Assignment #: 1

Thejesh Venkata Arumalla and Sai Shashank Gaddam

September 13, 2022

1. **P1 - Atleast one is true**
   We need to add a new clause which is disjunction of all propositions

   $P_1 \vee P_2 \vee P_3 .... P_n$
   Code execution -

   - Compiling: g++ p1.cpp -o p1.out
   - Enter inputs in p1Input.txt
   - Execute: ./p1.out
   - Output will be in p1Output.dimacs
   - Test it by running: picosat p1Output.dimacs

2. **P2 - Atmost one is true**
   Notations

   - Let the clauses of the given CNF ϕ be $C_1, C_2 ... C_m$, there are m clauses.
   - Let the propositions of the given CNF be ϕ be $P_1, P_2, ... P_n$, there are n propositions.
   - If the output is a tautology, then I am representing it as $P_1 \vee \neg P_1$
   - If the output is a unsatisfiable, then I am representing it as $P_1 \wedge \neg P_1$

   Logic

   - We have n iterations once for each iteration.In each iteration we generate a literal or a constant bool(i.e. 0 or 1).
     - In the i^{th} iteration we set all propositions except P_i to be false i.e. $P_j = 0 \quad \forall j != i$
       and compute a value which is either 1 or 0 or P_i or $\neg P_i$. This is similar to unit substitutions.
     - Basically the above encodes the statement where all propositions except P_i are false, P_i can either be 1 or 0, satisfying the atmost 1 constraint.
   - Finally after all iterations we do a disjunction ($\vee$) of all the outputs of the above iterations.
     So final output has only one clause.
   - The size of the final output will be $O(n)$ as in each iteration we atmost get one literal and add it to our clause

   Code execution -

   - Compiling: g++ p2.cpp -o p2.out
   - Enter inputs in p2Input.txt
   - Execute: ./p2.out

- Output will be in p2Output.dimacs

Example test cases -

- Input: $P1 \vee P2 \vee P3 \implies$ Output: $P1 \vee P2 \vee P3$
- Input: $P1 \wedge P2 \implies$ Output: unsatisfiable as both needs to be one
- Input: $P1 \vee \neg P1 \implies$ Output: tautology as it always true

3. **P4 - Subgraph isomorphism**
Notations used in code -

- Let the graphs be H and G
- Let H_n,G_n denote the number of vertices in H and G
- Let H_m,G_m denote the number of edges in H and G.
- Our proposition variables are $X_{i,j}$ where it gets the value 1 if there is a mapping between i^{th} node in H with j^{th} node in G.

Logic -

- Every vertex in H must be mapped to atleast one vertex in G.
  - For every $i \in [1..H_n]$ add a clause $(X_{i,1}, X_{i,2}, X_{i,3}.....X_{i,G_n})$
- No two vertices in H can be mapped to same vertex in G.
  - For every two vertices $i, j \in [1..H_n]$ graph H and every vertex $k \in [1, 2..G_n]$ graph G add a clause
    i.e $X_{i,k}$ & $X_{j,k}$ can't be true $\implies \neg(X_{i,k}\ \&\ X_{j,k}) \implies \neg X_{i,k} \vee \neg X_{j,k}$
- A vertex in H can't be mapped to more than one vertex in G.
  - For every two vertices $k1, k2 \in [1..G_n]$ graph G and every vertex $i \in [1, 2..H_n]$ graph H add a clause
    i.e $X_{i,k1}$ & $X_{i,k2}$ can't be true $\implies \neg(X_{i,k1}\ \&\ X_{i,k2}) \implies \neg X_{i,k1} \vee \neg X_{i,k2}$
- Every edge in H should be mapped to some edge in $G \implies$ Every edge in H should **not be mapped to any non edge in G**.

Code execution -

- Compiling: g++ p4.cpp -o p4.out
- Enter graph inputs in graphH.txt for graph H and graphG.txt for graph G
- Execute: ./p4.out
- Output will be in p4Output.dimacs

References - `https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190/Mitarbeiter/toran/fin.pdf`