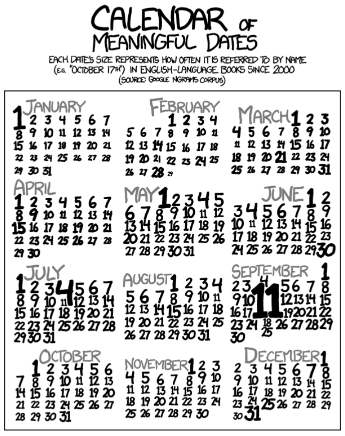
Proposed Course Calendar

CSE 4083/5210 Formal Languages & Automata Theory

Summer 2017 (June 20, 2017)

This course calendar predicts when class events are expected to happen. It is not written in stone. Nothing is certain. Things may change. Pay attention. It will be updated, but it may not be up to date. Colors are used to indicate an exam or assignment due date, a holiday, or a link to additional information. Course reference textbooks for the course include (Linz, 2017) and (Hopcroft et al., 2006).


CALENDAR of MEANINGFUL DATES
EACH DATE'S SIZE REPRESENTS HOW OFTEN IT IS REFERRED TO BY NAME (E.G. "OCTOBER 17TH") IN ENGLISH-LANGUAGE BOOKS SINCE 2000 (SOURCE: GOOGLE NGRAMS CORPUS)
JANUARY
FEBRUARY
MARCH
APRIL
MAY
JUNE
JULY
AUGUST
SEPTEMBER
OCTOBER
NOVEMBER
DECEMBER

Calendar of meaningful dates

May

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~
~~8~~	~~9~~	~~10~~	~~11~~	~~12~~	~~13~~	~~14~~
~~15~~	~~16~~	~~17~~	~~18~~	~~19~~	~~20~~	~~21~~
~~22~~	~~23~~	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~
~~29~~	~~30~~	~~31~~				

June

			~~1~~	~~2~~	~~3~~	~~4~~
~~5~~	~~6~~	~~7~~	~~8~~	~~9~~	~~10~~	~~11~~
~~12~~	~~13~~	~~14~~	~~15~~	~~16~~	~~17~~	~~18~~
~~19~~	~~20~~	21	22	23	24	25
26	27	28	29	30		

July

					1	2
3	4	5	6	7		

Week 1

- Tuesday, May 16:
  - Course structure (Syllabus)
  - Gradiance system for homework
  - Course management system for communication, link collection, grades, attendance
  - Mathematics preliminaries (Dr. Stansifer's notes)
    - Inductive proofs over well-founded data structures
    - Relations and Functions
  - Diagonalization: Let $f : \mathbb{X} \mapsto 2^{\mathbb{X}}$ (upshot: f cannot be onto; there exists a subset of $\mathbb{X}$ that is not in the range of f): $\mathbb{D} = \{x \in \mathbb{X} : x \notin f(x)\}$. Then $(\forall x \in \mathbb{X})(\mathbb{D} \neq f(x))$
    If $x \notin f(x)$, then $x \in \mathbb{D}$. If $x \in f(x)$, then $x \notin \mathbb{D}$. (N.B. The proof does not provide a construction of $\mathbb{X}$, only its existence.)
- Thursday, May 18:
  - Overview (Dr. Stansifer's notes)
  - Linz 1st slide deck
    - **Formal Language:** Finite alphabet Σ; String $s \in \Sigma^n$, $n \geq 0$, Σ^* Kleene closure (set of all strings), Σ^+ (all non-empty strings); Language $\mathbb{L} \subseteq \Sigma^*$
    - **Automaton:** I/O decision makers, with various memory models
    - **Grammar:** Rules for enumerating strings (words, sentences) in a formal language
  - Linz 2nd slide deck
    - Languages, strings, operations (concatenation, reverse, length, repetition)
    - Recursive definitions:

Strings — Basis: the empty string λ is a string. For each $a \in \Sigma$, a is a string
Construction: If w is a string then wa is a string ($\forall a \in \Sigma$).
Length — students respond
Reverse — students respond
Finite repetition — students respond

* Sub-string, prefixes, suffixes (paths in a graph, from the start state, to a final (or dead) state
* Set operations on languages: Union $\cup$, Intersect $\cap$, Difference $-$, Complement $\overline{}$,
* Finite accepter
* Transition graph
* Configuration (current state of the machine)
* **Deterministic finite automata (DFAs)** $M = (Q, \Sigma, \delta, q_0, \mathbb{F})$ where
  · $\mathbb{Q}$ is a finite set of states
  · Σ is a finite alphabet
  · $\delta :: Q \to \Sigma \to Q$ is a transition function $\delta : Q \times \Sigma \to Q$
  · $q_0 \in \mathbb{Q}$ is the initial state
  · $\mathbb{F}$ is set of final states
* Transition table representation of δ
* Extension of δ to δ^* to strings (paths/walks)
* Recursive definition of δ^*:

$$\delta^*(q, \lambda) = q; \qquad \delta^*(q, wa) = \delta(\delta^*(q,, w), a)$$

* **Definition:** Language accepted by DFA

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in \mathbb{F}\}$$

* Complement language
* **Definition:** A language L is regular if there exists a DFA M such that $L = L(M)$.

– Linz 3rd slide deck
  * Non-Deterministic Finite Automata (NFA) $N = (Q, \Sigma, \delta_N, q_0, \mathbb{F})$
    * δ_N is a *relation*

$$\delta_N \to Q \to (\Sigma \cup \{\lambda\}) \to 2^Q$$

in a state q a transition on $a \in \Sigma$ can be to many states.
    * λ transitions (movements from state to state without consuming input)

May

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~
~~8~~	~~9~~	~~10~~	~~11~~	~~12~~	~~13~~	~~14~~
~~15~~	~~16~~	~~17~~	~~18~~	~~19~~	~~20~~	~~21~~
~~22~~	~~23~~	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~
~~29~~	~~30~~	~~31~~				

June

			~~1~~	~~2~~	~~3~~	~~4~~
~~5~~	~~6~~	~~7~~	~~8~~	~~9~~	~~10~~	~~11~~
~~12~~	~~13~~	~~14~~	~~15~~	~~16~~	~~17~~	~~18~~
~~19~~	~~20~~	21	22	23	24	25
26	27	28	29	30		

July

					1	2
3	4	5	6	7		

      * NFA accepts w if some computation terminates in a final state.
      * NFA rejects w if no computation terminates in a final state.

- Tuesday, May 23:
  - Linz 3rd slide deck
    * Non-Deterministic Finite Automata (NFA)
      · **Theorem:** Language accepted by DFAs are identical to languages accepted by NFAs
        * DFAs are a special case of NFAs. Therefore, if $L = L(M)$ for some DFA M, then $L = L(N)$ for some NFA N [take $N = M$].
        * If $L = L(M')$ for some NFA M', then $L = L(M)$ for some DFA: Construction below)
          States $\mathbb{Q}$ of NFA map to states $2^{\mathbb{Q}}$ of DFA.

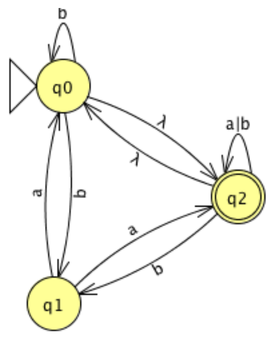
$$q_0 \mapsto \{q_0\}$$

          For each $a \in \Sigma$ and DFA state $\mathbb{X}$:

$$\delta(\mathbb{X}, a) = \{\delta^*(x, a) : x \in \mathbb{X}\} = \mathbb{X}'$$

          Add transition $(\mathbb{X}, a) \mapsto \mathbb{X}'$ to DFA
          If $f \in \mathbb{X}'$ is an NFA final state, then $\mathbb{X}'$ is a DFA final state

          Construct DFA from this NFA; Assume q and s are start and final states respectively.



          Construct a deterministic transition table
    * Linz 4th slide deck
      · NFA only requires a single finite state (create a new final state with λ transitions from original final (now none final) states

May

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~
~~8~~	~~9~~	~~10~~	~~11~~	~~12~~	~~13~~	~~14~~
~~15~~	~~16~~	~~17~~	~~18~~	~~19~~	~~20~~	~~21~~
~~22~~	~~23~~	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~
~~29~~	~~30~~	~~31~~				

June

			~~1~~	~~2~~	~~3~~	~~4~~
~~5~~	~~6~~	~~7~~	~~8~~	~~9~~	~~10~~	~~11~~
~~12~~	~~13~~	~~14~~	~~15~~	~~16~~	~~17~~	~~18~~
~~19~~	~~20~~	21	22	23	24	25
26	27	28	29	30		

July

					1	2
3	4	5	6	7		

- Regular languages are closed under: Union, Complement, Intersection, Set Difference, Concatenation, Kleene Star (Construct machines for these operations)
- **Definition:** Regular expressions

  Basis: $\emptyset$, λ, and a for all $a \in \Sigma$ are regular expressions

  Construction: If r_1 and r_2 are regular expressions then so are $r_1 + r_2$, $r_1 r_2$, r_1^*, and (r_1) (Or, Concatenation, Repetition, Parenthesizing)

* Machine and language for a regular expression — student response: define for primitives, demonstrate for constructions: Or, Concat, Repeat
* Interpreting a regular expression (semantics)
* **Theorem:** Regular languages are generated by regular expressions.
  - Regular expressions r define DFAs (proof by induction on length of r. `True` for primitive expressions, If $L(r_1)$ and $L(r_2)$ are regular languages, show the operations preserve regularity)
  - If L is a regular language construct a regular expression

– Linz 5th slide deck

* If L is regular, then the reverse language L^R is regular (Proof: invert transitions; make initial state q_0 for $M(L)$ the final state for $M(L^R)$; add new initial state q_0')
* Grammar: $G = (\mathbb{V}, \mathbb{T}, S, \mathbb{P})$ where
  - $\mathbb{V}$ is a set of variables (non-terminals)
  - $\mathbb{T}$ is a set of constants (terminals)
  - $S \in \mathbb{V}$ is the start symbol
  - $\mathbb{P}$ is a set production rules: $u \mapsto v$ where u and v are strings of terminals and non-terminals and u has at least one non-terminal
* Sentential forms and sentences
* Derivations
* **Definition:** Language of a Grammar

$$L(G) = \left\{ w : S \overset{*}{\Rightarrow} w, \text{ where } w \in \Sigma* \right\}$$

* Linear grammars: At most one variable on right-hand side in any production rule
* Non-Linear, Right-Linear, Left-Linear Grammars
* **Definition:** Regular Grammars: A grammar is regular if it is right-linear or left-linear

May

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~
~~8~~	~~9~~	~~10~~	~~11~~	~~12~~	~~13~~	~~14~~
~~15~~	~~16~~	~~17~~	~~18~~	~~19~~	~~20~~	~~21~~
~~22~~	~~23~~	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~
~~29~~	~~30~~	~~31~~				

June

			~~1~~	~~2~~	~~3~~	~~4~~
~~5~~	~~6~~	~~7~~	~~8~~	~~9~~	~~10~~	~~11~~
~~12~~	~~13~~	~~14~~	~~15~~	~~16~~	~~17~~	~~18~~
~~19~~	~~20~~	21	22	23	24	25
26	27	28	29	30		

July

					1	2
3	4	5	6	7		

- **Theorem:** Regular grammars generate regular languages
  - Let G be right-linear, let $L((G)$ be the language generated by G, Construct NFA M such that $L(G) = L(M)$.

    Every variable X is a state in M. Create new final state F. For each production $S \to aA$, create edge from S to A labeled a. For each production $S \to A$, create edge from S to B labeled λ. For each production $B \to b$ create edge from B to F labeled b. For each production $A \to wB$ create edge from A to B labeled w, where w is a word (can add intermediate states)

    For left-linear grammar G, reduce it right-linear grammar G' show $L(G) = L(G')^R$
  - Let $L = L(M)$ be a regular language. Construct a right-linear grammar G such that $L(G) = L$.

    Each state q is a non-terminal. For each transition $\delta(q, a) = p$ create production $q \to ap$. For any final state q_f, create production $q_f \to \lambda$.

– Friday (midnight) Homework 1 due

May

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~
~~8~~	~~9~~	~~10~~	~~11~~	~~12~~	~~13~~	~~14~~
~~15~~	~~16~~	~~17~~	~~18~~	~~19~~	~~20~~	~~21~~
~~22~~	~~23~~	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~
~~29~~	~~30~~	~~31~~				

June

			~~1~~	~~2~~	~~3~~	~~4~~
~~5~~	~~6~~	~~7~~	~~8~~	~~9~~	~~10~~	~~11~~
~~12~~	~~13~~	~~14~~	~~15~~	~~16~~	~~17~~	~~18~~
~~19~~	~~20~~	21	22	23	24	25
26	27	28	29	30		

July

					1	2
3	4	5	6	7		

Week 2 Thursday, May 25:

- Linz 6th slide deck
  - Representations of regular languages: DFAs, NFAs, regular expressions, regular grammars (choose the one that best fits the problem to solve)
  - Regular languages are closed under: Union, Concatenation, Closure, Reverse, Complement, Intersection
  - Equivalent representations of regular languages: DFA, NFA, Regular expression, Regular grammar
  - **Decision Problems:** Let L be a regular languages
    - Is $w \in L$? (use DFA to check membership)
    - Is $L = \emptyset$? (search paths from q_0 to see if any lead to a final state q_f)
    - Is L finite? (Is there a cycle in a walk from the initial state to a final state)
    - Are two languages described by regular expressions equivalence?? Given regular languages L_1 and L_2, $L_1 = L_2$ if and only if $L_1 - L_2 = \emptyset$ and $L_2 - L_1 = \emptyset$. (Depth/Breadth=First) search decides if a DFA recognizes the empty language.
  - Non-regular languages (standard examples)
    - $L = \{a^n b^n : n \geq 0\}$
    - $L = \{ww^r\}$ (palindromes)

Consider 5 pigeons and 3 pigeonholes: Some hole must contain $\lceil 5/3 \rceil = 2$ or more pigeons and some hole must contain $\lfloor 5/3 \rfloor = 1$ or fewer pigeons.

  - Pigeonhole principle: If n pigeons are placed in m pigeon-hole, then some hole has $\lceil n/m \rceil$ or more pigeons and some hole has $\lfloor n/m \rfloor$ or fewer pigeons
  - If a DFA has n states then some state is visited at least twice in any walk on a string w with $|w| \geq n$
- **The Pumping Lemma:** Let L be a regular language. Assume a DFA that accepts L has m states ($|Q| = m$). Let $w \in L$. If $|w| \geq m$, then some state q is repeated on the walk that accepts w. Write $w = xyz$ where x labels the walk from $q_0 to$q,$z labels the walk from$q to $f \in \mathbb{F}$, and y labels the cycle from q to q. Then

$$|xy| \leq m \quad |y| \geq 1$$

  and

$$xy^iz \in L \quad \forall\, i \in \mathbb{N}$$

- Use the pumping lemma to show some languages are *not regular* (proofs by contradiction)
  - $L = \{a^n b^n : n \in \mathbb{N}\}$
  - $L = \{ww^R : w \in \Sigma^*\}$
  - $L = \left\{a^n b^\ell c^{n+\ell} : n, l \in \mathbb{N}\right\}$
  - $L = \{a^{n!} : n \in \mathbb{N}\}$
  - Let $n_x(w)$ count the number of character x in string w. Let $p \sqsubseteq w$ denote that p is a prefix of w. The language of balanced parentheses is

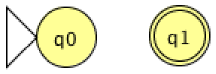
$$L = \{w : (n_a(w) = n_b(w)) \wedge (\forall p \sqsubseteq w)(n_a(v) \geq n_b(v))\}$$

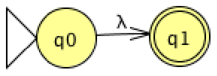
- Linz 7th slide deck
  - Lex: a lexical analyzer
    - Recognizes a string and takes and action (an important part of a compiler)
    - Converts a regular expression r into a NFA, then a DFA, then a minimal DFA

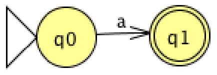
Week 3

- Tuesday, May 30:
  - Practice midterm
  - Linz 8th slide deck
    - Context-Free Languages: Pushdown Automata and Context-Free Grammars
    - Standard CFG Examples:
      - Language $\{a^n b^n : n \in \mathbb{N}\}$, Grammar $S \to aSb \mid \lambda$

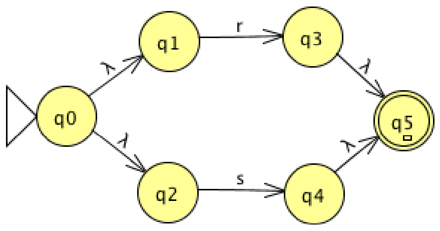
    · Palindromes $\{ww^R : w \in \Sigma^*\}$, Grammar $S \to aSa \mid bSb \mid \lambda$
    · Balanced Parentheses, Grammar $S \to (S) \mid SS \mid \lambda$
  - Grammar: $G = (\mathbb{V}, \mathbb{T}, S, \mathbb{P})$ where productions in $\mathbb{P}$ have the form
$$A \to x \quad \text{where } x \in (\mathbb{V} \cup \mathbb{T})^*$$
  - Derivation order: Leftmost or Rightmost
  - Derivation trees
  - Ambiguity: More than one leftmost (rightmost) derivation (operator precedence)
$$E \to E + E \mid\to E * E \mid\to (E) \mid\to a$$
  - Remove ambiguity by introducing new words/symbols: terms/T and factor/F
$$\begin{aligned} E &\to T \mid E + T \\ T &\to T * F \mid F \\ F &\to (E) \\ F &\to a \end{aligned}$$
  - Some CFL are ambiguous: $\{a^n b^n c^m\} \cup \{a^n b^m c^m\}$
  - CFG for arithmetic expressions
- Thursday, June 1:
  - Summary on regular languages (accepted by DFA).
    * NFA to DFA
      · NFA initial state q_0 becomes DFA initial state $\{q_0\}$
      · For any subset $\mathbb{A}$ of NFA states and character a compute DFA state
$$\mathbb{B} = \{q_k : \delta^*(q, a) = q_k, q \in \{A\}\}$$
      Add transition $\mathbb{A} \xrightarrow{a} \mathbb{B}$ to DFA
    * Regular expressions map to NFAs: Inductive proof (True for primitive regular expressions)

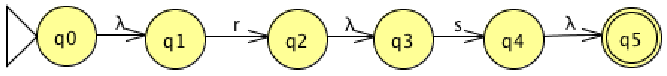


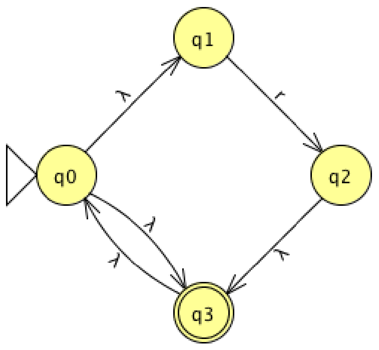




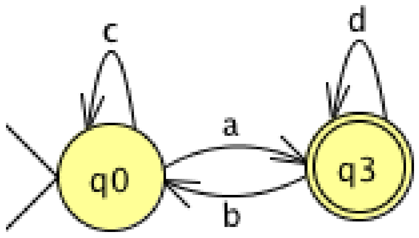
Construct NFAs for Union, Concatenation, and Star







* NFA to regular expression: Complete the transition graph by adding edges labeled ∅ where no edge exists. For a two-state graph



create regular expression

$$r = c^* a (d^* + bc^*)^*$$

To eliminate a (non-initial, non-final) state q_k introduce new edges labeled

$$\begin{aligned} &r_{00} + r_{0k} r_{kk}^* r_k 0 \\ &r_{0f} + r_{0k} r_{kk}^* r_{kf} \\ &r_{f0} + r_{fk} r_{kk}^* r_{k0} \\ &r_{ff} + r_{fk} r_{kk}^* r_{kf} \end{aligned}$$

Remove state q_k and all of its edges.

- DFA to regular (right-linear) grammar: States become non-terminals
  For transitions $\delta(q_i, a) = q_k$ create production $q_i \to a q_k$
  If q_k is a final state, create production $q_k \to \lambda$
- Regular grammar (right-linear) to DFA: Non-terminal S becomes DFA initial state.
  For productions $V_i \to a_1 a_2 \cdots a_j V_k$ create states and transitions labeled $a_1 a_2 \cdots aj$ from state V_i to V_k.
  For terminating productions $V_i \to a_1 a_2 \cdots a_j$ create states and transitions labeled $a_1 a_2 \cdots aj$ from state V_i to V_f (a final state)

– Linz 9th slide deck
  - Compilers
  - Simplifications of CFGs
    - Substitution Rule $(A \neq B)$

      $$(A \to xBy \quad B \to u \mid v \mid \cdots) \Rightarrow (A \to xuy \mid xvy \mid \cdots)$$

    - Useless non-terminals/productions (Variable A does not derive a terminal string or A cannot be reached from S)
      $V' = \emptyset$
      Repeat: Add to V' all variables A that derive terminal strings
      Until: V' does not change
      Construct Dependency Graph:
      State are variables
      Transitions from C to D if there is production $C \to xDy$
    - Nullable non-terminals
    - Unit productions
  - Linz 10th slide deck
    - Normal Forms for CFGs
      Chomsky Normal Form ($A \to BC$ and $A \to a$)
      Convert CFG to CNF

Step One: For productions $A \to x_1 x_2 \cdots x_n$ Replace by $A \to C_1 C_2 \cdots C_n$ where C_k is a new variable is $x_k \in T$.
Add production $C_k \to x_k$
Step Two: Replace productions $A \to C_1 C_2 \cdots C_n$ by sequence $A \to C_1 D_1$, $D_1 \to C_2 D_2, \ldots, Cn-2 \to C_{n-1} C_n$,
Palindromes to CNF:

$$S \to aSa \mid S \to bSb \mid S \to a \mid S \to b \mid S \to \lambda$$

Step One:

$$S \to ASA \mid S \to BSB \mid S \to A \mid S \to B \mid S \to \lambda \mid A \to a \mid \to b$$

Step Two:

$$S \to AC | C \to SA \mid S \to BD \mid D \to SB \mid S \to A \mid S \to B \mid S \to \lambda \mid A \to a \mid \to b$$

- Greibach Normal Form ($A \to ax$, where $x \in V^*$)
- Linz 10th slide deck
- CYK membership algorithm: Given Chomsky normal form grammar G, test if $w = a_1 a_2 \cdots a_n \in L(G)$.
  Let $V_{i,i} = \{A \in V : A \to a_i\}$ be the variables that produce a_i
  Let $V_{i,j} = \bigcup_{i<k<j} \left\{A : A \to BC \text{ where } B \in V_{i,k}, C \in V_{k+1,j}\right\}\}$ be the variables that produce $a_i \cdots a_j$
  Compute the table of $\binom{n+1}{2} = \frac{n(n+1)}{2} = O(n^2)$ values in row order (uses a dynamic programming algorithm paradigm)

$$\begin{array}{cccccc} V_{11} & V_{22} & \ldots & V_{n-2,n-2} & V_{n-1,n-1} & V_{nn} \\ V_{12} & V_{23} & \ldots & V_{n-2,n-1} & V_{n-1,n} & \\ V_{13} & V_{24} & \ldots & V_{n-2,n} & & \\ \vdots & & & & & \\ V_{1n} & & & & & \end{array}$$

  To compute V_{ij} requires scanning all productions $A \to BC$ to check if $B \in V_{ik}$ and $C \in V_{k+1,j}$ for $k = i+1, \ldots j-1$ (a total of $2(j-1-(i+1)+1) = 2(j-i-1) \leq 2n$ set membership tests). There are a fixed (constant) number of productions. Checking set membership can be executed in constant time
- Pushdown Automata:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$$

where

Q is a finite set of states
Σ is a finite (input) alphabet
Γ is a finite (stack) alphabet
$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow Q \times \Gamma*$
q_0 is the start state
$z \in \Gamma$ is the bottom of the stack
$F \subseteq Q$ are the final states

Week 4

- Tuesday, June 6:
  - Practice Midterm Key
  - Linz 11th slide deck
- Thursday, June 8:
  - Gradiance homework
  - Hopcroft-Ullman CFG Normal Forms slide deck
  - Hopcroft-Ullman Pushdown Automata slide deck
  - Hopcroft-Ullman Equivalence of CFG and PDA slide deck

Week 5

- Tuesday, June 13:
  - Linz 12th slide deck
  - Linz 13th slide deck
  - Linz 14th slide deck
- Wednesday, June 14: Homework 2 due
- Thursday, June 15:
  - Midterm Examination

Week 6

- Tuesday, June 20:
  - Linz 15th slide deck
  - Linz 16th slide deck
  - Linz 17th slide deck
  - Homework 3 due
- Thursday, June 22:
  - Linz 18th slide deck
  - Linz 18th+ slide deck
  - Linz 19th slide deck
  -

Week 7

- Tuesday, June 27:
  - Linz 20th slide deck
  - Proof That Computers Can't Do Everything (The Halting Problem)
- Thursday, June 29:
  - Homework 4 due

Week 8

- Tuesday, July 4: Independence Day
- Thursday, July 6: Final Examination

References

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. [page 1]

Linz, P. (2017). *An Introduction to Formal Languages and Automata, Sixth Edition*. Jones and Bartlett Publishers, Inc., USA, 6th edition. [page 1]