Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2018-2019

Project title: Futoshiki Solver
Tool: Clingo

Name: Diaconu Călin
Group: 30432
Email: diaconuccalin@gmail.com

Assoc. Prof. dr. eng. Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

The project and the tool

The project developed during the Artificial Intelligence laboratory is a Java application that solves the Japanese puzzle Futoshiki. The application uses the Clingo tool for solving the actual puzzle, while the Java part is used for the GUI and the interaction with the terminal.

Clingo is an Answer Set Programming system to ground and solve logic programs. It is actually composed of Gringo, which powers the grounding in Clingo, and Clasp, which the solver and it is responsible for the searching of the solution. Clingo is part of Potassco, the Potsdam Answer Set Solving Collection, developed at the Potsdam University in Germany.

Chapter 2

Installing the tool

The following steps are needed for installing the tool on a 64bit Ubuntu system[13]:

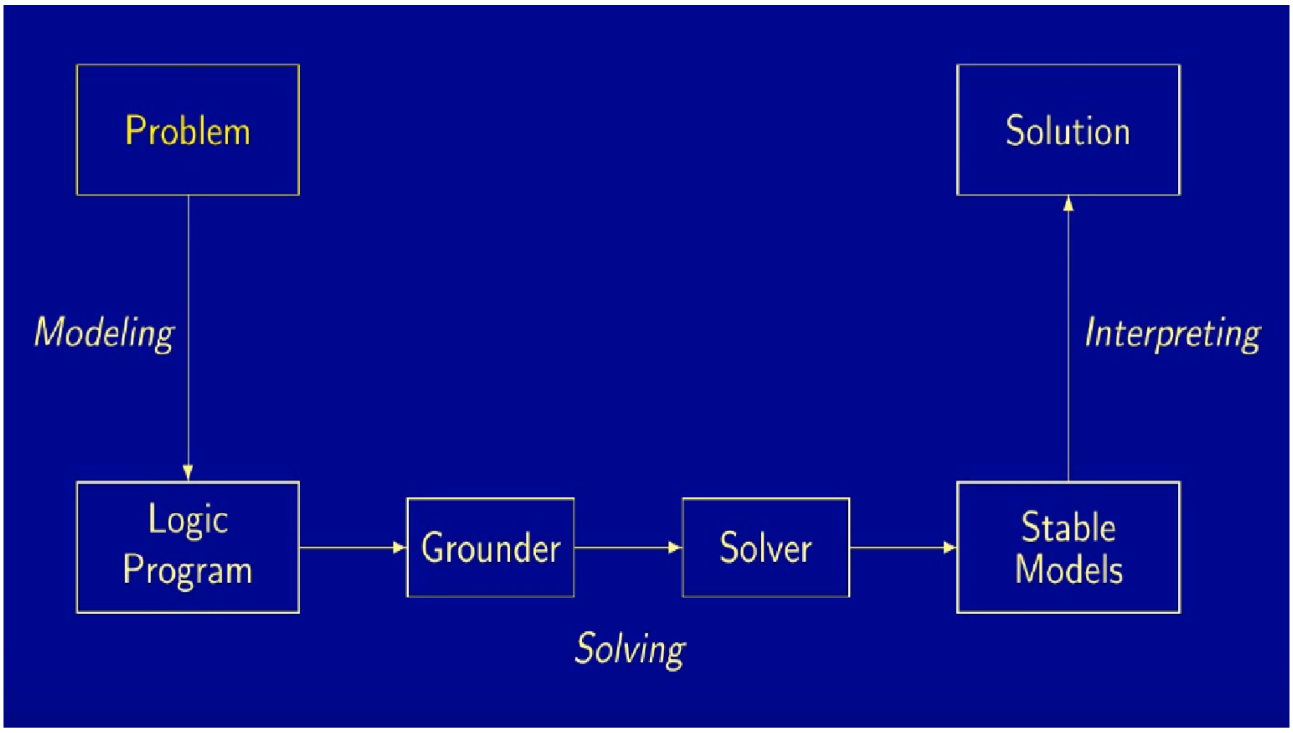
1. Open a terminal window.
2. Install Anaconda[1]:
   (a) Install the needed extended dependencies by running the following command:
   ```
   apt-get install libgl1-mesa-glx libegl1-mesa libxrandr2\
   libxrandr2 libxss1 libxcursor1 libxcomposite1 libasound2\
   libxi6 libxtst6
   ```
   (b) Download Conda by running the following command:
   ```
   wget https://repo.anaconda.com/miniconda/Miniconda3\
   -latest-Linux-x86_64.sh -O ~/miniconda.sh
   ```
   (c) Install Conda from the downloaded package by running the following command:
   ```
   bash ~/miniconda.sh -b -p $HOME/miniconda
   ```
   (d) Activate Conda by running the following command:
   ```
   eval "$(/Users/jsmith/anaconda/bin/\
   conda shell.YOUR_SHELL_NAME hook)"
   ```
   (e) Install Conda's shell functions by running the following command:
   ```
   conda init
   ```
3. Install Clingo from the Potassco channel by running the following command:
   ```
   conda install -c potassco clingo
   ```

Chapter 3

Running and understanding examples

Answer set programming (ASP) is a form of declarative programming oriented towards difficult (primarily NP-hard) search problems. It is based on the stable model (answer set) semantics of logic programming. In ASP, search problems are reduced to computing stable models, and answer set solvers - programs for generating stable models - are used to perform the search.[2]

The ASP solving process consists of the following steps[16]:



The ASP solving process

3.1 Example 1: Graph coloring

3.1.1 Problem

In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors"to elements of a graph subject to certain constraints. In its simplest form (the one used in this exercise), it is a way of coloring the vertices of a graph such that no two adjacent vertices are of the same color; this is called a vertex coloring.[6]

3.1.2 Modeling

Problem instance

A graph consisting of nodes and edges, along with some colors, is represented as[3]

1. facts formed by predicates node/1 and edge/2
2. facts formed by predicate col/1

The graph.lp file:

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).
```

Problem class

Assign each node one color such that no two nodes connected by an edge have the same color:

1. Each node has a unique color
2. Two connected nodes must not have the same color

The color.lp4 file:

```
col(r). col(g). col(b).

{ color(X,C) : col(C) } == 1 :- node(X).
:- edge(X,Y), color(X,C), color(Y,C).

#show color/2.
```

3.1.3 Grounding

In the terminal, run the following commands:

1. human-readable output:

```
gringo color.lp4 graph.lp -text
```

2. machine-readable output:

```
gringo color.lp4 graph.lp
```

3.1.4 Solving

In the terminal, run the following commands:

1. For obtaining only one solution:

```
gringo color.lp4 graph.lp | clasp
```

2. For obtaining all the solutions:

```
gringo color.lp4 graph.lp | clasp 0
```

3. For obtaining all the solutions and for displaying the stats:

```
gringo color.lp4 graph.lp | clasp 0 -stats
```

3.1.5 Miscellaneous

The `clingo` command can replace the gringo+clasp commands (the result from the grounder is directly sent to the solver).

3.2 Example 2: The n-Queens puzzle

3.2.1 Problem

Place n queens on an n x n chess board such that no two queens attack one another. [4]

3.2.2 Basic encoding

Check that no two queens are on the same row, on the same column, on the same right diagonal or on the same left diagonal.

```
{ queen(1..n,1..n) }.

:- not { queen(I,J) } == n.
:- queen(I,J), queen(I,JJ), J != JJ.
:- queen(I,J), queen(II,J), I != II.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I-J == II-JJ.
:- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I+J == II+JJ.
```

3.2.3 Advanced encoding

A faster, more succint encoding:

```
{ queen(I,1..n) } == 1 :- I = 1..n.
{ queen(1..n,J) } == 1 :- J = 1..n.

:- { queen(D-J,J) } >= 2, D = 2..2*n.
:- { queen(D+J,J) } >= 2, D = 1-n..n-1.
```

3.2.4 An even more advanced encoding

```
{ queen(I,1..n) } == 1 :- I = 1..n.
{ queen(1..n,J) } == 1 :- J = 1..n.

:- { queen(I,J) : diag1(I,J,D) } >= 2, D=1..2*n-1.
:- { queen(I,J) : diag2(I,J,D) } >= 2, D=1..2*n-1.

diag1(I,J,I+J-1) :- I = 1..n, J = 1..n.
diag2(I,J,I-J+n) :- I = 1..n, J = 1..n.

#show queen/2.
```

Chapter 4

Understanding conceptual instrumentation

Answer set programming (ASP) is a form of declarative programming oriented towards difficult (primarily NP-hard) search problems. It is based on the stable model (answer set) semantics of logic programming. In ASP, search problems are reduced to computing stable models and answer set solvers - programs for generating stable models - are used to perform search. The computational process employed in the design of many answer set solvers is an enhancement of the DPLL algorithm and, in principle, it always terminates.

In a more general sense, ASP includes all applications of answer sets to knowledge representation and the use of Prolog-style query evaluation for solving problems arising in these applications.

Lparse is the name of the program that was originally created as a grounding tool (front-end) for the answer set solver smodels. The language that Lparse accepts is now commonly called AnsProlog, short for Answer Set Programming in Logic. It is now used in the same way in many other answer set solvers, including clasp.

AnsProlog An AnsProlog program consists of rules of the form

```
<head> :- <body>.
```

The symbol `:-` (”if”) is dropped if `<body>` is empty; such rules are called facts. The simplest kind of Lparse rules are rules with constraints.

Another useful construct included in this language is choice. For instance, the choice rule `{p,q,r}.` says: choose arbitrarily which of the atoms p, q, r to include in the stable model. The lparse program that contains this choice rule and no other rules have eight stable models - arbitrary subsets of p, q, r. The definition of a stable model was generalized to programs with choice rules. Choice rules can be treated also as abbreviations for propositional formulas under the stable model semantics. For instance, the choice rule above can be viewed as shorthand for the conjunction of three ”excluded middle” formulas:
$(p \vee \neg p) \wedge (q \vee \neg q) \wedge (r \vee \neg r)$

The language of lparse allows us also to write ”constrained” choice rules, such as `1{p,q,r}2.`. This rule says: choose at least 1 of the atoms p, q, r, but no more than 2. The meaning of this rule under the stable model semantics is represented by the propositional formula
$(p \vee \neg p) \wedge (q \vee \neg q) \wedge (r \vee \neg r) \wedge (p \vee q \vee r) \wedge \neg(p \wedge q \wedge r)$.

Cardinality bounds can be used in the body of a rule as well, for instance: `:- 2{p,q,r}.`
Adding this constraint to an Lparse program eliminates the stable models that contain at least 2 of the atoms p, q, r. The meaning of this rule can be represented by the propositional formula
$\neg((p \wedge q) \vee (p \wedge r) \vee (q \wedge r))$.

Variables (capitalized,as in Prolog) are used in Lparse to abbreviate collections of rules that follow the same pattern, and also to abbreviate collections of atoms within the same rule.

A range is of the form `(start..end)` where start and end are constant valued arithmetic expressions. A range is a notational shortcut that is mainly used to define numerical domains in a compatible way. Ranges can also be used in rule bodies with the same semantics.

A conditional literal is of the form: `p(X):q(X)`. If the extension of q is q(a1); q(a2);...;q(aN). the condition is semantically equivalent to writing p(a1),p(a2),...,p(aN) in the place of the condition.[2]

Chapter 5

Project description

5.1 Narrative description

The project is a solver for the Futoshiki game, being made for entertainment purposes.

Futoshiki (meaning "inequality") is a logic puzzle game from Japan. It was developed by Tamaki Seto in 2001.

The puzzle is played on a square grid. The objective is to place the numbers such that each row and column contains only one of each digit. Some digits may be given at the start, Inequality constraints are initially specified between some of the squares, such that one must be higher or lower than its neighbor. These constraints must be honored in order to complete the puzzle.

Solving the puzzle requires a combination of logical techniques. Numbers in each row and column restrict the number of possible values for each position, as do the inequalities.

Once the table of possibilities has been determined, a crucial tactic to solve the puzzle involves AB elimination, in which subsets are identified within a row whose range of values can be determined. For example, if the first two squares within a row must contain 1 or 2, then these numbers can be excluded from the remaining squares. Similarly, if the first three squares must contain 1 or 2, 1 or 3 and 1 or 2 or 3, then those remaining must contain other values (4 and 5 in a 5x5 puzzle).

Another important technique is to work through the range of possibilities in open inequalities. A value on one side of an inequality determines others, which then can be worked through the puzzle until a contradiction is reached and the first value is excluded.

The first step to solve the puzzle is to enumerate possible values based on inequalities and non-duplication within rows and columns. Then AB elimination may be usable to narrow down the range of possibilities. Logical deduction within the inequalities can restrict the range of possibilities.

A solved futoshiki puzzle is a Latin square. As with the sudoku case, more difficult futoshiki puzzles require the use of various types of chain patterns.[5]

This solver gives the user the possibility to choose the size of the grid and complete any number of digits and inequality constraints. If the puzzle is unsolvable, the application will notice the user accordingly, but if it is solvable, it will complete the remaining blank squares with the appropriate digits. Although there may be more than one solution, the solver only chooses and displays one.

5.2 Facts

For a n x n board:

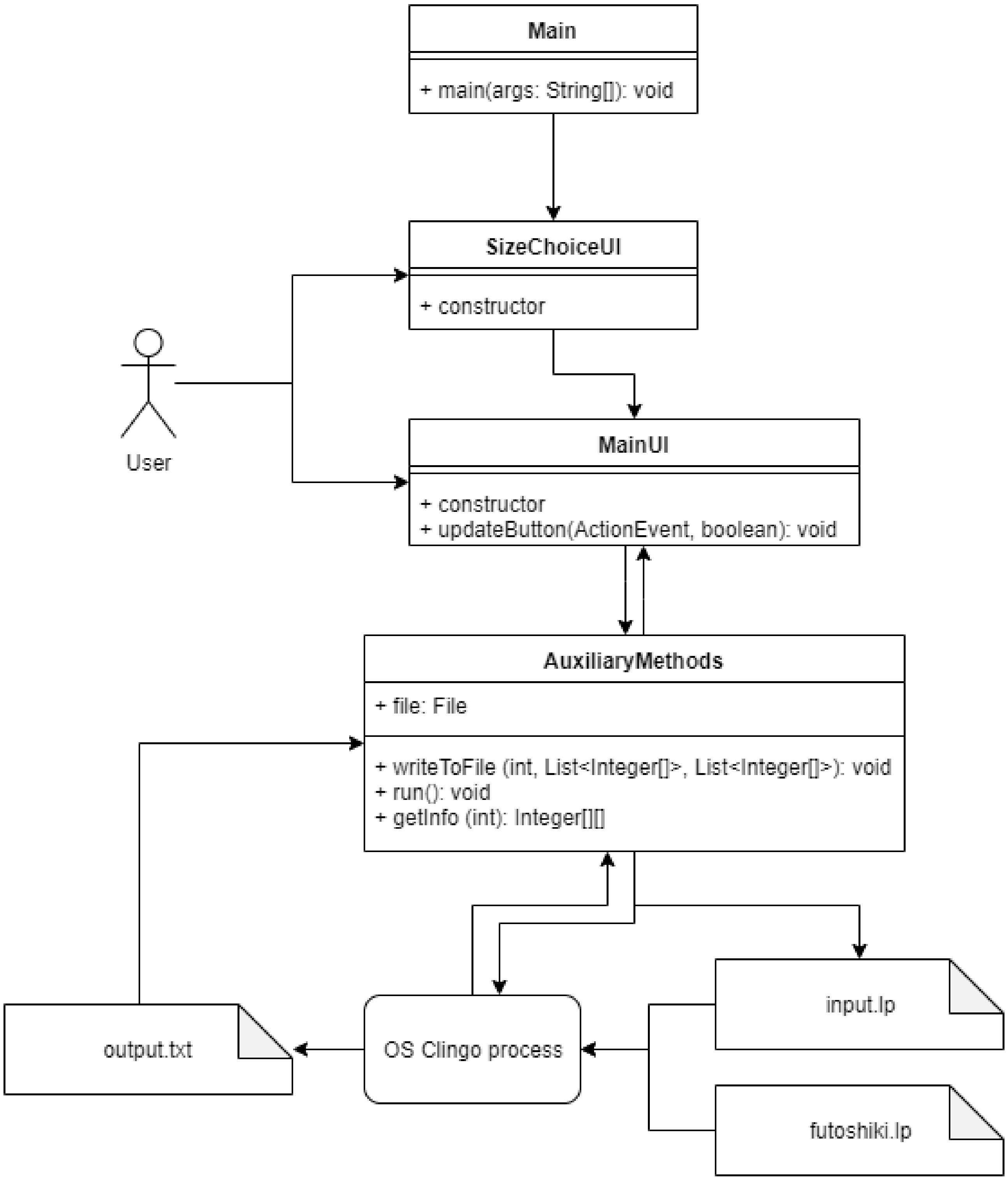
1. The cells only contain numbers from 1 to n.
2. Each cell contains exactly one number.
3. No two cells in the same column contain the same number.
4. No two cells in the same row contain the same number.
5. The given cells must not be changed.
6. The given lessThan rules must be respected (in the case of lessThan(X1, Y1, X2, Y2), the number in the cell that is on position (X1, Y1) must have a smaller value than the number in the cell that is on position (X2, Y2)).

5.3 Specifications

The grounder together with the solver will choose from all the possible combinations of numbers on a board the ones which fulfill the requirements from above. The application will first detect the cells that were filled by the user. Than it will choose only the results that fulfill the rules for a Latin Square. In the end it will take care of the lessThan rules.

The user must only give integers higher than 0 but smaller than the chosen size and give a solvable and correctly completed board. The user should receive a result according to the list of facts above.

5.4 Top level design of the scenario


Main
+ main(args: String[]): void
SizeChoiceUI
+ constructor
User
MainUI
+ constructor
+ updateButton(ActionEvent, boolean): void
AuxiliaryMethods
+ file: File
+ writeToFile (int, List<Integer[]>, List<Integer[]>): void
+ run(): void
+ getInfo (int): Integer[][]
output.txt
OS Clingo process
input.lp
futoshiki.lp

The top level design diagram

The user inputs the desired size of the board, the given cells and the lessThan rules.

After pressing the Submit button, the Java application generates an ”input.lp” text file, according to the given input, and makes a system call with the command ”clingo futoshiki.lp input.lp”, which runs Clingo (both the grounder Gringo and the solver Clasp) on the futoshiki.lp file, containing the general rules for solving a futoshiki puzzle and the input.lp file. The application stores the result of the call in a file that will be called ”output.txt”.

The application extracts the information about wether or not the given configuration is solvable. If it is, it updates the GUI according to the positioning of the values on the board. Otherwise, it shows an ”Unsolvable” message.

The Java application consists of 4 classes.

The AuxiliaryMethods class contains static methods that deal with writing to the input.lp file, according to the rules stated above, - writeToFile -, with making the system call and generating the output file - run - and with reading and interpreting the output file - getInfo.

There are two classes that deal with the user interface and a Main class that contains the callable main method.

5.5 Related work

For understanding the Futoshiki game, its rules and strategies for solving the puzzle can be found here [5]. Another way of understanding the game is playing it and one website with playable Futoshiki games is this one [7].

For understanding the answer set programming system, one starting point could be its Wikipedia page [2]. There is also a number of books and presentations that cover this subject thoroughly: [16] [8] [14] [11] [15].

For learning how to use Gringo, Clasp and Clingo, there are the Potassco Guide [13], the lecture on Answer Set Programming and Clingo from the University of Potsdam [16] and a series of YouTube tutorials presented by Torsten Schaub from the University of Potsdam [3].

A Sudoku solver represents a frequent project both for ASP and for Clingo. These can be a starting point for a Futoshiki solver as all of them include a Latin Square solver. Such projects can be found here: [12] [17] [10] [9].

Chapter 6

Implementation details

Futoshiki solver

```
number(1..size).
```

This line establishes the range of values for the "number" atom. "size" is a variable that will be established in by the user input.

```
x(X,Y,N) :- given(X,Y,N).
```

This line will fill all the given values into the final ("x") atom. The line can be read as "for every *given* atom with the X, Y, N values, there will be created an atom with the X, Y, N values.

```
1 { x(X,Y,N):number(N) } 1 :- number(X), number(Y).
1 { x(X,Y,N):number(X) } 1 :- number(N), number(Y).
1 { x(X,Y,N):number(Y) } 1 :- number(N), number(X).
```

These 3 lines generate three sets of atoms, for exactly 1 number in each cell, for unique numbers on columns and for unique numbers on rows. The "1" digits represent the lower and higher bounds of the number of atoms to be generated (because bothe bounds are 1, exactly 1 atom will be generated). This will happen for values of X, Y and N going through all the possible values of number, which are 1 to size. This strategy, although generates more atoms, it doesn't require as much time to compute because there are two less rules to pass through the solver.

```
:- x(X1,Y1,N1), x(X2,Y2,N2), lessThan(X1,Y1, X2,Y2), N1 >= N2.
```

This is the only constraint of the application. Going through all the lessThan rules given by the user, the solver eliminates the solutions with x atoms that have the coordinates mentioned by a lessThan rules but don't fulfill it (it eliminates the results where N1 >= N2).

```
#show x/3
```

This is the line that establishes the rule about how the information should be displayed, such that only x atoms with exactly 3 values from a solution will be shown.

User input

```
#const size = input.
```

This establishes the value of the variable "size" to "input", where input is an integer higher than 0.

```
given(x,y,n).
```

These will represent the "hints" received by the user at the beggining of a Futoshiki game, where x, y and n are all integers greater than 0 but smaller than size.

```
lessThan(x1, y1, x2, y2).
```

These will represent all the lessThan rules in the game, where x1, y1, x2, y2 are all integers greater than 0 but smaller than size.

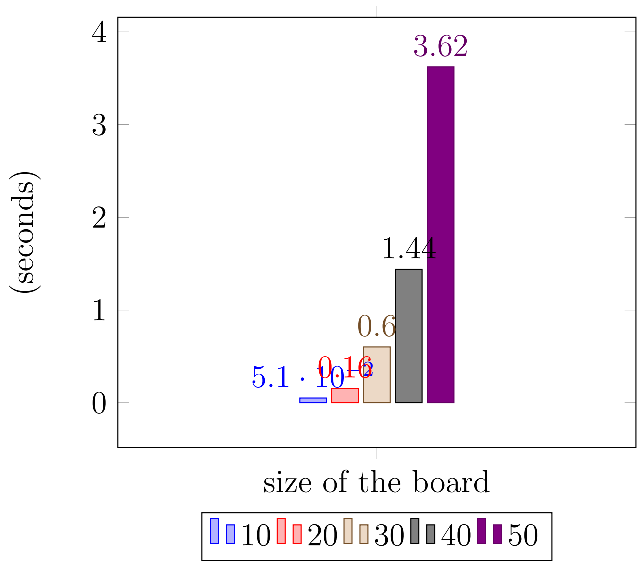
Chapter 7

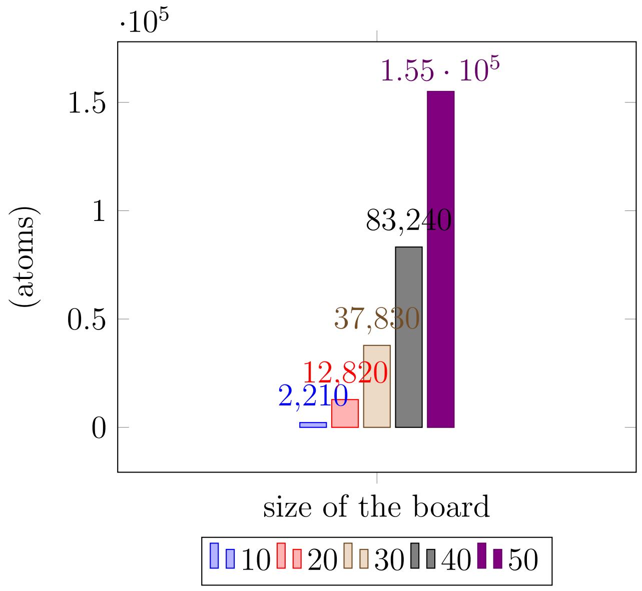
Graphs and experiments

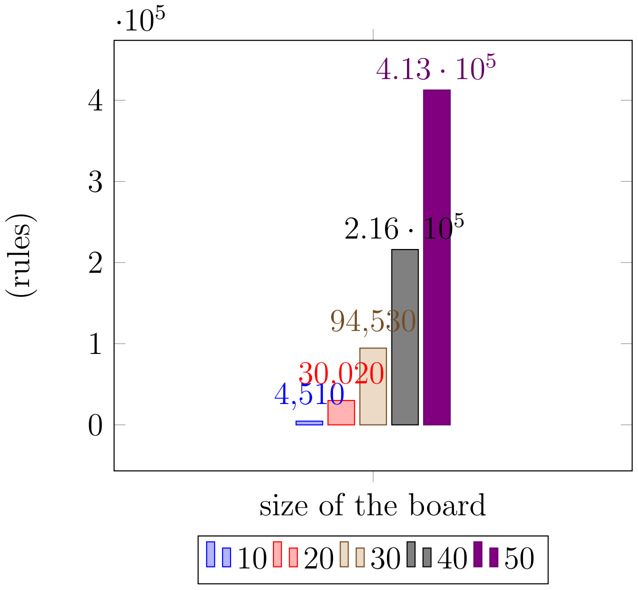
The tests were done on empty boards (with no lessThan rules, such that only the latin square solving is computed). The size of a row increseas from 10 to 20, 30, 40 and 50.

The results are obtained by running Clingo on the same computer, using the "–stats" option.

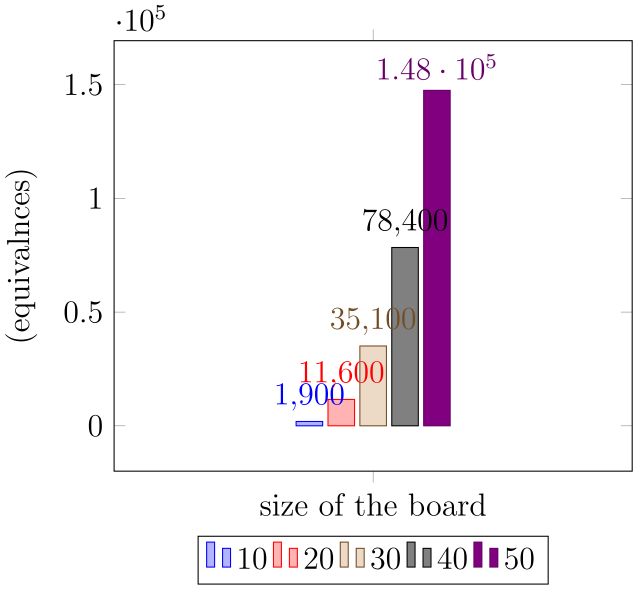
It can be noticed that the number of constraints grow the slowest.



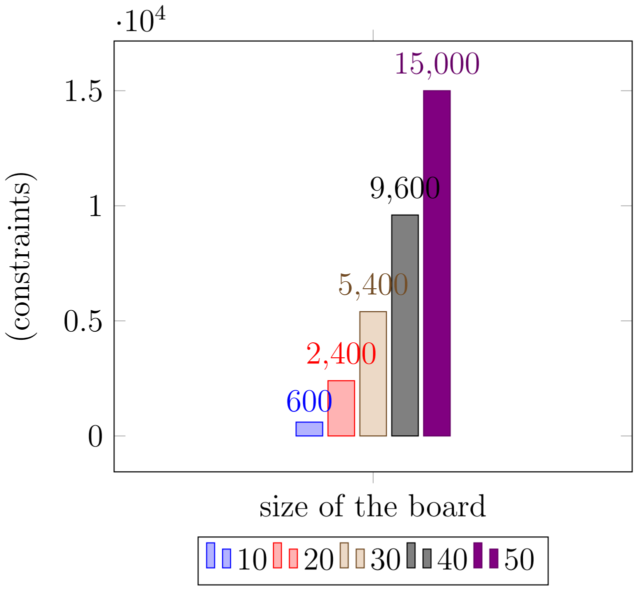




$\cdot 10^5$
(rules)
4
3
2
1
0
$4.13 \cdot 10^5$
$2.16 \cdot 10^5$
94,530
30,020
4,510
size of the board
10
20
30
40
50



$\cdot 10^5$
(equivalnces)
1.5
1
0.5
0
$1.48 \cdot 10^5$
78,400
35,100
11,600
1,900
size of the board
10
20
30
40
50



$\cdot 10^4$
(constraints)
1.5
1
0.5
0
15,000
9,600
5,400
2,400
600
size of the board
10
20
30
40
50

Chapter 8

Related work and documentation

8.1 Related approaches

The tests were done on a 50 x 50 empty board (with no lessThan rules, such that only the latin square solving is computed) for the time tests and on a 10 x 10 empty board for the rest of the computations.

The tests were done on the following implementations:

1. The one being used:

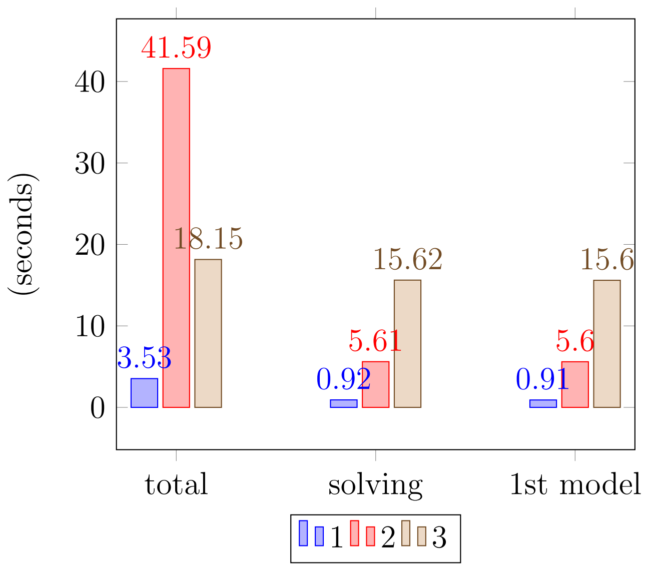
```
1 { x(X,Y,N):number(N) } 1 :- number(X), number(Y).
1 { x(X,Y,N):number(X) } 1 :- number(N), number(Y).
1 { x(X,Y,N):number(Y) } 1 :- number(N), number(X).
```

2.
```
1 { x(X,Y,N):number(N) } 1 :- number(X), number(Y).
:- x(X,Y1,N), x(X,Y2,N), Y1!=Y2.
:- x(X1,Y,N), x(X2,Y,N), X1!=X2.
```

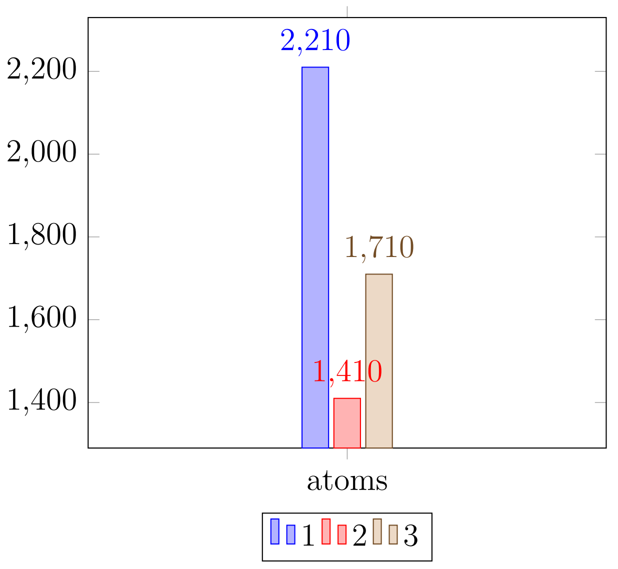
3.
```
1 { x(X,Y,N):number(N) } 1 :- number(X), number(Y).
:- 2 { x(X,Y,N) : number(N) }, number(X), number(Y).
:- 2 { x(X,Y,N) : number(X) }, number(N), number(Y).
:- 2 { x(X,Y,N) : number(Y) }, number(N), number(X).
```

The results are obtained by running Clingo on the same computer, using the ”–stats” option.

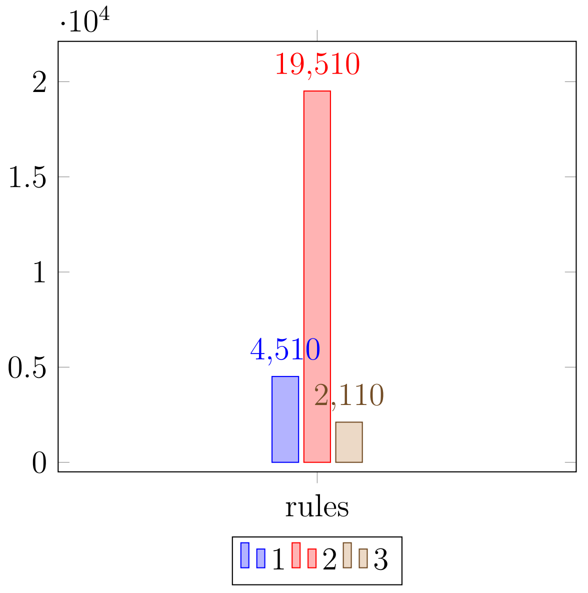
The 1 implementation scores the best times. Although it needs to go through more atoms and equivalnces, it has the least constraints that need to be solved (by a large margin).



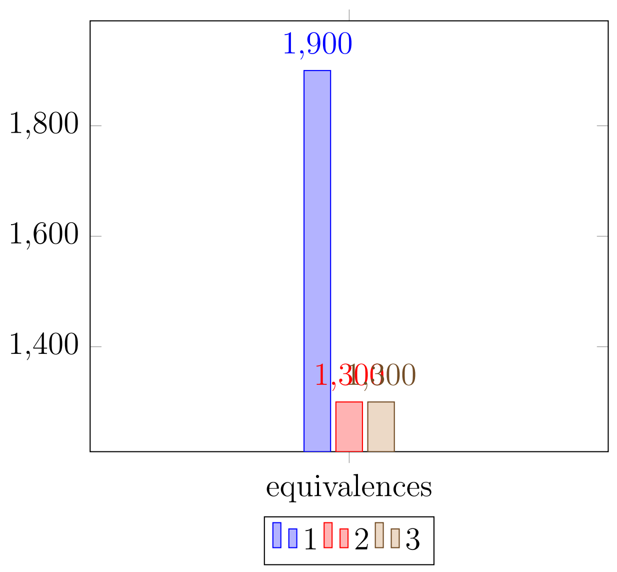
(seconds)
40
30
20
10
0
41.59
18.15
3.53
15.62
5.61
0.92
15.6
5.6
0.91
total
solving
1st model
1
2
3

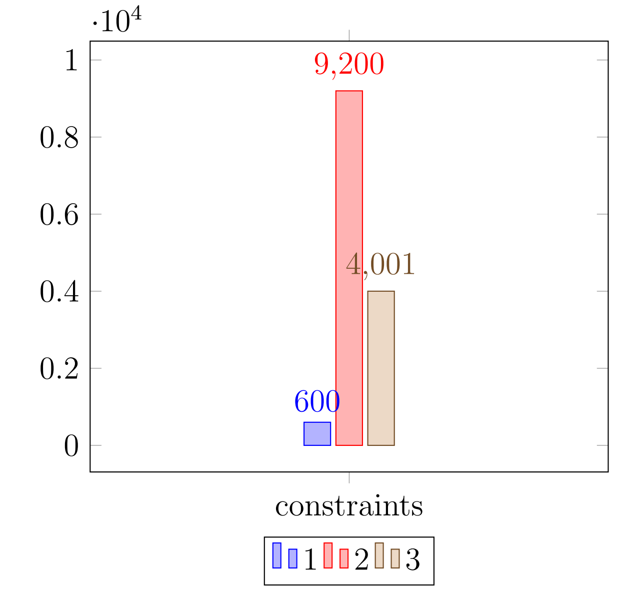


2,200
2,000
1,800
1,600
1,400
2,210
1,710
1,410
atoms
1
2
3



$\cdot 10^4$
2
1.5
1
0.5
0
19,510
4,510
2,110
rules
1
2
3





8.2 Advantages and limitations of your solution

Advantages

Because there are many implementations of Sudoku, I could choose the fastest way to solve the latin square.

This is the only graphical implementation for solving such a puzzle using Answer Set Programming.

Disadvantage

Because there is a lack of Futoshiki solvers using Answer Set Programming and because I didn't have enough time to familiarize myself with the Clingo tool, I am unable to tell if there are faster ways to implement the lessThan checking part or an all around faster solver for Futoshiki puzzles.

8.3 Possible extensions of the current work

1. Find puzzles that are more complex to solve.
2. Unite more such puzzle solvers into a single application.
3. Create a mobile application that recognizes a board of Futoshiki (or other similar game), using the device's camera, and completes it via AR.

Appendix A

Your original code

This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained. Including in this section any line of code taken from someone else leads to failure of IS class this year. Failing or forgetting to add your code in this appendix leads to grade 1. Don't remove the above lines.

AuxiliaryMethods.java

```java
import java.io.*;
import java.util.List;
import java.util.Scanner;

class AuxiliaryMethods {
    private static File file = new File("output.txt");

    static void writeToFile(int size, List<Integer[]> given,
                            List<Integer[]> lessThan) {
        try {
            BufferedWriter bufferedWriter = new BufferedWriter(
                    new FileWriter("input.lp"));

            bufferedWriter.write("#const size = " + size + ".\n\n");

            for (Integer[] el : given) {
                bufferedWriter.write("given(" + (el[0] + 1) + "," + (el[1] + 1)
                        + "," + el[2] + ").\n");
            }
            bufferedWriter.write("\n");

            for (Integer[] el : lessThan) {
                bufferedWriter.write("lessThan(" + (el[0] + 1) + "," + (el[1] + 1)
                        + ", " + (el[2] + 1) + "," + (el[3] + 1) + ").\n");
            }

            bufferedWriter.close();
        } catch (IOException e) {
            System.out.println(e.toString());
        }
    }

    static void run() {
        try {
            String[] args = new String[]{"clingo", "futoshiki.lp", "input.lp"};
            ProcessBuilder builder = new ProcessBuilder(args);
            File file = new File("output.txt");
            builder.redirectOutput(file);
            builder.redirectError(file);
            Process process = builder.start();
            InputStream is = process.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(is));

            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println(e.toString());
        }
    }

    static Integer[][] getInfo(int size) {
        Integer[][] toReturn = new Integer[size][size];
        try {
            Scanner scanner = new Scanner(file);

            String line, line1;
            line = "";
            line1 = "";

            while (scanner.hasNextLine()) {
                line1 = line;
                line = scanner.nextLine();
                if (line.compareTo("SATISFIABLE") == 0 ||
                        line.compareTo("UNSATISFIABLE") == 0)
                    break;
            }

            if (line.isEmpty() || line.compareTo("UNSATISFIABLE") == 0) {
                toReturn[0][0] = -1;
            } else {
                for (int i = 2; i < line1.length(); i += 9) {
                    int i1 = Integer.parseInt(line1.charAt(i) + "");
                    int j1 = Integer.parseInt(line1.charAt(i + 2) + "");
                    int val = Integer.parseInt(line1.charAt(i + 4) + "");

                    toReturn[i1 - 1][j1 - 1] = val;
                }
            }
            scanner.close();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
        return toReturn;
    }
}
```

Main.java

```
    public class Main {
    public static void main(String[] args) {
        new SizeChoiceUI();
    }
}
```

MainUI.java

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.util.ArrayList;
import java.util.List;

class MainUI extends JFrame {
    private static void updateButton(ActionEvent e, boolean flag) {
        JButton sourceButton = (JButton) e.getSource();
        String aux = sourceButton.getText();

        if (flag) {
            if (aux.compareTo("") == 0)
                sourceButton.setText("<");
            else if (aux.compareTo("<") == 0)
                sourceButton.setText(">");
            else
                sourceButton.setText("");
        } else {
            if (aux.compareTo("") == 0)
                sourceButton.setText("");
            else if (aux.compareTo("") == 0)
                sourceButton.setText("");
            else
                sourceButton.setText("");
        }
    }

    MainUI(int size) {
        int w = size * 100 + 80;
        int h = size * 100;
```

```
setLayout(null);
setSize(w, h);
setTitle("Futoshiki");
setLocationRelativeTo(null);
setDefaultCloseOperation(EXIT_ON_CLOSE);

JTextField[][] jTextFields = new JTextField[size][size];

for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        jTextFields[i][j] = new JTextField(1);
        jTextFields[i][j].setBounds((j + 1) * 100 - 90,
                (i + 1) * 100 - 90, 45, 45);
        add(jTextFields[i][j]);
    }
}

JButton[][] jButtons = new JButton[size][size - 1];
JButton[][] jButtons1 = new JButton[size - 1][size];

for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        if (j < size - 1) {
            jButtons[i][j] = new JButton("");
            jButtons[i][j].setBounds((j + 1) * 100 - 40,
                    (i + 1) * 100 - 90, 45, 45);

            jButtons[i][j].addActionListener(e -> updateButton(e, true));

            add(jButtons[i][j]);
        }

        if (i < size - 1) {
            jButtons1[i][j] = new JButton("");
            jButtons1[i][j].setBounds((j + 1) * 100 - 90,
                    (i + 1) * 100 - 40, 45, 45);

            jButtons1[i][j].addActionListener(e -> updateButton(e, false));

            add(jButtons1[i][j]);
        }
    }
}

JButton submitButton = new JButton("Submit");
submitButton.setBounds(size * 100 - 40, 10, 100, 25);

submitButton.addActionListener(e -> {
    List<Integer[]> given = new ArrayList<>();
```

```
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        String aux = jTextFields[i][j].getText();

        if (!aux.isBlank()) {
            try {
                int val = Integer.parseInt(aux);
                if (val < 1)
                    val = 1;
                else if (val > size)
                    val = size;
                Integer[] toAdd = {i, j, val};
                given.add(toAdd);
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(submitButton,
                        "Only numbers as input!");
            }
        }
    }
}

List<Integer[]> lessThan = new ArrayList<>();
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        if (j < size - 1) {
            String buttonText = jButtons[i][j].getText();
            if (buttonText.compareTo("<") == 0) {
                Integer[] toAdd = {i, j, i, j + 1};
                lessThan.add(toAdd);
            } else if (buttonText.compareTo(">") == 0) {
                Integer[] toAdd = {i, j + 1, i, j};
                lessThan.add(toAdd);
            }
        }

        if (i < size - 1) {
            String buttonText = jButtons1[i][j].getText();
            if (buttonText.compareTo("") == 0) {
                Integer[] toAdd = {i, j, i + 1, j};
                lessThan.add(toAdd);
            } else if (buttonText.compareTo("") == 0) {
                Integer[] toAdd = {i + 1, j, i, j};
                lessThan.add(toAdd);
            }
        }
    }
}

AuxiliaryMethods.writeToFile(size, given, lessThan);
AuxiliaryMethods.run();
```

```
            Integer[][] info = AuxiliaryMethods.getInfo(size);

            if (info[0][0] == -1) {
                JOptionPane.showMessageDialog(submitButton, "Unsolvable!");
            } else {
                for (int i = 0; i < size; i++) {
                    for (int j = 0; j < size; j++) {
                        jTextFields[i][j].setText(info[i][j] + "");
                    }
                }
            }
        });

        add(submitButton);

        JButton backButton = new JButton("Back");
        backButton.setBounds(size * 100 - 40, 40, 100, 25);
        backButton.addActionListener(actionEvent -> {
            new SizeChoiceUI();
            dispose();
        });
        add(backButton);

        setVisible(true);
    }
}
```

SizeChoiceUI.java

```
    import javax.swing.*;

class SizeChoiceUI extends JFrame {
    SizeChoiceUI() {
        int w = 300;
        int h = 140;

        setLayout(null);
        setSize(w, h);
        setTitle("Size choice");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        JLabel jLabel = new JLabel("Choose the board size:");
        JTextField jTextField = new JTextField(1);
        JButton jButton = new JButton("Submit");

        jButton.addActionListener(e -> {
            try {
                int size = Integer.parseInt(jTextField.getText());
                if (size < 2)
                    size = 2;
                new MainUI(size);
                dispose();
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(jButton,
                        "Only numbers as input!");
            }
        });

        jLabel.setBounds(20, 10, 200, 30);
        jTextField.setBounds(220, 10, 40, 30);
        jButton.setBounds(70, 55, 90, 30);

        add(jLabel);
        add(jTextField);
        add(jButton);

        setVisible(true);
    }
}
```

futoshiki.lp

```
% Define the possible range of values
number(1..size).

% Fill the given values
x(X,Y,N) :- given(X,Y,N).

% Check for rows and columns to have unique values

% Slower:
% 1 { x(X,Y,N):number(N) } 1 :- number(X), number(Y).
% :- x(X,Y1,N), x(X,Y2,N), Y1 != Y2.
% :- x(X1,Y,N), x(X2,Y,N), X1 != X2.

1 { x(X,Y,N):number(N) } 1 :- number(X), number(Y).
1 { x(X,Y,N):number(X) } 1 :- number(N), number(Y).
1 { x(X,Y,N):number(Y) } 1 :- number(N), number(X).


% Check for "less than" constraints
:- x(X1,Y1,N1), x(X2,Y2,N2), lessThan(X1,Y1, X2,Y2), N1 >= N2.

#show x/3.
```

Appendix B

Quick technical guide for running your project

1. Make sure you have the Ubuntu OS 64bit running.
2. Open a terminal window.
3. Check that java is installed with the `java -version` command. If it is not installed, follow the instructions from here in order to install it.
4. Check that clingo is installed with the `clingo -version` command. If it is not installed, follow the instructions in chapter 2.
5. `cd` to the project's location.
6. Run the command `java futoshiki`
7. Give the desired size of the board (the length of a side should be given here as an integer greater than zero).
8. Complete the desired values in the white spaces corresponding to cells and, using the buttons between the cells, complete the desired lessThan rules.
9. Press the solve button. The application should now complete the remaining cells with the appropriate numbers or give an "Unsolvable" message if the board is unsolvable (for example, the rules are impossible to fulfill, the user filled two identical numbers on a row or a column or the user filled two adjacent cells in such a way that they don't respect a lessThan rule).

Bibliography

[1] Anaconda documentation - installing on linux.

[2] The answer set programming wikipedia page.

[3] The asp solving process tutorial.

[4] The eight queens puzzle wikipedia page.

[5] The futoshiki wikipedia page.

[6] The graph coloring wikipedia page.

[7] Online futoshiki game.

[8] Chitta Baral. *Knowledge Represantation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.

[9] Davide Canton. sudoku.logic.

[10] Enrico Höschler. Answer set programming: Sudoku solver - website, 2018.

[11] T. Soinenen I Niemela, P. Simons. *Stable model semantics of weight constraint rules.* LPNMR '99, 2000.

[12] Håkan Kjellerstrand. The hakank website.

[13] Benjamin Kaufmann Marius Lindauer Max Ostrowski Javier Romero Torsten Schaub Sven Thiele Philipp Wanko Martin Gebser, Roland Kaminski. *Potassco User Guide.* University of Potsdam, 2019.

[14] Vladimir Lifschitz Bruce Porter Michael Gelfond, Frank In van Harmelen. *Handbook of Knowledge Representation.* Elsevier, 2008.

[15] V. Lifschitz P. Ferraris. *Weight constraints as nested expressions.* Theory and Practice of Logic Programming, 2005.

[16] Torsten Schaub. Answer set solving in practice.

[17] Joel Verhagen. Sudoku solver in clasp and gringo.