CS402 coursework 2 Report

20140515 Sungyoon Jeong

April 8, 2017

1 CNF

Basically, this CNF transformer follows algorithm from the lecture note. However, the case of $\equiv$ and $\leftarrow$ was not introduced in the lecture. Therefore, I added eqfree() and reverseimplfree() function. The role of eqfree() function is to transform $A \equiv B$ into $(A \rightarrow B) \wedge (B \rightarrow A)$, and the role of reverseimplfree() function is to transform $A \leftarrow B$ into $B \rightarrow A$. Other functions work just as the same as introduced in the lecture notes.

2 Nonogram solver

In the code, the logical formula contains some literals, written in formatted strings such as "Ri_j_k". Here are description of each of them:

- $R_{i,j,k}$: the j-th chunk from i-th row starts from k-th column.
- $C_{i,j,k}$: the j-th chunk from i-th column starts from k-th row.
- $X_{i,j}$: the pixel in the row i, column j (True means it is filled)

Following rules are used to construct the logical formula (Note: *row* and *col* means number of rows, and number of columns, correspondingly.):

1. The available starting point of a chunk is limited. Each chunk has its available leftmost position and rightmost position. This could be written as,

$$(\bigvee_{k=lm}^{rm} R_{i,j,k}) \wedge (\bigwedge_{0 \leq k < lm \text{ } or \text{ } rm < k < col} \neg R_{i,j,k})$$

   where lm, rm stands for leftmost, and rightmost position.

2. If j-th chunk starts from column k, then the next chunk cannot start less than {k + length of j-th chunk}. This could be written as, $R_{i,j,k} \rightarrow \{(\neg R_{i,j+1,0}) \wedge (\neg R_{i,j+1,1}) \wedge \ldots \wedge (\neg R_{i,j+1,k+l})\}$ where l is length of the j-th chunk.

3. Starting point of a chunk is unique - for fixed i, j, if $R_{i,j,k}$ is true, then no other l is able to make $R_{i,j,l}$ true. This could be written as, $R_{i,j,k} \rightarrow (\bigwedge_{n=0,n\neq k}^{col-1} \neg R_{i,j,n})$.

4. If j-th chunk starts from column k, then corresponding $X_{i,j}$ should be true. This could be written as, $R_{i,j,k} \rightarrow \{X_{i,k} \wedge X_{i,k+1} ... \wedge X_{i,k+l-1}\}$ where l is length of the j-th chunk.

   Similarly, rule 1 to 4 also applies to columns ($C_{i,j,k}$).

5. If a pixel is assigned to true, then there should be at least one corresponding row, and column chunk rules that is assigned to true. In other words,

$$X_{i,j} \rightarrow (\bigvee_{\forall \alpha, start \leq \beta < end} R_{i,\alpha,\beta}) \wedge (\bigvee_{\forall \alpha, start \leq \beta < end} C_{j,\alpha,\beta})$$

   where start is $max(0, j - length\ of\ \alpha th\ chunk + 1)$ and end is $min(col - length\ of\ \alpha th\ chunk, j)$ for the row's case. In the column's case, substitute j into i, col into row.

3 Usage and execution examples

Python version: 2.7

3.1 CNF

In the cnf/ directory, enter this to your shell:

$ python cnf.py "propositional_formula"

Here is the screenshot of executing cnf.py:

```
➜  cnf git:(master) ✗ python cnf.py "> & - p q & p > r q"
& | | p - q p | | p - q | - r q
(p | - q | p) & (p | - q | - r | q)
Valid
➜  cnf git:(master) ✗
```

3.2 Nonogram solver

In the nonogram/ directory, enter this to your shell:

$ python nonogram.py path_to_nonogram_input

Here is the screenshot of executing nonogram.py:

```
➜  nonogram git:(master) ✗ python nonogram.py bird.txt
...####..................
..######.............##..
..#.#####..........######
##############...#######.
..####################...
...####..##########......
....##..##########.......
.....#..############.....
.....##..##############..
......#...###########....
......##.......#.........
.......###..#####........
.........####...#........
...........#...##.....###
..........##...#..#######
.........##...###########
.........#..###########..
........###########......
......#########..........
.....#######.............
➜  nonogram git:(master) ✗
```