Big Picture:

Smallest to largest

SFinite = {L ⊆ Σ ∗ | L is finite}

SDFA = {L ⊆ Σ ∗ | L = L(D) for some DFA D}

SCFG = {L ⊆ Σ ∗ | L = L(G) for a CFG G}

STM-Dec = {L ⊆ Σ ∗ | L is decided by a deterministic TM M}

S2Tape-TM-Rec = {L ⊆ Σ ∗ | L = L(M) for a deterministic TM M with 2 tapes}

SAll = {L ⊆ Σ ∗}, the set of all languages over Σ ∗ .

describe a process used to **decide whether the empty string is in the language** L(G) of G. **Solution**: Use the iterative algorithm that has been described in class to compute the set of nullable variables of the given grammar G. The empty string is in the language of this grammar iff the start variable is found to be a nullable variable during this computation.

Describe a process that can be used to **decide whether the language L(G) of G is empty**, that is, whether L(G) = ∅. **Solution**: Apply the iterative process that has been described in class to compute the set of generating symbols in the given grammar; the language of the grammar is nonempty if and only if the start variable has been found to be generating when this process is applied.

**Every regular language is context-free.** the empty set, the set including the only the empty string, and the set including σ for every symbol σ in any given alphabet Σ, are all context-free languages — it is easy to generate context-free grammars for each. The set of context-free languages is also closed under each of the regular operations, union, concatenation, and closure. These can be used to prove that the language of every regular expression is context-free, and implies that every regular language is context-free, as well.

**Every context-free language is regular.** Solution: No, this claim is not true. Justification: Several different languages, including the language {a ^n b ^n | n ≥ 0} were proved not to be regular (using the pumping lemma for regular languages) in class, but are clearly context-free.

A **closure property** is a theorem about a set of languages, such as the set of regular languages, the set of context-free languages. it is a theorem stating that if certain languages belong to the given set, and another language is formed from these languages using a certain operation, then the resulting language belongs to the given set, as well. This result indicates that the given set of languages is closed under the given operation. if L1 and L2 are two regular languages, then the union L1 U L2 is a regular language as well. That is, the set of regular languages is closed under the union operation.

An ***alphabet*** is a finite nonempty set.

A ***string over an alphabet*** is a finite *sequence* of symbols from that alphabet, usually written next to each other and *not* separated by commas. • 000011010100

A ***deterministic finite automaton (DFA)*** is (formally modelled as) a 5-tuple (*Q*,Σ,δ,*q*0,*F*), where

*1Q* is a finite, nonempty, set of ***states***;   2 Σ is an ***alphabet*** (as previously defined) — we normally  require that *Q* ∩ Σ = ∅;

1. δ:*Q*×Σ→*Q* is a ***transition function***;  4 *q*0 ∈ *Q* is the ***start state***; and  5 *F* ⊆ *Q* is the set of ***accept states***.

A ***NFA*** is 5-tuple (*Q*,Σ,δ,*q*0,*F*),

where 1. *Q* is a finite (and nonempty) set of ***states***, 2. Σ is a finite (and nonempty) ***alphabet***, 3. δ : *Q* × Σλ → P(*Q*) is the ***transition function***, 4. *q*0 ∈ *Q* is the ***start state***, and 5. *F* ⊆ *Q* is the set of ***accept states***. For *q* ∈ *Q* and σ ∈ Σλ, δ(*q*,σ) is the *set* of states that can be reached by following a ***single*** transition for σ out of *q*.

**Context Free Grammar Sample**

This context-free grammar is G = (V, Σ, R, q0), where

• V ={q0,q1,q2}; • Σ = {a,b,c}; • q0 is the start variable; and • R includes the following rules: q0 →aq1

**What is the set of strings in T ∗ that can derived from the variable A?**

The language of the variable A is the set {a i b i | i ≥ 0}. 4 (d) Briefly explain how you could prove that your answer for part (c) is correct. could be proved by showing that each of the following claims is correct. • A ⇒∗ G a i b i for every integer i ≥ 0. • If w ∈ T ∗ and A ⇒∗ G w then w = a i b i for some integer i ≥ 0. The first of these claims can be established by induction on either i or the length of the string. The second can be established using induction on either the length of the given string or the length of its derivation from A.

**{a i b i | i ≥ 0}.**explain how you could prove that your answer for part (c) is correct. Explanation: This could be proved by showing that each of the following claims is correct. • A ⇒∗ G a i b i for every integer i ≥ 0. • If w ∈ T ∗ and A ⇒∗ G w then w = a i b i for some integer i ≥ 0. The first of these claims can be established by induction on either i or the length of the string. The second can be established using induction on either the length of the given string or the length of its derivation from A.

**Chomsky Normal Form** properties start variable, S, does not appear as part of the right-hand side of any rule. • The only rules in the grammar have one of the following forms: A→BC where A, B and C are variables (that is, A,B,C∈V); – A→σ where A is a variable and σ is a terminal ( that is, A∈V and σ∈Σ);  or  S → λ where S is the start variable.

**Chomsky Transformations**

1Make sure that the start variable is not included in the right-hand side of any rule (by creating a new start variable, if necessary).

2Make sure that the right-hand side of every rule has length at most two (by adding new variables and replacing rules with long right-hand sides by a bunch of new rules with short right-hand sides).

3Eliminate λ-rules (except for the rule S → λ if S is the start variable).  **Eliminate unit rules.**

4Eliminate rules that have right-hand sides with length two and include a terminal — by adding new variables that yield these terminals, and replacing the terminals in the right-hand sides of these rules by the corresponding variables.

**Turing Machines**

Start configuration for a **Turing machine** M = (Q, Σ, Γ, δ, q0, qA, qR) is the configuration in which M is in its start state q0,

* M’s tape stores the symbols in the input string ω (in order, at the left end  of the tape) followed by infinitely many blanks, and

the tape head points to the leftmost cell on the tape — storing the first  symbol in ω if ω is not the empty string, and storing ⊔ otherwise.

* M **loops** on an input string ω if M makes an infinite series of moves when executed on input ω (that is, beginning with the start configuration for ω) — without ever halting and either accepting or rejecting ω.
* M **recognizes** a language L ⊆ Σ\* if Σ is the input alphabet of M and, for every string x ∈ Σ\*,  • if x ∈ L then M accepts x, and • if x∈/ L then M either rejects or loops on x  — so that L is the language of M.
* M **decides** a language L ⊆ Σ\* if Σ is the input alphabet of M and, for every  string x ∈ Σ\*, • if x ∈ L then M accepts x, and  • if x ∈/ L then M rejects x, — so that L is the language of M and, furthermore, M halts when executed on  every string in Σ\*.
* A **many-one reduction** from L1 to L2 is a total function f : Σ\*1 → Σ\*2 satisfying the following properties.
* For all ω∈Σ\*1,ω∈ L1 if and only if f(ω)∈L2. (b) The function f is computable. A process, that can be used to prove that L is undecidable, is as follows.

Choose some language L ⊆ Σ\* that is already known to be undecidable.  **Clearly** describe a total function f : Σ\*→ Σ\*.

Prove that if ω∈Lthen f(ω)∈L, for all ω∈Σ\*.  **Prove** that if ω∈/L then f(ω)∈/L, for all ω∈Σ\*

Prove that f is computable — by describing a Python program, Java program, or Turing machine that computes the function f and showing that this algorithm is correct.

**Infinite Loop detector problem** No, Suppose, contradiction, that such a program existed. Recall that it is possible to write a Java program that receives an encoding of a Turing machine M and input string x and that simulates the execution of M on x. This program would halt on the input M and x if and only if M halted on x. This can be used to design a Java program that receives the input x alone and that halts on input x iff M does. If this program and the string x are supplied as input to the “infinite loop detector” then the detector would decide whether M halts on input x. This would imply that some Turing machine could be used to decide whether a given Turing machine M halts on a given string x. This implies that the Halting is undecidable. Since the Halting Problem is undecidable, thus a contradiction.

**If L1 is undecidable then L2 is also undecidable**. Yes, this claim is true. Suppose, to the contrary, that the claim is false. Then there exist languages L1 ⊆ Σ ∗ 1 and L2 ⊆ Σ ∗ 2 such that L1 is reducible to L2 and L1 is undecidable, but L2 is decidable. Since L1 is reducible to L2 there exists a function f : Σ ∗ 1 → Σ ∗ 2 such that f is a reduction from L1 to L2. The function f is :. computable, so that there is a TM Mf that computes f. Since L2 is decidable there exists a Turing machine M2 that decides L2: That is, L(L2) = M2 and M2 halts on every input string in Σ ∗ 2 . Now consider a Turing machine M1 with input alphabet Σ1, with subroutines Mf and M2, and that has the following structure. In the above picture the machine halts and accepts if the output that is shown is “yes” and the machine halts and rejects if the output that is shown is “no.” It is not difficult to prove that the language of this machine is L1 (using the definition of a “reduction” and the fact that f is one) and that this machine halts on all inputs in Σ ∗ 1 . It follows that L1 is decidable. However, this contradicts the fact (given above) that L1 is undecidable. We therefore have a contradiction. Since the only assumption that has been made is that the claim is false, this establishes the claim.

**Church-Turing Thesis** is a widely believed, but unprovable, assumption that any general way to compute will allow us to compute only (and exactly) what Turing machines or modern-day computers can compute. This is believed because numerous general models of computation have been proposed, since the early part of the twentieth century, and they have all been proved (via simulation) to be equivalent in power to Turing machines. It is unprovable because it refers to models of computation that have not even been proposed yet