Computer Science 690AB, Winter 2026
Assignment #1, Question #1
Name: XXXXXXXXXXXXXXXXXXXX
MUN #: XXXXXXXXXXXXXXXXXX

> All this is fact. Fact explains nothing. On the contrary, it is fact that requires explanation.
>
> Marilynne Robinson, *Housekeeping*

1 Background

1.1 Parameterized Complexity Analysis

We can now talk about algorithm efficiency. Typical computational resources of interest are time and space, which correspond to the number of instructions executed or the amount of memory used by the algorithm when it is implemented on some standard type of computer, e.g., a deterministic Turing machine (for detailed descriptions of the various kinds of Turing machines, see [2, 3, 4]). For some resource R and problem Π, let $R_A : D_\Pi \mapsto \mathcal{N}$ be be the function that gives the amount of resource R that is used by algorithm A to solve a given instance of Π. The resource-usage behavior of an algorithm over all possible instances of its associated problem is typically stated in terms of a function of instance size that summarizes this behavior in some useful manner. The creation of such functions has three steps:

1. Define an instance-length function such that each instance of the problem of interest can be assigned a positive integer size. Let the size of instance I be denoted by $|I|$.

2. Define a "raw" resource-usage function that summarizes the resource-usage behavior of A for each possible instance size. Let $R_A^n = \{R_A(I) \mid I \text{ is an instance of the problem solved by } A \text{ and } |I| = n\}$ be the R-requirements of algorithm A for all instances of size n. For each instance-size n, choose either one element of or some function of R_A^n to represent R_A^n. Several popular ways of doing this are:

   - The highest value in R_A^n (m worst-case).
   - The lowest value in R_A^n (m best-case).

- The average value of R_A^n relative to some probability distribution on instances of size n (m average-case).

Let $S_A : \mathcal{N} \mapsto \mathcal{N}$ be the function that gives this chosen value for $n > 0$.

3. "Smooth" the raw resource-usage function $S_A(n)$ via a function $C_A : \mathcal{N} \mapsto \mathcal{N}$ that asymptotically bounds $S_A(n)$ in some fashion. Several standard types of asymptotic bounding functions g on a function f are:

   - m Asymptotic upper bound: $f \in O(g)$ if there exists a constant c and $n_0 \geq 0$ such that for all $n > n_0$, $f(n) < c \cdot g(n)$.
   - m Asymptotic lower bound: $f \in \Omega(g)$ if there exists a constant c and $n_0 > 0$ such that for all $n > n_0$, $f(n) > c \cdot g(n)$.
   - m Asymptotic tight bound: $f \in \Theta(g)$ if $f \in O(g)$ and $f \in \Omega(g)$.

1.2 Phonological Mechanisms as Finite-State Automata

Definition 1 *A m configuration of a FSA $A = \langle Q, \Sigma, \delta, s, F \rangle$ is a pair (q, x) where $q \in Q$ and $x \in \Sigma^*$.*

Definition 2 *Given two configurations (q, x) and (q', x') of a FSA $A = \langle Q, \Sigma, \delta, s, F \rangle$, m (q, x) yields (q', x') in one step, i.e., $(q, x) \vdash (q', x')$, if $x = wx'$ for some $w \in \Sigma \cup \{\epsilon\}$ and $(q, w, q') \in \delta$.*

Let $\vdash^*$ represent the reflexive transitive closure of the yield relation, i.e., $(q, x) \vdash^* (q', x')$ if and only if either $q = q'$ and $x = x'$ or there exists some sequence $(q_1, x_1), (q_2, x_2), \ldots, (q_n, x_n)$ of one or more configurations such that $(q, x) \vdash (q_1, x_1) \vdash (q_2, x_2) \vdash \cdots \vdash (q_n, x_n) \vdash (q', x')$.

Definition 3 *Given a FSA $A = \langle Q, \Sigma, \delta, s, F \rangle$ and string $x \in \Sigma^*$, x is m accepted by A if and only if $(s, x) \vdash^* (q, \epsilon)$ for some $q \in F$.*

Essentially, a computation of a FSA on a given string x is a path p in the transition diagram for that FSA such that p starts at the vertex corresponding to s and the concatenation of the edge-labels of the edges in p is x; if the final vertex in p corresponds to a state in F, then x is accepted by the FSA.

Example 1 Consider the computation of the FSA in [3] on several given strings. If the given string is $x = baabb$, x is accepted as there is a computation of the FSA on x which ends at state $q_3 \in F$.

$$\begin{aligned}(q_1, baabb) &\vdash (q_1, aabb)\\ &\vdash (q_2, abb)\\ &\vdash (q_2, bb)\\ &\vdash (q_3, b)\\ &\vdash (q_3, \epsilon)\end{aligned}$$

However, if $x = baabbab$, as the state q_4 in the final configuration is not in F, x is not accepted.

$$\begin{aligned}(q_1, baabbab) &\vdash (q_1, aabbab)\\ &\vdash (q_2, abbab)\\ &\vdash (q_2, bbab)\\ &\vdash (q_3, bab)\\ &\vdash (q_3, ab)\\ &\vdash (q_4, b)\\ &\vdash (q_4, \epsilon)\end{aligned}$$

∎

Each FSA can be visualized as encoding a set of strings.

In the case of those operations defined above which create automata of exactly the same type as their given pair of automata, e.g., i/o-deterministic FST intersection, it is possible to define versions of those operations that take as input an arbitrarily large number of automata. Two possible ways of defining these operations are (1) extend the constructions given above relative to cross products on arbitrary numbers of rather than pairs of state-sets and (2) iterate the pairwise operations over the given set of automata, i.e., repeatedly remove two automata from the given set, apply the pairwise operation, and put the created automaton back in the set until only one automaton is left in the set. For the sake of simplicity, only alternative (2) will be considered in more detail here. In the case of intersection, the automata can be combined in a pairwise fashion in any order; however, as composition is sensitive to the order of its operands, e.g., the composition of A_1 and A_2 is not necessarily equivalent to the composition of A_2 and A_1, the automata must be combined in a specified order. For simplicity in the analyses below, assume that automata in a given set are combined in a pairwise manner relative to their order of appearance when that set is written down, i.e., given a set of automata $A = \{A_1, A_2, \ldots, A_k\}$, A_1 and A_2 will be

combined to create A', A' and A_3 will be combined to create A'', and so on. Under this scheme, for a given set of automata $A = \{A_1, A_2, \ldots, A_k\}$ of the appropriate type such that $|Q|$ is the maximum number of states in any automaton in A and $|\Sigma|$ is the maximum number of symbols in any alphabet associated with an automaton in A, the time complexities of this iterative process relative to several operations on automata are derived as follows:

- **ϵ-free FST composition**: Given that the composition FST of two ϵ-free FST $A_1 = \langle Q_1, \Sigma_{i,1}, \Sigma_{o,1}, \delta_1, s_1, F_1 \rangle$ and $A_2 = \langle Q_2, \Sigma_{o,1}, \Sigma_{o,2}, \delta_2, s_2, F_2 \rangle$ in A can be computed in $O((|Q_1||Q_2|)^2|\Sigma_{i,1}||\Sigma_{o,1}|^2|\Sigma_{o,2}|) = O(|Q|^4||\Sigma|^4) = c|Q|^4|\Sigma|^4$ time for some constant $c > 0$, the composition FST of A can be computed in
$$\begin{aligned} \Sigma_{i=2}^{k} O(|Q|^{2i}|\Sigma|^4) &= c|\Sigma|^4 \Sigma_{i=2}^{k} |Q|^{2i} \\ &\leq c|\Sigma|^4 k|Q|^{2k} \\ &= O(|Q|^{2k}|\Sigma|^4 k) \end{aligned}$$
time.

- **DFA intersection**: Given that the intersection DFA of two DFA $A_1 = \langle Q_1, \Sigma, \delta_1, s_1, F_1 \rangle$ and $A_2 = \langle Q_2, \Sigma, \delta_2, s_2, F_2 \rangle$ in A can be computed in $O(|Q_1||Q_2||\Sigma|^2) = O(|Q|^2|\Sigma|^2) = c|Q|^2|\Sigma|^2$ time for some constant $c > 0$, the intersection DFA of A can be computed in
$$\begin{aligned} \Sigma_{i=2}^{k} O(|Q|^i|\Sigma|^2) &= c|\Sigma|^2 \Sigma_{i=2}^{k} |Q|^i \\ &\leq c|\Sigma|^2 \Sigma_{i=0}^{k} |Q|^i \\ &= c|\Sigma|^2 (|Q|^{k+1} - 1)/(|Q| - 1) \\ &\leq c|\Sigma|^2 |Q|^{k+1} \\ &= O(|Q|^{k+1}|\Sigma|^2) \end{aligned}$$
time.

The time complexities of these operations and upper bounds on the sizes of the created automata are given for each of these operations in Table 1. Note that if the number of states in each of the given automata is the same, the given upper bounds on the sizes of automata created by these operations are exact, in that automata may be created that have numbers of states and transitions that are equal to these upper bounds. Hence, though there exist implementations of some of these operations that can be much more efficient than the given naive implementations in certain applications [5, 6], the worst-case running times of all such implementations are lower-bounded by the given upper bounds on the sizes of the created automata.

Automaton Operation	Upper Bound on Automaton Size		Asymptotic Worst-Case Time Complexity of Automaton Creation
	States	Transitions	
ϵ-free FST composition	$\|Q\|^k$	$\|Q\|^{2k}\|\Sigma\|^2$	$O(\|Q\|^{2k}\|\Sigma\|^4 k)$, $\Omega(\|Q\|^{2k}\|\Sigma\|^2)$
DFA intersection	$\|Q\|^k$	$\|Q\|^k\|\Sigma\|$	$O(\|Q\|^{k+1}\|\Sigma\|^2)$, $\Omega(\|Q\|^k\|\Sigma\|)$
i/o-FST intersection	$\|Q\|^k$	$\|Q\|^k\|\Sigma\|^2$	$O(\|Q\|^{k+1}\|\Sigma\|^4)$, $\Omega(\|Q\|^k\|\Sigma\|^2)$
ϵ-free FST intersection	$\|Q\|^k$	$\|Q\|^{2k}\|\Sigma\|^2$	$O(\|Q\|^{2k}\|\Sigma\|^4 k)$, $\Omega(\|Q\|^{2k}\|\Sigma\|^2)$

Table 1: Characteristics of Iterated Finite-State Automaton Operations. This table gives the asymptotic worst-case time complexities of the operations of (as well as upper bounds on the sizes of automata created by) iterating various operations defined on pairs of automata over sets of automata, where k is the number of automata in the given set, $|Q|$ is the maximum number of states in any automaton in that set, and $|\Sigma|$ is the maximum number of symbols in any alphabet associated with an automaton in that set.

2 Analysis of KIMMO System

As KIM-Encode and KIM-Decode are special cases of KIM(N)-Encode and KIM(N)-Decode, respectively, all NP- and W-hardness results derived above still hold for these new problems. Having insertions and deletions does allow certain hardness results to hold in more restricted cases; for instance, using the trick given in the reduction in [1, Section 5.7.2] in which a given form in a reduction consists of two dummy terminator symbols and the FST in A are restructured to construct arbitrary requested forms over the nulls in the given form, it is possible to rephrase all hardness results above such that the size of the given form alphabet and the length of the given form are both 2. It seems inevitable that allowing insertions and deletions will also both allow certain hardness results to hold relative to higher levels of the W hierarchy and allow parameterized problems that were formerly known to have FPT algorithms to be shown W-hard. The full extent of these changes will not be addressed here. For now, simply observe that the FPT algorithms based on FST intersection will still work if each FST is modified to accept arbitrary numbers of lexical or surface nulls at any point in processing (this can be ensured by adding to each FST the sets of transitions $\{\delta(q, \mathbf{0}, x) = q \mid x \in \Sigma_s\}$ and $\{\delta(q, x, \mathbf{0}) = q \mid x \in \Sigma_u\}$ for every state $q \in Q$), and the FPT algorithms based on brute-force enumeration of all possible re-

quested forms will work if the maximum number of nulls that can be added is also a aspect in the parameter (this is so because the number of possible null-augmented versions of a form f over an alphabet Σ that incorporate at most k nulls is $|\Sigma|^{|f|} \sum_{i=1}^{k} \begin{pmatrix} |f|+i \\ i \end{pmatrix} \leq |\Sigma|^{|f|} k(|f|+k)^k$, which is a function of $|\Sigma|$, $|f|$, and k).

References

[1] G. Edward Barton, Robert C. Berwick, and Eric S. Ristad. 1987. *Computational Complexity and Natural Language*. MIT Press, Cambridge, MA.

[2] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco.

[3] John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

[4] Harry R. Lewis and Christos H. Papdimitriou. 1981. *Elements of the Theory of Computation*. Prentice Hall, Englewood Cliffs, NJ.

[5] Fernando C.N. Pereira and Michael D. Riley. 1997. Speech Recognition by Composition of Weighted Finite Automata. In Emmanuel Roche and Yves Schabes, eds., *Finite-State Language Processing*, pages 431–453. MIT Press, Cambridge, MA.

[6] Pasi Tapanainen. 1997. Applying a Finite-State Intersection Grammar. In Emmanuel Roche and Yves Schabes, eds., *Finite-State Language Processing*, pages 311–327. MIT Press, Cambridge, MA.

Candidate Full Forms	Constraint Violations		
	c_1	c_2	c_3
f_1	$\{a_1, a_1, b_1\}$	$\{b_2\}$	ϕ
f_2	$\{b_1\}$	$\{a_2\}$	$\{a_3\}$
f_3	$\{b_1\}$	$\{b_2\}$	$\{b_3\}$

(a)

	Candidate Full Forms	Constraint Violations		
		c_1	c_2	c_3
	f_1	[$\underline{2}$ 1]	[0 1]	[0 0]
	f_2	[0 1]	[$\underline{1}$ 0]	[1 0]
$\Rightarrow$	f_3	[0 1]	[$\boxed{0}$ 1]	[0 1]

(b)

	Candidate Full Forms	Constraint Violations
		[c_1 c_2 c_3]
	f_1	[$\underline{2}$ 1 0 1 0 0]
	f_2	[0 1 $\underline{1}$ 0 1 0]
$\Rightarrow$	f_3	[0 1 $\boxed{0}$ 1 0 1]

(c)

Figure 1: Evaluation of Candidate Full Forms in Optimality Theory. (a) Marks assigned to candidate full forms f_1, f_2, and f_3 by binary constraints c_1, c_2, and c_3 which have the associated mark-sets $\{a_1, b_1\}$, $\{a_2, b_2\}$, and $\{a_3, b_3\}$, respectively. (b) Evaluation of candidates when $c_1 \gg c_2 \gg c_3$ and $a_i \succ b_i$, $1 \leq i \leq 3$. Note that sets of marks from (a) have been replaced by the appropriately-ordered weight vectors. Optimal candidates are flagged by an arrow ($\Rightarrow$), mark-values that caused the elimination of candidates are underlined, e.g., $\underline{1}$, and mark-values that resulted in a candidate being chosen as optimal are framed by a box, e.g., $\boxed{0}$. (c) Evaluation of (b) relative to appropriately concatenated weight vectors. This shows more clearly the lexicographic optimality ordering on weight vectors.