REs, Equivalences and the Pumping Lemma

Finite Automata Lecture 9

February 12, 2019

Overview

Overview of what's happening this week:

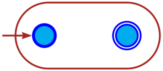
- LIC is away, Thomas Sewell taking over.
- Assignment 1 is now marked.
  - We might look at the Q2 tricky bit at the end.
- Converting RE's to DFA's.
- Proving equivalences on RE's.
- The pumping lemma.
- Closure properties of the set of regular languages.

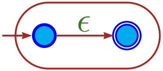
Converting RE's to ϵ-NFA's

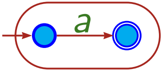
Previously we saw how to convert an ϵ-NFA into a RE.

We can *prove* that every regular expression is implemented by an ϵ-NFA.

Simple constructions for $\emptyset$, ϵ, a.

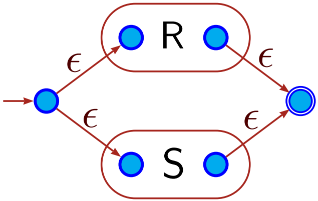


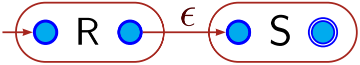


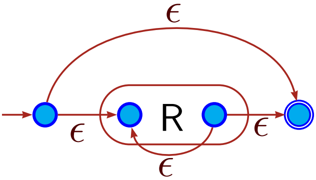


Converting RE's to ϵ-NFA's II

What language is accepted by these ϵ-NFAgadgets?







Converting RE's to ϵ-NFA's III

Proof that every RE is implemented by an ϵ-NFA
with a single accepting state
by structural induction on the language of REs.

Base cases: $\emptyset$, ϵ and a.

Step cases: R, S are implemented by an ϵ-NFA with a single accepting state, show RS is implemented by such an ϵ-NFA, etc.

The previous diagrams are sufficent to show all cases of the induction.
$\therefore$ every RE is implemented by an ϵ-NFA.

Regular Languages

We have seen conversions:

- DFA $\longleftrightarrow$ ϵ-NFA
  - By embedding or the subset construction.
- ϵ-NFA $\longleftrightarrow$ RE
  - By state elimination or recursive construction.

Now, when we talk about *regular languages*, we will mean any of:

- the languages represented by some DFA
- the languages represented by some ϵ-NFA
- the languages represented by some RE

Equivalences of RE's

Which of these regular expression equalities are true?

1. $AA^* = A^*A$
2. $(R+S)^* = R^* + S^*$
3. $L^*(L+M)^*M^* = (L+M)^*$
4. $R(SR)^* = (RS)^*R$
5. $((AB+B)^*A)^* = ((A+B)^*A)^*$
6. $(R^*S)^*R^* = (R+S)^*$

Equivalences of RE's

Which of these regular expression equalities are true?

1. $AA^* = A^*A$
2. $(R+S)^* = R^* + S^*$
3. $L^*(L+M)^*M^* = (L+M)^*$
4. $R(SR)^* = (RS)^*R$
5. $((AB+B)^*A)^* = ((A+B)^*A)^*$
6. $(R^*S)^*R^* = (R+S)^*$

4 & 6 are called the "Shifting Rule" and "Denesting rule".
(For loop rotation and nested loops.)

Proving RE Equivalences

The textbook describes some "hands on" proofs of RE equivalence. Let's talk about three more approaches:

- compose basic equalities
- convert to DFA
- proof by antisymmetry

Recall the antisymmetry property for relations?

$$x \leq y \ \wedge \ y \leq x \ \longrightarrow \ x = y$$

Proving RE Inclusion

There is an antisymmetric order on RE's:

$$\forall R\ L.\ R \leq L \ \longleftrightarrow\ \mathfrak{L}(R) \subseteq \mathfrak{L}(L)$$

The key RE operators are *monotonic*:

$$\begin{aligned}&\forall A\ B\ C\ D.\ A \leq B\ \wedge\ C \leq D\\ &\quad \longrightarrow\ A^* \leq B^*\ \wedge\ AC \leq BD\ \wedge\ A + C \leq B + D\end{aligned}$$

There are also many specific rules, e.g. for union:

$$\forall A\ B\ C.\ A + B \leq C\ \longleftrightarrow\ (A \leq C\ \wedge\ B \leq C)$$

$$\forall A\ B. A \leq A + B\ \wedge\ B \leq A + B$$

Using RE Inclusion

One example from the textbook: $(L+M)^* = (L^*M^*)^*$.

Show $(L^*M^*)^* \leq (L+M)^*$
if $(L^*M^*)^* \leq ((L+M)^*)^*$ as $(R^*)^* = R^*$

Using RE Inclusion

One example from the textbook: $(L+M)^* = (L^*M^*)^*$.

Show $(L^*M^*)^* \leq (L+M)^*$
    if $(L^*M^*)^* \leq ((L+M)^*)^*$ as $(R^*)^* = R^*$
    if $L^*M^* \leq (L+M)^*$ R^* monotonicity

Using RE Inclusion

One example from the textbook: $(L + M)^* = (L^*M^*)^*$.

Show $(L^*M^*)^* \leq (L + M)^*$
if $(L^*M^*)^* \leq ((L + M)^*)^*$ as $(R^*)^* = R^*$
if $L^*M^* \leq (L + M)^*$ R^* monotonicity
if $L^*M^* \leq (L + M)^*(L + M)^*$ as $R^*R^* = R^*$

Using RE Inclusion

One example from the textbook: $(L + M)^* = (L^* M^*)^*$.

Show $(L^* M^*)^* \leq (L + M)^*$
- if $(L^* M^*)^* \leq ((L + M)^*)^*$ as $(R^*)^* = R^*$
- if $L^* M^* \leq (L + M)^*$ R^* monotonicity
- if $L^* M^* \leq (L + M)^*(L + M)^*$ as $R^* R^* = R^*$
- if $L^* \leq (L + M)^* \ \wedge \ M^* \leq (L + M)^*$ RS monotonicity

Using RE Inclusion

One example from the textbook: $(L + M)^* = (L^* M^*)^*$.

Show $(L^* M^*)^* \leq (L + M)^*$

$$\begin{array}{ll}
\text{if } (L^* M^*)^* \leq ((L + M)^*)^* & \text{as } (R^*)^* = R^* \\
\text{if } L^* M^* \leq (L + M)^* & R^* \text{ monotonicity} \\
\text{if } L^* M^* \leq (L + M)^* (L + M)^* & \text{as } R^* R^* = R^* \\
\text{if } L^* \leq (L + M)^* \ \wedge \ M^* \leq (L + M)^* & RS \text{ monotonicity} \\
\text{if } L \leq (L + M) \ \wedge \ M \leq (L + M) & R^* \text{ monotonicity}
\end{array}$$

Using RE Inclusion

One example from the textbook: $(L + M)^* = (L^* M^*)^*$.

Show $(L^* M^*)^* \leq (L + M)^*$
- if $(L^* M^*)^* \leq ((L + M)^*)^*$ as $(R^*)^* = R^*$
- if $L^* M^* \leq (L + M)^*$ R^* monotonicity
- if $L^* M^* \leq (L + M)^*(L + M)^*$ as $R^* R^* = R^*$
- if $L^* \leq (L + M)^* \ \wedge \ M^* \leq (L + M)^*$ RS monotonicity
- if $L \leq (L + M) \ \wedge \ M \leq (L + M)$ R^* monotonicity

n.b. this proof is backwards. For presentation, it's neater to work in the opposite ($\therefore$) direction.

The Pumping Lemma

The thing about finite automata ... is that they're finite.

(Every language can be represented by a D-*infinite*-A.)

Intuitively, we know that $\{0^n 1^n | n \geq 2\}$ cannot be a regular language. How could a finite automaton remember n?

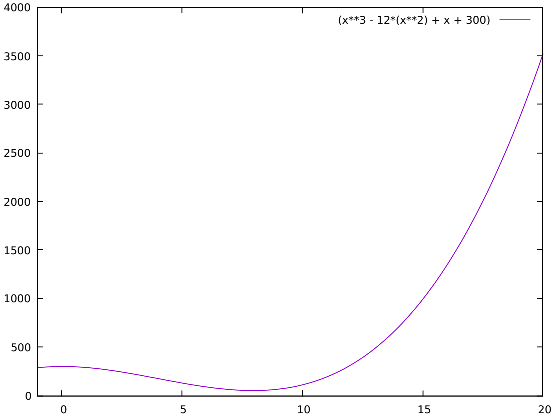
The *pumping lemma* formalises this: given a regular language L,

$$\exists n.\ \forall w.\ |w| \geq n$$
$$\longrightarrow\ (\exists x\ y\ z.\ w = xyz\ \wedge\ |xy| \leq n\ \wedge\ |y| \neq 0\ \wedge\ xy^*z \leq L)$$

Standard Pumping Lemma Exercises

Show that this language is not a regular language:

$$\{S^n \mid \exists m.\ n = x^3 - 12x^2 + x + 300\}$$



Standard Pumping Lemma Exercises II

The previous example was a bit silly, but, there are many similar exercises:

- $\{S^n \mid n \text{ is prime}\}$
- $\{w \in (a+b)^* \mid \#_a(w) = \#_b(w)\}$
- $\{S^n T^{n^2} \mid n \in \mathbb{N}\}$
- Arithmetic expressions formed from:
  - numerals
  - $x + y$
  - $x * y$
  - (x)

Standard Pumping Lemma Exercises II

The previous example was a bit silly, but, there are many similar exercises:

- $\{S^n \mid n \text{ is prime}\}$
- $\{w \in (a+b)^* \mid \#_a(w) = \#_b(w)\}$
- $\{S^n T^{n^2} \mid n \in \mathbb{N}\}$
- Arithmetic expressions formed from:
  - numerals
  - $x + y$
  - $x * y$
  - (x)

The last example is interesting, because we're interested in parsing, so we'll need *more general languages*.

Proving Languages Regular

We can use the pumping lemma to prove a set is *not* a regular language.

How can we prove a set *is* a regular language?

- Construct a DFA, ϵ-NFA or RE.

Proving Languages Regular

We can use the pumping lemma to prove a set is *not* a regular language.

How can we prove a set *is* a regular language?

- Construct a DFA, ϵ-NFA or RE.
- Implicitly construct a DFA.

Proving Languages Regular

We can use the pumping lemma to prove a set is *not* a regular language.

How can we prove a set *is* a regular language?

- Construct a DFA, ϵ-NFA or RE.
- Implicitly construct a DFA.

We've already seen how to construct DFAs representing intersection, union and complement of languages.

- Mostly done via the product construction.

Proving Set S is Regular

Let's consider an example exercise.

Let S be the set of words w with letters from *abcde* which satisfy:

- $|w| \leq 100$
- w contains one c and one d, and the c is before the d
- the a's and e's of w alternate

How can we prove S is a regular language?

Proving Set S is Regular

Let's consider an example exercise.

Let S be the set of words w with letters from *abcde* which satisfy:

- $|w| \leq 100$
- w contains one c and one d, and the c is before the d
- the a's and e's of w alternate

How can we prove S is a regular language?

We can write $S = S_1 \cap S_2 \cap S_3$.

Proving Set S is Regular

Let's consider an example exercise.

Let S be the set of words w with letters from *abcde* which satisfy:

- $|w| \leq 100$
- w contains one c and one d, and the c is before the d
- the a's and e's of w alternate

How can we prove S is a regular language?

We can write $S = S_1 \cap S_2 \cap S_3$.
Let $S_1 = \{w \mid |w| < 100\}$. Consider a DFA with 101 states.

Proving Set S is Regular

Let's consider an example exercise.

Let S be the set of words w with letters from *abcde* which satisfy:

- $|w| \leq 100$
- w contains one c and one d, and the c is before the d
- the a's and e's of w alternate

How can we prove S is a regular language?

We can write $S = S_1 \cap S_2 \cap S_3$.
Let $S_1 = \{w \mid |w| < 100\}$. Consider a DFA with 101 states.
Let $S_2 = (a + b + e)^* c (a + b + e)^* d (a + b + e)^*$.

Proving Set S is Regular

Let's consider an example exercise.

Let S be the set of words w with letters from *abcde* which satisfy:

- $|w| \leq 100$
- w contains one c and one d, and the c is before the d
- the a's and e's of w alternate

How can we prove S is a regular language?

We can write $S = S_1 \cap S_2 \cap S_3$.
Let $S_1 = \{w \mid |w| < 100\}$. Consider a DFA with 101 states.
Let $S_2 = (a+b+e)^*c(a+b+e)^*d(a+b+e)^*$.
S_3 is left as an exercise.

We know we can construct the intersection.

Using the Difference Construction

Given two DFAs which accept languages L and R,
how can we construct a DFA for $L - R$?

- by algebra $L - R = (L \cap \overline{R})$
- by revisiting the product construction

Using the Difference Construction

Given two DFAs which accept languages L and R,
how can we construct a DFA for $L - R$?

- by algebra $L - R = (L \cap \overline{R})$
- by revisiting the product construction

How can we compute whether two DFAs accept the same language?

Using the Difference Construction

Given two DFAs which accept languages L and R,
how can we construct a DFA for $L - R$?

- by algebra $L - R = (L \cap \overline{R})$
- by revisiting the product construction

How can we compute whether two DFAs accept the same language?

How can we compute whether a DFA represents $\emptyset$?

On Model Checking

This is a *computational* approach to checking $L = R$:

- compute DFAs for $L - R$ and $R - L$
- check both DFAs for emptiness

We could write a program to do this, and it would be a very simple model checker.

We could give a whole lecture series on improving (optimising) this computation and handling more complex kinds of automata which better model programs and specifications.

Inductive Sets

That's all for this week's lecture.

Let's talk quickly about Assignment 1. Generally the assignment went well, but it looks like Q2 was a tricky question.