Project sheet 1: Implementing a CSP Solver

Prof. Dr. Bernhard Nebel, Dr. Stefan Wölfl,
Dr. Julien Hué and Matthias Westphal

May 9, 2012

This work is due on 25.05.2012
Please register your group and members before 18.05.2012

1 How to hand in your code

Please form groups of two if possible. The code submission will be handled by the git source code management system. For this purpose we will host a git repository for each group. To access the repository *everyone* is required to create a private/public key pair.
On Linux (e.g. in the computer pool):
Create the private/public key pair by running:

```
ssh-keygen -t rsa -f csp2012
```

This will give you `csp2012` (the private key) and `csp2012.pub` (the public key).

- Copy the *private key* into `~/.ssh/`
- Sent the *public key* to Matthias Westphal (westpham@informatik.uni-freiburg.de) and state your group name and the other member of the group. We will then setup a repository for your group.

Add the following entries to your ssh config file (`~/.ssh/config`):

```
Host csp2012
HostName aermel.informatik.uni-freiburg.de
User csp2012
IdentityFile ~/.ssh/csp2012
```

We will notify you by email once we have created the repository for your group. Once the repository has been setup by us you can start to use the repository. First *clone* the git repository by running: (replace GROUP-NAME with your group name)

```
git clone csp2012:GROUP-NAME
```

This will give you the directory GROUP-NAME which already contains some simple example problems in the XCSP format.

1.1 A minimal guide to git

The only necessary actions to use the git repository are listed here. Adding files to git:

```
git add file
```

This tells git to keep track of the given file. Simply put, only tracked files can be uploaded to us (see `commit` and `push`). Tracked files must be all your *source files*. Untracked files will not be seen by us, this might be appropriate for temporary files or binaries from your build process. Adding also works on files within directories, e.g.:

```
git add path/file
```

will retain the path within the repository as well. Similarly, you can remove ("untrack") files from git by

```
git rm file
```

This will not affected previous uploads (or so-called commits) but causes future uploads and updates to not contain this file. To store a snapshot of the current state of the tracked files run

```
git commit -a -m "Commit message"
```

This *commit* contains the current version of all tracked files. The message serves as a label for this snapshot. Finally, to upload all your commits to us run

```
git push
```

Without `git push` we cannot access your work. Conversely, you can download and apply the changes from your group's repository by

```
git pull
```

This will automatically merge the latest version from the repository with your local snapshots (i.e. commits). More information and help on git can be found on e.g. `http://git-scm.com/`. Contact Matthias Westphal (westpham@informatik.uni-freiburg.de) or Julien Hué (hue@informatik.uni-freiburg.de) in case of problems.

1.2 Deadlines and programming languages

Your work must contain a simple Makefile in the root directory. The simple command *make* must be sufficient to compile your work. The *supported languages* are: C, C++, Java, PHP and Python. If you require support from us you *must* use one of these languages. If you do not require any support you can freely choose a (reasonable/realistic) programming language. However, your work *must* compile on a Ubuntu 11.10 32bit computer when the deadline is reached. You can submit before deadlines and ask for feedback. We can thus check whether the code is compiling and you will have some remaining time for correcting the issues.

2 Part 1: Reading the problem description

The first part of the implementation is about reading the instances and setting up the appropriate data types. You must create data types useful for the solver (e.g. Constraint, Variable). We do not impose the way you represent information in memory. You can represent the constraints the way you want: either storing allowed tuples or forbidden tuples or both.

Your program is expected to read the input stored in a file and give correct output. If the name your executable file is *solver* and the input *input.xml*, then the command *./solver input.xml* reads the input and display the result on the standard output precisely in the form we give below.

The input will be given in the format described in Section 2.1 and the output must be exactly in the format described in Section 2.2.

2.1 The XML format for input

The CSP instances will be encoded in the XCSP 2.1 format. A description of the format is available at the following address: http://arxiv.org/pdf/0902.2362v1.

We nevertheless apply some restrictions:

- We are only concerned with standard CSP instances (no Weighted CSPs and no Quantified CSP instances);
- We are only concerned with extensional constraints (no intensional constraints and no global constraints). Beware that extensional constraints can be given in the form of authorized tuples (*'supports'*) or forbidden tuples (*'conflicts'*)

Example 1. *In the following, an XML description of the 4-queen problem is given in order to help you understand the format:*

```
<instance>
 <presentation name="4queens" maxConstraintArity="2"
           format="XCSP 2.1"/>
 <domains nbDomains="1">
  <domain name="D0" nbValues="4">1..4</domain>
 </domains>
 <variables nbVariables="4">
  <variable name="c1" domain="D0"/>
  <variable name="c2" domain="D0"/>
  <variable name="c3" domain="D0"/>
  <variable name="c4" domain="D0"/>
 </variables>
 <relations nbRelations="3">
  <relation name="e1" arity="2" nbTuples="6"
         semantics="supports">
    1 3 | 1 4 | 2 4 | 3 1 | 4 1 | 4 2
  </relation>
  <relation name="e2" arity="2" nbTuples="8"
```

```
          semantics="supports">
    1 2 | 1 4 | 2 1 | 2 3 | 3 2 | 3 4 |
    4 3 | 4 1
   </relation>
   <relation name="e3" arity="2" nbTuples="10"
          semantics="supports">
    1 2 | 1 3 | 2 1 | 2 3 | 2 4 | 3 1 |
    3 2 | 3 4 | 4 2 | 4 3
   </relation>
  </relations>
  <constraints nbConstraints="6">
   <constraint name="r12" arity="2" scope="c1 c2"
            reference="e1"/>
   <constraint name="r23" arity="2" scope="c2 c3"
            reference="e1"/>
   <constraint name="r34" arity="2" scope="c3 c4"
            reference="e1"/>
   <constraint name="r13" arity="2" scope="c1 c3"
            reference="e2"/>
   <constraint name="r24" arity="2" scope="c2 c4"
            reference="e2"/>
   <constraint name="r14" arity="2" scope="c1 c4"
            reference="e3"/>
  </constraints>
</instance>
```

As you can see, the format is explicit and there are no particular difficulties:

domains *gives the list of domains in the problem. They will be linked with variables in the "variables" section.*

variables *gives the list of variables and associate them with their domain.*

relations *is where relations are described. Here is given sets of forbidden or allowed tuples. If the constraints describe allowed tuples the "semantics" attribute is set to "supports". If the constraints describe forbidden tuples the "semantics" attribute is set to "conflicts".*

constraints *allows to specify constraints, i.e. link variables (through the scope attribute) and a relation (through the reference attribute).*

The 4-queen problem is also already available as an example in the repository.

2.2 The text format for output

In order to verify that your XML parser correctly reads the XML description, for now we only require your program to write variables and constraints on the standard output.

The format is the following:

- First, the variables are given separated by commas. A semicolon ends the line.
- Then, all the constraints are described here. They are all given in the form of forbidden tuples. So, if the input format specified some constraints under the form of allowed tuples you will need to translate it. This does not imply how to store those informations in memory, you can do either way. If no constraints hold between some sets of variables, there is nothing to write on the output format. For example, there is no constraint of arity 3 in the 4-queens problem, so there is no constraints of size 3 in the output given below. First the scope is given (with variables separated by commas), then a pipe symbol and afterwards all the forbidden tuples are given separated by commas. A semicolon ends the line.

Example 2. *We here give the expected output for the 4-queens explained in the Example 1.*

```
c1, c2, c3, c4;
c1, c2 | 1 1, 1 2, 2 1, 2 2, 2 3, 3 2, 3 3, 3 4, 4 3, 4 4;
c2, c3 | 1 1, 1 2, 2 1, 2 2, 2 3, 3 2, 3 3, 3 4, 4 3, 4 4;
c3, c4 | 1 1, 1 2, 2 1, 2 2, 2 3, 3 2, 3 3, 3 4, 4 3, 4 4;
c1, c3 | 1 1, 1 3, 2 2, 2 4, 3 1, 3 3, 4 1, 4 4;
c2, c4 | 1 1, 1 3, 2 2, 2 4, 3 1, 3 3, 4 1, 4 4;
c1, c4 | 1 1, 1 4, 2 2, 3 3, 4 1, 4 4;
```