FSA properties that FSTs lack

1 No determinizability

The powerset construction allows us to determinize non-deterministic FSAs. That is to say, for any non-deterministic FSA we can construct a fully equivalent, deterministic counterpart. The potential cost of determinization is a greatly increased state space. In some cases, the number of states does not grow at all, but in the worst case the blow-up is exponential (2^n for a non-deterministic FSA with n states). While that is not insignificant, at least we can always determinize arbitrary FSAs if we're willing to pay the price. For some FSTs, that's not even an option.

Example 1.
Consider the FST below. It takes as its input strings of the form $\{a, b\}c^*\{a, b\}$ and maps them to d^+ or s^+ depending on whether the first symbol in the string matches the last symbol in the string. For instance, *acccb* becomes *ddddd*, but *accca* becomes *sssss*.

NFST

The FST is non-deterministic because the transducer has to guess right away whether the last symbol will be *a* or *b* in order to decide what to replace each symbol with. Since FSTs cannot look ahead arbitrarily far into the string, guessing is the only option. And this is exactly why the transducer cannot be determinized. A deterministic transducer does not get to guess. It can only do one thing when it sees the first symbol. Either it always rewrites it as *d*, or it always rewrites it as *s*. Neither option works in all cases.

Example 2.
The FST below maps every string in a^* to $\{b, c\}^*$.

NFST

This transducer is necessarily non-deterministic because one input can be mapped to multiple outputs. That's impossible with a deterministic transducer. Deterministic FSTs compute functions (no input has more than one output), whereas the transducer above computes a relation.

The examples show that deterministic FSTs are much more limited than non-deterministic ones. Whenever one input has multiple outputs, one needs a non-deterministic transducer. For natural language, this mostly affects cases of optionality, such as whether *dream+PAST* should be *dreamed* or *dreamt*. But even when every input has at most one output, the transduction might still require a non-deterministic FSTs. This is usually the case when the transducer would have to make a decision

based on a symbol that can be arbitrarily far to the right of the decision point. In these cases, non-determinism is needed to compensate for the lack of look-ahead.

2 No intersection closure

In contrast to FSAs, FSTs also lack closure under intersection. The intersection of two finite-state transductions is not guaranteed to be a finite-state transduction.

Example 3.
Consider the following two finite-state transductions: one rewrites every c^n by a^*b^n, the other one rewrites c^n by a^nb^*. Their intersection rewrites every c^n by a^nb^n. So the intersection maps the string language c^* to the string language a^nb^n. But we already know from the Myhill-Nerode theorem that a^nb^n is not regular. Howver, c^* is clearly regulary, and finite-state transductions preserve regularity. So if c^* is regular, but a^nb^n is not, then the transduction that maps the former to the latter cannot be finite-state. But this transduction was the intersection of two finite-state transductions, which show that closure under intersection does not hold.

Exercise 1.
Use the same procedure to show that the intersection of multiple finite-state transduction can produce the output language $a^nb^nc^nd^ne^n$.

3 Intersection closure for equal length relations

There is one specific subcase of finite-state transductions that are closed under intersection. Those are the **equal length** relations. A finite-state transduction is an equal length relation iff an output always has the same length as its input. In other words, the FST only rewrites symbols in the input string, it does not add or remove any symbols. Equal length relations can be regarded as regular languages, and since regular languages are closed under intersection, equal length relations are too.

To see why equal length relations are regular languages, consider the simplest FST possible, the identity function. This transduction maps every string to itself and can be regarded as a set of pairs $\langle s, s\rangle$, where s is some string over Σ. But since input and output have the same length, we may regard $\langle s, s\rangle$ as a single string of symbol pairs.

Example 4.
The pair $\langle aaba, aaba\rangle$ relating an input string to an output string can be regarded as the single string $\langle a, a\rangle\langle a, a\rangle\langle b, b\rangle\langle a, a\rangle$. So instead of a pair of strings over the

alphabet Σ, we have a single string over the alphabet $\Sigma \times \Sigma$.

As long as an FST cannot delete or insert arbitrarily many symbols, it can be converted into an equal length relation by padding out inputs and outputs with a special symbol, e.g. 0.

Example 5.
Consider an FST that simplifies consonant cluster such that *rdn* becomes *dn* (as is the case in Icelandic). Then a string like *bardn* becomes *badn*. Here the length of input and output are distinct, so the FST is not an equal length relation. However, we can opt to replace *r* by *0* instead of deleting it. Then the output for *bardn* is *ba0dn*. Now input and output have the same length.

Example 6.
Suppose that another language optionally resolves *rdn* clusers by inserting an *e* between *d* and *n*. Then *bardn* becomes *barden*. Again input and output have distinct lengths. But we can use *bard0n* as a padded out version of the input instead. This input can be mapped to *barden* (cluster resolution) or *bard0n* (no resolution). Either way the output has the same length as the input.

Some formalisms such as two-level morphology make extensive use of such padding symbols to ensure that all transductions are equal length relations. While this grants intersection closure, it means that inputs and outputs are cluttered with padding symbols, which makes them harder to work with.