CPSC 421/501 Homework Solutions 9

(1) A word $\sigma_1 \ldots \sigma_n$ lies in L^* iff there are words $w_1, \ldots, w_N \in L$ such that

$$w_1 \circ w_2 \circ \ldots \circ w_N = \sigma_1 \ldots \sigma_n .$$

If so, then

$$(*) \begin{cases} w_1 = \sigma_1 \sigma_2 \ldots \sigma_{i_1} & \text{for some } i_1 \geq 1 \\ w_2 = \sigma_{i_1+1} \ldots \sigma_{i_2} & \text{for some } i_2 \geq i_1 + 1 \\ \vdots & \\ w_N = \sigma_{i_{N-1}+1} \ldots \sigma_{i_N} & \text{for some } i_N \geq i_{N-1} + 1 . \end{cases}$$

This happens iff there are integers $i_1, \ldots, i_N$ such that

$$1 \leq i_1 < i_2 < \ldots < i_{N-1} < i_N = n$$

such that the above equations $(*)$ hold.

Hence to decide whether or not

$\sigma_1 \cdots \sigma_n$ lies in L^*, we can "guess"

(by a non-deterministic T.m)

which of $2, \ldots, n$ should lie in

$\{i_1, i_2, \ldots, i_N\} \subset \{1, \ldots, n\}$, and then

run a non-deterministic algorithm,

M to check

① if $w_1 = \sigma_1 \cdots \sigma_{i_1}$ lies in L,

and

② if $w_2 = \sigma_{i_1+1} \cdots \sigma_{i_2}$ lies in L,

etc.

If M runs in time Cn^k on an input

of size n, then ①, ②, etc. above each,

decides in time $\le Cn^k$. Since $N \le n$

we require time $\le Cn^k n = Cn^{k+1}$

to test membership of each of

$w_1, \ldots, w_N$ in L. The overhead for guessing which of $1, 2, \ldots, n$ lie in $\{i_1, \ldots, i_N\}$, plus the overhead for setting up M to run on $w_1, \ldots, w_N$ (which is deterministic, given $i_1, \ldots, i_N$) take time polynomial in n. Hence our algorithm for testing membership in L^* lies in NP.

---

(2) Yes. The idea is to use the fact that for any i, m,

$$\sigma_i \sigma_{i+1} \cdots \sigma_{i+m} \in L^* \quad \text{iff}$$

$$\left(\left(\sigma_i \cdots \sigma_{i+m} \in L \right) \text{ or } \left(\begin{array}{l} \text{for some } 0 \le j \le m-1, \\ \sigma_i \cdots \sigma_{i+j},\ \sigma_{i+j+1} \cdots \sigma_{i+m} \in L^* \end{array} \right) \right)$$

we then apply this for $m=0$, then $m=1, \ldots$, remembering for which i,m we have $\sigma_i \ldots \sigma_{i+m} \in L^*$. Here are the details:

Consider the following algorithm, with $n-1$ phases:

Phase (1): Determine which of

$$\sigma_1, \sigma_2, \ldots, \sigma_n \in L^*$$

by determining for $i=1,\ldots,n$, which $\sigma_i \in L$ (since $\sigma_i \in L^*$ iff $\sigma_i \in L$).

Phase (2): Determine which of

$$\sigma_1\sigma_2, \sigma_2\sigma_3, \sigma_3\sigma_4, \ldots, \sigma_{n-1}\sigma_n \in L^*$$

Since

$$\sigma_i \sigma_{i+1} \in L^*$$

iff

$$(\sigma_i \sigma_{i+1} \in L) \quad \text{or} \quad (\sigma_i \in L^* \text{ and } \sigma_{i+1} \in L^*).$$

⋮

Phase (m): Determine which of

$$\sigma_i \sigma_{i+1} \cdots \sigma_{i+m} \in L^*$$

for $i = 1, 2, \ldots, n-m$, using the fact that this happens iff

$$\sigma_i \cdots \sigma_{i+m} \in L \quad \text{OR} \quad \text{for some } 1 \le j \le m-1,$$

$$\sigma_i \cdots \sigma_{i+j} \in L^* \text{ and}$$

$$\sigma_{i+j+1} \cdots \sigma_{j+} \in L^*.$$

⋮

This type of algorithm is known as "dynamic programming," probably first appeared in the work of

Rufus Isaacs and Richard Bellman, who were colleagues at RAND at the time (neither of whom acknowledged the other in their then classified technical reports).

Time complexity of the above algorithm: say that L can be decided in time Cn^k. Then

Phase (m) requires:

$(n+1-m)\,C\,m^k$ time to check membership in L

$+$

$(n+1-m)\,2(m-1)$ time to check membership conditions in L^*, assuming time 1 for each check (which on a Turing machine would take longer, but still polynomial time)

Estimating crudely, this would take time

$$(n+1-m)\, C m^k \le C n^{k+1}$$

AND

$$(n+1-m)\, 2(m-1) \left(\begin{array}{l}\text{time to check } L^* \\ \text{membership table}\end{array}\right)$$

$$\le 2n^2 \left(\begin{array}{l}\text{" \quad " \quad " \quad "} \\ \text{" \quad\quad "}\end{array}\right)$$

AND

some additional bookkeeping,

For a total (over $n-1$ phases of)

$$\le C n^{k+2} + 2n^3 \left(\begin{array}{l}\text{time to} \\ \text{check table}\end{array}\right) + \text{bookkeeping}$$

which polynomial in n.

(3) If a_1 or $a_2 = T$, then we can satisfy f by taking

$$z_1 = z_2 = \dots = z_{n-3} = F$$

since then

$$a_1 \vee a_2 \vee z_3 = T,$$

and every other term has the negation of one of $z_2, \dots, z_{n-3}$.

Similarly, if a_{n-1} or $a_n = T$, then we can satisfy f by taking

$$z_1 = z_2 = \dots = z_{n-3} = T.$$

Similarly, if $a_i = T$ with $3 \le i \le n-2$ we can satisfy f by taking:

$z_{i-2} = T$, $z_{i-1} = F$ which

makes $(\neg z_{i-2} \vee a_i \vee z_{i-1})$

true, and then taking

$z_1 = z_2 = \ldots = z_{i-2} = T$ and

$z_{i-1} = z_i = \ldots = z_{n-3} = F$.

Finally, if $a_1 \vee \ldots \vee a_n = F$, then

$a_1 = a_2 = \ldots = a_n = F$, and f reduces

to

$$f(z_1, \ldots, z_{n-3}) =$$

$$z_1 \wedge (\neg z_1 \vee z_2) \wedge (\neg z_2 \vee z_3) \wedge$$

$$\ldots \wedge (\neg z_{n-4} \vee z_{n-3}) \wedge (\neg z_{n-3})$$

Hence if $f(z_1, \ldots, z_{n-3}) = T$, then

z_1, $\neg z_1 \vee z_2$, $\ldots$, $\neg z_{n-4} \vee z_{n-3}$, $\neg z_{n-3}$

must all be T.

But then

$$z_1 = T$$

and

$$\neg z_1 \vee z_2 = T \text{ , hence } z_2 = T$$

and similarly

$$z_3 = T, \quad z_4 = T, \ldots, \quad z_{n-3} = T,$$

but then $\neg z_3 = F$. Hence f is not satisfiable.

Hence

at least one of $a_1, \ldots, a_n$ is T

$\Longleftrightarrow$

f is satifiable.

(4) Since f is satisfiable, one of

$$g(x_2, \ldots, x_n) = f(T, x_2, \ldots, x_n)$$

OR

$$\tilde{g}(x_2, \ldots, x_n) = f(F, x_2, \ldots, x_n)$$

is ~~s~~satisfiable. Hence if $SAT \in P$, say SAT is decidable in time Cn^k, then in time $\leq 2Cn^k$ (plus some polynomial time overhead) we find an $a_1 = T, F$, such that $f(a_1, x_2, \ldots, x_n)$ is satisfiable. Next we find $a_2 = T, F$ such that

$$f(a_1, a_2, x_3, x_4, \ldots, x_n)$$

is satisfiable, again in time $\leq 2Cn^k$

plus some overhead.

We similarly find a_i for $i = 3, 4, \ldots, n$

such that

$$f(a_1, \ldots, a_i, x_{i+1}, x_{i+2}, \ldots, x_n)$$

is satisfiable. We wind up with

$a_1, \ldots, a_n = T, F$ such that

$$f(a_1, a_2, \ldots, a_{n-1}, a_n)$$

is satisfiable, and since this expression

has no variables, this means that

$$f(a_1, a_2, \ldots, a_n) = T.$$

This takes time $\leq 2Cn^k \cdot n = 2Cn^{k+1}$,

plus some overhead which is clearly

polynomial time.