1 More Unsolvable Problems

Many other problems can be shown to be unsolvable given the unsolvability of the halting problem. This is important to know, because it saves us the effort of trying to find algorithms for problems that are unsolvable. For such problems, one can find solutions for special cases, or find approximate solutions or solutions that work some of the time.

We illustrate the idea of finding other unsolvable problems on the blank tape halting problem, then give the general approach.

1.1 Blank Tape Halting Problem

The blank tape halting problem is, given a Turing machine T, does T halt when it starts on a blank tape? That is, does T halt when its input is ϵ, the empty string? As a language, this probem can be expressed as $BTHP =$

$$\{encode(T) : T \text{ halts when started on blank tape } \}$$

and the question is whether this language is decidable. It is conceivable that deciding halting on blank tape might be easier than deciding it in general; after all, in some cases, the halting problem is easier. For example, the halting problem can be decided if the Turing machine never writes on the tape, or if it has only one state.

1.2 Undecidability of the Blank Tape Halting Problem

Now, to show that the language $BTHP$ is undecidable (that is, not decidable), consider a Turing machine

$$sim(T, x)$$

which, given an input y on the tape, erases y, writes x on the tape, and transfers control to T. (The book calls this machine T_x.) Thus if $sim(T, 101)$ were called with an input of 11101 on the tape, then it would erase the 11101 from the tape, write 101 on the tape, and then transfer control to T. So initially the tape would look like this:

$sim(T, 101)$:

▷	⊔	1	1	1	0	1	⊔	⊔	⊔	⊔	⊔	⊔	...

Then the input would be erased:

$sim(T, 101)$:

Then 101 would be written on the tape and control would be transferred to T:

T :

Then either T would halt on the input 101, or T would not halt. Thus

> $sim(T, 101)$ halts on input 11101 iff T halts on input 101.

Now, suppose that instead of the input 11101, $sim(T, 101)$ is given a blank tape as input. Initially the tape would look like this:

$sim(T, 101)$:

Then the input would be erased; of course, there is nothing to erase, so after this the tape would look the same:

$sim(T, 101)$:

Then 101 would be written on the tape and control would be transferred to T:

T :

Then either T would halt on the input 101, or T would not halt. Thus

> $sim(T, 101)$ halts on blank tape iff T halts on input 101.

Suppose we had a "blank tape halting tester" that could test if an arbitrary Turing machine halted when started on blank tape. Then we could use this "blank tape halting tester" to test if $sim(T, 101)$ halts on blank tape. Because $sim(T, 101)$ halts on blank tape iff T halts on input 101, we can use

this "blank tape halting tester" to test if T halts on input 101. If $sim(T, 101)$ halts on blank tape, then T halts on input 101, and if $sim(T, 101)$ does not halt on blank tape, then T does not halt on input 101.

This result really does not depend on the input 101 at all, or on T; it would work for any Turing machine T and any input x to T:

$sim(T, x)$ halts on blank tape iff T halts on input x.

Thus, a "blank tape halting tester" would give us a way to test if an arbitrary Turing machine T halts on an arbitrary input x by testing if $sim(T, x)$ halts on blank tape using the "blank tape halting tester." If $sim(T, x)$ halts on blank tape, then T halts on input x, and if $sim(T, x)$ does not halt on blank tape, then T does not halt on input x. But the halting problem is unsolvable, which means that it is impossible to test if an arbitrary Turing machine T halts on an arbitrary input x. Therefore there can be no such "blank tape halting tester," so the blank tape halting problem, the problem of testing if an arbitrary Turing machine halts on blank tape, is also unsolvable.

So here is our method to test if T halts on input x:

1. Construct the encoding of $sim(T, x)$ from the encodings of T and x
2. Test if $sim(T, x)$ halts on blank tape
3. If $sim(T, x)$ halts on blank tape, halt in state y else halt in state n

This shows that if the blank tape halting problem is solvable, the original halting problem is solvable. Therefore the blank tape halting problem is not solvable (not decidable).

Note that this method depends on the fact that the encoding of $sim(T, x)$ is computable from the encodings of T and x. So the method of finding new unsolvable problems, depends on the existing of a computable function from the old problem to the new problem, in some sense. Later this idea is made more formal by the concept of a *reduction*.

Problem: Take an example Turing machine T from the text and compute the description of the Turing machine $sim(T, 101)$.

Problem: In some high level language, write a program which, when given the description of a Turing machine T and an input x, outputs the description of $sim(T, x)$. Run the program on a few examples to check its correctness.

Problem: Write the description of a Turing machine M which, when given the description of a Turing machine T and an input x, outputs the description of $sim(T, x)$. Run the machine M on a few examples to check its correctness.

1.2.1 Another Approach

Now let's look at the problem another way. Recall from above that

$sim(T, 101)$ halts on input 11101 iff T halts on input 101.

Because $sim(T, 101)$ does not look at its input,

For all inputs y, $sim(T, 101)$ halts on input y iff T halts on input 101.

This result would be true for any other string x in place of 101, so we have that

For all strings x and y, $sim(T, x)$ halts on input y iff T halts on input x.

Therefore, setting y to the empty string, we have that

For all strings x, $sim(T, x)$ halts on blank tape iff T halts on input x.

Then, from the definition of $BTHP$ given above,

For all strings x, $encode(sim(T, x)) \in BTHP$ iff T halts on input x.

Let $H = \{encode(T)encode(x) : T \text{ halts on input } x\}$. The language H expresses the halting problem. Then

For all strings x, $encode(sim(T, x)) \in BTHP$ iff
$encode(T)encode(x) \in H$.

Also,

$encode(sim(T, x))$ can be computed from $encode(T)$ and $encode(x)$.

Thus if we can decide if $encode(sim(T, x)) \in BTHP$ then we can decide if $encode(T)encode(x) \in H$ by computing $encode(sim(T, x))$ from $encode(T)encode(x)$ and then testing if $encode(sim(T, x)) \in BHTP$.

Now, we know that the Halting Problem is undecidable. This means that the language H is not decidable. But if the blank tape halting probem ($BTHP$) were decidable, H would also be decidable. Here's how to decide H given a way to decide the blank tape halting problem:

Given $encode(T)encode(x)$,

1. Compute $encode(sim(T, x))$.
2. Test if $encode(sim(T, x)) \in BTHP$ which can be done if $BTHP$ is decidable.
3. If $encode(sim(T, x)) \in BTHP$ then answer "yes" else answer "no".

This works because if T halts on x, then $sim(T, x)$ halts on blank tape, so $encode(sim(T, x)) \in BTHP$. If T loops on x,then $sim(T, x)$ loops on blank tape, so $encode(sim(T, x)) \notin BTHP$. Thus we have that

H is decidable if $BTHP$ is decidable.

Therefore,

Because H is undecidable, $BTHP$ is also undecidable.

1.3 General Method to Show Undecidability

We now generalize the above reasoning to give a general method for showing that problems are undecidable. The idea is this: Given that one language L_1 is undecidable, to show that another language L_2 is undecidable, give a procedure to decide L_1 given a way to decide L_2. Because L_1 is undecidable, there can be no such way to decide L_2. Therefore L_2 is undecidable also. In the above case, L_1 was H and L_2 was $BTHP$.

The method we present is based on *reductions*, which are computable functions from one language to another. By now thousands and thousands of problems have been shown to be unsolvable by using reductions starting from the halting problem.

The same method can show that problems are not partially solvable by starting from the complement of the halting problem, but we will not discuss this much. The idea for showing that a languuage L' is not partially decidable is that if a language L is partially decidable and not decidable, then its complement is not even partially decidable.

1.3.1 Reductions

Definition 1.1 (Reductions) *A* reduction *from language $L_1 \subseteq \Sigma^*$ to $L_2 \subseteq \Sigma^*$ is function τ such that*

1. *τ maps Σ^* to Σ^**

2. *τ is recursive (computable)*

3. *for all $x \in \Sigma^*$, $x \in L_1$ iff $\tau(x) \in L_2$.*

Theorem 1.1 (5.4.1) *If L_1 is not recursive and there is a reduction from L_1 to L_2 then L_2 is not recursive either.*

Proof: If L_2 were recursive (decidable), then we could decide L_1 as follows:

1. Given x, compute $\tau(x)$.
2. Test if $\tau(x) \in L_2$.
3. If $\tau(x) \in L_2$ then answer "yes" else answer "no".

This works because τ is computable, so we can compute $\tau(x)$, and it gives the right answer because for all $x \in \Sigma^*$, $x \in L_1$ iff $\tau(x) \in L_2$. Therefore, because L_1 is not decidable, L_2 must not be decidable either.

In the blank tape halting problem example given above, τ maps $encode(T)encode(x)$ to $encode(sim(T, x))$.

1.3.2 General Recipe

Here is a general recipe for showing a problem to be unsolvable:

1. Express the problem as a language L_2 over some alphabet Σ^*.

2. Choose another language L_1 over Σ^* that is known not to be recursive.

3. Construct a function τ and show that it has the properties of a reduction from L_1 to L_2.

L_1 should be chosen to be similar to L_2. Then choose τ to express the similarity between the two languages. Thus $\tau(x)$ is the problem in L_2 that corresponds to the problem x in L_1.

1.4 Unsolvable Problems about Turing Machines

Using the reduction technique, the following problems can be shown unsolvable (Theorem 5.4.2 page 255):

1. Given Turing machine M, is there any string w such that M halts on w?
2. Given M, does M halt on all inputs?
3. Given M_1 and M_2, do they halt on the same inputs?
4. Given M, is the language that M semi-decides, regular? context-free? recursive?
5. There is a fixed machine M such that the question whether M halts on input x is undecidable.

1.5 Unsolvable Problems about (General) Grammars

Unsolvability results can also be shown about grammars, using reductions. There are from theorem 5.5.1 in the text. These problems are unsolvable.

1. Given a grammar G and a string w, is $w \in L(G)$?
2. Given a grammar G, is $\epsilon \in L(G)$?
3. Given grammars G_1 and G_2, is $L(G_1) = L(G_2)$?
4. Given a grammar G, is $L(G) = \phi$?
5. There is a fixed grammar G such that it is undecidable given w whether $w \in L(G)$.

1.6 Unsolvable Problems about Context-Free Grammars

There are from theorem 5.5.2. The following are undecidable:

1. Given a context-free grammar G, is $L(G) = \Sigma^*$?
2. Given two cfg G_1 and G_2, is $L(G_1) = L(G_2)$?

3. Given two push-down automata M_1 and M_2, is $L(M_1) = L(M_2)$?

4. Given a push-down automaton M, find an equivalent push-down automaton with as few states as possible.

Here are some more unsolvable problems about context-free grammars:

Given a context-free grammar G, is G ambiguous?

Given context-free grammars G_1 and G_2, is $L(G_1) \cap L(G_2) = \phi$?

Some of the preceding problems are ones that we would very much like to be able to solve. Some problems are in areas not related to grammars or Turing machines at all. For example, Hilbert's Tenth problem has to do with diophantine equations, and was shown to be unsolvable in the 1970's by a very complicated series of reductions.

Hilbert's tenth problem is to give a computing algorithm which will tell of a given polynomial Diophantine equation with integer coefficients whether or not it has a solution in integers. Matiyasevic proved that there is no such algorithm.

`http://www.math.umd.edu/~laskow/Pubs/713/Diophantine.pdf`

1.7 Rice's Theorem

Rice's theorem (theorem 5.7.4) gives a general mechanism for showing that many problems are unsolvable. Here is one way it can be stated:

Theorem 1.2 *Let S be a set of Turing machines such that*

1. *There is at least one Turing machine in S*

2. *There is at least one Turing machine not in S*

3. *If M_1 and M_2 are two Turing machines and $L(M_1) = L(M_2)$, then $M_1 \in S$ iff $M_2 \in S$.*

Then the language $\{encode(M) : M \in S\}$ is not decidable.

Recall that $L(M)$ is the set of strings x such that M halts on input x.

Give examples of some non-semantic properties of Turing machines that are decidable.