MA/CSSE 474
Theory of Computation

More Reduction Examples
Non-SD Reductions

Your Questions?

- Previous class days' material
- Reading Assignments
- HW 15 problems
- Final Exam
- Anything else

Excerpt from an obituary in the Terre Haute *Tribune Star*. May 12, 2018:

Lane was a kind and generous person who never hesitated to offer help to those who asked for it. He had a brilliant mind and keen imagination. He will be missed by his loving family and friends, as well as, the leeches who fed upon his generosity and mischievous spirit; they will have to find a new host now.

Reducing *Language* L_1 to L_2

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

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

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

Using Reduction for Undecidability

(R is a reduction from L_1 to L_2) $\wedge$ (L_2 is in D) $\rightarrow$ (L_1 is in D)

If (L_1 is in D) is false, then at least one of the two antecedents of that implication must be false. So:

If (R is a reduction from L_1 to L_2) is true
and (L1 is in D) is false,
then (L_2 is in D) must be false.

Application: If L1 is a language that is known to not be in D, and we can find a reduction from L1 to L2, then L2 is also not in D.

Using Reduction for Undecidability

Showing that L_2 is not in D:

L_1	(known not to be in D)	L_1 in D	But L_1 not in D
	R ↓	↑	↓
L_2	(a new language whose decidability we are trying to determine)	if L_2 in D	L_2 not in D

To Show L2 undecidable

1. Choose a language L_1 that is already known not to be in D,
   A. Assume a TM *Oracle* that decides L_2
   B. show that L_1 can be reduced to L_2

Details:

2. Define the reduction *R.*

3. Describe the composition *C* of *R* with *Oracle*.

4. Show that *C* correctly decides L_1 iff *Oracle* exists. We do this by showing:
   - *R* can be implemented by Turing machines,
   - *C* is correct:
     - If $x \in L_1$, then $C(x)$ accepts, and
     - If $x \notin L_1$, then $C(x)$ rejects.

First Reduction Example:
H_ε = {<*M*> : TM *M* halts on ε}

Follow this outline in proofs that you submit.. We will see many examples in the next few sessions.

show H_ε in SD but not in D

1. **H_ε is in SD.** ***T*** **semidecides it:**

T(<*M*>) =
   1. Run *M* on ε.
   2. Accept.

T accepts <*M*> iff *M* halts on ε, so *T* semidecides H_ε.

* **Recall:** "M halts on w" is a short way of saying "M, when started with input w, eventually halts"

H_ε = {<*M*> : TM *M* halts on ε}

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

Proof: by reduction from H to H_ε :

$H_\varepsilon \leq H$ **is intuitive, the other direction is not so obvious.**

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 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#*>.

*

Proof, Continued

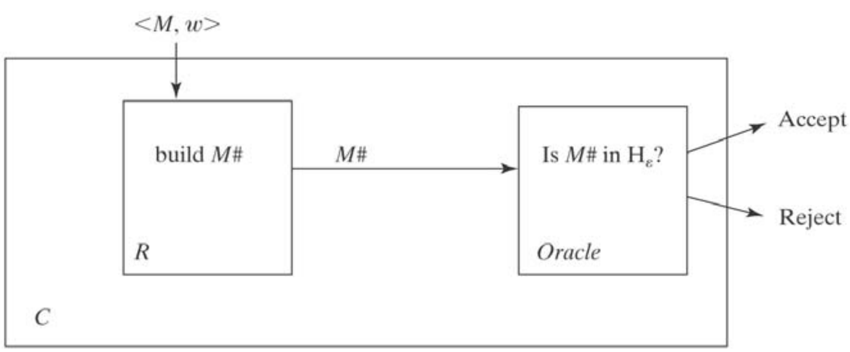
R(<*M*, *w*>) =

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 every input or no inputs. So there are two cases:
    - <*M*, *w*> $\in$ *H*: *M* halts on *w*, so *M#* halts on everything. In particular, it halts on ε. *Oracle* accepts.
    - <*M*, *w*> $\notin$ *H*: *M* does not halt on *w*, so *M#* halts on nothing and thus not on ε. *Oracle* rejects.

A Block Diagram of *C*

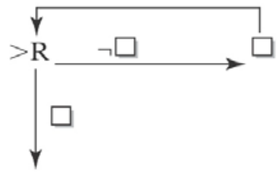


Note: In the last two places where M# appears in this diagram, it should be **<M#>**

R Can Be Implemented as a Turing Machine

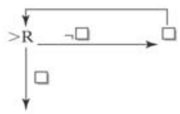
R must construct <*M#*> from <*M*, *w*>. Suppose *w* = aba.

M# will be:



So the procedure for constructing *M#* is: a R b R a $L_{\square}$ *M*

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 Are We 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

Different 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}

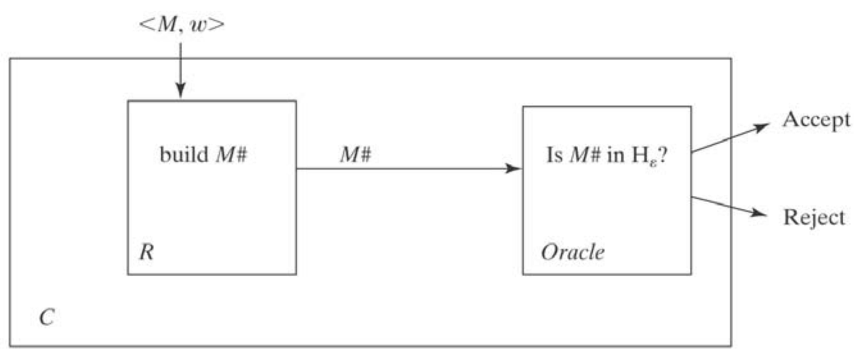
R is a reduction from H to Hε:

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 with an encoding whose membership in H we are interested in determining; thus also an input to *R*).

A Block Diagram of *C*



Note: In the last twoplaces where M# appears in this diagram, it should be <M#>

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 a Reduction Proof

- A clear declaration of the reduction "from" and "to" languages.
- A clear description of *R*.
- If *R* is doing anything nontrivial, argue that 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.
- Run through 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 *H*, 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 …

Next Example:

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

Hidden: H_{ANY} is not in D

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

R ↓

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

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

1. Construct <*M#*>, where *M#*(*x*) operates as follows:
    1.1. Examine *x*.
    1.2. If *x* = *w*, run *M* on *w*, else loop.
2. Return <*M#*>.

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

- *R* can be implemented as a Turing machine.
- *C* is correct: The only string on which *M#* can halt is *w*. So:
  - <*M*, *w*> ∈ H: *M* halts on *w.* So *M#* halts on *w*. There exists at least one string on which *M#* halts. *Oracle* accepts.
  - <*M*, *w*> ∉ H: *M* does not halt on *w*, so neither does *M#*. So there exists no string on which *M#* halts. *Oracle* rejects.

But no machine to decide H can exist, so neither does *Oracle*.

Hidden: (Another *R* That Works)

Proof: We show that H_{ANY} is not in D by reduction from H:

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

R ↓

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

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

1. Construct the description <*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#*>.

If *Oracle* exists, then *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. So it halts on at least one string. *Oracle* accepts.
  - <*M*, *w*> ∉ H: *M* does not halt on *w*, so *M#* halts on nothing. So it does not halt on at least one string. *Oracle* rejects.

But no machine to decide H can exist, so neither does *Oracle*.

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* halts on *w*}
Does TM *M* not halt on *w*?	¬H = {<*M*, *w*> : *M* does not halt on *w*}
Does TM *M* halt on the empty tape?	H_ε = {<*M*> : *M* halts on ε}
Is there any string on which TM *M* halts?	H_{ANY} = {<*M*> : there exists at least one string on which TM *M* halts }
Does TM *M* accept all strings?	A_{ALL} = {<*M*> : *L*(*M*) = Σ*}
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*) is regular}

Next: We examine proofs of some of these (some are also done in the textbook)