AbsCon 1.25 - User Manual

lecoutre@cril.fr

1 Installing AbsCon

To install and test AbsCon using Eclipse, download resources at `http://www.cril.univ-artois.fr/~lecoutre/`. Next, you have to :

1. use a computer equipped with both Java JSE 1.6 (or later) and Eclipse
2. run and customize Eclipse
   - if this is the first time Eclipse is run, accept the default proposed working directory, and then "Go to Workbench"
   - import the archive `AbsCon.zip` into Eclipse : File - Import - General - Existing Projects into Workspace - Next - Select Archive File - Finish
   - customize Eclipse : Package Explorer - View Menu - Package Presentation - Hierarchical
   - remove irrelevant libraries : Properties - Java Build Path - Libraries - poi.jar (remove)
   - clean the project : Project - Clean
3. download the configuration file `mac.xml` as well as the problem instance file `queens-12.xml`
4. run the solver with a command like :

   ```
   java abscon.Resolution mac.xml 1 xcsp queens-12.xml
   ```

   where 1 means that only one instance must be solved and xcsp refers to the problem type that has to be solved (here an instance in XCSP format).

   If necessary, add the following Java argument : `-cp .:~/workspace/abscon/bin`

2 Configuring AbsCon

The file `mac.xml` mentioned above looks like :

```
<configuration
    nbSearchedSolutions="1"
    timeout="" >

  <problem
      binaryEncoding=""
      symmetryBreaking=""
      inferAllDifferentConstraints="no"
      useGlobalConstraints="yes">
    <optimization
        way=""
        startBound=""
        stopBound="" />
```

```
    <extensionRepresentation
        type="str2" />
    <extensionConversion
        enabled="yes"
        arityLimit="2" />
  </problem>

  <propagation
      class="AC" >
      <queue
        class="QueueOfVariables" >
        <revisionHeuristic
          class="minDom" />
      </queue>
  </propagation>

  <solver
      class="IterativeSystematicSolver"
      enablePrepro="yes"
      enableSearch="yes"
      backtracking="SBT"
      branching="bin" >
    <restarts
        nbRuns="1"
        cutoff=""
        factor="1.1" />
    <learning
        nogoodRecording="rst" />
    <variableHeuristic
        class="maxWDegOnDom"
        lastConflictSize="0" />
    <valueHeuristic
        class="minFirst" />
  </solver>

  <xml
      savingResults=""
      savingInstance="" />
</configuration>
```

The attribute values given in this file are the default values. This means that the file `mac.xml` above is equivalent to the file `default.xml` that only contains :

```
<configuration />
```

Of course, to use the default values except for a timeout set to 10 seconds, I can use a configuration file that contains :

```
<configuration
   timeout="10s" />
```

because Abscon will use the default values for the other configuration parameters.

Now, let us explain the role of each XML attribute :

1. configuration
   (a) `nbSearchedSolutions` : this is the number of solutions that must be found before the solver stops. When the value is the empty string, `nbSearchedSolutions=""`, all solutions (if any) must be found.

(b) `timeout` : this is number of milliseconds that are passed before the solver stops. When the value is the empty string, `timeout=""`, there is no time limit. You can use the letter s as a suffix to denote seconds as e.g., `timeout="10s"` (10 seconds).

2. problem
   (a) `binaryEncoding` : this allows us to transform a CN only involving (non-binary) extensional constraints into a binary CN. The possible values are "dual", "hidden" and "double" (see the literature). When the value is the empty string, `binaryEncoding=""` no encoding is performed.
   (b) `symmetryBreaking` : this allows us to post (if possible) symmetry breaking constraints using the method described in [1]. The possible values are "le", "lex", "le_merged" and "lex_merged". When the value is the empty string, `symmetryBreaking=""` no symmetry breaking method is used. Saucy is required to run the different methods.
   (c) `inferAllDifferentConstraints` : when set to "yes", redundant allDifferent constraints are sought and posted (if any) to improve the filtering capabilities of the solver. Try this on a `pigeons` instance.
   (d) `useGlobalConstraints` : when set to "yes", global constraints that are posted (through programmation) are posted instead of a decomposition into smaller arity constraints.
3. problem/optimization
   (a) `way` : when the constraint network is a CSP instance, the value must be the empty string, `way=""`, and the two other attributes `startBound` and `stopBound` are not relevant. When the constraint network is a COP instance, the value must be "min" or "max" ; the former (resp. latter) case corresponds to a minimization (resp. maximization) of the objective function. Currently, a COP instance can only be defined through programming (i.e., not with xcsp, see the programmer manual). For a COP instance, typically we have `nbSearchedSolutions=""` because we need to find an optimal solution.
   (b) `startBound` : when `way` is set to "min" (resp. "max"), the real value set to `startBound` indicates the initial upper bound (resp. lower bound).
   (c) `stopBound` : when `way` is set to "min" (resp. "max"), the real value set to `stopBound` indicates a known lower bound (resp. upper bound).
4. problem/extensionRepresentation
   (a) `type` : for non-binary constraints defined in extension, there are many ways of representing and propagating them. Among possible values, we find "v" for GAC-valid,"a" for GAC-allowed, "va" for GAC-valid+allowed, "str" for simple tabular reduction, "str2" and "mdd".
5. problem/extensionConversion
   (a) `enabled` : for (typically binary) constraints defined in intension, it may be worthwhile to convert them into extensional form. This is performed at loading time. To allow such a conversion, the value must be "yes".
   (b) `arityLimit` : this indicates that any intensional constraint whose arity is less than or equal to the given arity must be converted into extension.
6. propagation
   (a) `class` : indicates the name of the class that must be used to propagate constraints. For (G)AC, use "AC". Among other possible values, we find, "SAC1", "SAC3", "ESAC3", "CDC1", "CPC1", "DC1", "DC2".
7. propagation/queue
   (a) `class` : indicates the scheme of propagation. Currently, only "QueueOfVariables" for variable-oriented propagation scheme is operational.
8. propagation/queue/revisionHeuristic

(a) `class` : indicates the name of the class that allows us to select elements in the queue in order to perform revisions, prefixed with "min" or "max". For example, with a variable-oriented propagation scheme, `class="minDom"` indicates that at each step the next variable in the queue to be selected is the one with the smallest domain.

9. solver
   (a) `class` : indicates the name of the class that allows us to explore the search space. Typically, this is `class="IterativeSystematicSolver"`.
   (b) `enablePrepro` : if set to "yes", indicates that a preprocessing stage must be performed.
   (c) `enableSearch` : if set to "yes", indicates that a search stage must be performed.
   (d) `backtracking` : indicates which mechanism of backtracking is used. Typically, this is `backtracking="SBT"` for standard backtracking.
   (e) `branching` : indicates the branching scheme. Possible values are "bin" for binary branching, "non" for non-binary (or d-way) branching, and "res" for restricted binary branching.
10. solver/restarts
   (a) `nbRuns` : indicates the number of runs to be performed. A value strictly greater than 1 is relevant only if a cutoff value is given.
   (b) `cutoff` : indicates the number of failed assignments before the solver restarts. When no value is given, the cutoff is set to the maximal integer value.
   (c) `factor` : indicates the increasing factor of the number of failed assignments between successive runs.
11. solver/learning
   (a) `nogoodRecording` : indicates the way nogoods can be collected. When restarts are performed, this is typically `nogoodRecording="rst"` for nogood recording from restarts.
12. solver/variableHeuristic
   (a) `class` : indicates the name of the class that selects the variables to be assigned, prefixed with "min" or "max". For example, `class="minDom"` indicates that at each step the next variable to be instantiated is the one with the smallest domain. Another example is `class="maxWDegOnDom"` that indicates that at each step the next variable to be instantaited is the one with the greatest ratio wdeg/dom (which is equivalent to min dom/wdeg).
   (b) `lastConflictSize` : with a value greater than or equal to 1, indicates that a reasoning from last conflicts must be used.
13. solver/valueHeuristic
   (a) `class` : indicates the name of the class that selects the next value to be assigned to the last selected variable, prefixed with "min" or "max". For example, `class="minNbConflicts"` indicates that at each step the next value to be assigned is the one with the smallest nuber of conflicts (computed at loading time). Another example is `class="minFirst"` that indicates that at each step the next value to be assigned is the first value in the current domain (a kind of lexicographic order).
14. xml
   (a) `savingResults` : indicates the name of a directory where results (XML files) will be stored. If the value is the empty string, results are not saved in XML format.
   (b) `savingInstance` : indicates if the instance must be saved in extensional form, `savingInstance ="extension"`, in intensional form if possible, `savingInstance="intension"` or if it must not be saved, `savingInstance=""`.

There are other technical configuration parameters that can be identified in the Java class `Data`.

3 Running AbsCon

To run AbsCon, you must use a command like :

java abscon.Resolution <configurationFile> <nbInstances> <problem> <problemParameters> <arguments>

where :

- <configurationFile> is the name of the XML file that contains all configuration parameters as shown in the previous section.
- <nbInstances> is the number of instances to be solved ; typically, this is 1, but it may be any positive value or also -1 that denotes that all available instances must be solved (this is used for example when a directory is given as a parameter for a problem).
- <problem> is the name of a package that denotes a problem type. For example, xcsp is the name used to read instances in XCSP format, acad.queens is the name for the academic n-queens problem, etc.
- <problemParameters> : this is a list of parameters separated by whitespace that depends on the problem type. For example :

```
java abscon.Resolution mac.xml 1 xcsp queens-12.xml
java abscon.Resolution mac.xml 1 acad.queens 12 2
```

  In the first case, xcsp requires only one parameter, the name of a file (or directory). In the second case, acd.queens requires two parameters : the number of queens and the model that must be used.
- <arguments>, they are optional and they are :
  - `trace` : displays every decision taken during search
  - `display <n>` : displays more or less information concerning the problem instance and the solution(s) found. n is a value in $\{0, 1, 2, 3\}$.
  - `II <idsOfInstantiatedVariables> II <valuesOfInstantiatedVariables> II` : instantiate the variables whose ids are given with the given values. For example :

```
java abscon.Resolution mac.xml 1 xcsp queens-12.xml II 4 II 7 II
java abscon.Resolution mac.xml 1 acad.queens 12 2 II 0 1 II 3 5 II
```

    In the first case, the variable with id 4 (i.e., the 5th variable) is necessarily assigned with the value 7. In the second case, the two variables with id 0 and 1 are necessarily assigned with the values 3 and 5.
  - `SS <idsOfSelectedVariables> SS` : every variable with an id not in this list is discarded as well as every constraint involving it (except for allDifferent constraints whose scope are reduced). This is only valid with xcsp. You can use i..j to denote all ids between i and j (both included). For example :

```
java abscon.Resolution mac.xml 1 xcsp queens-12.xml SS 2..10 SS
```

    only keeps variables with an id between 2 and 10, and constraints involving them.

Références

[1] C. Lecoutre and S. Tabary. Des symétries locales de variables aux symétries globales. In *Proceedings of JFPC'08 (in french)*, pages 181–190, 2008.