We deal with the whole question of weight pushing in a slightly different way from the traditional recipe. Weight pushing in the log semiring can be helpful in speeding up search (weight pushing is an FST operation that ensures that the arc probabilities out of each state "sum to one" in the appropriate sense). However, in some cases weight pushing can have bad effects. The issue is that statistical language models, when represented as FSTs, generally "add up to more than one" because some words are counted twice (directly, and via backoff arcs).

We decided to avoid ever pushing weights, so instead we handle the whole issue in a different way. Firstly, a definition: we call a "stochastic" FST one whose weights sum to one, and the reader can assume that we are talking about the log semiring here, not the tropical one, so that "sum to one" really means sum, and not "take the max". We ensure that each stage of graph creation has the property that if the previous stage was stochastic, then the next stage will be stochastic. **That is: if G is stochastic, then LG will be stochastic; if LG is stochastic, then det(LG) will be stochastic; if det(LG) is stochastic, then min(det(LG)) will be stochastic, and so on and so forth. This means that each individual operation must,** in the appropriate sense, **"preserve stochasticity".** Now, it would be quite possible to do this in a very trivial but not-very-useful way: for instance, we could just try the push-weights algorithm and if it seems to be failing because, say, the original G fst summed up to more than one, then we throw up our hands in horror and announce failure. This would not be very helpful.

We want to preserve stochasticity in a stronger sense, which is to say: first measure, for G, the min and max over all its states, of the sum of the (arc probabilities plus final-prob) out of those states. This is what our program "**fstisstochastic**" does. If G is stochastic, both of these numbers would be one (you would actually see zeros from the program, because actually we operate in log space; this is what "log semiring" means). We want to preserve stochasticity in the following sense: that this min and max never "get worse"; that is, they never get farther away from one. In fact, this is what naturally happens when we have algorithms that preserve stochasticity in a "local" way. **There are various algorithms that we need to preserve stochasticity, including:**

- **Minimization**
- **Determinization**
- **Epsilon removal**
- **Composition** (with particular FSTs on the left) There are also one or two minor algorithms that need to preserve stochasticity, like adding a subsequential-symbol loop. **Minimization** naturally preserves stochasticity, as long as we don't do any weight pushing as part of it (we use our program "**fstminimizeencoded**" which does minimization without weight pushing). **Determinization** preserves stochasticity as long as we do it in the same semiring that we want to preserve stochasticity in (this means the log semiring; this is why we use our program **fstdeterminizestar with the option –determinize-in-log=true**). **Regarding epsilon removal**: firstly, we have our own version of epsilon removal "RemoveEpsLocal()" (**fstrmepslocal**), which doesn't guarantee to remove all epsilons but does guarantee保证 to never **"blow up"**放大. This algorithm is unusual特别的 among FST algorithms in that, to to what we need it to do and preserve stochasticity, it needs to "keep track of" two semirings at the same time. That is, if it is to preserve equivalence in the tropical semiring and stochasticity in the log semiring, which is what we need in practice, it actually has to "know about" both semirings simultaneously. This seems to be an edge case where the "semiring" concept somewhat breaks down. **Composition** on the left with the lexicon L, the context FST C and the H tranducer (which encodes the HMM structure) all have to preserve stochasticity. Let's discuss this the abstract: we ask, when composing A o B, what are sufficient properties that A must have so that A o B will be stochastic whenever B is stochastic? We believe these properties are:
  - For any symbol sequence that can appear on the input of B, the inverse of A must accept that sequence (i.e. it must be possible for A to output that sequence), and:
  - For any such symbol sequence (say, S), if we compose A with an unweighted linear FST with S on its input, the result will be stochastic.

These properties are true of C, L and H, at least if everything is properly normalized (i.e. if the lexicon weights sum to one for any given word, and if the HMMs in H are properly normalized and we don't use a probability scale). However, in practice in our graph creation recipes we use a probability scale on the transition probabilities in the HMMs (similar to the acoustic scale). This means that the very last stages of graph creation typically don't preserve stochasticity. Also, if we are using a statistical language model, G will typically not be stochastic in the first place. What we do in this case is we measure at the start how much it "deviates from stochasticity" (using the program **fstisstochastic**), and during subsequent graph creation stages (except for the very last one) we verify that the non-stochasticity does not "get worse" than it was at the beginning.