# A Framework to build Abductive-Deductive Chatbots, based on Natural Language Processing and First-Order Logic

Carmelo Fabio Longo[1], Corrado Santoro[1]

[1]Department of Mathematics and Computer Science, University of Catania, Viale Andrea Doria, 6 - 95125 - Catania, IT

Abstract

This paper presents a framework based on natural language processing and first-order logic, which implicitly simulate the human brain features of selecting properly information related to a query from a knowledge base (abductive pre-stage), before to infer new knowledge from such a selection acting as deductive database. Such features are used with the aim of instantiating *cognitive* chatbots, able of human-like fashioned reasoning, supported by a module which automatically transforms polar and wh-questions into one or more likely assertions, in order to infer Boolean values or snippets with variable length as factoid answer from a *conceptual* knowledge base. The latter is splitted into two layers, representing both long- and short-term memory, and the transition of information between the two layers is achieved leveraging both a greedy algorithm and the engine's features of a NoSQL database, with promising timing performance than respect using one layer. Furthermore, such chatbots don't need any scripts updates or code refactory when new knowledge has to income, but just the knowledge itself in natural language.

**Keywords**

Artificial Intelligence, Chatbot, First-Order Logic, Cognitive Architecture, Deductive Database

## 1. Introduction

Nowadays, the momentousness which tech giants like Google, Meta (former Facebook), Microsoft, IBM and Amazon are giving to chatbots, is a strong indicator that this technology will play a significant role in the future. Among applications leveraging Natural Language Processing (NLP), those related to chatbots systems are growing very fast and present a wide range of choices depending on the usage, each with different complexity levels, expressive powers and integration capabilities. At the present, if you want to know trending movies in your area, you could use the Fandango Bot[1]; or, if you want to get NBA highlights and updates, you could use NBA's bot[2], and so forth. On the other hand, the way towards a human-like fashioned reasoning chatbot is quite long and challenging. Such an ideal agent-chatbot, in addition to having deductive, abductive and inductive capabilities, should be able of commonsense categorization, even making usage of the so-called *Frames* [1] and *Scripts* [2], and so on. In

---

*CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy*

fabio.longo@unict.it (C. F. Longo); santoro@dmi.unict.it (C. Santoro)
0000-0002-0877-7063 (C. F. Longo); 0000-0001-7116-9338 (C. Santoro)

CEUR Workshop Proceedings (CEUR-WS.org)

[1] https://www.facebook.com/Fandango/

[2] https://www.facebook.com/nba/

order to achieve that, the cognitive disciplines involving has become a mandatory step in the overall process of such agent's modelling, together with knowledge of the linguistic science.

In this work, which is somehow the evolution of [3] where we showed the effectiveness of a similar approach in the case of automation commands provided in natural language, we present a framework called AD-Caspar based on NLP and First-Order Logic (FOL), as baseline platform for implementing scalable and flexible *cognitive* chatbots with both goal-oriented and conversational features. A first overview of such an architecture has been presented here [4]. The first prototype of AD-Caspar is not yet provided with tools to build complex dialog systems, but differently from other platforms, in order to handle additional question-answer couples, the user has to provide just the related sentences in natural language, without the need of updating the chatbot code at design-time. After the agent has parsed every sentence, a FOL representation ot them is asserted in its conceptual KB, which will be able to act as a deductive database [5]. AD-Caspar inherits most of its features directly from its predecessor, namely Caspar [6], whose name stands for: *Cognitive Architecture System Planned and Reactive*. Caspar was designed to build goal-oriented agents (vocal assistants) with enhanced deductive capabilites, working on Internet of Things (IoT) scenarios. The additional features introduced in AD-Caspar, where AD- stands for: *Abductive Deductive*, are the usage of abduction as pre-stage of deduction, in order to make inferences only on a narrow set of query-related clauses, plus the application of question-answering techniques to deal with wh-questions and give back factoid answers (single nouns or snippets) in the best cases; otherwise, optionally, only a relevance-based output will be returned.

This paper is structured as follows: Section 2 is about the related literature of the typical approaches of chatbot applications; Section 3 shows in detail all the architecture's components and underlying modules, referring sometimes the reader to legacy-related literature for the sake of shortness; Section 4 shows how AD-Caspar deals with polar and wh-questions; Section 5 is about a case-study where it is shown a typical instance of a Telegram chatbot, working on small KBs; Section 6 summarizes the content of the paper and provides our conclusions, together with some future work perspective.

A Python prototype implementation of AD-Caspar is also provided for research purposes in a Github repository[3].

## 2. Related Work

In the plethora of known chatbot systems, first of all stand out the ones participating to the Loebner Prize Competition[4]. Typical sentences to prove Loebner price candidates effectiveness do not require particular logical inference, but mainly the ability to possible gloss, convincingly, as a human being would. Some chatbots make usage also of language tricks, such as monologues, not repeating itself, identify gibberish, play knock-knock jokes, etc. But despite such features, these chatbots can hardly aspire to be somehow *useful* for the task of decision-making, but just to fool their interlocutor. This is because of the historical approach of such a technology, which always aimed, before all, at passing the well-known Turing Test [7] as intelligence evaluation

---

[3] http://www.github.com/fabiuslongo/ad-caspar

[4] https://www.ocf.berkeley.edu/~arihuang/academic/research/loebner.html

criterion, event though the reader can find copious literature on this theme about which it is insufficient in such a task.

The first distinction between the chatbot platforms divides them into two big macro-categories: *goal-oriented* and *conversational*. The former is the most frequent kind, often designed for business platforms support, assisting users on tasks like buying goods or execute commands in domotic environments. In this case, it is crucial to extract from an utterance the intentions together with all related parameters, then to execute the wanted operation, providing also a proper feedback to the user. As for conversational ones, they are mainly focused on having a conversation, giving the user the feeling to communicate with a sentient being, returning back reasonable answers optionally taking into account discussions topics and past interactions. One of the most common platforms for building conversational chatbot is AIML[5] (Artificial Intelligence Markup Language), based on words pattern-matching defined at design-time; in the last decade it has become a standard for its flexibility to create conversation. In [8], AIML and Chatscript[6] are compared and mentioned as the two widespread opensource frameworks for building chatbots. On the other hand, AIML chatbots are difficult to scale if patterns are manually built, they have great limitations on information extraction capabilities and they are not suitable for task oriented chatbots. Other kinds of chatbots are based on deep learning techniques [9], making usage of huge corpus of examples of conversations to train a generative model that, given an input, is able to generate answers with arguable levels of accuracy.

In general, all chatbots are not easily scalable without writing additional code or repeating the training of a model with fresh datasets. As for the latter, unfortunately neural networks (in particular, the ones used in deep learning applications) suffer from the problem known as *catastrophic interference*: a process where new knowledge overwrites, rather than integrates, previous knowledge. At this regard, the usage of neural models working at semantic tier as dependency parsers, in order to build logical models of utterances in natural language, prevents much more such a drawback.

## 3. The Architecture

As outlined in the introduction, the proposed architecture, namely AD-Caspar, derives directly from its predecessor Caspar, but, differently from the latter, has been endowed with important features which permit to handle better large KBs and achieve abduction at the same time. The KB is divided into two distinct groups operating separately (orange boxes in Figure 1), which we will distinguish as *Beliefs KB* and *Clauses KB*: the former is managed by the *Belief-Desire-Intention* framework *Phidias* [10], and contains beliefs which support all framework operations, even those related with interaction with environment; the latter contains conceptual information on which we want the agent to make logical inference. Moreover, the Clauses KB is splitted into two different layers: *Low Clauses KB* and *High Clauses KB*. The whole knowledge is stored in the low layer, but the logical inference is achieved in the high one, whose clauses will be the most relevant for the query in exam, taking into account a specific confidence threshold which will be discussed ahead. The two KBs can be seen, somehow, as the two types of human

---

[5] http://www.aiml.foundation/

[6] https://github.com/ChatScript/ChatScript

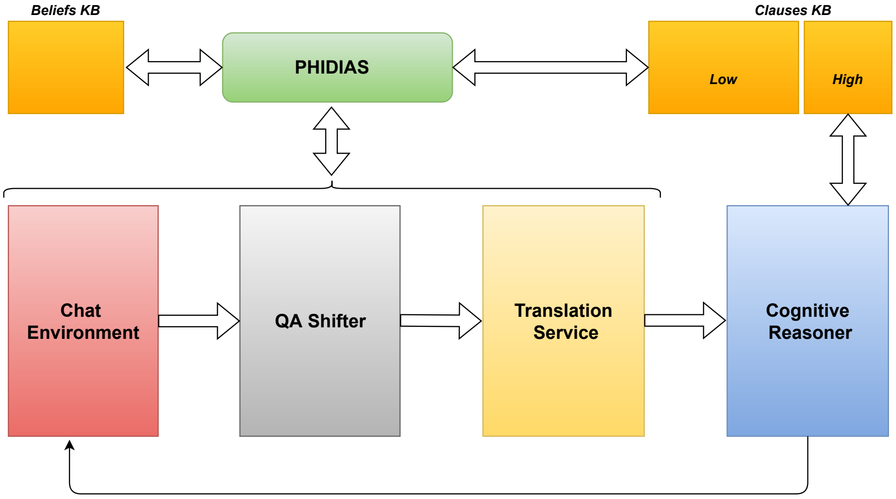


**Figure 1:** A sempliflied functional scheme of AD-Caspar.

being memory: the so called *procedural memory* or *implicit memory* [11], made of thoughts directly linked to concrete and physical entities; the *conceptual memory*, based on cognitive processes of comparative evaluation. Nevertheless, the two layers of the Clauses KB can be seen as *Short-Term Memory* (High Clauses KB) and *Long-Term Memory* (Low Clauses KB). As well as in human being, in this architecture, Beliefs KB and Clauses KB can interact with each other in a reactive decision-making process (meta-reasoning).

A simplified functional scheme of AD-Caspar is depicted in Figure 1, with each component highlighted with a distinct colour. Specifically, the *Chat Environment* (red box in Figure 1) is responsible of interfacing the agent with all required to establish a communication with a chatbot system. When a question is detected, the text is sent to another module called *QA Shifter* (gray box in Figure 1), whose details are reported in Section 4, with the task of transforming a question into one or more likely assertions as possible answers to such a question. The *Translation Service* (yellow box in Figure 1) is the component responsible of translating a natural language sentence, by leveraging a dependency parser [12] and a production rules system, into in a *neo-Davidsonian* FOL expression. The latter inherits the shape from the event-based formal representation of Davidson [13], with every term $\mathtt{t}$ as follows:

$$\mathtt{t := x \mid L{:}POS(t_1) \mid L{:}POS(t_1,\ t_2,\ t_3) \mid L{:}POS(t_1,\ t_2)},$$

where x is a variable bound either to a universal or to an existential quantifier, L a WordNet [14] Synset (selected with a disambiguation process taking in account of the context) or a lemma, POS a Part-of-Speech (POS) tag, and $t_1$, $t_2$, $t_3$ other terms (recursively defined). Implication symbols are also admitted, when a group of predicates subordinate the remaining ones. The outcome of the Translation Service, for a basic verbal phrase, will be as follows:

$$\mathtt{synset_1{:}POS_1(e_1,\ x_1,\ x_2) \land synset_2{:}POS_2(x_1) \land synset_3{:}POS_3(x_2)},$$

where $\mathtt{synset}_1$ and $\mathtt{POS}_1$ are related to a verb, while $\mathtt{synset}_2$, $\mathtt{POS}_2$, $\mathtt{synset}_3$, $\mathtt{POS}_3$ to nouns. Additional possible predicates corresponding to other grammatical elements (adjectives, adverbs and prepositions), will share the same variable of predicates which are referred to. Before being sent to the next module, the above formula will be furtherly translated as follows[7], by suppressing the first argument $e_1$ from the predicate with label $\mathtt{synset}_1\mathtt{:POS}_1$:

$$\mathtt{synset}_1\mathtt{:POS}_1\mathtt{(synset}_2\mathtt{:POS}_2\mathtt{(x}_1\mathtt{),\ synset}_3\mathtt{:POS}_3\mathtt{(x}_2\mathtt{)).}$$

The rationale behind such a notation choice is explained next: a definite clause is either atomic or an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. Because of such restrictions, in order to make clauses suitable for inference with the Backward-Chaining algorithm (which requires a KB made of definite clauses), we must be able to encapsulate all their information properly. The strategy followed is to create composite terms, taking into account of the POS and applying the following hierarchy to every noun expression as follows:

$$\mathtt{IN(JJ(NN(NNP(x))),\ t),} \tag{1}$$

where `IN` is a preposition label, `JJ` an adjective label, `NP` and `NNP` are noun and proper noun labels, x is a bound variable and t a predicate.
As for the verbal phrases, the nesting hierarchy will be the following:

$$\mathtt{ADV(IN(VB(t}_1\mathtt{,\ t}_2\mathtt{),\ t}_3\mathtt{)),}$$

where `ADV` is an adverb label, `IN` a preposition label, `VB` a verb label, and $t_1$, $t_2$, $t_3$ are predicates; the nesting hierarchy of `ADV` and `IN` can also be swapped; in the case of imperative or intransitive verb, instead of respectively $t_1$ or $t_2$, the arguments of `VB` will be left void. As we can see, a preposition (`IN`) might be related either to a noun or a verb. For a detailed description of the Translation Service, since such component is utterly inherited from CASPAR, the reader is referred to [6]. The *Cognitive Reasoner* (blue box in Figure 1) enables the architecture with reasoning and meta-reasoning capabilities. Differently from the correspondent component in CASPAR, each assertion is made on both High and Low Clauses KB. Since the High Clauses KB is a volatile memory, it will be emptied after the agent is restarted, but at the same time is the one where first the deduction is attempted; then, after a possible unsuccessful reasoning, the Low Clauses KB will be used to populate the High one with fresh nested definite clauses, having a specific number of features compliant with a wanted threshold described ahead in Section 4. After the High Clauses KB is being populated, the deductive reasoning is re-attempted.

## 4. Question Answering

This section shows how the proposed architecture is able to deal with question-answering. The Low Clauses KB has a very important role in such a task, because it supports abduction as pre-stage of deduction and it is the source from which the High Clauses KB is populated, in order to make logical inference. Each record in the Low Clauses KB is stored in a NoSQL

---

[7] After such a step the POS can be either removed from the formula as in Figure 6 and Figure 7, or not to keep a shallow label tipization as in Figure 4.

database and it is made of three fields related to a specific sentence in natural language. Each field is defined as follows:

- **Nested Definite Clause**: definite clause made possibly of composite predicates.
- **Features Vector**: vector of labels of the above definite clause.
- **Sentence**: the sentence in natural language.

For instance, let the sentence to be stored be:

*Barack Obama became president of United States in 2009.*

In this case, the record in the Low Clauses KB will be as follows:

- `In(Become(Barack_Obama(x1), Of(President(x2), United_States(x3))), 2009(x4))`
- `[In, Become, Barack_Obama, Of, President, United_States, 2009]`
- `Barack Obama became president of United States in 2009`

The abductive strategy of getting useful clauses from the Low Clauses KB, takes into account a metric $Confidence_c$ defined as follows:

$$Confidence_c = \frac{|\bigcap(Feats_q, Feats_c)|}{|Feats_q|} \tag{2}$$

where $Feats_q$ is the features vector extracted from the query, and $Feats_c$ is the features vector of a generic record stored in the Low Clauses KB. A typical access to the Low Clauses KB creates the sorted list of all features occurrences, together with the related clauses, then the most relevant clauses will be copied in the High Clauses KB.

**Algorithm 1** aggregate_clauses_greedy(*clause*, *aggregated*, *threshold*)

**Input:** (i) *clause*: a `definite clause`, (ii) *aggregated*: a set of `definite clauses` (empty in the first call), (iii) *threshold*: the minimum `confidence` threshold

**Output:** a set of `definite clauses`.

```
ft ← extract_features(clause);
aggr ← get_relevant_clauses_from_db(ft);
foreach record in aggr do
    occurrencies_found ← record.features_occurrencies
    confidence ← occurrencies_found / size(ft)
    foreach cls in record.clauses do
        if cls not in aggregated and confidence ≥ threshold then do
            aggregated.append(cls);
            aggregate_clauses_greedy(cls, aggregated, threshold);
return aggregated
```

Such an operation is accomplished via the `aggregate_clauses_greedy` algorithm shown in Algorithm 1. The latter, with a greedy heuristic, takes in input the query *clause*, the set *aggregated* and the wanted limitation of confidence *threshold* for abduction; as output, it gives back the set *aggregated* of clauses from the db that are going to be copied in the High Clauses KB, taking in account of the wanted distance (2) from the query.

**Table 1**
A simple instance of clauses related features.

| Clauses | Features |
|---|---|
| $\mathtt{cls}_1$ | `[a, x, z, y]` |
| $\mathtt{cls}_2$ | `[a, b]` |
| $\mathtt{cls}_3$ | `[a, b, x, y]` |
| $\mathtt{cls}_4$ | `[a, b, c, d]` |
| $\mathtt{cls}_5$ | `[a, b, c, w]` |

## 4.1. Algorithm

First, at line 1 of Algorithm 1, `aggregate_clauses_greedy` extracts all *clause* features; at line 2, it creates a list of tuples in descending order (by the first field) containing an integer value and a lists of clauses having in common such value as number of common features with *clause*. More in detail, considering the Table 1, having a query $\mathtt{cls}_q$ with features vector `[a, b, c]`, the function `get_relevant_clauses_from_db` will create a list made of the following two tuples, by excluding those with only one feature in common, since in such a domain a minimal meaningful verbal phrase[8] is made at least of two entities (verb plus subject):

$$\mathtt{(3,\ [cls_4,\ cls_5])}$$
$$\mathtt{(2,\ [cls_2,\ cls_3])}$$

The rationale behind such function's output is that the first value of each tuple is the size of the intersection between the features of the query and the features of Table 1 (*feature_occurencies* in Algorithm 1), while the second value (*clauses* in Algorithm 1) is a vector containing the clauses themselves having in common such an intersection size. At line 4, the first value of each tuple is extracted and used in line 5 to calculate the confidence (2). In the loop at line 6, all clauses having the same features occurrences are considered, and at line 7 the algorithm checks whether the clause has an admitted confidence level and it is not already in *aggregated*. In this case, the clause will be appended to the *aggregated* list. At line 9, there is a recursive call taking in input *cls* (the current clause which is being processed) instead of *clause*, the updated *aggregated* and the *threshold* of the same procedure call. Finally, at line 10, the list *aggregated* is returned.

Although there can be more strategies to implement the function at line 2 of Algorithm 1, for this work's prototype the desired result has been achieved by leveraging the Mongodb aggregation operator, in a single fast and efficient database operation shown in Figure 2. The latter is a pipeline of four different operations: the first, at line 4, is the processing of all possible sizes of intersections between features fields in the db and the features of the query clause. At line 6 all clauses with the common value of such a size are grouped in tuples. At line 7 such tuples are sorted by the intersection size. Finally, at line 8 the output is limited to the two most significant tuples. Nonetheless, we can affirm the algorithm is sound, accomplishing its job in at most:

$$O(t * s^2)$$

---

[8] For instance a reflective verbal phrase as: *Roy runs.*

```
aggr = db.clauses.aggregate([
       {"$project": {
           "value": 1,
           "intersection": {"$size": {"$setIntersection": ["$features", features]}}
       }},
       {"$group": {"_id": "$intersection", "group": {"$push": "$value"}}},
       {"$sort": {"_id": -1}},
       {"$limit": 2}
   ])
```

**Figure 2:** The Python Mongodb aggregate operator implementing the function `get_relevant_clauses_from_db` at line 2 of Algorithm 1.

with $0 < t < 1$ as the confidence threshold and $s$ the size of the Low Clauses KB.

### 4.2. Polar Questions

Polar questions in the shape of nominal assertion (excepting for the question mark at the end) are transformed into nested definite clauses and treated as query as they are, while those beginning with an auxiliary term, for instance:

*Has Margot said the truth about her life?*

which can be distinguished by means of the following dependency:

`aux(said, Has)`

will be treated by removing the auxiliary and considering the remaining text (without the ending question mark) as source to be converted into a clause-query.

### 4.3. Wh-Questions

Differently from polar questions, for dealing with wh-question we have to transform a question into one or more assertions that can be expected as likely answer, then to query the agent with them. To achieve that, after an analysis of several types of questions for each category[9], by leveraging the dependencies of the questions, we found it useful to divide the sentences text into specific chunks as it follows:

`[PRE_AUX][AUX][POST_AUX][ROOT][POST_ROOT][COMPL_ROOT]`

The delimiter indexes between every chunk are given starting from the AUX and ROOT dependencies positions in the sentence. The remaining chunks are extracted on the basis of the latters. For the likely answers composition, the module *QA Shifter* has the task of recombining the question chunks in a different permutation, depending on the idiom in exam, considering also the wh-question type. Such an operation, which is strictly language specific, is accomplished thanks to an *ad-hoc* production rule system which takes in account of languages diversity. For instance, let the question in exam be:

---

[9] Who, What, Where, When, How.

*Who could be Biden?*

In this case, the chunks sequence will be as follows:

`[PRE_AUX][could][POST_AUX][be][Biden][COMPL_ROOT]`

where only `AUX`, `ROOT` and `POST_ROOT` are populated, while the other chunks are empty. In this case a specific production rule of the QA Shifter will recombine all chunks in a different sequence, by adding also another specific word in order to compose a likely assertion as follows:

`[PRE_AUX][POST_AUX][Biden][could][be][COMPL_ROOT][Dummy]`

At this point, joining all the words in such a sequence, the candidate sentence as likely assertion to used as query, might be the following one:

*Biden could be Dummy.*

The meaning of the keyword *Dummy* will be clarified shortly. In all verbal phrases where `ROOT` is a copular[10] verb (like *be*), the verb has the same properties of the identity function. Then, in the case of the above sentence, we can consider also the following candidate sentence:

*Dummy could be Biden.*

All wh-questions for their own nature require a *factoid* answer, made of one or more words (snippet); so, in the presence of the question: *Who is Biden?* as answer we expect something like:

$$\textit{Biden is something.} \tag{3}$$

But *something* surely is not what we are looking for as information, but *the elected president of United States* or something similar. This means that, within the FOL expression of the query, *something* in (3) must be represented by a mere variable. In light of this, instead of *something*, this architecture uses the keyword *Dummy*: the rationale of this choice is that, during the creation of a FOL expression containing such a word, the Translation Service will impose the extra POS `DM` to *Dummy*, whose parsing is not expected to be used to build a nested definite clause, thus it will be discarded. At the end of this process, as FOL expression of the query we'll have the following literal:

$$\texttt{Be(Biden(x1), x2),} \tag{4}$$

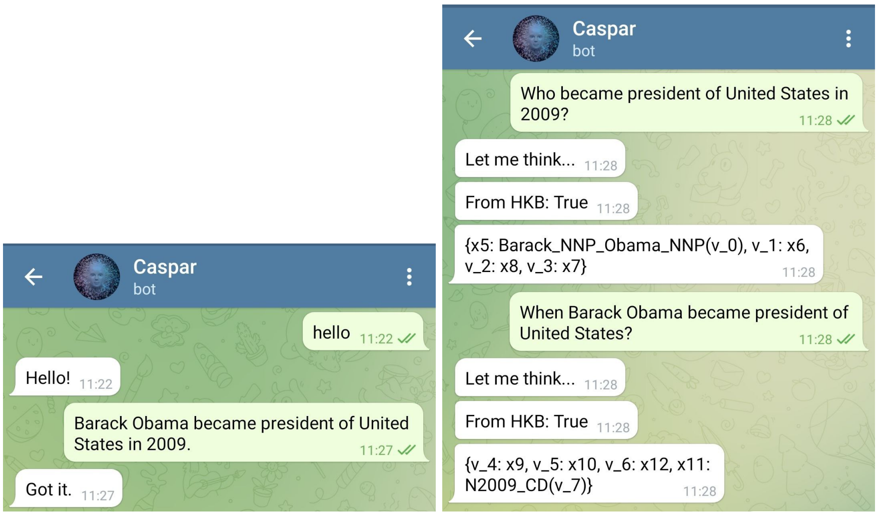
which means that, if the High Clauses KB contains the representation of *Biden is the president of America*, namely:

`Be(Biden(x1), Of(President(x2), America(x3))),`

querying with the (4) by using the Backward-Chaining algorithm, the latter will return back as result a unifying substitution with the previous clause as follows:

---

[10] A copular verb is a special kind of verb used to join an adjective or noun complement to a subject. Common examples are: be (is, am, are, was, were), appear, seem, look, sound, smell, taste, feel, become, and get. A copular verb expresses either that the subject and its complement denote the same thing or that the subject has the property denoted by its complement.



hello 11:22

Hello! 11:22

Barack Obama became president of United States in 2009. 11:27

Got it. 11:27

Who became president of United States in 2009? 11:28

Let me think... 11:28

From HKB: True 11:28

{x5: Barack_NNP_Obama_NNP(v_0), v_1: x6, v_2: x8, v_3: x7} 11:28

When Barack Obama became president of United States? 11:28

Let me think... 11:28

From HKB: True 11:28

{v_4: x9, v_5: x10, v_6: x12, x11: N2009_CD(v_7)} 11:28

**Figure 3:** Starting a Telegram chat session with an instance of AD-Caspar (left) and querying it with *who* and *when* questions (right).

```
{v_41: x1, x2: Of(President(v_42), America(v_43))},
```

which contains, in correspondence of the variable x2, the logic representation of the snippet: *president of America* as possible and correct answer. Furthermore, starting from the lemmas composing the only answer-literal within the substitution, with a simple operation on a string, it is possible to extract the minimum snippet from the original sentence containing such lemmas.

## 5. Case-study

In this section, a simple Python prototype of a Telegram chatbot based on the AD-Caspar architecture is shown, focusing on how it deals with each type of interaction with the user.

### 5.1. Starting, Asserting and Querying the chatbot

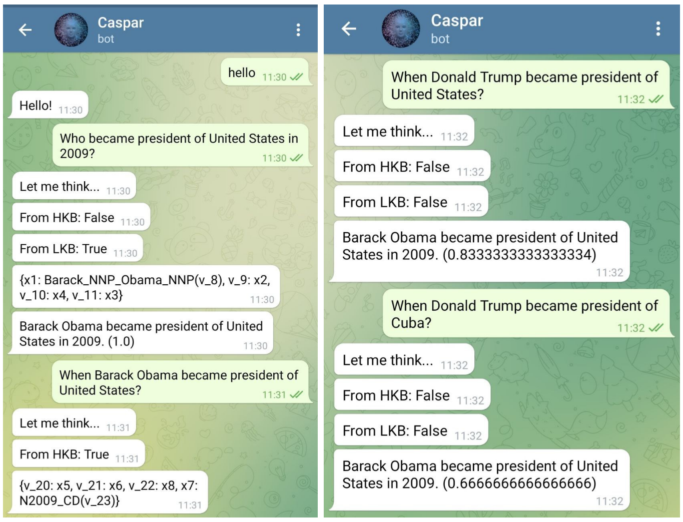
As the agent is running, to start a new session the user has to provide the keyword *hello*, to make the agent come out from its idle state, then feed or enquiry the chatbot with the wanted information.

In Figure 4 it is shown the content of both High and Low Clauses KB in the case of new assertions, after the events of Figure 3 (left), by means the AD-Caspar native commands `hkb()` and `lkb()`. In Figure 3 (right) we can see how the chatbot is queried with *wh-questions*, giving back as result a substitution from the High Clauses KB (`From HKB: True`) containing a literal, which is a logic representation of a snippet-result in natural language. Regarding the substitution of the variable x5, e.g., the literal is the representation of the snippet *Barack Obama*, whose words are concatenated together their POS.

After the chatbot reboot, as we can see for instance in Figure 5 (left), the result is extracted from Low Clauses KB (`From LKB: True`) taking into account the confidence threshold (0.6

```

eShell: main > hkb()

In_IN(Become_VBD(Barack_NNP_Obama_NNP(x1), Of_IN(President_NN(x2), United_NNP_States_NNP(x3))), N2009_CD(x4))

1 clauses in High Knowledge Base

eShell: main > lkb()

In_IN(Become_VBD(Barack_NNP_Obama_NNP(x1), Of_IN(President_NN(x2), United_NNP_States_NNP(x3))), N2009_CD(x4))
['In_IN', 'Become_VBD', 'Barack_NNP_Obama_NNP', 'Of_IN', 'President_NN', 'United_NNP_States_NNP', 'N2009_CD']
Barack Obama became the president of United States in 2009.

1  clauses in Low Knowledge Base
```

**Figure 4:** AD-Caspar High and Low Clauses KB changes, after assertions.



hello 11:30

Hello! 11:30

Who became president of United States in 2009? 11:30

Let me think... 11:30

From HKB: False 11:30

From LKB: True 11:30

{x1: Barack_NNP_Obama_NNP(v_8), v_9: x2, v_10: x4, v_11: x3} 11:30

Barack Obama became president of United States in 2009. (1.0) 11:30

When Barack Obama became president of United States? 11:31

Let me think... 11:31

From HKB: True 11:31

{v_20: x5, v_21: x6, v_22: x8, x7: N2009_CD(v_23)} 11:31

When Donald Trump became president of United States? 11:32

Let me think... 11:32

From HKB: False 11:32

From LKB: False 11:32

Barack Obama became president of United States in 2009. (0.8333333333333334) 11:32

When Donald Trump became president of Cuba? 11:32

Let me think... 11:32

From HKB: False 11:32

From LKB: False 11:32

Barack Obama became president of United States in 2009. (0.6666666666666666) 11:32

**Figure 5:** On the left, querying the chatbot with *who* and *when* questions, after the rebooting and the Low Clauses KB was fed. On the right, abductive results with confidence threshold equal to 0.6, after querying with *when* questions.

in this case), because High Clauses KB is still empty (`From HKB: False`). Such a threshold, depending on the domain, can be changed by modifying the value of a specific parameter. In the bot closed-world assumption, the agent can give back only answers unifying with the content of its own knowledge, otherwise it will return *False*. Optionally, by setting another parameter in a configuration file, the closest results can be shown together with their confidence (Figure 5 right).

## 5.2. Runtime Evaluation

In the scope of chatbot applications, the issue of responsiveness is worth of attention, because the user should have the feeling, somehow, of relating to a sentient being providing reasonable

```
Be(Nono(x1), Hostile(Nation(x2)))
Be(Colonel_West(x1), American(x2))
Be(Missile(x1), Weapon(x2))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3))
To(Sell(American(x1), Weapon(x2)), Hostile(Nation(x3))) ==> Be(American(x4), Criminal(x5))
```

**Figure 6:** AD-Caspar High Clauses after five assertions related to the *Colonel West* KB.

response times. In light of above, to address such an issue, since a chatbot relays on the internet, its real-time performances depends firstly on the bandwidth and secondly on the chatbot engine. Since the bandwidth is the more volatile parameter between the two, because it depends on physical features, protocols and temporally congestion, in this work we decided to focus only on the engine performances considering a widespread specific hardware. For this reason, in order to achieve a runtime evaluation of the engine, an instance of AD-Caspar was tested, whose timings in seconds are shown in Table 2. The hardware the chatbot has been tested on is the Intel i7-8550U 1.80Ghz with 8GB RAM. The first column `Knowledge Base` of Table 2 refers to three distinct KBs, each with different size, containing the clauses in Figure 7. Such clauses are asserted starting from the five sentences related to the *Colonel West* case, namely:

- *Colonel West is American.*
- *Nono is a hostile nation.*
- *missiles are weapons.*
- *Colonel West sells missiles to Nono.*
- *When an American sells weapons to a hostile nation, that American is a criminal.*

whose corresponding FOL notation, seen before in Section 3 (see also Figure 6), is not sufficient to infer that *Colonel West is a criminal.*

For this reason, together with each sentence related clause, AD-Caspar adopts a specific *expansion* of the KB, inherited from Caspar, which leverages the so-defined *assignment rules* and *clause conceptual generalizations.* For the sake of shortness we will not report the details of such an expansion, for which the reader is referred to [6]. Anyway, such an expansion improves the chances of successful reasoning, by increasing the clauses from 5 in Figure 6 up to 25 in Figure 7. The first column in Table 2 is about three distinct KBs: the first, namely *West25*, contains exactly such 25 clauses; *West104* and *West303* contain either such clauses, but respectively plus 79 and plus 278 random unrelated clauses. The column HKB of Table 2 refers to five computation timings for each of the KBs, considering only the High Clauses KB and the query: *Colonel West is a criminal.* We remind that an instance of AD-Caspar attempts firstly to achieve a reasoning making usage of only the High Clauses KB, as in the case of Caspar, otherwise it will get likely candidates from the Low Clauses KB, considering their relevance to the query according to a specific threshold confidence (2). The third column LKB+HKB of Table 2 shows timings in the case the High Clauses KB is initially empty, thus both High Clauses KB and Low Clauses KB are involved. Focusing on the third column, it appear clear timings in general are lesser than the second column, excepting for the values in first rows, which comprises the Mongodb access time and the filling of the High Clauses KB with clauses coming

```
Be(Nono(x1), Nation(x2))
Be(Nono(x1), Hostile(Nation(x2)))
Nono(x) ==> Nation(x)
Nono(x) ==> Hostile(Nation(x))
Be(Colonel_West(x1), American(x2))
Colonel_West(x) ==> American(x))
Be(Missile(x1), Weapon(x2))
Missile(x) ==> Weapon(x)
Sell(Colonel_West(x1), Missile(x2)) ==> Sell(American(v_0), Missile(x4))
Sell(Colonel_West(x1), Missile(x2)) ==> Sell(American(x3), Weapon(v_1))
Sell(Colonel_West(x1), Missile(x2)) ==> Sell(Colonel_West(x1), Weapon(v_2))
Sell(Colonel_West(x1), Missile(x2))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(x1), Missile(x2)), Nation(v_4))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_5), Missile(v_6)), Nation(v_4))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_7), Weapon(v_8)), Nation(v_4))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(v_9), Weapon(v_10)), Nation(v_4))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(x1), Missile(x2)), Hostile(Nation(v_11))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_12), Missile(v_13)), Hostile(Nation(v_11))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_14), Weapon(v_15)), Hostile(Nation(v_11))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(v_16), Weapon(v_17)), Hostile(Nation(v_11))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_18), Missile(v_19)), Nono(x3))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_22), Weapon(v_23)), Nono(x3))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(v_26), Weapon(v_27)), Nono(x3))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3))
To(Sell(American(x1), Weapon(x2)), Hostile(Nation(x3))) ==> Be(American(x4), Criminal(x5))
```

**Figure 7:** AD-Caspar *Colonel West* KB in Figure 6 after its expansion.

from the Low one, via the `aggregate_clauses_greedy` in Algorithm 1. The other values in the third column are lower than in the second one because the reasoning is achieved over a lesser number of clauses (19) for each distinct KB. The average timings in the bottom row show how the first rows' value is amortized, by compensating the loss, due to the gain achieved from reasoning on a fewer number of clauses than respect the initial content of all KBs in exam (*West25*, *West104* and *West303*). Intuitively it is expected such bias to be increased for larger KBs, which demonstrates the effectiveness of such approach for two distinct tasks: firstly, to deal with larger KBs considering only the most relevant clauses in the reasoning process; secondly, to permit at the same time abduction as pre-stage of deduction, in order to give back closer results also in presence of non-successful reasoning.

## 6. Conclusions and Future works

In this paper, a framework based on natural language processing and first-order logic, with the aim of instantiating *cognitive* chatbots able of abductive-deductive reasoning, was presented. By the means of its module Translation Service, AD-Caspar parses sentences in natural language in order to populate its KBs with beliefs or *nested* definite clauses endowed of rich semantic. Moreover, the module QA Shifter is able to reshape wh-questions into likely assertions one can expect as answer, thanks to a production rule system leveraging a dependency parser. The combination of Translation Service and QA Shifter makes the Telegram Bot proposed in this work easily scalable on the knowledge we want it to deal with, because the user has to provide

**Table 2**
Real-time cognitive performances (in seconds) of a Telegram chatbot engine based on AD-Caspar, in the case of successful reasoning with KBs of different sizes.

| `Knowledge Base` | HKB | LKB+HKB |
|---|---|---|
| *West25* | 0,377 | 0,469 |
| | 0,378 | 0,353 (19/25) |
| | 0,437 | 0,385 (19/25) |
| | 0,374 | 0,366 (19/25) |
| | 0,355 | 0,399 (19/25) |
| *West104* | 0,423 | 0,426 |
| | 0,362 | 0,327 (19/104) |
| | 0,342 | 0,327 (19/104) |
| | 0,353 | 0,388 (19/104) |
| | 0,731 | 0,323 (19/104) |
| *West303* | 0,407 | 0,463 |
| | 0,421 | 0,357 (19/303) |
| | 0,377 | 0,333 (19/303) |
| | 0,461 | 0,368 (19/303) |
| | 0,443 | 0,387 (19/303) |
| `Overall AVG` | 0,416 | 0,378 |

only the new sentences in natural language at runtime, like in a common conversation. As future work, we plan to include a module for the design of enhanced dialog systems, taking in account of contexts and history, and also integrate a module for Argumentation.

## References

[1] M. Minsky, A framework for representing knowledge. In P. Winston, Ed., *The Psychology of Computer Vision*, New York: McGraw-Hill, 1975.

[2] . A. Schank, R., Scripts, plans, goals, and understanding: An inquiry into human knowledge structures, Hillsdale, NJ: Erlbaum, 1977, pp. 211–277.

[3] C. F. Longo, C. Santoro, F. F. Santoro, Meaning Extraction in a Domotic Assistant Agent Interacting by means of Natural Language, in: 28th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE, 2019.

[4] C. S. Carmelo Fabio Longo, Ad-caspar: Abductive-deductive cognitive architecture based on natural language and first order logic reasoning, in: 4th Workshop on Natural Language for Artificial Intelligence (NL4AI 2020) co-located with the 19th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2020), 2020.

[5] J. H. Kotagiri Ramamohanarao, An introduction to deductive database languages and systems, The International Journal of Very Large Data Bases Journal, 3, 107-122 (1994).

[6] C. F. Longo, F. Longo, C. Santoro, Caspar: Towards decision making helpers agents for iot, based on natural language and first order logic reasoning, Engineering Applications of Artificial Intelligence 104 (2021) 104269. URL: https://www.sciencedirect.com/science/article/pii/S0952197621001160. doi:`https://doi.org/10.1016/j.engappai.2021.104269`.

[7] A. Turing, Computing machinery and intelligence, Mind, 1950, pp. 433–60.

[8] H. Madhumitha.S, Keerthana.B, Interactive chatbot using aiml, Int. Jnl. Of Advanced Networking & Applications Special Issue (2019).

[9] Q. V. L. Ilya Sutskever, Oriol Vinyals, Sequence to sequence learning with neural networks, Advances in Neural Information Processing Systems 27 (2014).

[10] C. S. Fabio D'Urso, Carmelo Fabio Longo, Programming intelligent iot systems with a python-based declarative tool, in: The Workshops of the 18th International Conference of the Italian Association for Artificial Intelligence, 2019.

[11] D. Schacter, Implicit memory: history and current status, Journal of Experimental Psychology: Learning, Memory, and Cognition vol. 13, 1987, pp. 501–518 (1987).

[12] A. S. Jinho D. Choi, Joel Tetreault, It depends: Dependency parser comparison using a web-based evaluation tool, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, 2015, p. 387–396.

[13] D. Davidson, The logical form of action sentences, in: The logic of decision and action, University of Pittsburg Press, 1967, p. 81–95.

[14] G. A. Miller, Wordnet: A lexical database for english, in: Communications of the ACM Vol. 38, No. 11: 39-41, 1995.