Homework 4

Rusu Wu
11694764

1. Since $f(x, y)$ is a computable function, the C program of $f(x, y)$ would halts on any input (x, y). If we input given x and y, it would halt and return output $c = f(x)$. After we get c, we replace x by c in $f(x, y)$. For given input c and y in $f(c, y)$, $f(c, y)$ takes input and halts with output $z = f(c, y)$. Clearly, $f(c, y)$ is a computable, then so is $f(f(x, y), y)$.

2. We can list $g(n)$ as follow;
$g(0) = 1$
$g(1) = f(g(0)) = f(1)$
$g(2) = f(g(1)) = f(f(1))$
...
$g(n) = f(g(n-1)) = f(f(...f(f(1))))$
$f(n)$ is totally computable, and $f(x)$ is defined for all x. Hence, not matter for $f(1)$ or $f(f(...f(f(1))))$, they are computable. So, for given input n, $g(n)$ is defined for all n. Therefore, $g(n)$ is totally computable.

3. In order to show that f is a totally computable funtion, we can construct an algorithm as follow;
Build a Turing M to recognize L;
Construct a TM M':
input n;
Catch any string in the length of $n/2$ $(mod(n, 2) = 0)$ from L, and double it to be ww. Then M' check whether the constructing string ww is in L. If yes, return 1 and halts, otherwise return 0 and halts.
Since L is recursive, M' will halt on any input n and return 1 or 0. Hence, f is defined for all input n, and so f is totally computable.

4. L is regular, so we have a finite atutomate M to accept it.
Input n;
We construct a FA M'. Also,construct M_1 andM_2, which are two copies of M ;
Input w, w is a continuous string ennumerate from L $(e.g. L = (abbc), w = (a, b, c, ab, bb, bc, abb, bbc, abbc))$. The length of w should satisfy $w \geq n/2$.
Run M_1 and M_2 in parallel on w on M'; M_1 starts with the initial state of M, M_2 starts with a guessed state of M;
At the end of w; M' check that whether M_1 ends with the starting of guessed state, and M_2 ends with the accepting state of M. If M' say YES, M return 1; If M' say No, M return 0;
Clearly, f is a totally computable function.

5. In order to show whether F terminates is undecidable, we can use the run of G to simulate a 2-counter mechine M.
Devide F into one step, and that is :

$(x_i, y_i, z_i) \xrightarrow{F} (x_{i+1}, y_{i+1}, z_{i+1})$

Here,we use the one-step F to simulate the program of a 2-counter machine; When F runs one step from (x_i, y_i, z_i) to $(x_{i+1}, y_{i+1}, z_{i+1})$, the counter also move from the program line from $z_(i)$ to z(i+1); at the same time, x_i, yi change to $(x_{i+1}, y_{i+1}$. In which, x represents the x counter; y reprsents y counter; z represents the change of line code.

Such that, we can define F as :

$F_k(x_i, yi, z_i, x_{i+1}, y_{i+1}, z_{i+1})$; $1 \leq k \leq$(number of lines in M)-1

Translation of $\exists n.G(n, 0, 0, 0, 1, 1, 1)$ in 2-counter mechine program is : M starts with $x = 0, y = 0$ from line 0, and it will reach line 1 with $x = 1, y = 1$. Since it is undeciable in a 2-counter mechine, $\exists n.G(n, 0, 0, 0, 1, 1, 1)$'s hold is also undecidable. Such that whether F terminates is undecidable.

6. In order to show that the LP problem is decidable, we can translate LP problem into Presburger.

Normally, we can write the Lp problem as

$\exists K(\forall x_1, .., x_n \subseteq Z, K \geq f(x_1, ..., x_n)) \wedge \exists K(\exists x_1, .., x_n \subseteq Z, K = f(x_1, ..., x_n))$

Since presbuger formular $F(x_1, ..., x_n, y)$ holds iff $f(x_1, ..., x_n) = y$, such that we can write the above one as:

$\exists K(\forall x_1, .., x_n \subseteq Z, F(x_1, ..., x_n, y), K \geq y) \wedge \exists x_1, .., x_n \subseteq Z, F(x_1, ..., x_n, K))$

Since Presburger formular is decidable, the LP problem is decidable.

7. From the definition of $linear\ formula\ F$, F is defined by a linear constraint, Boolean connection and linear constants, such that we need a quatifier-elimination Algrithm to show , $P(x_1, .., x_n)$ holds iff $F(x_1, ..., x_n)$ holds.

We can use the mod constraint as a quatifier-elimination Algrithm; For example,

$\exists x.(3x - 4y = 7)$

it can be $\exists x.(3x = 4y + 7)$

then we got $((7+4y)mod3 = 0)$, such that we can eliminate quatifier with the mod constraints.

8. In order to prove G's terminating is undecidable, we can use the four water tanks to simulate a 2-counter machine. We use T_1 to simulate x, use T_2 to simulate y. At the same time, T_3 and T_4 are used to assist the simulation.

Initially, the water level of all the four tanksis exactly 1.

For a 2-counter machine, it only can start with doing $x++$ or $y++$ or check $x == 0$ or $checky == 0$, such that it would do (stay, stay, in, stay) or (stay, stay, stay, in),or do checking; Dedfined that initially it do (stay, stay, in, stay) or (stay, stay, stay, in) .

If the 2-counter machine do $x++$

If $water - level(3) == 1$;guess the number q ,do (in,stay,in,out), test whether T_4==1?

If yes, continues, otherwise crashes.

If $water - level(4) == 1$;guess the number q ,do (in,stay,out,in), test whether T_3==1?

If yes, continues, otherwise crashes.

If the 2-counter machine do $y++$

If $water - level(3) == 1$;guess the number q ,do (stay,in,in,out), test whether T_4==1?

If yes, continues, otherwise crashes.

If $water - level(4) == 1$;guess the number q ,do (stay,out,out,in), test whether T_3==1?

If yes, continues, otherwise crashes.

Similary, for $x--$ and $y--$, we can simulate them in the same way. Hence, the four tanks can simulate a 2-counter machine. We know that 2-counter machines' halting problem is undecidable, so whether G is terminating is also undecidable.

9. A is a NPCM (nondeterministic pushdown atomation argumented with revversal-bounded counters). Construct an NFA B for $A_1, A_2 ... A_n$; N accepts a word $w = a_1 a_2 ... a_n$ iif there is an assignment of each symbol a_i to one of the $A_1, ..., A_r$ such that if the subsequence of symbols assigned to A_i is w_i, then w_i is accepted by A_i.

Construct B as follows;

Defined δ as the transition function of A_i

Assume the initial state of B is $(q_1^0, ..., q_r^0)$;

Defined the transition δ as

$\delta((q_1, ..., q_r), a = \{((p_1, ..., p_r \mid forsome\ 1 \leq i \leq r, p_i \in \delta_i(q_i, a))$ and$p_j = q_j$for all$j \neq i\}$

All states $(q_1, .., q_n)$ can be accepting states of B; q_i is the accepting state of A_i.

Construct a NPCM M that accepts a word x if it is accepted by A but not accepted by N.

C simulates A and B in parallel on x by guessing the symbol of x.

It is easy to simulate A. As for the simulation of B, C convers to B without doing the conversion and bulid / updates the subset as it processes the symbols of x;

At the end of w, C check that there is no accepting state in the reachable subset.

C appects if A accepts and B rejects.

Clearly, the system is not distributed-able iff C accepts a nonempty language. Since the emptiness problem for NPCMs is decidable, we can use above alg to decide whether the system is distributed-able.

10. From the definition, $(S, \ominus)$ does not contain any enable objects, then it halts. So, we can design algorithms to decide (# of enable objects)>0 for every $(S_i, \ominus_i)$ during its runs. If YES, then T is bounded. If there are at least one $(S_i, \ominus_i)$ does not contain any enable objects, it will halts; we could say T is unbounded.