Lecture 12: Undecidable Languages

Ryan Bernstein

1 Introductory Remarks

- Assignment 3 is available online, and is due a week from today (05/19)

1.1 Recapitulation

On Tuesday, we built a lot of Turing machines. These Turing machines decided languages based on encodings of other machines or objects. They included languages like:

- $\text{A}_{\text{DFA}} = \{\langle D, w\rangle \mid D \text{ is a DFA that accepts } w\}$
- $\text{E}_{\text{DFA}} = \{\langle D\rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}$
- $\text{ALL}_{\text{DFA}} = \{\langle D\rangle \mid D \text{ is a DFA and } L(D) = \Sigma^*\}$
- $\text{A}_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a context-free grammar capable of generating } w\}$

The key point here was that, since Turing machines are algorithms, we could use any algorithm that we've seen before as a step when constructing a machine M. This includes things like conversion to Chomsky normal form or between machines equivalent in power, but it also includes the Turing machines that we're constructing themselves. Once we've constructed as a machine, we can use that machine as a subroutine to solve larger problems, just as we can with programs, functions, and libraries that we write while programming.

By constructing Turing machines that decided all of these languages, we showed that they were Turing-decidable. Today, we'll be looking at languages that are *not* Turing-decidable. To do this, we'll need to start by showing that such languages actually exist.

2 Countable and Uncountable Infinity

First, a return to CS 250. In CS 250, we discussed the idea of infinite sets. We broke infinite sets into two categories: countably infinite and uncountably infinite. Countably infinite sets are those that can be put into a one-to-one mapping with the natural numbers or some subset thereof. Obviously, the natural numbers themselves fall into this category. We can also see that $\mathbb{Z}$ is countably infinite by generating a function $f : \mathbb{Z} \to N$:

$$f(x) = \begin{cases} 2x - 1 & x > 0 \\ -2x & x \le 0 \end{cases}$$

This generates a mapping like the following:

x	-3	-2	-1	0	1	2	3
$f(x)$	6	4	2	0	1	3	5

What's the point of creating such a mapping? Countably infinite languages are so named because we can enumerate all of their members by counting. Once we've mapped the set to the natural numbers, we can reach every member of the set simply by starting at zero and counting up. While this enumeration may never finish, we will *eventually* hit any arbitrary element.

Uncountably infinite sets, then, are sets that *cannot* be mapped to the natural numbers. The canonical example of this is the set of all infinite-length binary strings. We call this set uncountably infinite because of an argument called Cantor's Diagonalization. This assumes that we have some enumeration of all infinite-length binary strings, like so:

$$\begin{array}{rccccccc} s_1 = & 0 & 1 & 0 & 1 & 0 & 0 & ... \\ s_2 = & 1 & 1 & 0 & 1 & 0 & 0 & ... \\ s_3 = & 0 & 1 & 1 & 0 & 1 & 1 & ... \\ ... & & & & & & & \\ s_n = & 1 & 1 & 1 & 0 & 1 & 0 & ... \\ ... & & & & & & & \end{array}$$

We can generate another binary string s_d by inverting the first character of s_1, the second character of s_2, and so on.

$$\begin{array}{rccccccc} s_1 = & \underline{0} & 1 & 0 & 1 & 0 & 0 & ... \\ s_2 = & 1 & \underline{1} & 0 & 1 & 0 & 0 & ... \\ s_3 = & 0 & 1 & \underline{1} & 0 & 1 & 1 & ... \\ ... & & & & & & & \\ s_n = & 1 & 1 & 1 & 0 & 1 & 0 & ... \\ ... & & & & & & & \end{array}$$

$$s_d = 100...$$

Because s_d differs from *every string in this enumeration* at the point of diagonalization, we know that s_d will not have already been a member of this enumeration at any point. In other words, our enumeration that lists all possible infinite-length binary strings cannot have generated s_d, even though it is itself an infinite-length binary string.

The important part of this distinction, for our purposes, is that even though countable and uncountable sets can both have infinite cardinalities, an uncountably infinite set has many, many more members than does a countably infinite one.

3 Turing Machines and Languages

Now that we've established the difference between countable and uncountable infinity, we can make some statements about Turing machines and languages.

Theorem The set of all Turing machines is countably infinite.

We say that a set is countably infinite if we can establish a one-to-one mapping between it and some subset of the natural numbers. We learned in CS 250 that we can encode any object that can be represented with finite precision using only the alphabet $\{0,1\}$. We saw examples of the things that this allowed on Tuesday.

If we let $\langle G \rangle$ be the encoding of a Turing machine using $\{0,1\}$, we can also interpret $\langle G \rangle$ as a (probably very large) binary representation of a natural number n. By enumerating the natural numbers, we can therefore enumerate the binary representation of every possible Turing machine.

The set of all Turing machines is therefore countably infinite.

Theorem Σ^* is countably infinite.

We can use similar logic to show that Σ^* is countably infinite. We can enumerate all of the strings in Σ^* by generating them in what's known as *shortlex order*. Shortlex order orders strings first by length and then by standard lexicographical (i.e. dictionary) order. Mapping the strings in $\{0,1\}^*$ to the natural numbers might then look something like this:

Index	0	1	2	3	4	5	6	...
String	ϵ	0	1	00	01	11	000	...

Theorem The set of all languages is uncountably infinite.

We can now show that the set of all languages is uncountable. We define a language L as a subset of Σ^*, which means that every element of Σ^* is either present or absent in L. We can therefore draw L as a bit vector parallel to the enumeration of Σ^* that we just created. Each bit in this vector represents the presence or absence in L of the corresponding element of Σ^*. As an example, let $A = \{w \in \{0,1\}^* \mid w \text{ contains an even number of zeros}\}$. We can represent A as a bit vector parallel to Σ^* like this:

Index	0	1	2	3	4	5	6	...
String	ϵ	0	1	00	01	11	000	...
Present in A?	1	0	1	1	0	1	0	...

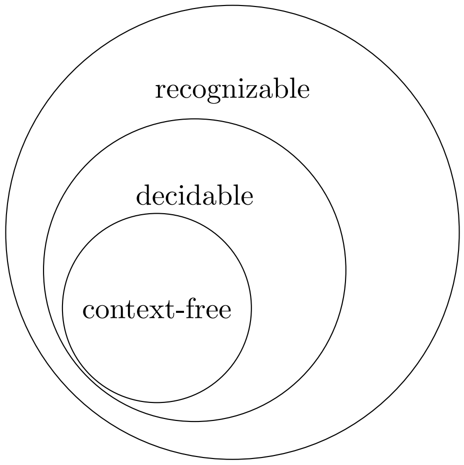
We can generalize this logic to say that any powerset of a countably infinite set is uncountable.

Conclusion There are many, many more languages than Turing machines.

Since the set of all languages is uncountably infinite and the set of all Turing machines is countable, there are many more languages than there are Turing machines. This means that there must be some languages for which no Turing machine exists. We can therefore conclude that there exist languages that are *not* Turing-recognizable.

4 $\mathbf{A_{TM}}$

We've drawn our language hierarchy like this, with Turing-recognizable languages as a strict superset of Turing-decidable ones:



As a reminder, our distinction was this:

A Turing machine M *decides* a language L if and only if:

1. $\forall s \in L$, M Accepts s
2. $\forall s \notin L$, M Rejects s

A Turing machine M *recognizes* a language L if and only if:

1. $\forall s \in L$, M Accepts s

We haven't actually shown that there's a meaningful distinction here. To prove that the set of all Turing-recognizable languages is a proper superset of the set of all Turing-decidable languages, we'll examine the language A_{TM}.

Let $\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle \mid M$ is a Turing machine and M Accepts $w\}$.

4.1 $\mathbf{A}_{\mathbf{TM}}$ is Turing-Recognizable

Building a recognizer for A_{TM} is pretty simple. We can use exactly the same strategy that we used for languages like $\mathrm{A}_{\mathrm{DFA}}$ and $\mathrm{A}_{\mathrm{NFA}}$.

```
$M$ = "On input $\langle M, w\rangle$:
    1. Simulate $M$ on $w$.
        • If $M$ accepts $w$, ACCEPT
        • If $M$ rejects $w$, REJECT"
```

This works just fine for DFAs, NFAs, or PDAs. But since Turing machines have the option of looping forever rather than entering q_{accept} or q_{reject}, we have a third case to take into account.

Note that we did *not* use the word "otherwise" or "else" (e.g. "If M accepts w, Accept. Otherwise, Reject") to determine when to Reject. If we're building a *decider* for a language, it's fine to use these words, because we know that every step should terminate. Since the simulation of M on w may never terminate, though, we cannot use this same strategy when constructing a recognizer.

4.2 $\mathrm{A_{TM}}$ is Not Turing-Decidable

Showing that $\mathrm{A_{TM}}$ is undecidable is a bit trickier. First, we'll assume that $\mathrm{A_{TM}}$ is decidable. Then there is a machine, $M_{A_{TM}}$, that decides it.

We can now construct a machine D as follows:

$D =$ "On input $\langle M\rangle$:

1. Simulate $M_{A_{TM}}$ on $\langle M, \langle M\rangle\rangle$.
    - If $M_{A_{TM}}$ accepts $\langle M, \langle M\rangle\rangle$, REJECT
    - If $M_{A_{TM}}$ rejects $\langle M, \langle M\rangle\rangle$, ACCEPT"

D takes an encoding of a machine as input. It uses $M_{A_{TM}}$ to determine whether or not that machine accepts an encoding of itself. If so, D rejects the machine. If not, D accepts.

What happens if we run D with input $\langle D\rangle$?

- If D accepts $\langle D\rangle$, then D must REJECT $\langle D\rangle$
- If D rejects $\langle D\rangle$, then D must ACCEPT $\langle D\rangle$.

Clearly, this is paradoxical. Since the construction of D was enabled by our assumption that $M_{A_{TM}}$ existed and this resulted in a contradiction, we can conclude that our assumption was incorrect, and $\mathrm{A_{TM}}$ is undecidable.

4.3 $\mathrm{A_{TM}}$'s Implications for Recognizability

We can now conclude that the set of recognizable (but not decidable) languages contains at least one member. $\mathrm{A_{TM}}$ is not only the first language in this class that we've seen, but also the most important.

Assume that $\mathrm{A_{TM}}$ was decidable, but that some other undecidable language B was Turing-recognizable. Then there would exist a machine M_B that recognized (but did not decide) B.

We could clearly construct a decider for B by running $M_{A_{TM}}$ on $\langle M_B, w\rangle$. In other words, if A_{TM} was decidable, then *every* Turing-recognizable language would also be decidable.

5 Recognizability vs. Decidability

Theorem If L is Turing-recognizable and $\overline{L}$ is Turing-recognizable, then L is Turing-decidable.

If L and $\overline{L}$ were both Turing-recognizable, we could construct a decider for L like so:

Let M_L be a machine that recognizes L and $M_{\overline{L}}$ be a machine that recognizes $\overline{L}$. We'll create a nondeterministic machine that decides L. $M =$ " On input w:

1. Simulate M_L and $M_{\overline{L}}$ in parallel.
    - If M_L accepts w, ACCEPT
    - If $M_{\overline{L}}$ accepts w, REJECT"

Since we know that either $w \in L$ or $w \in \overline{L}$ and we have recognizers for both languages, we can create a decider for L by determining which machine recognizes w. As a corollary, we can say that if L is recognizable but not decidable, then $\overline{L}$ is *not* recognizable. Otherwise, the class of decidable languages would equivalent to the class of recognizable languages, which we've just shown to be false.

This lets us conclude the following: Since A_{TM} is recognizable, but not decidable, $\overline{\mathrm{A}_{\mathrm{TM}}}$ must be undecidable. This will be important later, but for now, we can see that it validates the theorem that we proved using diagonalization at the beginning of the lecture: there exist languages that are not Turing-recognizable, and that therefore exist outside of all of the circles on our language hierarchy.

6 Reductions

We've now proven the existence of two undecidable languages (A_{TM} and $\overline{\mathrm{A}_{\mathrm{TM}}}$) and one unrecognizable language ($\overline{\mathrm{A}_{\mathrm{TM}}}$). The proof for both of these was a bit involved, and required us to generate a paradox.

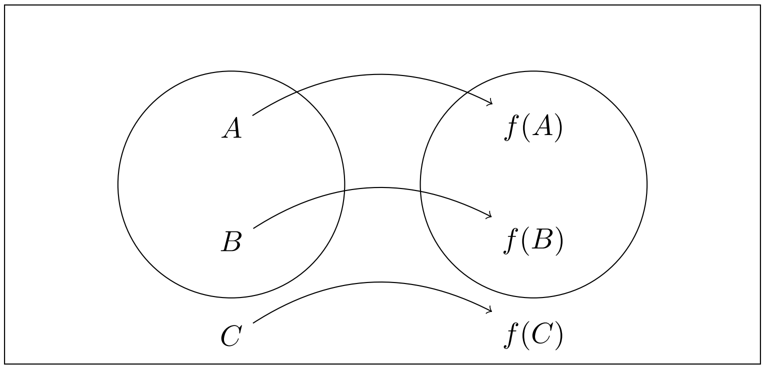
The good news is that we need not do this for *every* undecidable or unrecognizable language; we have a strategy that we can use to place languages in these categories. The bad news is that this process is probably the single most difficult concept in this course.

We say that a problem A is *reducible* to another problem B if a solution to B can be used to solve A.

To get even more specific, we say that a language A is *mapping reducible* to a language B if there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that:

1. $\forall s(s \in A \to F(s) \in B)$
2. $\forall s(s \notin A \to F(s) \notin B)$

In other words, F is a function whose result is in B if and only if its input was in A:



There's one element that we've glossed over, which is the definition of a computable function. We say that $f : \Sigma^* \to \Sigma^*$ is a computable function if there exists a Turing machine M that, on input w, halts with $f(w)$ on its tape.

If A is mapping reducible to B, we write $A \leq_m B$. To show that this is the case, we need only construct a Turing machine F that:

- Given some $w \in A$ outputs an $s \in B$
- Given some $w \notin A$ outputs an $s \notin B$

6.1 Creating Mapping Reductions

Since A_{TM} is our prototypical undecidable language, we'll show that another language B is undecidable by creating a mapping that shows that $\mathrm{A}_{\mathrm{TM}} \leq_m B$. This shows that if we could solve B, we could also solve A_{TM}.

All such mapping reductions work by implying the existence of a machine $M_{A_{TM}}$ that decides A_{TM}. As such, all mapping reductions use the same proof by contradiction, written below:

Proof Assume that B is Turing-decidable. Then there exists a machine M_B that decides it.

We can create construct a machine $M_{A_{TM}}$ that decides A_{TM} as follows:

```
$M_{A_{TM}}$ = "On input $\langle M, w \rangle$:
  1. Run $M_B$ on $F(\langle M, w \rangle)$.
       • If $M_B$ accepts $F(\langle M, w \rangle)$, ACCEPT
       • REJECT"
```

Since we've already shown that A_{TM} is undecidable, we've reached a contradiction. Our assumption is therefore false, and B is undecidable.

6.2 The Halting Problem

Many of you were in David Lu's section of Computer Science 251, for which I was a TA. During my lectures on program correctness, we discussed the Halting Problem: given an arbitrary input, does this program halt, or loop infinitely? We called the Halting Problem a canonical example of an uncomputable problem. We will prove that the Halting Problem is undecidable now, using a mapping reduction.

Proof Let $\mathrm{HALT}_{\mathrm{TM}} = \{\langle M, w \rangle \mid M \text{ is a Turing machine and } M \text{ halts on input } w\}$. Show that HALT_{TM} is undecidable.

To do this, we'll demonstrate that $\mathrm{A}_{\mathrm{TM}} \leq_m \mathrm{HALT}_{\mathrm{TM}}$. Doing this is as simple as constructing a Turing machine that, given an input $\langle M, w \rangle$ produces an output $\langle M', x \rangle$ such that M' halts on x if and only if M accepts w.

```
$F$ = "On input $\langle M, w \rangle$:
  1. Construct a machine $M'$ as follows: $M'$ = "On input $x$:
       (a) Simulate $M$ on $w$.
             • If $M$ accepts $w$, ACCEPT $x$
             • If $M$ rejects $w$, enter an infinite loop."
  2. Output $\langle M', w \rangle$."
```

Note that F creates M' and outputs it without ever running it. That means that F will always halt, even if the simulation step in M' does not.

What is $L(M')$? We have three cases to consider.

Case 1: M accepts w

> If M accepts w, then M' accepts x, regardless of what x actually is. This means that $L(M') = \Sigma^*$. Since M' accepts all inputs, M' halts on all inputs.
>
> This means that when we output $\langle M', w\rangle$, we've output a machine and a word on which that machine will halt. This in turn means that $\langle M', w\rangle \in \textsc{Halt}_{\text{TM}}$.

Case 2: M rejects w

> If M rejects w, M' enters an infinite loop, regardless of the value of x. This means that M' loops on all inputs. Outputting $\langle M', w\rangle$ is therefore outputting a machine and a word on which that machine will *not* halt. In this case, $\langle M', w\rangle \notin \textsc{Halt}_{\text{TM}}$.

Case 3: M loops on w

> In this case, the simulation step of M' will never terminate, and M' will loop regardless of the value of x. This means that M' does not halt on w, and $\langle M', w\rangle \notin \textsc{Halt}_{\text{TM}}$.

This means that the *only* case in which we output a machine and a word on which that machine halts will be Case 1, when M accepts w. As discussed before, this allows us to create a following decider for A_{TM}.

Assume that $\textsc{Halt}_{\text{TM}}$ is decidable. Then there is a Turing machine H that decides it.

$M_{A_{TM}}$ = "On input $\langle M, w\rangle$:

1. Run H on $F(\langle M, w\rangle)$.
    - If H accepts $F(\langle M, w\rangle)$, ACCEPT
    - REJECT

Since we know that A_{TM} is undecidable, we've reached a contradiction. We can therefore conclude that our initial assumption was false, and that $\textsc{Halt}_{\text{TM}}$ is undecidable.

Soluzioni Tutorato 08

Giulio Umbrella

Ex 1

Dimostrare che il seguente linguaggio e' indecidibile $L_R =$ (M,w | M accetta la stringa ww^R)

Soluzione

Per dimostrare che L_R e' indecidibile dimostriamo che esiste una riduzione $A_{TM} \leq_m L_R$. La funzione f opera nel seguente modo:

F = su input M,w dove M e' una TM e w una stringa:

```
1. Costruisci la seguente $M_R$:
   $M_R$ su input x:
     1. se $x \neq ww^R$ accetta.
     2. Se $x = ww^R$, esegue M su input w.
     3. Se $M$ accetta, *accetta*
     4. Se $M$ rifiuta, *rifiuta*
2. Restituisci $M_R, w$:
```

Dimostriamo che f e' una riduzione valida:

1. Se $(M, w) \in A_{TM}$ allora la TM M accetta w. Quindi la macchina M_R costruita accetta la parola ww^R. Quindi f(M,w) = $M_R \in L_R$
2. Se $(M, w) \notin A_{TM}$ allora la TM M non accetta w. Quindi la macchina M_R costruita non accetta la parola ww^R. Quindi f(M,w) = $M_R \notin L_R$

Che cosa rappresenta x?

La variabile x rappresenta l'input della TM M_R quando viene messa in esecuzione. E' importante sottolineare cha la variabile x **non** viene dato in input alla funzione F; gli input di F sono infatti la TM e la stringa w. Dobbiamo specificare il valore x per indicare il comportamento di M_R. M_R e' una TM e per funzionare ha bisogno di un input. Il restanti passaggi spiegano cosa fa M_R una volta che riceve un input, ma quella e' una computazione a se stante che non viene eseguita da F.

Ex 2

Pebbling e' un solitario giocato su un grafo non orientato G, in cui ogni vertice ha zero o piu' ciottoli. Una mossa del gioco consiste nel rimuovere due ciottoli da un vertice v e aggiungere un ciottolo ad un vertice u adiacente a v (il vertice v deve avere almeno due ciottoli all'inizio della mossa). Il problema PebbleDestruction chiede, dato un grafo G = (V,E) ed un numero di ciottoli p(v) per ogni vertice v, di determinare se esiste una sequenza di mosse che rimuove tutti i sassolini tranne uno.

Dimostra che PebbleDestruction e' **NP-hard** usando il problema del Circuito Ha- miltoniano come problema NP-hard noto (un circuito Hamiltoniano e' un ciclo che attraversa ogni vertice di G esattamente una volta).

Soluzione

Per dimostrare che un problema X e' NP-Hard dobbiamo prendere un problema Y che sappiamo essere NP-Hard A noto e dimostrare che si puo' effettuare una riduzione in tempo polinomiale.

La soluzione generale di un esercizio di riduzione a partire da un linguaggio NP-Hard e' composta da tre sotto punti:

1. Descrivere un algoritmo che prende una **qualsiasi** istanza di X e la trasforma in un'istanza **speciale** di Y
2. Dimostrare che una buona istanza di X e' convertita in una buona istanza di Y
3. Dimostrare che una buona istanza di Y e' convertita in una buona istanza di X

L'algoritmo di conversione e' generale, ma la dimostrazione verte su istanze particolari del problema.

- Dato un certificato di X, dobbiamo dimostrae

Algoritmo Conversione

Dato i nodi del grafo, trasformiamo i nodi aggiungendo un'informazione nel seguente modo:

1. Dato un vertice il valore 2 - rappresenta due sassolini
2. In tutti i nodi tranne l'ultimo, inseriamo il valore 1 - rappresenta un sassolino
3. Nell'ultimo nodo inseriamo il valore 0 - nessun sassolino
4. Rimuoviamo le direzioni dagli archi.

Dimostrazione Hamilton → PebbleDestruction

Se abbiamo una soluzione per un cammino sul grafo orientato, abbiamo una dimostrazione per il PebbleDestruction.

Nel grafo che otteniamo possiamo rimuovere tutti i sassolini tranne uno partendo dal nodo con due sassolini, rimuovendo i due sassolini e aggiungendone uno al nodo successivo del circuito. L'ultimo nodo non ha sassolini e ne riceve uno per via della mossa precedente.

La dimostrazione verte sul fatto che il certificato che abbiamo contiene un circuito Hamiltoniano, quindi possiamo seguirne i vertici seguendo le regole del gioco.

Dimostrazione PebbleDestruction $\rightarrow$ Hamilton

In questo caso il certificato di partenza e' un circuito Hamiltoniano, a cui aggiungiamo i ciottoli. La sequenza parte necessariamente dal vertice con due sassolini - per le condizioni del gioco - e prosegue visitando gli altri vertici. Dato che una volta che visito un vertice non posso piu' visitarlo - non ci sono ciottoli con cui fare una mossa - e dal momento che li visito tutti ottengono una soluzione per il ciclo Hamiltoniano.

Claim 1. The following language is undecidable.

$$\text{TOTAL} = \{\langle M\rangle \mid M \text{ is a TM that halts on all inputs}\}$$

Proof. We prove the claim is true by contradiction.

Assume to the contrary that TOTAL is decidable by the Turing machine M_{TOTAL}. We will now construct a Turing machine that decides the following language, which we've already seen is undecidable:

$$A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$$

Consider the following Turing machine:

```
$N =$ "On input $\langle M, w\rangle$ where $M$ is a TM,
    1. Construct the TM $M^*$ defined by
        $M^* =$ "On input $x$
            1. Run $M$ on $w$
            2. If $M$ accepts, then ACCEPT
            3. If $M$ rejects, then enter an infinite loop"
    2. Run $M_{\text{TOTAL}}$ on $\langle M^*\rangle$
    3. If it accepts, then ACCEPT; otherwise REJECT"
```

We now prove that N is a decider for A_{TM}.

Consider a string $\langle M, w\rangle \in A_{\text{TM}}$. By definition, we know that M accepts w; thus, the machine M^* will accept every input strings x and therefore halts on every input. This means that M^* is total, and therefore $\langle M^*\rangle \in$ TOTAL. Since M_{TOTAL} is a decider for TOTAL, this means that it will accept the input $\langle M^*\rangle$, and therefore the Turing machine N will accept $\langle M, w\rangle$.

It remains to be shown that if $\langle M, w\rangle \notin A_{\text{TM}}$ then the machine N rejects. Given such an input, we know that M does not accept w and either will reject or get into an infinite loop. Thus, the machine M^* will always loop forever, because even if M halts and rejects, then M^* will enter an infinite loop. This means that M^* is not total and therefore $\langle M^*\rangle \notin$ TOTAL. Thus, M_{TOTAL} will reject $\langle M^*\rangle$, and so N will reject $\langle M, w\rangle$.

Since N will accept an input $\langle M, w\rangle$ if it is in A_{TM} and it will reject all other strings, it is a decider for A_{TM}. However, we know that A_{TM} is undecidable—a contradiction.

□

Claim 2. The following language is undecidable.

$$S = \{\langle M\rangle \mid M \text{ is a TM that accepts } w \text{ if and only if it accepts } w^R\}$$

Proof. It suffices to show that $A_{\text{TM}} \leq_m S$, since A_{TM} is undecidable. This would then imply that S is also undecidable.

To prove that $A_{\text{TM}} \leq_m S$, we must show that there is a computable function f satisfying that $\langle M, w\rangle \in A_{\text{TM}}$ if and only if $f(\langle M, w\rangle) = \langle\widehat{M}\rangle \in S$. We define this function in the following way:

$$f(\langle M, w\rangle) = \langle\widehat{M}\rangle = \begin{cases} \text{On input } x, \\ \quad 1.\ \text{If } x = 01, \text{ then ACCEPT} \\ \quad 2.\ \text{If } x = 10, \text{ then} \\ \qquad \text{a. Run } M \text{ on } w \\ \qquad \text{b. If it accepts, then ACCEPT} \\ \qquad \text{c. Otherwise REJECT} \\ \quad 3.\ \text{If } x \text{ is any other string, then REJECT} \end{cases}$$

Notice that if M accepts w, then $L(\widehat{M}) = \{01, 10\}$, and therefore $\langle\widehat{M}\rangle \in S$. Similarly, if M does not accept w, then $L(\widehat{M}) = \{01\}$, and therefore $\langle\widehat{M}\rangle \notin S$. Thus, $\langle M, w\rangle \in A_{\text{TM}}$ if and only if $f(\langle M, w\rangle) \in S$.

We also note that f is computable since it only requires a finite number of steps to transform a finite description $\langle M, w\rangle$ into the description $\langle\widehat{M}\rangle$, so it will always halt on all inputs. This means that f is a reduction from A_{TM} to S, meaning that $A_{\text{TM}} \leq_m S$. This immediately implies that S is undecidable. $\square$

Theoretical Computer Science (Bridging Course)

Dr. G. D. Tipaldi
F. Boniardi
Winter semester 2014/2015

University of Freiburg
Department of Computer Science

Exercise Sheet 7

Due: 18th December 2014

Exercise 7.1 (Decidable Languages)
Let L and L' be decidable languages. Prove the following properties.

(a) The complement $\overline{L}$ is decidable.

Solution: Let $L = L(M)$ for some Turing Machine M that always halts. We construct a TM $\overline{M}$ that $\overline{L} = L(\overline{M})$ as follows

1. The accepting states of M are made nonaccepting states of $\overline{M}$ with no transitions, i.e., in these states $\overline{M}$ will halt without accepting.
2. $\overline{M}$ has a new accepting state, say r; there are no transitions from r.
3. For each combination of a nonaccepting state of M and a tape symbol of M such that M has no transition, add a transition to the accepting state r.

Since M is guaranteed to halt, we know that $\overline{M}$ is also guaranteed to halt. Moreover, $\overline{M}$ accepts exactly those strings that M does not accept. Thus, $\overline{M}$ accepts $\overline{L}$

(b) The union $L \cup L'$ is decidable.

Solution: Since L and L' are decidable, there exist Turing Machines M and M' that decide L and L' respectively. Thus, we can construct a non-deterministic Turing Machine $M_\cup$ that runs M and M' in parallel. It is easy to see that such TM decides the language $L \cup L'$.

Exercise 7.2 (Decidable Languages)
Show that the following languages are decidable:

(a) $EQ_{DFA\_RE} = \{\langle D, R\rangle \mid D \text{ is a DFA and } R \text{ is a regular expression and } L(D) = L(R)\}$

Solution: We know from the lecture that $EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ is decidable. Let M be a Turing machine that decides this language. We construct a new TM M' that uses M to decide EQ_{DFA_RE} as follows:

M' = "On input $\langle D, R\rangle$, where D is a DFA and R is a regular expression:

1. Convert R to an equivalent DFA A by using the procedure for this conversion given in Theorem 1.28.
2. Run M on $\langle D, A\rangle$.
3. If M accepts, accept; if M rejects, reject."

Since M accepts iff $L(D) = L(A)$, and since $L(A) = L(R)$ this procedure is correct.

(b) $A_{\epsilon CFG} = \{\langle G\rangle \mid G \text{ is a CFG that generates } \epsilon\}$

Solution: We know from the lecture that $A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates input string } w\}$ is decidable. Let M be a Turing machine that decides this language. We can obviously construct a TM M' that decides $A_{\epsilon CFG}$ as follows:

M' = "On input $\langle G\rangle$, where G is a CFG:

1. Run M on $\langle G, \epsilon \rangle$.
2. If M accepts, accept; if M rejects, reject."

(c) $ALL_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA that recognizes } \Sigma^*\}$

Solution: A DFA A recognizes Σ^* iff all states that are reachable from the initial state are goal states. This can easily be checked by a Turing machine. Alternatively, we can use that EQ_{DFA} is decidable. Let M be a TM that decides this language. We can obviously construct a TM M' that decides ALL_{DFA} as follows:

M' = "On input $\langle A \rangle$, where A is a DFA:

1. Create a DFA B that consists only of the initial state q_0 which is a goal state. For each symbol of the alphabet there is a transition from q_0 to q_0.
2. Run M on $\langle A, B \rangle$.
3. If M accepts, accept; if M rejects, reject."

M' decides ALL_{DFA}: M accepts iff $L(A) = L(B)$, and by construction $L(B) = \Sigma^*$.

Exercise 7.3 (Undecidable Languages)

Consider the problem of determining whether a two-tape Turing machine ever writes a non-blank symbol on its second tape, i.e.

$$N = \{\langle M, w \rangle \mid M \text{ is a two-tape Turing machine which writes a non-blank symbol onto its second tape when it runs on } w\}.$$

Show that N is undecidable. *Hint*: Use a reduction from A_{TM}.
Solution: Proof by contradiction: assume N is decidable and let D be a decider for N:

$$D(\langle M, w \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ running on } w \text{ writes a non-blank on its second tape} \\ \textit{reject} & \text{if } M \text{ running on } w \text{ does not write a non-blank on its second tape} \end{cases}$$

Use D to define a TM H as follows:
On input $\langle M, w \rangle$ where M is an arbitrary one-tape Turing machine and $w \in \Sigma^*$,

(1) Create a TM M' that differs from M only in being a two-tape Turing machine that does not use the second tape except for one case: If M accepts the input, it writes a non-blank symbol on the second tape. This can be done by a simple change of the transition function: Whenever the transition function δ of M maps the accept state of M for a tape symbol to the empty set (meaning that there is no transition and M halts), the transition function δ' of M' writes instead the non-blank on the second tape.

(2) If $D(\langle M', w \rangle) = \textit{accept}$, *accept*; if $D(\langle M', w \rangle) = \textit{reject}$, *reject*.

This means

$$H(\langle M, w \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

So, H decides A_{TM} which is known to be undecidable. Hence, N cannot be decidable.

CSE 105 Homework 5

Due March 12

- You may (and are encouraged to) discuss the problems with other students, but what you hand in *must be your own work*.
- If you do not understand a problem or find a mistake, please email William at `wgmatthews@cs.ucsd.edu`.
- In addition to these problems, we recommend doing Sipser problems 4.18, 4.19, 4.28, 5.1, 5.9, 5.20, 5.22, 5.23, 5.33 for more practice. However, you should *not* hand in your solutions to these problems.
- **On any problems where you are constructing TMs, make sure it is very clear why the TM solves the problem. This can either be because it is obvious from the desciption of the TM, or from additional explaination.**
- Typed homework solutions will receive 5% extra credit.

1 Decidable Languages

Let A, B, C be recognizable languages over an alphabet Σ, such that $A \cup B \cup C = \Sigma^*$ and $A \cap B = \emptyset$, $A \cap C = \emptyset$, and $B \cap C = \emptyset$. Show that A is not only recognizable, but also decidable.

1.1 Solution

Let M_A, M_B, and M_C be TMs that recognize languages A, B, and C respectively. From the definition of A, B, and C, we know that every string will be accepted by exactly one of these machines. We will use this idea to construct a TM that decides A:

"On input w:

1. Simulate M_A on w, M_B on w, and M_C on w **in parallel**.
2. As soon as M_A accepts, accept.
3. As soon as M_B or M_C accept, reject."

The TM we constructed is guaranteed to always halt because one of the three machines (M_A, M_B, or M_C) is guaranteed to accept and halt. The TM given accepts iff $w \in A$, since it accepts when M_A accepts, and $w \in B$ or $w \in C$ implies $w \notin A$.

2 Another Decidable Language

Show that the following language is decidable.

$$L = \{\langle M \rangle \mid M \text{ is a DFA and there exists some string } w \text{ such that both } w \text{ and } w^{\mathcal{R}} \text{ are in } L(M).\}$$

2.1 Solution

We know that regular languages are closed under both intersection and reverse. A the description of a DFA $\langle M\rangle$ is in L iff $L(M) \cap L(M)^{\mathcal{R}} \neq \emptyset$ because the intersection is exactly the set of strings w such that both w and $w^{\mathcal{R}}$ are in $L(M)$. We can use these properties to construct a TM to decide L:

Let T be a TM that decides E_{DFA}.

"On input w:

1. If w is not of the form $\langle M\rangle$, reject. Otherwise extract M from the input.
2. Construct a DFA N such that $L(N) = L(M)^{\mathcal{R}}$.
3. Construct a DFA P such that $L(P) = L(M) \cap L(N)$.
4. Run T on $\langle P\rangle$.
5. Accept if T rejects, and reject if T accepts."

We know that this TM always halts because T always halts and we can do the DFA transformations using the algorithms from Chapter 1. This TM decides L because it accepts iff $L(M) \cap L(M)^{\mathcal{R}} \neq \emptyset$.

3 Undecidable Languages

Prove that each of the following languages is undecidable. (This problem will be worth twice the points of the others.)

a)

$$NEVERLOOP_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } M \text{ halts on all inputs.}\}$$

b)

$$\begin{aligned}VERIFY_{TM} = \{\langle M, w, r\rangle \mid\ & M \text{ is a TM, } r \in \{\mathtt{A}, \mathtt{R}, \mathtt{L}\}\text{, and}\\ & r = \mathtt{A} \text{ if } M \text{ accepts } w;\ r = \mathtt{R} \text{ if } M \text{ rejects } w;\text{ and } r = \mathtt{L} \text{ if } M \text{ loops on } w.\}\end{aligned}$$

c)

$$KSTRINGS_{TM} = \{\langle M, k\rangle \mid M \text{ is a TM and } M \text{ accepts exactly } k \text{ strings.}\}$$

3.1 Solution

For each of the subproblems, I will give a map reduction. For each of the subproblems, either a map reduction or a Turing reduction works to prove that the languages are undecidable. The map reductions I give may be turned into Turing reductions using the following template (where f is the mapping function and A is a TM that we are assuming decides the language that we're reducing to): "On input w

1. Run f on w to get x.
2. Run A on x.
3. If A accepts then accept. Otherwise reject."

a) We will show $HALT_{TM} \leq_m NEVERLOOP_{TM}$ by constructing the following mapping function f. Since $HALT_{TM}$ is undecidable then $NEVERLOOP_{TM}$ is undecidable.

   Let z by a string that does not describe a TM. Define f as "On input w:

   1. If w is not of the form $\langle M, x\rangle$, where M is the description of a TM, then output z. Otherwise, parse w to get M and x.

2. Construct the following TM N: "On input s:
   a. Run M on x."
3. Output $\langle N \rangle$."

To prove that this is a map reduction from $HALT_{TM}$ to $NEVERLOOP_{TM}$, we need to show that $w \in HALT_{TM}$ iff $f(w) \in NEVERLOOP_{TM}$ and that f always halts. The second part, that f always halts is straightforward since it only ever does simple transformations of the descriptions of TMs. If $w \in HALT_{TM}$, then $w = \langle M, x \rangle$ such that M halts on x. Thus, $f(w)$ will be $\langle N \rangle$ where N always halts, since N simply runs M on x. If $w \notin HALT_{TM}$ then either w is not of the form $\langle M, x \rangle$, in which case we output z which will be rejected by $NEVERLOOP_{TM}$ since it is not of the right form. If $w \notin HALT_{TM}$ and w is of the form $\langle M, x \rangle$, then it means that M must loop on w. In this case, when we output $\langle N \rangle$, it will always loop, and therefore not be in $NEVERLOOP_{TM}$.

b) We will show $A_{TM} \leq_m VERIFY_{TM}$ by constructing the following mapping function f. Since A_{TM} is undecidable then $VERIFY_{TM}$ is undecidable.

Let z by a string that does not describe a TM. Define f as "On input w:

1. If w is not of the form $\langle M, x \rangle$, where M is the description of a TM, then output $\langle z, w, \texttt{A} \rangle$. Otherwise, parse w to get M and x.
2. Output $\langle M, w, \texttt{A} \rangle$."

If w is not of the form $\langle M, w \rangle$, we output a string where the first part is not a valid TM, and therefore not in $VERIFY_{TM}$. If $w = \langle M, x \rangle$ and $w \in A_{TM}$ then M must accept when run on x. We output $\langle M, x, \texttt{A} \rangle$ which is in $VERIFY_{TM}$ since M accepts x. Finally, if $w \langle M, x \rangle$ and $w \notin A_{TM}$ then M must not accept x. We output $\langle M, x, \texttt{A} \rangle \notin VERIFY_{TM}$ since M does not accept x.

c) We will show $E_{TM} \leq_m KSTRINGS_{TM}$ by constructing the following mapping function f. Since E_{TM} is undecidable then $KSTRINGS_{TM}$ is undecidable.

Let z by a string that does not describe a TM. Define f as "On input w:

1. If w is not of the form $\langle M \rangle$, where M is the description of a TM, then output $\langle z, 0 \rangle$. Otherwise, parse w to get M.
2. Output $\langle M, 0 \rangle$."

If w is not of the form $\langle M \rangle$ then we output a string where the first part is not a valid TM and therefore not in $KSTRINGS_{TM}$. If $w = \langle M \rangle \in E_{TM}$ then $|L(M)| = 0$ so the output $\langle M, 0 \rangle \in KSTRINGS_{TM}$. If $w = \langle M \rangle \notin E_{TM}$ then $|L(M)| \neq 0$ so the output $\langle M, 0 \rangle \notin KSTRINGS_{TM}$.

4 Miscellanea

Prove or disprove each of the following statements.

a) Every recognizable language can be recognized by a TM that either accepts or loops, but never rejects.

b) If a language is decidable, then every proper subset of that language is decidable.

c) If $A \leq_m B$ and B is a regular language then A is a regular language.

4.1 Solution

a) This statement is true. Proof: For any recognizable language L, we have a TM M that recognizes it. We can construct a new TM M' that also recognizes L but never rejects:

"On input w:

1. Run M on w.
2. If M accepts then accept.
3. If M rejects then loop."

This TM recognizes the same language as M because it accepts iff M accepts. It never rejects because instead of rejecting it loops.

b) This statement is false. Counterexample: We know that A_{TM} is undecidable. Let Σ be the alphabet that A_{TM} is defined over. $A_{TM} \subset \Sigma^*$ and Σ^* is regular and therefore decidable.

c) This statement is false. Counterexample: Let $A = \{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$ and $B = \mathtt{0}^*\mathtt{1}^*$. B is regular because it is defined by a regular expression, and A is one of our canonical examples of a non-regular language. The following is a mapping function from A to B:

"On input w:

1. If w is in A then output `01`.
2. Otherwise output `10`."

This is a computable function because testing whether a string is in a context-free language is decidable. If $w \in A$ then $f(w) = \mathtt{01} \in B$ and if $w \notin A$ then $f(w) = \mathtt{10} \notin B$.

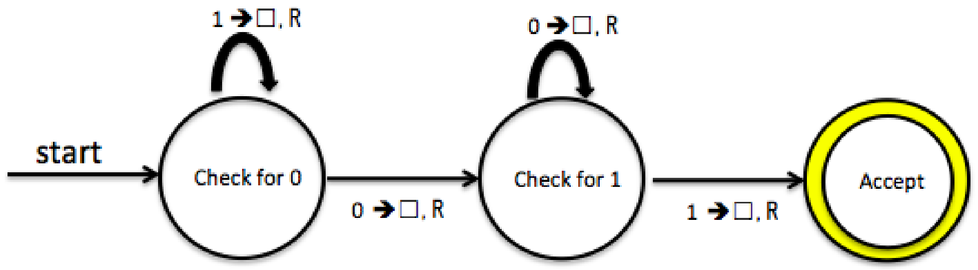
CS103 HW7: Solutions

Problem 1 (20 points)

Let $\Sigma = \{0, 1\}$. Draw the state transition diagram for a Turing machine whose language is $L = \{w \in \Sigma^* | w \text{ contains } 01 \text{ as a substring}\}$.

Solution

This Turing machine mimics the DFA for the same language, moving the tape head one step to the right at each step. In the following, assume that missing transitions implicitly cause the TM to reject.



Problem 2 (45 points)

(a) Prove that there exists a Turing machine M whose language L is decidable, but M is not a decider. This shows that just because a Turing machine's language is decidable, it's not necessarily the case that the Turing machine itself must be a decider.

(b) Prove that for every language L, there is a decider M that accepts every string in L and another decider M' that rejects every string not in L. Explain why this doesn't prove that every language is decidable.

(c) Find a pair of languages A and B such that A is a subset of B, B is decidable, but A is not decidable. This shows that a subset of a decidable language is not necessarily decidable, i.e. bigger languages are not necessarily 'harder.'

Solution

(a) Let M be the Turing machine that loops indefinitely on all inputs, which has language $L = \varnothing$. L is decidable (the Turing machine that rejects all inputs is a decider for L), but M is not a decider.

(b) We can set M as the TM that accepts all inputs and M' as the TM that rejects all inputs; then M accepts all strings in L and M' rejects all strings in L, and both are deciders. This doesn't show that L is decidable since the definition of decidability requires the same TM to accept all strings in L and reject all strings not in L, not two different TMs.

(c) Let A be any undecidable language (e.g. $HALT$), and B be Σ^*. Then B is decidable, and A is a subset of B.

Problem 3 (35 points)

Let $HALT$ be the language $\{\langle M, w\rangle : M \text{ is a TM that halts on } w\ \}$. Let $ALLHALT$ be the language $\{\langle M\rangle : M \text{ is a TM that halts on all inputs}\ \}$. Use a reduction from $HALT$ to show that $ALLHALT$ is not decidable.

Solution

We construct a mapping reduction f from $HALT$ to $ALLHALT$, showing that $HALT \leq_M ALLHALT$.

Let $f(\langle M, w\rangle)$ be the TM: "On input x: ignore x and run M on w." Now M halts on w iff $f(\langle M, w\rangle)$ halts on all inputs, so $\langle M, w\rangle \in HALT$ iff $f(\langle M, w\rangle) \in ALLHALT$, so f is a mapping reduction from $HALT$ to $ALLHALT$, showing that $HALT \leq_M ALLHALT$. Since $HALT$ is not decidable, we conclude that $ALLHALT$ is not decidable.

Prove that the following languages are undecidable.

1. AcceptIllini := $\{\langle M\rangle \mid M$ accepts the string ILLINI$\}$

   Solution: For the sake of argument, suppose there is an algorithm DecideAcceptIllini that correctly decides the language AcceptIllini. Then we can solve the halting problem as follows:

```
DecideHalt(⟨M, w⟩):
    Encode the following Turing machine M′:
        M′(x):
            run M on input w
            return True
    if DecideAcceptIllini(⟨M′⟩)
        return True
    else
        return False
```

   We prove this reduction correct as follows:

   $\Longrightarrow$ Suppose M halts on input w.
   Then M' accepts *every* input string x.
   In particular, M' accepts the string **ILLINI**.
   So DecideAcceptIllini accepts the encoding $\langle M'\rangle$.
   So DecideHalt correctly accepts the encoding $\langle M, w\rangle$.

   $\Longleftarrow$ Suppose M does not halt on input w.
   Then M' diverges on *every* input string x.
   In particular, M' does not accept the string **ILLINI**.
   So DecideAcceptIllini rejects the encoding $\langle M'\rangle$.
   So DecideHalt correctly rejects the encoding $\langle M, w\rangle$.

   In both cases, DecideHalt is correct. But that's impossible, because Halt is undecidable. We conclude that the algorithm DecideAcceptIllini does not exist. ■

   As usual for undecidablility proofs, this proof invokes *four* distinct Turing machines:

   - The hypothetical algorithm DecideAcceptIllini.
   - The new algorithm DecideHalt that we construct in the solution.
   - The arbitrary machine M whose encoding is part of the input to DecideHalt.
   - The special machine M' whose encoding DecideHalt constructs (from the encoding of M and w) and then passes to DecideAcceptIllini.

2. AcceptThree $:= \{\langle M\rangle \mid M \text{ accepts exactly three strings}\}$

Solution: For the sake of argument, suppose there is an algorithm DecideAcceptThree that correctly decides the language AcceptThree. Then we can solve the halting problem as follows:

```
DecideHalt($\langle M, w\rangle$):
    Encode the following Turing machine $M'$:
        $M'(x)$:
            run $M$ on input $w$
            if $x = \varepsilon$ or $x = 0$ or $x = 1$
                return True
            else
                return False
    if DecideAcceptThree($\langle M'\rangle$)
        return True
    else
        return False
```

We prove this reduction correct as follows:

$\Longrightarrow$ Suppose M halts on input w.
Then M' accepts exactly three strings: ε, 0, and 1.
So DecideAcceptThree accepts the encoding $\langle M'\rangle$.
So DecideHalt correctly accepts the encoding $\langle M, w\rangle$.

$\Longleftarrow$ Suppose M does not halt on input w.
Then M' diverges on *every* input string x.
In particular, M' does not accept exactly three strings (because $0 \neq 3$).
So DecideAcceptThree rejects the encoding $\langle M'\rangle$.
So DecideHalt correctly rejects the encoding $\langle M, w\rangle$.

In both cases, DecideHalt is correct. But that's impossible, because Halt is undecidable. We conclude that the algorithm DecideAcceptThree does not exist. ■

3. AcceptPalindrome := $\{\langle M\rangle \mid M \text{ accepts at least one palindrome}\}$

Solution: For the sake of argument, suppose there is an algorithm DecideAcceptPalindrome that correctly decides the language AcceptPalindrome. Then we can solve the halting problem as follows:

```
DecideHalt(⟨M, w⟩):
  Encode the following Turing machine M′:
      M′(x):
        run M on input w
        return True
      if DecideAcceptPalindrome(⟨M′⟩)
          return True
      else
          return False
```

We prove this reduction correct as follows:

$\Longrightarrow$ Suppose M halts on input w.
Then M' accepts *every* input string x.
In particular, M' accepts the palindrome **RACECAR**.
So DecideAcceptPalindrome accepts the encoding $\langle M'\rangle$.
So DecideHalt correctly accepts the encoding $\langle M, w\rangle$.

$\Longleftarrow$ Suppose M does not halt on input w.
Then M' diverges on *every* input string x.
In particular, M' does not accept any palindromes.
So DecideAcceptPalindrome rejects the encoding $\langle M'\rangle$.
So DecideHalt correctly rejects the encoding $\langle M, w\rangle$.

In both cases, DecideHalt is correct. But that's impossible, because Halt is undecidable. We conclude that the algorithm DecideAcceptPalindrome does not exist.

Yes, this is ***exactly*** the same proof as for problem 1. ■

Weekly exercises (with proposed solutions) on Undecidability

Exercise 1 Let $\overline{A}$ denote the *complement* of A, that is, $\overline{A} = \{w \mid w \notin A\}$.

Show that if A is a decidable language, then so is $\overline{A}$.

Solution proposal Since A is decidable, a decider M_A for A exists. Create $M_{\overline{A}}$:

"On input w:

(1) Simulate M_A on w.

(2) If M_A accepts, *reject.*

(3) If M_A rejects, *accept.*"

Exercise 2 For each language, show that it is undecidable by giving a reduction from $HALT$.

a) $L_1 = \{\langle M \rangle \mid \text{TM } M \text{ writes a \$ on the tape on every input}\}$

b) $L_2 = \{\langle M \rangle \mid \text{TM } M \text{ writes a \$ on the tape on input 010}\}$

c) $L_3 = \{\langle M \rangle \mid \text{There is no } y \text{ such that TM } M \text{ writes a \$ on the tape on input } y\}$

Solution proposal For L_1 and L_2 the same reduction as given in the lecture does the trick, since M' writes a \$ on every input iff M halts on w.

For L_3, we use the same reduction as in the lecture, but we flip the accept and reject in stage (3) of H. This way we reject if M_{L_3} accepts (since M did not halt on w), and vice versa.

Exercise 3 Let $L = \{\langle M \rangle \mid \text{TM } M \text{ writes a \$ on the tape, during the first 100 steps of computation, on every input}\}$.

Is L decidable? Prove your answer.

Solution proposal Here it looks like L might be undecidable, but it is actually decidable. We could simulate M for 100 steps, and reject if a \$ has not been written. We would need to simulate M on all different inputs, so it seems we would have to check infinitely many cases. The trick here, is that M only has 100 steps to write a \$, including the steps it takes to read the input. Thus, two inputs that are similar in the first 100 symbols can not make the TM behave differently for the first 100 steps. This means that we only have to simulate M for 100 steps on all inputs that are different in the first 100 symbols. This is a finite number, since the input alphabet is finite.

Exercise 4 Let $EQUAL = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$.

Is $EQUAL$ decidable? If yes, show a decider for $EQUAL$, if no, show a reduction from $HALT$.

Solution proposal The following reduction shows that $EQUAL$ is undecidable. Assume M_{EQUAL} is a decider for $EQUAL$. To decide an instance $\langle M, w\rangle$ of $HALT$, simulate M_{EQUAL} on $\langle M_1, M_2\rangle$, where M_1 is a TM that accepts all (well-formed) inputs, and where M_2 first simulates M on w, then accepts. This way, the machines will be equal iff M halts on w.

Exercise 5 Let $PI = \{k \mid$ There occurs k consecutive 0's in the decimal expansion of π $\}$.

Is PI a decidable language?

Solution proposal Yes, PI is decidable.

Case 1: there is a limit to the number of consecutive 0's in the decimal expansion of π, say n. That is, the longest sequence of 0's is n long, and no sequence of length $n+1$ occurs. In this case, a decider for PI checks its input, k, to compare with n and accepts if $k \leq n$, and rejects otherwise. Note that if the decimal expansion contains 17 0's in a row, then it also contains 16 in a row.

Case 2: no such upper limit exists, in which case the decider can accept right away.

In any case we have created a decider for PI, so the language is decidable.

Challenge 1 Recall from the lecture Rice's theorem.

Theorem 1 (Rice's theorem) *Let R be a language consisting of Turing machine descriptions, such that R contains some, but not all Turing machine descriptions. Furthermore, let the membership in R for any Turing machine M_1, depend solely on the language of M_1, that is: $L(M_1) = L(M_2) \implies (\langle M_1\rangle \in R \leftrightarrow \langle M_2\rangle \in R)$. Then R is an undecidable language.*

Give a proof of Rice's theorem.

Hint: Show a reduction from $HALT$ to R. Find a yes-instance and a no-instance of R, and create a TM that recognizes the one of those language depending on whether M halts on w on not. The result from Exercise 1, may come in handy.

Solution proposal We assume for contradiction that M_R decides R. We will show how to decide $HALT$ by using M_R as a subroutine in H. First, let $T_\emptyset$ be a TM that always rejects, that is, $L(T_\emptyset) = \emptyset$. We assume that $T_\emptyset \notin R$. If $T_\emptyset \in R$, we could continue the proof with $\overline{R}$ instead of R. (Here we use the result from Exercise 1). Since R contains some TM T, we have our yes-instance (T) and no-instance $(T_\emptyset)$ of R. Now, we create a TM called M', such that $L(M') = L(T_\emptyset) = \emptyset$ if M does not halt on w, and $L(M') = L(T)$ if M does halt on w:

M' = "On input x:

(1) Simulate M on w.

(2) If M halts: Simulate T on x.

(3) If T accepts, *accept*.

(4) If T rejects, *reject*.

Note that M' behaves like T if M halts on w and M' behaves like $T_\emptyset$ if M loops on w.

Finally, to decide $HALT$ we simulate M_R on $\langle M' \rangle$ and accept if M_R accepts and reject if M_R rejects.

CSE 6321 - Solutions to Problem Set 2

1. Let C be a language. Prove that C is Turing-recognizable iff a decidable language D exists such that $C = \{x | \exists y(\langle x, y\rangle \in D)\}$

Solution:

We need to prove both directions. To handle the easier one first, assume that the decidable language D exists. A TM recognizing C operates on input x by going through each possible string y and testing whether $\langle x, y\rangle \in D$. If such a y is ever found, accept; if not, just continue searching.

For the other direction, assume that C is recognized by TM, denoted by M. Define the language D to be $\{\langle x, y\rangle \mid M$ accepts x within $|y|$ steps$\}$. Language D is decidable since one can run M for y steps and accept iff M has accepted. If $x \in C$, then M accepts x within some number of steps, so $\langle x, y\rangle \in D$ for some sufficiently long y, but if $x \notin C$ then $\langle x, y\rangle \notin C$ for any y.

2. Let $T = \{\langle M\rangle | M$ is a TM that accepts w^R whenever it accepts $w\}$(w^R is the reverse of w). Show that T is undecidable. Is T recognizable?

Solution:

T is unrecognizable. We prove it by contradiction. Assume M_T is a recognizer of T, we can construct a recognizer of $\overline{A_{TM}}$.

M' = "On input $\langle M, w\rangle$:

1. Construct a TM M_w: "On input x:
   (a) If x=w^{r}, REJECT;
   (b) If x=w, run M on w and return the same result;
   (c) Else, REJECT."
2. Run M_T on $\langle M_w\rangle$, return the same."

Correctness:
M' accepts $\langle M, w\rangle \Leftrightarrow M_T$ accepts $\langle M_w\rangle \Leftrightarrow M$ does not accept $w \Leftrightarrow \langle M, w\rangle \in \overline{A_{TM}}$.
Therefore, T is unrecognizable.

3. Let

$$M = \{\langle a, b, c, p\rangle : \text{ a,b,c and p are binary integers, such that } a^b \equiv c \mod p\}$$

Show that $M \in P$.

Solution:

The most obvious algorithm multiplies a by itself b times then compares the result to c, modulo p. That uses $b - 1$ multiplications and so is exponential in the length of b. Hence this algorithm doesn't run in polynomial time. Here is one that does:

"On input $\langle a, b, c, p\rangle$, four binary integers:

- Let $r := 1$.
- Let $b_1, \ldots, b_k$ be the bits in the binary representation of b.
- For $i := 1, \ldots, k$:
  - If $b_i = 0$, let $r := r^2 \mod p$
  - If $b_i = 1$, let $r := ar^2 \mod p$
- If $(c \mod p) = (r \mod p)$, accept; otherwise reject."

The algorithm is called *repeated squaring.* Each of its multiplications and modular operations can be done in polynomial time in the standard way, because the r is never larger than p. The total number of multiplications is proportional to the number of bits in b. Hence the total running time is polynomial.

4. Prove that the following language is undecidable:

$$A = \{\langle M\rangle : M \text{ is a TM with running time } O(n)\}$$

Solution:
We give a mapping reduction from H_{TM} to it as follows.
Given an instance $\langle M, w\rangle$ of H_{TM}, construct the following TM M_0:
$M_0 =$ "On input x of length n:

- Ignore x
- Run M on w.
- If M accepts w, accept.
- If M rejects w, reject.

The reduction is correct: If M halts on w, then the running time of M_0 is a constant, and $O(n)$ in particular. If M loops on w, then M_0 loops on every input and is not in A.
ALTERNATIVE solution, mapping reduction from A_{TM}. $M_1 =$ "On input x of length n:

- Ignore x
- Run M on w.
- If M accepts w, accept.
- If M rejects w, loop.

$\langle M, w\rangle \in A_{TM} \Leftrightarrow M_1$ accepts $\Leftrightarrow M_1$ runs in constant time $\Rightarrow M_1 \in A$.

5. Prove that the following language is undecidable:

$$A = \{\langle M\rangle \mid L(M) \in \text{TIME}(n)\}.$$

Solution:
$L_1 = \{1\}^* \in \text{TIME}(n) \implies A \neq \emptyset$
A_{TM} is not decidable. Let R be its recognizer, clearly its running time is infinite $\implies \langle R\rangle \notin A \implies A \neq \{\langle M\rangle\}$. Hence the property is non-trivial. It is clearly a property of the language of M. By Rice's theorem, we know that A is undecidable.

CS 121 Section 8

Harvard University

Fall 2013

Overview

This week we will focus on reviewing the core concepts involved with undecidability, reducibility, Rice's theorem, incompleteness of mathematics, and so on.

1 Concept Review

1.1 Undecidability

By a cardinality argument, we know that almost all languages are undecidable. This argument, however, does not give us an explicit construction. The following theorem does just that.

Theorem 1.1. *The language $\{\langle M, w\rangle : M$ accepts the input $w\}$ is not decidable.*

Proof. Assume $\{\langle M, w\rangle : M$ accepts the input $w\}$ is decidable, then the language $D = \{\langle M\rangle : M$ accepts $\langle M\rangle\}$ is decidable, hence $\overline{D} = \{\langle M\rangle : M$ does not accepts $\langle M\rangle\}$ is decidable. Suppose $\overline{D}$ is decidable by M_1, then $\langle M_1\rangle \in \overline{D}$ iff M_1 accepts $\langle M_1\rangle$ iff $\langle M_1\rangle \in D$, which is a contradiction. (This is the standard diagonalization argument.) $\square$

1.2 Reducibility

Definition 1.1. *A function $f : \Sigma_1^* \to \Sigma_2^*$ is computable if there is a Turing machine such that for every input $w \in \Sigma_1^*$, M halts with just $f(w)$ on its tape.*

Definition 1.2. *A reduction of $L_1 \subseteq \Sigma_1^*$ to $L_2 \subseteq \Sigma_2^*$ is a computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that, for any $w \in \Sigma^*$, $w \in L_1$ if and only if $f(w) \in L_2$, and we write $L_1 \leq_m L_2$.*

Intuitively, L_1 reduces to L_2 means that L_1 is not harder than L_2. More formally, we can express this intution in the following lemma.

Lemma 1.1. *If $L_1 \leq_m L_2$ and L_1 is undecidable, then so it L_2.*

1.3 Rice's theorem

Theorem 1.2 (Rice's theorem)**.** *Let $\mathcal{P}$ be any subset of the class of r.e. languages such that $\mathcal{P}$ and its complement are both nonempty. Then the language $L_{\mathcal{P}} = \{\langle M\rangle : L(M) \in \mathcal{P}\}$ is undecidable.*

Intuitively, Rice's theorem states that Turing machines can not test whether another Turing machine satisfies a (nontrivial) property. For example, let $\mathcal{P}$ be the subset of the recursively enumerable languages which contains the string a. Then Rice's theorem claims that there is no Turing machine which can decide whether a Turing machine accepts a.

2 Exercises

Exercise 2.1. *Reductions can be tricky to get the hang of, and you want to avoid "going the wrong way" with them. In which of these scenarios does $L_1 \leq_m L_2$ provide useful information (and in those cases, what may we conclude)?*

(a) L_1's decidability is unknown and L_2 is undecidable

Nothing

(b) L_1's decidability is unknown and L_2 is decidable

L_1 is decidable. This is in Sipser.

(c) L_1 is undecidable and L_2's decidability is unknown

Undecidable. Corollary to the above question.

(d) L_1 is decidable and L_2's decidability is unknown

Nothing

Exercise 2.2. *Argue that $\leq_m$ is a transitive relation.*

Let f be the function that reduces A to B, i.e., $A \leq_m B$ by f, and let g be the function that reduces B to C, i.e., $B \leq_m C$ by g. Then $w \in A \iff f(w) \in B$. Furthermore, $x \in B \iff g(x) \in C$. It follows that for every $w \in A$, $g(f(w)) \in C$, and furthermore, for every $x \in C$, there exists a $y \in B$ such that $g(y) = x$, and for every $y \in B$, there exists a $w \in A$ such that $f(w) = x$, so that for every $x \in C$, there exists a $w \in A$ with $g(f(w)) = x \in C$. Thus $w \in A$ iff $g(f(w)) \in C$, so that $A \leq_m C$.

Exercise 2.3. *Determine, with proof, whether the following languages are decidable.*

(a) $L = \{\langle M, x\rangle :$ *At some point it its computation on* x, M *re-enters its start state*$\}$

Undecidable. Assume for the sake of contradiction that this language is decidable. We will show that a decider for this language can decide the halting problem. Given an input $\langle M, w\rangle$, *modify* M *so that the start state is split into two states, and the new start state has no incoming transitions except for those we will specifically add in this construction. Also, change the accept and reject states so that they are no longer accept / reject states, but just normal states that (effectively) epsilon transition to the new start state. Make new accept / reject states that are inaccessible. Now the machine re-enters its start state iff the original machine would have halted. This construction was a finite transformation, and something a TM could do, so by combining this machine with a decider for the language in question, we could decide the halting problem. Contradiction.*

(b) $L = \{\langle x, y\rangle : f(x) = y\}$ *where* f *is a fixed computable function.*

Decidable. Given $\langle x, y\rangle$, *compute* $f(x)$ *and compare it to* y.

(c) $\mathrm{CF}_{TM} = \{\langle M\rangle : L(M)$ *is context-free*$\}$

Undecidable. The subset of languages which are context-free and the subset of languages which are not are both non-empty. Apply Rice's theorem.

(d) $L = \{\langle M\rangle : M$ *calculates* $\pi\}$, *meaning that* $M : \mathbb{N} \to \mathbb{N}$, *and* $M(i)$ *is equal to the i-th digit in the decimal expansion of* π.

Undecidable. Assume $\{\langle M\rangle : M$ *calculates* $\pi\}$ *is decidable, then the language* $D = \{\langle M\rangle : M$ *accepts* $\langle M\rangle\}$ *is decidable, hence* $\overline{D} = \{\langle M\rangle : M$ *does not accepts* $\langle M\rangle\}$ *is decidable. Suppose* $\overline{D}$ *is decidable by* M_1, *then* $\langle M_1\rangle \in \overline{D}$ *iff* M_1 *accepts* $\langle M_1\rangle$ *iff* $\langle M_1\rangle \in D$, *which is a contradiction. (This is the standard diagonalization argument.)*

Exercise 2.4. *Show* $\{G : G$ *is a CFG generating* $x\} \leq_M \{G : G$ *is a CFG generating* $xy\}$.

Given a grammar G, *define* $f(G)$ *to be the grammar which is the same, except that it has a new start variable* S' *and a new rule* $S' \to Sy$ *(where* S *was the old start variable). If* $x \in L(G)$, *then* $x \in L(f(G))$, *since the same derivation that leads to* x *from* S *can be used to derive* xy *from* Sy. *If* $xy \in L(f(G))$, *then since* Sy *must have been a working string after the first step of the derivation,* x *must have been generated from* S, *which means* $x \in L(G)$.