# Problem-set 04

Mostafa Touny

## Contents

lectures 8, 9 (skimmed) Sipser Ch. 1.2; and, 3.2, 3.3 on nondeterminism. Sipser first half of Ch. 7.4 (and also Ch. 5.3 if it helps).

## Ex. 1

### a

Obviously the problem has a polynomial-time verifier by specifying the range of the subsequence common among all $w_i$. It's easy to loop on that range $k$, Checking whether corresponding positions of all $w_j$ are the same.

### b

Obviously a verifier is some input $x_0$ such that $C_1(x_0) \neq C_2(x_0)$. It's easy to compute both circuits and check their unequal output.

## Ex. 2

Intuitively, If a problem has some varifier, Then we can brute-force all possible verifiers. Observe we can choose the greatest branching factor $c_{max}$ so that time is upper-bounded by $c_{max}^{poly(n)}$. Alternatively, Our argument might be seen through the perspective of a tree of an $NP$ problem, where $poly(n)$ is the time required of the longest path of the tree.

I am not aware of any more rigorous proof.

## Ex. 3

### a

For clarity and brevity we list all cases:

- unit clause $x_i$
  - delete any clause containing $x_i$
  - delete $\neg x_i$ from any clase
- unit clause $\neg x_i$
  - delete any clause containing $\neg x_i$
  - delete $x_i$ from any clause

Due to symmetry we mention only the case of unit clause $x_i$.

We prove the new propagated $C'$ is satisfiable if and only if the given $C$ is satisfiable.

($\leftarrow$) Necessarily $x_i = True$. Hence any clause containing $x_i$ is immediately evaluated to $True$ as well. Since $\neg x_i = False$ one of the remaining units of the clause containing $\neg x_i$ must be evaluated to $True$.

($\rightarrow$) Following the same reasoning, Since $x_i = True$, adding any clause containig $x_i$ is still going to be true regardless of other units boolean values. Also adding

$\neg x_i = False$ to any clause evaluated to $True$ won't change the whole clause's boolean value.

### b

We give a constructive polynomial-time algorithm. We follow the hint mentioned in the problem statement.

For case (ii), Loop on clauses, and for each:

- If no negative literal is assigned any value, Conveniently pick-up the first negative literal $\neg x_k$ and assign $x_k = False$.
- If a negative literal $\neg x_k$ is assigned $x_k = False$, Continue to the next clause.

Observe we only assign $x_k = False$. As a result, the case of a negative literal $\neg x_k$ assigned $x_k = True$ won't ever be encountered.

For case (i), Keep applying the process of $a$, Until all unit-clauses are eliminated. If any clause is empty, It's concluded the given $C$ is unsatisfiable. Now we know every clause contains at least two literals, Including a negative literal. Case (i) is now reduced to case (ii).

## Ex. 4

### a

Clearly, If literals $x_i$ which are assigned to 1 are even, Then they can re-arranged as pairs, Each yielding 0, and in turn all pairs yield 0. Observe for an even number, $2k$ mod $2 = 0$.

Similarly, if literals $x_i$ which are assigned to 1 are odd, Then we obtain 1 XOR $0 = 1$, by separating one literal from the remaining even literals.

### b

We define summation as, $\epsilon_1 + \epsilon_2 = c_1 + c_2 + 1$ where are $\sum_i x_{1i} = c_1 \, mod \, 2$ and $\sum_i x_{2i} = c_2 \, mod \, 2$. Note if $c_1 = c_2 = 1$, Then $c_1 + c_2 + 1 = 1 + 1 + 1 = 1 \, mod \, 2$ and hence $(\epsilon_1 + \epsilon_2)(x) = 1$. On the other hand, If $1 + c_2 + 1 = 1 \, mod \, 2$, Then $c_2 = -1 = 1 \, mod \, 2$. Hence, $\epsilon_2(x) = 1$.

### c

Consider $k_1$ to be the number of equations in which $x_{11}$ appreas.

$$\text{Update } x_{11} \text{ to } \left\{ \begin{array}{ll} 0, & \text{if } k_1 \text{ is even} \\ k_1 x_{11}, & \text{if } k_1 \text{ is odd} \end{array} \right\} \tag{1}$$

Remark in case $k_1 = 2m$ is even, Then $\sum_{j=1}^{2m} x_{1j} = 2(m x_{11}) = 0 \, mod \, 2$. That, Regardless of $x_{1j}$ values, as we would always obtain an even number.

Now consider the whole system of equations by summing all equations as we defined in $b$. In case $k_1$ is even, Then we know $x_{1j}$ evaluates to zero, and hence their removal from the system doesn't affect. In case $k_1$ is odd, Then $k_1 x_{11}$ is exactly equivalent to $x_{11} + x_{12} + \cdots + x_{1n}$. Think of redistributing $x_{1j}$ to reconstruct the original equations before the transformation, which clarifies why the new system is equivalent to the original one.

**d**

If at any stage $0 = 1$ is concluded, Then no matter what $x$ is inputted, The system won't be satisfied. Since the transformed system is equivalent to the original one, It's trivial why the original system is unsatisfiable.

Observe in *modulus 2*, the only possible evaluation outcomes are 0 or 1. So if $LHS \neq RHS$, then necessarily we get $0 = 1$. If we don't get $0 = 1$, then all equations' evaluations are satisfied.

Observe also if $k_{ii} x_{ii} = c \, mod \, 2$ then by our definitions we know $k_{ii}$ is odd. It follows there's exactly one unique solution for $x_i i$.

As the instructor hinted, back-substitution can be applied recursively to compute $x_{nn}$, $x_{n-1\,n-1}$, .., $x_{11}$ where once $x_{ii}$ is computed, For all the above equations, Their number of variables are reduced by one. Only one literal $x_{i-1\,i-1}$ is left for the next equation.

As a valid assignment $x$ for the transformed system is constructed, The original system is satisfied by $x$ also.