# Chatbot-based assertion generation from natural language specifications

Oliver Keszocze
oliver.keszoecze@fau.de
*Hardware-Software-Co-Design*
*Friedrich-Alexander-Universität Erlangen-Nürnberg*

Ian G. Harris
harris@ics.uci.edu
*Department of Computer Science*
*University of California Irvine*

***Abstract*—We present an approach to simplify the task of extracting assertions from specifications given in natural language. Our goal is to accept and understand a broad range of linguistic variation, allowing the author of the natural language specifications to express herself freely. To enable this, we leverage the Dialogflow framework from Google. Dialogflow is usually used to build chatbots that understand and respond to conversational statements. We have trained a Dialogflow model to recognize a range of different natural language expressions of properties, and to identify key information inside the expression. The model responses to each statement with a generated SystemVerilog assertion whose semantic meaning is equivalent to that of the English statement.**

***Index Terms*—Assertion-based Verification, Natural Language Understanding, Automatic Assertion Generation, Assertion Mining**

## I. Introduction

Using assertions is a well established hardware verification technique (assertion-based verification, ABV, see, e.g., [1] and an important research topic. An assertion describes an invariant (or property) of the hardware that is evaluated during hardware simulation in order to perform automated test-based verification. The use of assertions normally requires the manual definition of executable assertions based on specifications written in natural language (usually English); a process which is time-consuming, difficult, and error-prone.

Natural language is semantically closer to the original intent of the assertion in the mind of the human designer, so natural language descriptions are easier and faster to formulate, and are less likely to contain errors than manually-generated formal assertion descriptions. In this work we generate assertions in the SystemVerilog hardware verification language (see, e.g., [2]). The choice is taken as SystemVerilog is a widely-used industry standard. Natural language can contain ambiguity in the interpretation of words or sentences which would complicate the generation process (i.e. should "or" be interpreted as "xor"?). However, we have found that the text in specifications is written in a consistent way and we have been able to generate correct assertions as a result.

Several previous research efforts have attempted to generate properties and assertions using different formal languages including CTL [3], ACTL [4], SystemVerilog [5, 6], and OCL [7]. Some of these techniques rely heavily on manual interaction to convert the original natural language into a form which is easier to process [6, 7, 4] while other approaches only allow a restricted version of the English language to be used [3, 8].

Previous work on natural language processing (NLP) for extracting formal descriptions/assertions from textual specifications (see, e.g., [9, 7]) note that the process seems to require some "conversation" between the presented techniques and the designer. The work presented in this paper builds upon this idea by exploiting recent advances in the development of frameworks the creation of chatbots. These bots are designed to interact with a human being, trying to understand the problem of the dialog partner, answering it and, hopefully, solving it. In the context of this work, the solution provided by the bot will be a SystemVerilog statement that was created by parsing and understanding a sentence taken from a specification.

## II. Background

### A. Natural Language Processing

Natural language processing (NLP), coming from the field of linguistics, traditionally tried to automate the process of analyzing natural language sentences and creating structured descriptions of these sentences. For this, usually some kind of grammar (e.g. constituency grammars [10] or dependency grammars [11]) to create a graph representing the structure of the sentence. For the sentence "The server delivers the website", the dependency graph would, for example, yield the link "delivers" $\overset{dobj}{\rightarrow}$ "website" indicating that the word "website" is the direct object for the word "delivers". Figure 1 shows (a) the phrase structure tree for the sentence and (b) the full dependency graph; for details, we refer to [12].

A human reader would immediately understand that the word "server" in the previous example does not refer to a person working in a restaurant. An automated processing of the sentence could make use of a dictionary to perform *word sense disambiguation* (e.g. using WordNet [13]).

Given the senses of the words and a structural understanding of the sentence, researchers tried to generate various formal

978-1-7281-4113-8/19/$31.00 © 2019 IEEE

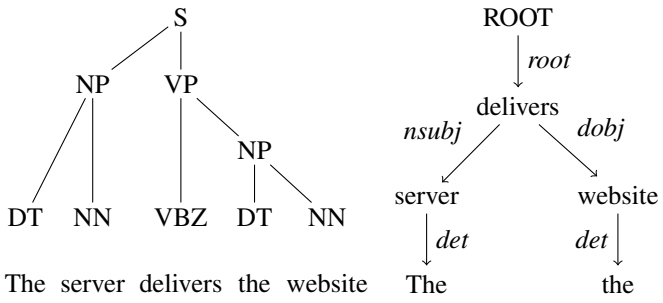


Fig. 1. (a) Phrase structure tree and (b) dependency graph of the sentence "The server delivers the website"

representations for various applications (see the related work discussion below). Recently, these techniques above have been supplemented by methods from statistics and artificial intelligence. The idea is to take a learning set of annotated (e.g. with the intent or sentiment) sentences and then train a classifier on them. Most prominent in this context is deep learning (see the recently published book [14]). Finding the intent of a sentences is the basic operation of the chatbot systems explained below. It should be noted that learning-based NLP can perform well with respect to finding the intend of a sentence but usually gives little insight into the linguistic structure of the sentence.

### B. Chatbots

Chatbots (more technically: dialog systems or dialog managers) are pieces of software that are usually used as an interface between a company and its customers. The tasks performed by these bots can range from simple technical support for products (e.g. the classical "Have you tried turning it off an don again?") to ordering products offered by the company.

In recent years, the market for chatbots has grown rapidly. Many companies offer frameworks with varying features and pricing models (e.g. Chatfuel, Google Dialogflow, Microsoft Bot Framework, Twilio, Rasa[1]). This trend is reflected in the amount of text books covering chatbot development (see, e.g., [15, 16, 17]).

The basic idea behind these systems is to accept a chat message by a user which is analyzed using NLP techniques discussed above. There are two important tasks which chatbots must perform and whose solutions we are leveraging to generate assertions from specifications. The first task is called *intent classification* [18] and it is the association of an utterance with one of a set of expected "intents" which capture the meaning being conveyed. Based on the intent classification, the chatbot generates a response which is appropriate. In the case of the Dialogflow [19] chatbot framework, the chatbot developer has to provide training sentences for each intent, that are used to classify the user input. The sentences "I want to order pizza." and "I'll take pizza", for example, could be used to train an intent for ordering pizza.

The second important task is referred to as *slot filling* [20]. Slot filling is the extraction of text spans from the utterance which have some meaning with respect to the intent. The Dialogflow framework refers to these text spans as *parameters* and each intent can be associated with a set of parameters which the chatbot can use to generate the response. In the previous example, a parameter could be "pizza", allowing a chatbot that is designed to deliver a huge variety of orders to answer with "We will deliver your $product right away" where "$product" is replaced by "pizza".

### C. Related Work

Automatically processing large text bases for knowledge extraction is an approach that is wide-spread. It ranges from collecting information from annotated sources in medicine where the authors (i.e. doctors and clinical personnel) had no restrictions when writing (see, e.g., [21]) to understanding legal documents that need to meet certain criteria with respect to the chosen words and sentence structure (see, e.g., [22]).

This work presents a method to extract assertions from natural language texts. The idea of automatically creating assertions is known as *assertion mining* and a well-established technique within the design automation community. Within this context, often the inputs to be analyzed are traces of the system under evaluation and the generated assertions are temporal logic (see,e.g. [23]).

Using UML/OCL [24] as an intermediate language before code generation is another commonly used approach. In [6], the authors create executable testbenches for embedded real-time systems. The approach starts by normalizing unstructured sentences into a *textual normal form* that is composed of unambiguous only. These sentences are then transformed to UML model from which, in turn, a SystemVerilog testbench is created. The authors of [7] follow a more interactive approach. They created an IDE which (semi-)automatically extracts a model and corresponding OCL constraints (which can be seen as assertions) of the system specified in the natural language sentences. For doing so, the tool makes use of the techniques introduced in Section II-A. The graphs are presented to the user who then can use it to annotate ambiguous parts of the sentence to match the intent.

Several recent approaches achieve good results using context-free grammars (CFGs) to generate a parse tree of each sentence and analyze its structure [25, 26]. CFG-based approaches have a long history in the field of NLP research and are often effective, but they have the disadvantage that they tend to require a large amount of manual effort to develop. For example, the approach presented in [25] presents a semantic grammar developed specifically for their dataset. The development of a semantic grammar is a completely manual task which would likely need to be repeated in order to apply the approach to a new text corpus. The approach in [26] uses a CFG for English which is generalized for all English sentences, but the approach relies on the identification of

[1]See https://chatfuel.com/, https://dialogflow.com/, https://dev.botframework.com/, https://www.twilio.com/, and https://www.rasa.com/ for the respective websites

subtree patterns in the parse tree. The identification of subtree patterns is a time-consuming and difficult manual process.

The advantage of the approach which is presented in this paper over previous CFG-based approaches is the reduces manual effort required to accommodate a new text corpus. Our approach relies on the machine learning approaches built into the Dialogflow tool, so modeling a new text corpus would only require providing new training examples from the corpus and performing training again. Dialogflow fully automates the training process and the number of examples required for training is very small (less than 30 in total) as described in our results. Our approach generates results which are of similar quality to CFG-based approaches [25, 26], but our approach is much easier to develop and extend to new types of text in the future.

## III. Methodology

We believe, that an interactive approach, such as in [7], is the right way of analyzing natural language. Our approach, though, does also automate the interaction between the NLP tool (i.e. the Dialogflow agent) and the designer who wants to create a SystemVerilog assertion by replacing the designer by a piece of software having a "dialog" with the agent until the assertion could be created.

Figure 2 shows the structure of the assertion generation system. The center of the system is the *Assertion Generator* which accepts an assertion sentence, written in English, and produces a SystemVerilog assertion whose function matches the meaning of the assertion sentence. The Assertion Generator communicates with the remote Dialogflow servers using the Dialogflow API. We have developed a chatbot "agent" which executes on the Dialogflow servers and is used to perform intent classification and slot filling. The Assertion Generator sends an English statement to the Dialogflow agent, and the agent returns the intent of the statement together with the intent parameters found in the statement. Given the intent and the parameters, the Assertion Generator constructs the SystemVerilog assertion.

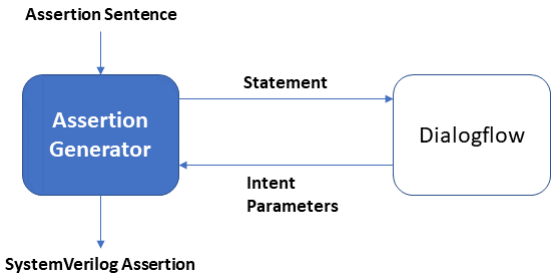


Fig. 2. Assertion Generation System

The assertion generation process is clarified with the following example, *Sentence 1: "SIGNAL1 must be HIGH"*. This sentence is transmitted to the Dialogflow agent which returns the intent and parameters as shown in Table I. The intent *equality* indicates that the SystemVerilog operator, "==", will be used to relate two values. The parameters *lhs* and *rhs* are the arguments of the "==" operator on the left-hand-side and right-hand-side, respectively. The resulting assertion is *"assert property (SIGNAL1 == 1);"* which combines the parameters and the equality operator.

TABLE I
Results from Dialogflow for Sentence 1

| | |
|---|---|
| **Intent:** | equality |
| **Parameters:** | lhs = "SIGNAL1", rhs = "1" |

In order to process more complicated assertions, the generation sends parameters to the Dialogflow agent recursively until all parameters are either signals or constants which are directly included in the assertion. An example is the processing of the following, *Sentence 2: "SIGNAL1 is HIGH when SIGNAL2 is LOW"*. The Dialogflow agent returns the intent and parameters as shown in Table II. The intent "implication" indicates that the SystemVerilog operator, "|->", will be used to relate the antecedent and the consequent. The antecedent "SIGNAL2 is LOW" and the consequent "SIGNAL1 is HIGH" are not purely signals or constants, so they are each recursively sent to the Dialogflow agent for further processing. The final assertion, *"assert property ((SIGNAL2 == 0) |-> (SIGNAL1 == 1));"*, is produced by combining the results of the antecedent and consequent with the "|->" operator.

TABLE II
Results from Dialogflow for Sentence 2

| | |
|---|---|
| **Intent:** | implication |
| **Parameters:** | antecedent = "SIGNAL2 is LOW",<br>consequent = "SIGNAL1 is HIGH" |

### A. Dialogflow Agent

We have opted to chose the Dialogflow framework [19] in our work. The reason for this choice is that Dialogflow offers an easy to use web-interface as well as support for multiple programming languages, a large community, and an extensive documentation. Unfortunately, it is very difficult to base the choice of the chatbot on the natural language parsing accuracy/quality as it is not published on the websites and furthermore difficult to measure in the first place (see, e.g., [27] for an introduction), especially as an end-user. Note that this problem applies to all presented chatbot frameworks.

The Dialogflow agent is used to perform intent classification and slot filling for each sentence and phrase. Developing the agent requires two steps which we describe here, (1) defining the intents and their parameters, and (2) training the agent to recognize each intent and parameter.

### B. Intents and Parameters

We developed an intent for each SystemVerilog operation which our tool can generate in final assertion. For this reason, each intent corresponds to an *operation string* which is the

string representing the corresponding SystemVerilog operation. Each intent has a set of parameters which correspond to the arguments of the SystemVerilog operation. Each parameter value is a span of text in the original sentence which may be a simple signal/constant, or may be a more complex phrase. If the parameter value is a signal name or a constant then it is used directly in the final assertion. If the parameter value is more complex then it is recursively sent to the Dialogflow agent in order to resolve it further.

The following is the list of intents which we have defined. Note that this set can be expanded to include a wider range of SystemVerilog operations in the future.

- **Implication** – This intent corresponds to the use of the implication operation string "|->" and is typically recognized by the use of words such as "if", "then", and "when". This intent has two parameters, *antecedent* and *consequent*.
- **Conjunction** – This intent corresponds to the use of the implication operation string "&&" and is recognized by the use of words such as "and" and "also". This intent has two parameters, the argument on the left-hand-side, *lhs*, and the right-hand-side, *rhs*.
- **Equality** – This intent corresponds to the use of the equality operation string "==" and is recognized by the use of words such as "is", "is the same as", and "must be". This intent has two parameters, the argument on the left-hand-side, *lhs*, and the right-hand-side, *rhs*.
- **Inequality** – This intent corresponds to the use of the inequality operation string "!=" and is recognized by the use of words such as "is not" and "must not be". This intent has two parameters, the argument on the left-hand-side, *lhs*, and the right-hand-side, *rhs*.
- **Stability** – This intent corresponds to the use of the SystemVerilog stability function "$stable()" and is recognized by the use of phrases such as "must be stable". This intent has one parameter the signal which is required to be stable, *sig*.
- **Assertion** – This intent corresponds to the use of the equality operation, together with the value 1, "== 1" and is recognized by the use of phrases such as "must be asserted" or "is asserted". This intent has one parameter the signal which is required to be asserted, *sig*.
- **Until** – This intent corresponds to the use of the "until" operation and is recognized by the use of the word "until". This intent has two parameters which correspond to two peroperties which are related in time. We assume that the two related properties are non-overlapping in time.

### C. Training

The Dialogflow agent must be trained to recognize sentences which match each intent, and to identify the values of the parameters of each intent inside a sentence. When developing the agent, we provided a set of training sentences for each intent. Each training sentence for an intent must be manually labeled in order to identify the parameter values in each sentence. After the training sentences have been provided, the Dialogflow system uses machine learning approaches to generalize the training sentences and recognize the intent of sentences which are not contained in the training set.

An important feature of the Dialogflow training approach is that it requires very few training examples in order to provide good results. For our agent, the number of training examples which we provided varied between 2 and 6 for each intent. This is very low compared to traditional supervised learning approaches. Although the precise nature of the machine learning approach used by Dialogflow is proprietary, it is reasonable to assume that the agents are pre-trained using a large, domain non-specific corpus. The pre-training would enable the agent to understand the structure of English sentences in general and only a few additional training sentences are needed to characterize the specific domain.

The *Implication* intent, for example, was trained with the sentences shown in Listing 1. The antecedent and consequent of the implications are annotated as indicated by the blue italic and green bold face fonts, respectively. These annotations are used by the Dialogflow agent in the slot filling step. As can bee seen, we used linguistic variants of the implication, i.e. sentences where the antecedent is placed before the consequent and vice versa. This, as well as sentence 4), where an antecedent consisting of more words than in the other training sentences, is done to allow the agent to generalize the sentences.

1) when *z is low*, **q is stable**
2) **q is stable** if *z is low*
3) **z is low** when *x is high*
4) if *x is high and y is high* then **z is low**
5) if *x is high* then **y is low**
6) if *z is stable* then **a is low**

Listing 1. Training sentences for the implication intent

## IV. Algorithm

Algorithm 1 presents the pseudo-code of our approach to assertion generation, which we refer to as AG for succinctness. The function takes a single argument $text$ which is a natural language statement. The algorithm is recursive, so line 2 checks the end condition of the recursion. If the text is a terminal (signal name or constant) then the text is returned directly for inclusion in the final assertion. Line 5 invokes $DialogflowRequest()$ which submits text to the Dialogflow agent and returns the response from the agent, including the results of intent classification and slot filling. The remaining conditionals, on lines 7–19, check the intent of the text and return the appropriate SystemVerilog assertion strings. For example, the condition on line 7 checks whether or not the text is an implication. If it is, then the AG() function is called recursively to resolve the antecedent and consequent, and the final string is generated by combining the antecedent and consequent with the '|->" operator. Similarly, the condition on line 6 checks for the conjunction intent and generates the final SystemVerilog assertion using the "&&" operator.

**Algorithm 1:** Assertion generation from sentences

**Data:** user input *text*
**Result:** SystemVerilog assertion

```
AG text
    if isTerminal(text) then
        return text
    end
    res ← DialogflowRequest(text);
    intent ← res.intent;
    if intent ="implication" then
        return AG(res.antecedent) + " |-> " +
                        AG(res.consequent)
    else if intent ="conjunction" then
        return AG(res.lhs) + "&&" + AG(res.rhs)
    else if intent ="equality" then
        return AG(res.lhs) + "==" + AG(res.rhs)
    else if intent ="inequality" then
        return AG(res.lhs) + "!=" + AG(res.rhs)
    else if intent ="stability" then
        return "$stable(" + AG(res.sig) + ")"
    else if intent ="assertion" then
        return AG(res.sig) + " == 1"
    else if intent ="until" then
        return AG(res.lhs) + "until" + AG(res.rhs)
    else
        return error
    end
end
```

It should be noted that our approach differs from "traditional" approaches rooted in linguistics in the sense that we make no use of linguistic concepts such as phrase structure trees of dependency graphs (see Section II-A) directly. The act of NLP is performed by the Dialogflow agent. The *Assertion Generator* mainly deals with the correct decomposition of logic structures.

## V. Experimental Results

To evaluate our approach, we applied it to natural language assertion descriptions presented in the AMBA 3 AXI Protocol Checker User Guide [28] which describes assertions for the AMBA 3 AXI bus protocol [29]. We considered the 77 natural language descriptions which refer only to signals which are constrained, and constant values used to define the constraints. Our tool successfully generated SystemVerilog assertions for 62 of the 77 sentences, a total of 81% (see Listing 2 for two examples for which assertions could be generated). The correctness of the generated assertions was checked manually.

Our approach failed on 15 of the 77 natural language descriptions because each required a SystemVerilog operation which was not supported by our current set of intents. The cases where assertion generation failed were obvious because the sentence matched an incorrect intent and the entities of the intent could not be found. For this reason, whenever assertion generation failed, the resulting assertion was not

**sentence 1**

AWID must remain stable when AWVALID is asserted and AWREADY is LOW

**assertion 1**

```
assert property (((AWVALID == 1) &&
(AWREADY == 0)) |-> ($stable(AWID)));
```

**sentence 2**

A value of X on AWSIZE is not permitted when AWVALID is HIGH

**assertion 2**

```
assert property (((AWVALID == 1)) |->
(AWSIZE != X));
```

Listing 2. Automatically generated assertions and the sentences from which they were extracted

syntactically correct because operators were missing their required operands. 13 of the 15 missed assertions were the timing-related expression "first cycle after". Examples of these descriptions are "AWVALID is LOW for the first cycle after ARESETn goes HIGH". As this timing constraint is supported by SystemVerilog, it would be possible to extend our tool to support it.

There are further cases where our approach incapable of dealing in principle. One of these cases is when the pronoun "it" is used to refer to some entity in a previous sentence. Finding this entity is known as *pronominal reference resolution* and would need to be dealt with explicitly. There are approaches to this issue (see, e.g., [30]), but they do not fit into the scope of our approach.

Other problems arise from sentences such as "A value of X on AWVALID is not permitted when not in reset". Here "not in reset" describes an abstract concept that needs to be understood before being able to create the correct assertion from this sentence. In general, sentences that are not self-contained but require further domain knowledge pose problem to our approach.

## VI. Conclusion & Outlook

We have presented an approach to automatically generate SystemVerilog assertions from natural language descriptions. Our approach leverages the online chatbot framework Dialogflow to perform the intent classification and slot filling tasks required. Using Dialogflow allowed us to benefit from a strong machine learning approach without dealing with the complexity of fully defining and training a model. Because the chatbot framework is pre-trained with a large corpus of natural language text, few additional training sentences are needed to adapt to our domain. Our approach shows good results and the same framework could be extended to accept many new natural language descriptions which constrain timing and signal transitions.

While developing our assertion generation tool, we noted that using an online tool for the automated processing of large specifications can lead to timeouts due to excessive requests to the website. An obvious next step is to test tools, such as, for example, Rasa [31], that allow for a local installation and to compare its capabilities with our Dialogflow implementation.

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1813858.

## REFERENCES

[1] Harry Foster. "Applied Assertion-Based Verification: An Industry Perspective". In: *Foundations and Trends® in Electronic Design Automation* 3.1 (Apr. 15, 2009), pp. 1–95.

[2] Christ Spear and Greg Tumbush. *SystemVerilog for Verification*. 2012.

[3] Alexander Holt and Ewan Klein. "A Semantically-Derived Subset of English for Hardware Verification". In: *Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*. 1999.

[4] Alessandro Fantechi et al. "Assisting Requirement Formalization by Means of Natural Language Translation". In: *Formal Methods in System Design* 4.3 (1994), pp. 243–263.

[5] Ian Harris. "Capturing Assertions from Natural Language Descriptions". In: *Workshop on Natural Language Analysis in Software Engineering*. San Francisco, CA, USA: Institute of Electrical and Electronics Engineers, 2013.

[6] Wolfgang Mueller et al. "Generation of Executable Testbenches from Natural Language Requirement Specifications for Embedded Real-Time Systems". In: *Distributed, Parallel and Biologically Inspired Systems*. 2010, pp. 78–89.

[7] Oliver Keszocze et al. "Lips: An IDE for Model Driven Engineering Based on Natural Language Processing". In: *Workshop on Natural Language Analysis in Software Engineering*. San Francisco, CA, USA, 2013.

[8] Norbert Fuchs and Rolf Schwitter. "Attempto Controlled English (ACE)". In: *International Workshop on Controlled Language Applications*. 1996.

[9] Rolf Drechsler, Mathias Soeken, and Robert Wille. "Towards Dialog Systems for Assisted Natural Language Processing in the Design of Embedded Systems". In: *Forum on Specification & Design Languages*. 2012.

[10] Noam Chomsky. "Three Models for the Description of Language". In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124.

[11] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher Manning. "Generating Typed Dependency Parses from Phrase Structure Parses". In: *Conference on Language Resources and Evaluation*. 2006, pp. 449–454.

[12] Dan Jurafsky and James Martin. *Speech and Language Processing*. Prentice Hall, 2008. 1024 pp. URL: http://www.cs.colorado.edu/~martin/slp.html.

[13] George Miller. "WordNet: A Lexical Database for English". In: *Communications of the ACM* 38.11 (Nov. 1995), pp. 39–41.

[14] Uday Kamath, John Liu, and Jimmy Whitaker. *Deep Learning for NLP and Speech Recognition*. 2019. 676 pp.

[15] Amir Shevat. *Designing Bots: Creating Conversational Experiences*. O'Reilly, 2017. 120 pp.

[16] Rashid Khan and Anik Das. *Build Better Chatbots*. 2017. 120 pp.

[17] Srini Janarthanam. *Hands-On Chatbots and Conversational UI Development*. Packt Publishing, 2017. 394 pp.

[18] Suman Ravuri and Andreas Stoicke. "A Comparative Study of Neural Network Models for Lexical Intent Classification". In: *Workshop on Automatic Speech Recognition and Understanding*. Scottsdale, AZ, USA: Institute of Electrical and Electronics Engineers, 2015.

[19] *Google Dialogflow*. URL: https://dialogflow.com/.

[20] Gokhan Tur, Asli Celikyilmaz, and Dilek Hakkani-Tür. "Latent Semantic Modeling for Slot Filling in Conversational Understanding". In: *International Conference on Acoustics, Speech and Signal Processing*. Institute of Electrical and Electronics Engineers, 2013.

[21] Hoang Nguyen and Jon Patrick. "Text Mining in Clinical Domain: Dealing with Noise". In: *International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA, USA, 2016, pp. 549–558.

[22] Oliver Keszocze et al. "(Semi-)Automatic Translation of Legal Regulations to Formal Representations: Expanding the Horizon of EDA Applications". In: *Forum on Specification & Design Languages*. Munich, Germany, Oct. 2014.

[23] Wenchao Li, Alessandro Forin, and Sanjit Seshia. "Scalable Specification Mining for Verification and Diagnosis". In: *Design Automation Conference*. Anaheim, CA, USA: Institute of Electrical and Electronics Engineers, 2010.

[24] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. 2nd ed. Addison-Wesley Professional, Aug. 27, 2003. 236 pp.

[25] Krishnamurthy, Rahul and Hsiao, Micahel S. "Controlled Natural Language Framework for Generating Assertions from Hardware Specifications". In: *International Conference on Semantic Computing*. Jan. 2019, pp. 367–370.

[26] Junchen Zhao and Ian G. Harris. "Automatic Assertion Generation from Natural Language Specifications Using Subtree Analysis". In: *Design, Automation and Test in Europe*. Mar. 2019, pp. 598–601.

[27] Tom Fawcett. "An Introduction to ROC Analysis". In: 27.8 (2006), pp. 861–874.

[28] ARM Ltd. *AMBA 3 AXI Protocol Checker User Guide*. 2009.

[29] ARM Ltd. *AMBA AXI Protocol Specification, v1.0*. 2003.

[30] Shalom Lappin and Herbert Leass. "An Algorithm for Pronominal Anaphora Resolution". In: *Computational Linguistics* 20.4 (1994).

[31] *Rasa*. URL: https://www.rasa.com/.