```
package yices;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ResultParser {

	private Map<String, String> values;

	public Map<String, String> getValues() {
		return values;
	}

	static Pattern pattern = Pattern.compile("\\(= a(\\d+) (\\d+)\\)");

	public void parse(InputStream inputStream) throws IOException {
		values = new HashMap<String, String>();
		BufferedReader br = new BufferedReader(new InputStreamReader(
				inputStream));
		String line = null;
		while ((line = br.readLine()) != null) {
			if ("unsat".equals(line)) {
				values = null;
				return;
			}
			Matcher matcher = pattern.matcher(line);
			if (matcher.matches()) {
				values.put("a" + matcher.group(1), matcher.group(2));
			}
		}
	}

}
```

```
package yices;

import java.io.PrintStream;
import java.text.MessageFormat;

public class RuleGenerator {

	private int n;

	private transient PrintStream pw;

	public RuleGenerator(int n) {
		super();
		this.n = n;
	}

	public void generateGlobalRules(PrintStream pw) {
		this.pw = pw;
		generateVariables();
		generateRules();
	}

	public void check() {
		pw.println("(check)");
	}

	protected void generateVariables() {
		int size = n * n;
		for (int i = 0; i < size; i++) {
			for (int j = 0; j < size; j++) {
				pw.println(MessageFormat.format("(define a{0}::int)", i * size
						+ j));
			}
		}
		for (int i = 0; i < size; i++) {
			for (int j = 0; j < size; j++) {
				pw.println(MessageFormat.format("(assert (<= a{0} {1}))", i
						* size + j, size));
				pw.println(MessageFormat.format("(assert (>= a{0} {1}))", i
						* size + j, 1));
			}
		}
	}

	protected void generateRules() {
		int size = n * n;
		for (int i = 0; i < size; i++) {
			for (int j = 0; j < size; j++) {
				for (int k = j + 1; k < size; k++) {
					pw.println(MessageFormat.format("(assert (/= a{0} a{1}
))",
							i * size + j, i * size + k));
					pw.println(MessageFormat.format("(assert (/= a{0} a{1}
))",
							j * size + i, k * size + i));
					pw.println(MessageFormat.format("(assert (/= a{0} a{1}
))",
							((i / n) * n + j / n) * size + (i % n)
 * n + j % n,
							((i / n) * n + k / n) * size + (i % n)
 * n + k % n));
				}
			}
```

```
                }
        }

        public void generateGivenRules(int i, int j, int val) {
                int size = n * n;
                if (i >= size || i < 0 || j >= size || j < 0) {
                        throw new IllegalArgumentException(MessageFormat.format(
                                        "({0},{1}) is not a valid coordinator", i, j));
                }
                if (val > size || val < 1)
                        throw new IllegalArgumentException(MessageFormat.format(
                                        "{0} is not a valid value", val));
                pw.println(MessageFormat.format("(assert (= a{0} {1}))", i * size + j,
                                val));
        }
}
```

```java
package ui;

import java.awt.Dimension;
import java.awt.Font;

import javax.swing.JButton;

public class NumberButton extends JButton {
	/**
	 *
	 */
	private static final long serialVersionUID = -7564018057338800979L;

	private static Font font = new Font("Verdana", Font.PLAIN, 12);

	private int index;

	public NumberButton(int index) {
		super();
		setPreferredSize(new Dimension(50, 50));

		setFont(font);
		this.index = index;
	}

	public int getIndex() {
		return index;
	}

	public void setIndex(int index) {
		this.index = index;
	}

}
```

```
package ui;


public class Main {

	/**
	 * @param args
	 */

	public static void main(String[] args) {
		new SudokuFrame();
	}
}
```

```
package ui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Map;

import javax.swing.AbstractAction;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JToolBar;
import javax.swing.SwingUtilities;

import model.SudokuSolver;

public class SudokuFrame extends JFrame {

	/**
	 *
	 */
	private static final long serialVersionUID = -6275020573594671300L;

	private SudokuSolver solver;

	private JPanel centerPanel;

	/**
	 * @param args
	 */
	public SudokuFrame() {

		solver = new SudokuSolver();

		setTitle("Sudoku");
		setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		setLayout(new BorderLayout());

		add(generateToolBar(), BorderLayout.NORTH);

		centerPanel = new JPanel();
		FlowLayout flow = new FlowLayout();
		flow.setHgap(1);
		flow.setVgap(1);
		centerPanel.setLayout(flow);

		add(centerPanel, BorderLayout.CENTER);
		rearrangeGrid();

		setVisible(true);
	}

	protected void rearrangeGrid() {
		int size = solver.getWidth();
		int preferredSize = (50 + 2) * size;
		setSize(new Dimension(preferredSize, preferredSize + 80));
		centerPanel.removeAll();
		for (int i = 0; i < size * size; i++) {
			NumberButton nb = new NumberButton(i);
			nb.addActionListener(new ActionListener() {
```

```
                                @Override
                                public void actionPerformed(ActionEvent e) {
                                        String input = JOptionPane.showInputDialog(
                                                        SudokuFrame.this, "Input Number For th
is val");
                                        try {
                                                int val = Integer.parseInt(input);
                                                int size = solver.getWidth();
                                                NumberButton button = (NumberButton) e.getSour
ce();
                                                int index = button.getIndex();
                                                solver.setNumber(index / size, index % size, v
al);
                                                button.setText(input);
                                        } catch (IllegalStateException exception) {
                                                JOptionPane.showMessageDialog(SudokuFrame.this
,
                                                                "Please reset to start new gam
e", "Need Reset",
                                                                JOptionPane.ERROR_MESSAGE);
                                        } catch (IllegalArgumentException exception) {
                                                JOptionPane.showMessageDialog(SudokuFrame.this
,
                                                                "Invalid Number", "Invalid Num
ber",
                                                                JOptionPane.ERROR_MESSAGE);
                                        } catch (Exception exception) {
                                                JOptionPane.showMessageDialog(SudokuFrame.this
,
                                                                "Invalid Number", "Invalid Num
ber",
                                                                JOptionPane.ERROR_MESSAGE);
                                        }
                                }
                        });
                        centerPanel.add(nb);
                }
                centerPanel.doLayout();
        }

        protected JToolBar generateToolBar() {
                JToolBar toolBar = new JToolBar();
                toolBar.add(new AbstractAction("Setting") {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                String input = JOptionPane.showInputDialog(SudokuFrame.this,
                                                "Input Size(2-10)");
                                try {
                                        int value = Integer.parseInt(input);
                                        if (value < 2 || value > 10)
                                                throw new IllegalArgumentException();
                                        solver.setSize(value);
                                        SudokuFrame.this.rearrangeGrid();
                                } catch (NumberFormatException ex) {
                                        JOptionPane.showMessageDialog(SudokuFrame.this,
                                                        "Not a valid number!");
                                }
                        }
                });
                toolBar.add(new AbstractAction("Solve") {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                final BlockDialog block = new BlockDialog(SudokuFrame.this);
```

```
                                // Create a thread to execute

                                Thread thread = new Thread() {
                                        public void run() {
                                                try {
                                                        final Map<String, Integer> values = so
lver.solve();
                                                        SwingUtilities.invokeAndWait(new Runna
ble() {
                                                                public void run() {
                                                                        int val = solver.getWi
dth();
                                                                        val *= val;
                                                                        for (int i = 0; i < va
l; i++) {
                                                                                NumberButton n
b = (NumberButton) centerPanel
.getComponent(i);
                                                                                int result = v
alues.get("a" + i);
                                                                                nb.setText(Str
ing.valueOf(result));
                                                                        }
                                                                }
                                                        });
                                                } catch (IllegalStateException e) {
                                                        JOptionPane.showMessageDialog(SudokuFr
ame.this,
                                                                        "Please reset to start
 a new game");
                                                } catch (Exception e) {
                                                        JOptionPane.showMessageDialog(SudokuFr
ame.this,
                                                                        "Cannot Solve this. Pl
ease reset.");
                                                } finally {
                                                        SwingUtilities.invokeLater(new Runnabl
e() {
                                                                public void run() {
                                                                        block.setVisible(false
);
                                                                }
                                                        });
                                                }
                                        }
                                };
                                thread.start();

                                block.setVisible(true);

                        }
                });
                toolBar.add(new AbstractAction("Reset") {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                solver.reset();
                                SudokuFrame.this.rearrangeGrid();
                                SudokuFrame.this.repaint();
                        }
                });
                return toolBar;
        }
```

```
        @Override
        public void print(Graphics g) {
                super.print(g);
        }
}
```

```java
package ui;

import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class BlockDialog extends JDialog {

	/**
	 *
	 */
	private static final long serialVersionUID = -7159941446081118846L;

	public BlockDialog(JFrame parent) {
		super(parent);
		setSize(new Dimension(200, 100));
		setTitle("Processing...");
		setLocation(parent.getLocation().x + parent.getSize().width / 2
				- getSize().width / 2,
				parent.getLocation().y + parent.getSize().height / 2
					- getSize().height / 2);
		setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		setResizable(false);
		setUndecorated(true);
		setModal(true);
		setModalityType(ModalityType.APPLICATION_MODAL);

		setLayout(new FlowLayout());
		add(new JLabel("Processing...."));
		doLayout();
	}

}
```

```
package model;

import java.io.PrintStream;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

import yices.ResultParser;
import yices.RuleGenerator;

public class SudokuSolver {

	private boolean ready = true;

	private int size = 3;

	private int[] values = new int[3 * 3 * 3 * 3];

	public int getSize() {
		return size;
	}

	public void setSize(int size) {
		this.size = size;
		reset();
	}

	public void setNumber(int i, int j, int val) {
		if (!ready)
			throw new IllegalStateException();
		if (val > size * size || val <= 0)
			throw new IllegalArgumentException();
		values[i * size * size + j] = val;
	}

	public void reset() {
		values = new int[size * size * size * size];
		for (int i = 0; i < values.length; i++)
			values[i] = 0;
		ready = true;
	}

	public Map<String, Integer> solve() {
		if (!ready)
			throw new IllegalStateException();
		try {
			Map<String, Integer> result = new HashMap<String, Integer>();

			Process p = Runtime.getRuntime().exec("yices -e");

			RuleGenerator rg = new RuleGenerator(getSize());
			int s = getWidth();
			rg.generateGlobalRules(new PrintStream(p.getOutputStream()));
			for (int i = 0; i < values.length; i++) {
				if (values[i] != 0) {
					rg.generateGivenRules(i / s, i % s, values[i]);
				}
			}
			rg.check();
			p.getOutputStream().write("(exit)\n".getBytes());
			p.getOutputStream().flush();
			Thread.sleep(1000);
```

```
                        ResultParser parser = new ResultParser();
                        parser.parse(p.getInputStream());
                        Thread.sleep(1000);

                        try {
                                p.exitValue();
                        } catch (IllegalStateException e) {
                                e.printStackTrace();
                                p.destroy();
                        }
                        ready = false;
                        if (null == parser.getValues())
                                throw new RuntimeException("Cannot Solve");
                        for (Entry<String, String> entry : parser.getValues().entrySet()) {
                                result.put(entry.getKey(), Integer.valueOf(entry.getValue()));
                        }
                        return result;
                } catch (Exception e) {
                        if (e instanceof RuntimeException)
                                throw (RuntimeException) e;
                        throw new RuntimeException(e);
                }
        }

        public int getWidth() {
                return getSize() * getSize();
        }
}
```