The basis of the constraint programming algorithm applied was backtracking, with constraint propagation. Depth first search was used as the backend to apply backtracking. The algorithm starts off with an empty puzzle. All permutations of the first row are generated and filtered. The filtering process here refers to constraint propagation. Constraint propagation is applied as follows:

1. Generate all permutation of row
2. For each row, check if the column constraint is broken. A column constraint is broken when that column does not contain that colour.
3. Return list of all feasible rows

The returned list of rows is used to set the states of the backtracking algorithm. Every time a row is set into the nonogram puzzle, the state of the nonogram is checked. Every column in the puzzle is checked. If the number of occurrences of any given colour is greater than the what the constraints dictate, then that row is removed. Figure 1 below displays this process. This checking is done every time a new row is added.

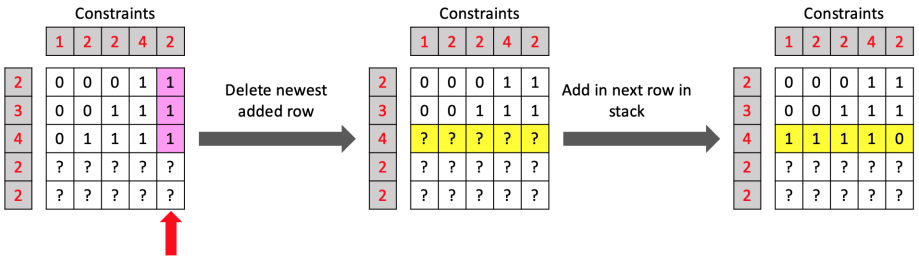

Fig. 1. Backtracking

The overall algorithm is shown below.

Algorithm 1 CSP Algorithm

```
stack = LIFO_stack
all_rows = generate_all_rows[row_index]
viable_rows = filter_rows(all_rows)
while stack != empty do
    row = stack.pop
    puzzle.insert_row(row)
    check = puzzle.check_constraints
    if check == False then
        puzzle.delete_row(row) → continue
    while index < puzzle.height do
        stack.push(generate_all_rows(index)
```