Rice's Theorem

S is a set of strings over some fixed alphabet.

Theorem. S is computable if and only if $\overline{S}$ is computable.

Proof. Suppose S is computable. Let a be a program that solves S. The following program b solves $\overline{S}$.

```
b(x)
  y = a(x)
  If y = 'yes'
    then answer 'no'
    else answer 'yes'
```

For the rest of this page, S is a set of programs. (Since programs are strings, S is also a set of strings.)

Definition. Say that S is ***nontrivial*** if $S \neq \emptyset$ and $\overline{S} \neq \emptyset$.

Definition. Two programs p and q are ***equivalent*** (written $p \approx q$) if, for every x, one of the following is true.

(a) $p(x) \uparrow$ and $q(x) \uparrow$.

(b) $p(x) \downarrow$ and $q(x) \downarrow$ and $p(x) = q(x)$

Definition. Say that S ***respects equivalence*** if, whenever p and q are programs where $p \approx q$,

$$p \in S \Leftrightarrow q \in S.$$

Theorem. (Rice's theorem) Suppose that S is a nontrivial set of programs that respects equivalence. Then S is not computable.

Proof.

1. The proof is by contradiction. So assume that S is computable. Choose a program solveS that solves S. That is, for every string (or program) x,

$$x \in S \quad \Leftrightarrow \quad \text{solveS}(x) \text{ answers yes} \tag{1}$$

$$x \notin S \quad \Leftrightarrow \quad \text{solveS}(x) \text{ answers no} \tag{2}$$

2. Define LOOP to be a program that loops forever on all inputs.

   We can assume without loss of generality that LOOP $\notin S$. (If LOOP $\in S$ then we show that $\overline{S}$ is not computable. Since $\overline{S}$ is computable if and only if S is computable, it does not matter which one we look at.)

3. Since S is nontrivial, it is not empty. Select a program G that is in S. (G is a "good" program for S.)

4. If x is a program, define program q_x as follows.

   $\underline{q_x(y)}$
   Run $x(x)$
   Return $G(y)$

   Notice that

$$\begin{aligned} x(x)\uparrow &\Rightarrow q_x(y)\uparrow \text{ for all } y \\ &\Rightarrow q_x \approx \text{LOOP} \\ &\Rightarrow q_x \notin S. \end{aligned}$$

$$\begin{aligned} x(x)\downarrow &\Rightarrow q_x(y) \text{ behaves like } G(y) \\ &\Rightarrow q_x \approx G \\ &\Rightarrow q_x \in S \end{aligned}$$

   The last step in each chain of implications follows because S respects equivalence. Putting those implications together, for every x,

$$x(x)\downarrow \;\Leftrightarrow\; q_x \in S. \tag{3}$$

5. Now write program F as follows.

   $\underline{F(p)}$
   Build q_p as indicated in step 4.
   If solveS(q_p) = 'yes'
   loop forever
   else
   stop

Notice that, based on the definition of $F(p)$, for every p:

$$\begin{aligned} \text{solveS}(q_p) = \text{'yes'} &\;\Rightarrow\; F(p)\uparrow \\ \text{solveS}(q_p) = \text{'no'} &\;\Rightarrow\; F(p)\downarrow \end{aligned}$$

Putting those two implications together gives

$$\text{solveS}(q_p) = \text{'yes'} \;\Leftrightarrow\; F(p)\uparrow . \tag{4}$$

6. Let's put equivalences (1), (3) and {refF) together. For every p,

$$\begin{aligned} F(p)\uparrow &\;\Leftrightarrow\; \text{solveS}(q_p) = \text{'yes'} && \text{by (4)} \\ &\;\Leftrightarrow\; q_p \in S && \text{by (1)} \\ &\;\Leftrightarrow\; p(p)\downarrow && \text{by (3)} \end{aligned} \tag{5}$$

7. Since (5) holds for all p, it must hold for $p = F$. Replacing p by F in (5) yields

$$F(F)\uparrow \;\Leftrightarrow\; F(F)\downarrow .$$

That is a contradiction.

The crucial idea in the proof of Rice's theorem is to convert a question about membership in S into an equivalent question about whether a program halts on itself as input. That is the role played by q_x. Other than that twist, the proof is very similar to the proof that the halting problem is uncomputable.

Note. Inspection of the proof shows that it is constructive in the sense that, given a program solveS that purports to solve S, it yields an input f on which solveS gives the wrong answer. Just choose $f = q_F$.

Examples.

(a) $A = \{p \mid p(\varepsilon) = \text{'yes'}\}$ is nontrivial because some programs accept the empty string and some don't. It respects equivalence because, given any two equivalent programs p and q,

$$\begin{aligned} p \in A &\;\Leftrightarrow\; p(\varepsilon) = \text{'yes'} \\ &\;\Leftrightarrow\; q(\varepsilon) = \text{'yes'} \\ &\;\Leftrightarrow\; q \in A. \end{aligned}$$

The second step is based on the fact that $p \approx q$.

Conclude: A is not computable.

(b) Say that program p is a *prime number program* if $p(n) =$ 'yes' when n is a prime number and $p(n) =$ 'no' otherwise.

Let $B = \{p \mid p \text{ is a prime number program}\}$.

B is nontrivial since some programs are prime number programs and some aren't.

B respects equivalence. Suppose that p and q are two equivalent programs. Then

$$\begin{aligned} p \in B &\Leftrightarrow p \text{ is a prime number program} \\ &\Leftrightarrow q \text{ is a prime number program} \\ &\Leftrightarrow q \in B \end{aligned}$$

where the middle line follows because $p \approx q$.

Conclude: B is not computable.

(c) If p is a program then $L(p)$ is the set of strings that p accepts.

Let $C = \{p \mid L(p) \text{ is a regular language}\}$.

C is nontrivial since there are some programs p where $L(p)$ is a regular language, and there are some programs p where $L(p)$ is not a regular language.

C respects equivalence. Suppose that p and q are two equivalent programs. Then

$$\begin{aligned} p \in C &\Leftrightarrow L(p) \text{ is a regular language} \\ &\Leftrightarrow L(q) \text{ is a regular language} \\ &\Leftrightarrow q \in C \end{aligned}$$

where the middle line follows because $p \approx q$.

Conclude: C is not computable.

(d) Let $K = \{p \mid p(p) \downarrow\}$. K is nontrivial, but it does not respect equivalence. You can find two equivalent programs p and q where $p(p) \downarrow$ but $q(q) \uparrow$. $p(p)$ and $q(p)$ must do the same thing, but $p(p)$ does not have to do the same thing as $q(q)$.

Since K does not respect equivalence, Rice's theorem has nothing to say about it. In fact, K is uncomputable.

(e) Let $D = \{p \mid p$ is a Java program that contains a variable called 'mango'$\}$. D is nontrivial since some Java programs have a variable called 'mango' and some don't, but D clearly does not respect equivalence, and D is clearly computable.