The app has some notable limitations. The major one is that even with the speak and listen actions added in part 2, the implementation duplicates a lot of code. I attempted to adhere to the flowchart in the lab description such that all states in the chart has its own outer state in the machine, but this (at least in the current implementation) results in quite bloated states with a lot of near-identical substates between the outer states. For example, all outer states have a substate “TryAgain”, which is entered when the ASR either does not pick up audio or does not have the input utterance in the grammar. The only thing different between the versions of this state is the state it transitions to (the current outer state). I could not find an xstate or javascript way to generalise over states to implement them in a more efficient way. An attempt was a function like this, but it does not mesh with state logic:

```
function tryAgain(state_string) {
  return const = {entry: {type: "speak", params: `I'm sorry, I didn't catch
that`}, on: { SPEAK_COMPLETE: state_string}}}
```

A state-based fix could be to have outer states for TryAgain (and Prompt, etc.) that is transitioned to. The specific strings to be uttered and states to be redirected to could use a history and/or context assignment.

Another issue is that the implementation can only understand utterances which are entirely in the grammar; it would not, for example, catch ‘Vlad’ in ‘I’m meeting Vlad’. A fix could be to just check if the grammar items are contained in the utterances. There are some additional difficulties with this, such as that negated affirmatives often contain their opposite. Similarly, the implementation only recognises full hours, which could be naturally be fixed by adding more options, but also by rounding non-included times to the closest time in the grammar.

Finally some issues that are also noted in the code. For some reason, my ‘speak’ action cannot take context values as part of their utterances. I assume the solution has something to do with dynamic parameters, but I could not figure it out. I also did not manage to include the logic [if guard condition met: go to state and perform action, elif other guard condition met: go to other state and perform other action, else go to third state, so I had to implement this in a clunky was as described in the code.