Recitation 9 - Homework 4 and Turing Machine Problems

John Chilton

June 8, 2007

- Recognizability versus. Decidability
- Homework 3
- Turing Machine Diagram

Understanding these two concepts and the distinction is important for this assignment.

Problem 7. (Problem 3.16 from text) Show that Turing-recognizable languages are closed under: concatenation (2), star (3), and intersection (1).

Start: Let A_1 and A_2 be two Turing-recognizable languages, and let M_1 and M_2 be two Turing-machines that recognize the respective languages.

Start: Let A_1 and A_2 be two Turing-recognizable languages, and let M_1 and M_2 be two Turing-machines that recognize the respective languages.

- Construct Turing machine M that recognizes $A_1 \cap A_2$ using M_1 and M_2.
- Construct Turing machine M that recognizes $A_1 \circ A_2$ using M_1 and M_2.
- Construct Turing machine M that recognizes A_1^* using M_1.
- Remember M_1 and M_2 recognize, not decide A_1 and A_2.

Start: Let A_1 and A_2 be two Turing-recognizable languages, and let M_1 and M_2 be two Turing-machines that recognize the respective languages.

- Construct Turing machine M that recognizes $A_1 \cap A_2$ using M_1 and M_2.
- Construct Turing machine M that recognizes $A_1 \circ A_2$ using M_1 and M_2.
- Construct Turing machine M that recognizes A_1^* using M_1.
- Remember M_1 and M_2 recognize, not decide A_1 and A_2.

Don't use diagram or implementation level description. Use pseudo code, examples on page 153, and 163 toward bottom.

Examples of union for Scheme for deciding and recognizing.

Let A_1 and A_2 be two Turing-*decidable* languages, and let M_1 and M_2 be two Turing-machines that *decide* the respective languages. The following machine M decides $A_1 \cup A_2$, hence Turing-decidable languages are closed under unioning.

$M :=$ "On input w:

- Run M_1 on input w, if it accepts, *accept*
- Run M_2 on input w, if it accepts, *accept*
- Else, *reject*."

Let A_1 and A_2 be two Turing-*recognizable* languages, and let M_1 and M_2 be two Turing-machines that *recognize* the respective languages. The following machine M recognizes $A_1 \cup A_2$, hence Turing-recognizable languages are closed under unioning.

$M :=$ "On input w:

- Repeat the following for $i = 1, 2, 3, \ldots$
-     Run M_1 on input w for i steps, if it accepts, *accept*
-     Run M_2 on input w for i steps, if it accepts, *accept*"

Let A_1 and A_2 be two Turing-*recognizable* languages, and let M_1 and M_2 be two Turing-machines that *recognize* the respective languages. The following machine M recognizes $A_1 \cup A_2$, hence Turing-recognizable languages are closed under unioning.

$M :=$ "On input w:

- Repeat the following for $i = 1, 2, 3, \ldots$
- Run M_1 on input w for i steps, if it accepts, *accept*
- Run M_2 on input w for i steps, if it accepts, *accept*"

Could also add "If on i^{th} step both machines reject and halt, *reject*."

Problem 1. Take the given implementation level description and construct a Turing machine diagram out of it. Then explicitly describe Q, Σ, Γ, mention that δ is described in your diagram, and specify your start, accept, and reject state.

Problem 2 (Exercise 3.8b). Give an implementation-level description of a Turing machine which decidesthe following language.

$$\{w \mid w \text{ contains twice as many 0s and 1s}\}$$

We talked about an approach for doing something like this last week. Take this approach and adapt it.

Problem 3 (Exercise 3.6). Theorem 3.21 states a language is Turing recognizable iff some enumerator enumerates it. Part of the proof was to construct an enumerator to enumerate the language recognized by some Turing machine M.

Problem 3 (Exercise 3.6). Theorem 3.21 states a language is Turing recognizable iff some enumerator enumerates it. Part of the proof was to construct an enumerator to enumerate the language recognized by some Turing machine M.

An enumerator is like a Turing machine, but instead of accepting or rejecting it prints out the strings of the language it enumerates. If E enumerates A it will only print out strings in A and given enough time it will print out any given string in A.

M *recognizes* A, $E_{:(}$ doesn't enumerate A, but $E_{:)}$ does. Why?

$E_{:(}$ = Ignore input.
1. Repeat for each string $s_i = s_1, s_2, s_3, \ldots$
2. Run M on s_i, if it accepts, print s_i

$E_{:)}$ = Ignore input.
1. Repeat for each string $i = 1, 2, 3,, \ldots$
2. Run M on $s_1, s_2, \ldots, s_i$ for i steps
3. Print each of the strings that are accepted, if any

Problem 4. Explain why the following is not a description of a legitimate Turing machine.

$M_{\text{bad}} =$ The input is a polynomial p over variables $x_1, \ldots, x_k$.

1. Try all possible settings of $x_1, \ldots, x_k$ to integer values.
2. Evaluate p on all of these settings.
3. If any of these settings evaluates to 0, *accept*; else, *reject*.

- The difference between the "right" and "wrong" answer is very subtle.

- The difference between the "right" and "wrong" answer is very subtle.
- Be as precise as possible, and pick your words carefully.

- The difference between the "right" and "wrong" answer is very subtle.
- Be as precise as possible, and pick your words carefully.
- Don't say the Turing machine cannot do something you could do in Java.

- The difference between the "right" and "wrong" answer is very subtle.
- Be as precise as possible, and pick your words carefully.
- Don't say the Turing machine cannot do something you could do in Java.
- Don't say the Turing machine cannot do something you cannot do in Java because of memory limitations or limitations on sizes of integers.

Problem 5 and 6 are the hard ones, be sure to do these last.

Problem 5. k-PDAs.

- A k-PDA is just like a PDA, but with k unique stacks.

Problem 5. k-PDAs.

- A k-PDA is just like a PDA, but with k unique stacks.
- At each step, you can pop an item or push and item or both or neither for each stack.

Problem 5. k-PDAs.

- A k-PDA is just like a PDA, but with k unique stacks.
- At each step, you can pop an item or push and item or both or neither for each stack.
- Still have a set of states, still have just one input.

Problem 5. k-PDAs.

- A k-PDA is just like a PDA, but with k unique stacks.
- At each step, you can pop an item or push and item or both or neither for each stack.
- Still have a set of states, still have just one input.
- When $k = 0$, this is just a NFA, $k = 1$ its just a normal PDA

Problem 5. k-PDAs.

- A k-PDA is just like a PDA, but with k unique stacks.
- At each step, you can pop an item or push and item or both or neither for each stack.
- Still have a set of states, still have just one input.
- When $k = 0$, this is just a NFA, $k = 1$ its just a normal PDA
- Example?

Two steps:

- Show how a TM can simulate a k-PDA.
- Show a 2-PDA can simulate a Turing-machine

Show a TM can simulate a k-PDA

- Given some k-PDA, P, describe how to construct a TM, M, the recognizes the same language.

Show a TM can simulate a k-PDA

- Given some k-PDA, P, describe how to construct a TM, M, the recognizes the same language.
- Use a multitape turning machine. How many tapes should you use?

Show a TM can simulate a k-PDA

- Given some k-PDA, P, describe how to construct a TM, M, the recognizes the same language.
- Use a multitape turning machine. How many tapes should you use?
- k-PDAs read in input, M will already have its input on first tape, how will you simulate reading input?

Show a TM can simulate a k-PDA

- Given some k-PDA, P, describe how to construct a TM, M, the recognizes the same language.
- Use a multitape turning machine. How many tapes should you use?
- k-PDAs read in input, M will already have its input on first tape, how will you simulate reading input?
- What should the set of states in M be?

Show a TM can simulate a k-PDA

- Given some k-PDA, P, describe how to construct a TM, M, the recognizes the same language.
- Use a multitape turning machine. How many tapes should you use?
- k-PDAs read in input, M will already have its input on first tape, how will you simulate reading input?
- What should the set of states in M be?
- When should you move to an accept state, explain how rejecting will work.

Show a TM can simulate a k-PDA

- Given some k-PDA, P, describe how to construct a TM, M, the recognizes the same language.
- Use a multitape turning machine. How many tapes should you use?
- k-PDAs read in input, M will already have its input on first tape, how will you simulate reading input?
- What should the set of states in M be?
- When should you move to an accept state, explain how rejecting will work.
- How do you pop, push, and replace items on a stack.

Show a TM can simulate a k-PDA

- Given some k-PDA, P, describe how to construct a TM, M, the recognizes the same language.
- Use a multitape turning machine. How many tapes should you use?
- k-PDAs read in input, M will already have its input on first tape, how will you simulate reading input?
- What should the set of states in M be?
- When should you move to an accept state, explain how rejecting will work.
- How do you pop, push, and replace items on a stack.
- Lots of intresting little issues, be sure to fully describe the Turing-machine M and explain or better yet prove why it accepts the same language as P.

Show a 2-PDA can simulate a TM.

- Given some Turning machine M, construct a 2-PDA, P, that recognizes the same language as M.

Show a 2-PDA can simulate a TM.

- Given some Turning machine M, construct a 2-PDA, P, that recognizes the same language as M.
- PDA uses it stack to simulate the tape, one stack for what is right of the head and one stack for what is left of the head.

Show a 2-PDA can simulate a TM.

- Given some Turning machine M, construct a 2-PDA, P, that recognizes the same language as M.
- PDA uses it stack to simulate the tape, one stack for what is right of the head and one stack for what is left of the head.
- A TM starts with input on tape and PDA reads the input in, so what should it do first?

Show a 2-PDA can simulate a TM.

- Given some Turning machine M, construct a 2-PDA, P, that recognizes the same language as M.
- PDA uses it stack to simulate the tape, one stack for what is right of the head and one stack for what is left of the head.
- A TM starts with input on tape and PDA reads the input in, so what should it do first?
- How do you simulate left movement, right movement, replacing the symbol under the head.

Show a 2-PDA can simulate a TM.

- Given some Turning machine M, construct a 2-PDA, P, that recognizes the same language as M.
- PDA uses it stack to simulate the tape, one stack for what is right of the head and one stack for what is left of the head.
- A TM starts with input on tape and PDA reads the input in, so what should it do first?
- How do you simulate left movement, right movement, replacing the symbol under the head.
- Explain how to implement accepting and rejecting.

Show a 2-PDA can simulate a TM.

- Given some Turning machine M, construct a 2-PDA, P, that recognizes the same language as M.
- PDA uses it stack to simulate the tape, one stack for what is right of the head and one stack for what is left of the head.
- A TM starts with input on tape and PDA reads the input in, so what should it do first?
- How do you simulate left movement, right movement, replacing the symbol under the head.
- Explain how to implement accepting and rejecting.
- Again lots of little issues, be sure to consider every case and argue why it works.

Problem 6. Consider a form of Turing machines where instead of having the head having the options to go left or right at each step, its options are to move to the right or stay put. Show that this variant has less power than Turing machines, and argue about what class of languages it does recognize. (Hint: Argue about the class of languages it recognizes first.)

What classes of languages have we seen?

What classes of languages have we seen?

- Regular languages
- Contex-free languages
- Turing-decidable languages
- Turing-recognizable languages (Not this one)

Ideally, you will show equivalence by picking one, and doing two constructive proofs.

- Let A be a [Regular,Contex-free,Turing-decidable] language, and let M be a [DFA,NFA, Regular Expression, CFG, PDA, TM] that [accepts, decides, recognizes] A, the following is a Stay-put Turing machines that also [accepts,decides, recognizes] A.
- Let A be the language [Accepted,decided, recognized] by some Stay-put Turning machines M. The following [DFA,NFA, Regular Expression, CFG, PDA, TM] also [accepts, decides, recognizes] A, hence A must be [Regular, Context-Free, Turing-Decidable].

Ideally, you will show equivalence by picking one, and doing two constructive proofs.

- Let A be a [Regular,Contex-free,Turing-decidable] language, and let M be a [DFA,NFA, Regular Expression, CFG, PDA, TM] that [accepts, decides, recognizes] A, the following is a Stay-put Turing machines that also [accepts,decides, recognizes] A.
- Let A be the language [Accepted,decided, recognized] by some Stay-put Turning machines M. The following [DFA,NFA, Regular Expression, CFG, PDA, TM] also [accepts, decides, recognizes] A, hence A must be [Regular, Context-Free, Turing-Decidable].

A formal construction is great and this is the ideal road, but...

Not sure where to start? Try to construct a Stay-put Turing-machine that accepts each of the following languages:

- (1) 0^*1^*
- (2) 0^n1^n
- (3) $0^n1^n0^n$

JUST SHOWING YOUR EXAMPLE IS NOT A PROOF OR ARGUMENT ABOUT ANYTHING. This step is for your own benefit.

$$A = \{w \mid w \text{ has an equal number of 0s and 1s}\}$$

$$A = \{w \mid w \text{ has an equal number of 0s and 1s}\}$$

Implementation level description:

- Mark off the first unmarked symbol, if all symbols have been marked then accept.
- If the first symbol was a 0, scan through the tape and mark off first 1
- If the first symbol was a 1, scan through the tape and also mark off first 0
- If a 0 or 1 is not found, reject, else return to beginning of the tape and repeat.