Analyzing negative *n*-gram grammars

In the previous unit, we entertained the idea that natural language phonotactics can be described in terms of a collection of forbidden sound sequences. Such a collection is called a forbidden *n*-gram grammar, and each *n*-gram represents one forbidden sound sequence of length *n*. But this idea ran into a problem. In German, words cannot start with *rb*, which is captured by adding the trigram $\rtimes rb$ to the list of illicit *n*-grams (remember that $\rtimes$ is a special symbol that indicates the left edge of a word). But German also has word-final devoicing, which means that no word can end in a voiced *s*-sound, which corresponds to *z* in English orthography. Using this notational convention, we can represent the illicit sequence as $z\ltimes$ (with $\ltimes$ as the right edge counterpart of $\rtimes$). This is a bigram, whereas $\rtimes rb$ is a trigram. What are the consequences of mixing bigrams and trigrams in a single *n*-gram grammar? Does this introduce inconsistencies such that a word is both forbidden and allowed? Let's try to answer this question. Fair warning: this will be a bit of a pain, and that's actually the point of this unit.

A plea for proofs

Unless you're a mathematician, your first instinct will be to work through one or more examples. If we, say, construct 100 grammars that include bigrams and trigrams and don't run into any problems, it's probably fine to mix and match. But this has several downsides:

- **So much work...**
  Constructing and testing 100 grammars doesn't exactly sound like fun. Sure, you could write a program to do it for you, but that's also work. And your program might have bugs, which takes us to the next point.
- **No guarantees**
  Just because your simulations produce a certain result does not mean that things always work this way. If you ask 1000 people if they've ever read Werner Schwab's *Fäkaliendramen* (feces dramas), the answers will probably all be No. But if your sample happened to include mostly literature buffs, you'll hear Yes a lot more often. Similarly, the grammars you construct for your simulation might exhibit a special property that allows them to pass the test. There might still be grammars that do not display this property and fail. If you construct the grammars with a program, your code might be written in a way that only produces grammars of a specific type. Bottom line: if you are testing based on examples, you can never be sure that your examples cover all possible cases that need testing.
- **No scalability**
  Alright, suppose you actually wrote a program that is free of any bias and ran a huge number of simulations for grammar with mixed bigrams and trigrams. You now feel very confident that mixing of bigrams and trigrams is unproblematic. What about 4-grams? 5-grams? 127-grams? How do you know that your results will carry over from bigrams and trigrams to arbitrary *n*-grams?

Computer-aided simulations are *en vogue* nowadays, but they're really a last resort. They are hard to design, often require significant resources, and do not provide perfect insight into how the specific aspects of a system determine its behavior. That's not to say that simulations are a bad thing — if you are dealing with a very complex system, they're often the best tool at your disposal. But there are other tools around, and in many cases they are a superior choice. Mathematics furnishes the best tool of them all: proofs!

A mathematical proof starts out from a fixed set of assumptions and shows how these assumptions entail a specific property. We will see a concrete example in a moment, but let's first focus on the specific advantages of proofs:

- **Laziness**
  Hard proofs are very, very hard. They are much harder than designing a simulation. But unless you are a professional mathematician, most proofs you'll ever see are fairly easy. And easy proofs are very, very easy. You can often work them out in less than 30 minutes, and they only take a few lines to write down.

- **Guarantees**
  Of course a proof may contain mistakes, just like a program may contain bugs. But mistakes in a proof are easier to spot than bugs in a program. Fixing a proof is also much simpler than verifying that a simulation has no hidden biases. Once you have a correct proof, you have a guarantee: as long as the assumptions of the proof hold, the property established by proof holds, too.

- **Scalability**
  Since a proof holds as long as its initial assumptions are satisfied, it can be extended to any object that satisfies these assumptions.

This may all sound awfully abstract to you. So let's finally turn to our first proof, because the proof of the pudding is in the eating (sorry, I couldn't resist). We will show that a negative n-gram grammar that also contains, say, bigrams and trigrams can be converted to an equivalent n-gram grammar that only contains n-grams. While reading through the proof, keep in mind the three properties above (laziness, guarantees, scalability), and think about how they're instantiated in the proof.

Our first proof: Mixed n-gram grammars

Formalizing the problem

Our initial question is whether there is any problem with mixing bigrams and trigrams in a single grammar. This is not a very precise question. What exactly constitutes a problem? If you have to write on a tiny piece of paper that can barely hold a single bigram, adding a trigram to the mix creates problems. That's obviously not our concern here. Remember that we use negative n-gram grammars as a model of natural language phonotactics. So their job is to describe which potential words are well-formed and which are ill-formed. We would have a problem if this failed for some reason: a word is both well-formed or ill-formed, a word that is ill-formed suddenly becomes well-formed when bigrams and trigrams are mixed, or the other way round.

We will show that none of these issues ever arise; we do so by establishing a **normal form theorem**. Whenever a grammar contains n-grams of different sizes, it can be converted to a grammar where all n-grams are of the same size. This converted grammar is equivalent to the original in the sense that they make the same phonotactic judgments: the first grammar deems a word well-formed iff the second one does, too, and the two also agree on which words are ill-formed. The second grammar thus behaves exactly like the first, but has a normalized form without any n-grams of different length. That's why the second grammar is called a **normal form** of the first one. **Theorem** is just a fancy term for a statement that follows from a fixed set of assumptions. So we are proving a theorem about the existence of a normal form, hence the term **normal form theorem**.

In order to avoid an overload of notation and terminology, we state the theorem in a slightly inaccurate manner as follows:

Theorem 1. Let G be some collection of sound n-grams of different lengths where k is the length of the longest n-gram. Then there exists an equivalent grammar G' such that every n-gram of G' has length k.

Proof

We start with a few key observations.

- Every language has only finitely many sounds. The precise number does not matter here, it could be 2, it could be 2 trillion trillion. The important thing is that it is finite. We use $|\Sigma|$ as the special symbol for this number.

Example 1 If a language only has the sounds *a*, *u*, *i*, *b*, *m*, *d*, *g*, and *h*, then $|\Sigma|$ for that specific language is 8.

- As stated in the theorem, the longest n-gram is assumed to have length k. Since each position is filled by either a sound or one of two edge markers ($\rtimes$ or $\ltimes$), there are $|\Sigma| + 2$ choices for each position. Consequently, there are at most $(|\Sigma| + 2)^k$ different n-grams of length k. This implies that G contains at most $(|\Sigma| + 2)^k$ n-grams and thus contains only finitely many.

Example 2 Suppose that the language only has the sounds *a* and *d*, barely enough for *dada*. It's $|\Sigma|^k$ is 2. There are $(2 + 2)^3 = 4^3 = 64$ different trigrams. Not all of them are ever useful. In particular, no word ever contains an edge marker in the middle, so $a\rtimes a$ and $a\ltimes a$ serve no purpose. It is also impossible for $\ltimes$ to occur to the left of $\rtimes$, which rules out trigrams like $\ltimes a\rtimes$ and $\ltimes\rtimes a$. Filtering out those useless trigrams leaves us with the following list:

1. aaa
2. aad
3. aa$\ltimes$
4. ada

5. add
6. ad⋉
7. a⋉⋉
8. daa
9. dad
10. da⋉
11. dda
12. ddd
13. dd⋉
14. d⋉⋉
15. ⋊aa
16. ⋊ad
17. ⋊a⋉
18. ⋊da
19. ⋊dd
20. ⋊d⋉
21. ⋊⋊a
22. ⋊⋊d
23. ⋊⋊⋊
24. ⋊⋊⋉
25. ⋊⋉⋉
26. ⋉⋉⋉

But even if we had included all useless trigrams, that would not change the fact that there are only finitely many trigrams over *a*, *d*, and the edge markers.

Now suppose that we pick one of the finitely many *n*-grams of *G*. Call it *g*. If the length of *g* is already *k*, it is one of the longest *n*-grams and we don't need to do anything. But if its length *i* is strictly less than *k*, we need to replace *g* by something equivalent. We remove the *n*-gram *g* from the grammar *G*, and instead add in a number of "padded" variants of *g*:

- Construct every possible *n*-gram of length $k - i$. For each such *n*-gram, put it **in front of** *g* and add the result back to *G*.

Example 3

Suppose *G* contains the bigram *z*⋉, the trigram ⋊*kn*, and the 4-gram *akzn*. Assume furthermore that the only possible sounds are *a*, *k*, *z*, and *n*.

We have to pad out *z*⋉ from a bigram to a 4-gram. The length difference between a bigram and a 4-gram is 2, so we have to put bigrams in front of *z*⋉. The list of possible (and useful) bigrams is as follows:

- ⋊⋊
- ⋊⋉
- ⋉⋉
- ⋊a
- ⋊k
- ⋊z
- ⋊n

- a⋉
- aa
- ak
- az
- an
- k⋉
- ka
- kk
- kz
- kn
- z⋉
- za
- zk
- zz
- zn
- n⋉
- na
- nk
- nz
- nn

So we remove *z*⋉ from *G* and instead add all of the following. Note that not all of those 4-grams are useful, but that doesn't matter here.

- ⋊⋊z⋉
- ⋊⋉z⋉
- ⋉⋉z⋉
- ⋊az⋉
- ⋊kz⋉
- ⋊zz⋉
- ⋊nz⋉
- a⋉z⋉
- aaz⋉
- akz⋉
- azz⋉
- anz⋉
- k⋉z⋉
- kaz⋉
- kkz⋉
- kzz⋉
- knz⋉
- z⋉z⋉
- zaz⋉
- zkz⋉
- zzz⋉
- znz⋉
- n⋉z⋉
- naz⋉
- nkz⋉

- nzz⋉
- nnz⋉

- Construct every possible n-gram of length $k - i$. For each such n-gram, put it **after** g and add the result back to G.

Example 4 We also add the following 4-grams to G:

1. z⋉⋊⋊
2. z⋉⋊⋉
3. z⋉⋉⋉
4. z⋉⋊a
5. z⋉⋊k
6. z⋉⋊z
7. z⋉⋊n
8. z⋉aa
9. z⋉ak
10. z⋉az
11. z⋉an
12. z⋉a⋉
13. z⋉ka
14. z⋉kk
15. z⋉kz
16. z⋉kn
17. z⋉k⋉
18. z⋉za
19. z⋉zk
20. z⋉zz
21. z⋉zn
22. z⋉z⋉
23. z⋉na
24. z⋉nk
25. z⋉nz
26. z⋉nn
27. z⋉n⋉

Except for $z⋉⋉⋉$ these are all useless because ⋉ cannot occur between two symbols that aren't edge markers. But we add them anyways to stick with the procedure.

- Finally, pick any two n-grams of length i' and i'' such that $i' + i + i'' = k$ (remember that i is the length of the n-gram g that must padded out, and k is the length of the longest n-gram in the grammar). Sandwich g between those n-grams and add the result to the grammar G.

Example 5 Since the difference between a 4-gram and a bigram is 2, each one of the two "sandwich" n-grams must have length 1. So we add the following:

1. ⋊z⋉⋊
2. ⋊z⋉⋉
3. ⋊z⋉a
4. ⋊z⋉k
5. ⋊z⋉z
6. ⋊z⋉n
7. az⋉a
8. az⋉k
9. az⋉z
10. az⋉n
11. az⋉⋊
12. az⋉⋉
13. kz⋉a
14. kz⋉k
15. kz⋉z
16. kz⋉n
17. kz⋉⋊
18. kz⋉⋉
19. zz⋉a
20. zz⋉k
21. zz⋉z
22. zz⋉n
23. zz⋉⋊
24. zz⋉⋉
25. nz⋉a
26. nz⋉k
27. nz⋉z
28. nz⋉n
29. nz⋉⋊
30. nz⋉⋉

Again there are many useless *n*-grams, but we do not care.

The new grammar G' constructed this way is equivalent to G. To see this, suppose that some string is ill-formed according to G. Then some n-gram g of G must occur in the string, otherwise it would not be deemed ill-formed by G.

- **Case 1**
  If g has length k, then it is also an n-gram of G', so G' would consider the string illicit, too.

Example 6 Remember that our example grammar G disallows $z⋉$, $⋊kn$, and $akzn$. The construction above constructs G' by padding out $z⋉$ and $⋊kn$ to 4-grams, but it keeps $akzn$ the same. So if some word is forbidden by G because it contains $akzn$, it will also be forbidden G'.

- **Case 2**
  Now assume that g's length is less than k. Then G' contains padded variants of g that have length k and are equivalent to g. Here is why:

- Remember that a string for an n-gram grammar is padded with $n-1$ edge markers. So with respect to G', whose longest n-gram has length k, every string has $k-1$ edge markers to its left and $k-1$ edge markers to its right. This means every string has at least length $2 \times (k-1) = 2k-2$, which is greater than k.
- Consider once more the illicit string, whatever it may be. Somewhere inside the string is an offending instance of the illicit n-gram g. There must be symbols to its left and right, at the very least some edge markers. We know this because $n \leq k < 2k-2$. But G' contains every padded version of g, i.e. g with 0 or more symbols to its left and right. So if a string contains g, it also contains some illicit padding of g.

Example 7 Consider the word *kaz*, which is illicit because it contains $z\ltimes$ (remember, we always add a sufficient number of edge markers). This string is still considered illicit by G'. The padded out word is $\rtimes\rtimes\rtimes kaz\ltimes\ltimes\ltimes$, and several of the illicit 4-grams we constructed from $z\ltimes$ are contained in this string:

- kaz⋉
- az⋉⋉
- z⋉⋉⋉

This shows that every string that is deemed illicit by G is also illicit with respect to G'. We still have to show the opposite, which is much easier. Suppose that a string is ruled out by G' because it contains the n-gram g.

- **Case 1**
  G contains g, too. Then G also deems the string illicit.
- **Case 2**
  G does not contain g. Then g was obtained by padding out some smaller n-gram f of G. But every string that contains an instance of g must also contain an instance of f. So G still considers the string illicit.

Since G and G' agree on which strings are illicit, they necessarily agree on which strings are well-formed. So by carrying out the procedure above for every n-gram of G, one obtains a grammar G' that is equivalent to G but only contains n-grams of a fixed length.

Some thoughts

You might cry foul at this point. I promised you that proofs are easy and only take a few lines, and the one above is neither. It's very long, and it's cumbersome to read, and the sentences are hard to make sense of. But that's because everything was explained in plain English rather than mathematical notation. This made the proof harder to read and much longer, and it also means that we had to rely on examples to explain what exactly is intended at each step of the proof. The next unit will present the same proof in mathematical notation, and while it may be initially harder for you, this format will be a lot easier for you once you have some experience.

In fact, this is why it is so helpful to learn math. Many things are intuitive enough that they can be explained in plain English. But it is clumsy, imprecise, and takes longer. Specialized notation and terminology makes things easier to talk and think about, not harder.