Experiment 1

September 30, 2022

1 Environment configuration

1. Please complete this experiment and subsequent experiments in the virtual machine image provided in this course. You can follow the instructions provided on canvas. Use the provided tutorial to download the virtual machine image and import it into your VMware Workstation.

2. After entering the virtual machine, please download *lab1-release.zip* from canvas , put it on the desktop and unzip it, and then enter lab1 folder.

3. Open the command line in the lab1 folder and enter ./install_minisat.sh in the command line to install all dependencies. This script will help you install all dependencies and tools required for experiment 1.

2 Experimental content

Through this experiment, we hope that students can actually use the SAT solver to solve some simple problems. We use MiniSat, which is an open source SAT solver. You can find its source code on the Internet. (But this is not important , we just hope that you can solve actual problems through the SAT solver. If you are very interested in its internal implementation, you can try to read its source code).

2.1 MiniSat

We will briefly introduce to you how to use **MiniSat .** Before that, please read *minisat.cpp* carefully.

content, this example program solves the proposition P = (¬A ÿ ¬B ÿ C) ÿ (¬A ÿ ¬B ÿ ¬C) ÿ (A ÿ ¬B ÿ C)

Simply understand each line of code through comments, which will help you understand how to use **MiniSat .**

First, MiniSat uses disjunctive normal form (CNF) as its input. **MiniSat** is used to solve propositional logic tables.

Expression, first we need to initialize a Solver object, that is, in *minisat.cpp*

```
1  // Create a solver
2  Solver solver;
```

Listing 1: Initializing the Solver object

Then we need to define some propositional variables. Here we use the newVar function provided by **MiniSat ,** such as

In *minisat.cpp* , we define literals A, B, C:

```
1  // Create variables
2  auto A = solver.newVar();
3  auto B = solver.newVar();
4  auto C = solver.newVar();
```

Listing 2: Defining text

Next we need to add clauses to the solver. Here we use mkLit and addClause provided by **MiniSat .**

Functions, mkLit can convert proposition variables into the format required by clauses, and addClause can add subclauses to the solver.

sentence. For example, in *minisat.cpp* , you can see that we added three clauses:

```
1  //Add the clauses
2  // (~A v ~B v C)
3  solver.addClause(~mkLit(A), ~mkLit(B), mkLit(C));
4  // (~A v ~B v ~C)
5  solver.addClause(~mkLit(A), ~mkLit(B), ~mkLit(C));
6  // (A v ~B v C)
7  solver.addClause(mkLit(A), ~mkLit(B), mkLit(C));
```

Listing 3: Adding clauses

Next, we formally call the solve function to solve. The return value of the function is a bool type, marking this

Whether a proposition has a set of explanations that make it true:

```
1  // Solve the problem
2  auto sat = solver.solve();
```

Listing 4: Solve

If the return value is true, we can get an explanation through the modelValue function. If the return value is false, we

Directly output "UNSAT":

```
// Check solution and retrieve model if found
if (sat)
{
    std::clog << "SAT\n"
              << "Model found:\n";
    std::clog << "A :=                    " << (solver.modelValue(A) == l_True) << '\n';
    std::clog << "B := std::clog          " << (solver.modelValue(B) == l_True) << '\n';
    << "C :=                              " << (solver.modelValue(C) == l_True) << '\n';
    return 0;
}
else
{
    std::clog << "UNSAT\n";
    return 1;
}
```

Listing 5: Output results

(Tip, in the experiment you want to complete, you cannot have any output using printf, cout or clog, you need

What we need to do is to pass the results out as function parameters so that we can test your program.)

Finally, you can open the command line in the lab1 directory, enter make example to run this example and see the output

The result is as follows:

```
SAT
Model found:
A := 0
B := 0
C := 0
```

2.2 Problem description

We hope you can use **MiniSat** to solve a practical problem. The problem is described as follows:

There are n stones in the river. Tom and Jerry want to cross the river by stepping on the stones. These stones have two states, either floating.

On the surface of the water, or at the bottom. There are m switches on the riverside to control the sinking and floating of stones. Each switch controls 1 or 2 stones,

Each stone is controlled by 1 or 2 switches. Every time you turn a switch on or off, the stone controlled by the switch changes

state, floating from water to water or sinking from water to water.

At the beginning, all switches are turned off, and some rocks are above the water and some are below the water. Please use **MiniSat** to determine if

How to control these switches to make all the stones float on the water at the same time.

2.3 Code framework

2.3.1 Input

The input data is in *test.txt* . The integer in the first line is the number of test cases, followed by all test cases,

Separate one line between different test cases.

The first line of each test case is two numbers m and n. m is the number of switches and n is the number of stones. next line

It is n numbers, corresponding to the initial state of n stones. 0 means that the stone is underwater, and 1 means that the stone is on the water. Take it

The next m lines contain information for m switches. The first number in each row indicates how many stones this switch controls, followed by the number of the stone, starting from 1.

2.3.2 Output

The framework code will output the results to *answer.txt* , with the results of each test case occupying one line. Each line

contains several 0 or 1, 0 means that the corresponding switch should be turned off, 1 means that the corresponding switch should be turned on, this This way every stone will float on the water. If there is no way to cross the river, there will be only one UNSAT in this row.

2.3.3 Examples

If the content of test.txt is as follows:

1

twenty two

1 0

1 1

1 2

The first line indicates that there is 1 test case. The second row indicates that there are 2 switches and 2 stones. Line 3 indicates that at the beginning, stone No. 1 is on the water and stone No. 2 is under the water. Line 4 indicates that the first switch controls one stone, which is stone No. 1, and the second switch controls one stone, which is stone No. 2.

The answer will be output to *answer.txt* with the following content.

0 1

This means that the first switch should be turned off and the second switch turned on.

2.3.4 Implementation requirements

main.cpp has completed the file reading and answer output functions. You need to complete the function lab1 in *lab1.cpp* . Please do not modify the code except function lab1. Function

lab1 includes 5 parameters.

- n is the number of stones.
- m is the number of switches.
- The array states is the initial state of n stones. The array length may be greater than n, and the extra part can be ignored.
- Button is a two-dimensional array that records the control information of the switch, where each element is the number of the stone. For example, button[3][0] and button[3][1] represent the stone number controlled by the 4th switch. Stones are numbered starting from 1. If button[3][0] = 1 and button[3][1] = 0, then this switch only controls stone No. 1.
- Inside the array answer is the answer to the question. For example, answer[2] = true means the 3rd switch should be on.

- The return value type of the lab1 function is bool. If it returns true, it means that there is a way to make all the stones float on the water at the same time.

  If false is returned, it means that Tom and Jerry cannot cross the river.

It is assumed that the values of these parameters are legal.

After completion, please open the command line under the lab1 folder, enter make lab1 to compile and run the code, and run the result.

After finishing, please go to *answer.txt* to check the answer.

Please note that we do not provide correct answers for test cases, and the correct answers for some test cases may not be unique.

3 Homework Submission

Open the command line under the lab1 folder and enter make handin. A *lab1.zip* compressed package will be generated with the

After saving your *lab.cpp* file, please submit it to canvas before the deadline. Please note that if you violate

assignment-related requirements, points will be deducted as appropriate.