Chapter 1

Time Stone

1.1 Extracts from wiki

"The Time Stone was one of the six Infinity Stones... ...the user is able to physically control and redirect the flow of time, and can specifically select the exact area to manipulate without affecting those outside its selected range. The stone can alter targets as small as an apple or as wide in scope as the timeline of the universe... ...Due to the selective nature of the Time Stone's power, it could be used to individually alter the timeline of individual objects or events, reversing them to a previous state or sending the object forward into a future state. This occurs regardless of any potential breaches in causality... ...the Stone could send objects forward to a potential future state that does not necessarily have to occur in the current timeline."

1.2 Abilities

- Can alter an arbitrary space.
- Can reverse such a space to a previous state.
- Can send the space forward into a future potential state (that does not necessarily have to occur in the current timeline.)
- Does not cause issues in causality.

1.3 Definition and Consequences (1st Attempt)

A TS is a tuple T = (R, q_{end})

- R is a deterministic run of some automata which is assumed to halt.
- q_{end} the final step of the run.

When activated, the TS alters the automaton, sending it forward in time to q_{end}. This process is assumed to take no computational power and is a constant $O(1)$ in time and memory complexity.

We define a new MCU complexity class, which in this case defines the number of times the TS must be activated.

1.3.1 P = NP

We create a proof by doing the following:

1. Prove using a TS on an arbitrary deterministic TM that is assumed to halt will result in a time complexity of O(1).
2. Simulate a non-deterministic TM on a deterministic TM.
3. Use steps 1 and 2 to show when using the TS, NP problems have a time complexity of O(1), hence P = NP.

Step 1: TS and deterministic TM interactions

Use the time stone to send a deterministic Turing machine forward in time to its end state such that its time complexity is reduced to $O(1)$.

Get any arbitrary deterministic TM that is assumed to halt.
We get the TS components:

- R is the deterministic run of the TM.
- There exists a q_{end} step, as the algorithm is assumed to halt.

We place the TM into its initial step. We activate the TS, sending the TM forward in time to q_{end}, where it either halt-accepts or halt-rejects. This takes $O(1)$ time complexity.

The arbitrary deterministic TM, using the TS with an MCU complexity of O(1), has now halted in $O(1)$ time.

We conclude that all deterministic TMs assumed to halt now have a time complexity of $O(1)$ and MCU complexity of $O(1)$ when using a TS.

Step 2: Simulating a non-deterministic TM using a deterministic TM

Have a TM that uses another TM to dictate which path to take at any non-deterministic choice using a BFS, where eventually all paths of the non-deterministic tree are taken. Therefore, the entire process is now deterministic in nature.

We construct a TM containing three tapes.

The Computer

The first tape, *the computer*, is designed to do an arbitrary run of some non-deterministic TM that is assumed to halt.

At each non-deterministic step, when facing an x number of choices, each choice is labelled from 0 to $x - 1$ such that the step can be chosen deterministically via an external input.

If the input path does not exist, the TM will halt-reject with an additional output specifying that the input path did not exist.

The tape, otherwise, can halt-reject or halt-accept as normal.

The Echo

The second tape, *the echo*, simply retains the initial step of *the computer*. It can copy its contents to *the computer*, such that it can return to its initial step after leaving it. Corollary, *the echo* shares the arbitrary tape language of *the computer*.

The Decider

The third tape, *the decider*, is designed to determine which route *the computer* will take given a non-deterministic step.

The input language consists of the set of natural numbers, and runs a BFS to determine what choice will be made when given a non-deterministic choice.

Usage

When ran, eventually, the BFS will either:

- reach a halt-accept state
- iterate through the entire tree and find no halt-accept state, and therefore will halt-reject.

The non-deterministic TM has now been simulated using a deterministic multi-tape TM. Multi-tape TMs area equivalent to single-tape TMs, therefore we conclude all non-deterministic algorithms that are assumed to halt can be converted to deterministic algorithms that are assumed to halt.

Step 3: TS and simulated non-deterministic TM interactions

As we have concluded that all non-deterministic TMs that are assumed to halt can be simulated using a deterministic TM, and all deterministic TMs when using a TS have a time complexity of $O(1)$, we conclude that all non-deterministic TMs which are assumed to halt, when using a TS, have a time complexity of $O(1)$ and MCU complexity of $O(1)$.

Therefore, as all P and NP problems halt, we conclude that given access to a TS, P = NP.

1.3.2 Halting Problem

In our formal definition, we state the given input TM is assumed to halt. However, this is a very weak assumption to make.

Let's attempt to bypass this requirement and instead see what happens when we run the TS on a TM which does not necessarily halt.

When activated, the TS alters the TM, sending it forward in time to q_{end}. This process is assumed to take no computational power and is a constant $O(1)$ in time and memory complexity.

Clearly if the TM does halt, it is sent forward in time to q_{end} as normal. However, what would occur if the TM was an infinite loop? Answering anything other than "the TM is sent forward in time to q_{end} as normal" inherently contradicts the halting problem, as if we define this new behaviour in any way that differs from the original, as a consequence we have created a device which determines if a given TM will halt.

Therefore, this definition cannot exist.

1.4 Definition and Consequences (2nd Attempt)

A TS is a tuple T = (R, c)

- R is a deterministic run of some automata which is assumed to halt.
- c is a constant multiplier, and accepts the input language of **N**.

When activated, the TS alters the automaton, speeding up its operation by a factor of c. This process is assumed to take no computational power and is a constant $O(1)$ in time and memory complexity.

1.4.1 P = NP

We simulate an arbitrary non-deterministic TM that is assumed to halt on a deterministic TM (specifically, using the BFS method from the previous definition).

As we traverse the BFS tree, the number of nodes at each level increases by a upperbound factor of n, where n is the maximum number of non-deterministic choices at some step in the non-deterministic TMs code.

At each level h of the tree, if we activate the TS to speed up the TM by a factor of $n * h$, clearly, we will run through the entire level less than or equal to the time it would take to run a single step.

Therefore, it will now run in the same time as a true non-deterministic TM.

Therefore, when using a TS, P = NP.

1.4.2 Halting Problem

We get any arbitrary TM and modify it such that at every step, an output is given to the TS to double its own multiplier speed. I.e., the first step will take 1 second to compute, the next step will take 0.5 seconds to compute, 0.25s, 0.125s ...

As we know that $1+0.5+0.25+0.125... \leq 2$, after 2 seconds of computation, we would have ran an infinite number of steps on this TM and will observe two possibilities:

1. The TM will have halted as it has completed some N amount of steps.
2. The TM will continues running as it is in an infinite loop.

Again, the consequence of this concept is we now have a device that determines if a given TM will halt.

Therefore, this definition cannot exist.