Logic Synthesis using Constraint Programming

Wilmer Uruchi Ticona

May 7, 2019

1 Approaches

So we start with 2 lines of thought. The first is that we should start with a tree of an adequate depth according to the number of variables of the input, if the input has n variables, then the tree must have depth $d = n+2$, and the number of nodes should be equal to $2^{(d-1)}$, then the combinatorial solver must try to solve the logic synthesis by selecting nodes to use and nodes to discard; however, this approach is discarded due to the high time and space complexity that it requires, and since we are constrained by a maximum execution time of 60 seconds, and that we are also trying to solve as many instances as possible.

The second approach is more moderate. It tries to only use the minimum number of constraints possible per instance, to optimize the execution time. Nevertheless, it requires that some code should be specifically written for each combination of depth d and number of nodes n, a potential $|d \times n|$ number of cases.

Let's use the second approach in this case.

2 Model

2.1 Variables:

General variables:

$numberRows$ = number of rows in the truth table of the input.
$numberVars$ = number of variables in the input.
$numberElements = 30$, works as an upper bound for the number of items of the tree.
$numberNodes$ = number of nodes that should be used, where a node is a NOR gate.
$depth$ = required depth of the circuit.
$numberItemsTree = numberElements + numberNodes$

Gecode Variables:

c = array of integers that defines each item in the circuit tree. The size of the array is equal to $numberItemsTree$.

indicator = array of booleans that indicates which set of values c is taking, and how should c be labeled according to that set of values. The array has a size of $numberItemsTree$ by $numberVars + 1$.

body = array of booleans that store the values c according to the selected sets. This variable is used to validate the NOR gate, or variable. The array has a size of $numberItemsTree$ by $numberRows$.

2.2 Model:

3 tuples are created, we represent the set of tuples as T, each $t \in T$ represents the sequence of values of the variables V of the truth table, t_0 always represents the values of the constant 0. In the case of 3 variables, an extra t_3 is created for the corresponding values.

body is ordered as a matrix $matBody$ of width $m = numberItemsTree$ and height $n = numberRows$, where each column represents an item in the circuit tree, and every row a truth assertion from the truth table. Also *indicator* is ordered as matrix $matIndicator$ of width $m = numberItemsTree$, and height $n = numberVars + 1$, where each column represents an item in the circuit tree, and each row represent a possible $v \in V$ assignment for that item, including the constant 0 assignment.

Using the **extensional** constraint we specify that each column in $matBody$ can take its values from one of the $t \in T$. Whenever a column takes its value from one of these t, the corresponding column of $matIndicator$ (that has the same index) sets value 1 to one of its rows, signaling in this way that the item is taking its values from a $v \in V$ or the constant 0, we can see that this is not enforced, but works with **half reification** because the item can take none of these sets of values, but be a NOR gate. In this case, if we are dealing with 3 variables, an extra column is taken into consideration. **We can easily increase the number of variables included in this way, but since the results.txt only has instances with up to 3 variables, it is left like that.**

Then, we traverse each item c_i in c, if the corresponding column i in $matIndicator$ signals that a variable $v \in V$ has been assigned to that column, c_i takes the value (in the case of 2 variables from $\{0, 1, 2\}$), that identifies it as a variable or the constant 0.

Until this point there is nothing enforcing that the values in c, $matBody$, $matIndicator$ should take any particular value. To fix this we are going to build these constraints according to the depth d and number of gates n send to the solver. This comes from the way our main function is set. In our main function we are performing a search from the most basic setup to the most complex; from example we start by looking at a solution that has $d = 0$ and $n = 0$, that is, no NOR gates, and only 1 variable $vinV$ is solving the logic synthesis. Then we increase d and n progressively in look for a solution that uses the minimum depth and quantity of NOR gates.

We are going to give only some examples, and let the reader infer the rest. From these examples it can be seen that there is a way to auto generate each case, thus, avoiding

the need to write each case manually. However, the added complexity did not seem very attractive for this particular case, specially given that our benchmark consisted of instances of at most 3 variables.

$depth = d = 0$, $numberNodes = n = 0$: The sum of the values of the column of index 0 of $matIndicator$ should be equal to 1, meaning that c_0 should take its value from one of $vinV$ or the constant 0, thus solving the circuit.

$d = 1$, $n = 1$ (obvious): The sum of the values of the columns of index 1 of $matIndicator$ should be equal to 1, same for the values of column 2, meaning that c_1 and c_2 should take their values from one of $vinV$ or the constant 0, the constant can be repeated for both columns, or any of v can be repeated for the columns. The column 0 will be the result of column 1 NOR column 2, applied in a sequential way to comply with the truth table. Given that column 0 is the result of a NOR operation, it is a **NOR gate**, thus c_0 will have the value -1 indicating it.

$d = 2$, $n = 2$: The sum of the values of the columns of index 2 of $matIndicator$ should be equal to 1, same for the values of column 3, 4, meaning that c_2, c_3, c_4 should take their values from one of $vinV$ or the constant 0, the constant can be repeated for both columns, or any of v can be repeated for the columns. The column 0 will be the result of column 1 NOR column 2, applied in a sequential way to comply with the truth table, column 0 will be the result of column 3 NOR column 4. Given that column 0 and 1 are the results of a NOR operation, they are **NOR gates**, thus c_0 and c_1 will have the value -1 indicating it.

Column 0 of $matBody$ will always be equal to the results of the truth table, it is the root of our circuit tree, completing in this way the constraints of the circuit.

Any c not enforced by a constraint (remember that we start with more $numberItemsTree$ than strictly necessary) will have the value -2 marking the limits of our circuit tree.

3 Attachments:

Together with this document, you should find:

1. /src/project.cpp : C++ source code file.
2. /src/p : C++ compiled.
3. /src/Makefile : Some paths should be changed depending on your environment.
4. /src/run.sh : Can be used to run p in several instances in a single instruction.
5. /src/README.txt : File with compiling instructions.
6. output.txt: list of instances that the program could solve, those that are not optimal results are marked accordingly.
7. /out : Directory containing all the instances that are optimally solved by the program.

8. results.txt : Optimal results provided.