Homework 5

Theory of Computation (CSCI 2210)

due: 2023-11-29

Problem 1
In class we used a *cardinality argument* to show that not all (string) languages are recursively enumerable. In this problem you will exhibit a *particular language* that is not recognized by any Turing machine.

Recall that the set of strings over a finite alphabet Σ, and the set of Turing machines over Σ, are each *countably infinite*, which is to say that they can be placed in bijection with the natural numbers. Suppose we have an arbitrary but fixed enumeration for strings ($[w_0, w_1, w_2, \cdots]$), and for TMs ($[\mathrm{M}_0, \mathrm{M}_1, \mathrm{M}_2, \cdots]$), respectively.

Use *diagonalization* to construct a language $\mathrm{D} \subseteq \Sigma^*$ that is not recognized by any TM: $\forall\, n \in \mathbb{N} \,.\, \mathrm{D} \neq \mathrm{L}(\mathrm{M}_n)$.

Hint: For each $n \in \mathbb{N}$ come up with a way to determine whether or not $w_n \in \mathrm{D}$ that guarantees that no TM recognizes D.

Problem 2
Consider the following λ-terms:

i. $(\lambda x \,.\, x)\,(\lambda x \,.\, x)$

ii. $(\lambda x\, y \,.\, y)\,((\lambda x \,.\, x\, x)\,(\lambda x \,.\, x\, x))\, z$

iii. $(\lambda x\, y\, z \,.\, x\, z\,(y\, z))\,(\lambda x\, y \,.\, x)\,(\lambda x \,.\, x)$

Do the following for each term above:

(a) *Elaborate* the term by making explicit the implied parentheses and expanding multiple bindings into successive single bindings.

(b) Annotate the term with its *binding structure* by drawing an arrow from each *bound occurrence* of a variable to its binding site.

(c) Give a β-reduction of the term to its *normal form*.

You may do this interactively at `https://lambdacalc.io/`, but please write down or provide a screenshot of your steps.

Problem 3
Prove that the following two λ-terms are *not* β-equivalent:

$$(\lambda x\, y \,.\, x)\,(\lambda x\, y \,.\, x)\,(\lambda x\, y \,.\, y) \qquad \text{and} \qquad (\lambda x\, y \,.\, y)\,(\lambda x\, y \,.\, x)\,(\lambda x\, y \,.\, y)$$

Please be precise in explaining how your argument is sufficient to establish the result.

Hint: You will need to appeal to one of the theorems that we learned about the λ-calculus.

Problem 4
Recall that we can encode the natural numbers and arithmetic operations in λ-calculus using *Church numerals*.

(a) Write out the Church-numeral encodings of the following two arithmetic expressions:

$$1 + 1 \qquad\qquad \text{and} \qquad\qquad 2$$

(b) Verify the mathematical fact that $1 + 1 = 2$ by normalizing the encoding of the term on the left to the encoding of the term on the right.

You may do this interactively at `https://lambdacalc.io/`, but please write down or provide a screenshot of your steps.

Problem 5
Here is Curry's fixed point combinator implemented in *Python*:

```
Y = lambda f : \
    (lambda x : f (lambda y : x (x) (y))) \
    (lambda x : f (lambda y : x (x) (y)))
```

Recall that due to Python's eager evaluation strategy we need to add one more layer of functions relative to the Y-combinator we wrote in λ-calculus. However, this need not concern us, and you can rely on the fact that this Python function computes fixed points.

Write a Python expression F such that `Y (F)` computes the *Fibonacci function*.

For example:

```
> list (map (Y (F) , range (10)))
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

You may use the ordinary Python numerals (`0`, `1`, etc.), arithmetic operators (`+`, `-`, etc.), comparisons (`==`, `<`, etc.), and conditionals (`if...else`). So in particular, you do *not* need to encode *Church booleans* or *Church numerals* in Python.

Submit your solution as a Python file `fib.py`.