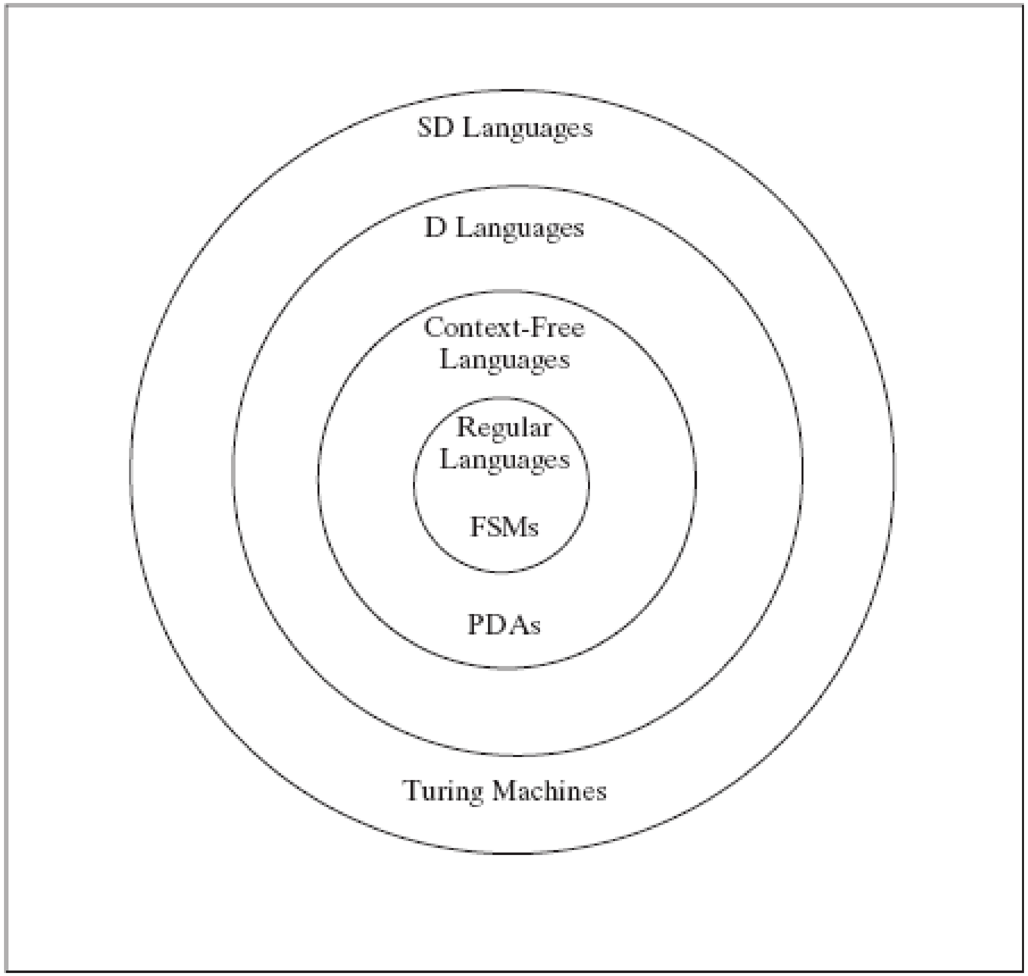
Theory of Computation
Week 12
Review

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

THE HIERARCHY



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

GENERAL DEFINITIONS

- An **alphabet** (Σ) is a finite set of symbols (or characters)
- A **string** is a finite sequence of symbols chosen from some alphabet Σ
- A **language** is a set (finite or infinite) of strings chosen from some finite alphabet Σ
- A ***decision problem*** is simply a problem for which the answer is yes or no (True or False). A ***decision procedure*** answers a decision problem.

Closure

- A binary relation *R* on a set *A* is ***closed under*** property *P* if and only if *R* ***possesses*** *P*.

  Examples

  < on the integers, *P* = transitivity

  ≤ on the integers, *P* = reflexive

- The ***closure*** of *R* under *P* is a smallest set that includes *R* and that is closed under *P*.

DECISION PROBLEMS

What If We're Not Working with Strings?

Anything can be encoded as a string.

<*X*> is the string encoding of *X*.

<*X*, *Y*> is the string encoding of the pair *X*, *Y*.

DECISION PROBLEMS

- Problem: Verify the correctness of the addition of two numbers.
- Encoding: encode each of the numbers as a string of decimal digits. Each instance of the problem is a string of the form:
  $<integer_1> + <integer_2> = <integer_3>$
- The language to be decided:

  INTEGERSUM = {w of the from: $<integer_1> + <integer_2> = <integer_3>$*:* each of the substrings $<integer_1>$, $<integer_2>$ and $<integer_3>$ is an element of $\{0,1,2,3,4,5,6,7,8,9\}^+$ and $integer_3$ is sum of $integer_1$ and $integer_2$}.

DECISION PROBLEMS

Turning Problems into Decision Problems

The Traditional Problems and their Language Formulations are Equivalent

By equivalent we mean that either problem can be ***reduced to*** the other.

That is: if we have a ***machine*** to solve one, we can use it to build a machine to do the other using just the starting machine and other functions that can be built using a machine of equal or lesser power.

Equivalence of Decision Problem

Suppose we have a program P that multiplies a pair of integers. Then the following program decides the language INTEGERMUL where INTEGERMUL={*w* of the form:<$integer_1$>$\times$<$integer_2$>= <$integer_3$>, where: <$integer_n$> is any well formed integer, and $integer_3 = integer_1 * integer_2$}

Given a string of the form <$integer_1$> $\times$ <$integer_2$>=<$integer_3$>

1. Let x = convert-to-integer(<$integer_1$>).
2. Let y = convert-to-integer(<$integer_2$>).
3. Let z = P(x,y)
4. If z = convert-to-integer(<$integer_3$>) then accept Else reject.

Equivalence of Decision Problem

Alternatively, if we have a program T that decides the language INTEGERMUL then the following program computes the sum of two integers x and y:

1. Lexicographically enumerate the strings that represent decimal encodings of nonnegative integers.
2. Each time a string s is generated, create the new string <x> × <y> = s.
3. Feed the string to T.
4. If T accepts <x> × <y> = s, halt and return conver-to-integers(s).

DETERMINISTIC FSM
Definition

- A Finite State Machine M is a quintuple

  $M = (K, \Sigma, \delta, s, A)$, where:

  - K is a finite set of states
  - Σ is an alphabet
  - δ is the transition function from $(K \times \Sigma)$ to K
  - $s \in K$ is the initial state, and
  - $A \subseteq K$ is the set of accepting states

NONDETERMINISTIC FSM
Definition

- A Finite State Machine M is a quintuple

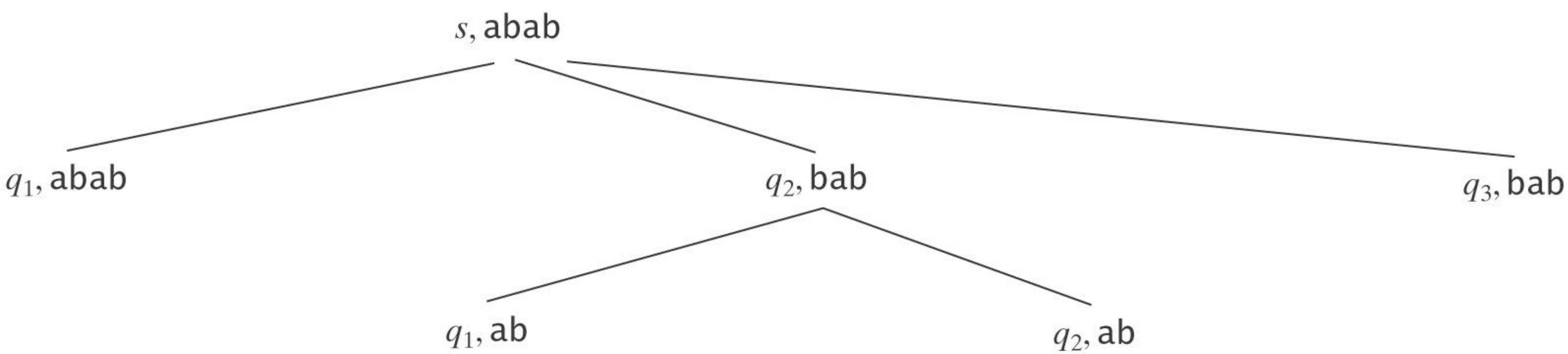
  $M = (K, \Sigma, \Delta, s, A)$, where:

  - K is a finite set of states
  - Σ is an alphabet
  - Δ is the transition relation. It is a finite subset of $(K \times (\Sigma \cup \{\varepsilon\}) \times K$
  - $s \in K$ is the initial state, and
  - $A \subseteq K$ is the set of accepting states

NONDETERMINISTIC FSMs
Analysing Nondeterminism

Two approaches:

- Explore a search tree:



- Follow all paths in parallel

REGULAR EXPRESSION
Definition

The regular expressions over an alphabet Σ are all and only the strings that can be obtained as follows:

1. $\varnothing$ is a regular expression.
2. ε is a regular expression.
3. Every element of Σ is a regular expression.
4. If α , β are regular expressions, then so is $\alpha\beta$.
5. If α , β are regular expressions, then so is $\alpha\cup\beta$.
6. If α is a regular expression, then so is α^*.
7. α is a regular expression, then so is α^+.
8. If α is a regular expression, then so is (α).

REGULAR GRAMMAR
Definition

A ***regular grammar*** G is a quadruple (V, Σ, R, S), where:

- V is the rule alphabet, which contains nonterminals and terminals,
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite set of rules of the form:

$$X \rightarrow Y,$$

- S (the start symbol) is a nonterminal.

REGULAR LANGUAGES
Summary Of Concepts

A language is ***regular*** iff it is accepted by some FSM

- Given any DFSM *M*, there exists an algorithm *minDFSM* that constructs a minimal DFSM that also accepts *L*(*M*).
- Given any NDFSM *M*, there exists an algorithm *ndfsmtodfsm* that constructs a DFSM that also accepts *L*(*M*).

REGULAR LANGUAGES
Summary Of Concepts

- Given any DFSM *M*, there exists an algorithm *fsmtoregex* that constructs a regular expression that recognises *L*(*M*).

- Given any grammar *G*, there exists an algorithm *grammartofsm* that constructs a DFSM that also accepts *L*(*G*).

- The class of languages that can be defined with regular grammars, DFSMs, NFSMs and regular expressions is exactly the regular languages.

CLOSURE PROPERTIES OF REGULAR LANGUAGES

- Union
- Concatenation
- Kleene star
- Complement
- Intersection
- Difference
- Reverse
- Letter substitution

DON'T TRY TO USE CLOSURE BACKWARDS

One Closure Theorem:

If L_1 and L_2 are regular, then so is

$$L = \underline{L_1} \cap \underline{L_2}$$

But if L is regular, what can we say about L_1 and L_2?

$$\underline{\underline{L}} = L_1 \cap L_2$$

THE PUMPING THEOREM FOR REGULAR LANGUAGES

If L is regular, then every “long” string in L is pumpable.

To show that L is not regular, we find one that isn’t.

To use the Pumping Theorem to show that a language L is not regular, we must:

1. Choose a string w where $|w| \geq k$. Since we do not know what k is, we must state w in terms of k.
2. Divide the possibilities for y into a set of equivalence classes that can be considered together.
3. For each such class of possible y values where $|xy| \leq k$ and $y \neq \varepsilon$:

   Choose a value for q such that xy^qz is not in L.

CONTEXT-FREE GRAMMAR
Definition

A context-free grammar G is a quadruple, (V, Σ, R, S), where:

- V is the rule alphabet, which contains nonterminals and terminals.
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite subset of $(V - \Sigma) \times V^*$,
- S (the start symbol) is an element of $V - \Sigma$.

Example:

$(\{S, \mathtt{a}, \mathtt{b}\}, \ \{\mathtt{a}, \mathtt{b}\}, \ \{S \to \mathtt{a}\ S\ \mathtt{b}, S \to \varepsilon\}, \ S)$

CONTEXT-FREE LANGUAGES

A language *L* is ***context-free*** if and only if it is generated by some context-free grammar *G*.

PARSE TREES
Definition

A parse tree, derived by a grammar $G = (V, \Sigma, R, S)$, is a rooted, ordered tree in which:

- Every leaf node is labeled with an element of $\Sigma \cup \{\varepsilon\}$,
- The root node is labeled S,
- Every other node is labeled with some element of:
  $V - \Sigma$, and
- If m is a nonleaf node labeled X and the children of m are labeled $x_1, x_2, \ldots, x_n$, then R contains the rule
-   $X \rightarrow x_1, x_2, \ldots, x_n$.

AMBIGUITY

A grammar is ***ambiguous*** iff there is at least one string in $L(G)$ for which G produces more than one parse tree.

For most applications of context-free grammars, this is a problem.

INHERENT AMBIGUITY

Both of the following problems are undecidable:

- Given a context-free grammar *G*, is *G* ambiguous?
- Given a context-free language *L*, is *L* inherently ambiguous?

REDUCING AMBIGUITY

We can get rid of:

- ε rules like $S \to \varepsilon$,

- rules with symmetric right-hand sides, e.g.,

  $S \to SS$

  $E \to E + E$

- rule sets that lead to ambiguous attachment of optional postfixes.

CONVERSION TO CHOMSKY NORMAL FORM

1. Remove all ε-rules, using the algorithm *removeEps*.

2. Remove all unit productions (rules of the form $A \rightarrow B$).

3. Remove all rules whose right hand sides have length greater than 1 and include a terminal:

   (e.g., $A \rightarrow \mathtt{a}B$ or $A \rightarrow B\mathtt{a}C$)

4. Remove all rules whose right hand sides have length greater than 2:

   (e.g., $A \rightarrow BCDE$)

PUSHDOWN AUTOMATON
Definition

A ***pushdown automaton*** is a 6-tuple $M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

- K is a finite set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $s \in K$ is the initial state
- $A \subseteq K$ is the set of accepting states, and
- Δ is the transition relation.

PUSHDOWN AUTOMATON
Definition

Δ, the transition relation, is a finite subset of

$$\underbrace{(K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*)}_{} \times \underbrace{(K \times \Gamma^*)}_{}$$

state	input or ε	string of symbols to pop from top of stack	state	string of symbols to push on top of stack

PDAs AND CONTEXT-FREE GRAMMARS

Theorem: The class of languages accepted by PDAs is exactly the class of context-free languages.

Recall: context-free languages are languages that can be defined with context-free grammars.

Restate theorem:

Can describe with context-free grammar

$=$

Can accept by PDA

PDAs AND CONTEXT-FREE GRAMMARS
From CFG to PDA

Lemma*:* Each context-free language is accepted by some PDA.

Proof (by construction):

The idea: Let the stack do the work.

Two approaches:

- Top down
- Bottom up

CLOSURE PROPERTIES OF CONTEXT-FREE LANGUAGES

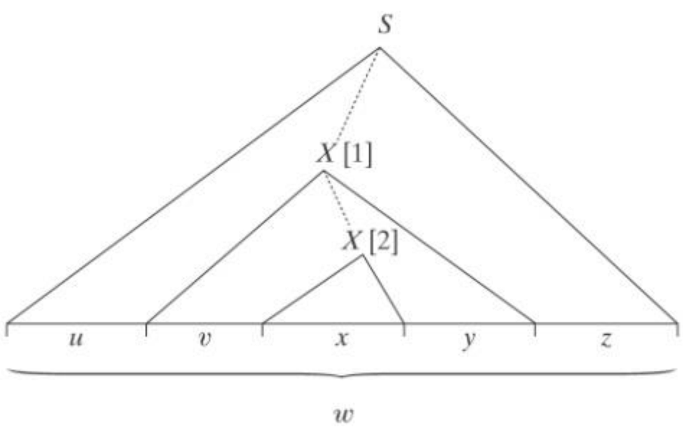
The context-free languages are:

CLOSED	NOT CLOSED
Union	Intersection
Concatenation	Complement
Kleene star	Difference
Reverse	
Letter substitution	

THE PUMPING THEOREM FOR CONTEXT-FREE LANGUAGES

If L is a context-free language, then $\exists k \geq 1$, such that

$\forall$ strings $w \in L$, where $|w| \geq k$, $\exists$ u, v, x, y, z, such that:
$w = uvxyz$, and $vy \neq \varepsilon$, and $|vxy| \leq k$, and
$\forall q \geq 0$, $uv^q xy^q z$ is in L.



TURING MACHINES

A Turing machine *M* is a sixtuple (K, Σ, Γ, δ, s, H):

- *K* is a finite set of states;
- Σ is the input alphabet, which does not contain ❑;
- is the tape alphabet, which must contain ❑ and have Σ as a subset.
- $s \in K$ is the initial state;
- $H \subseteq K$ is the set of halting states;
- δ is the transition function:

$(K - H)$	$\times$	Γ	to	K	$\times$	Γ	$\times$	$\{\rightarrow, \leftarrow\}$
non-halting state	$\times$	tape char		state	$\times$	tape char	$\times$	action (R or L)

SHORTHANDS

$\mathrm{L}_{\mathtt{a}}$

Find the first occurrence of `a` to the left of the current square.

$\mathrm{R}_{\mathtt{a,b}}$

Find the first occurrence of `a` or `b` to the right of the current square.

$$\begin{tikzcd} \mathrm{L}_{\mathtt{a,b}} \arrow[r, "\mathtt{a}"] \arrow[d, "\mathtt{b}"] & M_1 \\ M_2 & \end{tikzcd}$$

Find the first occurrence of `a` or `b` to the left of the current square, then go to M_1 if the detected character is `a`; go to M_2 if the detected character is `b`.

$\mathrm{L}_{x\leftarrow\mathtt{a,b}}$

Find the first occurrence of `a` or `b` to the left of the current square and set *x* to the value found.

$\mathrm{L}_{x\leftarrow\mathtt{a,b}}\mathrm{R}x$

Find the first occurrence of `a` or `b` to the left of the current square, set *x* to the value found, move one square to the right, and write *x* (`a` or `b`).

COMPUTING FUNCTIONS

Let $M = (K, \Sigma, \Gamma, \delta, s, \{h\})$. Its initial configuration is $(s, \underline{\square}w)$.

Define $M(w) = z$ iff $(s, \underline{\square}w) \vdash_M^* (h, \underline{\square}z)$.

Let $\Sigma' \subseteq \Sigma$ be M's output alphabet.
Let f be any function from Σ^* to Σ'^*.

M ***computes*** *f* iff, for all $w \in \Sigma^*$:

- If w is an input on which f is defined: $M(w) = f(w)$.
- Otherwise $M(w)$ does not halt.

A function f is ***recursive*** or ***computable*** iff there is a Turing machine M that computes it and that always halts.

IMPACT OF NONDETERMINISM

- FSMs
  - Power — NO
  - Complexity
    - Time — NO
    - Space — YES
- PDAs
  - Power — YES
- Turing machines
  - Power — NO
  - Complexity — ?

CHURCH'S THESIS (CHURCH-TURING THESIS)

All formalisms powerful enough to describe everything we think of as a computational algorithm are equivalent.

This isn’t a formal statement, so we can’t prove it. But many different computational models have been proposed and they all turn out to be equivalent.

DECIDABLE LANGUAGES

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***.

SEMIDECIDABLE LANGUAGES

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.

Theorems about D and SD

Theorem: The set of context-free languages is a *proper* subset of D.
Theorem: There are languages that are not in SD.
Theorem: The set D is closed under complement.
Theorem: The set SD is not closed under complement.
Theorem: A language is in D iff both it and its complement are in SD.

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.

COMPLEMENT OF THE HALTING PROBLEM

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

Is not in SD

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.

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

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

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

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

$$\Big\downarrow R$$

(?*Oracle*) $$H_\varepsilon \ \{<M> : \text{TM } M \text{ halts on } \varepsilon\}$$

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*}

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.

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.

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*.

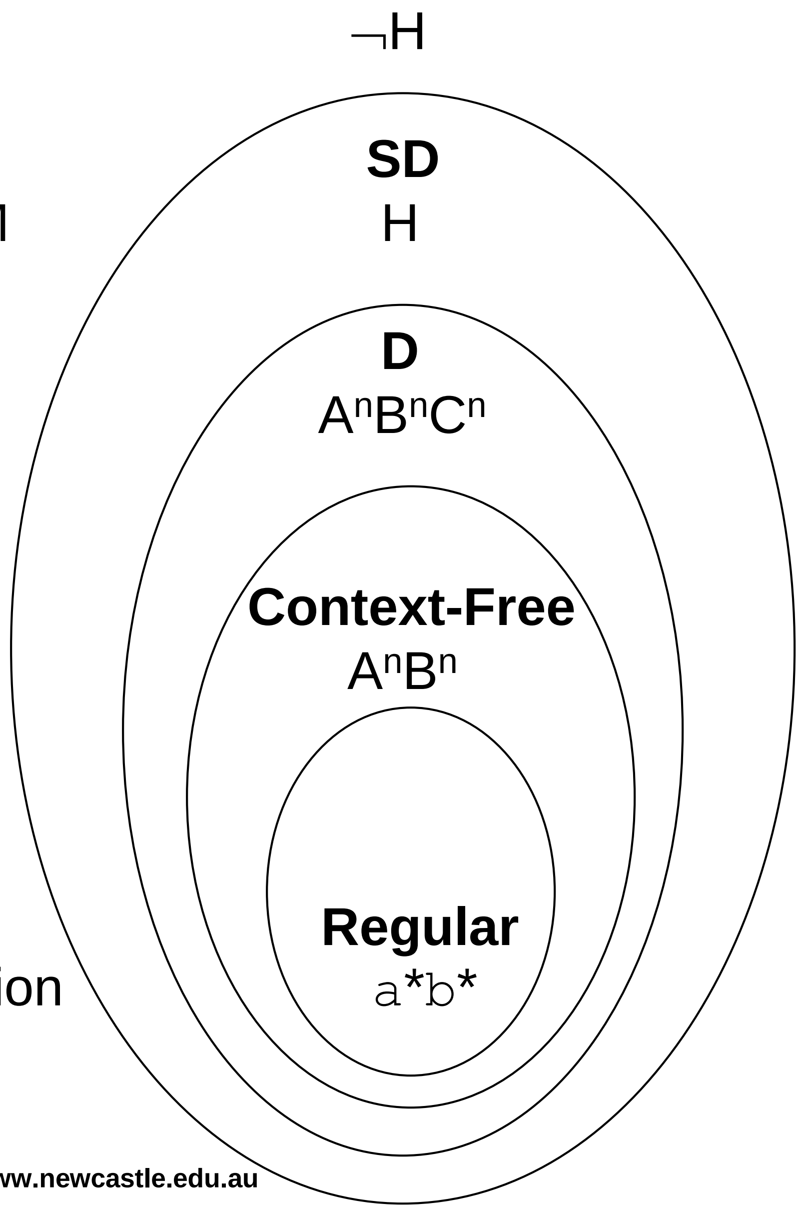
Note: R may not swap accept for loop.

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*}	¬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

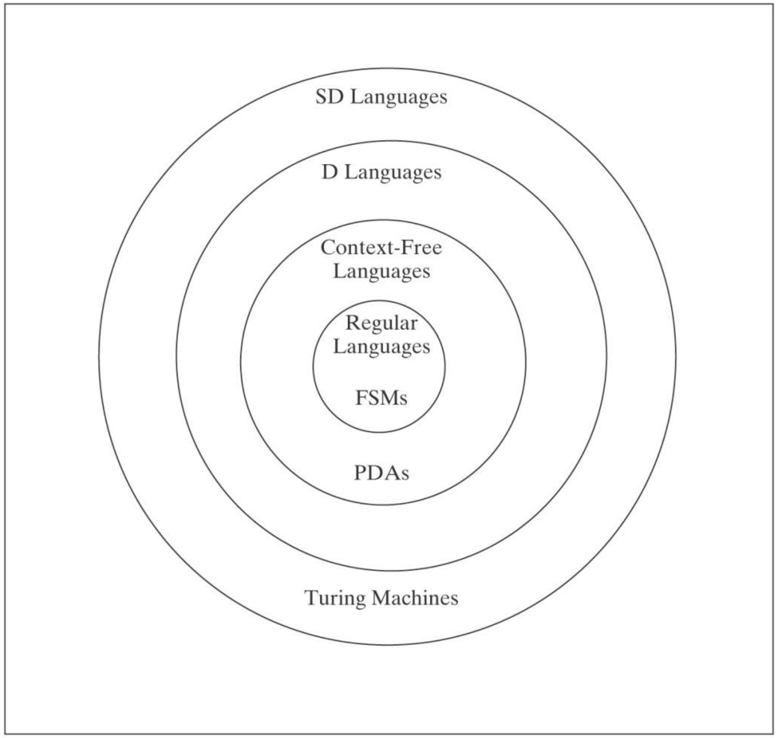
Pumping

Closure

Pumping

Closure

LANGUAGES AND MACHINES



Rule of Least Power: “Use the least powerful language suitable for expressing information, constraints or programs on the World Wide Web.”