Homework 1 Tutorial: Abscon, Webgrader, and the C++ XML Parser

Robert Woodward

August 31, 2015

The goal of Homework 1 is to set-up your data-structures that you will be using for the homework assignments in this class. The homework assignments build on one another so it is vital you give your code a solid foundation. Knowing how to access your data-structures storing the CSP model is an important step in being able complete the subsequent homework assignments. This tutorial will give you a biref introduction to using Abscon and webgrader. It then also gives you a (very) brief introduction to the C++ XML Parser.

1 Abscon Tutorial

Abscon will parse in XML files in the XCSP 2.1 format. This section of the tutorial will show you how to utilize the existing parser and its data-structures in helping you create your own data-structures for the homework assignment. The code provided here gives you one way to start implementing the homework. You are not required to use any of the code here.

1. Download `http://www.cril.univ-artois.fr/~lecoutre/research/tools/Tools2008.zip`
2. Abscon has a XML parser for the XCSP 2.1 format located in "src/abscon/tools/InstanceParser.java". The InstanceParser class will load the XML file into various .
3. Create a new file "src/csp/MyParser.java" with contents:

```
package csp;

import csp.Variable;
import abscon.instance.tools.InstanceParser;
import abscon.instance.components.PConstraint;

import java.util.List;
import java.util.ArrayList;

public class MyParser {
    private List<Variable> variables;
    public MyParser(String filename) {
        InstanceParser parser = new InstanceParser();
        parser.loadInstance(filename);
        parser.parse(false);

        variables = new ArrayList<Variable>();

        System.out.println("Instance name: <Not currently parsed!  Need to modify the InstanceParser()>");
```

```
        System.out.println("Variables");
        for(int i = 0; i < parser.getVariables().length; i++) {
            // System.out.println(parser.getVariables()[i].getName());
            Variable newVar = new Variable(parser.getVariables()[i]);
            System.out.println(newVar);
            variables.add(newVar);
        }

        System.out.println("Constraints");
        for(String key : parser.getMapOfConstraints().keySet()) {
            PConstraint con = parser.getMapOfConstraints().get(key);
            System.out.println(con.getName());
        }
    }
    public static void main(String[] args) {
        MyParser parser = new MyParser("./4queens-supports.xml");
    }
}
```

And a new file "src/csp/Variable.java" with contents:

```
package csp;

import abscon.instance.tools.InstanceParser;
import abscon.instance.components.PVariable;

import java.lang.String;

public class Variable {
    /// Keep a reference to the original variable, just in case it is needed later
    protected PVariable varRef;
    /// Best to create a *deep copy* of the data-structures that are needed for the homework
    protected String name;

    public Variable(PVariable var) {
        varRef = var;
        name = var.getName();
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return "Name: " + name + ", initial-domain: x, constraints: x, neighbors: x";
    }
}
```

4. Let's make it compile into a jar file. Create a file "META-INF/MAINIFEST.MF" with contents:

```
Manifest-Version: 1.0
Main-Class: csp/MyParser
```

And a Makefile:

```
all:
    javac -J-Xmx256m -d bin -sourcepath src src/csp/MyParser.java src/abscon/instance/intension/*/*.java
    jar -J-Xmx256m cfm csp.jar src/META-INF/MANIFEST.MF -C bin .
clean:
    rm -rf bin/* csp.jar
```

5. We can now execute "make", which makes csp.jar. We can execute the jar by running "java -jar csp.jar".

6. Notice, we still have lots to do for the homework assignment! You might find that the InstanceParser class does not parse everything you want. For example: you need to print out the instance name (given in the XML file), but the parser does not parse this in. You will need to modify the method parsePresentation() in the InstanceParser to get access to this information.

2 Webgrader

The webgrader executes the program that you have handed in on handin. Webgrader allows you to see if your program works correctly outside of your personal computer, and is the same system that the grader will use to evaluate your homework. You are encouraged to start handing in your assignment early and use webgrader as a tool for testing the output of your program. You can submit your code to handin multiple times.

1. You need to tell webgrader how to run your program, through a file called "runProgram.sh". We will create a file called "runProgram.sh":

   ```
   java -Xmx256m -jar csp.jar $@
   ```

2. Zip up the contents of the folder with your Makefile, runProgram.sh, and your source code. The makefile and runProgram.sh files should be in the root of the zip. *Do not include them inside of a folder.* (Note: We used a makefile earlier to compile our program. Webgrader can also make using maven and ant, if you choose to use them instead.)

3. Handin the newly created zip file on handin `https://cse-apps.unl.edu/handin/`.

4. Open up webgrader `https://cse.unl.edu/~cse421/grade/`, and run the assignment.

3 Using the C++ Parser

An existing C++ XML Parser is available from `http://www.cril.fr/~roussel/CSP-XML-parser/`. The parser includes a file called "examples/c++/example.cc" that shows an example of how to use the C++ parser.

The parser uses a series of virtual functions (like "callbacks") for when the parser finds certain tags in the XCSP file. For example, the "addVariable()" function will give you a variable's name and domain-name. From here you can start to create your variable object for later use.

Some of the callbacks will not be used, and you will not have to handle them. For example, anything with global constraints (but you will have to handle intension constraint predicates).