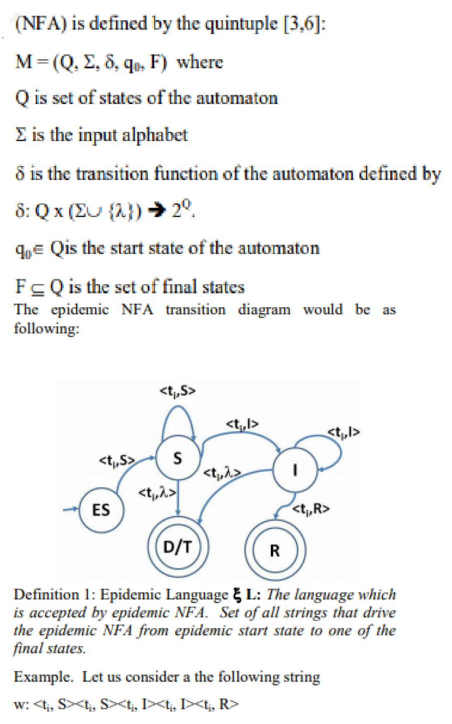
CSE396 Lecture Tue. Feb. 2, 2021. Formal Languages (and Syllabus Overview)

Thoughts entering the Spring 2021 term? (Notice the word "epidemic" at bottom left.)



(NFA) is defined by the quintuple [3,6]:

$M = (Q, \Sigma, \delta, q_0, F)$ where

Q is set of states of the automaton

Σ is the input alphabet

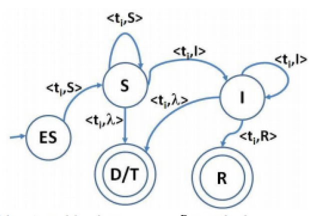
δ is the transition function of the automaton defined by

$\delta: Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$.

$q_0 \in Q$is the start state of the automaton

$F \subseteq Q$ is the set of final states

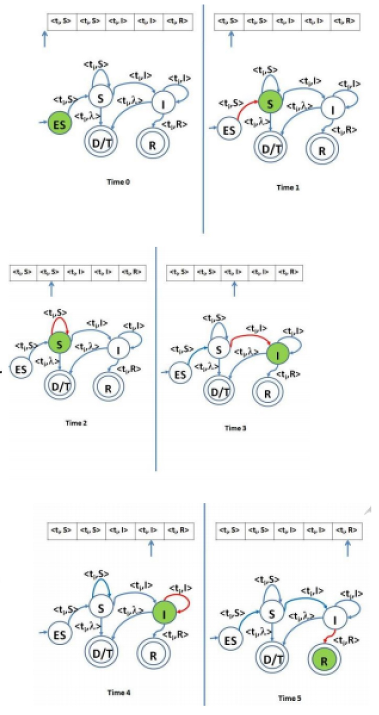
The epidemic NFA transition diagram would be as following:



Definition 1: Epidemic Language ξ L: *The language which is accepted by epidemic NFA. Set of all strings that drive the epidemic NFA from epidemic start state to one of the final states.*

Example. Let us consider a the following string

w: $<t_i, S><t_i, S><t_i, I><t_i, I><t_i, R>$



Some remarks relevant to multiple aspects of the course, including Academic Integrity:

- I paid $11 for license from CartoonStock to use the middle part in classroom environments. (Web publishing would have been $55, print publishing $50---what does that say?)
- The side panels come from an academic paper by researchers at the Mody Institute in India, https://www.ijert.org/research/finite-automata-for-sir-epidemic-model-IJERTV2IS90886.pdf, from the International Journal of Engineering Research & Technology (IJERT). Usage from an academic paper *with citation* is generally understood as free if the original access is free.

- The paper, "Finite Automata for SIR Epidemic Model," tries to apply concepts from early weeks of an intro undergrad theory course---like ours---to regularize models of pandemics.
- As such, the paper is (IMPO) *premature*. It should really develop the concept of a *probabilistic* finite automaton or *probabilistic cellular automaton*. It kind-of does the latter in its section 4.
- Probabilistic automata are not on our syllabus or in the text, nor even covered in CSE596. But this course should equip you to understand how to "read" and develop formal models in general. If it expands your capacity to appreciate the difference betweem "SIR" and "SEIR", and/or other models from https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology, then that would be one instance of "mission accomplished."

Alas, the pandemic is affecting a second Spring term, "Groundhog Year" one could say. What will be the same, and what different?

- The main difference will be reducing material by 20% in the two prelim exams---basically you'll have 75 minutes to do material that would be scaled for 60 minutes in-person.
- Syllabus breakdown: 2 x 12% = **24%** for prelims, **36%** for a 3-hour cumulative final, **40%** homeworks *including points for attendance and auto-graded questions*, both via *TopHat*.
- Assignments will be individual-work only; I will say more about integrity groundrules when giving the first assignment to be due (tentatively) Tuesday night the 16th.
- Lectures will be more "PowerPoint-y"; actually, this is MathCha.io, which has a mathematical environment based on LaTeX and good drawing tools. I've always used slides-style in CSE250 and CSE305, but for CSE396 and CSE191, I ordinarily prefer the buildup of a "chalk talk" especially for presenting proofs and problem solutions.
- That said, the twin goals will be (a) to equalize the experience for those in-person and those remote as much as possible, while (b) preserving the in-class atmosphere. I do sometimes "call audibles" in lectures in response to questions or requests for examples.
- Exams will be remote-only, using the same Autograder logistics as for homeworks.
- TopHat will be used for parts of homeworks as well as attendance logging. Possible further use during lecture may depend on how smoothly interaction works for both sets of attendees.
- Other aspects treated the same as in previous years.

[Go over printed syllabus from 2019 while again noting above adjustments.]

Brief Course Overview

1. Formal Languages as "Math With Symbols" (this week).
2. Finite Automata and Regular Expressions (this month into next, long Sipser ch. 1).
3. Context-Free Grammars and Languages (next month, Sipser ch. 2 but not section 2.4).
4. Computability and Undecidability (April, Sipser chs. 3--5 skimming section 5.2).
5. A bit of Computational Complexity (May, Sipser ch. 7 and one page of ch. 9 for a proof).

The very last lectures will be on the Cook-Levin Theorem for showing **NP**-*hardness* and **NP**-*completeness*. (About which, here is another reference this weekend on our downbeat theme: https://psyche.co/ideas/the-mathematical-case-against-blaming-people-for-their-misfortune)

Formal Languages

Which comes first, the number or the symbol? Let us multiply

```
    47
 x  43
-------
   141
  188_
-------
  2021
```

Our year is the product of two nearly-equal primes, both congruent to 3 modulo 4. This makes 2021 into a Blum Integer, and the importance of this to cryptography will also be touched on in the last week. But staying in the first week, would you say the operations here are *numeric* or *symbolic*? What if we do this in binary notation?

```
     101111
   x  101011
-----------
      101111
     101111
   101111
 101111
-----------
 11111100101
```

Maybe this feels more symbolic. In my youth there was more a balance between "analog" and "digital" as monikers for computing, but that's all gone digital. (Music, however, has made a large move back to analog.) In any event, this course handles the symbolic side.

This begins with defining the **alphabet** of symbols used. The alphabet for binary arithmetic, and binary strings in general, is $\{0,1\}$. In many ways, this is the only alphabet we formally need to consider. The alphabet $\{a,b\}$ will have a different "feel"---I will use it more often than the text because it feels like using words, and some examples will put other letters $c, d, e, \ldots$ to good use. But formally, $\{a,b\}$ works just like $\{0,1\}$ if you think $a = 0, b = 1$ (or vice-versa). Moreover, to a computer, all letters on our keyboards get translated to binary strings, variously by the ASCII code, Unicode, or UTF-8, which is an amalgam of the two.

A motive for making alphabets more general is to incorporate **tokens** as basic units. Tokens are often notated inside angle brackets $\langle \ldots \rangle$ and that is exactly what the above paper does. For example, it refers to the "string"

<t_i, S> <t_i, S> <t_i, I> <t_i, I> <t_i, R>

where each "t_i" is a person and S,I,R are the epidemiological labels for people who are in the state of being Susceptible, Infected, or Recovered. This is "meta"---we have symbols inside our symbols, but the point is that the (Nondeterministic) Finite Automaton defined in the paper takes these tokens as its basic inputs.

So we want to think abstractly of a general alphabet---and the convention is to use a capital Greek Σ to denote one. This may be confusing---Σ usually stands for a sum, and we may have a few of those too. But we use a limited set of letters in our notation, and Σ took hold---as exemplified in the same paper. Most of the time we will have $\Sigma = \{0,1\}$ or $\Sigma = \{a,b\}$, however.

A **string** is a sequence of characters. In C++ terms, `string = list<char>` whereas `alphabet = set<char>`.

A set of strings is called a **language**. This is "barebones"---it's like saying the English language equals the set of words you can play in the game *Scrabble*. Chapter 2 will be all about defining rules on top of words and the kinds of more-human-like languages you can get as a result.

[Continue with examples of strings and languages, and some other basics from Sipser "Chapter 0", as time allows.]

[Here are some pertinent examples.]

If `language = set<string>` isn't "up there" enough, there's also the term that a **class** is a set of languages. The first major example will be the class **REG** of *regular languages*.

The *empty language*, like any empty set, is denoted by $\emptyset$. The empty string will be denoted by ϵ (Greek lowercase epsilon) in this course. [Other sources---including the above paper---use λ (Greek lowercase lambda) for the empty string. I will often mention notational variants in sources you may see on the Web.]

What's the difference between $\emptyset$ and ϵ? First, the former is a `set`, the other a `string`. Second, we will see the difference is like that between the numerical 0 and 1 as numbers. Observe:

- The concatenation $x \cdot c$ of a `string` x and a `char` c is the string xc. For example, $aab \cdot a = aaba$. An English rendering of $\cdot$ is "*and then*".
- The concatenation $x \cdot y$ of strings x and y is the string xy. E.g., $aab \cdot aba = aababa$.
- This is the same as what you get by "catting on" to x the chars in y one at a time.
- If $y = \epsilon$, then y has no chars, so the last point is a no-op. So: $x \cdot \epsilon = x$ is a general rule, for all strings x. Likewise, $\epsilon \cdot x = x$ is a general rule. That's how ϵ is like 1. (Well, this makes $\cdot$ analogous to multiplication, but it's not commutative: $aab \cdot aba \neq aba \cdot aab$.)

To really compare it with $\emptyset$, we need to involve ϵ in a language. So consider: $\{\epsilon\}$. This is a *set* whose only member is a *string*, so it is a `set<string>`, which is a `language`. Next we need to "lift" the concatenation operation up to work between languages. This needs a definition:

Definition 1: Given any two languages A and B (their being "over" the same alphabet Σ is understood here), their concatenation is the language $A \cdot B$ defined by

$$A \cdot B = \{x \cdot y : x \in A \wedge y \in B\}.$$

An intuition for this is that strings are like streams of data from sensors, and languages $A, B, \ldots$ are tests telling whether chunks of data meet respective conditions for being OK. So a string z passes the $A \cdot B$ test if it consists of a portion x that passes the A test *and then* a portion y that passes the B test. Here's a little swervy test of notation: Does $A \cdot A = \{x \cdot x : x \in A\}$? The answer is that this is too narrow. Suppose $A = \{0,1,2,3,4,5,6,7,8,9\}$ represents the condition of being a digit character (`\d` if you've done string-matching). Then $A \cdot A$ should allow any two digits, not just the doubled cases $00, 11, \ldots, 99$. Instead, $A \cdot A = \{x \cdot y : x, y \in A\}$.

Having understood that about Definition 1, let us try the "edge cases" $B = \emptyset$ and $B = \{\epsilon\}$:

- $A \cdot \emptyset = \{x \cdot y : x \in A \wedge y \in \emptyset\} = \{x \cdot y : x \in A \wedge \mathit{false}\} = \{x \cdot y : \mathit{false}\} = \emptyset$.
- $A \cdot \{\epsilon\} = \{x \cdot y : x \in A \wedge y \in \{\epsilon\}\} = \{x \cdot \epsilon : x \in A\} = \{x : x \in A\} = A.$

Likewise, $\emptyset \cdot A = \emptyset$ for any language A, whereas $\{\epsilon\} \cdot A = A$ always. Intuitively, $A \cdot \emptyset = \emptyset$ says that if a sensor at a required stage fails then the whole test series fails. Whereas, $A \cdot \{\epsilon\}$ means that the second condition passes automatically on the heels of the first, without needing (or allowing) any more data to be taken.

Now let us abbreviate $A \cdot A$ as A^2, $A \cdot A \cdot A = A^3$, and so on. This is OK even though concatenation isn't commutative on languages either---hey, neither is matrix multiplication, but A^3 is like raising a matrix to the third power, and that's fine. Just like with numbers and matrices, strings and languages obey the additive power law: $\boldsymbol{A^i \cdot A^j = A^{i+j}}$.

We have $A^1 = A$, of course, but what is A^0? In particular, what is $\emptyset^0$? Well, suppose we wrote a program loop to fetch and test a mandated number n of chunks of sensor data?

```
for (int i = 0; i < n; i++) {
   string xi = getNewSensorData();
   if (!A(xi)) { throw Failure; }
}
```

If we invoke this loop with a given number n then it will perform n tests and allow processing to continue without exception only if all n of the tests pass.

- If $A = \emptyset$ and we enter the loop, then the body surely fails, *finito*.
- If $n = 0$ then what happens? The loop is a fall-through. Do we die? No: processing continues undisturbed.
- So what happens if $A = \emptyset$ and $n = 0$? *The same as the second case*: we never get put to the death test, and processing continues undisturbed.

In the third case, nor is any data taken. Thus this is exactly the same situation as when concatenating with $\{\epsilon\}$. It is a "free pass". So $A^0 = \{\epsilon\}$ for any language A, and in particular:

$$\emptyset^0 = \{\epsilon\}.$$

Well, this is just like the numerical convention $0^0 = 1$. In all cases, this is needed to make the additive power law $\boldsymbol{A^i \cdot A^j = A^{i+j}}$ work even when $j = 0$.

[If even more time allows, tell the story at https://rjlipton.wordpress.com/2015/02/23/the-right-stuff-of-emptiness/ . Wherever the break comes, the rest will be part of notes for next week's recitations. In 2019, the second lecture was wiped out by snow, so https://cse.buffalo.edu/~regan/cse396/CSE396lect020519.pdf (while I was in India) corresponds to Thursday's lecture on deterministic finite automata from the beginning of chapter 1.]