of int: to,
                  array [int] of int: w,
                  array [int] of var bool: ns,
                  array [int] of var bool: es,
                  var int: K)
```

esW

N

E

fromN

toN

w

ns

es

K

```
predicate subcircuit(array [$$E] of var $$E: x)
```

xxijjixiii

```
1.  predicate subgraph(int: N,
                       int: E,
                       array [int] of int: from,
                       array [int] of int: to,
                       array [int] of var bool: ns,
                       array [int] of var bool: es)

2.  predicate subgraph(array [$$E] of $$N: from,
                       array [$$E] of $$N: to,
                       array [$$N] of var bool: ns,
                       array [$$E] of var bool: es)
```

nses

N

E

fromN

toN

ns

es

nses

from

to

ns

es

```
1.  predicate tree(int: N,
                   int: E,
                   array [int] of int: from,
```

```
                    array [int] of int: to,
                    var int: r,
                    array [int] of var bool: ns,
                    array [int] of var bool: es)

2.  predicate tree(array [$$E] of $$N: from,
                    array [$$E] of $$N: to,
                    var $$N: r,
                    array [$$N] of var bool: ns,
                    array [$$E] of var bool: es)
```

nsesr

N

E

fromN

toN

r

ns

es

nsesr

from

to

r

ns

es

```
predicate weighted_spanning_tree(int: N,
                                 int: E,
                                 array [int] of int: from,
                                 array [int] of int: to,
                                 array [int] of int: w,
                                 array [int] of var bool: es,
                                 var int: K)
```

esW

N

E

fromN

toN

w

es

K

4.2.2.12 Extensional constraints (table, regular etc.)

```
predicate cost_mdd(array [int] of var int: x,
                   int: N,
                   array [int] of int: level,
                   int: E,
                   array [int] of int: from,
                   array [int] of set of int: label,
                   array [int] of int: cost,
                   array [int] of int: to,
                   var int: totalcost)
```

xtotalcost

N

levellength

E

fromN

label

cost

to

totalcostx

```
predicate cost_regular(array [int] of var int: x,
                       int: Q,
                       int: S,
                       array [int,int] of int: d,
                       int: q0,
                       set of int: F,
                       array [int,int] of int: c,
                       var int: C)
```

xSQSdQSQq0QFQcCx

```
predicate mdd(array [int] of var int: x,
              int: N,
              array [int] of int: level,
```

```
              int: E,
              array [int] of int: from,
              array [int] of set of int: label,
              array [int] of int: to)
```

x

N

levellength

E

fromN

labelx

to

```
predicate mdd_nondet(array [int] of var int: x,
                     int: N,
                     array [int] of int: level,
                     int: E,
                     array [int] of int: from,
                     array [int] of set of int: label,
                     array [int] of int: to)
```

x

N

levellength

E

fromN

labelx

to

```
1.  predicate regular(array [int] of var $$Val: x,
                      array [$$State,$$Val] of opt $$State: d,
                      $$State: q0,
                      set of $$State: F)

2.  predicate regular(array [int] of var int: x,
                      int: Q,
                      int: S,
```

```
                    array [int,int] of int: d,
                    int: q0,
                    set of int: F)

3.  predicate regular(array [int] of var int: x,
                    int: Q,
                    set of int: S,
                    array [int,int] of int: d,
                    int: q0,
                    set of int: F)

4.  predicate regular(array [int] of var int: x, string: r)
```

xdq0FStateVal

xSQSdQSQq0QFQ

xSQSdQSQq0QFQ

xr

“”

“”

“”

“”

“”

“”

“”

“”

“”

“”

“”

“”

“”

```
1.  predicate regular_nfa(array [int] of var $$Val: x,
                        array [$$State,$$Val] of set of $$State: d,
                        $$State: q0,
                        set of $$State: F)

2.  predicate regular_nfa(array [int] of var int: x,
                        int: Q,
```

```
                          int: S,
                          array [int,int] of set of int: d,
                          int: q0,
                          set of int: F)

3.  predicate regular_nfa(array [int] of var int: x,
                          int: Q,
                          set of int: S,
                          array [int,int] of set of int: d,
                          int: q0,
                          set of int: F)
```

xdq0FStateVal

xSQSdQSQq0QFQ

xSQSdQSQq0QFQ

```
1.  predicate table(array [$$E] of var bool: x, array [int,$$E] of bool: t)

2.  predicate table(array [$$E] of var int: x, array [int,$$E] of int: t)

3.  predicate table(array [int] of var opt int: x,
                    array [int,int] of opt int: t)
```

xttt

xtx_it_itt

4.2.2.13 Machine learning constraints

```
predicate neural_net(array [int] of var float: inputs,
                     array [int] of int: input_ids,
                     array [int] of var float: outputs,
                     array [int] of int: output_ids,
                     array [int] of float: bias,
                     array [int] of float: edge_weight,
                     array [int] of int: edge_parent,
                     array [int] of int: first_edge,
                     NEURON_TYPE: neuron_type)
```

inputs

input_ids

outputs

output_ids

bias

edge_weight

edge_parent

first_edge

neuron_type

4.2.2.14 Deprecated constraints

```
predicate at_least(int: n, array [$X] of var $$E: x, $$E: v)
```

nxv

```
predicate at_most(int: n, array [$X] of var $$E: x, $$E: v)
```

nxv

```
predicate exactly(int: n, array [$X] of var $$E: x, $$E: v)
```

nxv

```
predicate global_cardinality_low_up(array [$X] of var int: x,
                                    array [$Y] of int: cover,
                                    array [$Y] of int: lbound,
                                    array [$Y] of int: ubound)
```

icoverilboundiuboundix

```
predicate global_cardinality_low_up_closed(array [$X] of var int: x,
                                           array [$Y] of int: cover,
                                           array [$Y] of int: lbound,
                                           array [$Y] of int: ubound)
```

icoverilboundiuboundix

xcover

4.2.3 FlatZinc builtins

4.2.3.1 Integer FlatZinc builtins

```
predicate array_int_element(var int: b,
                            array [int] of int: as,
                            var int: c)
```

asbc

```
predicate array_int_maximum(var int: m, array [int] of var int: x)
```

mx

```
predicate array_int_minimum(var int: m, array [int] of var int: x)
```

mx

```
predicate array_var_int_element(var int: b,
                                array [int] of var int: as,
                                var int: c)
```

asbc

```
predicate int_abs(var int: a, var int: b)
```

ba

```
predicate int_div(var int: a, var int: b, var int: c)
```

abc

```
predicate int_eq(var int: a, var int: b)
```

ab

```
predicate int_eq_reif(var int: a, var int: b, var bool: r)
```

ab↔r

```
predicate int_le(var int: a, var int: b)
```

ab

```
predicate int_le_reif(var int: a, var int: b, var bool: r)
```

ab↔r

```
predicate int_lin_eq(array [int] of int: as,
                     array [int] of var int: bs,
                     int: c)
```

$= \sum_i [i] * [i]$

```
predicate int_lin_eq_reif(array [int] of int: as,
                          array [int] of var int: bs,
                          int: c,
                          var bool: r)
```

$\leftrightarrow (= \sum_i [i] * [i])$

```
predicate int_lin_le(array [int] of int: as,
                     array [int] of var int: bs,
                     int: c)
```

∑asibsic

```
predicate int_lin_le_reif(array [int] of int: as,
                          array [int] of var int: bs,
                          int: c,
                          var bool: r)
```

r↔∑asibsic

```
predicate int_lin_ne(array [int] of int: as,
                     array [int] of var int: bs,
                     int: c)
```

$\neq \sum_i [i] * [i]$

```
predicate int_lin_ne_reif(array [int] of int: as,
                          array [int] of var int: bs,
                          int: c,
                          var bool: r)
```

$\leftrightarrow (\neq \sum_i [i] * [i])$

```
predicate int_lt(var int: a, var int: b)
```

ab

```
predicate int_lt_reif(var int: a, var int: b, var bool: r)
```

r↔ab

```
predicate int_max(var int: a, var int: b, var int: c)
```

abc

```
predicate int_min(var int: a, var int: b, var int: c)
```

abc

```
predicate int_mod(var int: a, var int: b, var int: c)
```

abc

```
predicate int_ne(var int: a, var int: b)
```

ab

```
predicate int_ne_reif(var int: a, var int: b, var bool: r)
```

r↔ab

```
predicate int_plus(var int: a, var int: b, var int: c)
```

abc

```
predicate int_pow(var int: x, var int: y, var int: z)
```

z1 div pow(x, abs(y)) < 0

```
predicate int_times(var int: a, var int: b, var int: c)
```

abc

```
predicate set_in(var int: x, set of int: S)
```

x∈S

4.2.3.2 Bool FlatZinc builtins

```
predicate array_bool_and(array [int] of var bool: as, var bool: r)
```

$\leftrightarrow \bigwedge_i [i]$

```
predicate array_bool_element(var int: b,
                             array [int] of bool: as,
                             var bool: c)
```

asbc

```
predicate array_bool_xor(array [int] of var bool: as)
```

$\oplus_i [i]$

```
predicate array_var_bool_element(var int: b,
                                 array [int] of var bool: as,
                                 var bool: c)
```

asbc

```
predicate bool2int(var bool: a, var int: b)
```

$\in \{0,1\} \leftrightarrow\ = 1$

```
predicate bool_and(var bool: a, var bool: b, var bool: r)
```

$\leftrightarrow\ \wedge$

```
predicate bool_clause(array [int] of var bool: as,
                      array [int] of var bool: bs)
```

$\bigvee_i [i] \vee \bigvee_j \neg [j]$

```
predicate bool_eq(var bool: a, var bool: b)
```

ab

```
predicate bool_eq_reif(var bool: a, var bool: b, var bool: r)
```

r↔ab

```
predicate bool_le(var bool: a, var bool: b)
```

ab

```
predicate bool_le_reif(var bool: a, var bool: b, var bool: r)
```

r↔ab

```
predicate bool_lin_eq(array [int] of int: as,
                      array [int] of var bool: bs,
                      var int: c)
```

$= \sum_i [i] * [i]$

```
predicate bool_lin_le(array [int] of int: as,
                      array [int] of var bool: bs,
                      int: c)
```

$\sum_i [i] * [i] \leq$

```
predicate bool_lt(var bool: a, var bool: b)
```

ab

```
predicate bool_lt_reif(var bool: a, var bool: b, var bool: r)
```

r↔ab

```
predicate bool_not(var bool: a, var bool: b)
```

ab

```
predicate bool_or(var bool: a, var bool: b, var bool: r)
```

$\leftrightarrow \vee$

```
1.  predicate bool_xor(var bool: a, var bool: b, var bool: r)

2.  predicate bool_xor(var bool: a, var bool: b)
```

$\leftrightarrow \oplus$

a⊕b

4.2.3.3 Set FlatZinc builtins

```
predicate array_set_element(var int: b,
                            array [int] of set of int: as,
                            var set of int: c)
```

asbc

```
predicate array_var_set_element(var int: b,
                                array [int] of var set of int: as,
                                var set of int: c)
```

asbc

```
predicate set_card(var set of int: S, var int: x)
```

xS

```
predicate set_diff(var set of int: x,
                   var set of int: y,
                   var set of int: r)
```

rx\y

```
predicate set_eq(var set of int: x, var set of int: y)
```

xy

```
predicate set_eq_reif(var set of int: x,
                      var set of int: y,
                      var bool: r)
```

r↔xy

```
predicate set_in(var int: x, var set of int: S)
```

x∈S

```
predicate set_in_reif(var int: x, set of int: S, var bool: r)
predicate set_in_reif(var int: x, var set of int: S, var bool: r)
```

$\leftrightarrow$ ($\in$)

```
predicate set_intersect(var set of int: x,
                        var set of int: y,
                        var set of int: r)
```

rx∩y

```
predicate set_le(var set of int: x, var set of int: y)
```

xy

```
predicate set_le_reif(var set of int: x,
                      var set of int: y,
                      var bool: r)
```

$\leftrightarrow$ ($\leq$)

```
predicate set_lt(var set of int: x, var set of int: y)
```

xy

```
predicate set_lt_reif(var set of int: x,
                      var set of int: y,
                      var bool: r)
```

$\leftrightarrow$ ($<$)

```
predicate set_ne(var set of int: x, var set of int: y)
```

xy

```
predicate set_ne_reif(var set of int: x,
                      var set of int: y,
                      var bool: r)
```

r↔xy

```
predicate set_subset(var set of int: x, var set of int: y)
```

x⊆y

```
predicate set_subset_reif(var set of int: x,
                          var set of int: y,
                          var bool: r)
```

↔ (⊆)

```
predicate set_superset(var set of int: x, var set of int: y)
```

x⊇y

```
predicate set_superset_reif(var set of int: x,
                            var set of int: y,
                            var bool: r)
```

↔ (⊆)

```
predicate set_symdiff(var set of int: x,
                      var set of int: y,
                      var set of int: r)
```

rxy

```
predicate set_union(var set of int: x,
                    var set of int: y,
                    var set of int: r)
```

rx∪y

4.2.3.4 Float FlatZinc builtins

```
predicate array_float_element(var int: b,
                              array [int] of float: as,
                              var float: c)
```

asbc

```
predicate array_float_maximum(var int: m, array [int] of var int: x)
```

mx

```
predicate array_float_minimum(var int: m, array [int] of var int: x)
```

mx

```
predicate array_var_float_element(var int: b,
                                  array [int] of var float: as,
                                  var float: c)
```

asbc

```
predicate float_abs(var float: a, var float: b)
```

ba

```
predicate float_acos(var float: a, var float: b)
```

ba

```
predicate float_acosh(var float: a, var float: b)
```

ba

```
predicate float_asin(var float: a, var float: b)
```

ba

```
predicate float_asinh(var float: a, var float: b)
```

ba

```
predicate float_atan(var float: a, var float: b)
```

ba

```
predicate float_atanh(var float: a, var float: b)
```

ba

```
predicate float_ceil(var float: x, var int: y)
```

$= \lceil \rceil$

```
predicate float_cos(var float: a, var float: b)
```

ba

```
predicate float_cosh(var float: a, var float: b)
```

ba

```
predicate float_div(var float: a, var float: b, var float: c)
```

abc

```
predicate float_dom(var float: x, array [int] of float: as)
```

xasasiasiin

```
predicate float_eq(var float: a, var float: b)
```

ab

```
predicate float_eq_reif(var float: a, var float: b, var bool: r)
```

r↔ab

```
predicate float_exp(var float: a, var float: b)
```

ba

```
predicate float_floor(var float: x, var int: y)
```

$= \lfloor \rfloor$

```
predicate float_in(var float: a, float: b, float: c)
```

$\in [,]$

```
predicate float_in_reif(var float: a, float: b, float: c, var bool: r)
```

r↔ $\in [,]$

```
predicate float_le(var float: a, var float: b)
```

ab

```
predicate float_le_reif(var float: a, var float: b, var bool: r)
```

r↔ab

```
predicate float_lin_eq(array [int] of float: as,
                       array [int] of var float: bs,
                       float: c)
```

$= \sum_i [i] * [i]$

```
predicate float_lin_eq_reif(array [int] of float: as,
                            array [int] of var float: bs,
```

```
                              float: c,
                              var bool: r)
```

$\leftrightarrow (= \sum_i [i] * [i])$

```
predicate float_lin_le(array [int] of float: as,
                       array [int] of var float: bs,
                       float: c)
```

$\sum_i [i] * [i] \leq$

```
predicate float_lin_le_reif(array [int] of float: as,
                            array [int] of var float: bs,
                            float: c,
                            var bool: r)
```

$\leftrightarrow (\sum_i [i] * [i] \leq)$

```
predicate float_lin_lt(array [int] of float: as,
                       array [int] of var float: bs,
                       float: c)
```

$\sum_i [i] * [i] <$

```
predicate float_lin_lt_reif(array [int] of float: as,
                            array [int] of var float: bs,
                            float: c,
                            var bool: r)
```

$\leftrightarrow (\sum_i [i] * [i] <)$

```
predicate float_lin_ne(array [int] of float: as,
                       array [int] of var float: bs,
                       float: c)
```

$\neq \sum_i [i] * [i]$

```
predicate float_lin_ne_reif(array [int] of float: as,
                            array [int] of var float: bs,
                            float: c,
                            var bool: r)
```

$\leftrightarrow$ ($\neq \sum_i [i] * [i]$)

```
predicate float_ln(var float: a, var float: b)
```

ba

```
predicate float_log10(var float: a, var float: b)
```

ba

```
predicate float_log2(var float: a, var float: b)
```

ba

```
predicate float_lt(var float: a, var float: b)
```

ab

```
predicate float_lt_reif(var float: a, var float: b, var bool: r)
```

r$\leftrightarrow$ab

```
predicate float_max(var float: a, var float: b, var float: c)
```

abc

```
predicate float_min(var float: a, var float: b, var float: c)
```

abc

```
predicate float_ne(var float: a, var float: b)
```

ab

```
predicate float_ne_reif(var float: a, var float: b, var bool: r)
```

r↔ab

```
predicate float_plus(var float: a, var float: b, var float: c)
```

abc

```
predicate float_pow(var float: x, var float: y, var float: z)
```

z

```
predicate float_round(var float: x, var int: y)
```

yx

```
predicate float_sin(var float: a, var float: b)
```

ba

```
predicate float_sinh(var float: a, var float: b)
```

ba

```
predicate float_sqrt(var float: a, var float: b)
```

$= \sqrt{}$

```
predicate float_tan(var float: a, var float: b)
```

ba

```
predicate float_tanh(var float: a, var float: b)
```

ba

```
predicate float_times(var float: a, var float: b, var float: c)
```

abc

```
predicate int2float(var int: x, var float: y)
```

yx

4.2.3.5 FlatZinc builtins added in MiniZinc 2.0.0.

```
predicate array_float_maximum(var float: m,
                              array [int] of var float: x)
```

mx

```
predicate array_float_minimum(var float: m,
                              array [int] of var float: x)
```

mx

```
predicate array_int_maximum(var int: m, array [int] of var int: x)
```

mx

```
predicate array_int_minimum(var int: m, array [int] of var int: x)
```

mx

```
predicate bool_clause_reif(array [int] of var bool: as,
                           array [int] of var bool: bs,
                           var bool: b)
```

$\leftrightarrow \bigvee_i [i] \vee \bigvee_j \neg[j]$

4.2.3.6 FlatZinc builtins added in MiniZinc 2.0.2.

```
predicate array_var_bool_element_nonshifted(var int: idx,
                                            array [int] of var bool: x,
                                            var bool: c)
```

xidxc

```
predicate array_var_float_element_nonshifted(var int: idx,
                                             array [int] of var float: x,
                                             var float: c)
```

xidxc

```
predicate array_var_int_element_nonshifted(var int: idx,
                                           array [int] of var int: x,
                                           var int: c)
```

xidxc

```
predicate array_var_set_element_nonshifted(var int: idx,
                                           array [int] of var set of int: x,
                                           var set of int: c)
```

xidxc

4.2.3.7 FlatZinc builtins added in MiniZinc 2.1.0.

```
predicate float_dom(var float: x, array [int] of float: as)
```

xas

```
predicate float_in(var float: x, float: a, float: b)
```

axb

4.2.3.8 FlatZinc builtins added in MiniZinc 2.1.1.

```
function var $$E: max(var set of $$E: s)
```

s

```
function var $$E: min(var set of $$E: s)
```

s

4.2.3.9 FlatZinc builtins added in MiniZinc 2.2.1.

```
predicate int_pow_fixed(var int: x, int: y, var int: z)
```

z1 div pow(x, abs(y)) < 0

4.2.3.10 FlatZinc builtins added in MiniZinc 2.3.3.

```
predicate float_set_in(var float: x, set of float: S)
```

x∈S

4.2.3.11 FlatZinc builtins added in MiniZinc 2.5.2.

```
predicate array_var_bool_element2d_nonshifted(var int: idx1,
                                              var int: idx2,
                                              array [int,int] of var bool: x,
                                              var bool: c)
```

xidx1idx2c

```
predicate array_var_float_element2d_nonshifted(var int: idx1,
                                               var int: idx2,
                                               array [int,int] of var float: x,
                                               var float: c)
```

xidx1idx2c

```
predicate array_var_int_element2d_nonshifted(var int: idx1,
                                             var int: idx2,
                                             array [int,int] of var int: x,
                                             var int: c)
```

xidx1idx2c

```
predicate array_var_set_element2d_nonshifted(var int: idx1,
                                             var int: idx2,
                                             array [int,int] of var set of int: x,
                                             var set of int: c)
```

xidx1idx2c

4.2.3.12 Deprecated FlatZinc builtins

```
predicate array_bool_or(array [int] of var bool: as, var bool: r)
```

$\leftrightarrow \bigvee_i [i]$

4.2.4 Additional declarations for Gecode

“”

4.2.4.1 Additional Gecode search annotations

```
annotation action_max
```

```
annotation action_min
```

```
annotation action_size_max
```

```
annotation action_size_min
```

```
annotation afc_max
```

```
annotation afc_min
```

```
annotation afc_size_max
```

```
annotation afc_size_min
```

```
annotation bool_default_search(ann: varsel, ann: valsel)
```

varselvalsel

```
annotation float_default_search(ann: varsel, ann: valsel)
```

varselvalsel

```
annotation int_default_search(ann: varsel, ann: valsel)
```

varselvalsel

```
annotation random
```

```
1.  annotation relax_and_reconstruct(array [int] of var int: x,
                                     int: percentage)

2.  annotation relax_and_reconstruct(array [int] of var int: x,
                                     int: percentage,
                                     array [int] of int: y)
```

xpercentage

xpercentagey

```
annotation set_default_search(ann: varsel, ann: valsel)
```

varselvalsel

4.2.4.2 Additional Gecode constraints

```
1.  predicate among_seq(array [int] of var int: x,
                        set of int: S,
                        int: l,
                        int: m,
                        int: n)

2.  predicate among_seq(array [int] of var bool: x,
                        bool: b,
                        int: l,
                        int: m,
                        int: n)
```

xlmnS

xlmnb

```
predicate circuit_cost(array [int] of int: c,
                       array [int] of var int: x,
                       var int: z)
```

xxijjizc

```
predicate circuit_cost_array(array [int] of int: c,
                             array [int] of var int: x,
                             array [int] of var int: y,
                             var int: z)
```

xxijjizcyx

```
predicate gecode_array_set_element_intersect(var set of int: x,
                                             array [int] of var set of int: y,
                                             var set of int: z)
```

zyx$i \in \ \leftrightarrow \forall j \in \ : (i \in [j])$

```
predicate gecode_array_set_element_intersect_in(var set of int: x,
                                                array [int] of var set of int: y,
                                                var set of int: z,
                                                set of int: u)
```

zuzyx$i \in \ \leftrightarrow \forall j \in \ : (i \in [j])$

```
predicate gecode_array_set_element_partition(var set of int: x,
                                             array [int] of var set of int: y,
                                             var set of int: z)
```

zyx$i \in \ \leftrightarrow \exists j \in \ : (i \in [j]) i \in \ \wedge j \in \ \wedge i \neq j \rightarrow [i] \cap [j] = \emptyset$

4.2.5 Additional declarations for Chuffed

“”

4.2.5.1 Additional Chuffed search annotations

```
annotation assume(array [int] of var bool)
```

```
annotation largest_smallest
```

```
annotation priority_search(array [int] of var int: x,
                           array [int] of ann: search,
                           ann: select,
                           ann: explore)
```

searchxselectexplore

```
annotation random_order
```

```
annotation smallest_largest
```

4.2.5.2 Other declarations

```
predicate chuffed_minimal_spanning_tree(int: N,
                                        int: E,
                                        array [int] of int: from,
                                        array [int] of int: to,
                                        array [int] of int: w,
                                        array [int] of var bool: vs,
                                        array [int] of var bool: es,
                                        var int: K)
```

vsesK

N

E

fromN

toN

w

vs

es

K

4.2.6 MiniZincIDE tools

4.2.6.1 MiniZincIDE solution visualisation tools

output

```
var 1..10: x;
output vis_line(x, "x-value");
```

vis_server

```
int: n;
var 1..10: x;
output :: vis_server("my-vis.html", (n: n)) (x: x);
```

my-vis.html

```
1.  annotation vis_bar(array [$$D] of var int: x)

2.  annotation vis_bar(array [$$D] of var float: x)

3.  annotation vis_bar(array [$$D] of var int: x,
                       array [$$D] of string: data_labels)

4.  annotation vis_bar(array [$$D] of var float: x,
                       array [$$D] of string: data_labels)

5.  annotation vis_bar(array [$$S,$$D] of var int: x)

6.  annotation vis_bar(array [$$S,$$D] of var float: x)

7.  annotation vis_bar(array [$$S,$$D] of var int: x,
                       array [$$S] of string: series_labels,
                       array [$$D] of string: data_labels)

8.  annotation vis_bar(array [$$S,$$D] of var float: x,
                       array [$$S] of string: series_labels,
                       array [$$D] of string: data_labels)
```

x

xdata_labels

xijji

xijjiseries_labelsdata_labels

```
1.  annotation vis_column(array [$$D] of var int: x)

2.  annotation vis_column(array [$$D] of var float: x)

3.  annotation vis_column(array [$$D] of var int: x,
                          array [$$D] of string: data_labels)

4.  annotation vis_column(array [$$D] of var float: x,
                          array [$$D] of string: data_labels)

5.  annotation vis_column(array [$$S,$$D] of var int: x)

6.  annotation vis_column(array [$$S,$$D] of var float: x)
```

```
7.  annotation vis_column(array [$$S,$$D] of var int: x,
                          array [$$S] of string: series_labels,
                          array [$$D] of string: data_labels)

8.  annotation vis_column(array [$$S,$$D] of var float: x,
                          array [$$S] of string: series_labels,
                          array [$$D] of string: data_labels)
```

x

xdata_labels

xijji

xijjiseries_labelsdata_labels

```
1.  annotation vis_digraph(array [$$E] of $$N: from,
                           array [$$E] of $$N: to,
                           array [$$E] of string: edge_labels,
                           array [$$N] of var bool: ns,
                           array [$$E] of var bool: es)

2.  annotation vis_digraph(array [$$E] of $$N: from,
                           array [$$E] of $$N: to,
                           array [$$N] of var bool: ns,
                           array [$$E] of var bool: es)

3.  annotation vis_digraph(array [$$N] of string: node_labels,
                           array [$$E] of $$N: from,
                           array [$$E] of $$N: to,
                           array [$$E] of string: edge_labels,
                           array [$$N] of var bool: ns,
                           array [$$E] of var bool: es)
```

nsesns

from

to

edge_labels

ns

es

nsesnses

from

to

ns

es

nses

node_labels

from

to

edge_labels

ns

es

```
1.  annotation vis_digraph_highlight(array [$$E] of $$N: from,
                                     array [$$E] of $$N: to,
                                     array [$$E] of string: edge_labels,
                                     array [$$N] of var bool: ns,
                                     array [$$E] of var bool: es)

2.  annotation vis_digraph_highlight(array [$$E] of $$N: from,
                                     array [$$E] of $$N: to,
                                     array [$$N] of var bool: ns,
                                     array [$$E] of var bool: es)

3.  annotation vis_digraph_highlight(array [$$N] of string: node_labels,
                                     array [$$E] of $$N: from,
                                     array [$$E] of $$N: to,
                                     array [$$E] of string: edge_labels,
                                     array [$$N] of var bool: ns,
                                     array [$$E] of var bool: es)
```

nsesns

from

to

edge_labels

ns

es

nsesnses

from

to

ns

es

nses

node_labels

from

to

edge_labels

ns

es

```
1.  annotation vis_gantt(array [$$E] of var int: start,
                         array [$$E] of var int: dur)

2.  annotation vis_gantt(array [$$E] of var int: start,
                         array [$$E] of var int: dur,
                         array [$$E] of string: labels)

3.  annotation vis_gantt(array [$$E] of var int: start,
                         array [$$E] of var int: dur,
                         array [$$E] of string: labels,
                         array [$$E] of string: colors)
```

startdurstart

startdurlabels

startdurlabelscolors

```
1.  annotation vis_geost_2d(array [$$E] of int: rect_x,
                            array [$$E] of int: rect_y,
                            array [$$E] of int: rect_dx,
                            array [$$E] of int: rect_dy,
                            array [$$K] of set of $$E: shape,
                            array [$$T] of var int: x,
                            array [$$T] of var int: y,
                            array [$$T] of var $$K: kind)

2.  annotation vis_geost_2d(array [$$E,1..2] of int: rect_size,
                            array [$$E,1..2] of int: rect_offset,
                            array [$$K] of set of $$E: shape,
                            array [$$T,1..2] of var int: x,
                            array [$$T] of var $$K: kind)

3.  annotation vis_geost_2d(array [$$E] of tuple(int, int): rect_size,
```

```
                            array [$$E] of tuple(int, int): rect_offset,
                            array [$$K] of set of $$E: shape,
                            array [$$T] of var tuple(var int, var int): x,
                            array [$$T] of var $$K: kind)
```

rect_x

rect_y

rect_dx

rect_dy

shape

x

y

kind

rect_size

rect_offset

shape

x

kind

```
1.  annotation vis_graph(array [$$E] of $$N: from,
                         array [$$E] of $$N: to,
                         array [$$E] of string: edge_labels,
                         array [$$N] of var bool: ns,
                         array [$$E] of var bool: es)

2.  annotation vis_graph(array [$$E] of $$N: from,
                         array [$$E] of $$N: to,
                         array [$$N] of var bool: ns,
                         array [$$E] of var bool: es)

3.  annotation vis_graph(array [$$N] of string: node_labels,
                         array [$$E] of $$N: from,
                         array [$$E] of $$N: to,
                         array [$$E] of string: edge_labels,
                         array [$$N] of var bool: ns,
                         array [$$E] of var bool: es)
```

nsesns

from

to

edge_labels

ns

es

nsesnses

from

to

ns

es

nses

node_labels

from

to

edge_labels

ns

es

```
1.  annotation vis_graph_highlight(array [$$E] of $$N: from,
                                   array [$$E] of $$N: to,
                                   array [$$E] of string: edge_labels,
                                   array [$$N] of var bool: ns,
                                   array [$$E] of var bool: es)

2.  annotation vis_graph_highlight(array [$$E] of $$N: from,
                                   array [$$E] of $$N: to,
                                   array [$$N] of var bool: ns,
                                   array [$$E] of var bool: es)

3.  annotation vis_graph_highlight(array [$$N] of string: node_labels,
                                   array [$$E] of $$N: from,
                                   array [$$E] of $$N: to,
                                   array [$$E] of string: edge_labels,
                                   array [$$N] of var bool: ns,
                                   array [$$E] of var bool: es)
```

nsesns

from

```
        to
        edge_labels
        ns
        es
    nsesnses

        from
        to
        ns
        es
    nses

        node_labels
        from
        to
        edge_labels
        ns
        es
```

```
1.  annotation vis_line(var int: x, string: label)

2.  annotation vis_line(var float: x, string: label)

3.  annotation vis_line(array [$$E] of var int: x)

4.  annotation vis_line(array [$$E] of var float: x)

5.  annotation vis_line(array [$$E] of var int: x,
                        array [$$E] of string: series_labels)

6.  annotation vis_line(array [$$E] of var float: x,
                        array [$$E] of string: series_labels)

7.  annotation vis_line(array [$$E] of var int: x,
                        array [$$E] of var int: y,
                        string: x_label,
                        string: y_label)

8.  annotation vis_line(array [$$E] of var float: x,
                        array [$$E] of var float: y,
                        string: x_label,
                        string: y_label)
```

xlabel

x

xseries_labels

xyx_labely_label

```
annotation vis_scatter(array [$$E] of var int: x,
                       array [$$E] of var int: y,
                       string: x_label,
                       string: y_label)
annotation vis_scatter(array [$$E] of var float: x,
                       array [$$E] of var float: y,
                       string: x_label,
                       string: y_label)
```

xyx_labely_label

```
1.  annotation vis_scatter_cumulative(var int: x,
                                      var int: y,
                                      string: x_label,
                                      string: y_label,
                                      string: series_label)

2.  annotation vis_scatter_cumulative(var float: x,
                                      var float: y,
                                      string: x_label,
                                      string: y_label,
                                      string: series_label)

3.  annotation vis_scatter_cumulative(array [$$E] of var int: x,
                                      array [$$E] of var int: y,
                                      string: x_label,
                                      string: y_label)

4.  annotation vis_scatter_cumulative(array [$$E] of var float: x,
                                      array [$$E] of var float: y,
                                      string: x_label,
                                      string: y_label)

5.  annotation vis_scatter_cumulative(array [$$E] of var int: x,
                                      array [$$E] of var int: y,
                                      string: x_label,
                                      string: y_label,
                                      array [$$E] of string: series_labels)

6.  annotation vis_scatter_cumulative(array [$$E] of var float: x,
                                      array [$$E] of var float: y,
```

```
                                    string: x_label,
                                    string: y_label,
                                    array [$$E] of string: series_labels)
```

xy

x

y

x_label

y_label

series_label

xyixy

x

y

x_label

y_label

xyi

x

y

x_label

y_label

series_labels

```
1.  annotation vis_server(string: file)
2.  annotation vis_server(string: file, opt $T: user_data)
3.  annotation vis_server(string: file, array [$X] of opt $T: user_data)
```

file

fileuser_dataMiniZincIDE.getUserData()

4.2.7 Experimental Features

4.2.7.1 On Restart

Constants

```
set of int: STATUS
```

Functions and Predicates

```
predicate basic_lns(var bool: nbh)
```

‘’

```
test complete()
```

‘’

```
predicate last_val(var bool: x)
function var float: last_val(var float: x)
function var $$E: last_val(var $$E: x)
function var opt $$E: last_val(var opt $$E: x)
function var set of $$E: last_val(var set of $$E: x)
```

```
predicate round_robin(array [int] of var bool: nbhs)
```

```
predicate sol(var bool: x)
function var float: sol(var float: x)
function var $$E: sol(var $$E: x)
function var opt $$E: sol(var opt $$E: x)
function var set of $$E: sol(var set of $$E: x)
function array [$$E] of var bool: sol(array [$$E] of var bool: x)
function array [$$E] of var float: sol(array [$$E] of var float: x)
function array [$$E] of var $$F: sol(array [$$E] of var $$F: x)
```

```
1.  function var $$E: uniform_on_restart($$E: low, $$E: high)
2.  function var $$E: uniform_on_restart(set of $$E: S)
3.  function var float: uniform_on_restart(float: low, float: high)
```

low high

S

low high

Annotations

```
annotation on_restart(string: pred)
```

CHAPTER 4.3

Interfacing Solvers to Flatzinc

节节节节

’ 节

’ minizinc节

$SOLVERNAME$PREFIX$PREFIX/bin$PREFIX/share/minizinc/$SOLVERNAME/$PREFIX/share/minizinc/solvers/$
/usr/local

4.3.1 Specification of FlatZinc

$r_1 r_2 x_1, x_2, \ldots, x_k, i, j, k y_1, y_2, \ldots, y_k, y_i$

4.3.1.1 Comments

%

4.3.1.2 Types

var

Parameter types

<par-type>节

Type	Values
`bool`	`truefalse`
`float`	
`int`	
`set of int`	
`array [1..`n`] of bool`	
`array [1..`n`] of float`	
`array [1..`n`] of int`	
`array [1..`n`] of set of int`	

Type	Values
$r_a..r_b$	
$x_a..x_b$	
$\{x_a, x_b, \ldots, x_k\}$	
`set of`$x_a..x_b$	
`set of`$\{x_a, x_b, \ldots, x_k\}$	
`array [1..`n`] of`$r_a..r_b$	
`array [1..`n`] of`$x_a..x_b$	
`array [1..`n`] of set of`$x_a..x_b$	
`array [1..`n`] of set of`$\{x_a, x_b, \ldots, x_k\}$	

$x_a..x_b \{x | x_a \leq x \leq x_b\}$

`int1..`n

Variable types

<basic-var-type><array-var-type>节

Variable type
`var bool`
`var float`
`var`$r_a..r_b$
`var int`
`var`$x_a..x_b$
`var`$\{x_a, x_b, \ldots, x_k\}$
`var set of`$x_a..x_b$
`var set of` $\{x_a, x_b, \ldots, x_k\}$
`array [1..`n`] of var bool`
`array [1..`n`] of var float`
`array [1..`n`] of var`$r_a..r_b$
`array [1..`n`] of var int`
`array [1..`n`] of var`$x_a..x_b$
`array [1..`n`] of var set of`$x_a..x_b$
`array [1..`n`] of var set of` $\{x_a, x_b, \ldots, x_k\}$

Variable type
var set of int
array [1..n] of var set of int

int1..n

The string type

```
""                              % The empty string.
"Hello."
"Hello,\nWorld\t\"quoted!\""    % A string with an embedded newline, tab and quotes.
```

4.3.1.3 Values and expressions

<expr>节

booltruefalsefloat2.718-1.03.0e8int-42069set of int{}{2, 3, 5}1..10arrays[][y_a,...,y_k]

y_iv

```
[1, 2, 3]             % Just literals
[x, y, z]             % x, y, and z are variables or parameters.
[x, 3]                % Mix of identifiers and literals
```

节

4.3.1.4 FlatZinc models

fooFoo[A-Za-z][A-Za-z0-9_]

annotationanyarrayboolcaseconstraintdiffdivelseelseifendifenumfalsefloatfunctionifin includeintintersectletlistmaximizeminimizemodnotofsatisfysubsetsupersetoutputparpredicate recordsetsolvestringsymdifftestthentruetupleuniontypevarwherexor

4.3.1.5 Predicate declarations

<predicate-item>节

```
<predicate-item> ::= "predicate" <identifier> "(" [ <pred-param-type> : <identifier>
↪ "," ... ] ")" ";"
```

<identifier>

```
    % m is the median value of {x, y, z}.
    %
predicate median_of_3(var int: x, var int: y, var int: z, var int: m);

    % all_different([x1, .., xn]) iff
    % for all i, j in 1..n: xi != xj.
    %
predicate all_different(array [int] of var int: xs);

    % exactly_one([x1, .., xn]) iff
    % there exists an i in 1..n: xi = true
    % and for all j in 1..n: j != i -> xj = false.
    %
predicate exactly_one(array [int] of var bool: xs);
```

4.3.1.6 Parameter declarations

param_decl节

```
<par-decl-item> ::= <par-type> ":" <var-par-identifier> "=" <par-expr> ";"
```

<par-type><var-par-identifier><par-expr>

```
float: pi = 3.141;
array [1..7] of int: fib = [1, 1, 2, 3, 5, 8, 13];
bool: beer_is_good = true;
```

4.3.1.7 Variable declarations

var_decl节

```
<var-decl-item> ::= <basic-var-type> ":" <var-par-identifier> <annotations> [ "="
↪<basic-expr> ] ";"
                  | <array-var-type> ":" <var-par-identifier> <annotations> "="
↪<array-literal> ";"
```

<basic-var-type><array-var-type><var-par-identifier><annotations><basic-expr><array-literal>

```
var 0..9: digit;
var bool: b;
var set of 1..3: s;
var 0.0..1.0: x;
var int: y :: mip;        % 'mip' annotation: y should be a MIP variable.
array [1..3] of var 1..10: b = [y, 3, digit];
```

4.3.1.8 Constraints

<constraint-item>节

```
<constraint-item> ::= "constraint" <identifier> "(" [ <expr> "," ... ] ")"
↪<annotations> ";"
```

<expr>

```
constraint int_le(0, x);    % 0 <= x
constraint int_lt(x, y);    % x <  y
constraint int_le(y, 10);   % y <= 10
    % 'domain_propagation': use domain consistency for this constraint:
    % 2x + 3y = 10
constraint int_lin_eq([2, 3], [x, y], 10) :: domain_propagation;
```

4.3.1.9 Solve item

<solve-item>节

```
<solve-item> ::= "solve" <annotations> "satisfy" ";"
               | "solve" <annotations> "minimize" <basic-expr> ";"
               | "solve" <annotations> "maximize" <basic-expr> ";"
```

<basic-expr>

```
solve satisfy;      % Find any solution using the default strategy.

solve minimize w;   % Find a solution minimizing w, using the default strategy.

    % First label the variables in xs in the order x[1], x[2], ...
    % trying values in ascending order.
solve :: int_search(xs, input_order, indomain_min, complete)
    satisfy;    % Find any solution.

    % First use first-fail on these variables, splitting domains
    % at each choice point.
solve :: int_search([x, y, z], first_fail, indomain_split, complete)
    maximize x; % Find a solution maximizing x.
```

4.3.1.10 Annotations

::

```
<annotations> ::= [ "::" <annotation> ]*

<annotation> ::= <identifier>
               | <identifier> "(" <ann-expr> "," ... ")"

<ann-expr>   := <basic-ann-expr>
              | "[" [ <basic-ann-expr> "," ... ] "]"

<basic-ann-expr>   := <basic-literal-expr>
                    | <var-par-identifier>
                    | <string-literal>
                    | <annotation>

<string-contents> ::= ([^"\n\] | \[^\n(])*

<string-literal> ::= """ <string-contents> """
```

::

Search annotations

```
seq_search([<searchannotation>, ...])
```

<searchannotation>

```
int_search(<vars>, <varchoiceannotation>, <assignmentannotation>,
↪<strategyannotation>)

bool_search(<vars>, <varchoiceannotation>, <assignmentannotation>,
↪<strategyannotation>)

set_search(<vars>, <varchoiceannotation>, <assignmentannotation>,
↪<strategyannotation>)
```

<vars>

<varchoiceannotation>

input_order	⋆	vars
first_fail	⋆	
anti_first_fail		
smallest		
largest		
occurrence		
most_constrained		
max_regret		
dom_w_deg		

<assignmentannotation>

indomain_min	⋆	’
indomain_max	⋆	’
indomain_middle		’
indomain_median		’
indomain		
indomain_random		’
indomain_split		’
indomain_reverse_split		’
indomain_interval		’ ’

bool_searchindomain_split

<strategyannotation>complete

Output annotations

output_varoutput_array([1..x1..x节

Variable definition annotations

```
var int: x :: is_defined_var;

...

constraint int_plus(y, z, x) :: defines_var(x);
```

defines_varxy+zis_defined_varx “”

Intermediate variables

```
var int: X_INTRODUCED_3 :: var_is_introduced;
```

,

Constraint annotations

bounds	
boundsZ	
boundsR	
boundsD	boundsZ
domain	
value_propagation	
priority(k)	k

4.3.2 Specification of FlatZinc-JSON

节

列表

```
{
  "variables": {
    "b" : { "type" : "int", "domain" : [[0, 3]] },
    "c" : { "type" : "int", "domain" : [[0, 6]] },
    "X_INTRODUCED_0_" : { "type" : "int", "domain" : [[0, 85000]], "defined" : true␣
↪}
  },
  "arrays": {
    "X_INTRODUCED_2_" : { "a": [250, 200] },
    "X_INTRODUCED_6_" : { "a": [75, 150] },
    "X_INTRODUCED_8_" : { "a": [100, 150] }
  },
  "constraints": [
    { "id" : "int_lin_le", "args" : ["X_INTRODUCED_2_", ["b", "c"], 4000]},
    { "id" : "int_lin_le", "args" : ["X_INTRODUCED_6_", ["b", "c"], 2000]},
    { "id" : "int_lin_le", "args" : ["X_INTRODUCED_8_", ["b", "c"], 500]},
    { "id" : "int_lin_eq", "args" : [[400, 450, -1], ["b", "c", "X_INTRODUCED_0_"],␣
↪0],
      "ann" : ["ctx_pos"], "defines" : "X_INTRODUCED_0_"}
  ],
  "output": ["b", "c"],
  "solve": { "method" : "maximize", "objective" : "X_INTRODUCED_0_" },
  "version": "1.0"
}
```

"rhs" “”

```
{1,2,4,6,7}[[1,2],[4,4],[6,7]]

"a"

"id""args""defines"

"output"节

"solve""method""satisfy""minimize""maximize""objective"

"ann"indomain_minint_search"id""args"

{ "string" : "<string contents>" }
```

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "$id": "https://www.minizinc.org/schemas/fznjson",
  "title" : "FlatZincJSON",
  "description" : "A JSON representation of a FlatZinc model",

  "$defs" : {
    "identifier" : { "type" : "string", "pattern" : "[A-Za-z][A-Za-z0-9_]*" },
    "literal" : { "oneOf" : [
      { "type" : "number" },
      { "$ref" : "#/$defs/identifier" },
      { "type" : "boolean" },
      { "type" : "object",
        "properties" : {
          "set" : {
            "type" : "array",
            "items" : {
                        "type" : "array",
                        "items" : [
                          { "type" : "number" },
                          { "type" : "number" }
                        ]
                      }
          }
        },
        "required" : [ "set" ]
      },
      { "type" : "object",
        "properties" : {
          "string" : { "type" : "string" }
        },
        "required" : [ "string" ]
      }
    ]},
    "literals" : { "type" : "array", "items" : { "$ref" : "#/$defs/literal" } },
    "argument" : { "oneOf" : [
      { "$ref" : "#/$defs/literals"},
      { "$ref" : "#/$defs/literal" }
    ]},
    "annotation" : { "oneOf" : [
      { "$ref" : "#/$defs/annotationCall" },
      { "type" : "string" }
    ]},
```

```
    "annotations" : {
      "type" : "array",
      "items" : { "$ref" : "#/$defs/annotation" }
    },
    "annotationArgument" : { "oneOf" : [
      { "$ref" : "#/$defs/annotationLiterals"},
      { "$ref" : "#/$defs/annotationLiteral" }
    ]},
    "annotationCall" : {
      "type" : "object",
      "properties" : {
        "id": { "$ref": "#/$defs/identifier" },
        "args": {
          "type": "array",
          "items": { "$ref" : "#/$defs/annotationArgument" }
        }
      },
      "required": [
        "id",
        "args"
      ]
    },
    "annotationLiterals" : { "type" : "array", "items" : { "$ref" : "#/$defs/
↪annotationLiteral" } },
    "annotationLiteral" : { "oneOf" : [
      { "$ref" : "#/$defs/literal"},
      { "$ref" : "#/$defs/annotationCall" }
    ]},
    "domain" : {
      "type" : "array",
      "items" : {
        "type" : "array",
        "items" : [
          { "type" : "number" },
          { "type" : "number" }
        ]
      }
    }
  },

  "type": "object",
  "properties": {
    "version" : { "type" : "string" },
    "variables": {
      "type": "object",
      "patternProperties": {
        "[A-Za-z][A-Za-z0-9_]*": {
          "type": "object",
          "properties": {
            "type": {
              "enum": [
                "bool",
                "float",
                "int",
                "set of int"
              ]
            },
```

```
            "domain": { "$ref" : "#/$defs/domain" },
            "rhs": { "$ref" : "#/$defs/literal" },
            "introduced": {
              "type": "boolean"
            },
            "defined": {
              "type": "boolean"
            },
            "ann" : { "$ref" : "#/$defs/annotations"}
          },
          "required": [
            "type"
          ]
        }
      }
    },
    "arrays": {
      "type": "object",
      "patternProperties": {
        "[A-Za-z][A-Za-z0-9_]*": {
          "type": "object",
          "properties": {
            "a": {
              "type": "array",
              "items": { "$ref" : "#/$defs/literal" }
            },
            "ann" : { "$ref" : "#/$defs/annotations"},
            "introduced": {
              "type": "boolean"
            },
            "defined": {
              "type": "boolean"
            }
          },
          "required": [
            "a"
          ]
        }
      }
    },
    "constraints": {
      "type": "array",
      "items": {
        "type" : "object",
        "properties" : {
          "id": { "$ref": "#/$defs/identifier" },
          "args": {
            "type": "array",
            "items": { "$ref" : "#/$defs/argument" }
          },
          "ann" : { "$ref" : "#/$defs/annotations"},
          "defines": { "$ref": "#/$defs/identifier" }
        },
        "required": [
          "id",
          "args"
        ]
```

```
      }
    },
    "output": {
      "type": "array",
      "items": { "$ref" : "#/$defs/identifier" }
    },
    "solve": {
      "type": "object",
      "properties": {
        "method": {
          "enum": [ "satisfy", "minimize", "maximize" ]
        },
        "objective": { "$ref" : "#/$defs/literal" },
        "ann" : { "$ref" : "#/$defs/annotations"}
    },
      "required": [
        "method"
      ]
    }
  },
  "required": [
    "version",
    "variables",
    "arrays",
    "output",
    "constraints",
    "solve"
  ]
}
```

4.3.3 Output

4.3.3.1 Solution output

output_varoutput_array"output"

```
var 1..10: x :: output_var;
var 1..10: y;       % y is not output.
array [1..4] of var int: zs :: output_array([1..4]);
```

```
<var-par-identifier> = <basic-literal-expr> ;
```

```
<var-par-identifier> = [<y1>, <y2>, ... <yk>];
```

<y1>, <y2>, ... <yk>

```
output_arrayoutput_arrayarrayNdNoutput_arrayarrayNd



----------

-a节

==========

=====UNSATISFIABLE=====

=====UNBOUNDED=====

=====UNKNOWN=====
```

```
var 1..3: x :: output_var;
solve satisfy
```

```
x = 1;
----------
```

```
array [1..2] of var 1..3: xs :: output_array([1..2]);
constraint int_lt(xs[1], xs[2]);    % x[1] < x[2].
solve satisfy
```

```
xs = array1d(1..2, [1, 2]);
----------
xs = array1d(1..2, [1, 3]);
----------
xs = array1d(1..2, [2, 3]);
----------
==========
```

```
var 1..10: x :: output_var;
solve maximize x;
```

```
x = 10;
----------
==========
```

```
var 1..10: x :: output_var;
solve maximize x;
```

```
x = 1;
----------
x = 2;
----------
x = 3;
----------
```

==========

```
var 1..3: x :: output_var;
var 4..6: y :: output_var;
constraint int_lt(y, x);    % y < x.
solve satisfy;
```

```
=====UNSATISFIABLE=====
```

4.3.3.2 Statistics output

‘_____’

```
%%%mzn-stat: <name>=<value>
```

```
%%%mzn-stat-end
```

```
%%%mzn-stat: objective=1e+308
%%%mzn-stat: objectiveBound=0
%%%mzn-stat: nodes=0
```

```
%%%mzn-stat: solveTime=2.3567
%%%mzn-stat-end

(no feasible solution found yet but something can be printed...)

%%%mzn-stat: objective=12345
%%%mzn-stat: objectiveBound=122
%%%mzn-stat: nodes=35
%%%mzn-stat: solveTime=78.5799
%%%mzn-stat-end

(the corresponding feasible solution with value 12345 goes here
   or before its statistics but above the separator)
----------               (<- the solution separator)

%%%mzn-stat: objective=379
%%%mzn-stat: objectiveBound=379
%%%mzn-stat: nodes=4725
%%%mzn-stat: solveTime=178.5799
%%%mzn-stat-end

(the corr. optimal solution with value 379 goes here)
----------
==========               (<- the 'search complete' marker)

%%%mzn-stat: objective=379      (<- this is the concluding output)
%%%mzn-stat: objectiveBound=379
%%%mzn-stat: nodes=13456
%%%mzn-stat: solveTime=2378.5799
%%%mzn-stat-end
```

<name><value>

Name	Type	Explanation
nodes		
openNodes		
objective		
objectiveBound		
failures		
restarts		
variables		
intVariables		
boolVariables		
floatVariables		
setVariables		
propagators		
propagations		
peakDepth		
nogoods		
backjumps		
peakMem		
initTime		
solveTime		

4.3.3.3 Error and warning output

4.3.4 Solver-specific Libraries

share/minizinc/gecodeshare/minizinc/chuffed

’ 节

4.3.4.1 Standard predicates

FlatZinc builtins节

redefinitions.mzn’ abort

redefinitions.mzn

```
% Redefine float_sinh function in terms of exp
predicate float_sinh(var float: a, var float: b) =
    b == (exp(a)-exp(-a))/2.0;

% Mark float_tanh as unsupported
predicate float_tanh(var float: a, var float: b) =
    abort("The builtin float_tanh is not supported by this solver.");
```

节’ redefinitions-2.0.mzn

redefinitions-2.0.mzn

```
predicate bool_clause_reif(array[int] of var bool: as,
                           array[int] of var bool: bs,
                           var bool: b);
predicate array_int_maximum(var int: m, array[int] of var int: x);
predicate array_float_maximum(var float: m, array[int] of var float: x);
predicate array_int_minimum(var int: m, array[int] of var int: x);
predicate array_float_minimum(var float: m, array[int] of var float: x);
```

4.3.4.2 Solver-specific predicates

’

all_differentfzn_all_different_int.mzn

```
predicate fzn_all_different_int(array[int] of var int: x) =
  forall(i,j in index_set(x) where i < j) ( x[i] != x[j] );
```

’ fzn_all_different_int.mzn

```
predicate optisolve_alldifferent(array[int] of var int: x);

predicate fzn_all_different_int(array[int] of var int: x) =
  optisolve_alldifferent(x);
```

all_differentoptisolve_alldifferent

bin_packing.mznbin_packingbin_packingbin_packing.mznfzn_bin_packingfzn_bin_packing.mzn bin_packingbin_packing.mznfzn_bin_packing.mzn

4.3.4.3 Reified and half-reified predicates

var bool: b = all_different(x);bx

_reifvar bool

```
var bool: b;
constraint all_different_reif(x, b);
```

_reiflet

fzn_<constraintname>_reif.mzn

pred-context_imp_reifall_different

constraint y=0 \/ all_different(x)

```
var bool: X_INTRODUCED_1;
var bool: X_INTRODUCED_2;
constraint int_eq_imp(y,0,X_INTRODUCED_1);
constraint all_different_imp(x, X_INTRODUCED_2);
constraint bool_clause([X_INTRODUCED_1,X_INTRODUCED_2]);
```

_imp

4.3.5 Command-Line Interface and Standard Options

minizinc

```
$ <executable-name> [options] model.fzn
```

<executable-name>

-a

-n <i>
i

-i
-a

-f
“”

-s
节

-v
%

-p <i>
i

-r <i>

i

-t <ms>

ms

4.3.6 Solver Configuration Files

.dzn’ 节

.msc

minizinc/solvers//usr/share/minizinc/solversProgram Files\MiniZinc IDE (bundled)

$HOME/.minizinc/solvers

MZN_SOLVER_PATH:;

mzn_solver_path节

节

minizinc--config-dirs

.dzn.dzn

```
{
  "name" : "My Solver",
  "version": "1.0",
  "id": "org.myorg.my_solver",
  "executable": "fzn-mysolver"
}
```

.dzn

```
name = "My Solver";
version = "1.0";
id = "org.myorg.my_solver";
executable = "fzn-mysolver";
```

nameversionidexecutable

nameminizinc --solvers

version

id “”

executable

mznlib""-G<solverlib>-Ggecode

tags--solver’

"cp"

"mip"

"float"

"api"

stdFlags-a-n-i-s-v-p-r-f-t

extraFlags"--special-algorithm""which special algorithm to use""int""bool""float"
"string""opt"

"int:n:m"nm

"float:n:m"nm

"bool:onstring:offstring"onstringoffstring["-interrupt","whether to catch Ctrl-C","bool
-interrupt true-interrupt false"bool"

"opt:first option:second option:...:last option"

inputTypeFZNMZNFZNNLJSON

supportsMznfalseinputType

supportsFzntrueinputType

supportsNLfalseinputType

needsSolns2Outtrue

needsMznExecutablefalsemzn-executable

needsStdlibDirfalsestdlib-dir

isGUIApplicationfalse

4.3.7 Grammar

节节

```
% A FlatZinc model
<model> ::=
  [ <predicate-item> ]*
  [ <par-decl-item> ]*
  [ <var-decl-item> ]*
  [ <constraint-item> ]*
  <solve-item>

% Predicate items
<predicate-item> ::= "predicate" <identifier> "(" [ <pred-param-type> : <identifier>
↪ "," ... ] ")" ";"

% Identifiers
<identifier> ::= [A-Za-z][A-Za-z0-9_]*

<basic-par-type> ::= "bool"
                   | "int"
                   | "float"
                   | "set of int"

<par-type> ::= <basic-par-type>
             | "array" "[" <index-set> "]" "of" <basic-par-type>
```

```
<basic-var-type> ::= "var" <basic-par-type>
                   | "var" <int-literal> ".." <int-literal>
                   | "var" "{" <int-literal> "," ... "}"
                   | "var" <float-literal> ".." <float-literal>
                   | "var" "set" "of" <int-literal> ".." <int-literal>
                   | "var" "set" "of" "{" [ <int-literal> "," ... ] "}"

<array-var-type> ::= "array" "[" <index-set> "]" "of" <basic-var-type>

<index-set> ::= "1" ".." <int-literal>

<basic-pred-param-type> ::= <basic-par-type>
                          | <basic-var-type>
                          | <int-literal> ".." <int-literal>
                          | <float-literal> ".." <float-literal>
                          | "{" <int-literal> "," ... "}"
                          | "set" "of" <int-literal> .. <int-literal>
                          | "set" "of" "{" [  <int-literal> "," ... ] "}"

<pred-param-type> ::= <basic-pred-param-type>
                    | "array" "[" <pred-index-set> "]" "of" <basic-pred-param-type>

<pred-index-set> ::= <index-set>
                   | "int"

<basic-literal-expr> ::= <bool-literal>
                       | <int-literal>
                       | <float-literal>
                       | <set-literal>

<basic-expr> ::= <basic-literal-expr>
               | <var-par-identifier>

<expr>       ::= <basic-expr>
               | <array-literal>

<par-expr>   ::= <basic-literal-expr>
               | <par-array-literal>

<var-par-identifier> ::= [A-Za-z_][A-Za-z0-9_]*

% Boolean literals
<bool-literal> ::= "false"
                 | "true"

% Integer literals
<int-literal> ::= [-]?[0-9]+
                | [-]?0x[0-9A-Fa-f]+
                | [-]?0o[0-7]+

% Float literals
<float-literal> ::= [-]?[0-9]+.[0-9]+
                  | [-]?[0-9]+.[0-9]+[Ee][-+]?[0-9]+
                  | [-]?[0-9]+[Ee][-+]?[0-9]+

% Set literals
<set-literal> ::= "{" [ <int-literal> "," ... ] "}"
```

```
                | <int-literal> ".." <int-literal>
                | "{" [ <float-literal> "," ... ] "}"
                | <float-literal> ".." <float-literal>

<array-literal> ::= "[" [ <basic-expr> "," ... ] "]"

<par-array-literal> ::= "[" [ <basic-literal-expr> "," ... ] "]"

% Parameter declarations

<par-decl-item> ::= <par-type> ":" <var-par-identifier> "=" <par-expr> ";"

% Variable declarations

<var-decl-item> ::= <basic-var-type> ":" <var-par-identifier> <annotations> [ "="
↪<basic-expr> ] ";"
                  | <array-var-type> ":" <var-par-identifier> <annotations> "="
↪<array-literal> ";"


% Constraint items

<constraint-item> ::= "constraint" <identifier> "(" [ <expr> "," ... ] ")"
↪<annotations> ";"

% Solve item

<solve-item> ::= "solve" <annotations> "satisfy" ";"
               | "solve" <annotations> "minimize" <basic-expr> ";"
               | "solve" <annotations> "maximize" <basic-expr> ";"

% Annotations

<annotations> ::= [ "::" <annotation> ]*

<annotation> ::= <identifier>
               | <identifier> "(" <ann-expr> "," ... ")"

<ann-expr>   := <basic-ann-expr>
              | "[" [ <basic-ann-expr> "," ... ] "]"

<basic-ann-expr>   := <basic-literal-expr>
                    | <var-par-identifier>
                    | <string-literal>
                    | <annotation>

<string-contents> ::= ([^"\n\] | \[^\n(])*

<string-literal> ::= """ <string-contents> """

% End of FlatZinc grammar
```

CHAPTER 4.4

Machine-readable JSON output format

--json-stream

type

4.4.1 Solution messages

```
{
  "type": "solution",
  "time": 1000,
  "output": {
    "foo": "foo output section",
    "bar": "bar output section"
  },
  "sections": ["foo", "bar"]
}
```

----

time--output-time

output

    json_json

sections

outputraw

4.4.2 Checker messages

```
{
  "type": "checker",
  "messages": [
    {
      "type": "solution",
      "output": {
        "default": "foo output section"
      },
      "sections": ["default"]
    }
  ]
}
```

messagessolution

4.4.3 Status messages

```
{
  "type": "status",
  "status": "ALL_SOLUTIONS",
  "time": 1000
}
```

==========

status

"ALL_SOLUTIONS"

"OPTIMAL_SOLUTION"

"UNSATISFIABLE"

"UNBOUNDED"

"UNSAT_OR_UNBOUNDED"

"UNKNOWN"

"ERROR"

time--output-time

4.4.4 Statistics messages

```
{
  "type": "statistics",
  "statistics": {
    "method": "satisfy",
    "flatTime": 1000
  }
}
```

%%%mzn-stat-end

statistics

4.4.5 Timestamp messages

```
{
  "type": "time",
  "time": 1000
}
```

--canonicalize--output-timetime

time

4.4.6 Comment messages

```
{
  "type": "comment",
  "comment": "% comment produced by solver\n"
}
```

comment%

4.4.7 Trace messages

```
{
  "type": "trace",
  "section": "default",
  "message": "traced message\n"
}
```

trace_stdouttrace_to_section()

sectiondefaulttrace_stdout

messagetrace_exp

trace()

4.4.8 Profiling messages

```
{
  "type": "profiling",
  "entries": [...]
}
```

--output-detailed-timing

entries

```
{
  "filename": "model.mzn",
  "line": 1,
  "time": 100
}
```

4.4.9 Paths messages

```
{
  "type": "paths",
  "paths": [...]
}
```

`--output-paths-to-stdout`

paths

```
{
  "flatZincName": "X_INTRODUCED_0_",
  "niceName": "x[1]",
  "path": "model.mzn|1|27|1|27|id|x;|0|0|0|0|il|0;"
}
```

```
{
  "constraintIndex": 9,
  "path": "model.mzn|3|12|3|59|ca|forall;model.mzn|3|12|3|59|ac;model.
↪mzn|3|20|3|20|i=4;model.mzn|3|23|3|23|j=5;model.mzn|3|47|3|58|bin|'!=';model.
↪mzn|3|47|3|58|ca|int_lin_ne;"
}
```

4.4.10 Error messages

```
{
  "type": "error",
  "what": "type error",
  "location": {...},
  "message": "cannot determine coercion from type float to type var int"
}
```

what

message

location

stack

what

syntax errorincludedFrom

cyclic include errorcycle

4.4.10.1 Locations

```
{
  "filename": "model.mzn",
  "firstLine": 1,
  "firstColumn": 1,
  "lastLine": 3,
  "lastColumn": 10
}
```

4.4.10.2 Stack traces

```
{
  "location": {...}
  "isCompIter": false,
  "description": "variable declaration"
}
```

4.4.11 Warning messages

```
{
  "type": "warning",
  "location": {...},
  "stack": [...],
  "message": "Warning message"
}
```

location

stack

message

-Werrorerror

符号

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

?

不固定的

?

函数
参数
命令行选项