Undecidability

Sections 21.1 – 21.3

Problems / Languages

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□ = $\{<M> : M$ halts on □$\}$
Is there any string on which TM M halts?	HANY = $\{<M>$: there exists at least one string on which TM M halts $\}$
Does TM M accept all strings?	AALL = $\{<M> : L(M) =$ □$^*\}$
Do TMs Ma and Mb accept the same languages?	EqTMs = $\{<M\text{a}, M\text{b}> : L(M\text{a}) = L(M\text{b})\}$
Is the language that TM M accepts regular?	TMREG = $\{<M>:L(M)$ is regular$\}$

Reduction

- **Example:** Computing a function

multiply(*x*, *y*) =

1. *answer* := 0.
2. For *i* := 1 to *y* do:

   answer = ***sum***(*answer* + *x*).
3. Return *answer.*

- Computing ***multiply*** reduces to computing ***sum****.*

or

- If we can do ***sum*** then we can do ***multiply****.*

Using Reduction for Undecidability

Theorem: There exists no general procedure to solve the following problem:

> Given an angle *A*, divide *A* into sixths using only a straightedge and a compass.

Proof: Suppose that there were such a procedure, which we'll call *sixth*. Then we could trisect an arbitrary angle:

trisect(*a*: angle) =

1. Divide *a* into six equal parts by invoking *sixth*(*a*).
2. Ignore every other line, thus dividing *a* into thirds.

trisect(*a*)

sixth(*a*) ignore lines

sixth exists $\Rightarrow$ *trisect* exists.

But we know that *trisect* does not exist. So, *sixth* cannot exist either.

Turing Reduction

A ***reduction*** *R* from *L*1 to *L*2 is one or more Turing machines such that:

If there exists a Turing machine *Oracle* that decides (or semidecides) *L*2, then the Turing machines in *R* can be composed with *Oracle* to build a deciding (or a semideciding) Turing machine for *L*1.

L1 ≤ *L2* means that *L1* is **reducible** to *L2*.

Using Reduction for Undecidability

Assume:

(*L1* □ *L2*) □ (*L*2 is in D) □ (*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 (*L*1 □ *L*2) is true,
then (*L*2 is in D) must be false.

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

$R \downarrow$ $\uparrow$ $\downarrow$

*L*2 (a new language whose decidability we are trying to determine) if *L*2 in D So *L*2 not in D

$L1 \longrightarrow L2$

Hard
(known)
"hard" = difficult,
e.g., not in D or SD

Harder
(unknown)
"harder" = more difficult,
because *L*1 reduces to it
(may be equally difficult)

To Use Reduction for Undecidability

0. Assume *Oracle* that decides *L2* exists

1. Choose a language *L*1:
- that is already known not to be in D, and
- that can be reduced to *L*2.

2. Define the reduction *R.*

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

$$C(x) = Oracle(R(x))$$

4. Show that *C* does correctly decide *L*1 if *Oracle* exists. We do this by showing:
- *R* can be implemented by Turing machines,
- *C* is correct:
- If $x \in L1$, then $C(x)$ accepts, and
- If $x \notin L1$, then $C(x)$ rejects.

Mapping Reductions

*L*1 is ***mapping reducible*** to *L*2 ($L1 \leq_M L2$) iff there exists some computable function *f* such that:

$$\forall x \in \Sigma^* \ (x \in L1 \leftrightarrow f(x) \in L2).$$

To decide whether *x* is in *L*1, we transform it, using *f*, into a new object and ask whether that object is in *L*2.

Note: mapping reduction is a particular case of Turing reduction.

Hε = {<*M*> : TM *M* halts on ε}

Hε 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 Hε.

H□ = {<*M*> : TM *M* halts on □}

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

Proof: by reduction from H:

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

R ↓

(?*Oracle*) H□ = {<*M*> : TM *M* halts on □}

R is a mapping reduction from H to H□:

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

1. Construct <*M#*>, where *M#*(*x*) operates as follows:
   1.1. Erase the tape (ignore its input)
   1.2. Write *w* on the tape.
   1.3. Run *M* on *w*.
2. Return <*M#*>.

Proof, Continued

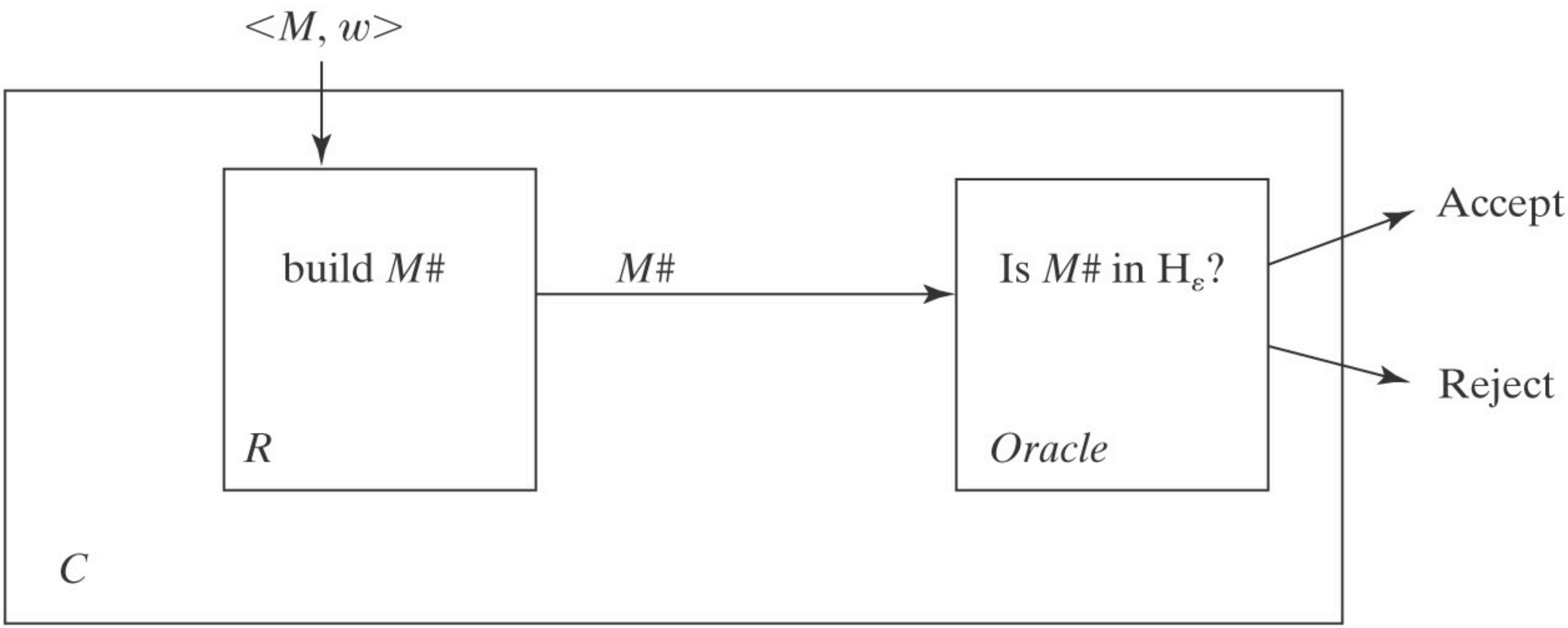
R(<*M*, *w*>) =

1. Construct <*M#*>, where *M#*(*x*) operates as follows:
    1.1. Erase the tape.
    1.2. Write *w* on the tape.
    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*> $\in$ *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.

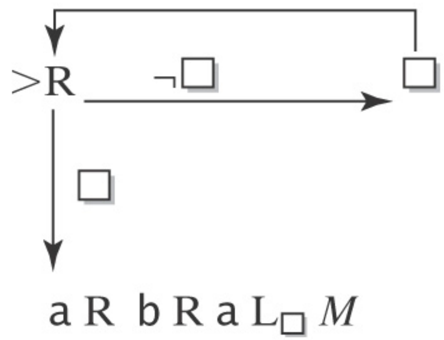
A Block Diagram of *C*



R Can Be Implemented as a Turing Machine

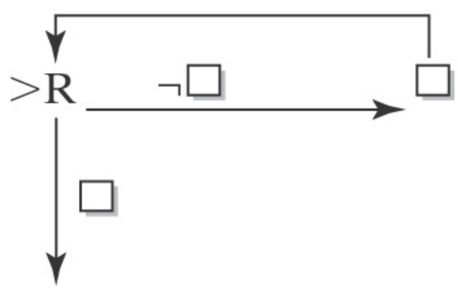
R must construct <*M#*> from <*M*, *w*>. Suppose *w* = `aba`.

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 Lq *M*.

Conclusion

R can be implemented as a Turing machine.

C is correct.

So, if *Oracle* exists:

$C = \mathit{Oracle}(R(<M, w>))$ 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 □, 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 doing a reduction in that direction would tell us nothing about whether H□ was decidable.

The significant thing that we just saw in this proof is that there also exists a reduction in the direction that does tell us that H□ is
not decidable.

We proved that H□ is “harder” than H. Clearly, H is also “harder” than H□ (H□ is a subproblem of H), so they are equally

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.
- 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

- Right way: to show that *L*2 is not in D:

1. Reduce a known hard one, *L*1, to *L*2: *L*1 → *L*2
2. Given that *L*1 is not in D,
3. 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)

- Wrong way: reduce *L*2 (the unknown one) to *L*1 (the known hard):

Example (wrong):
If there exists a machine *M*H that solves *H*, then we could build a machine that solves *H*□ as follows:

1. Return (*M*H(<*M*, □>)).

This proves nothing. It's an argument of the form:
If *False* then …
(We show that a known hard one is harder than our language …)

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

Theorem: HANY is in SD.

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

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

HANY 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 HANY.

HANY is not in D

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

$\downarrow$ *R*

(?*Oracle*) HANY = {<*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:

1. *R* can be implemented as a Turing machine.
2. *C* is correct: The only string on which *M#* can halt is *w*. So:
    1. <*M*, *w*> ∈ H: *M* halts on *w*. So *M#* halts on *w*. There exists at least one string on which *M#* halts. *Oracle* accepts.
    2. <*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*.

HANY is not in D: another reduction

Proof: We show that HANY is not in D by reduction from H:

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

$\downarrow$ *R*

(?*Oracle*) HANY = {<*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:

1. *C* is correct: *M#* ignores its own input. It halts on everything or nothing. So:
   1. <*M*, *w*> $\in$ H: *M* halts on *w*, so *M#* halts on everything. So it halts on at least one string. *Oracle* accepts.
   2. <*M*, *w*> $\notin$ H: *M* does not halt on *w*, so *M#* halts on nothing. So it does not halt on at least one string. *Oracle* rejects.

The Steps in a Reduction Proof

1. Assume *Oracle* exists.

2. Choose an undecidable language to reduce from.

3. Define the reduction *R*.

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

HALL = {<*M*> : TM *M* halts on all inputs}

We show that HALL is not in D by reduction from H□.

H□ = {<*M*> : TM *M* halts on □}

R

(?*Oracle*) HALL = {<*M*> : TM *M* halts on all inputs }

R(<*M*>) =

1. Construct the description <*M#*>, where *M#*(*x*) operates as follows:
    1.1. Erase the tape.
    1.2. Run *M*.
2. Return <*M#*>.

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

- *R* can be implemented as a Turing machine.
- *C* is correct: *M#* halts on everything or nothing, depending on whether *M* halts on □. So:
    - <*M*> □ H□: *M* halts on □, so *M#* halts on all inputs. *Oracle* accepts.
    - <*M*> □ H□: *M* does not halt on □, so *M#* halts on nothing. *Oracle* rejects.

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

The Membership Question for TMs

We next define a new language:

$$A = \{<M, w> : M \text{ accepts } w\}.$$

Note that A is different from H since it is possible that M halts but does not accept. An alternative definition of A is:

$$A = \{<M, w> : w \in L(M)\}.$$

A = {<*M*, *w*> : *w* ∈ *L*(*M*)}

We show that A is not in D by reduction from H.

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

R

(?*Oracle*) A = {<*M, w* > : *w* ∈ *L*(*M*) }

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*.
    1.4. **Accept**
2. Return <*M#, w*>.

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

- *R* can be implemented as a Turing machine.
- *C* is correct: *M#* accepts everything or nothing. So:
- <*M, w*> ∈ H: *M* halts on *w*, so *M#* accepts everything. In particular, it accepts *w*. *Oracle* accepts.
- <*M, w* > ∉ H: *M* does not halt on *w*. *M#* gets stuck in step 1.3 and so accepts nothing. *Oracle* rejects.

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

A□, AANY, and AALL

Theorem: A□ = {<*M*> : TM *M* accepts □} is not in D.

Proof: Analogous to that for H□.

Theorem: AANY = {<*M*> :TM *M* accepts at least one string} is not in D.

Proof: Analogous to that for HANY.

Theorem: AALL = {<*M*> : *L*(*M*) = □*} is not in D.

Proof: Analogous to that for HALL.

EqTMs={<*Ma*, *Mb*>: *L*(*Ma*)=*L*(*Mb*)}

AALL = $\{\langle M\rangle : L(M) = \Sigma^*\}$

$\downarrow R$

(*Oracle*) EqTMs = $\{\langle Ma, Mb\rangle : L(Ma) = L(Mb)\}$

R(<*M*>) =

1. Construct the description of *M#*(*x*):
   1.1. Accept. (*M#* accepts everything)
2. Return <*M*, *M#*>.

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

- *C* is correct: *M#* accepts everything. So if *L*(*M*) = *L*(*M#*), *M* must also accept everything. So:
- $\langle M\rangle \in$ AALL: $L(M) = L(M\#)$. *Oracle* accepts.
- $\langle M\rangle \notin$ AALL: $L(M) \neq L(M\#)$. *Oracle* rejects.

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

Sometimes Mapping Reducibility Isn't Right

Recall that a mapping reduction from $L1$ to $L2$ is a computable function f where:

$\forall x \in \Sigma^* \; (x \in L1 \leftrightarrow f(x) \in L2)$.

When we use a mapping reduction, we return:

Oracle(f(x))

Sometimes we need a more general ability to use *Oracle* as a subroutine and then to do other computations after it returns.

{<*M*> : *M* accepts no even length strings}

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

$$\Big\downarrow R$$

(?*Oracle*) *L*2 = {<*M*> : *M* accepts no even length strings}

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*.
    1.4. Accept.
2. Return <*M#*>.

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

- *C* is correct: *M#* ignores its own input. It accepts everything or nothing, depending on whether it makes it to step 1.4. So:
- <*M*, *w*> ∈ H: *M* halts on *w*. *Oracle*:
- <*M*, *w*> ∉ H: *M* does not halt on *w*. *Oracle*:

Problem:

{<*M*> : *M* accepts no even length strings}

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

R $\downarrow$

(?*Oracle*) *L2* = {<*M*> : *M* accepts no even length strings}

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*.
   1.4. Accept.
2. Return <*M#*>.

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

- *R* and □ can be implemented as Turing machines.
- *C* is correct:
- <*M*, *w*> □ H: *M* halts on *w. M#* accepts everything, including some even length strings. *Oracle* rejects so *C* accepts.
- <*M*, *w*> □ H: *M* does not halt on *w*. *M#* gets stuck. So it accepts nothing, so no even length strings. *Oracle* accepts. So *C* rejects.

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

Are All Questions about TMs Undecidable?

L = {<M> : TM M contains an even number of states}

L = {<M, w> : M halts on w within 3 steps}.

Lq = {<M, q> : there is some configuration (p, $u\underline{a}v$) of M, with p □ q, that yields a configuration whose state is q}.