Notes for Comp 497 (Comp 454)               Week 4       2/15/05

Chapter 7

Errata (Chapter 7):
  p. 95, line 7, The empty set is not a regular expression, according to our definition
  p. 98, line 5, replace "unless" by "useless"
  p. 100, line 13, replace "being" by "begin"
  p. 107, 2nd graph, place an "a" on the bottom loop.
  p. 110, line 30, replace "send" by "sends".
  p. 116, line 4, replace "form" by "from".
  p. 120, line 17, add "(we have just looped from y1 back to y1, while running on FA2)".
  p. 124, line 7, replace "loose" by "lose".
  p. 131, line 11, replace "z2" by "z4".
  p. 132, line 25, replace "FA2" by "FA*".
  p. 139, 2nd graph, add a "b" to the edge from x4 to empty.
  4th graph, replace "x2 or x3" by "x1 or x2 or x3".
  p. 140, 2nd graph,
    replace in the upper right state "1 or 2 or 3" by "1 or 2 or 3 or 4",
    replace in the lower middle state "1 or 3 or 6" by "1 or 5 or 6", and
    replace in the lower right state "1 or 4" by "1 or 4 or 5 or 6".

Today we will look at Kleene’s Theorem - probably the most important result in finite automata theory (Part 1 of the book).

THEOREM 6 (Kleene’s Theorem)

  Finite Automata (FA)
  Transition Graphs (TG)
  Regular Expressions (RE)

have equivalent power when it comes to defining languages; that is, any language that can be defined by one can be defined by all.

Practically, it means, for example, that if we have a language defined by a regular expression we know that we can devise a finite-state recognizer for it.

Cohen breaks the proof into three parts which we can summarize as

  1. For every FA there is a corresponding TG
  2. For every TG there is a corresponding RE
  3. For every RE there is a corresponding FA
Proof of Kleene’s Theorem: part 1

It is trivially true because every FA is a TG (TGs being more general).

Proof of Kleene’s Theorem: part 2 (pages 93-107)

Show that for every TG there is a corresponding RE. Proof is by construction – give an algorithm that we can apply to any TG that will produce the RE.

Strategy is elimination of states one by one making appropriate changes to the REs on the transitions until we are left with just a start state and a final state and a RE on the transition between them.

Just need some guidelines on the kinds of transformation to make. Here are six, (a) through (f), none changes the language that the TG accepts.

(a) if there are multiple start states, replace them by a single start. For example

(b) Similarly, if there are multiple finish states, replace them by a single finish state
(c) if a state has more than one loop, replace by single loop. If the RE on the original loops are R1 and R2, the RE on the new single loop is R1+R2.

(d) Is a generalization of (c). If there is more than one transition from state X to state Y, replace by a single transition. If the RE on the original transitions are R1 and R2, the RE on the new single transition is R1+R2

(e) (state bypass and elimination) If we have single transition from state X to state Y and from state Y to state Z we can bypass state Y and go directly from X to Z. If R1 is the RE on the transition from X to Y and R2 is the RE on the transition from Y to Z then the RE on the transition from X to Z is R1R2. For example

(b) becomes

Note the variation on this (page 96) where the middle state has a loop. If we are left with a state having no incoming transitions, we can eliminate it.

(f) Is a generalization of (e). If there is a transition from state X to state Y and single transitions from state Y to states Z1, Z2, … we can bypass state Y with transitions directly from X to Z1,
Z2, … Again, we have to take into account looping transitions at Y but we can create appropriate REs for the new transitions. If we are left with a state having no incoming transitions, we can eliminate it.

Careful application of transitions (a) through (f) to an arbitrary TG will result in a TG having just two states (start and final) and a RE on the single transition from start to final. Thus, by construction we show that for every TG there is a corresponding RE.

See how this construction process works on

- the example starting at the foot of page 100
- the example (EVEN-EVEN) starting at the top of page 104
- the example starting on page 105.
- the example starting on page 107.

Cohen encapsulates the transformation process into an algorithm on page 106

Proof of Kleene’s Theorem: part 3 (pages 108-135)

To demonstrate the equivalence of our three language representations (FA, TG, RE) we now just need to show that for any RE we can construct a FA that recognizes the language it represents. The proof will be by construction – showing how the components of the RE lead to elements of the FA. We need to deal with choice, sequence and iteration in the RE.

Cohen’s proof takes the form of a recursive definition and parallel construction.

Overview

1. There is an FA that accepts a single character in Σ. There is an FA that accepts Λ

   Rule 2. If FA1 accepts RE1 and FA2 accepts RE2 we can construct FA3 that accepts (RE1 + RE2) (page 109)

   Rule 3. If FA1 accepts RE1 and FA2 accepts RE2 we can construct FA3 that accepts (RE1RE2) (page 117)

   Rule 4. If FA1 accepts RE1 we can construct FA2 that accepts (RE1 )* (page 125)

Rule 1

Easy to demonstrate – see foot of page 108 and top of page 109.
Rule 2 (choice)

We show how the new machine FA₃ is based on FA₁ and FA₂. Algorithm is on page 113, example is on page 109-112.

States of FA₁ are designated x₁, x₂, … States of FA₂ are designated y₁, y₂, … States of the new FA₃ are designated z₁, z₂, … We are dealing with the union of two machines. We imagine an input string being processed simultaneously on FA₁ and FA₂ and each z state represents “xsomething or ysomething.” We keep track of what state the input would take us to in FA₁ and what state it would take us to in FA₂ and we create enough composite states to represent the reachable two-state combinations. This must be a finite number because the number of states in FA₁ and the number of states in FA₂ are both finite.

Rather than create only the states we need we could create machine FA₃ having a state for each combination of (state from FA₁, state from FA₂) then just eliminate the ones that are not reachable.

Go through the example pp. 109-112 to see how this works.

There is another example pp. 113-114

There is another example pp. 114-115

There is another example pp. 115-117

Rule 3 (sequence)

How do we construct a machine that recognizes strings that have their first part recognized by FA₁ and their second part recognized by FA₂?

Unfortunately, we cannot just make a new machine by making that end state of FA₁ the start state of FA₂ – the example on page 118 shows why we can’t do this in general.

We create FA₃ with states z₁, z₂, … For the non-terminal states xᵢ in FA₁ we create corresponding states zᵢ. For terminal states we may have to create composite states (if there is a loop exit). See the extended example that starts in the last paragraph of page 118 and runs to the top of page 121. There are ideas familiar from the treatment of Rule 2 – the idea that a state in the new machine represents a combination of states in the two “input” machines.

The algorithm for constructing FA₃ from FA₁ and FA₂ is on page 121. You can see that it really only some final states in FA₁ that might cause problems.

There is a second example on pages 122-124 of constructing a Rule 3 FA.
Rule 4 (iteration)

If we have FA₁ that recognizes RE₁, we can construct FA₂ that recognizes (RE₁)⁺.

It is, of course, not just a matter of connecting the final state(s) back to the beginning. Start by looking at the example that begins at the foot of page 125 and continues almost to the end of page 127. Again it shows the general idea of creating z states that represent some combination of states in the original machine. Note that the first version of the construction algorithm (on page 127) does not allow for the machine accepting Λ. The second version of the construction algorithm (page 129) seems clearer as well as more complete and we could almost imagine implementing it. Hmm ... idea for programming project?

Look at the example that starts at the foot of Page 132. Note the comments about having to deal with Λ.

Having shown how we can construct a machine for each of Rule 1 .. Rule 4 we have finished the proof that from any RE we can construct an FA to recognize it and thus we finish Part 3 of the proof of Kleene’s theorem and have shown that the following, in terms of defining languages, are equivalent:

Finite Automata, Transition Graphs, Regular Expressions

Nondeterministic Finite Automata (NFA)

Introduced for completeness. If we imagine a regular FA but allow the possibility that from a given state there might be more than one transition with the same input symbol we get NFA. Also we could start with TG and require only one start state and single symbols on the transitions.

See three examples on pages 136, 137.

THEOREM 7

For every NFA, there is some FA that accepts exactly the same language.

Proof 1: We can use the results and techniques from Kleene’s theorem to go from NFA to TG to RE to FA.

Proof 2: More direct way is to use the algorithm from Part 4 above and create a new machine (FA) based on old (NFA).

There are three examples of this in action on pages 138 and 139. Rather than invent ζ labels for the new states, Cohen shows what old states they correspond to.
Looks like using NFAs in the proof of Kleene’s theorem leads to shorter proofs.

**Homework**

Homework 1 has been graded.

Here is homework 2 on chapter 5, 6 and 7. Each of the five questions is worth 20 points. The homework is due on Tuesday, February 22, 2005.

1. Build an FA that accepts only the words \(baa\), \(ab\) and \(abb\) and no other strings longer or shorter. Assume \(\Sigma \{a, b\}\).

2. (i) Build an FA that accepts the language of strings that contain both an \(a\) and a \(b\) (though not necessarily in that order).

   (ii) Build an FA that accepts the language of all words with only \(a\)’s or only \(b\)’s in them. Give a regular expression for this language.

3. Show that any language that can be accepted by a TG can be accepted by a TG with an even number of states.

4. Using the bypass algorithm in the proof of Theorem 6, Part 2, convert each of the following TGs into regular expressions.

   (a) 
   
   ![Diagram](image.png)
5. Convert each of the following NFAs into FAs

(a)

(b)

Reading Assignment

Read chapter 7. Next week’s class will cover chapters 9 and 10. We are skipping chapter 8.