# A Small Example DFA Program

Luke Mathieson

August 15, 2019

## 1 Going from a language to a program

While the process of converting a language into a DFA an thence into a program is relatively mechanical, it is not necessarily transparent, particularly for the first time. This document and associated code gives a small example of the process that may be useful in thinking about how to approach assignment 1.

### 1.1 The Language

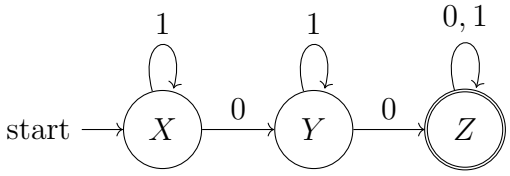
We'll use a relatively simple language, but one with enough components to make it not-quite-trivial:

$$L = \{x \in \mathbb{B}^* \mid x \text{ has at least two '0's and no more than one '1'.}\}$$

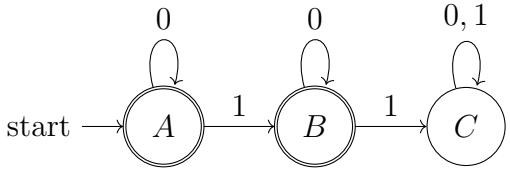
### 1.2 The DFA

While this language is small enough to puzzle out a DFA for manually, we can use the product construction to combine simpler DFAs, of which we may be more sure.

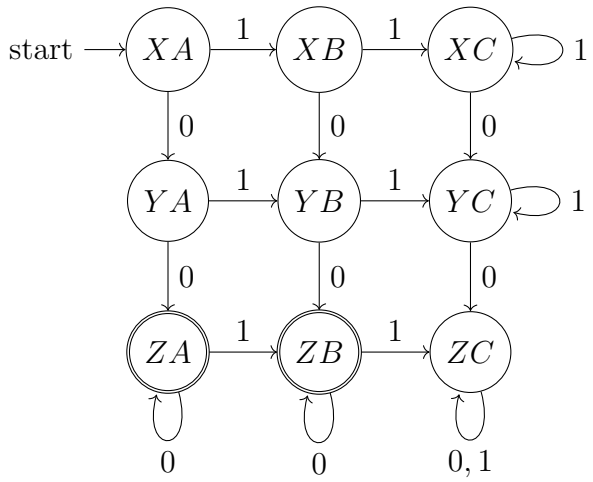
Thus we start with a DFA for the language $L_1 = \{x \in \mathbb{B}^* \mid x \text{ has at least two '0's}\}$:



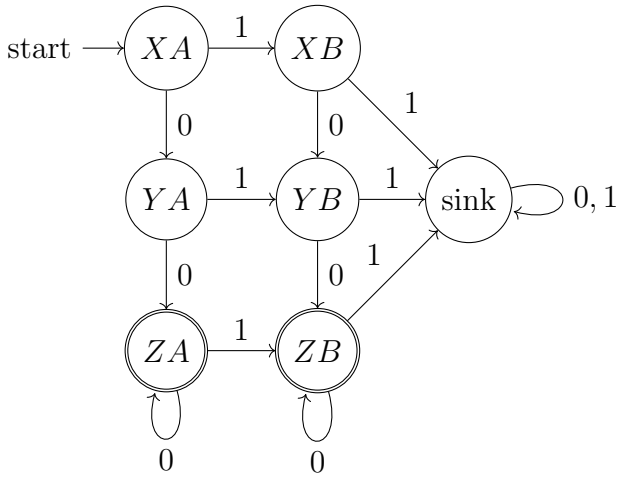

and a DFA for the language $L_2 = \{x \in \mathbb{B}^* \mid x \text{ has no more than one '1'}\}$:




It should be apparent that $L = L_1 \cap L_2$ (i.e. the set of strings that satisfy the conditions of both $L_1$ and $L_2$). Thus we can use the product construction to build the following DFA:




Before moving on, think for a moment about the meaning of states in all three DFAs.
We can minimize this latter DFA to get the following:




While at the point of writing we haven't seen the algorithm to do this, it should be relatively clear why we can make this simplification, if you understand the roles of each of the states.

## 1.3 Converting this to code

Now that we have a transition diagram for our language (i.e. a DFA), constructing this in any programming language is relatively simple. The core functionality is to replicate the action of the DFA, so we need three basic things:

1. a variable to keep track of the current state,
2. a variable to keep track of where we're up to in the input, and
3. code that replicates the operation of the transition function $\delta$.

We can add some nice things to this to make it more readable, or easier to program, or more robust, but these are the basics. The replication of the transition function is also quite simple in an of itself, all you need is a bunch of `if` statements. Of course a `switch` is a bit nicer, and some transition functions may be more amendable to clever approaches, but at base, that's it.

Now take a look at the accompanying code, and match up the pieces of the code to the DFA.