UCLA COMPUTER SCIENCE FIELD NOTES

CS 181

Theory of Computing and Automata

Regular languages, context-free languages, Turing machines, decidability, and computability

How these notes are written. I wrote each chapter the way I wish the material had been explained the first time: the real idea, the point where I usually get stuck, the simplest mental model that still stays honest, and a worked decision instead of a polished answer appearing from nowhere. Every chapter closes with practice and a fully written solution. If a sentence sounds impressive but does not help me solve or explain something, it does not belong here.

Daniel Mastick

Curriculum map, working notes, practice, and solutions

Course map

Week	Core thread	What should be solid by the end
1	**Languages, proofs, and finite automata**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
2	**Regular expressions and closure**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
3	**Nonregularity and pumping**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
4	**Context-free grammars and parse structure**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
5	**Pushdown automata and CFL properties**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
6	**Turing machines and recognizability**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
7	**Reductions and undecidability**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
8	**Computability boundaries and classification**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
9	**Integration workshop**	Connect at least three chapters in one end-to-end problem and explain where their contracts meet.
10	**Synthesis and project review**	Audit assumptions and explain a complete solution from interface to failure handling.

Scope anchor: https://catalog.registrar.ucla.edu/course/2025/COMSCI181. The sequence is aligned to the official catalog description and expanded into a practical ten-week study plan.

How I would use this packet

Treat each numbered section as one chapter. Read the formal explanation, pause at the student interpretation, and see whether the easy version still says the same thing. Rework the example without looking, then cover the solution in the chapter practice box. The cumulative set at the end is deliberately mixed: knowledge becomes durable when I have to choose the tool instead of being told which chapter I am in.

Contents

CHAPTER 1 Languages, proofs, and finite automata

The idea

A language is a set of strings. DFAs have one transition per symbol; NFAs permit several and epsilon moves but recognize the same regular languages. Extended transition functions formalize complete-string behavior.

Rebuild the chapter from first principles

The claim I need to own is this: A language is a set of strings. DFAs have one transition per symbol; NFAs permit several and epsilon moves but recognize the same regular languages. Extended transition functions formalize complete-string behavior. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of languages, proofs, and finite automata. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach languages, proofs, and finite automata on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from the course foundations to regular expressions and closure. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** A language is a set of strings.
- **Mechanism.** DFAs have one transition per symbol; NFAs permit several and epsilon moves but recognize the same regular languages.
- **Boundary.** Extended transition functions formalize complete-string behavior.
- **Decision test.** I should be able to say when languages, proofs, and finite automata is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I separate the machine drawing from the proof of what its state means.

The easy version

A language is a set of strings.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Convert an NFA to a DFA. The conclusion is Each DFA state is a set of possible NFA states. Start with epsilon closure, transition the whole set on each symbol, close again, and mark sets containing an accepting NFA state.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Convert an NFA to a DFA.
Solution. Each DFA state is a set of possible NFA states. Start with epsilon closure, transition the whole set on each symbol, close again, and mark sets containing an accepting NFA state.

Practice 2. Give a short oral explanation of languages, proofs, and finite automata that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct languages, proofs, and finite automata from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a languages, proofs, and finite automata problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 2 Regular expressions and closure

The idea

Regular expressions, DFAs, and NFAs are equivalent descriptions. Product constructions prove closure under union, intersection, and difference; state elimination or structural construction moves between regex and machines.

Rebuild the chapter from first principles

The claim I need to own is this: Regular expressions, DFAs, and NFAs are equivalent descriptions. Product constructions prove closure under union, intersection, and difference; state elimination or structural construction moves between regex and machines. I do not count that as learned just because the sentence

looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of regular expressions and closure. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach regular expressions and closure on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from languages, proofs, and finite automata to nonregularity and pumping. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Regular expressions, DFAs, and NFAs are equivalent descriptions.
- **Mechanism.** Product constructions prove closure under union, intersection, and difference; state elimination or structural construction moves between regex and machines.
- **Boundary.** The useful test is whether I can apply regular expressions and closure to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when regular expressions and closure is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

My shorthand for this chapter is: my shorthand for this chapter is: closure proofs are recipes for building a recognizer, not just facts to memorize.

The easy version

Regular expressions, DFAs, and NFAs are equivalent descriptions.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Build a DFA for intersection of L1 and L2. The conclusion is Use state pairs (q1,q2), advance both components on each symbol, and accept only pairs where both component states accept.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Build a DFA for intersection of L1 and L2.
Solution. Use state pairs (q1,q2), advance both components on each symbol, and accept only pairs where both component states accept.

Practice 2. Give a short oral explanation of regular expressions and closure that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that

licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct regular expressions and closure from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a regular expressions and closure problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 3 Nonregularity and pumping

The idea

The pumping lemma states a necessary repetition property of every infinite regular language. To disprove regularity, choose a witness after the pumping length and defeat every legal split and pump choice.

Rebuild the chapter from first principles

The claim I need to own is this: The pumping lemma states a necessary repetition property of every infinite regular language. To disprove regularity, choose a witness after the pumping length and defeat every legal split and pump choice. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of nonregularity and pumping. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.

4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach nonregularity and pumping on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from regular expressions and closure to context-free grammars and parse structure. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** The pumping lemma states a necessary repetition property of every infinite regular language.
- **Mechanism.** To disprove regularity, choose a witness after the pumping length and defeat every legal split and pump choice.
- **Boundary.** The useful test is whether I can apply nonregularity and pumping to a new case and explain the assumption that makes the answer valid.

- **Decision test.** I should be able to say when nonregularity and pumping is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I keep the quantifier order visible; choosing the split myself proves nothing.

The easy version

The pumping lemma states a necessary repetition property of every infinite regular language.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Use pumping to show {0^n1^n} is not regular. The conclusion is Choose 0^p1^p. Any first p-symbol pump segment contains only zeros. Pumping changes the zero count without changing ones, leaving the language and contradicting the lemma.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Use pumping to show {0^n1^n} is not regular.
Solution. Choose 0^p1^p. Any first p-symbol pump segment contains only zeros. Pumping changes the zero count without changing ones, leaving the language and contradicting the lemma.

Practice 2. Give a short oral explanation of nonregularity and pumping that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct nonregularity and pumping from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a nonregularity and pumping problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent

check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 4 Context-free grammars and parse structure

The idea

CFGs generate strings through variables and productions. Leftmost derivations and parse trees expose hierarchical structure; ambiguity means one string has multiple parse trees. Normal forms support proofs and parsing algorithms.

Rebuild the chapter from first principles

The claim I need to own is this: CFGs generate strings through variables and productions. Leftmost derivations and parse trees expose hierarchical structure; ambiguity means one string has multiple parse trees. Normal forms support proofs and parsing algorithms. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of context-free grammars and parse structure. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.

- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach context-free grammars and parse structure on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from nonregularity and pumping to pushdown automata and cfl properties. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** CFGs generate strings through variables and productions.
- **Mechanism.** Leftmost derivations and parse trees expose hierarchical structure; ambiguity means one string has multiple parse trees.
- **Boundary.** Normal forms support proofs and parsing algorithms.
- **Decision test.** I should be able to say when context-free grammars and parse structure is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I ask what substring each variable is responsible for generating.

The easy version

CFGs generate strings through variables and productions.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Give a grammar for balanced parentheses. The conclusion is S -> SS | (S) | epsilon generates concatenation and nesting. Prove soundness

by induction on derivations and completeness by splitting any nonempty balanced string at its matching structure.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Give a grammar for balanced parentheses.
Solution. S -> SS | (S) | epsilon generates concatenation and nesting. Prove soundness by induction on derivations and completeness by splitting any nonempty balanced string at its matching structure.

Practice 2. Give a short oral explanation of context-free grammars and parse structure that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct context-free grammars and parse structure from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a context-free grammars and parse structure problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 5 Pushdown automata and CFL properties

The idea

A PDA adds an unbounded stack to finite control and recognizes context-free languages. Nondeterminism matters for PDAs. CFLs have their own pumping lemma and are closed under some, but not all, Boolean operations.

Rebuild the chapter from first principles

The claim I need to own is this: A PDA adds an unbounded stack to finite control and recognizes context-free languages. Nondeterminism matters for PDAs. CFLs have their own pumping lemma and are closed under some, but not all, Boolean operations. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of pushdown automata and cfl properties. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach pushdown automata and cfl properties on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from context-free grammars and parse structure to turing machines and recognizability. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** A PDA adds an unbounded stack to finite control and recognizes context-free languages.
- **Mechanism.** Nondeterminism matters for PDAs.
- **Boundary.** CFLs have their own pumping lemma and are closed under some, but not all, Boolean operations.
- **Decision test.** I should be able to say when pushdown automata and cfl properties is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

My shorthand for this chapter is: my shorthand for this chapter is: the stack is memory with one accessible end, which is exactly enough for nested structure.

The easy version

A PDA adds an unbounded stack to finite control and recognizes context-free languages.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Describe a PDA for 0^n1^n. The conclusion is Push one marker per zero, nondeterministically or on first one switch phases, pop one per one, and accept only when input and marker stack finish together.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Describe a PDA for 0^n1^n.
Solution. Push one marker per zero, nondeterministically or on first one switch phases, pop one per one, and accept only when input and marker stack finish together.

Practice 2. Give a short oral explanation of pushdown automata and cfl properties that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct pushdown automata and cfl properties from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a pushdown automata and cfl properties problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 6 Turing machines and recognizability

The idea

A Turing machine has finite control and an unbounded read-write tape. Deciders halt on every input; recognizers may loop on nonmembers. Multitape, nondeterministic, and high-level variants preserve computability.

Rebuild the chapter from first principles

The claim I need to own is this: A Turing machine has finite control and an unbounded read-write tape. Deciders halt on every input; recognizers may loop on nonmembers. Multitape, nondeterministic, and high-level variants preserve computability. I do not count that as learned just because the sentence looks

familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of turing machines and recognizability. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach turing machines and recognizability on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from pushdown automata and cfl properties to reductions and undecidability. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** A Turing machine has finite control and an unbounded read-write tape.
- **Mechanism.** Deciders halt on every input; recognizers may loop on nonmembers.
- **Boundary.** Multitape, nondeterministic, and high-level variants preserve computability.
- **Decision test.** I should be able to say when turing machines and recognizability is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I distinguish 'does not accept' from 'halts and rejects'; that difference drives recognizability.

The easy version

A Turing machine has finite control and an unbounded read-write tape.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why are deciders closed under complement? The conclusion is Run the decider and swap accept with reject. Because it always halts, the complemented machine also decides and cannot get stuck on a former nonmember.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why are deciders closed under complement?
Solution. Run the decider and swap accept with reject. Because it always halts, the complemented machine also decides and cannot get stuck on a former nonmember.

Practice 2. Give a short oral explanation of turing machines and recognizability that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct turing machines and recognizability from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a turing machines and recognizability problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 7 Reductions and undecidability

The idea

A mapping reduction computes f such that x is in A exactly when f(x) is in B. It transfers impossibility: if A is undecidable and A reduces to B, then B is undecidable. Diagonalization and self-reference power canonical proofs.

Rebuild the chapter from first principles

The claim I need to own is this: A mapping reduction computes f such that x is in A exactly when f(x) is in B. It transfers impossibility: if A is undecidable and A reduces to B, then B is undecidable. Diagonalization and self-reference power canonical proofs. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of reductions and undecidability. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach reductions and undecidability on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from turing machines and recognizability to computability boundaries and classification. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** A mapping reduction computes f such that x is in A exactly when f(x) is in B.
- **Mechanism.** It transfers impossibility: if A is undecidable and A reduces to B, then B is undecidable.
- **Boundary.** Diagonalization and self-reference power canonical proofs.
- **Decision test.** I should be able to say when reductions and undecidability is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I read the reduction direction as a hypothetical solver: a B solver would give me an A solver.

The easy version

A mapping reduction computes f such that x is in A exactly when f(x) is in B.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Outline the halting-problem contradiction. The conclusion is Assume H decides whether machine M halts on input w. Construct D that loops when H says D halts on itself and halts otherwise. Running D on D contradicts either answer.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Outline the halting-problem contradiction.
Solution. Assume H decides whether machine M halts on input w. Construct D that loops when H says D halts on itself and halts otherwise. Running D on D contradicts either answer.

Practice 2. Give a short oral explanation of reductions and undecidability that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct reductions and undecidability from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a reductions and undecidability problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 8 Computability boundaries and classification

The idea

Decidable, recognizable, co-recognizable, and unrecognizable languages form different classes. Rice's theorem makes every nontrivial semantic property of recognized languages undecidable; syntactic properties may remain decidable.

Rebuild the chapter from first principles

The claim I need to own is this: Decidable, recognizable, co-recognizable, and unrecognizable languages form different classes. Rice's theorem makes every nontrivial semantic property of recognized languages undecidable; syntactic properties may remain decidable. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of computability boundaries and classification. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach computability boundaries and classification on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from reductions and undecidability to the cumulative course model. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Decidable, recognizable, co-recognizable, and unrecognizable languages form different classes.
- **Mechanism.** Rice's theorem makes every nontrivial semantic property of recognized languages undecidable; syntactic properties may remain decidable.
- **Boundary.** The useful test is whether I can apply computability boundaries and classification to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when computability boundaries and classification is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I ask whether a property concerns program text or what the program ultimately computes.

The easy version

Decidable, recognizable, co-recognizable, and unrecognizable languages form different classes.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why is 'does this TM recognize an empty language?' undecidable? The conclusion is Emptiness is a nontrivial semantic property of the recognized language, so Rice's theorem applies. It can also be shown by reduction from acceptance.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why is 'does this TM recognize an empty language?' undecidable?
Solution. Emptiness is a nontrivial semantic property of the recognized language, so Rice's theorem applies. It can also be shown by reduction from acceptance.

Practice 2. Give a short oral explanation of computability boundaries and classification that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct computability boundaries and classification from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a computability boundaries and classification problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

Final study check

I should be able to close this packet and reconstruct the core models, not merely recognize their names. My final check is to explain one complete problem aloud: what the inputs mean, which invariant or contract controls the work, why the method is legal, what it costs, how it can fail, and what evidence would convince another engineer.