The configuration problem

"This problem introduces the idea of using constraints to eliminate symmetries and contains some interesting modelling issues. It also illustrates structures, arrays indexed by variables, and logical connectives. The problem consists of plugging a set of electronic cards into racks with electric connectors. Each card is characterized by the power it requires, while each rack model is characterized by the maximal power it can supply, its number of connectors, and its price. Each card plugged into a rack uses a connector. The purpose of the model is to find an allocation of a given set of cards into the available racks."

To access this example, go to: examples/opl/config.

Now we explain the model file in detail.
Define number and range for models, cards, and racks.

```
int nbModel = ...;

int nbCard = ...;

int nbRack = ...;

range Models = 0..nbModel-1;

range Cards = 0..nbCard-1;

range Racks = 0..nbRack-1;
```

Define tuple parameter for each model containing power limit, number of connectors and price; Define tuple parameter for each card type containing card power and total quantity to plug in.

```
tuple modelType {
  int power;
  int connectors;
  int price;
};

// for cards, each card has a power parameter and total quantity for this card type
tuple cardType {
  int power;
  int quantity;
};
```

Load these parameters.

```
modelType model[Models] = ...;
cardType car[Cards] = ...;
```

Extract power, connector and price data from models. The maximum cards can be plugged into each rack has an upper bound as well.

```
int powerData[i in Models] = model[i].power;
int connData[i in Models ] = model[i].connectors;
int priceData[i in Models ] = model[i].price;
int maxQuantity= max(m in Models) model[m].connectors
```

Define decision variables: model for each rack and number for each card type to plug in each rack with upper limit as nbCards.

```
// each rack plug one model
dvar int rack[Racks] in Models;
// count how many cards? each counter is number of cards for a paritcular card, max is
maxQuantity
dvar int counters[Racks][Cards] in 0.. maxQuantity;
```

Define objective to be the total model price of each rack.

```
minimize
  sum(r in Racks) priceData[rack[r]];
```

Define constraints that total power of cards at each rack is limited by the model power.

```
  forall(r in Racks)
    sum(c in Cards) counters[r][c] * car[c].power <= powerData[rack[r]];
```

Define constraints that total number of connectors is limited by the model.

```
  forall(r in Racks)
    sum(c in Cards) counters[r][c] <= connData[rack[r]];
```

Define constraints that quantity of each card type is met.

```
forall(c in Cards)
  sum(r in Racks) counters[r][c] == car[c].quantity;
```