Why do we need to convert from NFA to DFA?

The short answer is you don’t. A recent pattern matching tool I worked on at Intel called hyperscan worked directly with NFAs for certain aspects of the problem. There are even pattern matching tools that work directly from regular expression notation.

However, when you can convert from an NFA to a DFA, there are easy implementations of a DFA that run quite fast, generally faster than an NFA. The problem with converting an NFA to a DFA is that the process can cause an explosion the number of states. Depending on how the NFA is run, this explosion can be present as a large number of states active simultaneously and thus in memory.

In slight contrast to what the other answer said, a DFA generally has more states than a Glushkov NFA, but might have less than the more familiar Thompson NFA, because the Thompson NFA can have lots of epsilon states.

The hint is in the expanded name. Computer programs generally need to know all possible transitions and states for a given state machine. A non-deterministic finite automaton can have a transition that goes to any number of states for a given input and state. This is a problem for a computer program because it needs precisely one transition for a given input from a given state. The process of converting NFA to DFA eliminates this ambiguity and

allows a program to be made (the context here, I assume, is compiler construction).

NFA also suffers from combinatorial explosion. Reducing to a DFA can reduce state and transitions.