Theory of Computation
Week 11

Much of the material on this slides comes from the recommended textbook by Elaine Rich

Detailed content

Weekly program

- ✓ Week 1 – Background knowledge revision: logic, sets, proof techniques
- ✓ Week 2 – Languages and strings. Hierarchies. Computation. Closure properties
- ✓ Week 3 – Finite State Machines: non-determinism vs. determinism
- ✓ Week 4 – Regular languages: expressions and grammars
- ✓ Week 5 – Non regular languages: pumping lemma. Closure
- ✓ Week 6 – Context-free languages: grammars and parse trees
- ✓ Week 7 – Pushdown automata
- ✓ Week 8 – Non context-free languages: pumping lemma and decidability. Closure
- ✓ Week 9 – Decidable languages: Turing Machines
- ✓ Week 10 – Church-Turing thesis and **the unsolvability of the Halting Problem**

➡ **Week 11 – Decidable, semi-decidable and undecidable languages (and proofs)**

- ❑ Week 12 – Revision of the hierarchy. Safety-critical systems
- ❑ Week 13 – Extra revision (if needed)

Week 11 Lecture

Decidable, semi-decidable and undecidable languages

- ❑ The Halting Problem H
- ❑ Implications of the undecidability of H
- ❑ Relation between D and SD classes
- ❑ Reduction
- ❑ Using Reduction to prove undecidablity

Videos to watch before lecture

Additional videos to watch for this week

THE UNIVERSAL TURING MACHINE

To define the Universal Turing Machine *U* we need to:

1. Define an encoding operation for TMs.

2. Describe the operation of *U* given input <*M*, *w*>, the encoding of:

    - a TM *M*, and
    - an input string *w*.

AN ENCODING EXAMPLE for U

Consider *M* = ({*s*, *q*, *h*}, {a, b, c}, {❑, a, b, c}, δ, *s*, {*h*}):

state	symbol	δ
s	❑	(*q*,❑, →)
s	a	(*s*,b,→)
s	b	(*q*,a, ←)
s	c	(*q*,b, ←)
q	❑	(*s*,a, →)
q	a	(*q*,b,→)
q	b	(*q*,b, ←)
q	c	(*h*,a, ←)

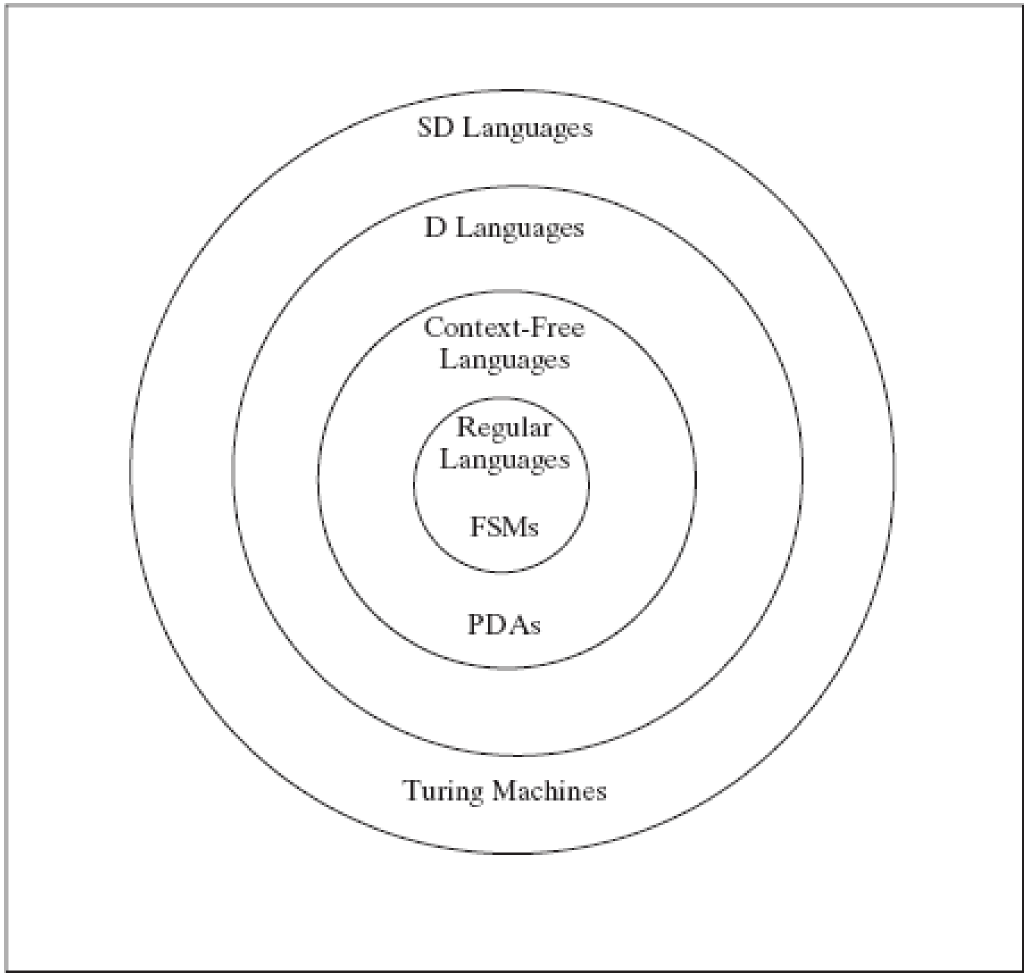
state/symbol	representation
s	q00
q	q01
h	h10
❑	a00
a	a01
b	a10
c	a11

<*M*> = (q00,a00,q01,a00,→), (q00,a01,q00,a10,→),
(q00,a10,q01,a01, ←), (q00,a11,q01,a10,←),
(q01,a00,q00,a01,→), (q01,a01,q01,a10,→),
(q01,a10,q01,a11,←), (q01,a11,h11,a01,←)

CONCEPTS

- Turing Machines are described as strings
  - For example: <M> , <M, w> , $<M_1, M_2>$
- Languages are set of strings over finite alphabet
  - These strings are encoding of Turing machines
  - L = {<M,w>: Turing machine M that halts on input w}
- We can build Turing machines to enumerate some other Turing machine
- Decidable (D) / Semidecidable languages (SD)

THE HIERARCHY



DEFINING THE UNIVERSE

L_1 = {<M, w> : TM M halts on input string w}.
L_2 = {<M> : M halts on nothing}.
L_3 = {<M_a, M_b> : M_a and M_b halt on the same strings}.

TM: (,),q,y,n,a,0,1,→,←, ' ,
Tape Char: x, y, ❑

For a string s to be in L_1, it must

- be syntactically well-formed.
- encode a machine M and a string w such that M halts when started on w.

Define the universe from which we are drawing strings to contain only those strings that meet the syntactic requirements of the language definition.

This convention has no impact on the decidability of any of these languages since the set of syntactically valid strings is in D.

DEFINITION OF COMPLEMENT

Define the ***complement*** of any language L whose member strings include at least one Turing machine description to be with respect to a universe of strings that are of the same syntactic form as L.

$L_1 = \{<M, w> :$ TM M halts on input string $w\}$.

$\neg L_1 = \{<M, w> :$ TM M does not halt on input string $w\}$.

THE LANGUAGE H

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

Theorem: The language:

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

- is semidecidable, but
- is not decidable.

DOES THIS PROGRAM HALT?

times3(x: positive integer) =

While $x \neq 1$ do:

If x is even then $x = x/2$.

Else $x = 3x + 1$

25

DOES THIS PROGRAM HALT?

times3(*x*: positive integer) =

While $x \neq 1$ do:

If x is even then $x = x/2$.

Else $x = 3x + 1$

25

76

38

19

58

DOES THIS PROGRAM HALT?

times3(*x*: positive integer) =

While $x \neq 1$ do:

If x is even then $x = x/2$.

Else $x = 3x + 1$

25	29
76	88
38	44
19	22
58	11

DOES THIS PROGRAM HALT?

times3(*x*: positive integer) =

While $x \neq 1$ do:

If x is even then $x = x/2$.

Else $x = 3x + 1$

25	29	34
76	88	17
38	44	52
19	22	26
58	11	13

DOES THIS PROGRAM HALT?

times3(x: positive integer) =

While $x \neq 1$ do:

If x is even then $x = x/2$.

Else $x = 3x + 1$

25	29	34	40
76	88	17	20
38	44	52	10
19	22	26	5
58	11	13	16

DOES THIS PROGRAM HALT?

times3(*x*: positive integer) =
  While $x \neq 1$ do:
    If x is even then $x = x/2$.
    Else $x = 3x + 1$

25	29	34	40	8
76	88	17	20	4
38	44	52	10	2
19	22	26	5	1
58	11	13	16	

http://www.numbertheory.org/php/collatz.html

H IS SEMIDECIDABLE

Lemma: The language:

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

is semidecidable.

Proof:

H IS SEMIDECIDABLE

Lemma: The language:

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

is semidecidable.

Proof: The TM M_H semidecides H:

M_H(<M, w>) =

1. Run M on w.
2. Accept.

M_H accepts iff M halts on w. Thus M_H semidecides H.

THE UNSOLVABILITY OF THE HALTING PROBLEM

Lemma: The language:

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

is not decidable.

Proof: If H were decidable, then some TM M_H would decide it. M_H would implement the specification:

halts(<*M*: string, *w*: string>) =
  If <*M*> is a Turing machine description
    and *M* halts on input *w*
      then accept.
      Else reject.

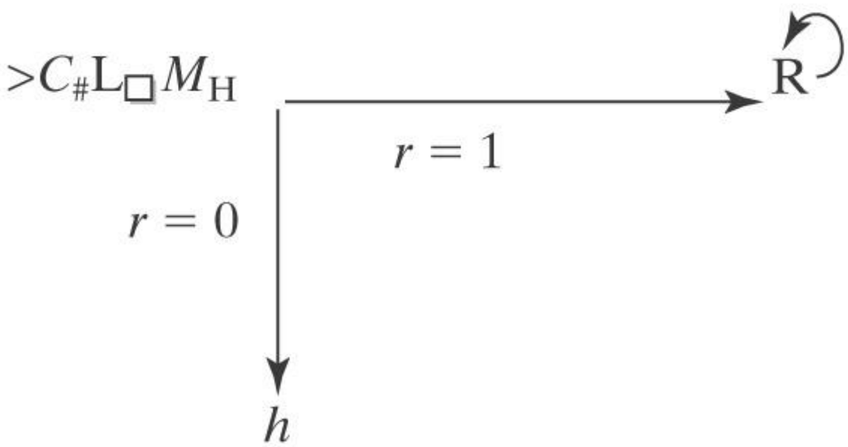
Trouble

Trouble(*x*: string) =

if *halts* accepts (*x*, *x*) then loop forever, else halt.

If there exists an M_H that computes the function *halts*, *Trouble* exists:

$C_{\#}$ writes onto its tape a second copy of its input, separated from the first by a comma



What is *Trouble*(<*Trouble*>)?

What is M_H(<*Trouble*, *Trouble*>)?

- If M_H reports that *Trouble*(<*Trouble*>) halts, *Trouble* loops.
- But if *halts* reports that *Trouble*(<*Trouble*>) does not halt, then *Trouble* halts.

Unsolvability of Halting Problem

https://www.youtube.com/watch?v=92WHN-pAFCs

VIEWING THE HALTING PROBLEM AS DIAGONALIZATION

- Lexicographically enumerate Turing machines.
- Let 1 mean halting, blank mean non halting.

	i_1	i_2	i_3	…	<*Trouble*>	…
$machine_1$	1					
$machine_2$		1				
$machine_3$					1	
…				1		
Trouble			1			1
…	1	1	1			
…				1		

But *Trouble* behaves as:

Trouble			1		1	1

Or maybe *halts* said that *trouble*(<*trouble*>) would halt. But then *trouble* would loop.

A Very Important Result

Can be stated in any of the following three ways:

- The language H is not decidable
- The halting problem is unsolvable
  - No implementation of the specification of the *halts* function
- The membership problem for the SD languages is not solvable

IMPLICATIONS

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

Theorem: If H were in D then every SD language would be in D.

Proof: Let *L* be any SD language. There exists a TM M_L that semidecides it.

If H were also in D, then there would exist an *Oracle*, *O,* that decides it.

IMPLICATIONS

To decide whether w is in $L(M_L)$:

M'(w: string) =

1. Run O on <M_L, w>.
2. If O accepts (i.e., M_L will halt), then:
    2.1. Run M_L on w.
    2.2. If it accepts, accept. Else reject.
3. Else reject.

So, if H were in D, all SD languages would be.

BACK TO THE ENTSCHEIDUNGSPROBLEM

Theorem: The Entscheidungsproblem is unsolvable.

Proof: (Due to Turing)

1. If we could solve the problem of determining whether a given Turing machine ever prints the symbol 0, then we could solve the problem of determining whether a given Turing machine halts.

   if $< M_0>$ exists then H is decidable.

2. But we can't solve the problem of determining whether a given Turing machine halts, so neither can we solve the problem of determining whether it ever prints 0.

   But H is not decidable so $< M_0>$ does not exist

BACK TO THE ENTSCHEIDUNGSPROBLEM

Theorem: The Entscheidungsproblem is unsolvable.

Proof: (Due to Turing)

3. Given a Turing machine *M*, we can construct a logical formula *F* that is true iff *M* ever prints the symbol 0.

   if < M_0> exists then then we can construct a logical F that is theorem

4. If there were a solution to the Entscheidungsproblem, then we would be able to determine the truth of any logical sentence, including *F* and thus be able to decide whether *M* ever prints the symbol 0.

   If entscheidungsproblem were solvable we can prove F is a theorem

5. But we know that there is no procedure for determining whether *M* ever prints 0.
6. So there is no solution to the Entscheidungsproblem.

EVERY CF LANGUAGE IS IN D

Theorem: The set of context-free languages is a *proper* subset of D.

Proof:

- Every context-free language is decidable, so the context-free languages are a subset of D.
- There is at least one language, $A^nB^nC^n$, that is decidable but not context-free.

So the context-free languages are a *proper* subset of D.

DECIDABLE AND SEMIDECIDABLE LANGUAGES

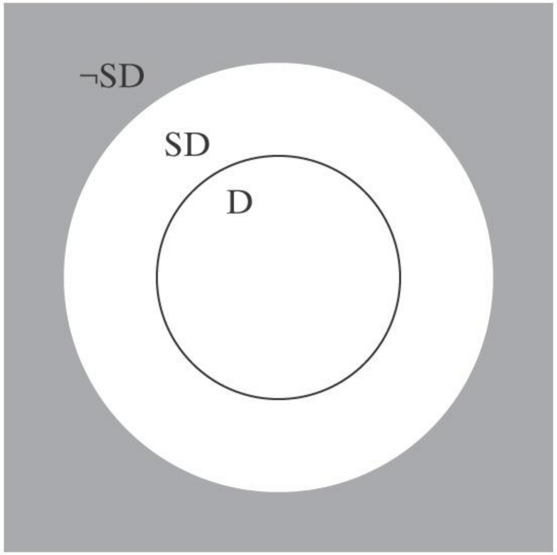
Almost every obvious language that is in SD is also in D:

- $A^nB^nC^n = \{a^n b^n c^n, n \geq 0\}$
- $\{wcw, w \in \{a, b\}^*\}$
- $\{ww, w \in \{a, b\}^*\}$
- $\{w = x*y=z: x,y,z \in \{0, 1\}^*$ and, when x, y, and z are viewed as binary numbers, $xy = z\}$

But there are languages that are in SD but not in D:

- H = {<*M*, *w*> : *M* halts on input *w*}
- {*w*: *w* is the email address of someone who will respond to a message you just posted to your newsgroup}

D and SD



1. There exists at least one language that is in SD\D, the donut in the picture.
2. D is a subset of SD. In other words, every decidable language is also semidecidable.
3. There exist languages that are not in SD. In other words, the gray area of the figure is not empty.

LANGUAGES THAT ARE NOT IN SD

Theorem: There are languages that are not in SD.

Proof: Assume any nonempty alphabet Σ.

Lemma: There is a countably infinite number of SD languages over Σ

However, there is an uncountably infinite number of languages over Σ.

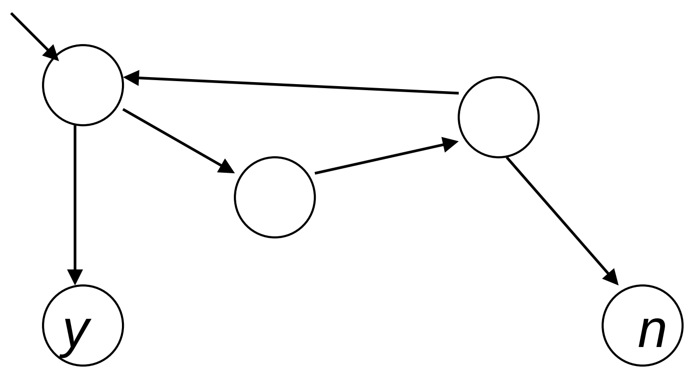
So there are more languages than there are languages in SD. Thus there must exist at least one language that is in $\neg$SD.

CLOSURE OF D UNDER COMPLEMENT

Theorem: The set D is closed under complement.

Proof: (by construction) If L is in D, then there is a deterministic Turing machine M that decides it.

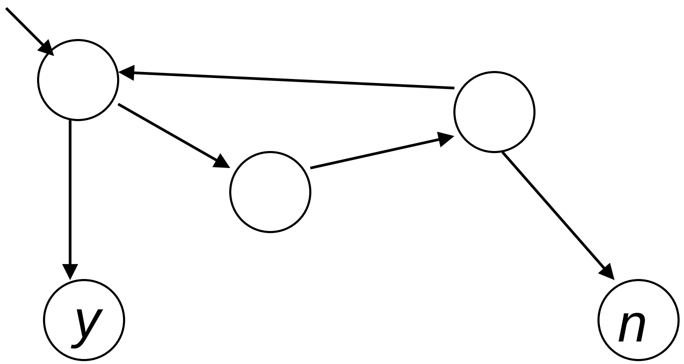
M:



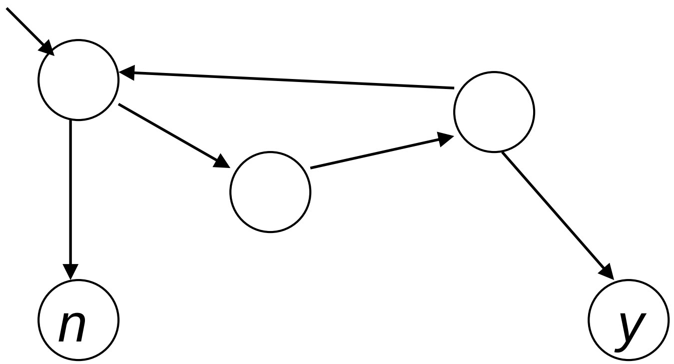
From M, we construct M' to decide $\neg L$:

CLOSURE OF D UNDER COMPLEMENT

M:



M':



This works because, by definition, *M* is:

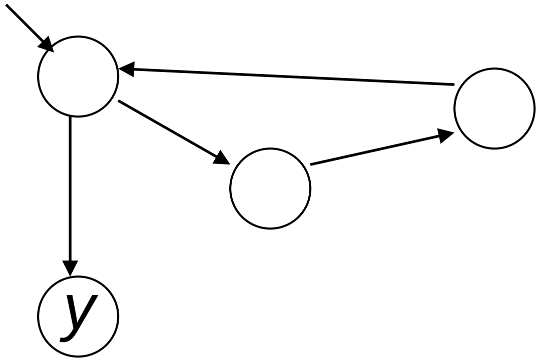
- deterministic
- complete

Since *M'* decides $\neg L$, $\neg L$ is in D.

SD IS NOT CLOSED UNDER COMPLEMENT

Can we use the same technique?

M:



M':

SD IS NOT CLOSED UNDER COMPLEMENT

Suppose we have a TM M_L that semidecided L,
And there is another TM $M_{\neg L}$ that semidecided $\neg L$,

Then, from M_L and $M_{\neg L}$ we could construct a new TM $M_\#$ that decided L as follows:

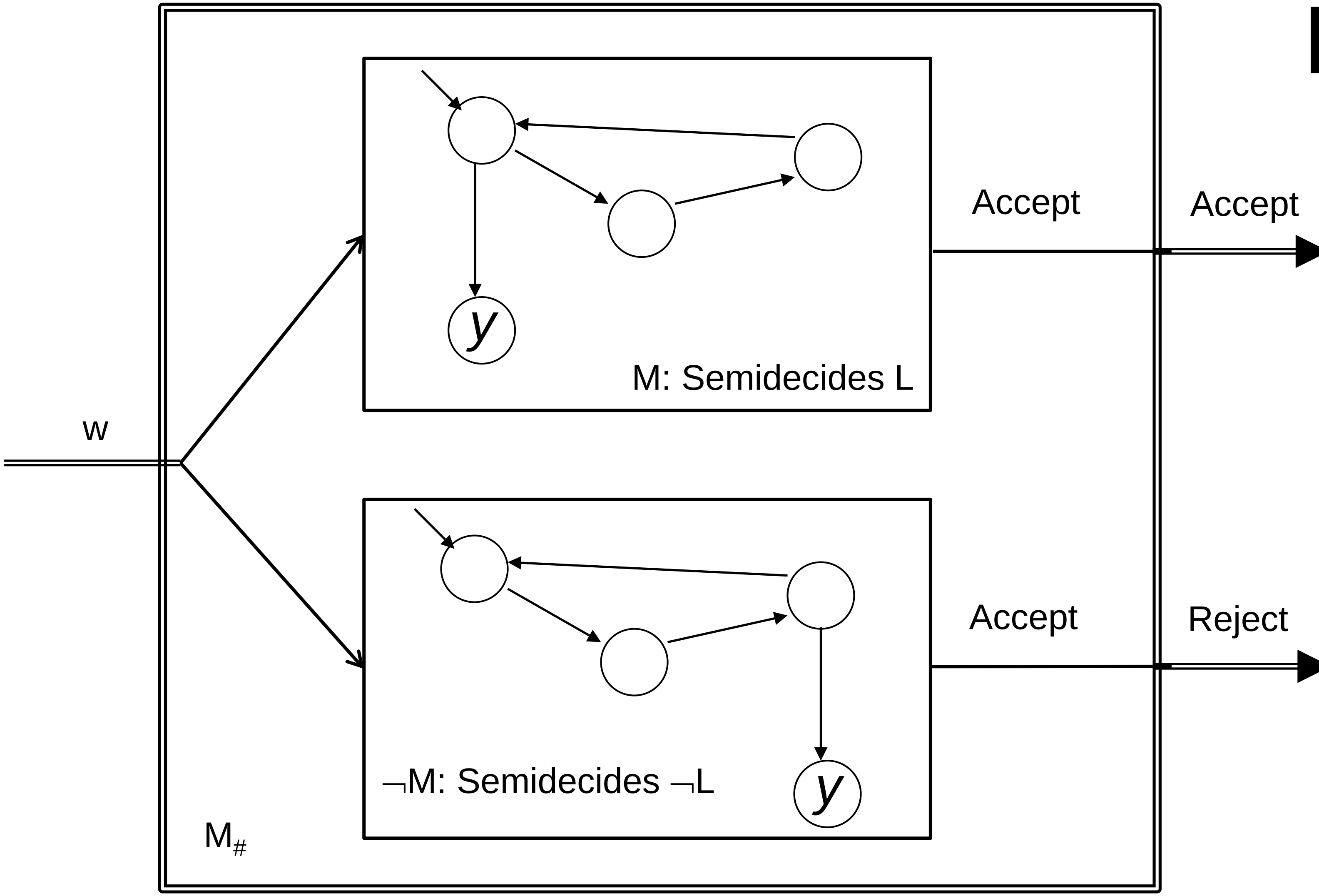
On input w, $M_\#$ would simulate M and M' in parallel, running on w. Since w must be an element of either L or $\neg L$, then one of M or M' would eventually accept.

- If M accepts, then $M_\#$ would halt and accept
- If M' accepts, then $M_\#$ would halt and reject

$M_\#$ decides L, and thus every language in SD would also be in D.

But we know that there is at least one language (H) that is in SD but not in D. Contradiction.

SD IS NOT CLOSED UNDER COMPLEMENT



D AND SD LANGUAGES

Theorem: A language is in D iff both it and its complement are in SD.

Proof:

➔ L in D implies L and $\neg L$ are in SD:

- L is in SD because D $\subset$ SD.
- D is closed under complement
- So $\neg L$ is also in D and thus in SD.

🡐 L and $\neg L$ are in SD implies L is in D:

- M_1 semidecides L.
- M_2 semidecides $\neg L$.
- To decide L:
  - Run M_1 and M_2 in parallel on w.
  - Exactly one of them will eventually accept.

A LANGUAGE THAT IS NOT IN SD

Theorem: The language $\neg H$ =

$\{<M, w> :$ TM M does not halt on input string $w\}$

is not in SD.

Proof:

- H is in SD\D.
- If $\neg H$ were also in SD then H would be in D.
- But H is not in D.
- So $\neg H$ is not in SD.

LANGUAGE SUMMARY

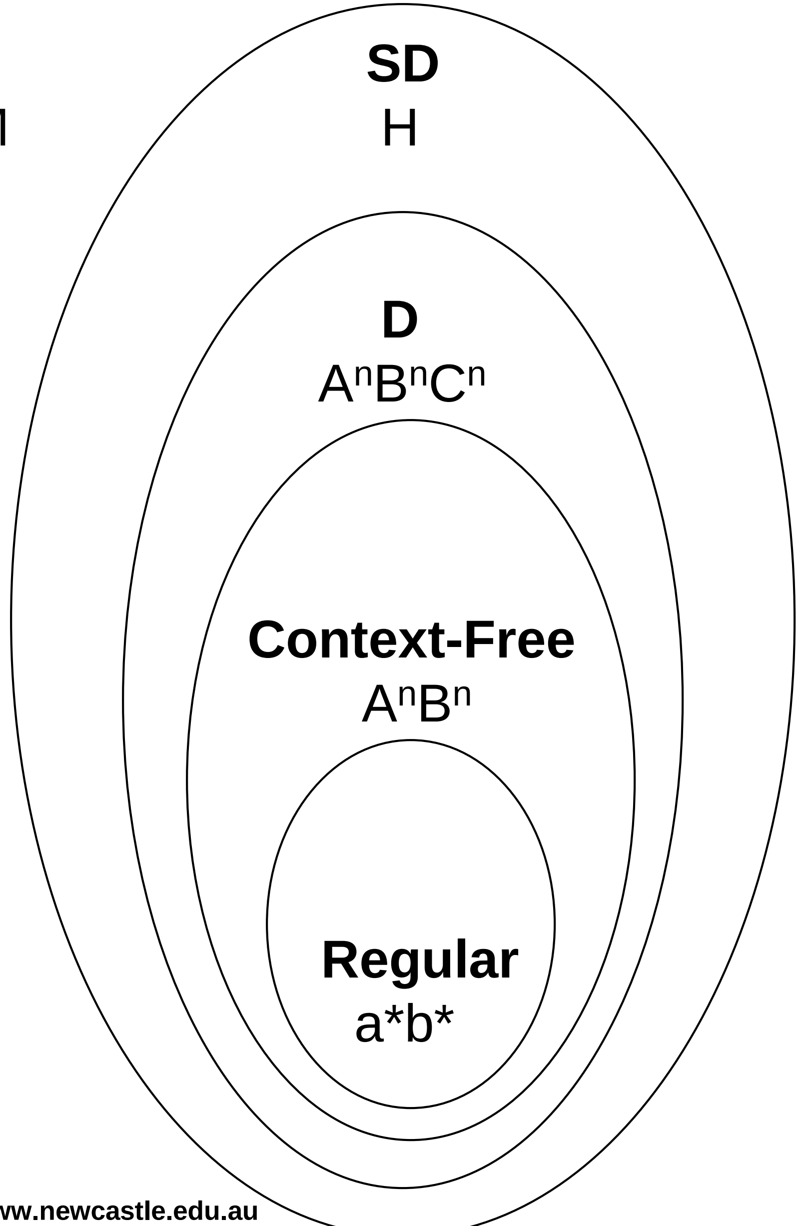
IN

Semideciding TM

Deciding TM

CF grammar
PDA
Closure

RE, RG
FSM, Closure



OUT

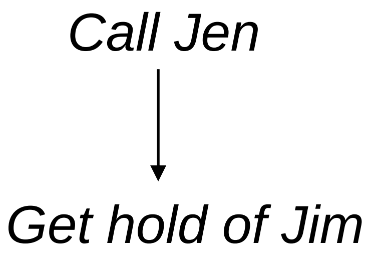
Diagonalize

Pumping
Closure

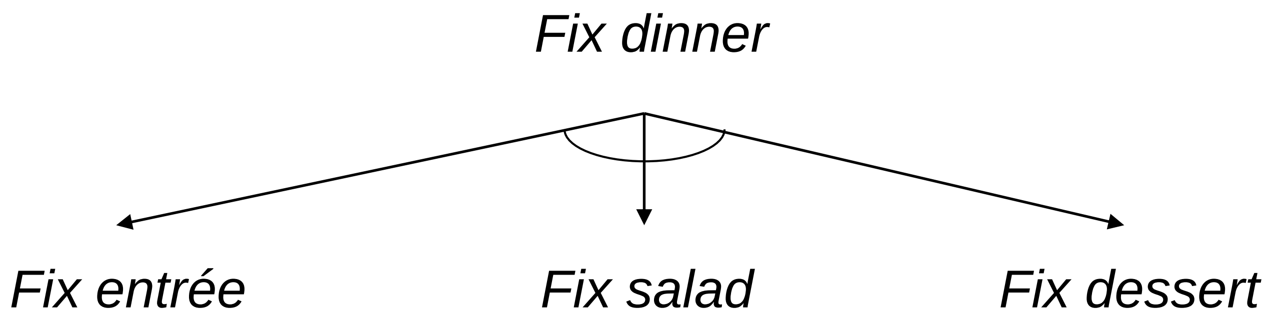
Pumping
Closure

REDUCTION

Calling Jen



Fixing dinner



REDUCTION

Computing a function

multiply(*x*, *y*) =

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

   answer = add(*answer, x*).
3. Return *answer.*

USING REDUCTION FOR UNDECIDABILITY

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

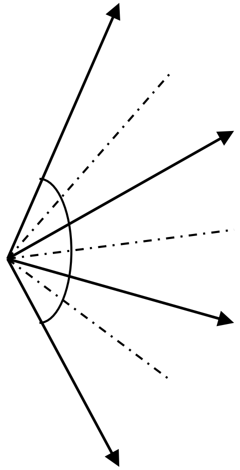
trisect: Given an arbitrary angle *A*, it is not possible to trisect *A* using only a straightedge and a compass.

New 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.



sixth exists $\rightarrow$ *trisect* exists.

But we know that *trisect* does not exist!
So neither can sixth.

USING REDUCTION FOR UNDECIDABILITY

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.

$P \leq P'$ means that P is reducible to P'.

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,

then (L_2 is in D) must be false.

USING REDUCTION FOR UNDECIDABILITY

Showing that L_2 is not in D:

$$
\begin{array}{llll}
L_1 & \text{(known not to be in D)} & L_1 \text{ in D} & \text{But } L_1 \text{ not in D} \\
 & \quad R \downarrow & \quad \Uparrow & \qquad \Downarrow \\
L_2 & \text{(a new language whose decidability we are trying to determine)} & \text{if } L_2 \text{ in D} & \text{So } L_2 \text{ not in D}
\end{array}
$$

USING REDUCTION FOR UNDECIDABILITY

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*.

4. Show that C does correctly decide 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.

H_ε = {<*M*> : TM *M* halts on ε}

$H_\varepsilon = \{<M> : \text{TM } M \text{ halts on } \varepsilon\}$

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.

$H_\varepsilon = \{<M> : \text{TM } M \text{ halts on } \varepsilon\}$

Theorem: $H_\varepsilon = \{<M> : \text{TM } M \text{ halts on } \varepsilon\}$ is not in D.

Proof: by reduction from H (i.e. we show $H \leq H_\varepsilon$)

$$
\begin{array}{c}
H = \{<M, w> : \text{TM } M \text{ halts on input string } w\} \\
\Big\downarrow R \\
H_\varepsilon\ \{<M> : \text{TM } M \text{ halts on } \varepsilon\}
\end{array}
$$

(?*Oracle*)

R is a mapping reduction from H to H_ε:

- transforms the input of H into an input suitable for *Oracle*, which we will call *M#*.
- Builds a new TM that halts on ε if and only iff *M* halts on *w*

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

$\downarrow R$

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

Oracle(<M>)
Accepts if M halts on ε
Rejects if M does not halt on ε

C: Oracle + R
C: Oracle (R <M,w>)
R<M,w> : a TM <M#> as input for oracle

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#>.

How C works:
<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.

H_ε = {<*M*> : TM *M* halts on ε}

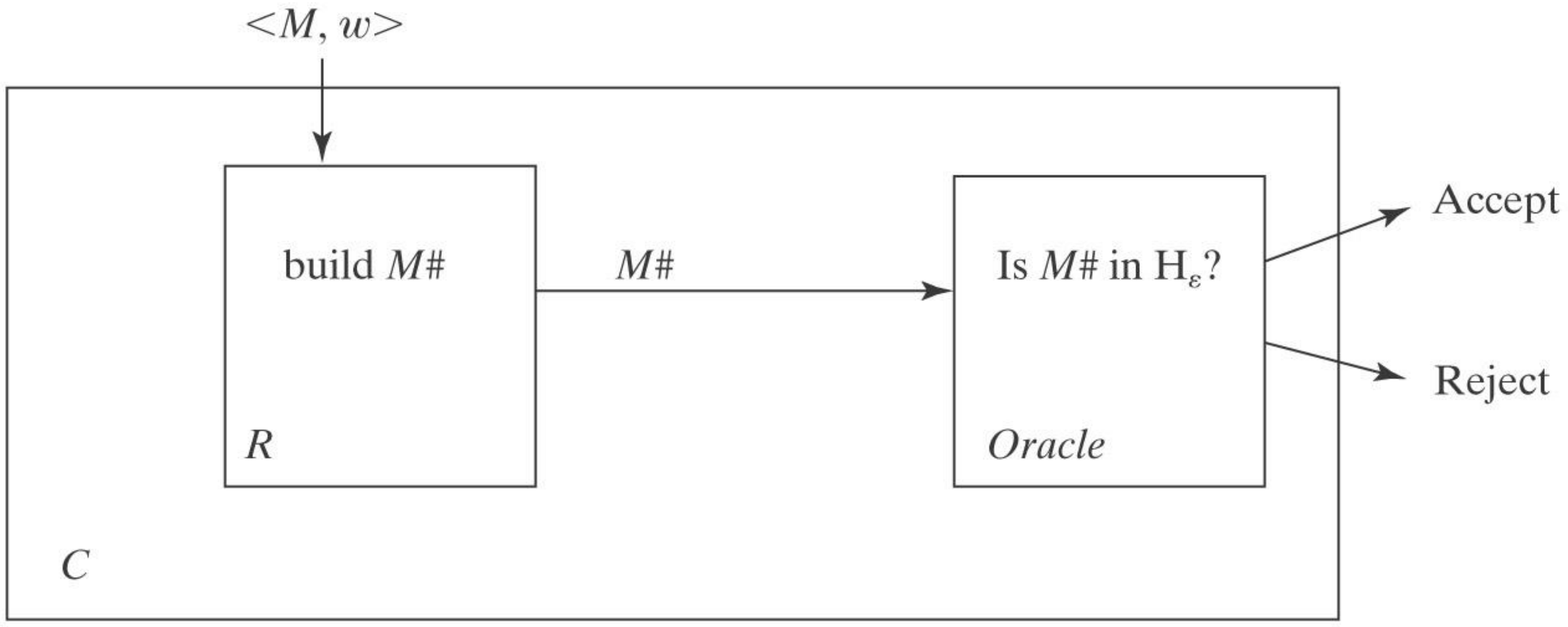
R(<*M*, *w*>) =

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

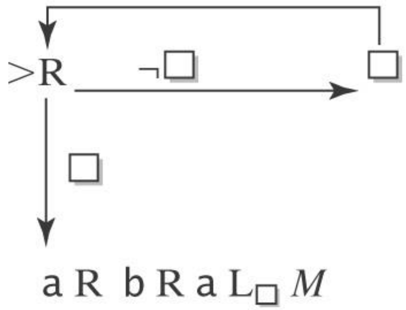
A Block Diagram of *C*



R CAN BE IMPLEMENTED AS A TURING MACHINE

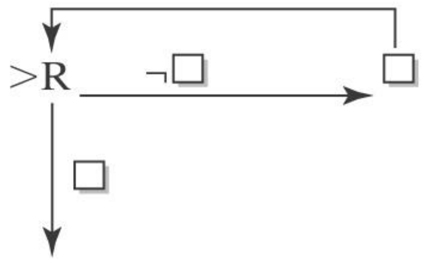
R must construct <*M#*> from <*M*, *w*>. Suppose *w* = `aba`.

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 $L_{\square}$ *M*.

CONCLUSION

R can be implemented as a Turing machine.

C is correct.

So, if *Oracle* exists:

$C = Oracle(R(\langle M, w\rangle))$ 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 doing a reduction in that direction would tell us nothing about whether H_ε 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_ε is not decidable.

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.

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}.

H_{ANY} is not in D

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

$\downarrow R$

(?*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 and decides H_{ANY}, 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*> $\in$ H: *M* halts on *w.* So *M#* halts on *w*. There exists at least one string on which *M#* halts. *Oracle* accepts.
  - <*M*, *w*> $\notin$ 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*.

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.

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:

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

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

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

$$H = \{<M, w> : \text{TM } M \text{ halts on input string } w\}$$
$$\Big\downarrow R$$
(?*Oracle*) $$A = \{<M, w> : w \in L(M)\}$$

$R(<M, w>) =$

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 $<M\#, w>$.

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

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*> $\in$ H: *M* halts on *w*, so *M#* accepts everything. In particular, it accepts *w*. *Oracle* accepts.
  - <*M, w* > $\notin$ 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_ε, A_{ANY}, and A_{ALL}

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

Proof: Analogous to that for H_ε.

Theorem:

A_{ANY} = {<*M*> : TM *M* accepts at least one string}

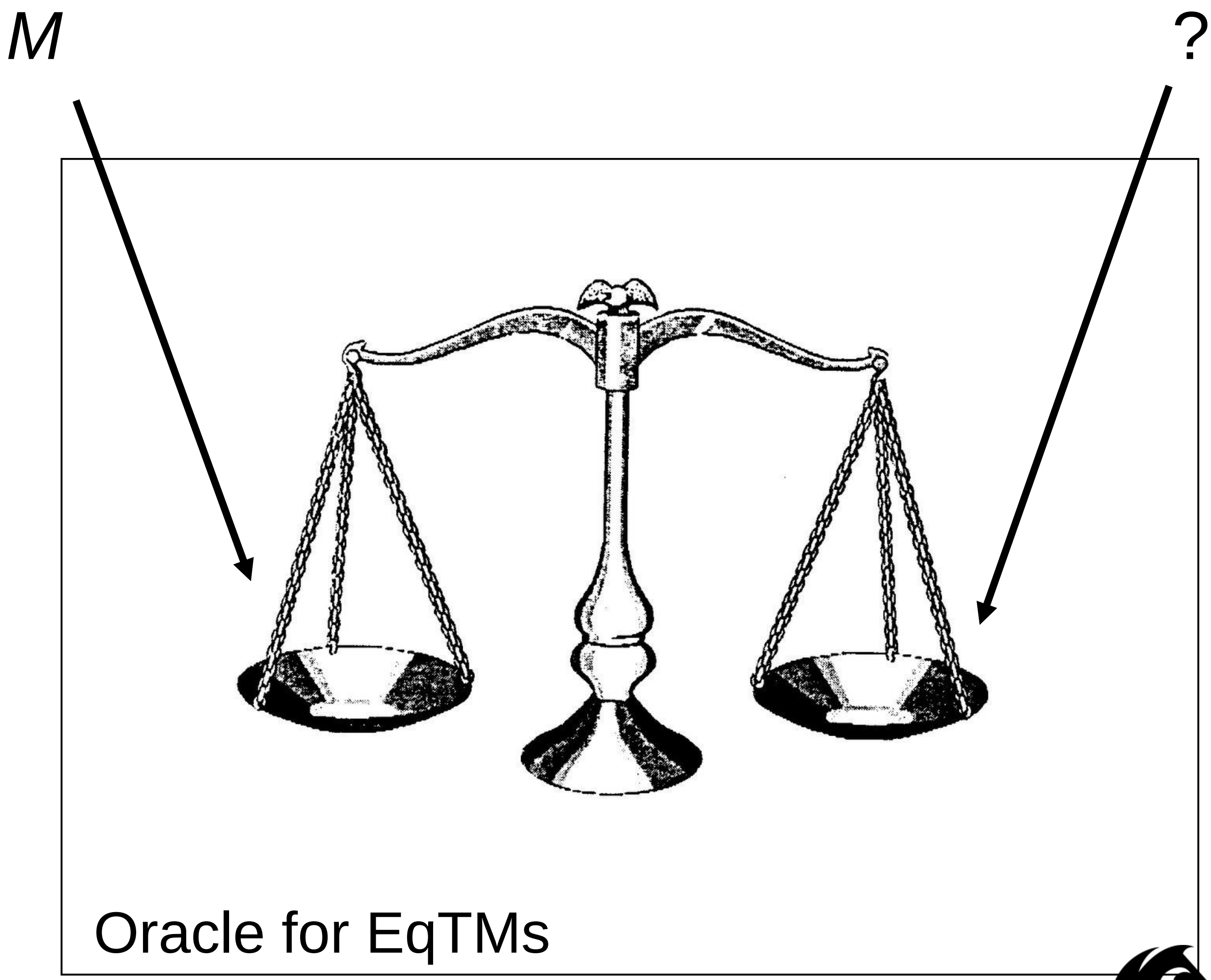
is not in D.

Proof: Analogous to that for H_{ANY}.

Theorem: A_{ALL} = {<*M*> : $L(M) = \Sigma^*$} is not in D.

Proof: Analogous to that for H_{ALL}.

EqTMs={<M_a, M_b>: $L(M_a)=L(M_b)$}



EqTMs={$<M_a, M_b>$: $L(M_a)=L(M_b)$}

$$A_{ALL} = \quad \{<M> : L(M) = \Sigma^*\}$$

$$\Big\downarrow R$$

(*Oracle*) EqTMs = $\{<M_a, M_b>: L(M_a)=L(M_b)\}$

$R(<M>) =$

1. Construct the description of $M\#(x)$:
   1.1. Accept.
2. Return $<M, M\#>$.

If *Oracle* exists, then $C = Oracle(R(<M>))$ decides A_{ALL}:

- C is correct: $M\#$ accepts everything. So if $L(M) = L(M\#)$, M must also accept everything. So:
  - $<M> \in A_{ALL}$: $L(M) = L(M\#)$. *Oracle* accepts.
  - $<M> \notin A_{ALL}$: $L(M) \neq L(M\#)$. *Oracle* rejects.

But no machine to decide A_{ALL} can exist, so neither does *Oracle*.

Are All Questions about TMs Undecidable?

Let $L = \{<M> : \text{TM } M \text{ contains an even number of states}\}$

Are All Questions about TMs Undecidable?

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

LANGUAGE SUMMARY

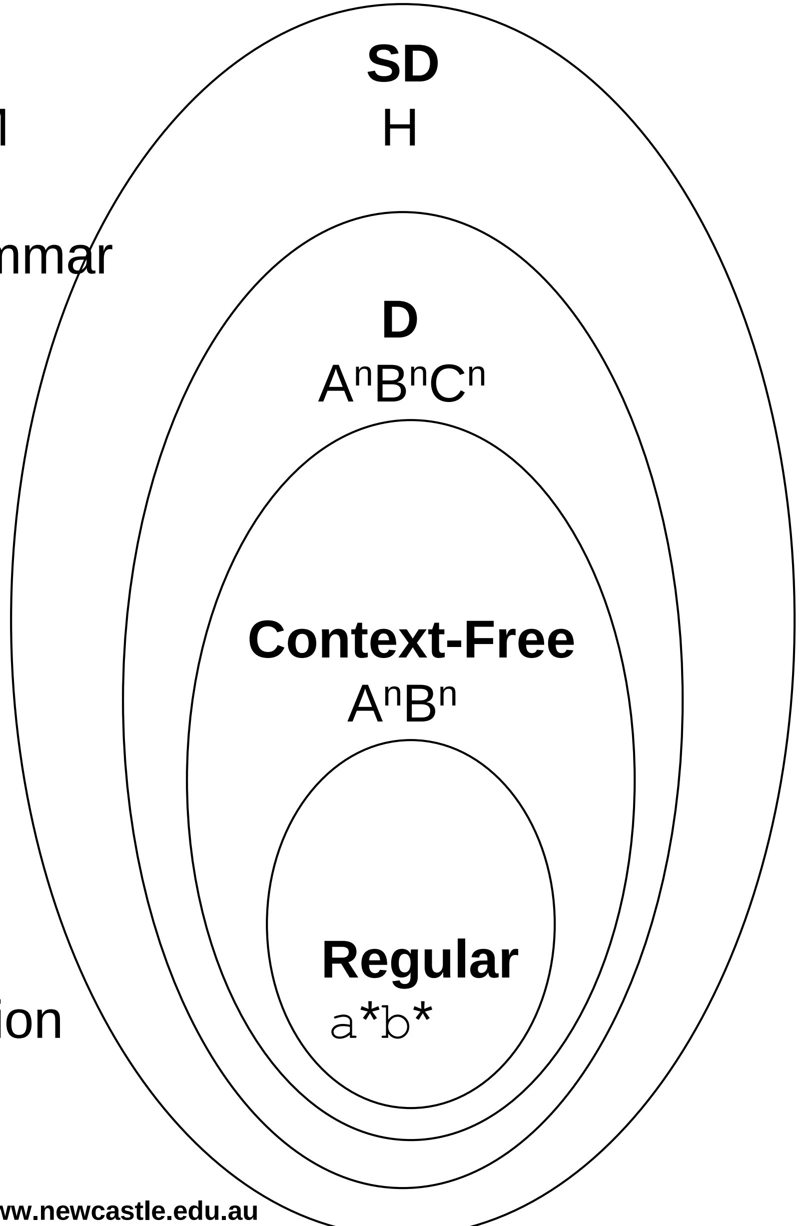
IN

Semideciding TM
Enumerable
Unrestricted grammar

Deciding TM
Lexic. enum
L and $\neg L$ in SD

CF grammar
PDA
Closure

Regular Expression
FSM



OUT

Reduction

Diagonalize
Reduction

Pumping
Closure

Pumping
Closure

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}

References

- **Automata, Computability and Complexity. Theory and Applications**

  - By Elaine Rich
- Chapter 19:

  - Page : 426-433.
- Chapter 20:

  - Page : 435-440.
- Chapter 21:

  - Page : 449-467.

DECIDING A LANGUAGE

M ***decides*** a language $L \subseteq \Sigma^*$ iff:
For any string $w \in \Sigma^*$ it is true that:

- if $w \in L$ then M accepts w, and
- if $w \notin L$ then M rejects w.

A language L is ***decidable*** iff there is a Turing machine M that decides it. In this case, we will say that L is in ***D***.

SEMIDECIDING A LANGUAGE

Let Σ_M be the input alphabet to a TM M. Let $L \subseteq \Sigma_M{}^*$.

M ***semidecides*** L iff, for any string $w \in \Sigma_M{}^*$:

- $w \in L \rightarrow M$ accepts w
- $w \notin L \rightarrow M$ does not accept w.

M may either:
reject or
fail to halt.

A language L is ***semidecidable*** iff there is a Turing machine that semidecides it. We define the set ***SD*** to be the set of all semidecidable languages.

THE LANGUAGE H

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

Theorem: The language:

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

- is semidecidable, but
- is not decidable.

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?	$TM_{REG} = \{<M>:L(M) \text{ is regular}\}$

Is There a Pattern?

- For some language L in the SD class we want to know
  - Does *L* contain some particular string *w*?
  - Does *L* contain ε?
  - Does *L* contain any strings at all?
  - Does *L* contain all strings over some alphabet Σ?

- Given a semi-deciding TM M,
  - Does M accept some particular string w?
  - Does *M* accept ε?
  - Does *M* accept any strings at all?
  - Does *M* accept all strings over some alphabet Σ?
- In the language form:
  - A = {<*M*, *w*> : TM *M* accepts *w*}.
  - A_ε = {<*M*> : TM *M* accepts ε}.
  - A_{ANY} = {<*M*> : there exists at least one string that TM *M* accepts}.
  - A_{ALL} = {<*M*> : TM *M* accepts all inputs}.

RICE'S THEOREM

No nontrivial property of the SD languages is decidable.

or

Any language L that can be described as:

$L=\{<M>: P(L(M)) = True\}$

for any nontrivial property *P*, L is not in D.

A ***nontrivial property*** is one that is not simply:

- *True* for all languages, or
- *False* for all languages.

APPLYING RICE'S THEOREM

To use Rice's Theorem to show that a language *L* is not in D we must:

- Specify property *P.*
- Show that the domain of *P* is the SD languages.
- Show that *P* is nontrivial:
  - *P* is true of at least one language
  - is false of at least one language

APPLYING RICE'S THEOREM

- {<*M*> : *L*(*M*) contains only even length strings}.
  1. Specify property *P.*
     - P is "True if L contains only even length string and False otherwise"
  2. Show that the domain of *P* is the SD languages.
     - Domain of P is the set of SD language
  3. Show that *P* is nontrivial:
     - *P* is true of at least one language
       - P is true for {aa, bb}
     - is false of at least one language
       - P is false for {a, aa}

GIVEN A TM *M*, IS *L*(*M*) REGULAR?

The problem: Is $L(M)$ regular?

As a language: Is {<*M*> : $L(M)$ is regular} in D?

No, by Rice's Theorem:

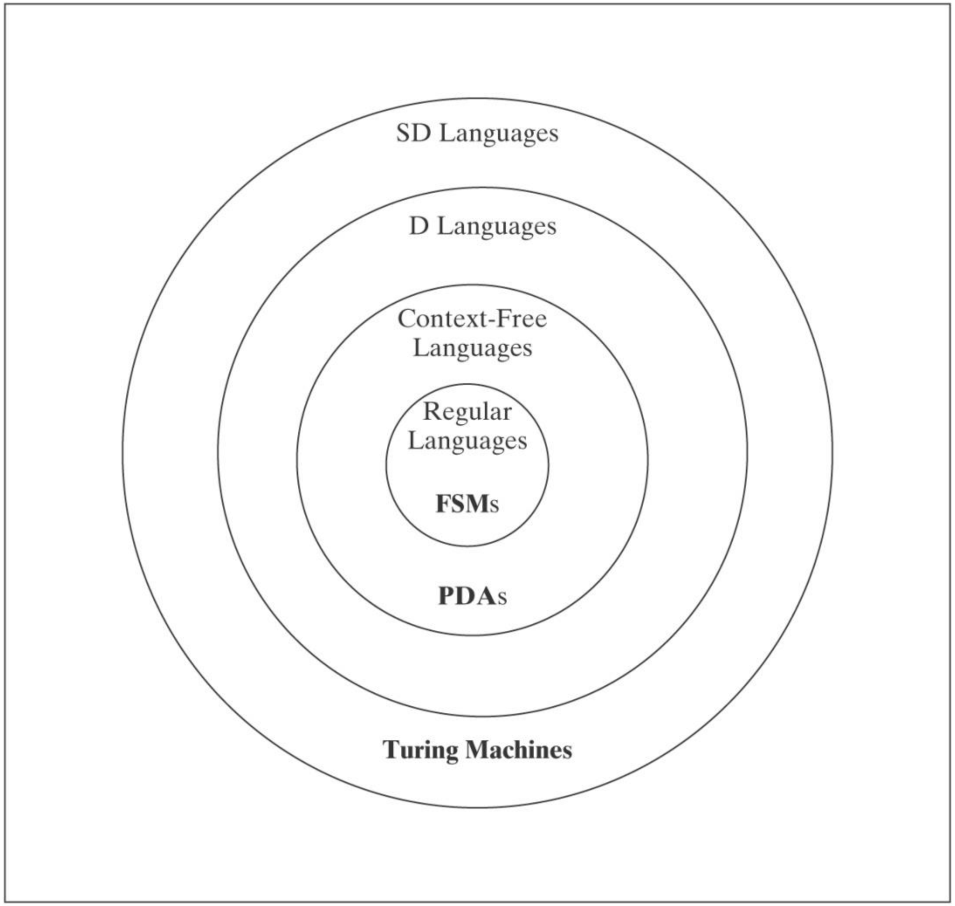
- *P* = *True* if *L* is regular and *False* otherwise.
- The domain of *P* is the set of SD languages since it is the set of languages accepted by some TM.
- *P* is nontrivial:
  - $P(\mathtt{a}^*)$ = *True*.
  - $P(A^nB^n)$ = *False*.

APPLYING RICE'S THEOREM

1. {<*M*> : *L*(*M*) contains only even length strings}.
2. {<*M*> : *L*(*M*) contains an odd number of strings}.
3. {<*M*> : *L*(*M*) contains all strings that start with `a`}.
4. {<*M*> : *L*(*M*) is infinite}.
5. {<*M*> : *L*(*M*) is regular}.
6. {<*M*> : *M* contains an even number of states}.
7. {<M> : M has an odd number of symbols in its tape alphabet}.
8. {<M> : M accepts ε within 100 steps}.
9. {<*M*>: *M* accepts ε}.
10. $\{<M_a, M_b> : L(M_a) = L(M_b)\}$.

NON-SD LANGUAGES

There is an uncountable number of non-SD languages, but only a countably infinite number of TM's (hence SD languages). ∴The class of non-SD languages is <u>much</u> bigger than that of SD languages!



NON-SD LANGUAGES

Intuition: Non-SD languages usually involve either

(i) knowing a TM will infinite loop

(ii) infinite search

(iii) both above

Examples:

- $\neg$H = {<*M*, *w*> : TM *M* does *not* halt on *w*}.
- {<*M*> : *L*(*M*) = Σ^*}.
- {<*M*> : TM *M* halts on nothing}.

PROVING LANGUAGES ARE NOT SD

- Contradiction
- *L* is the complement of an SD/D Language.
- Reduction from a known non-SD language

CONTRADICTION

Theorem: TM_{MIN} = {<*M*>: Turing machine *M* is minimal} is not in SD.

Theorem 1: A language is in SD iff it is Turing enumerable

Theorem 2: There exists a subroutine, ***obtainSelf***, available to any Turing Machine **M**, that constructs ***<M>***, the description of **M**.

CONTRADICTION

Theorem: TM_{MIN} = {<*M*>: Turing machine *M* is minimal} is not in SD.

Proof: If TM_{MIN} were in SD, then there would exist some Turing machine *ENUM* that enumerates its elements. Define the following Turing machine:

M#(*x*) =

1. Invoke *obtainSelf* to produce <*M*#>.
2. Run *ENUM* until it generates the description of some Turing machine *M*′ whose description is longer than |<*M*#>|.
3. Invoke *U* on the string <*M*′, *x*>.

Since TM_{MIN} is infinite, *ENUM* must eventually generate a string that is longer than |<*M*#>|. So *M*# makes it to step 3 and so is equivalent to *M*′ since it simulates *M*′. But, since |<*M*#>| < |<*M*′>|, *M*′ cannot be minimal. Yet it was generated by *ENUM*. Contradiction.

THE COMPLEMENT OF *L* IS IN SD/D

Suppose we want to know whether L is in SD and we know:

- $\neg L$ is in SD, and
- At least one of L or $\neg L$ is not in D.

Then we can conclude that L is not in SD, because, if it were, it would force both itself and its complement into D, which we know cannot be true.

Example:

- $\neg$H (since $\neg(\neg$H) = H is in SD and not in D)

USING COMPLEMENT WITH $H_{\neg ANY}$

Theorem: $H_{\neg ANY}$ = {<*M*> : there does ***not*** exist any string on which TM *M* halts} is not in SD.

Proof: $\neg H_{\neg ANY}$ is H_{ANY} =

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

We already know:

- $\neg H_{\neg ANY}$ (namely H_{ANY}) is in SD.
- $\neg H_{\neg ANY}$ (namely H_{ANY}) is not in D.

So $H_{\neg ANY}$ is not in SD because, if it were, then H_{ANY} would be in D but it isn't.

REDUCTION

Theorem: If there is a reduction R from L_1 to L_2 and L_1 is not SD, then L_2 is not SD.

So, we must:

- Choose a language L_1 that is known not to be in SD.
- Hypothesize the existence of a ***semideciding*** TM *Oracle*.

Note: R may not swap accept for loop.

USING REDUCTION WITH $H_{\neg ANY}$

$H_{\neg ANY}$ = {<*M*> : there does not exist a string on which TM *M* halts}

USING REDUCTION WITH $H_{\neg ANY}$

$\neg H = \{<M, w> : \text{TM } M \text{ does not halt on input string } w\}$

$$\Big\downarrow R$$

(?*Oracle*) $H_{\neg ANY} = \{<M> : \text{there does not exist a string on which TM } M \text{ halts}\}$

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

1. Construct the description <*M#*> of *M#*(*x*):
   1.1. Erase the tape.
   1.2. Write *w* on the tape.
   1.3. Run *M* on *w*.
2. Return <*M#*>.

USING REDUCTION WITH $H_{\neg ANY}$

If *Oracle* exists, then *C* = *Oracle*(*R*(<*M*, *w*>)) semidecides $\neg$H:

- *C* is correct: *M#* ignores its input. It halts on everything or nothing, depending on whether *M* halts on *w*. So:
  - <*M*, *w*> $\in \neg$H: *M* does not halt on *w*, so *M#* halts on nothing. *Oracle* accepts.
  - <*M*, *w*> $\notin \neg$H: *M* halts on *w*, so *M#* halts on everything. *Oracle* does not accept.

But no machine to semidecide $\neg$H can exist, so neither does *Oracle*.

R(<*M*, *w*>) =
1. Construct the description <*M#*> of *M#*(*x*):
   1.1. Erase the tape.
   1.2. Write *w* on the tape.
   1.3. Run *M* on *w*.
2. Return <*M#*>.

$A_{anbn} = \{\langle M \rangle : L(M) = A^nB^n\}$

A_{anbn} contains strings that look like:

(q00,a00,q01,a00,→),
(q00,a01,q00,a10,→),
(q00,a10,q01,a01,←),
(q00,a11,q01,a10,←),
(q01,a00,q00,a01,→),
(q01,a01,q01,a10,→),
(q01,a10,q01,a11,←),
(q01,a11,q11,a01,←)

It does not contain strings like `aaabbb`.

But A^nB^n does.

$A_{anbn} = \{<M> : L(M) = A^nB^n\}$ IS NOT IN SD

$$\neg H = \{<M, w> : \text{TM } M \text{ does not halt on } w\}$$

$$\downarrow R$$

(?*Oracle*) $A_{anbn} = \{<M> : L(M) = A^nB^n\}$

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

1. Construct the description <*M#*>:
   1.1. If $x \in A^nB^n$ then accept. Else:
   1.2. Erase the tape.
   1.3. Write *w* on the tape.
   1.4. Run *M* on *w*.
   1.5. Accept.
2. Return <*M#*>.

A_{anbn} = {<*M*> : *L*(*M*) = A^nB^n} IS NOT IN SD

R(<*M*, *w*>) reduces $\neg$H to A_{anbn}

If *Oracle* exists, then *C* = *Oracle*(*R*(<*M*, *w*>)) semidecides $\neg$H:
M# immediately accepts all strings in A^nB^n. If *M* does not halt on *w*, those are the only strings *M#* accepts. If *M* halts on *w*, *M#* accepts everything:

- <*M*, *w*> $\in \neg$H: *M* does not halt on *w*, so *M#* accepts strings in A^nB^n in step 1.1. Then it gets stuck in step 1.4, so it accepts nothing else. It is an A^nB^n acceptor. L(M#) = A^nB^n. *Oracle* accepts.
- <*M*, *w*> $\notin \neg$H: *M* halts on *w*, so *M#* accepts everything. L(M#) $\neq A^nB^n$. *Oracle* does not accept.

But no machine to semidecide $\neg$H can exist, so neither does *Oracle*.

THE DETAILS MATTER

$L_1 = \{<M>: M$ has an even number of states$\}$.

$L_2 = \{<M>: |<M>|$ is even$\}$.

$L_3 = \{<M>: |L(M)|$ is even$\}$.

$L_4 = \{<M>: M$ accepts all even length strings$\}$.

THE DETAILS MATTER

L_1 = {<M>: M has an even number of states}. ➔ D

L_2 = {<M>: |<M>| is even}. ➔ D

L_3 = {<M>: |L(M)| is even}. ➔ NOT IN D

L_4 = {<M>: M accepts all even length strings}. ➔ NOT IN D

ACCEPTING, REJECTING, HALTING, AND LOOPING

Consider :

$L_1 = \{\langle M, w\rangle: M$ does not halt on $w\}$.

$L_2 = \{\langle M, w\rangle: M$ rejects $w\}$.

$L_3 = \{\langle M, w\rangle: M$ is a deciding TM and rejects $w\}$.

ACCEPTING, REJECTING, HALTING, AND LOOPING

Consider :

L_1 = {<*M*, *w*>: *M* does not halt on *w*}. ➔ NOT IN SD ($\neg$H)

L_2 = {<*M*, *w*>: *M* rejects *w*}. ➔ NOT IN D BUT IN SD

L_3 = {<*M*, *w*>: *M* is a deciding TM and rejects *w*}. ➔ NOT IN SD

WHAT ABOUT THESE?

$L_1 = \{\texttt{a}\}$.

$L_2 = \{<M> : M \text{ accepts } \texttt{a}\}$.

$L_3 = \{<M> : L(M) = \{\texttt{a}\}\}$.

WHAT ABOUT THESE?

$L_1 = \{\mathtt{a}\}$. ➔ OBVIOUSLY IN D!

$L_2 = \{<M> : M \text{ accepts } \mathtt{a}\}$. ➔ IN SD BUT NOT IN D

$L_3 = \{<M> : L(M) = \{\mathtt{a}\}\}$. ➔ NOT IN SD

$\neg H \geq_M L_3$: $R(<M, w>) =$

1. Construct the description $<M\#>$, where $M\#(x)$ operates as follows:
   - 1.1 If $x = \mathtt{a}$, accept.
   - 1.2 Erase the tape.
   - 1.2 Write w on the tape.
   - 1.3 Run M on w.
   - 1.4 Accept.
2. Return $<M\#>$.

- $<M, w> \in \neg H$: M does not halt on w, so M# accepts the string a and nothing else. So L(M#) = {a}. Oracle accepts.
- $<M, w> \notin \neg H$: M halts on w. M# accepts everything. So L(M#) $\neq$ {a}. Oracle does not accept.

But no machine to semidecide $\neg H$ can exist, so neither does Oracle.

The Problem View	***The Language View***	***Status***
Does TM *M* have an even number of states?	{<*M*> : *M* has an even number of states}	D
Does TM *M* halt on *w*?	H = {<*M*, *w*> : *M* halts on *w*}	SD/D
Does TM *M* halt on the empty tape?	H_ε = {<*M*> : *M* halts on ε}	SD/D
Is there any string on which TM *M* halts?	H_{ANY} = {<*M*> : there exists at least one string on which TM *M* halts }	SD/D
Does TM *M* halt on all strings?	H_{ALL} = {<*M*> : *M* halts on Σ^*}	$\neg$SD
Does TM *M* accept *w*?	A = {<*M*, *w*> : *M* accepts *w*}	SD/D
Does TM *M* accept ε?	A_ε = {<*M*> : *M* accepts ε}	SD/D
Is there any string that TM *M* accepts?	A_{ANY} {<*M*> : there exists at least one string that TM *M* accepts }	SD/D

Does TM *M* accept all strings?	$A_{ALL} = \{<M> : L(M) = \Sigma^*\}$	¬SD
Do TMs M_a and M_b accept the same languages?	$EqTMs = \{<M_a, M_b> : L(M_a) = L(M_b)\}$	¬SD

Does TM *M* not halt on any string?	$H_{\neg ANY} = \{<M>$: there does not exist any string on which *M* halts}	¬SD
Does TM *M* not halt on its own description?	$\{<M>$: TM *M* does not halt on input $<M>\}$	¬SD
Is TM *M* minimal?	$TM_{MIN} = \{<M>: M$ is minimal}	¬SD
Is the language that TM *M* accepts regular?	TMreg = $\{<M> : L(M)$ is regular}	¬SD
Does TM *M* accept the language A^nB^n?	$A_{anbn} = \{<M> : L(M) = A^nB^n\}$	¬SD

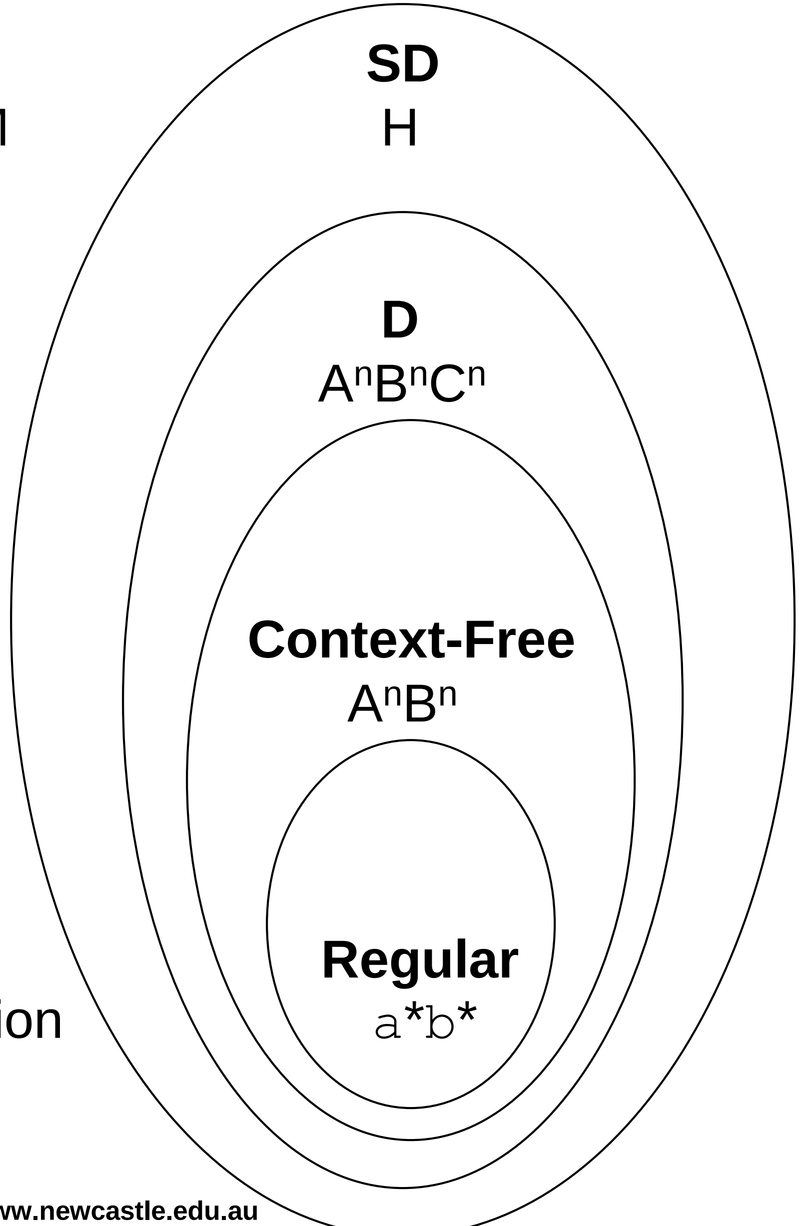
LANGUAGE SUMMARY

IN
Semideciding TM

Deciding TM
L and ¬*L* in SD

CF grammar
PDA
Closure

Regular Expression
FSM



OUT
Reduction

Diagonalise
Reduction

Pumping
Closure

Pumping
Closure

References

- **Automata, Computability and Complexity. Theory and Applications**

  - By Elaine Rich
- Chapter 21:

  - Page : 468-482.