Combinatorial Problem Solving (CPS)

Project: Logic Synthesis.

Updated: March 7, 2019

1 Description of the Problem

In digital circuit theory, a *logical circuit* has a given number of inputs and one output. By assigning 0/1 signals to the inputs, a 0/1 signal is obtained at the output, which is a Boolean function of the input signals.

Given a specification of a logical circuit, e.g., by means of the truth table of the output as a function of the inputs, there are several ways to implement the circuit physically. In this project we consider the problem of synthesizing circuits built up of *NOR gates* (henceforth referred to as *NOR-circuits*). A NOR gate is a device with two inputs x_1 and x_2 and one output y that behaves as described in the truth table below:

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	0

In other words, the output has signal 1 if and only if neither of the inputs has signal 1.

It is not difficult to see that any logical circuit can be implemented as a NOR-circuit. For example, in Figure 1 we can see the truth table of the AND function of x_1 and x_2, and a NOR-circuit that implements it. Notice that, in addition to x_1 and x_2, we also allow that some of the circuit inputs are constant 0 signals.

Another example can be seen in Figure 2. Note that input signals as well as constant 0 signals may be repeated, and not all input signals need to be used (input signal x_1 is not used in this example).

The *depth* of a NOR-circuit is the maximum distance (i.e., the number of gates in the path) between any of the inputs and the output. This is an important parameter, as the time that the circuit needs to compute the

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

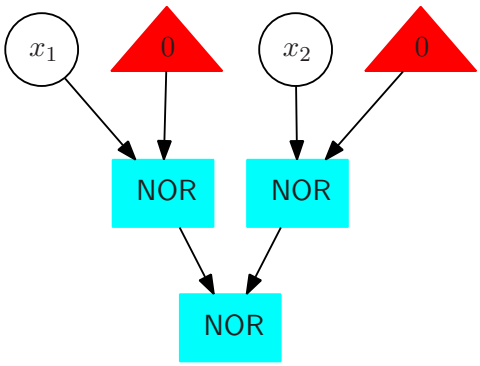

Figure 1: Truth table of $y = \mathsf{AND}(x_1, x_2)$ and NOR-circuit implementing it.

x_1	x_2	x_3	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

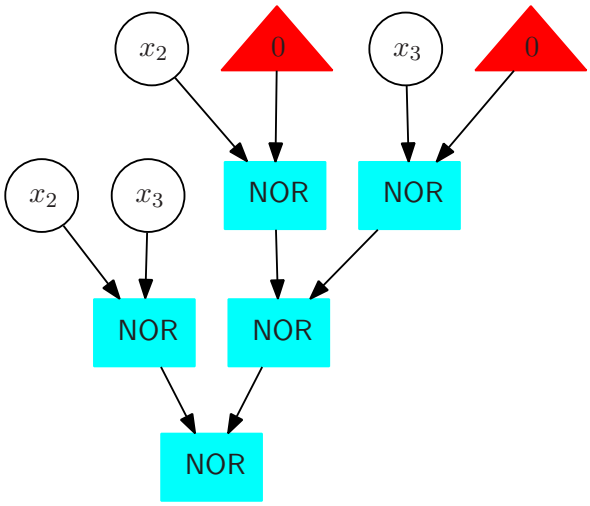

Figure 2: Truth table of a function and NOR-circuit implementing it.

output signal when given the input signals is proportional to this value. For example, the depth of the circuit of Figure 1 is 2, as the paths from the inputs to the output all cross 2 NOR gates. Similarly, the depth of the circuit of Figure 2 is 3.

Another relevant parameter related to the performance of the circuit is its *size*, i.e., the total number of NOR gates: the more gates, the larger area the circuit needs and the more power it consumes. For example, the NOR circuit in Figure 1 has size 3, while the one in Figure 2 has size 5.

In this project our goal is to solve the *NOR Logic Synthesis Problem (NLSP)*: given a specification of a Boolean function $f(x_1, ..., x_n)$ in the form

of a truth table, to find a NOR-circuit satisfying the specification that minimizes depth (and, in case of a tie in depth, with minimum size).

In order to simplify the problem, several assumptions are made:

- Only NOR gates with 2 inputs and 1 output can be used: more general NOR gates with more inputs are not allowed (i.e., the *fan-in* of NOR gates is always 2).
- The output of a NOR gate can only be the input of a single gate: outputs cannot be reused as inputs of more than one gate (i.e., the *fan-out* of NOR gates is always 1).
- In addition to the input signals of the Boolean function to be implemented, constant 0 signals can also be used as inputs of NOR gates. Constant 0 signals as well as input signals can be used as many times as needed as inputs of NOR gates. On the other hand, the circuit does not need to use all input signals. Similarly, the constant 0 signal does not have to be used if it is not needed. See Figures 1 and 2.

2 Input and Output Formats

This section describes the format in which instances of NLSP are written (Sect. 2.1), as well as the format for the corresponding solutions (Sect. 2.2).

2.1 Input Format

An instance of NLSP consists of several lines of integer values (Booleans are represented with 0/1 as usual). The first line consists of the number of input signals n of the Boolean function $f(x_1, \ldots, x_n)$ to be implemented (where $n \geq 2$). Then 2^n lines follow, which describe the truth table of f. For each $0 \leq i < 2^n$, the i-th of these lines has the following form. Let $\alpha_{i_1}\ \alpha_{i_2}\ \cdots\ \alpha_{i_n}$ be the binary representation of i with n bits, and let $\beta_i = f(\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_n})$. Then the i-th line contains the value β_i.

As an example, the instance that corresponds to the function AND(x_1, x_2) of Figure 1 is:

```
2
0
0
0
1
```

And the instance that corresponds to the function of Figure 2 is:

```
3
0
1
1
0
0
1
1
0
```

2.2 Output Format

The output starts with a copy of the input data. Then there is a line with the integer values d and s, respectively the depth and the number of NOR gates of an optimal circuit, separated with a blank space.

Then several lines describing each of the nodes of the circuit follow. Here by node we mean a NOR gate, an input signal or a constant 0 signal.

Now, each line that describes a node should have the following format:

`<id>` ␣ `<code>` ␣ `<left>` ␣ `<right>`

where ␣ represents a blank space and:

- `id` is an integer identifier of the node, which should range from 1 to $2^{d+1} - 1$ (note that a circuit with depth d can have at most $2^{d+1} - 1$ nodes). The node whose output is the result of the Boolean function should have identifier 1.
- `code` is an integer code meaning:
  - -1: NOR gate
  - 0: constant 0 signal
  - i (with $1 \leq i \leq n$): input signal x_i
- `left` is the identifier of the node whose output is the left input of the node under consideration. If the node is not a NOR gate, and hence it has no inputs, then `left` is equal to 0 (which is an invalid node identifier).
- `right` is the same but for the right input.

For instance, the NOR-circuit in Figure 1 is optimal, and the output could be:

```
2
0
0
0
1
2 3
1 -1 2 3
2 -1 4 5
4  1 0 0
5  0 0 0
3 -1 6 7
6  2 0 0
7  0 0 0
```

For the function in Figure 2 the circuit in Figure 2 is optimal, and the output could be:

```
3
0
1
1
0
0
1
1
0
3 5
1 -1 2 3
2 -1 4 5
4 2 0 0
5 3 0 0
3 -1 6 7
6 -1 12 13
12 2 0 0
13 0 0 0
7 -1 14 15
14 3 0 0
15 0 0 0
```

3 Project

The purpose of this project is to model and solve NLSP with the three problem solving technologies considered in the course: constraint programming (CP), linear programming (LP) and propositional satisfiability (SAT).

For the development of the project, in addition to this document, students will be provided with the following materials:

- a suite of *problem instances* (in the format specified in Sect. 2.1). Files are named `nlsp_`n`_`β`.inp`, where n is the number of input signals of the circuit and β is the decimal representation of the truth table.
- a *checker* that reads the output of solving a problem instance (following the format given in Sect. 2.2) from `stdin`, verifies that the circuit fulfills its specification[1] and plots it in PNG format. Use option `-h` or `--help` to see all available options.

  Note: GraphViz tools are required for the visualization. Although this software is already installed in the lab machines, it can also be easily installed in personal laptops. E.g., in Ubuntu, one just needs to type:

  ```
  sudo apt-get install graphviz
  ```

- a *result table* with the depth and size of the optimal solution of some of the problem instances, to make debugging easier.

As a reference, solving processes that exceed a limit of 60 seconds of wall clock time should be aborted (Linux command `timeout` may be useful). Take into account that, depending on the solving technology, on the machine, etc., some of the instances may be too difficult to solve within this time limit. For this reason, it is not strictly necessary to succeed in solving all instances to pass the project. However, you are encouraged to solve as many instances as possible.

There are three deadlines, one for each problem solving technology:

- **CP:** 7 May.
- **LP:** 4 Jun.
- **SAT:** 25 Jun.

[1] The checker does **not** check that the circuit is optimal.

For each technology, a `tgz` or `zip` compressed archive should be delivered via Racó (`https://raco.fib.upc.edu`) with the following contents:

- a directory `out` with the output files of those instances that could be solved. Please only provide the outputs of those instances for which an optimal solution could be found. The output file corresponding to an instance `nlsp_`n`_`β`.inp` should be named `nlsp_`n`_`β`.out`.

- a directory `src` with all the source code (C++ programs, scripts, `Makefile`, etc.) used to solve the problem, together with a `README` file with basic instructions for compiling and executing so that results can be reproduced.

  Programs should read the instance from `stdin` and write the solution to `stdout`. Other output (debugging information, etc.) should be written to `stderr`. In particular, suboptimal solutions should not be written to `stdout`, only the optimal solution (if found).

  If say your executable is named `p`, then your results will be recomputed by calling `p < nlsp_`n`_`β`.inp` .

- a document in PDF describing the variables and constraints that were used in the model, as well as any remarks or comments you consider appropriate.

Please follow these indications when submitting your deliveries.