UNIVERSITY OF GOTHENBURG

# Assignment 2 - VG part

Student name: *Dominik Künkele*

Course: *Dialogue Systems (LT2216)*
Due date: *February 17th, 2022*

### Improvements

**duplicate state nodes.** All state nodes which ask for an answer have the same structure. There is a *prompt* state, in which the machine asks the question. After the machine is finished uttering, it moves on to the *listen* state in which it listens to a response of the user. When the machine recognised something, it either moves to another state or utters a clarification request if something wasn't understood. If the user didn't answer at all, it repeats the question. The only difference over all prompts is, how the machine handles the answer and to which state the *FSM* moves on. With a naive aproach, this results in a lot of duplicate code for each state node for prompts. If one would like to add for example a *pause* state in the prompt machine, this would need to be added in all state nodes.

To tackle this problem, I tried to encapsulate the prompt machine in a function *abstractPromptMachine*. This holds all the basic necessary states and can be extended with transitions or additional information. There is for example the *binaryPromptMachine*, which uses two conditions to decide, in which state the *FSM* should move on. Furthermore, I implemented the *categoryPromptMachine*, which can handle different assignments of context variables based on multiple grammars. Lastly, the *abstractPromptMachine* can also be extended directly in the state node defintion, as done for example for the main menu.

This approach gives the developer a lot more flexibility, to do changes necessary for all or only some specific prompt machines.

**non-flexible grammar.** Defining a grammar, which translates each possible utterance character by character into the target text has many disadvantages: First of all, the voice recognition can change. When in one NLU module an uttered sentence may be recognized as "Thank you.", in another NLU module it may be recognized as "Thank you!". A change of one character may invalidate the current implemented grammar. This could be handled, by extending the grammar with many possible different ways how the sentences could be uttered. That would lead to the next problem: There are far too many possible ways, to include each in the grammar. This becomes especially visible, when trying to parse an uttered time. Just including all different minutes of an hour would be a huge effort and would bloat the grammar.

A solution for this could be, to define the grammar in a more flexible way. I tried several different approaches. For the grammar for *yes/no-questions*, I used the basic approach of just writing out the different possible ways of accepting or declining, since the user will most likely only use a few different. But still, the machine can't handle any deviation of the grammar. Furthermore, there are some utterances, which can be applied to only specific yes/no-questions like for example "Don't".

For recognizing *days*, I implemented a somewhat more flexible grammar. All possible keywords (in this case weekdays) are written out in the grammar. The machine will now check if the utterance includes the keywords instead of comparing them directly to the grammar. This gives the user more flexibility for saying things like "On Saturday", "Next Saturday" or "I think

Saturday would be nice", but it would also recognize "Not on Saturday". Assuming that the user won't use negations in the answer, this solution works ok.

I implemented the most flexible solution for recognizing the *time*. Here, I used a regular expression to catch many different possible ways of saying the time (e.g. "3 PM", "02:42", "8", "10 o'clock"). Using more complex regular expressions, can give the user even more flexible ways of saying something. But still, regular expressions can't represent all natural language and will fail at a certain point. This applies especially, when the user is not using grammatically correct sentences.

**unknown possible intents.** Without looking at the code, the user cannot know, which options he has in the *main menu*. For this, I tried to implement a help option. After invoking this option by saying e.g. "How can you help me?", the machine lists all possible options. Their descriptions are defined also in the *menuGrammar* and can be easily switched out by the developer. Still, the user does not know excactly, which keywords he needs to use, to invoke a specific option, but he already gets a general overview of what is possible. Furthermore, this only works well if there are only a few options. If there are more, another kind of help action would need to be implemented.

**repetitive machine answers.** When conversing with a machine, getting always the same sentence as an answer can be tiring. It would be more natural, if the machine would also variate with it's utterances. I tried to implement this in the clarification requests, where I defined multiple different utterances the machine could use. When an utterance by the user wasn't understood, the machine will select one of these predefined utterances randomly. In this case of general clarification requests, this was simple to implement. It would be a lot more complex, if the machine should react to specific utterances or words the user said or if the machine should inlcude variable information in its utterance.

**self correction.** Finally, a problem of this kind of dialogue is that the user has no possibility of self corrections. When he tries to self correct, the machine won't understand anything or even worse in my implementation of parsing the day, it would understand the first uttered day instead of the corrected day. Correcting this would require a deeper understanding of the parsed sentence. In this *FSM*, this problem is not too big, since the machine will utter a clarification request in most of the states if it couldn't understand the sentence (except for the day). This *just* feels unnatural for the user.