CSCI3390-Assignment 3.

due October 2, 2018

1 Turing-recognizable? Decidable?

Below I've described a number of computational problems. For each problem, you are asked to show either that the problem is Turing-recognizable or decidable. To do this, you need to describe an algorithm that either decides the problem (gives a correct answer on every input) or 'semi-decides' it, in that it gives a correct answer of 'yes' on every positive instance, but may fail to answer on some negative instances.

(a) *Sudoku.* You probably know this one. You are given a 9×9 square grid, partially filled in with the integers 1 through 9.

> *Input:* The partially completed grid.
>
> *Output:* Yes if it is possible to fill in all the cells with the numbers 1 through 9 such that every number appears exactly once in each row, column, and 3×3 subsquare, No if this is not possible.

Show that this problem is decidable.

(b) *Rush Hour.* My favorite. You are given a grid of cars, as in the example shown below. Each car is oriented either along a row or a column, and can move into adjacent empty spaces as long as it remains in its row or column. The problem is to get the red car out of grid.

> *Input:* A grid of cars.
>
> *Output:* Yes if it is possible to drive the red car out of the exit slot at the right of the grid, No if it is not possible.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Figure 1: *At left, ann instance of the Sudoku problem. The output for this instance is Yes, as the filled-in grid at right shows.*

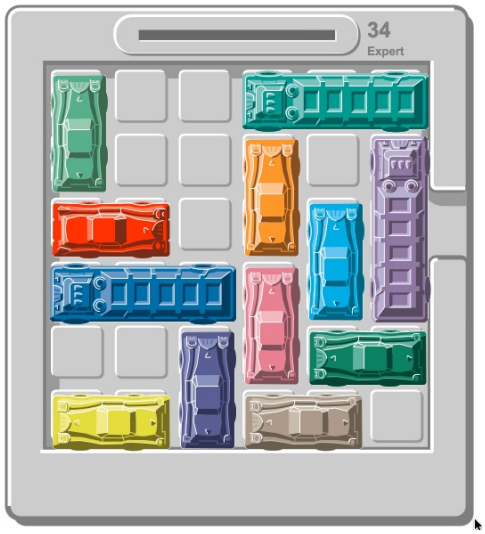

Figure 2: *An instance of the Rush Hour problem. The answer here is 'Yes', but as you can see, this is an 'Expert' instance, so finding a solution is hard!*

Show that this problem is Turing-recognizable, and then (as a bit more of a challenge), show that it is decidable.

(c) Here we consider a *very* special case of Hilbert's Tenth Problem. In the previous solution you showed that the general problem is Turing-recognizable. Here you are to show that in the special case of linear equations, the problem is decidable.

> *Input:* A linear equation in several variables with integer coefficients:
>
> $$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b,$$
>
> where $a_1, \ldots, a_n, b \in \mathbf{Z}$. (For instance, $2x + 9y - 3z = 5$ is such an equation.)
>
> *Output:* Yes if there is an integer solution to the equation, No otherwise. (In the example above, the output is Yes, because$x = y = 1$, $z = 2$ is an integer solution.)

(HINT: This problem requires a little bit of very basic number theory knowledge—I'll bet you've already seen the underlying algorithm, which even tells you how to find a solution to the equation if one exists. To get you thinking in the right direction, what would happen if we changed the equation in the example above to $3x + 9y - 3z = 5$.?)

(d) (This is a somewhat harder problem.) The input is the code for a Python function `f` with a single parameter, and whose only return statements are of the form `return True`,`return False`. The problem is to determine if there is any positive integer `i` such that a call to `f(i)` returns `True`. Show that this problem is *Turing-recognizable*.

HINT: An example of such a function is given below.

```
def f(n):
    while n != 0:
        n=n-6
    return True
```

This function returns the value `True` on 6,12,18, *etc.*, however it loops forever if the argument is any positive integer that is not divisible by 6. Thus the

answer to our problem on this instance is 'Yes'. The problem is undecidable—in fact, it is essentially the same problem as the non-emptiness problem for Turing machines that we showed to be undecidable. Proving that it is Turing-recognizable is not so easy: In earlier examples we proved Turing-recognizablility by a strategy of 'try out every possibility, and answer Yes if you find one that works'. But we cannot use this strategy directly in the present problem. For instance, if the function above is the input, and we 'tried out 1', the function would never terminate, and we would never get to try out 2, or 3,....So we need to find some refinement of this simple-minded strategy. What if when you tried out a candidate value, you only ran the function for a fixed number of steps?

2 Decidable and undecidable problems about Turing machines

(a) Show that the following problem is undecidable:

> *Input:* A Turing machine $\mathcal{M}$.
> *Output:* Yes if $\mathcal{M}$ eventually halts when started on a blank tape, no otherwise.

HINT: As usual, we prove this by a reduction from a known undecidable problem. In this case, use a reduction from the first version of the halting problem: Given $\mathcal{M}$ and w, determine whether $\mathcal{M}$ ever halts when started on w. Show how, given $\mathcal{M}$ and w, we can devise another Turing machine $\mathcal{N}$ that halts when started on a blank tape if and only if $\mathcal{M}$ halts when started on w. Make sure you understand the logic of why this reduction shows that the problem above is undecidable.

(b) Show that the following problem is undecidable:

> *Input:* A Turing machine $\mathcal{M}$ and a tape symbol a.
> *Output:* Yes if $\mathcal{M}$ eventually writes a when started on an blank tape, no otherwise.

HINT: Once again, reduce from the first version of the halting problem.

(c) Show that the following problem is *decidable*:

Input: A Turing machine $\mathcal{M}$. *Output:* Yes if $\mathcal{M}$ ever writes a non-blank symbol when started on a blank tape, No otherwise.

(d) Show that the following problem is *decidable*:

Input: A Turing machine $\mathcal{M}$ and a string w over the input alphabet of $\mathcal{M}$. *Output:* Yes if $\mathcal{M}$ ever moves to the left when started on w.