NFA \to DFA
and
Intro to Regular Expressions
Wed, February 10, 2021
Logistics

• **Reminder:** no class next Monday 2/15/2021

• Welcome **new TA:** Benjamin Kwapong
  • See website for additional office hour times

• HW1: solutions posted (to piazza)
  • May use ideas, but not copy (obvi)

• HW2: due Sunday 2/14 11:59pm EST

• HW3: posted, due Sunday 2/21 11:59pm EST
  • Includes a non-coding question

• **Questions?**
Last time: Is Concat Closed for Reg Langs?

• Concatentation of DFAs produces an NFA

• But, regular lang defined using only DFA:

  A language is called a **regular language** if some DFA recognizes it.

• To show: *Concatenation is closed for regular languages*, we must prove that NFAs also recognize regular languages.

• Specifically:
  • **Theorem (1.40):** NFAs $\Leftrightarrow$ regular languages
**Last time:** How to Prove a Theorem: $X \Leftrightarrow Y$

- $X \Leftrightarrow Y = \text{“} X \text{ if and only if } Y \text{”} = X \text{ iff } Y = X \leftrightarrow Y$
- Proof at minimum has 2 parts:
  1. => if $X$, then $Y$
     - i.e., assume $X$, then use it to prove $Y$
     - “forward” direction
  2. <= if $Y$, then $X$
     - i.e., assume $Y$, then use it to prove $X$
     - “reverse” direction
Proving that NFAs Recognize Reg Langs

• **Theorem:**
  • A language $A$ is regular **if and only if** some NFA $N$ recognizes it.

• Must prove:
  • $\Rightarrow$ If $A$ is regular, then some NFA $N$ recognizes it
    • Easy
    • **We know:** if $A$ is regular, then a DFA recognizes it
    • **To complete this part of proof:** convert DFA to an NFA! (how?)
  • $\Leftarrow$ If an NFA $N$ recognizes $A$, then $A$ is regular
    • Hard
    • **We know:** if a DFA recognizes a lang, then it is regular
    • **Idea:** Convert NFA to DFA
How to convert NFA -> DFA?

**Proof idea:**
Each “state” of the DFA must be a set of states in the NFA.
In a DFA, all these states at each step must be only one state.

So design a state in the DFA to be a set of NFA states!

This is a generalization of the proof strategy from Thm 1.25 (closure of union), where a state = pair of “states”
Convert NFA -> DFA, Formally

• Let NFA $N = (Q, \Sigma, \delta, q_0, F)$

• An equivalent DFA $M$ has states $Q' = \mathcal{P}(Q)$ (power set of $Q$)
Example:

**Figure 1.42**  
The NFA $N_4$

**Figure 1.43**  
A DFA $D$ that is equivalent to the NFA $N_4$
NFA -> DFA (ignore empty transitions)

1. $Q' = \mathcal{P}(Q)$. A state for $M$ is a set of states in $N$

2. For $R \in Q'$ and $a \in \Sigma$, $R = a$ state in $M = a$ set of states in $N$

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

To compute next state for $R$, compute next states of each NFA state $r$ in $R$, then union results into one set

3. $q_0' = \{q_0\}$

4. $F' = \{R \in Q' | R$ contains an accept state of $N\}$
NFA -> DFA (with empty transitions)

- Have: $N = (Q, \Sigma, \delta, q_0, F)$
- Want to: construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$

1. $Q' = \mathcal{P}(Q)$.

2. For $R \in Q'$ and $a \in \Sigma$,
   
   $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a) \cup E(\delta(r, a))$

3. $q_0' = \bigcap E(\{q_0\})$

4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$
Proving that NFAs Recognize Reg Langs

• **Theorem:**
  • A language $A$ is regular **if and only if** some NFA $N$ recognizes it.

• **Must prove:**
  • $\Rightarrow$ If $A$ is regular, then some NFA $N$ recognizes it
    • Easy
      • We know: if $A$ is regular, then a DFA recognizes it
      • Convert DFA to an NFA
  • $\Leftarrow$ If an NFA $N$ recognizes $A$, then $A$ is regular
    • Hard
      • We know: if a DFA recognizes a lang, then it is regular
      • Idea: Convert NFA to DFA
      • Using NFA $\rightarrow$ DFA algorithm we just created!

■ (Q.E.D.)
So Concatenation is Closed for Reg Langs!

- Concatentation of DFAs produces an NFA
“Regular” Operations

• Regular languages are closed under these operations:
  • Union (already proved with DFAs)
  • Concatenation
  • Kleene Star (repetition, zero or more times)

• Easier to prove closure (by construction) using NFAs
Union

New start state, $\varepsilon$-transitions to old start states
Let $N_1$ recognize $A_1$, and $N_2$ recognize $A_2$.

Construction of $N$ to recognize $A_1 \circ A_2$
New start (and accept) state, \( \varepsilon \)-transitions to old start state

Old accept states \( \varepsilon \)-transition to old start state

Kleene Star
Why do we care?

• Union, concat, and kleene star represent all regular languages

• I.e., they define regular expressions

**Definition 1.52**

Say that $R$ is a *regular expression* if $R$ is

1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\varepsilon$,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions, or
6. $(R_1^*)$, where $R_1$ is a regular expression.
Poll: Regexes
Ways to Recognize a Regular Language

• Instead of:

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. $\delta$ is described as

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

• Or:

4. $q_1$ is the start state, and
5. $F = \{q_2\}$.

• We can write a regexp: $\Sigma^*001\Sigma^*$

These all define a computer (program) that accepts all strings containing 001

Which would you rather write?
Regular Expressions are Super Useful

- IntelliJ
Regular Expressions are Super Useful

- Visual Studio
Regular Expressions are Super Useful

- Grep (Linux)
Regexp supported in every language

- Perl
- Python
- Java
- Every lang!
Caveat: Regexps are useful, if used correctly

- Regexps: potentially **useful** ...

... only if used correctly

Where “used correctly” = only use it to recognize regular languages

(To do this, you must know what is, and is not, a regular language!)
HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The `<center>` cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regex will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow it is too late it is too late we cannot be saved the trangression of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) dear lord help us how can anyone survive this scourge using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes using regex as a tool to process HTML establishes a breach between this world and the dreadful realm of corrupt entities (like SGML entities, but more corrupt) a mere glimpse of the world of regex parsers for HTML will instantly transport a programmer's consciousness into a world of ceaseless screaming, he comes, the pestilent slithy regex-infection will devour your HTML parser, application and existence for all time like Visual Basic only worse he comes he does not fight he comes, his unholy radiance destroying all enlightenment. HTML tags leaking from your eyes/like liquid pain, the song of regular expression parsing will extinguish the voices of mortal man from the sphere

I need to match all of these opening tags:

```
<p>
<a href="foo">
```

But not these:

```
You can't parse [X]HTML with regex. Because HTML can't be parsed.
```

```
[Regular language and hence cannot be parsed by regular expression queries are not equipped to break down HTML into its meaningful parts but it is not getting to me. Even enhanced irregular regular expression used by Perl are not up to the task of parsing HTML. You will never]
```

Have you tried using an XML parser instead?
Big Picture Road Map

• We ultimately want to prove:
  • Regular Languages $\Leftrightarrow$ Regular Expressions

• First, we need to show these operations are closed for reglangs:
  • Union (done!)
  • Concatentation (done!)
  • Kleene star (done!)
Thm: A lang is regular iff some regexp describes it

• => If a language is regular, it is described by a regexp

• <= If a language is described by a regexp, it is regular
  • Easy!
  • Construct the NFA!
  • See Lemma 1.55
Say that $R$ is a regular expression if $R$ is

1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\varepsilon$,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions,
6. $(R_1^*)$, where $R_1$ is a regular expression.

Constructions from before!
Thm: A lang is regular iff some regexp describes it

• => If a language is regular, it is described by a regexp
  • Hard!
  • Need something new: a GNFA

• <= If a language is described by a regexp, it is regular
  • Easy!
  • Construct the NFA! (Done)
GNFA = NFA with regexp transitions

- To convert to regexp, keep “ripping out” states until only 2 are left
Check-in Quiz 2/10

On gradescope