An Open Framework to Develop and Validate Techniques for Speech Analysis

Luca Buoncompagni[†] and Fulvio Mastrogiovanni

University of Genoa

Abstract This paper presents the Concept-Action Grammar Generator (CAGG) system, which is an *open source* framework for natural speech-based interaction to be used in the semantic analysis of sentences. The goal of our framework is two-fold: i) to deliver a formalism associating a context-aware semantics to the words in a sentence; ii) to provide a common framework to benchmark different implementations of evaluators with the aim of improving speech-based interaction. To this aim, CAGG includes a specifically designed formalism for grammars definition as well as parsing mechanisms to automatically create and evaluate their structure. CAGG implements a deterministic evaluator which provisional performance related to a number of natural queries is discussed in terms of correct semantic associations and computational time.

Keywords: Speech analysis, Natural speech-based interface, Human-robot interaction

1 Introduction

We make, as human beings, a large use of speech-based interaction in organising our daily activities, teaching to someone else how to do something, understanding other people's interests, or asking for help in difficult situations. Therefore, a *natural* and *context-aware* speech-based interaction if fundamental for human-robot interaction and collaboration as well.

There is no shortage of advanced speech-to-text systems, both as open source, e.g., CMU Sphinx [4] and HTK [10], and well-known commercial products such as Google Now [9], Microsoft Cortana [6], Siri [2] and Alexa [1]. On the one hand, these systems rely on formal grammar specifications, advanced data processing techniques, web searching and machine learning to provide services to their users. On the other hand, there is no widespread consensus about how to deal with the typical ambiguities we employ during communication and clarifying conversations, which are due to the common sense knowledge we imbue our speech-based interactions with. When interacting with robots, it is often the case that *contextualised* knowledge is available as *robot's belief* and it is reasonable for a robot to exploit such information to better understand a human sentence.

[†] Corresponding author's email: luca.buoncompagni@edu.unige.it

```
   ! start <M>;
 <M> : <R1> {@?;@∼;}
     | <R2> {@T;@bb;};
<R1> : (hi | hello);
<R2> : !optional(good{@!T;}) bye
     | !repeat(bye{@!!;},1,2);
```

Figure 1. A simple example grammar in the CAGG syntax.

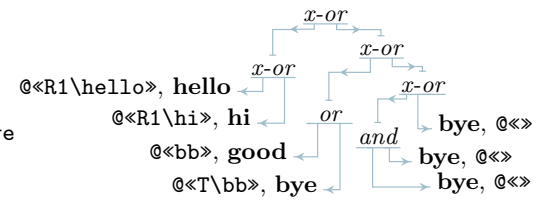

Figure 2. The Semantic expression Tree (ST) generated from the grammar shown in Figure 1.

An open source framework to generate grammars and to easily prototype and benchmark sentence evaluators is of the utmost importance. In this paper, we propose the Concept-Action Grammar Generator (CAGG) framework, which allows for: i) the context-aware semantic tagging of words in a sentence based on a specified grammar, and ii) developing and validating different evaluators. Semantic tagging is the approach we adopt to combine different knowledge sources, e.g., the knowledge a robot may have about its environment and the context. CAGG comes with a deterministic multi-thread implementation of a grammar evaluator for a basic speech-based interaction, and it is available open source[1].

2 System's Architecture and Information Flow

The CAGG framework adopts an *ad hoc* language based on the Backus-Naur Form syntax [3]. The language defines a grammar with *rules* and *sub*-rules (identified by `<` and `>`), which represent a logic expression as a binary tree. Rules have a head and a tail, separated by `:`. Within rules, a number of context-aware *directives* can be introduced, denoted by `!`. In particular, `!start` specifies the root of an expression, `!optional` identifies its optional parts, and `!repeat` enables looping over an expression within a lower and an upper bound on the range. Each line always ends with `;`. Sub-rules or *terms* (i.e., words in a sentence, which are an expression's leaves) can be aggregated using: an empty space, ' ' (to denote an *and* logical operator), or `|` (which is a *x-or* operator). It is noteworthy that CAGG also supports C-like comments and annotations. Each rule or term can be semantically tagged using `{@...;...}`, which specifies an array of symbols *activated* in a verified parsed path. Special tags can be used using `?`, which is replaced during compilation with the rule's head or term. To indicate that the last tag must be *augmented* with the path's leaf, one can use ∼. Furthermore, a tag can be removed with `!`, while `!!` clears the tag list.

Figure 1 shows a simple example of the CAGG syntax, which defines a grammar having at least a solution containing the tags: `R1` if the input is "hi" or "hello", as well as «`T\bb`» if "bye" is given or «`bb,T\bb`» for "good bye", whereas "bye bye" returns «». Accordingly, a tree representation is sketched in Figure 2.

Figure 3 shows the CAGG architecture. In order to build a grammar evaluator, CAGG implements a parsing procedure (based on ANTLR [7]) able to

[1] See: https://github.com/EmaroLab/concept_action_grammar_generator.

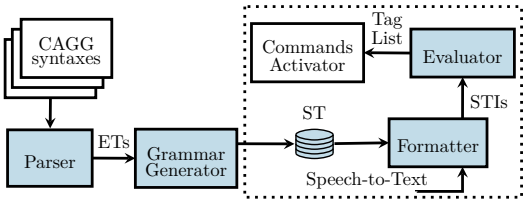

Figure 3. The CAGG flow to create grammars for on-line speech recognition (highlighted in a dashed box): *blue* boxes identify available modules, *white* boxes are application-specific modules.

generate an Abstract Syntax Tree (AST) for each rule. Each AST is transformed in a binary Expression Tree (ET) via syntax manipulation and, finally, the *Grammar Generator* module builds a unique Semantic Tree (ST) (e.g., Figure 2) by associating semantic tags to leaves (i.e., words) and all composition of rules to tree paths. The process can be supervised through a Graphical User Interface and stored in memory.

When audio information is given, a speech-to-text interface (in the current system we use the Google Speech API [8]) is used to determine the *most likely* corresponding text string. The string must be formatted. A recursive function generates different Semantic Tree Instances (STIs). In each instance, leaf nodes are initialised with a Boolean value by matching the word with the corresponding leaf in the ST. Each STI represents a possible solution that must be feasible in terms of words consequentiality and robustness with respect to non modelled words. Our current implementation searches for all the possible solutions recursively reducing the sentence by removing the first words until the sentence itself is empty. In this procedure, the *Formatter* module searches for the occurrences of each word in the leaves and manages possible multiple usages of the same term (e.g., "bye bye" in Figure 1 and 2) using a depth first approach. It is noteworthy that the implementation of such an algorithm strongly affects the overall system's performance thus, current work is focused on exploring other approaches. Again on Figure 3, as soon as a new STI is generated it is provided to the *Evaluator* module, which computes the logical expression from the leaf to the root, by adopting an *any-time* approach. If the root is in true state, the solution is satisfied and all the semantics tags related to the true leaves are returned as an array. Finally, the Commands Activator module is in charge to map such an array of semantic symbols into application-based services.

CAGG proposes a framework for the analysis and evaluation of different approaches to grammar definition and evaluation, and provides a method to event-based action generation which employs the analysis of sentences, by means of the tags array. It allows designers and engineers to implement new *Formatter* or *Evaluator* modules based on the ST-based representation.

3 Examples

We discuss here a simplified grammar to analyse questions posed in English (Figure 4). Table 1 contains a few examples, the corresponding computation time and the resulting sentence analysis. We see how the same word can be

```
      ! start <MAIN>;
  <MAIN> : !optional(<WH>) <QUERY>  !optional(<OBJ>{@Obj~;});
    <WH> : what | where;
 <QUERY> : <AUX>{@?~;} <OBJ>{@Subj~;} <VERB>{@?~;}
         | <VERB>{@?~;} <OBJ>{@Subj~;};
    <BE> : am | are | is;
  <HAVE> : have | has;
<BE-HAVE> : <HAVE> | <BE>;
   <AUX> : do | does | <HAVE>;
  <VERB> : <BE-HAVE> | seen;
   <OBJ> : <PRONOUN> | <THING>{@?;} | <CONCEPT>{@?;};
 <THING> : robot | door;
<CONCEPT> : name | surname;
<PRONOUN> : I | you;
```

Figure 4. A grammar for simple questions with a limited set of worlds.

Do you have a name?	0.463 *sec*	«AUX\do;Subj\you;VERB\Have;Obj\CONCEPT\name»
What have you seen?	0.238 *sec*	«AUX\have;Subj\you;VERB\seen»
Are you a robot?	1.161 *sec*	«VERB\are;Subj\you;Obj\THING\robot»
Are you a machine?	0.402 *sec*	«VERB\are;Subj\you»
You are smart.	1.520 *sec*	«»

Table 1. Examples of system usage using the grammar in Figure 4.

interpreted according to different semantics on the basis of the question and where it is in the parsing tree. Moreover, CAGG can deal with unknown words, which can be identified for further assessments. It is possible to characterise the worst case computation time, corresponding to a full parsing. This is very important for real-world human-machine interaction scenarios.

It is noteworthy that the *Evaluator* module may find also other solutions based on the grammar's definition. For example, in the grammar shown in Figure 1, the input "bye" could generate solutions without any semantic tags (in place of «bb»). This can happen when paths to such a term are verified. As an example it would not happen if the directive `!repeat(bye{@!!;},2,2)` were used, e.g., Figure 2. An idea to overcome this is to enable the *Command Activator* module to improve sentences analysis over time, where the improvement is related to the number of identified tags.

4 Conclusions

We propose the Concept-Action Grammar Generator (CAGG) system, an open source framework for natural speech-based interaction providing an efficient semantic sentences analysis. CAGG provides an open infrastructure to easily prototype and implement grammar evaluators for benchmarking purposes. Current work is focused on the integration with a system for spatial reasoning in human-robot interaction scenarios [5].

References

1. Amazon: Alexa voice service (AVS) (2016), https://developer.amazon.com/alexa-voice-service
2. Apple: iOS-siri (2016), http://www.apple.com/ios/siri/
3. Farrel, A.: Routing backus-naur form (RBNF): A syntax used to form encoding rules in various routing protocol specifications (2009)
4. Lamere, P., Kwok, P., Gouvea, E., Raj, B., Singh, R., Walker, W., Warmuth, M., Wolf, P.: The cmu sphinx-4 speech recognition system. In: IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong. vol. 1, pp. 2–5. Citeseer (2003)
5. Luca Buoncompagni, F.M.: A software architecture for object perception and semantic representation. 2nd Workshop on Artificial Intelligence and Robotics (AIRO) (2015)
6. Microsoft: What is cortana? (2016), https://support.microsoft.com/en-us/help/17214/windows-10-what-is
7. Parr, T.: The definitive ANTLR 4 reference. Pragmatic Bookshelf (2013)
8. Shires, G., Wennborg, H.: Web speech api specification. Final Report, W3C (2012)
9. Stanford, V.M., Williamson, O.J., Sherwin Jr, E.B., Castellucci, F.V.: Continuous speech recognition and voice response system and method to enable conversational dialogues with microprocessors (Mar 25 1997), uS Patent 5,615,296
10. Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., et al.: The htk book. Cambridge university engineering department 3, 175 (2002)