AbsCon - Short Guide (Draft)

Once installed, to run AbsCon, you must use a command like :

java abscon.Resolution {<problem>} <problemParameters> <solverParameters>

where :

<**problem**> is the name of a Java class that extends the class `Problem`. Such classes are put in various sub-folders of the folder `problems`, such as e.g., `acad`, `patt`, `puzz`, `rand`, `real` and `xcsp`. So, possibilities for <problem> are for example `problems.acad.Queens`, `problems.real.Crossword` and `problems.xcsp.XCSP`. Note that <problem> is optional, in which case, the default value is `problems.xcsp3.XCSP3`.

<**problemParameters**> is the list of parameters (separated by whitespace) specific to the chosen problem. For `problems.acad.Pigeons`, which is described below, this list only contains one integer. For `problems.xcsp3.XCSP3`, this is the name of a file (potentially compressed with lzma) representing an instance in format XCSP3.

<**solverParameters**> is the list of parameters (separated by whitespace) used for solving the instance. Any such parameter is of the form -name=value (potentially -name for some flag parameters).

1 Modeling using AbsCon

1.1 Constraint Satisfaction Problems

Suppose that you want to model the so-called pigeons problem, where n pigeons must be put in $n-1$ holes, with the interdiction of putting two pigeons in the same hole. Of course, this problem is trivially unsatisfiable, and does not look very interesting. However, it captures a structural form that some solving systems cannot handle easily. Besides, this is a very simple example that is perfect for our introduction purpose.

```
class Pigeons extends Problem {
  int nPigeons = askInt("Number of pigeons");

  void model() {
    Var[] x = array("x", dom(0, nPigeons - 2), nPigeons);
    allDifferent(x);
  }
}
```

As you can see, modeling a constraint satisfaction problem is quite easy : you just have to ask the parameters, and specify the variables and the constraints of the problem. You just need to define a method `void model()` (an alternative is to define a constructor).

To ask one parameter, or specify one variable (array) or one constraint (group, you just call one predefined method of class `Problem`. Here are the main methods :

Methods for specifying parameters		
String	askString	(String message)
int	askInt	(String message)
long	askLong	(String message)
double	askDouble	(String message)
boolean	askBoolean	(String message)

Methods for specifying constraints		
Ctr	ctr	(Var x, int[] values, Option... options)
Ctr	ctr	(Var[] scope, int[][] tuples, Option... options)
Ctr	ctr	(Var[] scope, String[][] tuples, Option... options)
Ctr	ctr	(String predicate, Option... options)
Ctr	allDifferent	(Var... scope)
Ctr	allDifferent	(Var[] scope, int except, boolean... weak)
Ctr	allDifferentMatrix	(Var[][] matrix)
Ctr	allEqual	(Var... scope)
Ctr	notAllEqual	(Var... scope)
Ctr	ordered	(Var[] x, int[] lengths, ERelationalOperator op)
Ctr	ordered	(Var[] x, ERelationalOperator op)
Ctr	lex	(Var[] t1, Var[] t2, ERelationalOperator operator)
Ctr	lex	(Var[][] vectors, ERelationalOperator operator)
Ctr	lexMatrix	(Var[][] matrix, ERelationalOperator operator)
Ctr	cardinality	(Var[] scope, int[] keys, int[] nbOccs)
Ctr	cardinality	(Var[] scope, int[] keys, int[] minOccs, int[] maxOccs)
Ctr	element	(Var index, Var[] vector, int result)
Ctr	element	(Var index, int[] vector, Var result)
Ctr	element	(Var index, Var[] vector, Var result)
Ctr	channel	(Var[] x, Var[] y)
Ctr	noOverlap	(Var x1, Var x2, int w1, int w2)
Ctr	noOverlap	(Var x1, Var x2, Var y1, Var y2, int w1, int w2, int h1, int h2)
Ctr	noOverlap	(Var[] origins, int[] lengths)
Ctr	noOverlap	(Var[][] origins, int[][] lengths)
Ctr	regular	(Var[] scope, Automata automata)
Ctr	mdd	(Var[] scope, Automata automata)
Ctr	mdd	(Var[] scope, int[][] tuples)
Ctr	sumKnapsack	(Var[] scope, int[] coefficients, int min, int max)
Ctr	sum	(Var[] vars, int[] coeffs, ERelationalOperator op, long limit)
Ctr	sum	(Var[] vars, ERelationalOperator op, long limit)
Ctr	sum	(Var[] vars, int[] coeffs, ERelationalOperator op, Var var)
Ctr	sum	(Var[] vars, ERelationalOperator op, Var var)
Ctr	atLeast1	(Var[] scope, int val)
Ctr	atLeast	(Var[] scope, int val, int k)
Ctr	atMost1	(Var[] scope, int val)
Ctr	atMost	(Var[] scope, int val, int k)
Ctr	exactly1	(Var[] scope, int val)
Ctr	eactly	(Var[] scope, int val, int k)
Ctr	exactly	(Var[] scope, int val, Var k)
Ctr	among	(Var[] scope, int[] vals, int k)
Ctr	distinctVectors	(Var[] t1, Var[] t2)
Ctr	distinctVectors	(Var[][] m)
Ctr	maximum	(Var maximum, Var... vector)
Ctr	minimum	(Var minimum, Var... vector)

Methods for specifying (arrays of) variables		
Var	var	(String name, Dom dom)
Var[]	array	(String name, Dom dom, int size, IntPredicate... p)
Var[]	array	(String name, IntToDom f, int size)
Var[][]	array	(String name, Dom dom, int size1, int size2, IntBiPredicate... p)
Var[][]	array	(String name, IntBiToDom f, int size1, int size2)
Var[][][]	array	(String name, Dom dom, int size1, int size2, int size3, IntTriPredicate... p)
Var[][][]	array	(String name, IntTerToDom f, int size1, int size2, int size3)
Var[][][][]	array	(String name, Dom dom, int size1, int size2, int size3, int size4, IntQuaterPredicate... p)
Var[][][][]	array	(String name, IntQuaterToDom f, int size1, int size2, int size3, int size4)

You can also post constraints per groups, using the methods `group` and `range`.

Methods for specifying constraints		
Ctr[]	group	(Ctr... ctrs)
Ctr[]	group	(Range range, CtrSupplier s)
Ctr[]	group	(Range range1, Range range2, CtrSupplierBi s)
Ctr[]	group	(Range range1, Range range2, Range range3, CtrSupplierTer s)
Ctr[]	group	(Range range1, Range range2, Range range3, Range range4, CtrSupplierQuater s)

Look at the javadoc associated with class `Problem` to see what are the various methods.

To solve an instance of problem `Pigeons`, the command line must be (assuming that the class `Pigeons` is put in your current directory and that "." is present in your classpath environment variable) :

java abscon.Resolution Pigeons

Then, you are asked :

Number of pigeons :

After giving an integer value (for example, 8), the instance is solved using the default configuration of the solver. Note that you can directly provide the value of the parameter at the command line, as follows :

java abscon.Resolution Pigeons 8

In our model above, we have used the famous global constraint `allDifferent`. Suppose that, instead, you want to post binary constraints of difference. Then, you would write : :

```
group(range(nPigeons),range(nPigeons), (i,j) -> i < j ? ctr(ne(x[i],x[i+1])) : null);
```