Unit 4

Automata as Transition Networks

Exercise 4.1. Optionality is often very easy to handle with finite-state automata, as is the case with iterable adverbs:

(1) a. John likes Mary.
b. John really likes Mary.
c. John really, really likes Mary.
d. John really, really, really likes Mary.

Sentence-final optionality (or what looks like optionality) is also readily accounted for.

(2) a. John ate the pasta.
b. John ate.
c. John devoured the pasta.
d. * John devoured.

(3) a. The Bride wants Bill dead.
b. The Bride wants Bill.
c. The player looked the opponent off.
d. * The player looked the opponent.

Draw epsilon-free automata for sentences of this form (an automaton is epsilon-free iff no transition is labeled with the empty string ε). Do not use transition networks to represent the automata more succinctly. ⊙

Exercise 4.2. Continuing the previous exercise, consider now the following apparent case of optionality:

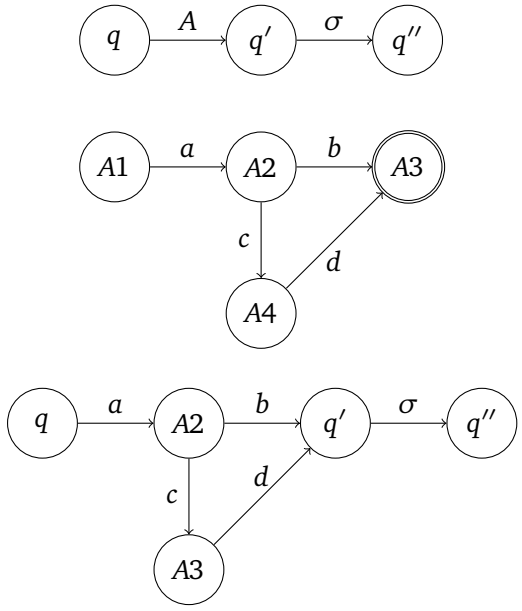
(4) a. Who might John like and may his friend from many years ago hate?
b. Who might John like and his friend from many years ago hate?
c. Who might John like and hate?

Draw an epsilon-free automaton that correctly handles the sentences above and other cases with reasonably complex subject NPs (proper names, pronouns, various determiners, adjectives, PP modifiers, but no relative clauses). Do not use a transition network to represent the automaton more succinctly. Is this case as easily handled as previous instances of optionality? What might be the explanation for this? ⊙

Exercise 4.3. Now draw an automaton for the same pattern, but with ε-transitions where appropriate. Which one of the two seems preferable to you? Explain why. ⊙

Exercise 4.4. Still continuing the previous exercise, give a transition network for the same pattern such that all automata are epsilon-free. Remember that we can pop out of a sub-automaton whenever we are in an empty state. How does your transition network fare in comparison to the two automata from the previous exercise? ⊙

Exercise 4.5. Specifying finite-state automata via transition networks is not without downsides. So far, we saw that the transition network can be compiled out into a finite-state automaton as long as limit cycles. The procedure is simple: if we have an arc labeled A from state q to q', replace that arc with sub-automaton A.



But there are side effects. Show that compiling out a transition network of deterministic automata may yield a non-deterministic automaton.

Hint: A minor modification to the example above is sufficient to produce a non-deterministic automaton. ⊙

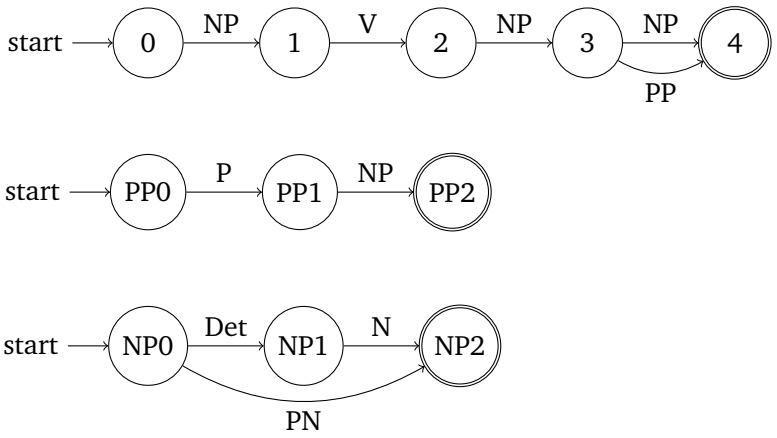
Since determinizing a non-deterministic automaton with n states may produce an automaton with 2^n states in the worst case, this shows just how succinct a description format transition networks are.

Exercise 4.6. The simple algorithm for compiling out transition networks is also lacking in efficiency, and often one can construct a more efficient automaton by hand. But the gain in efficiency comes at the cost of clarity and generality.

Consider the well-known NP/PP alternation with ditransitive verbs in English:

(5) a. John gave the woman the book.
    b. John gave the book to the woman.

Arguably the most natural account is depicted by the transition network below.



Use our standard algorithm to construct the automaton described by this transition network. Do you notice any inefficiencies? How could one make this automaton more compact? ⊙

Deterministic finite-state automata can be minimized automatically, and one is guaranteed to always end up with the smallest possible automaton. So if one wishes to convert a transition network into a compact format that is easily parsed, one should proceed as follows:

- Use the standard algorithm to construct an automaton from the transition network.
- Determinize the automaton (so that parsing can be done in linear time).
- Minimize the automaton (so that the number of spaces is small).

Nonetheless, unless one sets the cut-off point for cycles very low, one will usually end up with an automaton that is so large that it is actually faster to use a parsing algorithm that works directly with transition networks.