Combinatorial Problem Solving (CPS)

Project: Logic Synthesis.

Updated: March 26, 2017

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.

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.

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 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 inputs to the output all cross 2 NOR gates.

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

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

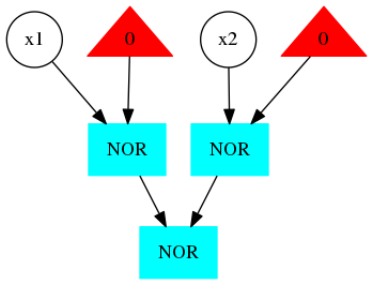

Figure 1: Truth table of $y = \mathsf{AND}(x_1, x_2)$ and NOR-circuit implementing it.

the circuit needs and the more power it consumes. The NOR circuit in the example has size 3.

In this project our goal is to solve the *NOR Logic Synthesis Problem (NLSP)*: given a value d and a specification of a Boolean function $f(x_1, ..., x_n)$ in the form of truth table, to determine whether or not there exists a NOR-circuit that fulfills the specification whose depth is at most d; and if so, find the one that minimizes the 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 used as inputs of more than one gate (i.e., the *fan-out* of NOR gates is always 1).
- In addition to the input variables 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 variable signals can be used as many times as wanted as inputs of NOR gates (see Figure 1).
- The Boolean function to be implemented requires at least one NOR gate (i.e., it is neither the constant 0 function, nor any function of the form x_i for $1 \leq i \leq n$).

2 Input and Output Formats

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

(Section 2.2). **Programs should read from `stdin` and write to `stdout`.**

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 contains the maximum depth d of the circuit (where $d \geq 1$). The next line consists of the number of input variables 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 $\alpha_{i1}\ \alpha_{i2}\ \cdots\ \alpha_{in}$ be the binary representation of i, and let $\beta_i = f(\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{in})$. Then the i-th line contains the value β_i.

As an example, the instance that corresponds to the function $\mathsf{AND}(x_1, x_2)$ of Figure 1 with maximum depth $d = 2$ is:

```
2
2
0
0
0
1
```

2.2 Output Format

The output should start with a copy of the instance in the input format. Then additional lines follow, according to these cases:

- For instances for which no solution exists, a single line with the integer value `-1`.
- For instances for which there is a solution, a line with the integer value `<size>`, the number of NOR gates of the optimal circuit.

  Then several lines describing each of the nodes of the circuit should follow. Here by node we mean a NOR gate, an input variable 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 between 1 and $2^{d+1}-1$. 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 variable 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 not a valid node identifier).
- `right` is the same but for the right input.

For instance for the NOR-circuit in Figure 1, which is optimal for maximum depth $d = 2$, the output should be:

```
2
2
0
0
0
1
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 same Boolean function but maximum depth $d = 1$, since there is no circuit the output would be:

```
1
2
0
0
0
1
-1
```

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 realization 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 Section 2.1). Files are named `nlsp_`d`_`n`_`β`.inp`, where d and n are the maximum depth and the number of input variables of the circuit respectively, 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 Section 2.2) from `stdin`, verifies that the circuit (if it exists) indeed 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 this. 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
  ```

These materials will be available at the webpage of the course:

`http://www.cs.upc.edu/~erodri/cps.html`

As a reference, solving processes that exceed a time limit of over 100 seconds 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 with all technologies.

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

- **CP:** 25 Apr.
- **LP:** 30 May.
- **SAT:** 13 Jun.

[1] Except that the size of 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 processed successfully (with a positive or with a negative answer). The output file corresponding to an instance `nlsp_`d`_`n`_`β`.inp` should be named `nlsp_`d`_`n`_`β`.out`.
- a directory `src` with all the source code (C++ programs, scripts, etc.) used to solve the problem, together with a `README` file with basic instructions for compiling and executing (so that results can be reproduced).
- a document in PDF describing the variables and constraints that were used in the model.