Writing a Self-Reproducing Program

Amisi Kiarie

December 17, 2021

1 The Problem Defined

The problem given by Ken Thompson[1] is as follows: Write a source program that, when compiled and executed, will produce as output an exact copy of its source.

To understand the difficulty involved in the problem, let us try to understand it somewhat. Let $\mathscr{T}$ be the set of all strings and $\mathscr{U} \subset \mathscr{T}$ be the set of source programmes in our language, and $\mathscr{C}$ be the set of compiled programs. Then we may denote the *compilation* and *execution maps*

$$\mathbf{CMPL} : \mathscr{U} \to \mathscr{C} \text{ and } \mathbf{EXEC} : \mathscr{C} \to \mathscr{T}$$

respectively, provided we restrict $\mathscr{U}$ to programs that, when compiled and executed, only produce text output.

A *self-reproducing program* $S \in \mathscr{U}$ is one with the property

$$S = \mathbf{EXEC}(\mathbf{CMPL}(S)). \tag{1}$$

This may motivate the definition of the *interpretation map*

$$\mathbf{EVAL} \coloneqq \mathbf{EXEC} \circ \mathbf{CMPL},$$

which allows us to rewrite (1) as

$$S = \mathbf{EVAL}(S).$$

In a word, S is an identity element of **EVAL**. By the same token, we have

$$\begin{aligned} S &= \mathbf{EVAL}(S) \\ &= \mathbf{EVAL}(\mathbf{EVAL}(S)) \\ &= \ldots \\ &= \mathbf{EVAL}^N(S) \end{aligned}$$

for any positive integer N.

[1] *Reflections on Trusting Trust*, Turing Award Lecture, 1984

2 Constructing S

2.1 A naive first attempt

Since we are focused on **EVAL** mainly, we may use the language python to arrive at a solution. Let P_0 be the one-line application

```
print("print()")
```

Then the output $\textbf{EVAL}(P_0)$ is

```
print()
```

If we attempt to modify P_0 so that the portion interior to the *print* command is included in the output, we obtain P_1

```
print("print(\"print()\")")
```

But this fails to solve our problem because $\textbf{EVAL}(P_1)$ is

```
print("print()")
```

However, we see that $\textbf{EVAL}(P_1) = P_0$, and it is not hard to see that we may easily construct the sequence $\{P_0, P_1, P_2, \dots\}$ with

$$\textbf{EVAL}(P_n) = P_{n-1}, \quad \text{for } n \geq 0.$$

Observe, then, that each step in writing a new 'upgrade' from $P_n \to P_{n+1}$ is something akin to applying $\textbf{EVAL}^{-1}$ to P_n. Now consider the following function:[2]

```
def upgrade(s):
    escaped = s.replace("\"", "\\\"")
    return f"print(\"{escaped}\")"
```

Note that if we write

```
upgrade("print()")
```

we obtain a string equivalent to P_0, i.e.

[2] We are well aware that python offers both single- and double-quotes, but for the sake of illustrating our technique in a manner that is relatively language-agnostic, we have kept only to the latter.

```
print(upgrade("print()"))
```

outputs P_0. Moreover,

```
print(upgrade(upgrade("print()")))
```

outputs

```
print("print(\"print()\")")
```

which is simply P_1. The *upgrade* function thus maps $P_n \to P_{n+1}$.

Our aim is to obtain the program S such that $\mathbf{EVAL}(S) = S$, and we have the following:

- $\mathbf{EVAL}(P_n) = P_{n-1}$
- $\mathit{upgrade}(P_n) = P_{n+1}$.

If we can somehow write a program that connects these two properties, we will obtain our self-reproducing S.

Reflection At this stage the way to unite these two is not evident, so we shall attempt another route.

2.2 Recursive construction of S

Perhaps we may learn something from $\{P_n\}$, other than humility! Let S be self-reproducing. We shall assume that it takes the form

$$S = \rho \cdot \varsigma, \tag{2}$$

where '$\cdot$' is the *concatenation operator*, and $\rho, \sigma \in \mathscr{T}$. We shall define ρ and σ shortly.

Our exercise with $\{P_n\}$ teaches us that it is well-nigh impossible to construct S through printing a single string because that string must contain itself *and* other things: if nothing else, it must contain the code that contains the command to *print* it.

The idea now is to make use of a recursive routine ρ, which takes as its input part of its own source code. By passing in this portion of ρ to ρ, we allow ρ to be able to output the calling line σ, because it has a reference to this content. With $\{P_n\}$ the critical issue was we could only reference P_n by writing P_{n+1}, but here ρ will reference its calling line by using only its name and the portion of its source that it is given.

The final challenge is to determine how to terminate the recursion, i.e. what the base step is in this context.