Lecture 18: More reductions; Rice's Theorem

30 March 2010

This lecture covers more reductions for undecidability as well as Rice's theorem.

1 Equality

An easy corollary of the undecidability of E_{TM} is the undecidability of the language

$$EQ_{\mathsf{TM}} = \left\{ \langle M, N \rangle \;\middle|\; M \text{ and } N \text{ are } \mathsf{TM}\text{'s and } L(M) = L(N) \right\}.$$

Lemma 1.1 *The language EQ_{TM} is undecidable.*

Proof: Suppose that we had a decider **DeciderEqual** for EQ_{TM}. Then we can build a decider for E_{TM} as follows:

TM R:

1. Input $= \langle M \rangle$
2. Include the (constant) code for a TM T that rejects all its input. We denote the string encoding T by $\langle T \rangle$.
3. Run **DeciderEqual** on $\langle M, T \rangle$.
4. If **DeciderEqual** accepts, then accept.
5. If **DeciderEqual** rejects, then reject.

■

Since the decider for E_{TM} (i.e., $\mathsf{TM}_{E_{\mathsf{TM}}}$) takes one input but the decider for EQ_{TM} (i.e. **DeciderEqual**) requires two inputs, we are tying one of **DeciderEqual**'s input to a constant value (i.e., T).

There are many Turing machines that reject all their input and could be used as T. Building code for R just requires writing code for one such TM.

2 Regularity

It turns out that almost any property defining a TM language induces a language which is undecidable, and the proofs all have the same basic pattern. Let us do a slightly more complex example and study the outline in more detail.

Let

$$\text{Regular}_{\mathsf{TM}} = \left\{ \langle M \rangle \;\middle|\; M \text{ is a } \mathsf{TM} \text{ and } L(M) \text{ is regular} \right\}.$$

Suppose that we have a TM **DeciderRegL** that decides $\text{Regular}_{\mathsf{TM}}$. In this case, doing the reduction from halting, would require to turn a problem about deciding whether a TM M accepts w (i.e., is $w \in \text{A}_{\mathsf{TM}}$) into a problem about whether some TM accepts a regular set of strings.

Given M and w, consider the following TM M'_w:

TM M'_w:

(i) Input $= x$

(ii) If x has the form $\mathtt{a}^n\mathtt{b}^n$, halt and accept.

(iii) Otherwise, simulate M on w.

(iv) If the simulation accepts, then accept.

(v) If the simulation rejects, then reject.

Again, we are ***not*** going to execute M'_w directly ourself. Rather, we will feed its description $\langle M'_w \rangle$ (which is just a string) into **DeciderRegL**. Let **EmbedRegularString**denote this algorithm, which accepts as input $\langle M \rangle$ and w, and outputs $\langle M'_w \rangle$, which is the encoding of the machine M'_w.

If M accepts w, then every input x will eventually be accepted by the machine M'_w. Some are accepted right away and some are accepted in step (i). So if M accepts w then the language of M'_w is Σ^*.

If M does not accept w, then some strings x (that are of the form $\mathtt{a}^n\mathtt{b}^n$) will be accepted in step (ii) of M'_w. However, after that, either step (iii) will never halt or step (iv) will reject. So the rest of the strings (that are in the set $\Sigma^* \setminus \left\{ \mathtt{a}^n\mathtt{b}^n \;\middle|\; n \geq 0 \right\}$) will not be accepted. So the language of M'_w is $\mathtt{a}^n\mathtt{b}^n$ in this case.

Since $\mathtt{a}^n\mathtt{b}^n$ is not regular, we can use our decider **DeciderRegL** on M'_w to distinguish these two cases.

Notice that the test in step (ii) was cooked up specifically to match the capabilities of our given decider **DeciderRegL**. If **DeciderRegL** had been testing whether our language contained the string "uiuc", step (ii) would be comparing x to see if it was equal to "uiuc". This test can be anything that a TM can compute without the danger of going into an infinite loop.

Specifically, we can build a decider for A_{TM} as follows.

```
YetAnotherDecider-A_TM(⟨M, w⟩)
    ⟨M'_w⟩ ← EmbedRegularString(⟨M, w⟩)
    r ← DeciderRegL(⟨M'_w⟩).
    return r
```

The reason why **YetAnotherDecider-**A_{TM} does the right thing is that:

— If **DeciderRegL** accepts, then $L(M'_w)$ is regular. So it must be Σ^*. This implies that M accepts w. So **YetAnotherDecider-A_{TM}** should accept $\langle M, w\rangle$.

— If **DeciderRegL** rejects, then $L(M'_w)$ is not regular. So it must be $\mathtt{a}^n\mathtt{b}^n$. This implies that M does not accept w. So **YetAnotherDecider-A_{TM}** should reject $\langle M, w\rangle$.

3 Rice's Theorem

3.1 Another Example - The language L_3

Let us consider another reduction with a very similar outline. Suppose we have the following language

$$L_3 = \left\{ \langle M\rangle \;\middle|\; |L(M)| = 3 \right\}.$$

That is L_3 contains all Turing machines whose languages contain exactly three strings.

Lemma 3.1 *The language L_3 is undecidable.*

Proof: Proof by reduction from A_{TM}. Assume, for the sake of contradiction, that L_3 was decidable and let $\mathtt{decider}_{L_3}$ be a TM deciding it. We use $\mathtt{decider}_{L_3}$ to construct a Turing machine $\mathtt{decider}_9$-A_{TM} deciding A_{TM}. The decider TM$\mathtt{decider}_9$-A_{TM} is constructed as follows:

$\mathtt{decider}_9$-A_{TM} ($\langle M, w\rangle$)

Construct a new Turing machine M_w:

M_w(x): // x: input

$res \leftarrow$ Run M on w

if ($res =$ **reject**) **then**

reject

if $x =$ `UIUC` or $x =$ `Iowa` or $x =$ `Michigan` **then**

accept

reject

return $\mathtt{decider}_{L_3}$ ($\langle M_w\rangle$).

(We emphasize here, again, that constructing M_w involve taking the encoding of $\langle M\rangle$ and w, and generating the encoding of $\langle M_w\rangle$.)

Notice that the language of M_w has only two possible values. If M loops or rejects w, then $L(M_w) = \emptyset$. If M accepts w, then th the language of M_w contains exactly three strings: "`UIUC`", "`Iowa`", and "`Michigan`".

So $\mathtt{decider}_9$-A_{TM}($\langle M_w\rangle$) accepts exactly when M accepts w. Thus, $\mathtt{decider}_9$-A_{TM} is a decider for A_{TM} But we know that A_{TM} is undecidable. A contradiction. As such, our assumption that L_3 is decidable is false. ■

3.2 Rice's theorem

Notice that these two reductions have very similar outlines. Our hypothetical decider **decider** looks for some property P. The auxiliary TM's tests x for membership in an example set with property P. The big difference is whether we simulate M on w before or after testing x and, consequently, whether the second possibility for $L(M_w)$ is $\emptyset$ or Σ^*.

It's easy to cook up many examples of reductions similar to this one, all involving sets of TM's whose *languages* share some property (e.g. they are regular, they have size three). Rice's Theorem generalizes all these reductions into a common result.

Theorem 3.2 (Rice's Theorem.) *Suppose that* L *is a language of Turing machines; that is, each word in* L *encodes a TM. Furthermore, assume that the following two properties hold.*

(a) Membership in L *depends only on the Turing machine's language, i.e. if* $L(M) = L(N)$ *then* $\langle M \rangle \in \mathsf{L} \Leftrightarrow \langle N \rangle \in \mathsf{L}$.

(b) The set L *is "non-trivial," i.e.* $\mathsf{L} \neq \emptyset$ *and* L *does not contain all Turing machines.*

Then L *is a undecidable.*

Proof: Assume, for the sake of contradiction, that L is decided by TM**deciderForL**. We will construct a TM$\texttt{Decider}_4\text{-}\mathrm{A}_{\mathsf{TM}}$ that decides A_{TM}. Since $\texttt{Decider}_4\text{-}\mathrm{A}_{\mathsf{TM}}$ does not exist, we will have a contradiction, implying that **deciderForL** does not exist.

Remember from last class that $\mathsf{TM}_\emptyset$ is a TM (pick your favorite) which rejects all input strings. Assume, for the time being, that $\mathsf{TM}_\emptyset \notin \mathsf{L}$. This assumption will be removed shortly.

Since L is non-trivial, also choose some other TM $Z \in \mathsf{L}$. Now, given $\langle M, w \rangle$ $\texttt{Decider}_4\text{-}\mathrm{A}_{\mathsf{TM}}$ will construct the encoding of the following TM M_w.

TM M_w:

(1) Input $= x$.

(2) Simulate M on w.

(3) If the simulation rejects, halt and reject.

(4) If the simulation accepts, simulate Z on x and accept if and only if T halts and accepts.

If M loops or rejects w, then M_w will get stuck on line (2) or stop at line (3). So $L(M_w)$ is $\emptyset$. Because membership in L depends only on a Turing machine's language and $\langle \mathsf{TM}_\emptyset \rangle$ is not in L, this means that M_w is not in L. So M_w will be rejected by N.

If M accepts w, then M_w will proceed to line (4), where it simulates the behavior of Z. So $L(M_w)$ will be $L(Z)$. Because membership in L depends only on a Turing machine's language and T is L, this means that M_w is in L. So M_w will be accepted by N.

As usual, our decider for A_{TM} looks like:

```
Decider_4-A_TM (⟨M, w⟩)
        Construct ⟨M_w⟩ from ⟨M, w⟩
        return deciderForL (⟨M_w⟩)
```

So $\texttt{Decider}_4\text{-}\mathrm{A}_{\mathsf{TM}}$ ($\langle M, w\rangle$) will accept $\langle M, w\rangle$ iff **deciderForL** accepts M_w. But we saw above that **deciderForL** accepts M_w iff M accepts w. So $\texttt{Decider}_4\text{-}\mathrm{A}_{\mathsf{TM}}$ is a decider for A_{TM}. Since such a decider cannot exist, we must have been wrong in our assumption that there was a decider for L.

Now, let us remove the assumption that $\mathsf{TM}_\emptyset \notin \mathsf{L}$. The above proof showed that L is undecidable, assuming that $\langle \mathsf{TM}_\emptyset \rangle$ was not in L. If $\mathsf{TM}_\emptyset \in \mathsf{L}$, then we run the above proof using $\overline{\mathsf{L}}$ in place of L. At the end, we note that $\overline{\mathsf{L}}$ is decidable iff L is decidable. ■

A More examples

The following examples weren't presented in lecture, but may be helpful to students.

A.1 The language L_{UIUC}

Here's another example of a reduction that fits the Rice's Theorem outline.

Let

$$L_{\mathrm{UIUC}} = \left\{ \langle M \rangle \;\middle|\; L(M) \text{ contains the string "UIUC"} \right\}.$$

Lemma A.1 *L_{UIUC} is undecidable.*

Proof: Proof by reduction from A_{TM}. Suppose that L_{UIUC} were decidable and let R be a Turing machine deciding it. We use R to construct a Turing machine deciding A_{TM}. S is constructed as follows:

- Input is $\langle M, w\rangle$, where M is the code for a Turing Machine and w is a string.
- Construct code for a new Turing machine M_w as follows:
  - Input is a string x.
  - Erase the input x and replace it with the constant string w.
  - Simulate M on w.
- Feed $\langle M_w \rangle$ to R. If R accepts, accept. If R rejects, reject.

If M accepts w, the language of M_w contains all strings and, thus, the string "UIUC". If M does not accept w, the language of M_w is the empty set and, thus, does not contain the string "UIUC". So R($\langle M_w \rangle$) accepts exactly when M accepts w. Thus, S decides A_{TM}

But we know that A_{TM} is undecidable. So S does not exist. Therefore we have a contradiction. So L_{UIUC} must have been undecidable. ■

A.2 The language Halt_Empty_TM

Here's another example which isn't technically an instance of Rice's Theorem, but has a very similar structure.

Let

$$\mathsf{Halt\_Empty\_TM} = \left\{ \langle M \rangle \;\middle|\; M \text{ halts on blank input} \right\}.$$

Lemma A.2 Halt_Empty_*TM is undecidable.*

Proof: By reduction from A_{TM}. Suppose that Halt_Empty_TM were decidable and let R be a Turing machine deciding it. We use R to construct a Turing machine deciding A_{TM}. S is constructed as follows:

- Input is $\langle M, w \rangle$, where M is the code for a Turing Machine and w is a string.
- Construct code for a new Turing machine M_w as follows:
  - Input is a string x.
  - Ignore the value of x.
  - Simulate M on w.
- Feed $\langle M_w \rangle$ to R. If R accepts, then accept. If R rejects, then reject.

If M accepts w, the language of M_w contains all strings and, thus, in particular the empty string. If M does not accept w, the language of M_w is the empty set and, thus, does not contain the empty string. So $R\big(\langle M_w \rangle\big)$ accepts exactly when M accepts w. Thus, S decides A_{TM}

But we know that A_{TM} is undecidable. So S can not exist. Therefore we have a contradiction. So Halt_Empty_TM must have been undecidable. ■

A.3 The language L_{111}

Here is another example of an undecidable language defined by a Turing machine's behavior, to which Rice's Theorem does not apply.

Let

$$L_{111} = \left\{ \langle M \rangle \;\middle|\; M \text{ prints three one's in a row on blank input} \right\}.$$

Lemma A.3 *The language L_{111} is undecidable.*

Proof: Suppose that L_{111} were decidable. Let R be a Turing machine deciding L_{111}. We will now construct a Turing machine S that decides A_{TM}.

The decider S for A_{TM} is constructed as follows:

- Input is $\langle M, w \rangle$, where M is the code for a Turing Machine and w is a string.
- Construct the code for a new Turing machine M', which is just like M except that
  - every use of the character `1` is replaced by a new character `1'` which M does not use.

  - when M would accept, M' first prints `111` and then accepts

- Similarly, create a string w' in which every character `1` has been replaced by $\mathtt{1}'$.
- Create a second new Turing machine M'_w which simulates M' on the hard-coded string w'.
- Run R on $\langle M'_w \rangle$. If R accepts, accept. If R rejects, then reject.

If M accepts w, then M'_w will print `111` on any input (and thus on a blank input). If M does not accept w, then M'_w is guaranteed never to print `111` accidently. So R will accept $\langle M'_w \rangle$ exactly when M accepts w. Therefore, S decides A_{TM}.

But we know that A_{TM} is undecidable. So S can not exist. Therefore we have a contradiction. So $L_{\mathtt{111}}$ must have been undecidable. ■