Nondeterminism

Monday Sept 21, 2020
Hw1 questions?
How to Code (Recap)

Since 2011, engineers at Amazon Web Services (AWS) have been using formal specification and model checking to help solve difficult design problems in critical systems. This paper describes our motivation.

We found what we were looking for in TLA+[^4], a formal specification language. TLA+ is based on simple discrete math, i.e. basic set theory and predicates, with which all engineers are familiar. A TLA+

"Translating the math into code" is exactly the definition of "knowing how to code". Typically, the "math" is called a "specification" or "requirements", and it's a combination of vague English and actual math, just like the B description near as clear and detailed as my writing of course).

And from this specification you will be expected to ship a fully working product (testing with an autograder either) at the end of a tight schedule.

For non-software industry programming jobs, you'll get even less direction. Again, I say this not to belittle or discourage, but to try to prepare you all for whatever futures as best I can. My door is always open to anyone who wants to talk.

[^4]: http://ltl.com/tla/
How to Code: **Step 1, Data Defs**

- Design your **Data Definitions**
- I.e., representation of real-world thing(s) your program operates on

A **User** is a **struct** containing
- **String** name
- **String** screenname
- **Int** internal_ID
- **List**<Post> posts
- **List**<User> followers

A **Post** is a (140 character) **String**
How to Code: **Step 2**, Data Operations

- Design **Operations** for your data from step 1
- **Users** need to:
  - `Post()`
  - `Delete()`
  - `Like()`
  - `Follow()`
- A good specification/requirement (like the hw) gives this to you
How to Code: **Step 3**, start coding

• **Implement** the operations, **step-by-step**

• Start with one tiny, simple, observable piece of code
  • E.g., read input; print as output

• Add more code slowly, step-by-step
  • Should be guided by your data definitions and operations
  • E.g., read input as xml file
    • Then Parse xml file, print states
    • Then Parse transitions, then construct DFA object
  • Make sure the program changes how you expect **at each step**
How to Code: Step 4, testing

• **Build up** to a **small** test case
  • The Hw always gives one

• Eventually, create more tests
  • You write tests, right?
  • Each should test different parts of your program
  • 100% code coverage is minimum requirement
  • Easy way to test union problem?
    • Use you solution from parts 1-3 of the hw
How to Code: Step 5, debugging
FAQ: Is the autograder broken?
No, the autograder is not broken

• If the autograder is crashing, then your program is broken

• The autograder is **not** a debugging tool
  • So don’t use it to debug
  • Debugging is solely your job

• The autograder’s only obligation: report your grade score

• However, all your errors are reported in the summary section
How to Code: Step 5, debugging

• If you followed steps 1-4, then debugging should be obvious
  • Program in small, composable pieces (ie, fns, methods, classes)

• Still have big chunk of code fails, what to do?
  • Narrow it down.
  • Do something observable, eg, print(“made it here”), halfway
  • Keep narrowing down (binary search) until you find the right line
Final notes about coding

• It’s a requirement for the course

• Coding hws will likely end around hw4 (maybe)

• Remember: lowest hw score dropped

• Can still do well in the course without writing any code
Nondeterminism
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, last class!)
  • Concatenation
  • Kleene star

• To prove the last 2, we need non-determinism and NFAs!
  • We know Regular Languages $\Leftrightarrow$ DFAs (by definition)
  • But are Regular Languages $\Leftrightarrow$ NFAs???
Last time: Concatenation Operation

- Example: Want to match street addresses

212 Beacon Street

M3: CONCAT

M1: recognize numbers

M2: recognize words
Last time: Concatenation Closed?

**Theorem 1.26**

The class of regular languages is closed under the concatenation operation.

In other words, if $A_1$ and $A_2$ are regular languages then so is $A_1 \circ A_2$.

- Construct a **new** machine $M$?
  - using DFA $M_1$ (which recognizes $A_1$),
  - and DFA $M_2$ (which recognizes $A_2$)
N is a new kind of machine, an NFA!

Let $N_1$ recognize $A_1$, and $N_2$ recognize $A_2$.

Want: Construction of $N$ to recognize $A_1 \circ A_2$.

$\varepsilon = \text{empty string} = \text{no input}$

So $N$ can:
- stay in current state **and**
- move to next state

So is concatenation not closed???
NFA = Nondeterministic Finite Automata
Example fig1.27 (JFLAP demo): 010110
Nondeterministic machine can be in multiple states at once

**DEFINITION 1.37**

A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set of states,
2. \(\Sigma\) is a finite alphabet,
3. \(\delta: Q \times \Sigma^* \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

A *finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
5. \(F \subseteq Q\) is the *set of accept states.*
Power Sets

- A power set is the set of all subsets of a set

- **Example:** $S = \{a, b, c\}$

- Power set of $S =
  - $\{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$
Formal Definition of “Computation”

• DFA:

$M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ in $Q$ exists with three conditions:

1. $r_0 = q_0,$
2. $\delta(r_i, w_{i+1}) = r_{i+1},$ for $i = 0, \ldots, n - 1,$ and
3. $r_n \in F.$

• NFA:

$N$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_m$ exists in $Q$ with three conditions:

1. $r_0 = q_0,$
2. $r_{i+1} \in \delta(r_i, y_{i+1}),$ for $i = 0, \ldots, m - 1,$ and
3. $r_m \in F.$

Requires only one path to an accept state in the computation tree
In-class exercise

• Come up with a formal description of the following NFA:

**Definition 1.37**

A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set of states,
2. \(\Sigma\) is a finite alphabet,
3. \(\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.
The formal description of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. $\delta$ is given as

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>${q_1}$</td>
<td>${q_1, q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${q_3}$</td>
<td>$\emptyset$</td>
<td>${q_3}$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$\emptyset$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>${q_4}$</td>
<td>${q_4}$</td>
<td>$\emptyset$,</td>
</tr>
</tbody>
</table>

4. $q_1$ is the start state, and
5. $F = \{q_4\}$. 

![Diagram](image-url)
So is concat not closed for regular langs?

• Concat produces an NFA

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

• Concat is closed!

• Because **NFAs also recognize regular languages**!
  • But we must prove it!

• To show concatenation is closed, we must prove
  • NFAs $\Leftrightarrow$ regular languages
How to prove the theorem: $X \iff Y$

- $X \iff Y = "X$ if and only if $Y" = X$ iff $Y = X \iff Y$
- **Proof at minimum** has 2 parts:
  1. $\Rightarrow$ if $X$, then $Y$
     - i.e., assume $X$, then use it to prove $Y$
     - “forward” direction
  2. $\Leftarrow$ if $Y$, then $X$
     - i.e., assume $Y$, then use it to prove $X$
     - “reverse” direction
Proving NFAs recognize regular langs

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

• Must prove:
  • => If A is regular, then some NFA N recognizes it
    • Easy
    • We know: if A is regular, then a **DFA** recognizes it.
    • Easy to convert DFA to an NFA! (how?)
  • <= If an NFA N recognizes A, then A is regular.
    • Hard
    • Idea: Convert NFA to DFA
Need a way to convert NFA -> DFA

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the **states**, 
2. \(\Sigma\) is a finite set called the **alphabet**, 
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**, 
4. \(q_0 \in Q\) is the **start state**, and 
5. \(F \subseteq Q\) is the **set of accept states**.

A nondeterministic finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set of states, 
2. \(\Sigma\) is a finite alphabet, 
3. \(\delta: Q \times \Sigma_e \rightarrow \mathcal{P}(Q)\) is the transition function, 
4. \(q_0 \in Q\) is the **start state**, and 
5. \(F \subseteq Q\) is the **set of accept states**.

**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 converted DFA to be a set of NFA states!
Next time: Convert NFA -> DFA

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

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

• (implement for hw2)
Check-in Quiz 9/21

On gradescope
End of Class Survey 9/21

See course website