13 Partially Computable Sets and m-Complete Problems

13.1 Partially Computable Sets

For a program p, we have defined

$$L(p) = \{x \mid p(x) \cong 1\}$$

We also added a stipulation that p can only be said to decide $L(p)$ if p halts on all inputs. What if we remove the stipulation? Does the definition of $L(p)$ make any sense then?

Let's say that a language X is *partially computable* if there exists a program p so that $X = L(p)$, without the stipulation that p must halt on all inputs. Is there a difference between the notion of a computable language and a partially computable language?

It is easy to see that the class of partially computable languages is not the same as the class of computable language. We know that HLT is not computable.

Theorem 13.1. HLT is partially computable.

Proof. Here is a program halt(x) where $L(\text{halt}) = \text{HLT}$.

```
"{halt(p, x):
   Run p(x)
   return 1
}"
```

You can check that HLT is, indeed, the set of all ordered pairs (p, x) where $\text{halt}(p, x) \cong 1$. What is going on here?

The trick lies in the fact that halts(p, x) answers "no" by looping forever. That is unpleasant because you can't tell the difference between a program

that is looping forever and a program that is just taking a very long time to stop.

Notice that every computable language is also partially computable. That is easy. The requirement

$$L(p) = \{x \mid p(x) \cong 1\}$$

is the same as the requirement for a language to be computable. We have just omitted the extra requirement that p must stop on every input.

13.2 M-Complete Problems

BIG IDEA: We can identify hardest problems in particular classes.

What is the point of defining partially computable languages? Well, for one thing, they give us a third level of difficulty. A language can be computable; partially computable (but not computable); and not even partially computable. For example, ALL is known not to be partially computable.

For our purposes, the class of partially computable languages gives us an opportunity to identify languages are the most difficult partially computable problems.

Suppose that you want to show that person t is a tallest person in the room. (It is possible for two people to be equally tall, so there might be more than one tallest person). First, you would obviously need for person t to be in the room. Next, you would need to compare person t's height with the height of every other person in the room and find that t is at least as tall as every person in the room.

We can use an analogous idea to identify most difficult partially computable problems.

Definition 13.2. Say that language T is *m-complete* if both of the following are true.

1. T is partially computable.
2. $X \leq_m T$ for every partially computable language X.

If you think of relation $\leq_m$ as meaning "no more difficult than,", then you can see that an m-complete language is one of the hardest partially computable languages.

Do m-complete languages exist? Yes, and HLT is one of them.

Theorem 13.3. HLT is m-complete.

Proof. We have seen that HLT is partially computable, so condition (1) is met. Now we must show that $X \leq_m$ HLT for every partially computable language X.

Suppose that X is partially computable. Select a program r that partially computes X. We can modify r so that it never answers "no" by making it go into an infinite loop instead of answering no. Then we find that

$$X = \{x \mid r(x)\downarrow\}$$

since, for the modified program r, halting is equivalent to answering "yes."

Keep in mind the goal here: show that $X \leq_m$ HLT. An m-reduction from X to HLT is

$$f(x) = (r, x).$$

It is easy to check that $f(x)$ has both of the properties needed of an m-reduction from X to HLT. Clearly, f is computable, since it just writes the fixed program r along with x. Also, for every x,

$$x \in X \;\leftrightarrow\; f(x) \in \text{HLT}.$$

$\diamondsuit$

It should not be surprising that HLT is one of the most difficult partially computable languages. If you could solve HLT then you could solve every partially computable language; just use your solution to HLT to tell whether or not a program is looping forever, making $\perp$ just as good an answer as 'no'.

A most difficult partially computable language ought not to be computable. Take HLT as an example.

Theorem 13.4. If L is m-complete then L is uncomputable.

Proof. HLT is partially computable. By the definition of an m-complete language, HLT $\leq_m L$. By Corollary 11.15, L is not computable. (Intuitively: L is at least as hard as uncomputable problem HLT.)

$\diamondsuit$

Later, we will see another context where we want to find problems that are among the most difficult problems in a particular class.

13.3 Co-Partially Computable Sets

There is an asymmetry in the definition of partially computable sets. A language L is partially computable if it is solvable by a program that stops on inputs that are in L and loops forever on inputs that are not in L. What if we define an analogous class of problems that are solvable by programs that stop when the answer is no and loop forever when the answer is yes?

Definition 13.5. Language L is *co-partially computable* if there exists a program p so that stops when x is not in L and that loops forever when x is in L.

Is that the same class as the partially computable languages? No, because of the asymmetry! Look at $\overline{\text{HLT}}$. You already know that there is a program p that stops on an input that is in HLT and loops forever on inputs that are not in HLT. The same program p halts when $x \notin \overline{\text{HLT}}$, and shows that $\overline{HLT}$ is co-partially computable.

The following theorem is enough to show that the partially computable languages cannot be the same as the co-partially computable language.

Theorem 13.6. For every language L, L is computable if and only if L is both partially computable and co-partially computable.

Proof. First, suppose that L is both partially computable and co-partially computable. Get a program p that stops on inputs that are in L and another program q that stops on inputs that are not in L. To decide whether $x \in L$, run $p(x)$ and $q(x)$ concurrently. Eventually, one of them must stop. If the program that stops is p, say that x is in L. If the program that stops is q, say that x is not in L.

Next, suppose that L is computable. It is easy to show that L is both partially computable and co-partially computable. For the former result, just take a program that decides p and make it loop forever when it is about to stop and answer 'no'. For the latter result, make p loop forever when it is about to answer 'yes'.

♢

Later, we will see another class that shares the asymmetry in its definition as the class of partially computable sets and has a notion of most difficult problems. Whether or not an analog of Theorem 13.6 holds is critical to whether there exist public-key cryptosystems.

prev **next**