Subregular Languages

Jacob Johnson

Suppose you are given the following words and told that they all come from an unknown language:

shashekush *sasokos* *shakushush* *sakesas*
sheshukush *sesukos* *shekeshush* *sekoses*
shishokesh *sisekos* *shikishosh* *sikisis*
shoshokesh *sosakos* *shokashesh* *sokusas*
shushakish *susukos* *shukoshosh* *sukasus*
shashokush *sasakus* *shakushish* *sakisos*
sheshokish *sesukis* *shekeshosh* *sekeses*
shishukosh *sisokos* *shikeshash* *sikasos*
shoshokush *sosikas* *shokeshish* *sokosis*
shushakesh *susokis* *shukoshesh* *sukesas*

Now suppose you are asked to determine, for each of the below pairs, which word in the pair also came from this same language:

shoshukosh *shosukosh*
sukesus *sukeshus*
sisakus *sishakus*
shakashosh *shakasosh*
shokushosh *shokusosh*
susekus *sushekus*
sikasus *sikashus*
shashakosh *shasakosh*
soshakas *sosakas*
shusekish *shushekish*
sakushes *sakuses*
shekosash *shekoshash*
sashukes *sasukes*
shesokash *sheshokash*
sokashas *sokasas*
shukesish *shukeshish*

Take a minute to look through this list for yourself. Pronounce them out loud to get a better feel for them and consider writing down your answers to check back later. This may seem like a strange task, but it was real for participants in the studies Lai (2015) and Avcu & Hestvik (2020). And if you think about it, you have been doing the same thing your whole life as you have learned languages: for English, when you hear the words "gnocchi", "tzatziki" and "sriracha", you can probably tell that they originated from another language, based on the way they sound. Conversely, when you hear the words "blick", "nord", or "saf" you might get the impression that they *could have been* natural-sounding English words — maybe a name, or an item — even though they don't have any English meaning to you. However, you likely don't get that same impression about "bnick", "ngord", or "sapf".

Why is this the case? And what were Lai and Avcu & Hestvik investigating with their studies?

In this article, we will investigate the answers to these questions, look at foundational material in phonotactics and computability theory, and gain insight along the way into the connection between language and computation. By the end, you will understand what these researchers predicted about your answers to the above questions, and why they predicted that.

Phonotactics

Phonotactics describes the patterns of sounds (called phonemes when referring to types of sounds, or segments when referring to individual instances of a phoneme) that are used by a language. We've already seen examples of this: a phonologist (a linguist who studes speech sounds) would describe our observation that "bnick" doesn't *sound English* by saying that "bnick" *violates English phonotactics*. Phonologists usually denote this ungrammaticality or ill-formedness (meaning that a speaker of the language in question would say this word sounds strange) with an asterisk: *bnick

Phonotactics is an additional dimension of a language's phonology (its complete set of sound patterns) beyond its phonemic inventory (the set of types of sounds, or phonemes, it contains) — you might associate Spanish with rolled r's, French with

nasalized vowels, or German and Arabic with throaty sounds — and it's true that different languages do have different phonemic inventories (English is uncommon in having its "th" sounds - both the kind in "that" and in "think"). Instead, phonotactics has to do with the sounds that a language allows to cooccur next to each other or within a word — the "bn" consonant cluster from "bnick" is not allowed in English, for example. These phonotactic restrictions, just like phonemic inventories, vary from language to language: German allows clusters like "kn" and "pf" which English prohibits, while a lot of languages allow the "ng" sound from the end of "sing" to occur in places other than at the end of words. Some languages, on the other hand, are more restrictive than English in some ways, like Spanish, which doesn't allow clusters of "s" followed by a consonant (except if it bridges a syllable boundary), or Turkish, which has restrictions on which vowels can cooccur in the same word.

Local Restrictions

Many of these phonotactic rules, you may have noticed, prohibit certain phonemes from occurring adjacent to each other in a word. Computational phonologists call these sort of restrictions, and the languages that contain them Strictly Local. This name derives from the fact that you can tell whether a word is well formed or not simply by looking at each substring (that is, each piece of the word, without skipping any segments) of the word, and making sure there isn't anything that should be restricted: "blick" sounds ok to English speakers because "bl" is ok ("blend", "blunder", "able"), "li" is ok ("little", "flint", "alliteration") and "ick" is okay ("sick", "trick", "pick" - notice that we treat "ck" as just one sound). ▮

Non-Local Restrictions

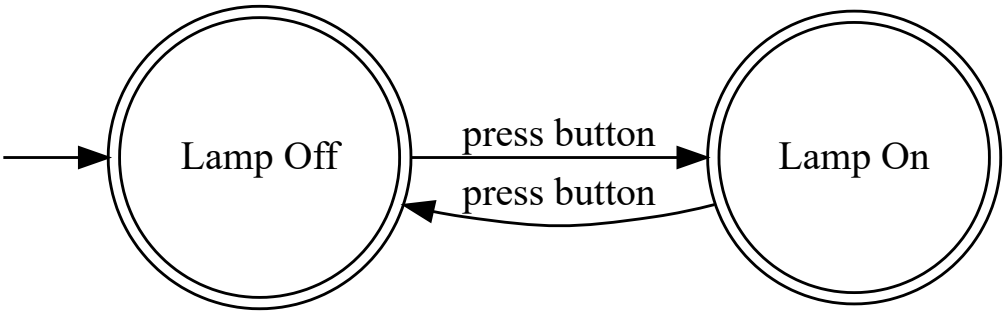
However, many languages have restrictions which occur at a distance, like Turkish, which I introduced as having restrictions on which vowels can cooccur in the same word. Specifically, vowels in Turkish words agree, or harmonize, in their qualities of backness (how far back the tongue is when the vowel is pronounced) and rounding (whether the speakers lips are rounded). This interaction occurs across syllables and most morphemes (which are parts of words, like suffixes for example), even though there are other consonants in between. Heinz et al. 2011 proposes to account for this observation by introducing the idea of a phonological tier: "a level of

representation where not all speech sounds are present". That is, Turkish vowels might be separated by consonants when they are pronounced, but we can also think of them as existing within a tier, or projection, from the word, which only contains the vowels that participate in these interactions. Within this tier, the restrictions on front vowels cooccuring with back vowels is still the same Strictly Local - without the consonants in the way, the vowels are right next to each other. This was a very important idea and introduced the concept of Tier-based Strictly Local (also shortened to TSL) languages - ones whose patterns could be expressed as a tier and a set of restricted substrings on that tier. This same paper also introduced what is known as the **Subregular Hypothesis**, which starts to bring us back to the study question from the beginning of the article, but we are not quite ready to understand what that means, or why it is called that, so lets move on to our next topic.

Computability Theory

Lamp

Suppose you have a lamp. When you first get it, it is turned off, but you can turn it on by pressing a button, or off again by pressing the same button. You might represent this lamp with the following diagram:



Let's notice a few things about this diagram. First, there are two **states**: "Lamp Off" and "Lamp On", represented here by circles. Secondly, there is an extra arrow pointing to the "Lamp Off" state, indicating that the lamp is turned off when you first begin using it. Finally, there are transitions between the states, which in this case both represent the action of pressing the button.

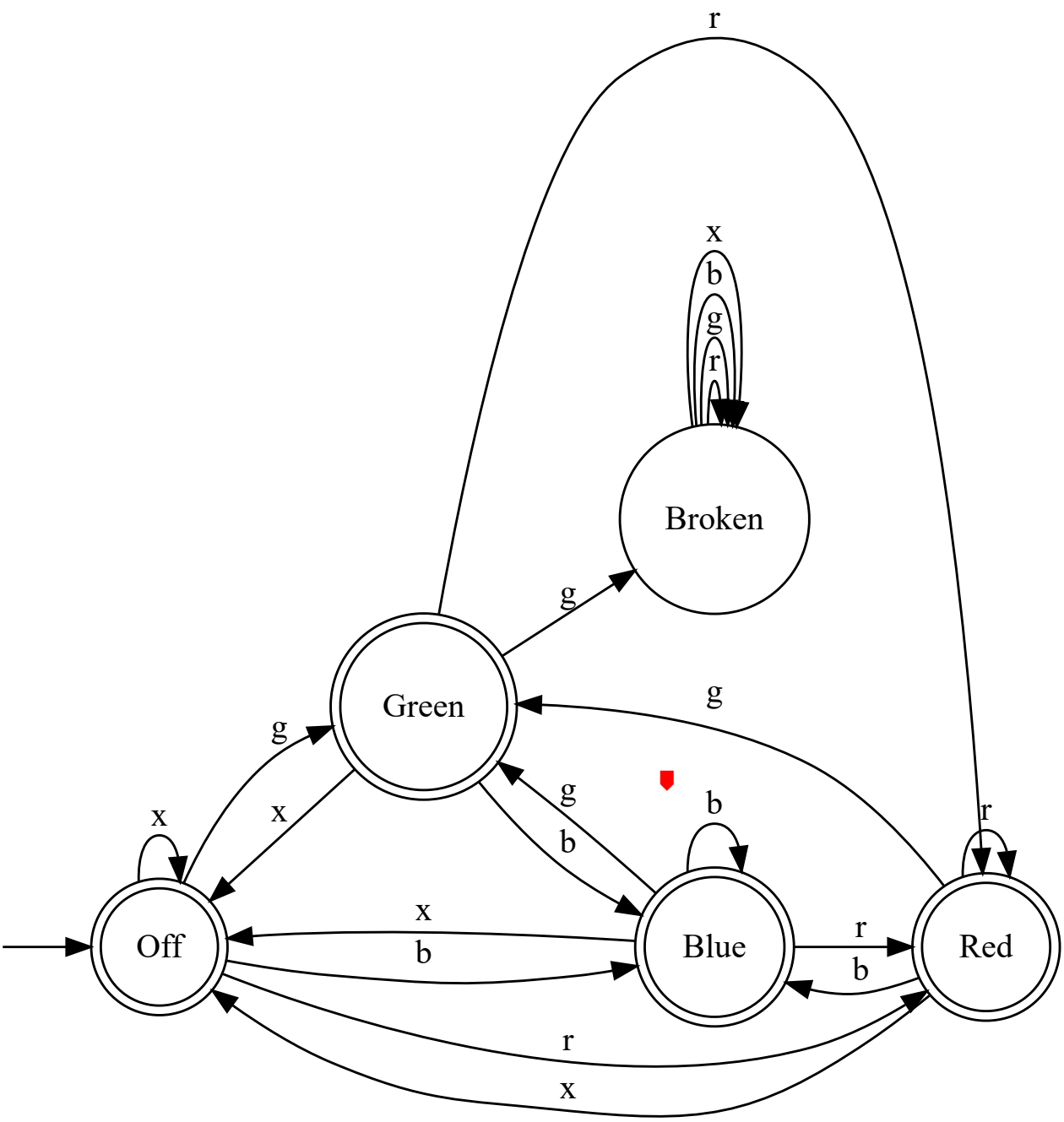
Computer scientists refer to the sort of systems that can be represented by this sort of diagram as **Finite-State-Automata** or **FSA**s. In this example, there are

only two states: "Lamp On" and "Lamp Off". But in theory, you can imagine a FSA with arbitrarily (but finitely) many states. The only requirements are that this FSA should have:

- A set of finitely many states
- Some subset of those states designated as **starting states** (in the diagram, they would have the extra arrow pointing to them, like "Lamp Off" in the above example)
- A set of **transitions** between states, associated with some action/operation or **symbol** ("press button" in this case)
- Some subset of the states designated as **final states**. These are denoted with double lines around their border (so in the above example, both states were final states). These allow an FSA to **accept or reject** a string of input actions/operations/symbols: if that string can lead the FSA to end up in a final state, then the FSA is said to accept that string, but if that is not the case, the FSA rejects the string. To illustrate an FSA with final states, consider the next example:

RGB Lamp I

Suppose you decide that a single-color lamp is too boring, so you decide to upgrade. You buy a lamp that allows you to select between three colors: red, green and blue. Correspondingly, this new lamp has more than just one button; it has four: one for each color (labeled 'r','g','b', respectively), plus one to turn the lamp off (labeled 'x'). However, unfortunately for you, this lamp shipped with an unfortunate defect: if you ever press the 'g' button while the lamp is already shining green, the whole lamp will short-circuit and stop working. To visualize this lamp as an FSA, your diagram might look something like the following, where you can simulate the FSA's classification of an input string by following the edges corresponding to symbols in the input from one state to another:

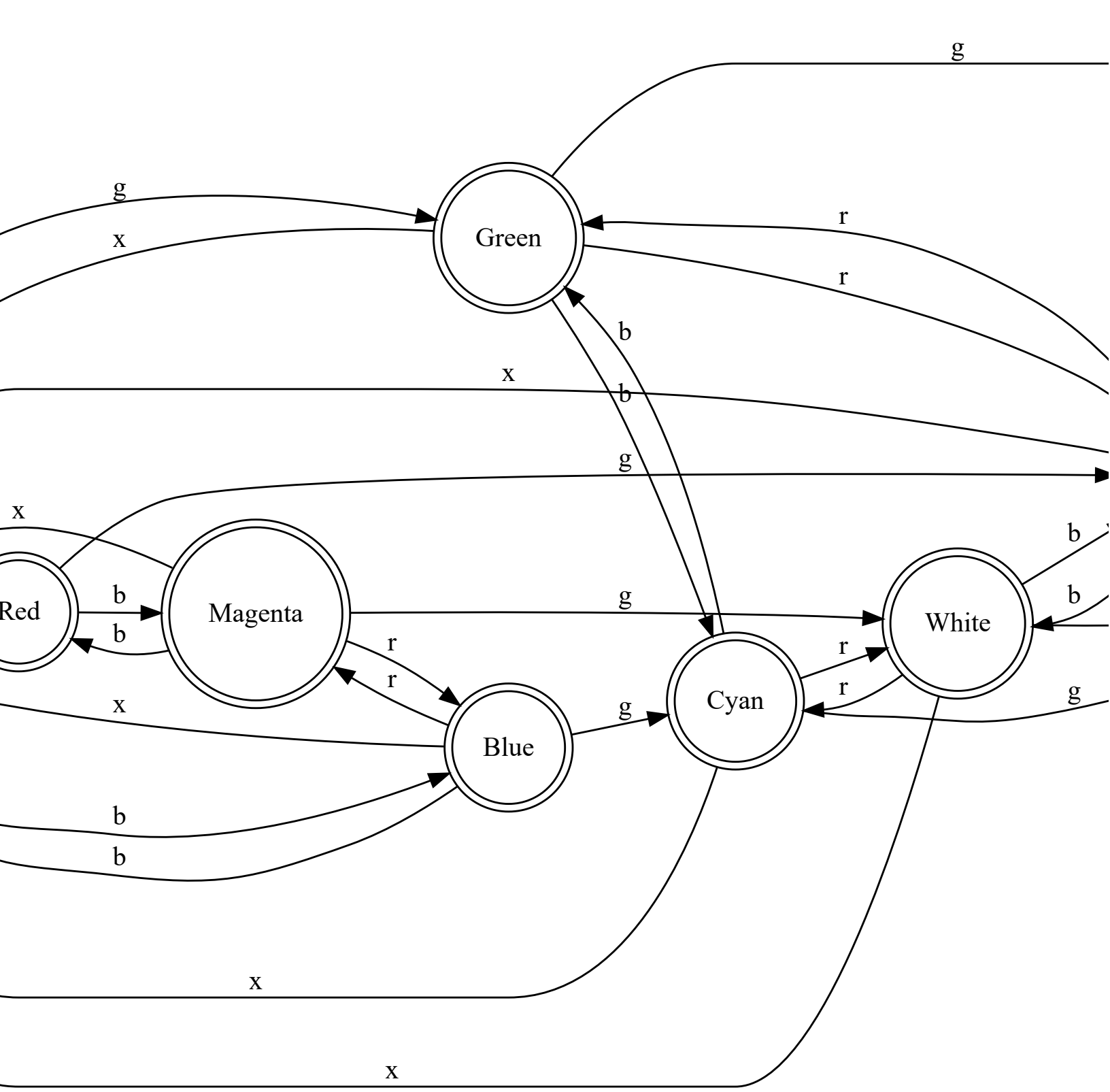


Looking at this new diagram, we see how, even though we only have three more states and three more symbols than before, the automaton is much more complicated than before! Shoutout to Jove, and edotor.net, which I used to create and render the dot code, for putting up with it. But importantly, look at the function performed by the final vs non-final states: any string of transitions that ends with the FSA in a final state corresponds to a string of button presses that ends with a functional lamp, while any string of transitions that ends with the FSA in the non-final "Broken" state, corresponds to a string of button presses that leaves you with a broken lamp. Try this for yourself: the strings 'rgbrgbrgb', 'rrrrrrrrrr', and 'gbgrgxgbgrgxgbgrgx' all leave you in a final state and with a still-functioning lamp.

Because the FSA accepts these strings, they are said to be in the FSA's **formal language**, (a set of strings which are defined by a mathematical rule. In this case, being accepted by an FSA). On the other hand, 'gg', 'rgbrgbrgbrrggbb', and 'grgbgg' are strings not in the language of this FSA.

RGB Lamp II

Suppose you take your new lamp apart and reassemble it, and you don't succeed at fixing the short in the green button, but you do manage to unlock the hidden colors that the manufacturer was trying to keep from you: Yellow, Cyan, Magenta, and White. Now, instead of a new color immediately deactivating an old color, multiple colors are allowed to shine at the same time, which means you can have Red+Green to make Yellow, Red+Blue to make Magenta, and Green+Blue to make Cyan. If you press a color's button a second time, it will turn that color's bulb off (except for green, which still causes the lamp to short). The 'x' button shuts off all three bulbs. Here is an FSA to model your upgraded lamp:



Notice that you now have to be even more careful about when you press the 'g' button: no matter how many times you press the 'r' and 'b' buttons in between two presses of the 'g' button, you will still break your lamp, unless if you press the 'x' button in between! Follow the transitions in the FSA diagram to verify that the strings 'rgbrgbrgb' and 'gbgrgxgbgrgxgbgrgx', which were in the language of the previous FSA, are not in the language of this new FSA. Strings such as 'rbrbrb', 'gxgxgx', and 'rgbrxgbrrbbxgxbbbbbb' still are accepted by this FSA.

Regular Languages and the Chomsky Hierarchy

There are infinitely many FSAs possible, and each FSA defines a corresponding language of every string they accept. The set of all sets of strings for which an FSA can be devised, is known as the **Regular Languages**. These are generally considered to be the simplest models for computation: If you have a system that can exist in one of finitely many states, and transitions from one state to the next based on some input, it can be considered a finite state automaton. This allows the system to act as a classifier for strings of inputs as within or outside of their language.

However, not every possible pattern of symbols in strings can be modeled as transitions between states of an FSA. Imagine the language of properly-nested brackets, which contains '[]', '[[]]', '[][]', and '[[[]][]][[[]]]', but not '[[[[[', ']]][[[', or '[]]]]]'. Try to design an FSA where each transition is labeled with '[' or ']', and where only valid arrngements of open and close brackets brings it to a final state. It can't be done; a valid string in this language might have arbitrarily many open brackets, but each FSA only has finitely many states to keep track of how many open brackets have been used so far, so any FSA with n-states that you might claim encodes the language of properly nested brackets, I can just come up with strings consisting of $2 * n$ layers of properly-nested brackets, which your FSA will eventually get wrong. As this bracket language shows, this is only a subset of all possible formal languages. This bracket language is known to be a **Context-Free Language**, which is a more computationally complex model for computation. Where regular languages can be modeled with a finite state automaton, context-free languages require a **push-down automaton**, which is very similar to an FSA, but with an extra capacity to remember which states it has passed through, which allows it to, for instance, keep track of how many open brackets it has read so far. Both regular languages and context-free languages are part of the **Chomsky hierarchy** of formal language complexity, where the most complex automaton is the **Turing Machine**, which is able to compute anything that can possibly be computed.

Although the regular languages are the least complex of the formal language families, they still allow for an incredible range of detailed languages. Chess, for instance, contains finitely many board states (there are 64 squares and 13 possible occupants of a square: King, Queen, Bishop, Knight, Rook, Pawn (Black or White),

or nothing; this allows us to determine a very loose upper bound of 13^{64} possible board states, although Shannon (1950) calculated an upper bound around 10^{43}) and finitely many possible transitions between states (moves), so the set of all possible sequences of moves in chess is a regular language! This brings us back to the **Subregular Hypothesis** from earlier.

Subregular Hypothesis

If you were reading closely in both the Phonotactics and Computability Theory sections above, you might have noticed something similar about the natural language phonotactic restrictions and the formal languages of the FSAs devised to model the RGB lamps: the same kinds of restrictions existed in both cases! In English, the substring 'bn' of phonemes is restricted, just like the sequence 'gg' is restricted in the first RGB lamp. In Turkish, the subsequence FILL IN THIS EXAMPLE HERE is restricted, unless if it is intervened by FILL, while in your modified lamp, the subsequence 'gg' is restricted, unless if intervened by 'x'. This means that regular language patterns can be used to model natural language phonotactics! In particular, we don't even need the full breadth of regular languages: Heinz et al. 2011 posits that a particular subset of regular languages, including TSL patterns, is enough to model these sort of restrictions. This subset, known as the **Subregular Languages** might be able to model all phonotactic patterns. This is the **subregular hypothesis**.

A lot of the work that has gone into Subregular Phonology has been related to drawing connections between these formal subsets of the regular languages and real phonotactic patterns from natural languages. De Santo & Graf (2019), for instance, define **Input-Sensitive TSL** (or **ITSL**) languages, where the decision whether a segment is projected to a tier depends not only on what phoneme it represents, but also on the phoneme represented by the adjacent segments.

Additionally, there has been work related to **Grammatical Inference**, a field that was introduced by Gold (1967). Grammatical inference refers to the ability of an algorithm to learn patterns in inputs. Human language acquisition can be seen as a complex case of grammatical inference, and particular algorithms can help prove that certain patterns can be learned in theory. I REALLY NEED TO INCLUDE

MORE HERE, BUT I AM RUNNING OUT OF TIME BEFORE THE PEER REVIEW, SO I AM SKIPPING THIS PART FOR NOW

So where did the experiment from the beginning of this article come from, and how does it connect to Subregular Languages?

If you look closely at the language sample that was given to you at first, you might notice that each word either contains "sh" OR "s" but never both (note that the sound English-speakers represent with "s" is not part of the sound represented by "sh" - so even though the letter "s" is technically reused, this is a quirk of English orthography). That is, the subsequences "s", "sh" and "sh", "s" are forbidden on the tier consisting of {"s","sh"}. So when these sounds are later found to cooccur in words in the test samples later on, it is possible that you will notice that it does not fit. Did you pick up on this pattern? Or maybe some other pattern? Most of the participants in these studies did!

Conclusion

The findings of subregular phonology show an interesting pattern that occurs in human language behavior. It is hoped that further research will provide insight into human language faculty and allow us to better model language.

But as a thought experiment, it is also an interesting journey into ideas about how to quantify the complexity of patterns, and how to design the minimal machine that can detect these patterns! It shows us how interesting it is to try to create mathematical models of complicated systems!

References

Shannon Chess Computer 1950 I WILL GO BACK THROUGH AND PROPERLY ADD ALL THESE LATER, BUT FOR WHAT ITS WORTH, I DO LINK TO A SOURCE EACH TIME I CITE SOMETHING