MA/CSSE 474
Theory of Computation

More Reduction Proofs

Recap: Reducing Decision Problem P_1 to another Decision Problem P_2

We say that P1 is **reducible** to P_2 (written $P_1 \leq P_2$) if

- there is a Turing-computable function f that finds, for an arbitrary instance I of P_1, an instance f(I) of P_2, and
- f is defined such that for every instance I of P_1,
  I is a yes-instance of P_1 if and only if
  f(I) is a yes-instance of P_2.

So $P_1 \leq P_2$ means "if we have a TM that decides P_2, then there is a TM that decides P_1.

Reducing *Language* L_1 to L_2

- L_1 (over alphabet Σ_1) is **reducible** to L_2 (over alphabet Σ_2) and we write $L_1 \leq L_2$ if

  there is a Turing-computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that

  $\forall x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$

Using reducibility

- If P_1 is reducible to P_2, then
  - If P_2 is decidable, so is P_1.
  - If P_1 is not decidable, neither is P_2.
- The second part is the one that we will use most.

Recap: H_ε = {<*M*> : TM *M* halts on ε}

Theorem: H_ε = {<*M*> : TM *M* halts on ε} is not in D.

Proof: by reduction from H:

H = {<*M*, *w*> : TM *M* halts on input string *w*}

↓ *R*

(?*Oracle*) H_ε {<*M*> : TM *M* halts on ε}

R is a mapping reduction from H to H_ε:

R(<*M*, *w*>) =

1. Construct <*M#*>, where *M#*(*x*) operates as follows:
   1.1. Erase the tape.
   1.2. Write *w* on the tape and move the head to the left end.
   1.3. Run *M* on *w*.
2. Return <*M#*>.

Recap: Proof, Continued

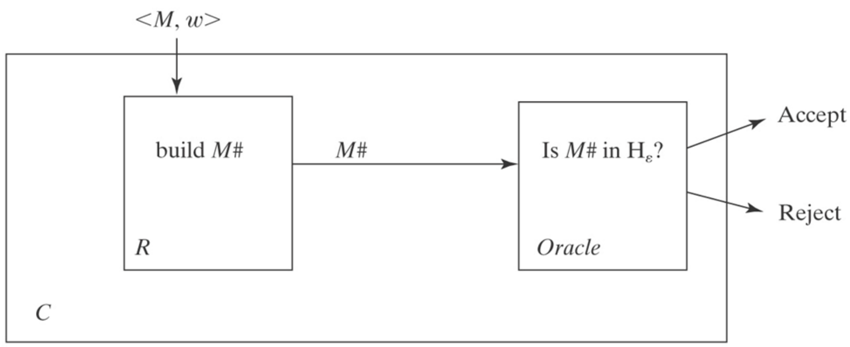
R(<*M*, *w*>) =

1. Construct <*M#*>, where *M#*(*x*) operates as follows:
   1.1. Erase the tape.
   1.2. Write *w* on the tape and move the head to the left end.
   1.3. Run *M* on *w*.
2. Return <*M#*>.

If *Oracle* exists, *C* = *Oracle*(*R*(<*M*, *w*>)) decides H:

- *C* is correct: *M#* ignores its own input. It halts on everything or nothing. So:
  - <*M*, *w*> ∈ *H*: *M* halts on *w*, so *M#* halts on everything. In particular, it halts on ε. *Oracle* accepts.
  - <*M*, *w*> ∉ *H*: *M* does not halt on *w*, so *M#* halts on nothing and thus not on ε. *Oracle* rejects.

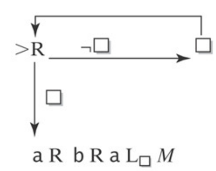
Recap: A Block Diagram of *C*



R Can Be Implemented as a Turing Machine

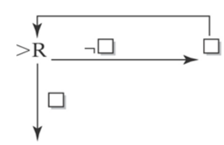
R must construct <*M#*> from <*M*, *w*>. Suppose *w* = `aba`.

M# will be:



So the procedure for constructing *M#* is:

1. Write:



2. For each character *x* in *w* do:
   2.1. Write *x*.
   2.2. If *x* is not the last character in *w*, write R.
3. Write L$_\square$ *M*.

Conclusion

R can be implemented as a Turing machine.

C is correct.

So, if *Oracle* exists:

C = *Oracle*(*R*(<*M*, *w*>)) decides H.

But no machine to decide H can exist.

So neither does *Oracle*.

This Result is Somewhat Surprising

If we could decide whether *M* halts on the specific string ε, we could solve the more general problem of deciding whether *M* halts on an arbitrary input.

Clearly, the other way around is true: If we could solve H we could decide whether *M* halts on any one particular string.

But we used reduction to show that H undecidable implies H_ε undecidable; this is not at all obvious.

Different Languages We Are Dealing With

H = {<*M*, *w*> : TM *M* halts on input string *w*}

↓ *R*

(?*Oracle*) H_ε {<*M*> : TM *M* halts on ε}

H contains strings of the form:
(*q*00,*a*00,*q*01,*a*10,←),(*q*00,*a*00,*q*01,*a*10,→),…,aaa

H_ε contains strings of the form:
(*q*00,*a*00,*q*01,*a*10,←),(*q*00,*a*00,*q*01,*a*10,→),…

The language on which some *M* halts contains strings of some arbitrary form, for example,

(letting Σ = {a, b}): aaaba

How Many Machines Are We Dealing With?

H = {<*M*, *w*> : TM *M* halts on input string *w*}

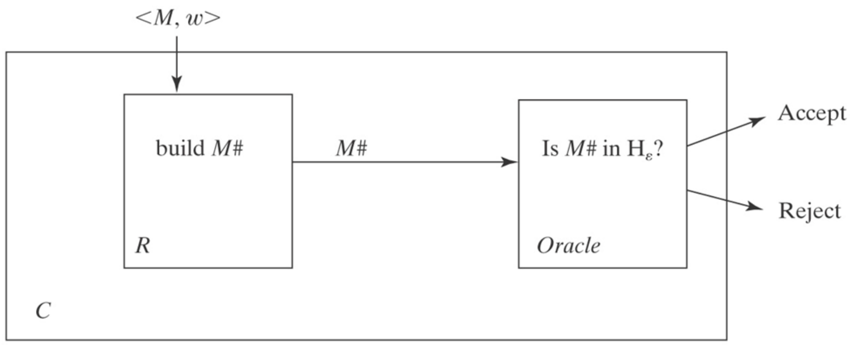
↓ *R*

(?*Oracle*) H_ε {<*M*> : TM *M* halts on ε}

R is a reduction from H to Hε:
R(<*M*, *w*>) =
1. Construct <*M#*>, where *M#*(x) operates as follows:
    1.1. Erase the tape.
    1.2. Write *w* on the tape.
    1.3. Run *M* on *w*.
2. Return <*M#*>.

- *Oracle* (the hypothesized machine to decide H_ε).
- *R* (the machine that builds *M#*. Actually exists).
- *C* (the composition of *R* with *Oracle*).
- *M#* (the machine we will pass as input to *Oracle*). Note that we never run it.
- *M* (the machine whose membership in H we are interested in determining; thus also an input to *R*).

A Block Diagram of *C*



Another Way to View the Reduction

```
// let L = {<M> | M is a TM that halts when its input is epsilon}
// if L is decidable, let the following function decide L:

boolean haltsOnEpsilon(TM M);   // defined in magic.h

//    HaltsOn decides H using HaltsOnEpsilon
// .: HaltsOn reduces to HaltsOnEpsilon:

bool haltsOn(TM M, string w)    {
      void wrapper(string iDontCare) {// a nested TM
            M(w);
       } {// end of nested TM
       return haltsOnEpsilon(wrapper);
  }
```

If HaltsOnEpsilon is a decision procedure, so is HaltsOn.
But of course HaltsOn is not, so neither is HaltsOnEpslipn

Important Elements in using a Reduction Proof to show a language is not in D

- A clear declaration of the reduction “from” and “to” languages. The "from" language should be a known undecidable language.
- A clear description of the reduction function *R*.
- If *R* does anything nontrivial, explain how it can be implemented as a TM.
- Note that machine diagrams are not necessary or even sufficient in these proofs. Use them as thought devices, where needed.
- Explain the logic that demonstrates how the “from” language is being decided by the composition of *R* and *Oracle*. You must do both accepting and rejecting cases.
- Declare that the reduction proves that your “to” language is not in D.

The Most Common Mistake: Doing the Reduction Backwards

The right way to use reduction to show that L_2 is not in D:

1. Given that L_1 is not in D,
2. Reduce L_1 to L_2, i.e., show how to solve L_1 (the known one) in terms of L_2 (the unknown one)

$$L_1 \downarrow L_2$$

Doing it wrong by reducing L_2 (the unknown one) to L_1:

If there exists a machine M_1 that solves L_1, then we could build a machine that solves L_2 as follows:

1. Return (M_1(<M, ε>)).

This proves nothing. It’s an argument of the form:

If *False* then …

H_{ANY} = {<*M*> : there exists at least one string on which TM *M* halts}

Theorem: H_{ANY} is in SD.

Proof: by exhibiting a TM *T* that semidecides it.

What about simply trying all the strings in Σ* one at a time until one halts?

H_{ANY} is in SD

T(<*M*>) =

1. Use **dovetailing*** to try *M* on all of the elements of Σ*:

ε [1]
ε [2] a [1]
ε [3] a [2] b [1]
ε [4] a [3] b [2] aa [1]
ε [5] a [4] b [3] aa [2] ab [1]

2. If any instance of *M* halts, halt and accept.

T will accept iff *M* halts on at least one string. So *T* semidecides H_{ANY}.

* http://en.wikipedia.org/wiki/Dovetailing_(computer_science)

H_{ANY} is not in D

The Steps in a Reduction Proof

1. ✪ Choose an undecidable language to reduce from.

2. ✪ Define the reduction *R*.

3. Show that *C* (the composition of *R* with *Oracle*) is correct.

✪ indicates where we make choices.

Undecidable Problems (Languages That Aren't In D)

The Problem View	**The Language View**
Does TM M halt on w?	$H = \{<M, w> : M \text{ halts on } w\}$
Does TM M not halt on w?	$\neg H = \{<M, w> : M \text{ does not halt on } w\}$
Does TM M halt on the empty tape?	$H_{\varepsilon} = \{<M> : M \text{ halts on } \varepsilon\}$
Is there any string on which TM M halts?	$H_{ANY} = \{<M> : \text{there exists at least one string on which TM } M \text{ halts}\}$
Does TM M accept all strings?	$A_{ALL} = \{<M> : L(M) = \Sigma^*\}$
Do TMs M_a and M_b accept the same languages?	$EqTMs = \{<M_a, M_b> : L(M_a) = L(M_b)\}$
Is the language that TM M accepts regular?	$TMreg = \{<M> : L(M) \text{ is regular}\}$

Tomorrow: We will prove some of these (most are also done in the book)