1 Leon's (WIP) ITCS Exam Notes

Basically adapted from Chris Dalziel's notes :)
In collaboration with Alex Brodbelt :)

Finite Automata

Definition: Finite Automata

A finite automaton takes a string as input and replies "yes" or "no". If an automaton A replies "yes" on a string S we say that A "accepts" S.

Definition: Deterministic Finite Automata

A deterministic finite automaton (DFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where

- Q is a finite set of states
- Σ is an alphabet
- $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $F \subseteq Q$ is the set of final states

A DFA accepts a string $w \in \Sigma^*$ iff $\delta^*(q_0, w) \in F$, where δ^* is δ applied successively for each symbol in w.
The language of a DFA A is the set of all strings accepted by a, $\mathcal{L} \subseteq \Sigma^*$ is the set of all strings accepted by A.
The transition function is a total function which gives exactly one next state for each input symbol, i.e. it is deterministic

Definition: Nondeterministic Finite Automata

Non-determinism would mean that δ can return more than one successor state, it instead returns a set of possible states - no states is an empty set. A NFA is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a finite set of states
- Σ is an alphabet
- $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function
- $F \subseteq Q$ is the set of final states

The only difference between the definition of a DFA and that of an NFA is that in an NFA δ returns an element from the powerset of Q, $\mathcal{P}(Q)$

Adding non-determinism doesn't change "expressivity". Given an NFA A there is an equivalent DFA D such that $\mathcal{L}(D) = \mathcal{L}(A)$ and vice versa.

Definition: ϵ-NFA

If we allow non-deterministic state chnages that don't consume any input symbols, we can label silent moves using ϵ - meaning the empty string We define the ϵ closure $E(q)$ of a state q as the set of all states reachable from q by silent moves. That is, $E(q)$ is the least set satisfying:

- $q \in E(q)$
- For any $s \in E(q)$ we also have $\delta(s, \epsilon) \subseteq E(q)$

DFA, NFA, ϵ-NFA are all equal in expressive power

Regular Languages

Definition: Regular Languages

Any language which can be accepted by a finite automaton is called a regular language.
Regular languages are also those recognised by Regular Expressions

Definition: Regular Language Closure Properties

For two languages L_1 and L_2, the following operations satisfy the closure property, i.e. for a member $x \in X$, and an operation ϕ we have that $\phi(x) \in \mathbb{R}$ for all x.

- **Union**: $L_1 \cup L_2$ is the language that includes all strings of L_1 and all strings of L_2.
- **Intersection**: $L_1 \cap L_2$ is the language that includes all strings of L_1 that are not in L_2, and vice versa
- **Sequential Composition**: $L_1 L_2$ is the language of strings that consist of strings in L_1 followed by a string in L_2.
- **Kleene closure**: L^* is the language of strings that consist wholly of zero or more strings in L.

$$L^* = \bigcup_{i \in \mathbb{N}} L^i$$

- **Complement**: $\bar{L}$ is the language of every string not in L.

Definition: Regular Expressions

Regular characterise the regular languages, just like finite automata do. The following table shows the syntax and semantics of a regex.

Syntax	Semantics	
a	$[\![a]\!] = \{a\}$	$(a \in \Sigma)$
$\emptyset$	$[\![\emptyset]\!] = \emptyset$	
ϵ	$[\![\epsilon]\!] = \{\epsilon\}$	
$R_1 \cup R_2$	$[\![R_1 \cup R_2]\!] = [\![R_1]\!] \cup [\![R_2]\!]$	
$R_1 \circ R_2$	$[\![R_1 \circ R_2]\!] = [\![R_1]\!][\![R_2]\!]$	
R^*	$[\![R^*]\!] = [\![R]\!]^*$	

Definition: Generalised NFAs

A **generalised NFA**, or GNFA is an NFA where:

- Transitions have **regular expressions** on them instead of symbols
- There is only one unique final state
- The transition relation if **full**, except that the initial state has no incoming transitions, and the final state has no outgoing transitions

Theorem 1: Pumping Lemma

If $L \subseteq \Sigma^*$ is regular, then there is a **pumping length** $p \in \mathbb{N}$ such that for any $w \in L$ where $|w| \geq p$, we may split w into three piexes $w = xyz$ satisfying three conditions:

- $xy^iz, \quad \forall i \in \mathbb{N}$
- $|y| > 0$
- $|xy| \leq p$

Note that if the pumping lemma fails then the language is not regular, but the inverse is not necessarily true.

Theorem 2: Myhill-Nerode Theorem

Let $L \subseteq \Sigma^*$ and $x, y \in \Sigma^*$. If there exists a suffix string z such that $xz \in L$, but $yz \notin L$ or vice versa, then x and y are **distinguishable** by L. If x and y are not distinguishable by L, then we say that $x \equiv_L y$ - this is an equivalence relation. A regular language satisfies the following

- The number of equivalence classes $\equiv_L$ is finite.
- The number of equivalence classes is equal to the number of states in the minimal DFA accepting L (not as important)

Therefore, to show a language is non-regular, show that it has infinite equivalence classes - that is, we find an infinite sequence $u_0 u_1 \ldots$ of strings such that for any i, j where $i \neq j$, there is a string w_{ij} such that $u_i w_{ij} \in L$ but $u_j w_{ij} \notin L$ or vice-versa

Context-Free Languages

Definition: Context-free Languages

By adding recursion to regexs we can begin to recognise some non-regular languages. All regular languages are also context free.

Definition: Context-free Grammars

A language is context-free iff it is recognised by a Context-free Grammar (CFG), which is a 4-tuple (N, Σ, P, S) where:

- N is a finite set of variables or non-terminals
- Σ is a finite set of terminals
- $P \subseteq N \times (N \cup \Sigma)^*$ is a finite set of rules or productions
  - Typically productions are written $A \to aBc$
  - Productions with common heads can be combined, $A \to a$ and $A \to Aa$ can be combined into $A \to a \mid Aa$
- $S \in N$ is the starting variable

We use α, β, γ to refer to sequences of terminals
We make a derivation step $\alpha A \beta \Rightarrow_G \alpha\gamma\beta$ whenever $(A \to \gamma) \in P$;
The language of a CFG G is:

$$\mathcal{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w^*\}$$

Where $\Rightarrow_G^*$ is the reflexive, transitive, closure of $\Rightarrow_G$.
Context-free grammars are ambiguous. They are closed under union, concatenation, and kleene star, but not under intersection or complementation

Definition: Eliminating Ambiguity

We want to eliminate ambiguity in CFGs while still accepting all the same strings. This can be done for our language of regular expressions:

- First defining atomic expressions: $A \to (S)|\emptyset|\epsilon|a|b$
- Then ones which use Kleene Star: $K \to A|A^*$
- Then ones which may use left-associative composition: $C \to K|C \circ K$
- Finally expressions which use unions: $S \to C|S \cup C$

The order of operations here is therefore bottom to top; unions come before compositions, which come before Kleene etc

Definition: Push-down Automata

Push-down automata are to CFGs what finite automatas are to regular expressions. They are implementationally identical to ϵ-NFAs with the addition of a stack. The recursive element of CFGs is implemented using a standard last-in-first-out stack.

Transitions in a push-down automata take the form $x, y \to z$ which is read as "consume the input x, popping y off the stack, and push z onto the stack". We can allow actions that don't consume, pop, or push by setting variables to ϵ.

Definition: Formal Def. of PDAs

A **push-down automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ are all finite sets. Γ is the stack alphabet, and δ now may take a stack symbol as input or return one as output:

$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$$

All other compoments are as with ϵ-NFAs

A string w is accepted by a PDA if it ends in a final state, i.e. $\delta^*(q_0, w, \epsilon)$ gives a state q and a stack γ wuch that $q \in F$.

Theorem 3: CFG to PDA

A language is context-free if and only if it is recognised by a push-down automaton. The proof is left as an exercise to the reader.

Theorem 4: Pumping CFLs intro

Suppose a CFG has n non-terminals, and we have a parse tree of height $k > n$. Then the same non-terminal V must have appeared as its own descendant in the tree

- **Pumping down**: Cut the tree at the higher occurance of V and replace it with the subtree at the lower occurance of V
- **Pumping up**: Cut at the lower occurance and replace it with a fresh copy of the higher occurance

Theorem 5: Pumping Lemma for CFLs

If L is context-free then there exists a pumping length $p \in \mathbb{N}$ such that if $w \in L$ with $|w| \geq p$ then w may be split into **five** pieces $w = uvxyz$ such that

- $uv^ixy^iz \in L$ for all $i \in \mathbb{N}$
- $|vy| > 0$
- $|vxy| \leq p$

Definition: Chomsky Grammars

Context-free grammars are a special case of Chomsky Grammars. Chomsky grammars are similar to CFGs, except that the left-hand side of a production may be any string that includes at least one non-terminal. An example is shown below

$$\begin{aligned} S &\to abc \mid aAbc \\ Ab &\to bA \\ Ac &\to Bbcc \\ bB &\to Bb \\ aB &\to aaA \mid aa \end{aligned}$$

Such a grammar is called **context-sensitive**

Definition: The Chomsky Heirarchy

A grammar $G = (N, \Sigma, P, S)$ is of type:

0. (or **computably enumerable**) in the general case
1. (or **context sensitive**) if $|\alpha| \leq |\beta|$ for all productions $\alpha \to \beta$, except we also allow $S \to \epsilon$ if S foes not occur on the RHS of any rule
2. (or **context free**) if all productions are of the form $A \to \alpha$ (i.e. a CFG)
3. (or **right-linear/regular**) if all productions are of the form $A \to w$ or $A \to wB$, where $w \in \Sigma$ and $B \in N$

Algorithms for Languages

Theorem 6: Emptiness for Regular Languages

Can we write a program to determine if a given regular language is empty?
Given a finite-automaton this is an instance of graph reach-ability, so we can use a depth-first search.

Theorem 7: Emptiness for Context-free languages

Can we write a program to determine if a given context-free language is empty?
Given a CFG for our language, we can perform the following process:

1. Mark the terminals and ϵ as generating
2. Mark all non-terminals which have a production with only generating symbols in their right hand side as generating
3. Repeat until nothing new is marked
4. Check if S is marked as generating or not

Theorem 8: Equivalence of DFA

Is it possible to write a program to determine if two discrete finite automata are equivalent?

Given two DFA for L_1 and L_2, we can use our standard constructions to produce a DFA of the symmetric set difference:

$$(L_1 \cap \overline{L}_2) \cup (L_2 \cap \overline{L}_1)$$

Register Machines

Definition: Register Machines

A **register machine**, or RM, consists of:

- A fixed number m of registers $R_0 \dots R_{m-1}$, which each holds a natural number
- A fixed program P which is a sequence of n instructions $I_0 \dots I_{n-1}$

Each instruction is one of the following:

- `INC(i)`: which increments the register R_i by one
- `DECJZ(i, j)`: which decrements register R_i unless $R_i = 0$ in which case it jumps to instruction I_j

RMs can compute anything any other computer can

Definition: Pairing Functions for RMs

A **pairing function** is an injective function $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. An example is $f(x, y) = 2^x 3^y$.
We write $\langle x, y\rangle_2$ for $f(x, y)$. If $z = \langle x, y\rangle_2$, let $z_0 = x$ and $z_1 = y$.
This lets us encode multiple values into a single value, and a 2-tuple pairing function is enough to cram an arbitrary sequence of natural numbers into one $\mathbb{N}^* \to \mathbb{N}$

Definition: Turing Machine

A **turing machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

- Q: states
- Σ: input symbols
- $\Gamma \subseteq \Sigma$: **tape** symbols, including a **blank** symbol $\sqcup$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 1\}$
- $q_0, q_{\text{accept}}, q_{\text{reject}} \in Q$: start, accept, reject states

Theorem 9: Church-Turing Thesis

The Church-Turing thesis states that any problem is computable by any model of computation iff it is computable by a **Turing machine**.
For our purposes this matters for RMs, TMs, and λ-calculus.
Other examples are combinator calculus, general recursive functions, pointer machines, counter machines, cellular automata, queue automata, enzyme-based DNA computers, Minecraft, Magic the Gathering, and others.

Decidability

Definition: Problems with no Algorithm

While the above problems are all computable, this is not always the case. The questions asked, i.e. "can we determine x?", are not always something we can answer using a program - if this is the case then that problem is considered undecidable.
Many such problems exist for Context-free Languages

- Are two CFG equivalent?
- Is a given CFG ambiguous?
- Is there a way to make a given CFG unambiguous?
- Is the intersection of two CFLs empty?
- Does a CFG generate all strings Σ^*?
- ...

Theorem 10: The Halting Problem

Given a register machine encoding, can we write a program to determine if the simulated machine will halt or not?
If we suppose H is such a register machine, which takes a machine encoding $\lceil M \rceil$ in R_0, halts with 1 if M halts, and halts with 0 if M doesn't halt.
We can construct a new machine $L = (P_L, R_0, \dots)$ which, given a program $\lceil P \rceil$ runs H on the program with itself as input, the machine $(P, \lceil P \rceil)$ and loops if it halts.
If we run L on P_L itself we get a problem. If L halts on $\lceil P_L \rceil$ that means H says $(P_L, \lceil P_L \rceil)$. If L loops on $\lceil P_L \rceil$ that means that H says $(P_L, \lceil P_L \rceil)$ halts.
This is a contradiction!
The halting problem proves that there are some programs which cannot be decided by register machines. What about other machines?

Definition: Computability

A (total) function $\mathbb{N} \to \mathbb{N}$ is **computable** if there is an RM/TM which computes f, i.e., given an x in R_0, leaves $f(x)$ in R_0
A **decision problem** is a set D and a quiery subset $Q \subseteq D$. A problem is **decidable** or **computable** if $d \in Q$ is characterised by a computable function $f : D \to \{0, 1\}$.

Definition: Reductions

A **reduction** is a transformation from one problem to another.
To prove that a problem P_2 is hard, show that there is an easy reduction from a known hard problem P_1 to P_2.
To show a problem P_2 is undecidable, show that there is a computable reduction from a known undecidable P_1 to P_2. The direction here matters - it tells us nothing to know that there's an easy way to make an easy problem difficult!

Example : Reduction analogy

If it is a well known fact that Hyunwoo cannot lift a car, we can prove that Hyunwoo cannot lift a loaded truck by making the following reduction: If we suppose he could lift the loaded truck, then we could have him lift the car by putting it in the loaded truck - but we know he cannot lift a car.

Definition: Mapping Reductions

A **Turing transducer** is a RM (or TM) which takes an instance d of a problem $P_1 = (D_1, Q_1)$ in R_0 and halts with an instance $d' = f(d)$ of $P_2 = (D_2, Q_2)$ in R_0. Thus, f is a computable function $D_1 \to D_2$
A **mapping reduction** (or a many-to-one reduction) from P_1 to P_2 is a turing transducer f such that $d \in Q_1$ iff $f(d) \in Q_2$
If A is mapping reducible to B, and A is undecidable, then B is undecidable.

Definition: Turing Reductions

A **Turing reduction** from P_1 to P_2 is an RM/TM equipped with an oracle for P_2 which solves P_1.
Decidability results carry across during Turing reductions as with mapping reductions, but mapping reductions make finer distinctions of computing power.

Theorem 11: Rice's Theorem

- A **property** is a set of RM (or TM) descriptions
- A property is **non-trivial** if it contains some but not all descriptions
- A property P is **semantic** if
$$\mathcal{L}(M_1) = \mathcal{L}(M_2) \Rightarrow (\lceil M_1 \rceil \in P \Leftrightarrow \lceil M_2 \rceil \in P)$$
In other words, it concerns the **language** and not the particular implentation of the language

---

Rice's Theorem - All non-trivial semantic properties are undecidable.

Theorem 12: Rice's Theorem part 2

Rice's theorem is useful for deciding propertiese like whether a language is empty, non-empty, regular, context-free etc
It cannot be applied to questions like whether a TM has fewer than 7 states, a final state, a start state, etc. These properties are of machines and not languages
It also doesn't apply to questions like is a language a subset of Σ^* or whether a language of a RM is a language of a TM - these are trivial properties! The consequences of this is that we cannot write programs which answer non-trivial questions about the black-box behaviours of programs

Computability in depth

Definition: Semi-decidability

A problem (D, Q) is **semi-decidable** if there is a TM/RM that returns "yes" for any $d \in Q$, but may return "no" or loop forever when $d \notin Q$

A problem (D, Q) is **co-semi-decidable** if ther eis a TM/RM that returns "no" for any $d \notin Q$, but may return "yes" or loop forever when $d \in Q$

Definition: Enumerable Computability

A set S is **enumerable** if there is a bijection between S and $\mathbb{N}$
A set S is called **computably enumerable** (or c.e.) if the enumeration function $f : \mathbb{N} \to S$ is computable
In terms of RM and TM we can think of enumeration as outputting an infinite list as it executes forever.

Theorem 13: Decidability theorems

Any problem that is both semi-decidable and co-semi-decidable is decidable

---

If a problem P is semi-decidable then its complement $\overline{P}$ is co-semi-decidable, and vice versa

---

All semi-decidable problems are computably enumerable, and any computably enumerable problem is semi-decidable - being semi-decidable is the same as being computably enumerable.

Theorem 14: Decidability Reductions

To prove that a problem P_2 is not c.e. we show that there is a mapping reduction from a known not-c.e. problem P_1 to P_2. We must use mapping reductions, as H is c.e. but L is not - but L is Turing reducible to H by flipping the answer.

Time Complexity

Definition: Time Complexity

The **time complexity** of a (deterministic) machine M that halts on all inputs is a functino $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps that M uses on any input of size n.

Definition: Complexity Measures

When performing addition in a TM we have a time complexity of $\mathcal{O}(\log n)$, where it is $\mathcal{O}(n)$ for RMs - there's an exponential penalty for using a register machine!
Addition can be $\mathcal{O}(1)$ if we add dedicated `ADD(i,j)` and `SUB(i,j)` instructions - but this doesn't remove the inaccuracy it just makes it smaller.

Complexity is useful, but the measures are slightly bogus:

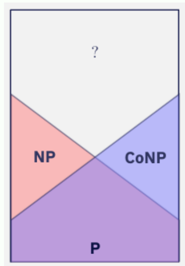
- $\mathcal{O}(n)$ is not always easy - if n is massive for example
- $\Omega(2^n)$ isn't always hard - there are problems much worse than this that are still solvable for real examples
- $\mathcal{O}(n^{10})$ and $\Omega(n^{10})$ seem extremely difficult, but a new model or algorithm could reduce that significantly
- Since we also ignore coefficients, we could have something like $f(n) \geq 10^{100} \log n$ which is slow but still only logarithmic - this isn't common enough to worry generally however.

Definition: Complexity Classes

Let $t : \mathbb{N} \to \mathbb{R}_{\geq 0}$. A time complexity class $\mathbf{TIME}(t(n))$ is the collection of all problems that are decidable by a deterministic machine RM, TM etc.) in $\mathcal{O}(t(n))$ time.

Given $A = \{0^i 1^i \mid i \in \mathbb{N}\}$, a TM can decide this in $\mathcal{O}(n^2)$, which means that $A \in \mathbf{TIME}(n^2)$.

We also define $\mathbf{NTIME}(t(n))$ to be the collection of all problems decidable by a nondeterministic machine NRM, NTM, etc.) in $\mathcal{O}(t(n))$.



Definition: P / Polynomial Time

The polynomial complexity class $\mathbf{P}$ is the class of problems decidable with some deterministic polynomial time complexity.

$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \mathbf{TIME}(n^k)$$

Problems in $\mathbf{P}$ are called tractable. Any problem not in $\mathbf{P}$ is $\Omega(n^k)$ for every k.

The class itself is robust, reasonable changes to model don't change it and reasonable translations between problems preserves membership in $\mathbf{P}$.

A polynomially-bounded RM together with a polynomial (n^k for some k, without loss of generality), such that given an input w it will always halt after executing $|w|^k$ instructions. A problem Q is in $\mathbf{P}$ iff it is computed by such a machine.

Definition: NP / Nondeterministic-polynomial time

The polynomial complexity class $\mathbf{NP}$ is the class of problems decidable with some nondeterministic polynomial time complexity.

$$\mathbf{NP} = \bigcup_{k \in \mathbb{N}} \mathbf{NTIME}(n^k)$$

We don't know if every exponentially bounded problem is in $\mathbf{NP}$, we think that it's probably not the case.

Definition: CoNP

We don't know if the class NP is closed under complement. We cannot flip the result because the result of flipping in a nondeterministic machine involves turning it from angelic nondeterminism to demonic nondeterminism (otherwise known as co-nondeterminism) or vice versa.

Definition: AP / Alternating Poly time

The class **AP** is the class of all problems decidable by an alternating machine in polynomial time with no restriction on swapping quantifiers.

AP is known to be equal to PSPACE.

Definition: PSPACE / Polynomial Space

An RM/TM is $f(n)$-space-bounded if it may use only $f(inputsize)$ space. For TMs this is the number of cells on the tape, where register machines uses bits in registers.

Theorem 15: Polynomial Reductions

A polynomial reduction from $P_1 = (D_1, Q_1)$ to $P_2 = (D_2, Q_2)$ is a $\mathbf{P}$-computable function $f : D_1 \to D_2$ such that $d \in Q_1$ iff $d \in Q_2$.

If P_2 is in $\mathbf{P}$, then P_1 is in $\mathbf{P}$ straightforwardly. Therefore to prove a problem is not in $\mathbf{P}$ we can show that there is a polynomial reduction from a problem P_2 which isn't in $\mathbf{P}$ to our problem P_1

- A problem P_1 is **polynomially reducible** to P_2, written $P_1 \leq_P P_2$ if there is a polynomially-bounded reduction from P_1 to P_2.
- A problem P is $\mathbf{NP}$-Hard if for every $A \in \mathbf{NP}$, $A \leq_P P$.

  That is, if a problem P_1 is $\mathbf{NP}$-Hard and P_1 is polynomially reducible to P_2, then P_2 is also $\mathbf{NP}$-Hard.

  We can use this fact to prove other problems are $\mathbf{NP}$-Hard by showing a reduction from a known $\mathbf{NP}$-hard problem.
- A problem is $\mathbf{NP}$-Complete if it is both $\mathbf{NP}$-Hard and in $\mathbf{NP}$.

Theorem 16: Cook-Levin Theorem

The Cook-Levin theorem states that the NP problem SAT is NP-Complete.
The proof of this involves showing it is NP by nondeterministically guessing assignments and checking them in polynomial time, and then showing it's NP- Hard by reducing any NP problem to SAT

The Polynomial Heirarchy

Definition: Sigma Notation

The set Σ_1^P describes all problems that can be phrased as

$$\{y \mid \exists^p x \in \mathbb{N}.\, R(x, y)\}$$

where R is a **P**-decidable predicate and $\exists^p x \ldots$ indicates that x is of size polynomial in the size of y.
We can say that x is a certificate showing which guesses can be made by our NRM giving an accepting run.
If a problem $Q \in \Sigma_1^P$ then Q is **NP**, because it is a problem for which we can verify the answer in polynomial time. If a problem Q is in **NP** then $Q \in \Sigma_1^P$.
So, $\mathbf{NP} = \Sigma_1^P$

Definition: Pi notation

The set Π_1^P describes all problems that can be phrased as

$$\{y \mid \forall^P x \in \mathbb{N}.\, R(x, y)\}$$

where R is a **P**-decidable predicate, and $\forall^P x \ldots$ indicates that x is of size polynomial in the size of y.
We have that

$$\Pi_1^P = \overline{\Sigma_1^P}, \quad \text{and} \quad \Pi_1^P = \mathbf{CoNP}$$

Definition: Delta Notation

There are two conflicting definitions of Δ_1^P "For reasons that are unknown to me" - lecturer

- The set Δ_1^P describes the intersection of Σ_1^P and Π_1^P
- The set Δ_1^P describes the set **P**

From our characterisations of Σ_1^P and Π_1^P, we have that $\Delta_1^P \subseteq \mathbf{P}$, but we don't konw if these definitions are equal

Definition: Moving higher

The next layer of the heirarchy goes as follows:

- Σ_2^P is all problems of the form $\{x \mid \exists^P y.\, \forall^P z.\, R(x, y, z)\}$
- Π_2^P is all problems of the form $\{x \mid \forall^P y.\, \exists^P z.\, R(x, y, z)\}$
- $\Delta_2^P = \Sigma_2^P \cap \Pi_2^P$

We can also use oracles to get an alternate definition:

- Δ_2^P is all problems that are decidable in polynomial time by some deterministic RM/TM with an $\mathcal{O}(1)$ oracle for some problem in Σ_1^P (it is **P** with an $\mathcal{O}(1)$ oracle for **NP**)
- Σ_2^P allows the TM/RM to be nondeterministic (it is **NP** with an $\mathcal{O}(1)$ oracle for **NP**)
- Π_2^P is **CoNP** with an oracle for **NP**

In general for any $n > 1$:

- Δ_n^P is all problems decidable by a deterministic polynomially bounded TM/RM with an $\mathcal{O}(1)$ oracle for some problem in Σ_{n-1}^P.
- Σ_n^P is all problems decidable by some nondeterministic polynomially bounded TM/RM with an $\mathcal{O}(1)$ oracle for some problem in Σ_{n-1}^P.
- Π_n^P is all problems decidable by some co-nondeterministic polynomially bounded TM/RM with an $\mathcal{O}(1)$ oracle for some problem in Σ_{n-1}^P.

Note: Co-nondeterminism could also be called **demonic** nondeterminism, like regular (angelic) nondeterminism but only accepts if **all** paths accept

Definition: Alternation

Equivalently Σ_n^P are all problems that can be phrased as some **alternation** of (**P**-bounded) quantifiers, starting with $\exists^P$

$$\{w \mid \exists^P x_1.\exists^P x_2.\exists^P x_3.\exists^P x_4.\ldots x_n.R(w, x_1, \ldots, x_n)\}$$

Π_n^P has a similar definition, starting instead with $\forall^P$

$$\{w \mid \forall^P x_1.\exists^P x_2.\exists^P x_3.\exists^P x_4.\ldots x_n.R(w, x_1, \ldots, x_n)\}$$

Alternating machines combine the acceptance modes of both angelic and demonic non-deterministic machines.

Alternating register machines would replace the NRM's `MAYBE` instruction with the `MAYBE`$^\exists$ and `MAYBE`$^\forall$ instructions, which are nondeterministic branching choices where acceptance depends on if one branch accepts ($\exists$) or both branches accept ($\forall$).

Alternating Turing Machines are defined by labelling states with either $\forall$ or $\exists$.

The class Σ_n^P can therefore be described as the class of problems decided in polynomial time by an alternating machine that initially uses $\exists$-nondeterminism and swaps quantifiers at most $n-1$ times. This extends to Π_n^P swapping starting with $\exists$ to starting with $\forall$.

blank filler text: Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Definition: Name

Example : Regular Expressions

At least one 0:

$$(0 \cup 1)^* 0 (0 \cup 1)*$$

At least one 1 and at least one 0:

$$((0 \cup 1)^* 01 (0 \cup 1)^*) \cup ((0 \cup 1)^* 10 (0 \cup 1)^*)$$