

TEAM MEMBERS
Raghavan Srinivasan
Sabrina Burleigh
Ankit Srivastava
Yohei Sakata

LING 571 HW7 WRITEUP
Final Project – Language Generation

****NOTE****

This code was designed to run with Python 2.5 and has been tested with ALL possible combinations of data from the KB for all goals.

To run this program type :

> run.sh <path to goals file> <path to kb>

SECTION 1: Introduction and general architecture

We were given a Genealogy Knowledge Base (KB) over a domain of family relations and a series of goals to access information from the KB. Our task was to display this information in response to the queried goal using natural language sentences. We successfully generate well-formed English sentences in a 4-step Natural Language Generation (NLG) system. These 4 stages are Text Planning, Micro Planning, Surface Realization and I/O Generation . Note that in addition to being open-ended, this task was an exercise in decision planning. We will justify our implementation choices below.

SECTION 2: Design Decisions and Deviations in Each Module

Knowledge Base (KB)

The role of the KnowledgeBase is to represent information about a person/family and to maintain the object ontology hierarchy leading from an abstract Entity to a specific Person. This is useful in pronominal resolution, e.g. converting 'Bill' to 'him', or 'Jane' to 'she'.

Text Planner

The function of the Text Planner is to determine the content and structure of the target sentences/goals. This is the first step in a NLG system that lays the groundwork for the translation of KB facts to English sentences. The central code is in the file labeled *text_planner.py*. We proceeded with a result-oriented approach. We were given five possible goals as input: describePerson, describeFamily, comparePerson, compareFamily, and describeAll.

Our first step was to manually create target sentences respective to each goal. Although our list of sentences is by no means exhaustive, it is representative of the KB facts relevant to a particular goal. Next, we simplified each sentence to minimize natural language complexities and ambiguities. This generally resulted in the decomposition of a large sentence into smaller units. For example, instead of generating a sentence like “Fred is married to a Lisa Jones of Tacoma,” we simplify it into two sentences, “Fred married Lisa Jones. She is from Tacoma.” Thus having a list of target sentences we moved on to defining basic message types and determining what chunks to introduce in the Text Planning Phase. Note we also edited our target sentences so that all sentences under a particular message type fit the same structure.

As per project specifications, we do not deal with grammar and syntax in this initial step. Thus two sentences classified under the same message type (ex. A Birth Message – sentences dealing with year of births) could have a different verb (ex. “is born”, “are born”, “were born”). Such technicalities were dealt with in the Micro planning stage. Our NLG system covers 7 types of messages – Age message, Birth message, Death message, Child message, Parents message, Spouse message, and Place message. We have also included a set of four relations which are used to connect two sentences to generate a compound; Coordination (and), Juxtaposition (while), Contrast (or), and Sequence. While the first three are coherence relations, the last one is simply a macro-grouping that is used to group all sentences of a particular goal into one paragraph.

. In compareFamily and comparePerson relations, we do not output an exhaustive list of comparison sentences. We limit this to a select few like comparing location, children, etc, and try to focus on what two people might have in common. We’ve also made our text planner “intelligent” in that it lays the structure of certain sentences only in the presence of certain facts, such as if two people in two different families were born in the same year. Thus our generator is in no way merely robotic that prints the same structure of sentences regardless of goal. We have also tried to include variety, in that the Spouse Message for describing persons will map to a sentence wherein the head verb is “is married to.” On the other hand, our Spouse Message for families will map to a sentence wherein the

head verb is “married.” Also note, our TextPlan (ie the output of the TextPlanner) is labeled DocPlan, which determines only part of our final sentences. The verbs, grammar and inflections are filled in the subsequent phases.

Micro Planner

The Micro Planner takes a text plan as input and accomplishes three main goals: 1) lexicalization of the text plan, 2) aggregation of messages which have relations, and 3) referring expression generation. The lexicalization method of each message object creates a lexical template with the following features: [SUBJECT], [OBJECT], [HEAD [VERB, NUMBER, TENSE]], [MODIFIER [HEAD, OBJECT]]. The feature [HEAD] is specific to each message and each goal. The other features are extracted from the message objects created in the Text Planner.

There is no aggregation of messages occurring in the Text Planner—it all happens in this phase. Aggregation is accomplished by lexicalizing each message in each pair that is joined by a relation, deciding which is the nucleus and which is the satellite, and combining them into one abstract syntax object. This object essentially now contains the features for two sentences. The cue word for each relation is extracted from the respective relation object, and added to the lexical template as a feature [CUE].

Referring expression generation looks for consecutive and repetitive instances of the same pronoun in the subject position. To avoid the confusion of using pronouns when there are two subjects, we chose to perform referring expression generation only on the methods describePerson() and describeAll(). The microplanner outputs a list of abstract syntaxes in the form of a text specification.

Surface Realizer

The surface realizer is the last main step in the generation algorithm. Its input is a text specification, which is essentially a list of abstract syntaxes. It is the job of the realizer to take this input and output naturally-sounding strings. It is in this phase that the surface forms of verbs are finally realized. Until this point, the verbs have been represented in the abstract syntax as a head with three features: base form of the verb, tense, and number. These features are used to access the appropriate verb in the lexicon (which contains verb lexemes and their inflected forms). The ordering of constituents is derived from the grammar, which contains a full set of all possible features in the order appropriate for English. Constituents in the grammar which do not appear in a lexical template

will be ignored for that sentence. In this way, features from each template will be added to a final string in the correct order, whether they are all the features in the grammar or just a select few.

After the final string is produced, we do a few important post-processing techniques, namely assigning grammatical conventions to the string. Commas are inserted before relation words such as “and”, “but”, and “while”, and spaces are inserted between each word. A period is appended to the end of the string, and the first letter of the sentence is capitalized.

Generator

The function of the Generator is to conduct the whole process of the NLG system which is divided into three parts: textplanner, microplanner, and SurfaceRealizer. It imports four classes: Goal, TextPlanner, Microplanner, and SurfaceRealizer. First, generator.generate() opens two files: test_goals and test_kb. It then creates object of the class “Goal”. TextPlanner.plan() takes the object and returns another object "textplan". Microplanner.plan() then takes the object and returns another object "textspec". Third, SurfaceRealizer.realize() takes the object and produces string result (e.g. Bill Green is from Bellingham. He is single and 56. He was born in 1950.) The result is output to a results file "generator_results"

SECTION 3: Performance, Strengths and Weaknesses

Our code provides output for all possible combinations of all goals and all instances of Human in the knowledge base. It provides natural sounding transitions, and a variety of sentence and verb constructions depending on goal type. It includes aggregation based on the relations of coordination, contrast, and juxtaposition. It also performs referring expression generation for subsequent sentences after a subject is introduced. The code is flexible for *any* predications that are currently in the knowledge base. Where it would need improvement in algorithmic design is if a system was desired which could “learn” new information from the knowledge base. For example, if the predications (likes Tom pizza) and (likes Wendy pasta) were added to the knowledge base, our code would not be able to include this information in our generated sentences because we do not have a Likes message type. All of the message types that are appropriate to this generation system had to be created ahead of time. A machine learning algorithm would be required to acquire new predications each time they are added to the knowledge base. There is also room for improvement in adding more relations and aggregation to the dialogue, and resolving pronouns in sentences with multiple subjects,

SECTION 4: Printing

We have implemented the `__str__()` standard Python method for objects to provide the print functionality. The Python 'print' function understands and calls `str()` on its arguments. We have used this standard functionality as opposed to providing separate functions such as “`printTextPlan()`”

Running our *generator* script will provide evidence of this functionality. The console output will display the `TextPlan` and the `TextSpecification` for a given goal.