shouldn't be that surprising that there is no easy way of figuring out whether a computer program eventually stops running; if there was, there would be an easy way of determining the truth of a large number of very difficult mathematical problems.

Note that the fact that we are running φ_n on input n is not so important to incomputability of K. For example:

Exercise 3.6. Show that $\{n : \varphi_n(0)\downarrow\}$ is incomputable.

3.2 The need for studying partial computable functions

We are mostly only interested in studying total computable functions in computability theory. Yet, our theory includes partial functions. One reason for this is that there is no set of functions $(f_n)_{n\in\mathbb{N}}$ (which we would like to be all total computable functions from $\mathbb{N} \to \mathbb{N}$) with the following properties:

1. The functions f_n are all total.
2. The functions f_n are closed under composition and include the function $x \mapsto x + 1$.
3. There function $x \mapsto f_x(x)$ is in the collection (i.e. there is a computable "universal machine").

This is because if we had all three properties, the function $x \mapsto f_x(x) + 1$ would be in this collection (by composing the function from (3) with the addition function using (2)). So $x \mapsto f_x(x)$ would be equal to f_n for some n, but then $f_n(n)$ is defined (since every f_n is total by (1)) and $f_n(n) = f_n(n) + 1$. Contradiction!

To make a reasonable theory of computation we have to include property (2). We also should have property (3): if executing the nth algorithm on the nth input is not computable, then we haven't made a notion of algorithm that we can actually compute. So we are forced to drop property (1).

Questions about what computations halt, and what programs are total, etc. will be quite important in computability theory.

3.3 Rice's theorem

The halting problem poses a very serious limitation on our ability to computably understand the behavior of Turing machines. However, the situation is actually much worse. Using the incomputability of the halting problem, we can show that there is not a single nontrivial property of partial computable functions which we can distinguish in a computable manner.

Definition 3.7. Say that a set $A \subseteq \mathbb{N}$ is an **index set** if for all n, m, if φ_n and φ_m compute the same partial function (i.e. $\varphi_n = \varphi_m$), then $n \in A \leftrightarrow m \in A$.

So an index set A consists of all programs that give partial computable functions that have some property. For example, all programs that compute total functions, all functions φ_n so that $\varphi_n(m)\downarrow$ for some n, all n such that $\varphi_n(m) = m + 1$, etc.

It turns out that there are no nontrivial computable index sets:

Theorem 3.8 (Rice's theorem)**.** *Suppose $A \subseteq \mathbb{N}$ is a computable index set. Then either $A = \mathbb{N}$ or $A = \emptyset$.*

Proof. By contradiction assume that A is a computable index set such that $A \neq \mathbb{N}$ and $A \neq \emptyset$. Let φ_{n_0} a partial computable function which is undefined on every input. Since A is computable iff its complement is computable, by exchanging A for its complement, we may as well assume $n_0 \in A$. Since $A \neq \mathbb{N}$, there is some n_1 such that $n_1 \notin C$.

Now we obtain a contradiction by showing that the halting problem is computable. Consider the partial computable function $f(n, m)$ deifned as follows to compute $f(n, m)$ we first compute

$\varphi_n(n)$, and if this halts, then we compute $\varphi_{n_1}(m)$ and output the answer. Let x be the program computing f so $\varphi_x(n,m) = f(n,m)$, and by the S-m-n theorem there is a computable s so that $\varphi_{s(x,n)}(m) = f(n,m)$. Now by the definition of f, if $\varphi_n(n)$ does not halt, then $\varphi_{s(x,n)}(m) = f(n,m)$ is undefined for all m, so $\varphi_{s(x,n)} = \varphi_{n_0}$, so $s(x,n) \in A$ since $n_0 \in A$. If $\varphi_n(n)$ does halt, then $\varphi_{s(x,n)} = f(n,m) = \varphi_{n_1}(m)$ for all m, so $\varphi_{s(x,n)} = \varphi_{n_1}$ and so $s(x,n) \notin A$ since $n_1 \notin C$.

Thus, $\varphi_n(n)$ halts iff $s(x,n) \notin A$. Since we are assuming A is computable and s is computable, we conclude that the halting problem K is computable. Contradiction! $\square$

We are explicitly explaining where we are using the S-m-n theorem above. However, this type of argument – computably producing a computer program from finitely many inputs which has some desired behavior – is constantly used in the computability, and most books don't bother pointing out that by doing this they are implicitly using the S-m-n theorem. So you'll often see definitions like the following: Define a computable function $g\colon \mathbb{N} \to \mathbb{N}$ so that

$$\varphi_{g(n)}(m) = \begin{cases} \varphi_{n_1}(m) & \text{if } \varphi_n(n)\downarrow \\ \text{undefined} & \text{if } \varphi_n(n)\uparrow. \end{cases}$$

Then $\varphi_{g(n)} = \varphi_{n_1}$ if $\varphi_n(n)\downarrow$ and $\varphi_{g(n)} = \varphi_{n_0}$ if $\varphi_n(n)\uparrow$.