Today's Lecture: Proofs

Reading:

November 4: Proofs.

November 7: Modified Turing Machines
Read: *Kozen* lecture 30 (or *Sipser* 3.2).

November 9: Second Midterm: in class.

November 14: Diagonalization and the Halting Problem
Read: *Kozen* lecture 31 (or *Sipser* 4.2).

November 16: Decidability
Read: *Kozen* lecture 32 (or *Sipser* 4.1).

November 18: Review and Examples.

November 21: Reductions
Read: *Kozen* lecture 33 (or *Sipser* Chapter 5).

November 23: Gödel's Theorem
Read: *Kozen* lecture 38 (or *Sipser* 6.2).

November 25: Review and Examples.

November 28: Everything Else About Turing Machines.

November 30: Theorem Proving.

December 2: Something Fun.

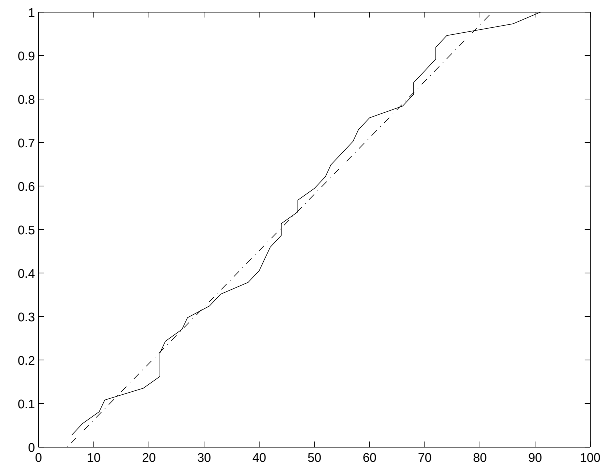

Figure 1: Distribution of midterm scores (cumulative fraction vs. score)

I. Midterm statistics

- **A.** Figure 1 shows the distribution.
- **B.** Mean $= 44.8$
- **C.** Median $= 44$
- **D.** Standard deviation $= 22.4$

II. Induction

- **A.** Induction on strings
  - **1.** Strings are defined inductively
    - **a.** ϵ is a string.
    - **b.** For any string x and symbol c, $x \cdot c$ is a string.
    - **c.** All strings can be derived by these two rules.
  - **2.** The standard induction schema
    - **a.** State your induction hypothesis. It should be a predicate over strings. Let $P : \Sigma^* \rightarrow \{0, 1\}$ be this predicate.
    - **b.** Base case: prove $P(\epsilon)$.
    - **c.** Induction step: Prove that $P(x) \Rightarrow P(x \cdot c)$ for any $x \in \Sigma^*$ and any $c \in \Sigma$. Because $x \in \Sigma^*$, about the only thing you can assume about it is that $P(x)$ holds.
  - **3.** Examples:
    - **a.** Prove that $(x \cdot y)^{\mathcal{R}} = y^{\mathcal{R}} \cdot x^{\mathcal{R}}$, where $x^{\mathcal{R}} = \mathsf{rev}(x)$ denotes the reverse of x (i.e. reverse the order of the symbols in x). This if from HW1, Q1, see page 13 of the solution set.
      - **i.** Induction Hypothesis: $(x \cdot y)^{\mathcal{R}} = y^{\mathcal{R}} \cdot x^{\mathcal{R}}$.

        In this case, the induction hypothesis is exactly the main result that we are trying to prove. In this class, you should always state your induction hypothesis. This makes it clear that you know what you are trying to accomplish. It also makes it easier to determine where you made a mistake (if you make one) and give appropriate partial credit.

ii. Base case – $y = \epsilon$:

$$\begin{array}{rcll} (x \cdot y)^{\mathcal{R}} & = & (x \cdot \epsilon)^{\mathcal{R}}, & \text{Base case hypothesis: } y = \epsilon \\ & = & x^{\mathcal{R}}, & x \cdot \epsilon = x \\ & = & \epsilon \cdot x^{\mathcal{R}}, & \epsilon \cdot x = x \\ & = & \epsilon^{\mathcal{R}} \cdot x^{\mathcal{R}}, & \epsilon = \epsilon^{\mathcal{R}} \\ & = & y^{\mathcal{R}} \cdot x^{\mathcal{R}}, & y = \epsilon \end{array}$$

In this argument, I showed all of the steps. A shorter version is also completely acceptable:
This follows directly from the facts that $x \cdot \epsilon = x = \epsilon \cdot x$ and $\epsilon^{\mathcal{R}} = \epsilon$.

iii. Induction step – assume for $y = z$, prove for $y = z \cdot c$:

$$\begin{array}{rcll} (x \cdot y)^{\mathcal{R}} & = & (x \cdot (z \cdot c))^{\mathcal{R}}, & \text{case hypothesis: } y = z \cdot c \\ & = & ((x \cdot z) \cdot c)^{\mathcal{R}}, & \cdot \text{ is associative} \\ & = & c \cdot (x \cdot z)^{\mathcal{R}}, & \text{def. rev} \\ & = & c \cdot (z^{\mathcal{R}} \cdot x^{\mathcal{R}}), & \text{induction hypothesis: } (x \cdot z)^{\mathcal{R}} = z^{\mathcal{R}} \cdot x^{\mathcal{R}} \\ & = & (c \cdot z^{\mathcal{R}}) \cdot x^{\mathcal{R}}, & \cdot \text{ is associative} \\ & = & (z \cdot c)^{\mathcal{R}} \cdot x^{\mathcal{R}}, & \text{def. rev} \\ & = & y^{\mathcal{R}} \cdot x^{\mathcal{R}}, & \text{case hypothesis: } y = z \cdot c \end{array}$$

Again, I showed all of the steps in detail. A shorter version is acceptable. The key point is that the proof should explicitly use the induction hypothesis In this case, the definition of rev, should also be used explicitly – this shows why the property that we claim holds for rev, but not for all imaginable functions over strings. With these things in mind, here's the short version:

$$\begin{array}{rcll} (x \cdot y)^{\mathcal{R}} & = & (x \cdot z \cdot y)^{\mathcal{R}}, & y = z \cdot c \\ & = & c \cdot (x \cdot z)^{\mathcal{R}}, & \text{def. rev} \\ & = & c \cdot z^{\mathcal{R}} \cdot x^{\mathcal{R}}, & \text{induction hypothesis} \\ & = & y^{\mathcal{R}} \cdot x^{\mathcal{R}}, & \text{def. rev}, y = z \cdot c \end{array}$$

In this shorter version, I've used properties such as the associativity of $\cdot$ without stating them explicitly, assuming that the reader (i.e. you) have seen enough examples with strings to take such things for granted.

b. Another example: HW1, solution set, page 6. A proof of a property of the "half" function.

4. Variations

The concatenation operator, $\cdot$, is associative: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$. This can be used to show that the following variations are also sound:

a. Replace the induction step with
prove that $P(x) \Rightarrow P(c \cdot x)$.

b. Replace the base case with
prove $P(\epsilon)$ and prove $\forall c \in \Sigma.\ P(c)$.
Then, replace the induction step with
prove $P(x) \wedge P(y) \Rightarrow P(x \cdot y)$ for arbitrary x and y in Σ^*.

B. Induction for regular languages

1. Induction on the sequence of states of a DFA or NFA

a. The operation of a DFA (or NFA) is defined inductively.

i. $\hat{\delta}(q, \epsilon) = q$

ii. $\hat{\delta}(q, x \cdot c) = \delta(\hat{\delta}(q, x), c)$

b. The standard induction schema

i. State your induction hypothesis. It should be a predicate over strings and states. Typically something of the form: $\hat{\delta}(q_0, x) = f(x)$ or $\hat{\delta}(q, x) = f(q, x)$, where δ is the state transition function for the DFA, and f expresses the key property that you want it to have. Let $P(q, x)$ be this predicate.

ii. Base case: prove $P(q, \epsilon)$.

iii. Induction step: prove $P(q,x) \Rightarrow P(q, x \cdot c)$ for any $x \in \Sigma^*$ and any $c \in \Sigma$.

c. Examples:

i. Kozen, pp. 19-21 (proof that a particular DFA accepts strings that represent multiples of three when interpretted as binary numbers).

ii. Kozen, lemma 4.1 (p. 23)

iii. Kozen, lemma 6.1 (p. 34)

iv. Kozen, lemma 6.2 (pp. 34-35)

v. Kozen, lemma 6.3 (p. 36)

vi. HW1, Q1 (page 2 of the solution set)

vii. HW1, Q3 (pages 6-8 of the solution set)

2. Induction on the structure of regular expressions

a. Regular expressions are defined inductively.

i. $\emptyset$, ϵ, and c (for any $c \in \Sigma$) are the basic regular expressions.

ii. The operators $+$, $\cdot$, and * construct more complicated regular expressions out of simpler ones.

b. The outline of an induction proof for regular expressions:

i. State the induction hypothesis.

ii. Three cases for the base step: $\emptyset$, ϵ, and c (for any $c \in \Sigma$).

iii. Three cases for the induction step $\alpha_1 + \alpha_2$, $\alpha_1 \cdot \alpha_2$, and α_1^*, where α_1 and α_2 are regular expressions. For at least one of these cases, you'll need to use the assumption that the induction hypothesis holds for α_1 and/or α_2; otherwise, this isn't an induction proof. Typically, you'll need the induction hypothesis for the proofs of each of these three cases.

c. Examples:

i. Solution set for HW1, Q1, "short solution" (pages 12-13).

ii. Kozen, theorem 8.1, proof that every regular expression can be recognized by a NFA, pages 46-47.

C. Induction for context-free languages

1. Induction on the sequence of configurations of a PDA

a. The operation of a PDA is defined inductively.

i. A configuration is a tuple, (p, x, α) where $p \in Q$ is the current state of the PDA, $x \in \Sigma^*$ is the unread portion of the input string, and $\alpha \in \Gamma^*$ is the string of symbols on the stack (leftmost symbol is the top-of-stack).

ii. Let M be a PDA. We say that

$$(q, cx, A\beta) \xrightarrow[M]{1} -(q', x, \mu\beta)$$

if $(q, c, A) \rightarrow (q', \mu) \in \delta$, where δ is the transition relation for the PDA. Likewise, we say that

$$(q, x, A\beta) \xrightarrow[M]{1} (q', x, \mu\beta)$$

if $(q, \epsilon, A) \rightarrow (q', \mu) \in \delta$.
We say that

$$(q, xy, \beta) \xrightarrow[M]{n+1} (q', y, \mu\nu)$$

iff there is some $c \in \Sigma \cup \{\epsilon\}$, $z \in \Sigma^*$, $p \in Q$, and $A \in \Gamma$ such that $x = zc$ and

$$(q, z, \beta) \xrightarrow[M]{n} (p, \epsilon, A\nu)$$

and $(p, c, A) \rightarrow (q', \mu) \in \delta$.
We write $(q, xy, \alpha) \xrightarrow[M]{*} (q', y, \beta)$ to denote that there exists some n such that $(q, xy, \alpha) \xrightarrow[M]{n} (q', y, \beta)$

iii. Note that $(q, xy, \alpha) \xrightarrow[M]{*} (q', y, \beta)$ holds iff there are configurations U_0, U_1, $\ldots U_k$ such that $U_0 = (q, xy, \alpha)$, $U_k = (q', y, \beta)$, and for all $i \in \{1 \ldots k\}$

$$U_{i-1} \xrightarrow[M]{1} U_i$$

Thus, we reason about the behaviour of PDAs by reasoning about sequences of configurations, and these sequences are defined inductively. Proofs about the behaviour of PDAs are naturally formulated using induction.

iv. M accepts x iff $(q_0, x, \perp) \xrightarrow[M]{*} (q', \epsilon, \beta)$ where $q' \in F$ if the M accepts by final state, and $\beta = \epsilon$ if M accepts by empty stack.

b. The standard induction schema

i. State your induction hypothesis. It should be a predicate over configurations. Typically something of the form:

$$(q_0, xy, \perp) \xrightarrow[M]{n} (q, y, \alpha) \quad \Leftrightarrow \quad P(q, x, y, \alpha)$$

for some predicate P.

ii. Base case: prove $P(q_0, \epsilon, x, \perp)$, in other words, show that your claim holds in the initial state of the machine when all of the input string, x, is unread.

iii. Induction step: Show that If $P(q, x, cz, A\beta)$ and $(q, c, A) \to (q', \gamma) \in \delta$ (for any $c \in \Sigma \cup \{\epsilon\}$, etc.), then $P(q', z, \gamma\beta)$.

c. Example: a PDA for balanced parentheses.

From Kozen example 23.1 (p. 161):

$$\begin{array}{rcl} M & = & (Q, \Sigma, \Gamma, \delta, q, \perp, \emptyset) \\ Q & = & \{q\} \\ \Sigma & = & \{[,]\} \\ \Gamma & = & \{\perp, [\} \\ \delta & = & \{\ (q, [, \perp) \to (q, [\ \perp), \\ & & \quad (q, [, [) \to (q, [[), \\ & & \quad (q,], [) \to (q, \epsilon), \\ & & \quad (q, \epsilon, \perp) \to (q, \epsilon) \\ & & \} \end{array}$$

This machine accepts on empty stack (as implied by the set of accepting states being empty).

i. Specifying the balance parentheses language.

Let B be the balanced parenthesis language. Let $\#\mathsf{left}(x)$ be the number of left-parenthesis symbols in x. Likewise, let $\#\mathsf{right}(x)$ be the number of right-parenthesis symbols. The string x is in the balanced parentheses language iff (see Kozen, lecture 20, p. 135):

- $\#\mathsf{left}(x) = \#\mathsf{right}(x)$.
- $\forall y, z.\ x = yz \Rightarrow \#\mathsf{left}(x) \geq \#\mathsf{right}(x)$.

In the following, we prove $L(M) = B$.

ii. We need to connect configurations of M with the number of left and right parentheses in the prefix of the input read up to that configuration. Let (q, z, α) be a configuration reached by reading y (i.e., if x ws the original input, then $x = yz$). We note that $\#\mathsf{left}(\alpha) = \#\mathsf{left}(y) - \#\mathsf{right}(y)$. Furthermore, the PDA can only reach this configuration if there was no prefix of y that had more fewer left parentheses than right.

We formalize this with the predicate:

$$\begin{array}{rl} & (q, yz, \perp) \xrightarrow[M]{*} (q, z, \alpha) \\ \Leftrightarrow & ((\#\mathsf{left}(y) - \#\mathsf{right}(z)) = \#\mathsf{left}(\alpha)) \wedge (\forall u, v.\ y = uv \Rightarrow \#\mathsf{left}(u) \geq \#\mathsf{right}(u)) \end{array}$$

Our proof, of course, is by induction.

iii. Induction hypothesis:

$$\begin{array}{rl} & (q, yz, \perp) \xrightarrow[M]{n} (q, z, \alpha) \\ \Leftrightarrow & \quad (\alpha = [^{\#\mathsf{left}(y) - \#\mathsf{right}(y)} \perp) \wedge (\forall u, v.\ y = uv \Rightarrow \#\mathsf{left}(u) \geq \#\mathsf{right}(u)) \\ \vee & \quad (\alpha = \epsilon) \wedge (z = \epsilon) \wedge (yz \in B) \end{array}$$

iv. Base case – $n = 0$:

$$(q, yz, \perp) \xrightarrow[M]{0} (q, z, \alpha) \quad \Leftrightarrow \quad (y = \epsilon) \wedge (\alpha = \perp)$$

It is straightforward to show that this satisfies the induction hypothesis.

v. Induction step – assume for n prove for $n+1$: Let x, the original input, equal ycz with $y, z \in \Sigma^*$, and $c \in \Sigma \cup \{\epsilon\}$. Let $A \in \Gamma$ and $\beta \in \Gamma^*$ such that

$$(q, ycz, \bot) \xrightarrow[M]{n} (q, cz, A\beta) \text{and} \quad (q, cz, A\alpha) \xrightarrow[M]{1} (q, z, \alpha)$$

We consider each possibility for this last transition:

$(q, [, \bot) \rightarrow (q, [\ \bot)$: We have $\beta = [^0\ \bot$, and the induction hypothesis yields $\#\mathsf{left}(y) - \#\mathsf{right}(y) = 0$. Furthermore $c = [$. Thus,

$$\begin{array}{rcll} \#\mathsf{left}(yc) - \#\mathsf{right}(yc) & = & \#\mathsf{left}(y[) - \#\mathsf{right}(y[), & c = [\\ & = & \#\mathsf{left}(y) + 1 - \#\mathsf{right}(y), & \text{def. } \#\mathsf{left} \text{ and } \#\mathsf{right} \\ & = & 1, & \#\mathsf{left}(y) - \#\mathsf{right}(y) = 0 \end{array}$$

Thus, the first clause of the induction hypothesis is satisfied.

The induction hypothesis further yields that all prefixes of y have at least as many left parentheses as right. Appending a left parenthesis preserves this property. Thus, the induction hypothesis is satisifed by this case.

As usual, I gave lots of details for this first example. I will give shorter explanations for the remaining cases to make it clear what is expected for homework or exam solutions.

$(q, [, [) \rightarrow (q, [[)$: An argument similar to the one above shows that yc has one more left parenthesis and the same number of right parentheses as y and that $\alpha = [\beta$. From these observations, it's straightforward to show that the induction hypothesis is maintained.

$(q,], [) \rightarrow (q, \epsilon)$: In this case, yc has one more right parenthesis than y. However, β had at least one left parenthesis; thus, y has more left parentheses than rights. This means that yc has at least as many left parentheses as rights. From these observations, it's straightforward to show that the induction hypothesis is maintained.

$(q, \epsilon, \bot) \rightarrow (q, \epsilon)$: In this case, $c = \epsilon$. By the induction hypothesis, $\#\mathsf{left}(y) = \#\mathsf{right}(y)$, and $\beta =\bot$. Thus, $\alpha = \epsilon$. By the induction hypothesis, all prefixes of y have at least as many left parentheses as right. Thus $y \in B$, and $ycz = y$ is in B as well. This satisfies the second disjunct of the induction hypothesis.

Finally, we have to show that if all prefixes of yc have at least as many left parentheses as right, then the machine can reach a configuration that satisfies the induction hypothesis. By the induction hypothesis, M was in such a configuration after reading y. If c is a left parenthesis, then because the stack is always of the form $[^*\ \bot$ (by the induction hypothesis), a transition is allowed. Likewise, if c is a right parenthesis, then if yc has at least as many left parentheses as right, then y had more lefts than rights, and the top of stack symbol was a $[$ in the configuration after reading y. Thus, M can make a transition.

vi. Completing the proof. Having completed the induction lemma, the proof that $L(M) = B$ is simple. Let $w \in B$. By the induction result above,

$$\begin{array}{rl} & (q, cz, \bot) \xrightarrow[M]{*} (q, \epsilon, \bot) \\ \text{or} & (q, cz, \bot) \xrightarrow[M]{*} (q, \epsilon, \epsilon) \end{array}$$

In the second case, $w \in L(M)$. In the former, M can make the ϵ move $(q, \epsilon, \bot) \rightarrow (q, \epsilon)$ and accept w.

Conversely, let $w \in L(M)$. This means that

$$(q, cz, \bot) \xrightarrow[M]{*} (q, \epsilon, \epsilon)$$

and by the induction result, we have that $w \in B$.

2. Induction on the steps of a derivation of a CFG

a. The proof schema

i. Induction Hypothesis: this will make some assertion about the strings in partial derivations. These strings are mixtures of terminals and non-terminals.

ii. Base case – zero steps: you must show that the induction hypothesis holds for the start symbol.

iii. Induction step – assume for n steps prove for $n+1$: show that each production of the grammar preserves the induction hypothesis.

b. Important note: note that induction on the steps of the derivation is **not** the same as induction on the length of the final string. With a CFG, it is not the case that all strings of length $n+1$ are obtained by deriving a string of length n and appending one terminal. Once a string consisting entirely of terminals has been derived, no further productions can be applied – productions only apply to non-terminals.
More concretely, consider the balanced parenthesis language. How many strings are in this language of length 11? None. Every string in the balanced parenthesis language has an equal number of left and right parentheses and thus an even number of terminals total. Thus, we can't obtain the strings of length 12 (there are 394) by starting with a non-existent string of length 11 and adding a non-terminal.
When reasoning about a CFG, the induction will be over the steps of the derivation. As noted above, the induction hypothesis will typically make an assertion about the relationships that must hold between the various terminals and non-terminals in the partial derivations at each step along the way.

c. Examples:

i. Kozen theorem 20.1 (proving that a CFG generates the balanced parentheses language, p. 136 – 139).

ii. Oct. 26 midterm, problem 3.

III. The pumping lemmas

A. The pumping lemma for regular languages

1. Formal statement: If B is a regular langauge, then there exist some integer k, such that for every string, $xyz \in B$ with $|y| \geq k$, there exist strings u, v, w with $uvw = y$ and $|v| \geq 1$ such that $xuv^iwz \in B$ for any $i \geq 0$.

2. How to use the pumping lemma to show that a language is *not* regular (see Kozen's "game with a demon" p. 70-71):

a. Find a way to generate a string, xyz for any proposed k, such that y has no substring that can be repeated an arbitrary number of times and still keep the string in the language.

b. A suitable answer can be along the lines of:
"Let $w = a^k(bc)^{2k+1}d^{k-2}$. Force the demon to pump the a^k portion. A sentence or two to explain why this produces a string not in B."
Note that $i = 0$ and $i = 2$ are common values for showing that the string is not in the language, but any value will work (e.g. 4, 17, k^2, etc.).

c. Note that the string that you present to the demon *must* depend on the value of k.

3. Examples

a. Kozen example 12.1 (a^nb^n, p. 72)

b. Kozen example 12.2 ($a^{n!}$, p. 73)

c. Homework 2, question 1.

d. Oct. 26 midterm, question 1.b.

B. An alternative to pumping

1. Although we didn't cover it in class, many solutions that I've received to various problems attempt to use the following property of regular langauges. I'll state it explicitly so you can use it from now on.

2. Let B be a language. We write $x \equiv_B y$ iff for all strings z, $xz \in B \Leftrightarrow yz \in B$. The language B is regular iff it has a finite set of equivalence classes.

3. Example (Oct. 26 midterm, Q1.b): show that $\{w|\#\mathsf{a}(w) - \#\mathsf{a}(w) < 3\}$ is not regular.

a. For $i \geq 1$, let $w_i = a^{2+i}$.

b. Note that $w_i \notin B$, but w_ib^i is in B.

c. Thus, all of the w_i are in different classes. In particular, consider $i < j$. Then, $w_ib^i \in B$ but $w_jb_i \notin B$. Thus, w_i and w_j are distinguishable. There are an infinite number of choices for i. Thus, B is not regular.

4. A general remark

a. This approach can also help you decide whether or not a language is regular. If you think that it's regular, try to identify a finite set of equivalence classes for the language. Conversely, if you think that it's not regular, try to find an infinite set of string such that no two are in the same equivalence class.

b. Example: figure out what the equivalence classes are for each of the languages from the Oct. 26 midterm, Q1.

C. The pumping lemma for context-free languages

1. Formal statement: If B is a context-free langauge, then there exist some integer k, such that for every string, $z \in B$ with $|z| \geq k$, there exists strings u, v, w, x, y with $uvwxy = z$, $|vx| \geq 1$, and $|vwx| \leq k$ such that $uv^iwx^iz \in B$ for any $i \geq 0$.

2. How to use the pumping lemma to show that a language is *not* context-free (see Kozen's "game with a demon" p. 153-154):

a. Find a way to generate a string, z for any proposed k, such that y has no pair of substrings (i.e. v and x) that are close (i.e. $|vwx| \leq k$) and can be repeated an arbitrary number of times and still keep the string in the language.

b. A suitable answer can be along the lines of:
"Let $z = a^k(bc)^{2k+1}d^{k-2}$. Consider $uvwxy = z$, such that $uv^iwx^iz \in B$ for all i. To keep the number of a'ss, bc's, and d's suitably balanced (the details depend on the particular language, B). Therefore, v must contain some a's and x must contain some d's. But this means $|vwx| \geq k+3$ which violates the conditions of the pumping lemma. Thus, z cannot be pumped and B is not context free.

c. Note that the string that you present to the demon *must* depend on the value of k.

3. Examples

a. Kozen example 22.3 ($a^nb^na^n$, p. 154)

b. Kozen example 22.4 (ww, pp. 154-155)

c. Homework 3, question 3.

IV. Closure properties

A. The regular languages are closed under:

1. union, intersection, and complement (and therefore under arbitrary boolean/set operations)

2. concatenation and asteration

3. homomorphisms and inverse homomorphisms

4. plus a numerous other operations that we've seen at one point or another such as:

a. reversal

b. half, middle-third, shuffle, etc.

B. The context-free languages are closed under

1. union

2. concatenation and asteration

3. homomorphisms and inverse homomorphisms

C. A few more remarks

1. Relationships between the language classes

a. Every DCFL (deterministic CFL) is a CFL.

b. Every regular language is a DCFL (and therefore a CFL).

c. There are CFLs that are not DCFLs (e.g. $\{x | x = ww\}$), and DCFLs that are not regular (e.g. a^nb^n). Thus, these containments are strict.

2. What is a class of languages?

a. An alphabet, Σ, is a *finite* set of symbols.

b. Σ^* is the set of all strings formed by symbols in Σ.

c. A language is a subset of Σ^*.

d. A class of languages (e.g. the regular languages, or the context-free languages) is a subset of 2^{Σ^*}.

3. One final clarification

a. The regular languages are a subset of the context free languages. Both classes are subsets of 2^{Σ^*}.

b. Let B_R be a regular language, and B_{CFL} be context-free. It can well be the case that $B_{CFL} \subset B_R$. This does not contradict the fact that the set of regular languages is a subset of the set of context-free languages.

c. In particular, Σ^* is regular. Every context-free language is a subset of Σ^*.

d. Thus, you cannot prove that B is not regular by showing some language $\tilde{B}$ such that $\tilde{B} \subseteq B$ and $\tilde{B}$ is not regular.

e. Note that Σ^* and $\emptyset$ are both regular. CFLs give us sets that can make finer distinctions between strings than regular languages can. We'll show shortly that Turing machines can make finer distinction yet.
You can think of the more powerful models as being able to resolve more detail to decide which strings are in and which are out. You can always find a regular language that contains any other language (e.g. Σ^* contains all languages). Likewise, you can find a regular language that is contained in any other language (e.g. $\emptyset$). But, these may be very coarse approximations of the language that you actually want.