1. ***Punched-tape machine.*** Prove that the punched-tape machine is equivalent in power to the standard Turing machine. In other words, the languages accepted by the punched-tape machines are exactly all the computable languages.

Solution: Any punched-tape machine can be simulated by a Turing machine because by definition a punched-tape machine is a Turing machine with a more restricted set of instructions. Now given an arbitrary Turing machine M, we will describe how to simulate M using a punched-tape machine P. Without loss of generality we assume that M has binary input alphabet $\{\mathtt{0}, \mathtt{1}\}$, along with the empty cell symbol $\square$.

First we create another Turing machine M' mimicking the behavior of M. Machine M' contains a marked version of the binary symbols $\{\dot{\mathtt{0}}, \dot{\mathtt{1}}\}$, a pointed version of the binary symbols $\{\check{\mathtt{0}}, \check{\mathtt{1}}\}$, together with a delimiter symbol $|$ and a special prep symbol $\mathtt{x}$ whose use will be explained later. For each transition of M (that is, read a symbol, write a symbol, and move the head), the new machine M' performs the following:

- Move all the way beyond the last delimiter at the right end of (the dirty portion of) the tape.
- Copy the old section of the tape between the last and the second-to-last delimiters symbol-by-symbol from left to right, into a new section on the right of the last delimiter, except for the two symbols to be changed.
  - To remember the positions of the old symbol to be copied and the new symbol to be written, whenever we are about to copy a symbol in the section, (1) mark the position in the new section with the prep symbol $\mathtt{x}$, (2) move to the old section and mark the symbol to be copied (using the new characters introduced in M'), and (3) move to the new section and write down the unmarked version to the symbol to replace the prep symbol.
  - When copying the two symbols around the one that contains the pointer ˇ, the machine enters the special states and makes all the proper changes based on the transition of M that we are trying to emulate.
- Add a delimiter at the end of the new section.

This concludes the description of M' that emulates M, and therefore M' decides the exact same language as M.

Now we describe how to encode each symbol of M' as a sequence of five (un)punches using the punched-tape machine P, and thus proving that P and M' (and thus M) are equivalent in power. Let

$$\begin{aligned}
&\langle \mathtt{0} \rangle = \square\square\square\blacksquare\blacksquare, \quad \langle \mathtt{1} \rangle = \blacksquare\square\square\blacksquare\blacksquare, \quad \langle \dot{\mathtt{0}} \rangle = \square\blacksquare\square\blacksquare\blacksquare, \quad \langle \dot{\mathtt{1}} \rangle = \blacksquare\blacksquare\square\blacksquare\blacksquare,\\
&\langle \check{\mathtt{0}} \rangle = \square\square\blacksquare\blacksquare\blacksquare, \quad \langle \check{\mathtt{1}} \rangle = \blacksquare\square\blacksquare\blacksquare\blacksquare, \quad \langle | \rangle = \blacksquare\blacksquare\blacksquare\square\blacksquare, \quad \langle \mathtt{x} \rangle = \square\square\square\square\blacksquare.
\end{aligned}$$

The empty symbol $\square$ is naturally mapped to $\square\square\square\square\square$. One can summarize the encoding as follows: the last punch decides if the cell is empty; the fourth one decides if it is a special character (prep symbol $\mathtt{x}$ or delimiter $|$); if the last two punches are $\blacksquare\blacksquare$, then the first punch decides if the symbol was $\mathtt{0}$ or $\mathtt{1}$, the second punch decides if the symbol was marked, and the third punch decides if the symbol was pointed at by the head.

During the execution of M', only the following symbol overwrites can happen:

$$\begin{aligned}
\square &\to \mathtt{x}, |,\\
\mathtt{x} &\to \mathtt{0}, \mathtt{1}, \check{\mathtt{0}}, \check{\mathtt{1}},\\
\mathtt{0}, \mathtt{1} &\to \dot{\mathtt{0}}, \dot{\mathtt{1}}.
\end{aligned}$$

It is straightforward to check that the corresponding sequence of punched can be obtained by making new punches alone, without removing any existing punches. ■

Rubric: Standard 5-point grading scale (plus deadly-sins and sudden-death rules).

Maximum 3 points if the idea of representing a single symbol in the original Turing machine using multiply punches is missing. Maximum 1 point if the idea of copying the whole current working tape is missing.

2. ***Self-referential machines.***

(a) Design an algorithm B that, assuming that its own source code $\langle B \rangle$ is conveniently given as input when starts, outputs the *source code* of another algorithm A that prints the source code of B on execution. (The algorithm should be described using pseudocode, not an explicit Turing machine.)

Solution: Consider the following programs:

```
$A_w()$:
    input: none
    write $w$
```

```
$B(w)$:
    input: some data $w$
    print $\langle A_w \rangle$
```

Algorithm B will construct and print out the source code of A_w, whose sole purpose is to write the data w in memory. If we are given $\langle B \rangle$ as input to B, the algorithm will treat $\langle B \rangle$ as yet another input and correctly print out the source code of $A_{\langle B \rangle}$.

■

(b) Design an algorithm that outputs its own source code without any input. (Remember that your algorithm does not have access to its own source code. The executable of the algorithm alone has to generate its source code back.)

Solution: First we modify the definition of B slightly. The only difference is that after printing out $\langle A_w \rangle$, B also prints out a copy of the input w.

```
$A_w()$:
    input: none
    print $w$
```

```
$B(w)$:
    input: some data $w$
    print $\langle A_w \rangle \cdot w$
```

From here, we consider the program AB by concatenating program $A_{\langle B \rangle}$ and B together. Notice that because we have properly defined B, the source code $\langle B \rangle$ exists and is well-defined.

```
$AB()$:
    run $A_{\langle B \rangle}$
    run $B$ on input $\langle B \rangle$
```

Running AB on empty input, we will see the program perform the following tasks:

- Program $A_{\langle B \rangle}$ writes $\langle B \rangle$ in memory,
- Program B read $\langle B \rangle$ from the memory as input,
- Program B runs on $\langle B \rangle$ and prints $\left\langle A_{\langle B \rangle} \right\rangle \cdot \langle B \rangle$.

Assume that we choose our programming language (encoding of the machines) such that concatenation of source codes translates to sequential execution of the programs, the final output $\left\langle A_{\langle B \rangle} \right\rangle \cdot \langle B \rangle$ is indeed the source code of AB. ■

Rubric: Standard 5-point grading scale (plus deadly-sins and sudden-death rules).

Maximum 2 points if the definition of the machines is *circular*; in particular, it is not allowed for the source code of program M to mention the program itself, or if the description of B includes the source code of A and in the description of A there's mention to the machine B (because one has to properly define A before defining B, so machine B does not exist when you are still describing A).