A Reactive Cognitive Architecture based on Natural Language Processing for the task of Decision-Making using a Rich Semantic

Carmelo Fabio Longo[a], Francesco Longo[b] and Corrado Santoro[a]

[a]Department of Mathematics and Computer Science, University of Catania, Viale Andrea Doria, 6, 95125 Catania, Italy
[b]Department of Engineering, University of Messina, Contrada di Dio, S. Agata, 98166 Messina, Italy

Abstract

The field of cognitive architectures is rich of approaches featuring a wide range of typical abilities of human mind, like perception, action selection, learning, reasoning, meta-reasoning and others. However, those leveraging Natural Language Processing are quite limited in both domain and reasoning capabilities. In this work, we present a cognitive architecture called CASPAR, based on a Belief-Desire-Intention framework, capable of reactive reasoning using a highly descriptive semantic made of First Order Logic predicates parsed from natural language utterances.

Keywords

Cognitive Architecture, Natural Language Processing, Artificial Intelligence, First Order Logic, Internet of Things

1. Introduction

In the last decade, a large number of devices connected together and controlled by AI has entered in millions of houses: the pervasive market of Internet of Things (IoT). Such a phenomenon is extended also in domains other than the domestic one, such as smart cities, remote e-healthcare, industrial automation, and so on. In most of them, especially the usual domestic ones, *vocal assistants* assume an important role, because voice is the most natural way to give the user the feeling to deal with an intelligent sentient being who cares about the proper functioning of the home environment. But how *intelligent* are these vocal assistants actually? Although there can be more definitions of *intelligence*, in this work we are interested only in those related to autonomous agents acting in the scope of decision-making.

Nowadays, companies producing vocal assistants aim more at increasing their pervasiveness than at improving their native reasoning capabilities; with *reasoning capabilities*, we can intend not only the ability to infer the proper association *command* $\rightarrow$ *plan* from utterances, but also to be capable of combining facts with rules in order to infer new knowledge and help the user in decision-making tasks.

Except the well known cloud-based vocal assistants [1], other kind of solutions [2, 3, 4] are based on neural models exclusively trained on the domotic domain; or they exploit chat

WOA 2020: Workshop "From Objects to Agents", September 14–16, 2020, Bologna, Italy
fabio.longo@unict.it (C. F. Longo); flongo@unime.it (F. Longo); santoro@dmi.unict.it (C. Santoro)
0000-0002-2536-8659 (C. F. Longo); 0000-0001-6299-140X (F. Longo); 0000-0003-1780-5406 (C. Santoro)

engines [5, 6] whose understanding skills are strictly depending on syntax. This makes the range of their capabilities quite limited.

In light of the above, in this paper our aim is the design of a cognitive architecture, called CASPAR, based on Natural Language Processing (NLP), that makes it possible the implementation of intelligent agents able to outclass the available ones in performing deductive activities. Such agents could be used for both domotic purposes and any other kind of applications involving common deductive processes based on natural language. As a further motivation, we have to highlight that, as claimed in [7], cognitive architectures have been so far mainly used as research tools, and very few of them have been developed outside of academia; moreover, none of them has been specifically designed for IoT. Of course, most of them have features and resources which could be exploited in such a domain, but the starting motivations were different from ours.

Although cognitive architectures should be distinguished from models that implement them, our architecture can be used as domotic agent *as is*, after the definitions of both the involved entities and the I/O interfaces.

This paper is structured as follows: Section 2 describes the state of the art of related literature; Section 3 shows in detail all the architecture's components and underlying modules; Section 4 shows the architecture reasoning heuristic in the presence of clauses made of composite predicates, taking into account possible argument substitutions as well; Section 5 summarizes the content of the paper and provides our conclusions, together with future work perspectives. A Python implementation of CASPAR is also provided for research purposes in a Github repository[1].

2. Related work

The number of existing cognitive architectures has reached several hundreds according to the authors of [7]. Among the most popular ones, which also influenced several subsequent works, there are SOAR, CLARION and LIDA, mentioned in a theoretical comparison in [8]. Most of them got inspired either by neuroscience or psychanalysis/philosophy studies; the former are surely less fancy, being supported by scientific data regarding functions of brain modules in specific conditions and their interactions. The Integrated Information Theory [9] provides even a metric *Phi* to evaluate the consciousness level of a cognitive system, which would be proportional to those overall interactions. In this section, we will focus mostly on those architectures implementing Reasoning/Action Selection, Natural Language Processing and Decision-Making, being the main basis on which CASPAR has been built.

In [10] the authors describe three different spoken dialog systems, one of them based on the FORR architecture and designed to fulfill the task of ordering books from the public library by phone. All the three dialog systems are based on a local Speech-to-Text engine called PocketSphinx which is notoriously less performing than cloud-based systems [11]. This leads to a greater struggle to reduce the bias between user's request and result.

The authors of [12] present a computational model called MoralIDM, which integrates multiple AI techniques to model human moral decision-making, by leveraging a two-layer

[1]http://www.github.com/fabiuslongo/pycaspar

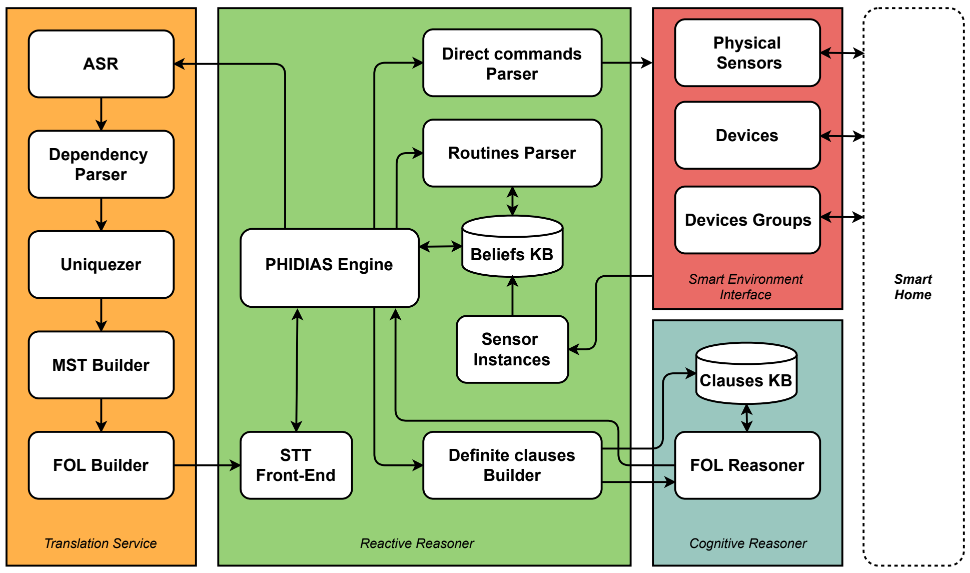

Figure 1: The Software Architecture of CASPAR

inference engine which takes into account prior cases decisions and a knowledge base with a formal representation of moral quality-weighted facts. Such facts are extracted from natural language by using a semi-automatic translator from simplified English (which is the major weakness of such approach) scenarios into predicate calculus.

The DIARC architecture [13] has been designed for addressing the issue of recognizing morally and socially charged situations in human-robot collaborations. Although it exploits several well known NLP resources (such as Sphinx, Verbnet, and Framenet), it has been tested only on trivial examples in order to trigger robot reactions, using an ad-hoc symbolic representation of both known and perceived facts.

In general, probing the existing cognitive architectures leveraging NLP, we have found that most of them are limited in both domain of application and in term of semantic complexity.

3. The Architecture

The name that has been chosen for the architecture presented in this paper is CASPAR. It derives from the following words: **C**ognitive **A**rchitecture **S**ystem **P**lanned **and** **R**eactive, whom summarize its two main features. In Figure 1, all interacting components are depicted, filled with distinct colours.

The main component of this architecture, namely the *Reactive Reasoner*, acts as "core router" by delegating operations to other components, and providing all needed functions to make the whole system fully operative.

This architecture's Knowledge Base (KB) is divided into two distinct parts operating separately, which we will distinguish as *Beliefs KB* and *Clauses KB*: the former contains information of physical entities which affect the agent and which we want the agent to affect; the latter contains conceptual information not perceived by agent's sensors, but on which we want the agent to make logical inference.

The Beliefs KB provides exhaustive cognition about what the agent could expect as input data coming from the outside world; as the name suggests, this cognition is managed by means of proper beliefs that can - in turn - activate proper plans in the agent's behaviour.

The Clauses KB is defined by the means of assertions/retraction of *nested* First Order Logic (FOL) definite clauses, which are possibly made of composite predicates, and it can be interrogated providing answer to any query (*True* or *False*).

The two KBs represent, somehow, two different kinds of human being memory: the so called *procedural memory* or *implicit memory*[14], made of thoughts directly linked to concrete and physical entities; the *conceptual memory*, based on cognitive processes of comparative evaluation.

As well as in human being, in this architecture the two KBs can interact with each other in a very reactive decision-making process.

3.1. The Translation Service

This component (left box in Figure 1) is a pipeline of five modules with the task of taking a sound stream in natural language and translating it in a *neo-davidsonian* FOL expression inheriting the shape from the event-based formal representation of Davidson [15], where for instance the sentence:

$$\textit{Brutus stabbed suddenly Caesar in the agora} \tag{1}$$

is represented by the following notation:

$$\exists \texttt{e stabbed(e, Brutus, Caesar)} \land \texttt{suddenly(e)} \land \texttt{in(e, agora)}$$

The variable e, which we define *davidsonian* variable, identifies the verbal action related to *stabbed.* In the case a sentence contains more than one verbal phrases we'll make usage of indexes for distinguish e_i from e_j with $i \neq j$.
As for the notation used in this work, it does not use ground terms as arguments of the predicates, in order to permit the sharing of different features related to the same term like it follows, whether we include the adjective *evil*:

$$\exists \texttt{e stabbed(e, Brutus(x), Caesar(y))} \land \texttt{evil(x)} \land \texttt{suddenly(e)} \land \texttt{in(e, agora(z))}$$

which can also be represented, *ungrounding* the verbal action arguments, as it follows:

$$\exists \texttt{e stabbed(e, x, y)} \land \texttt{Brutus(x)} \land \texttt{Caesar(y)} \land \texttt{evil(x)} \land \texttt{suddenly(e)} \land \texttt{in(e, z)} \land \texttt{agora(z)}$$

Furthermore, in the notation used for this work each predicate label is in the form `L:POS(t)`, where `L` is a lemmatized word and `POS` is a Part-of-Speech (POS) tag from the Penn Treebank tagset[16].

The first module in the pipeline, i.e., the *Automatic Speech Recognition* [17, 18, 19] (ASR), allows a machine to understand the user's speech and convert it into a series of words.

The second module is the *Dependency Parser*, which aims at extracting the semantic relationships, namely *dependencies*, between all words in a utterance. In [20], the authors present a comparative analysis of ten leading statistical dependency parsers on a multi-genre corpus of English.

The third module, the *Uniquezer*, aims at renaming all the entities within each dependency in order to make them unique. Such a task is mandatory to ensure the correctness of the outcomes of the next module in the pipeline (the *Macro Semantic Table*), whose data structures need a distinct reference to each entity coming from the dependency parser.

The fourth module, defined as *MST Builder*, has the purpose to build a novel semantic structure defined as *Macro Semantic Table* (MST), which summarizes in a canonical shape all the semantic features in a sentence, starting from its dependencies, in order to derive FOL expressions.

Here is a general schema of a MST, referred to the utterance u:

```
MST(u) = {ACTIONS, VARLIST, PREPS, BINDS, COMPS, CONDS}
```

where

$$\begin{aligned}
\texttt{ACTIONS} &= [(\texttt{label}_k,\ \texttt{e}_k,\ \texttt{x}_i,\ \texttt{x}_j),\ldots] \\
\texttt{VARLIST} &= [(\texttt{x}_1,\ \texttt{label}_1),\ldots(\texttt{x}_n,\ \texttt{label}_n)] \\
\texttt{PREPS} &= [(\texttt{label}_j,\ (\texttt{e}_k \mid \texttt{x}_i),\ \texttt{x}_j),\ldots] \\
\texttt{BINDS} &= [(\texttt{label}_i,\ \texttt{label}_j),\ldots] \\
\texttt{COMPS} &= [(\texttt{label}_i,\ \texttt{label}_j),\ldots] \\
\texttt{CONDS} &= [\texttt{e}_1,\ \texttt{e}_2,\ldots]
\end{aligned}$$

All tuples inside such lists are populated with variables and labels whose indexing is considered disjoint among distinct lists, although there are significant relations which will be clarified later. The MST building takes into account also the analysis done in [21] about the so-called *slot allocation*, which indicates specific policies about entity's location inside each predicate, depending on verbal cases. This is because the human mind, in the presence of whatever utterance, is able to populate implicitly any semantic role (identified by subject/object slots) taking part in a verbal action, in order to create and interact with a logical model of the utterance. In this work, by leveraging a step-by-step dependencies analysis, we want to create artificially such a model, to give an agent the chance to make logical inference on the available knowledge. All the dependencies used in this paper are part of the ClearNLP[22] tagset, which is made of 46 distinct entries. For instance, considering the dependencies of 1:

```
nsubj(stabbed, Brutus)
ROOT(stabbed, stabbed)
advmod(stabbed, suddenly)
dobj(stabbed, Caesar)
prep(stabbed, In)
det(agora, The)
pobj(in, agora)
```

from the couple nsubj/dobj it is possible to create new a tuple inside ACTIONS as it follows, taking also in account of variables indexing counting:

(stabbed, e_1, x_1, x_2)

and inside VARLIST as well:

(x_1, Brutus)
(x_2, Caesar)

Similarly, after an analysis of the couple prep/pobj it is possibile to create further tuples inside PREPS like it follows:

(in, e_1, x_3)

and inside VARLIST:

(x_3, agora)

The dependency advmod contains informations about the verb (*stabbed*) is going to modify by means the adverb *suddenly*. In light of this, a further tuple inside VARLIST will be created as it follows:

(e_1, suddenly)

As for the BINDS list, it contains tuples with a quality modifiers role: in the case the 1 had *the brave Caesar* as object, a further dependency amod will be created as it follow:

amod(Caesar, brave)

In this case, a *bind* between *Caesar* and *brave* will be created inside BINDS as it follows:

(Caesar, brave)

As with BINDS, COMPS contains tuples of terms related to each other, but in this case they are part of multi-word nouns like *Barack Hussein Obama*, whose nouns after the first will be classified as compound by the dependency parser.

As for the CONDS lists, it contains davidsonian variable whose related predicates subordinate the remaining others. For instance in the presence of utterances like:

if the sun shines strongly, Robert drinks wine

or

while the sun shines strongly, Helen smiles in both cases, the dependency `mark` will give information about subordinate conditions related to the verb *shines*, which are `mark(shines, If)` and `mark(shines, while)`. In those cases, the davidsonian variable related to *shines* will populate the list CONDS. In the same way, in presence of the word *when*, a subordinate condition might be inferred as well; but since any adverbs are classified as `advmod` (as we have seen for *suddenly* before), it will be considered as subordinate condition only when its POS is WRB and not RB; the former denotes a wh-adverb, the latter a qualitative adverb.

The fifth and last module, defined as *FOL Builder*, aims to build FOL expressions starting from the MSTs. Since (virtually) all approaches to formal semantics assume the Principle of Compositionality[2], formally formulated by Partee [23], every semantic representation can be incrementally built up when constituents are put together during parsing. In light of the above, it is possible to build FOL expressions straightforwardly starting from a MST, which summarizes all semantic features extracted during a step-by-step dependencies analysis.
For the rest of the paper, the labels inside the MST tuples will be in the form of `lemma:POS`. Then, for instance, instead of `stabbed` we'll have `stab:VBD`, where *stab* is the lemmatization of *stabbed* and VBD is the POS representing a past tense.
For each tuple `(var, lemma:POS)` in VARLIST the following predicate will be created:

`lemma:POS(var)`

which represents a noun, such as `tiger:NN(`x_1`)` or `Robert:NNP(`x_1`)`[3]. `var` can also be a davidsonian variable when POS has the value of RB. In such cases, the tuples represent adverbs, such as `Hardly:RB(`e_1`)` or `Slowly:RB(`e_2`)`.
For each tuple `(lemma:POS, dav, subj, obj)` in ACTIONS, the following predicate will be created:

`lemma:POS(dav, subj, obj)`

representing a verbal action, such as `be:VBZ(`e_1`, `x_1`, `x_2`)` or `shine:VBZ(`e_2`, `x_3`, `x_4`)`.
For each tuple `(lemma:POS, dav/var, obj)` in PREPS the following predicate will be created:

`lemma:POS(dav/var, obj)`

where `dav/var` is a variable either in a tuple of ACTIONS or of VARLIST, respectively, while obj is a variable in a tuple of VARLIST. Such predicates represent verbal/noun prepositions.
For each tuple `(lemma:`POS_1`, lemma:`POS_2`)` in COMPS, whose first entity `lemma:`POS_1 is in a tuple of VARLIST, a predicate will be created as follows:

`lemma:`POS_2`(var)`

where `var` is the variable of a the tuple in VARLIST with `lemma:`POS_1 as second entity. In case of multi-word nouns, each further noun over the first of them in VARLIST will be encoded

[2]"The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined."

[3]Without considering entities enumeration.

within COMPS.
As for CONDS, its usage will be explained next with an example. Let the sentence in exam be:

When the sun shines strongly, Robert is happy (2)

the related MST is:

$$\texttt{ACTIONS = [(shine01:VBZ, e}_1\texttt{, x}_1\texttt{, x}_2\texttt{), be01:VBZ(e}_2\texttt{, x}_3\texttt{, x}_4\texttt{)]}$$
$$\texttt{VARLIST = [(x}_1\texttt{, sun01:NN), (x}_2\texttt{, ?), (x}_3\texttt{, Robert01:NNP), (x}_4\texttt{, happy01:JJ)]}$$
$$\texttt{CONDS = [e}_1\texttt{]}$$

It has to be noticed the numeration of the entities within each list, as effect of the Uniquezer processing before the MST building. As final outcome we'll have an implication like the following:

$$\texttt{shine01:VBZ(e}_1\texttt{, x}_1\texttt{, \_)} \wedge \texttt{sun01:NN(x}_1\texttt{)} \implies \texttt{be01:VBZ(e}_2\texttt{, x}_3\texttt{, x}_4\texttt{)} \wedge \texttt{Robert01:NNP(x}_3\texttt{)} \wedge \texttt{happy01:JJ(}x_4\texttt{)}$$

3.2. The Reactive Reasoner

As already mentioned, this component (central box in Figure 1) has the task of letting other modules communicate with each other; it also includes additional modules such as the Speech-To-Text (SST) Front-End, IoT Parsers (Direct Command Parser and Routine Parser), Sensor Instances, and Definite Clauses Builder. The Reactive Reasoner contains also the Beliefs KB, which supports both Reactive and Cognitive reasoning.

The core of this component processing is managed by the Belief-Desire-Intention Framework Phidias[24], which gives Python programs the ability to perform logic-based reasoning (in Prolog style) and lets developers write reactive procedures, i.e., pieces of program that can promptly respond to environment events.

The agent's first interaction with the outer world happens through the STT Front-End, which is made of production rules reacting on the basis of specific words asserted by an Instance Sensor; the latter, being instance of the superclass *Sensor* provided by Phidias, will assert a belief called STT(X) with X as the recognized utterance, after the sound stream is acquired by the microphone and translated in text by means of the ASR.

The Direct Command Parser has the task of combining FOL expressions predicates with common variables coming from the Translation Services, via a production rules system. The final outcome of such rules is a belief called INTENT, which might trigger another rule in the Smart Environment Interface. A similar behaviour is reserved to the Routine Parser, when subordinating conditions within an IoT command are detected; it produces two types of beliefs: ROUTINE and COND, linked together by a unique code. The belief ROUTINE is a sort of *pending* INTENT, which cannot match any production rule and execute its plan until the content of its related COND meets those of another belief asserted by a Sensor Instance and called SENSOR. Then, the ROUTINE belief will be turned into INTENT and get ready for the execution as direct command, as shown in lines 2, 3, 5, 7, 8 of Listing 1 in the Appendix

The Definite Clauses Builder is responsible of combining FOL expression predicates with common variables, through a production rules system, in order to produce nested definite clauses. Considering the 2 and its related FOL expression producted by the Translation Service, the production rule system of the Definite Clauses Builder, taking in account of the POS of each predicate, will produce the following *nested* definite clause:

$$\texttt{shine01:VBZ(sun01:NN(x}_1\texttt{), \_)} \implies \texttt{be01:VBZ(Robert01:NNP(x}_3\texttt{),}\\ \texttt{happy01:JJ(}x_4\texttt{))}$$

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 MST derived clauses suitable for doing inference with the Backward-Chaining algorithm (which works only with KB made of definite clauses), we must be able to incapsulate all their informations properly. The strategy followed is to create composite terms, taking into account of the POS tags and applying the following hierarchy to every noun expression as it follows:

$$\texttt{IN(JJ(NN(NNP(x))), t)} \tag{3}$$

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 actions, the nesting hierarchy will be the following:

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

where ADV is an adverb label, IN a preposition label, VB a verb label, and t_1, t_2, t_3 are predicates; in the case of intransitive or imperative verb, instead of respectively t_2 or t_1, the arguments of VB will be left void. As we can see, a preposition might be related either to a noun or a verb.

3.3. The Smart Environment Interface

This component (upper right box in Fig.1) provides a bidirectional interaction between the architecture and the outer world. In Listing 1 in the Appendix, a simple example is shown, where a production rules system is used as reactive tool to trigger proper plans in the presence of specific asserted beliefs. In [25] we have shown the effectiveness of this approach by leveraging the Phidias predecessor Profeta[26], even with a shallower analysis of the semantic dependecies, as well as an operations encoding via WordNet[27] in order to make the operating agent multi-language and multi-synonimous.

Such an interface includes a production rules system containing different types of entities definitions and operation codes involving the entities themself, which trigger specific procedures containing high level language (e.g., lines 11 and 12 in Listing 1 in the Appendix). The latter should contain all required functions for driving each device in order to get the desired behaviour, whose implementation in this work is left to the developer. Each production rule contains also subordinating conditions defined as *Active Beliefs*: `lemma_in_syn(X, S)` checks the membership of the lemma X to the synset S, to make the rule multi-language and multi-synonimous (after having defined the entities depending on the language); while, the Active

Belief `eval_cls(Y)` lets Belief KB and Clauses KB interact with each other in a very decision-making process, where the agent *decides* either to execute or not the related plan within the square brackets, accordingly to the reasoning of the query `Y`; the latter in line 12-13 of Listing 1 in the Appendix is the representation of the sentence *an inhabitant is at home*.

Finally, this module contains also production rules to change routines into direct commands according to the presence of specific belief related to conditionals, which might be asserted or not by some Sensor Instance (see lines 2, 3, 5, 8, 9 of Listing 1 in the Appendix).

3.4. The Cognitive Reasoner

This component (bottom right box in Figure 1) allows an agent to assert/query the Clauses KB with *nested* definite clauses, where each predicate argument can be another predicate and so on, built by the *Definite Clauses Builder* module (within the *Reactive Reasoner*).

Beyond the nominal FOL reasoning with the known Backward-Chaining algorithm, this module exploits also another class of logical axioms, the so-called *assignment rules*. We refer to a class of rules of the type "P is-a Q" where P is a predicate whose variable travels across one hand-side to another, with respect to the implication symbol. For example, if we want to express the concept: *Robert is a man*, we can use the following closed formula:

$$\forall \mathtt{x}\ \mathtt{Robert(x)} \implies \mathtt{Man(x)} \tag{4}$$

but before that, we must consider a premise: the introduction of such rules in a KB can be possible only by shifting all its predicates from a strictly semantic domain to a pure conceptual one, because in a semantic domain we have just the knowledge of morphological relationships between words given by their syntactic properties. Basically, we need a medium to give additional meaning to our predicates, which is provided by WordNet [27]. This allows us to make logical reasoning in a conceptual space thanks to the following functions:

$$F_I : P_S \longrightarrow P_C \quad F_{Args(F_I)} : X_S^n \longrightarrow Y_C^n \tag{5}$$

F_I is the *Interpreter Function* between the space of all semantic predicates which can be yield by the `MST` sets and the space of all conceptual predicates P_C; it is not injective, because a single semantic predicate might have multiple corrispondences in the codomain, one for each different synset containing the lemma in exam. $\mathrm{F}_{Args(F_I)}$ is between domain and codomain of all predicate's argument of F_I, which have equal arity. For instance, considering the FOL expression of (4):

$$\mathtt{be:VBZ(e_1,\ x_1,\ }x_2\mathtt{)} \wedge \mathtt{Robert:NNP(x_1)} \wedge \mathtt{man:NN(}x_2\mathtt{)}$$

After an analysis of *be*, we find the lemma within the WordNet synset encoded by `be.v.01` and defined by the gloss: *have the quality of being something*. This is the medium we need for the domain shifting which gives a common sense meaning to our predicates.

In light of above, in the new conceptual domain given by (5), the same expression can be rewritten as:

$$\mathtt{be\_VBZ(d_1,\ y_1,\ y_2)} \wedge \mathtt{Robert\_NNP(y_1)} \wedge \mathtt{man\_NN(y_2)}$$

where `be_VBZ` is fixed on the value which identify y_1 with y_2, `Robert_NNP(x)` means that x identify *Robert*, and `man_NN(x)` means that x identify *a man*.

Considering the meaning of `be_VBZ`, it does make sense also to rewrite the formula as:

$$\forall y \ \texttt{Robert\_NNP(y)} \implies \texttt{man\_NN(y)} \tag{6}$$

whrere y is a bound variable like x in (4).

Having such a rule in a KB means that we can implicitly admit additional clauses having `man_NN(y)` as argument instead of `Robert_NNP(y)`.

The same expression, of course, in a conceptual domain can also be rewritten as a composite fact, where `Robert_NNP(x)` becomes argument of `man_NN(x)` as it follows:

$$\texttt{man\_NN(Robert\_NNP(y))} \tag{7}$$

which agrees with the hierarchy of 3 as outcome of the Definite Clauses Builder.

As claimed in [28], not every KB can be converted into a set of definite clauses, because of the single-positive-literal restriction, but many KB can, like the one related to this work for the following reasons:

1. No clauses made of one single literal will ever be negative, due to the closed world assumption. Negations, initially treated like whatever adverb, when detected and related to *ROOT* dependency are considered as polarity inverter of verbal phrases; so, in this case, the *assert* will be turned into *retract*.
2. When the right hand-side of a clause is made by more than one literals, it is easy to demonstrate that, by applying the implication elimination rule and the principle of distributivity of $\vee$ over $\wedge$, a non-definite clause can be splitted into n definite clauses (where n is the number of consequent literals).

4. Nested Reasoning and Clause Conceptual Generalizations

The aim of the Cognitive Reasoner is to query a KB made of nested clauses that are also made closer to any possible related query, thanks to an appropriate pre-processing at assertion-time. Such a pre-processing, which creates a runtime *expansion* of the KB for every asserted clause, takes advantage of assignment rules for derivation of new knowledge.

The Backward-Chaining algorithm, as is, in presence of clauses where argument manipulation is required, might not be effective. To achieve such a goal, when required clauses are not present in the KB but deductible by proper arguments substitutions, the clauses evaluations at reasoning-time can be quite heavy and not feasible in term of complexity, because the process requires unifications at every single step. Instead, we will show how, by expanding properly the KB at assertion-time, the reasoning itself can be achieved acceptably. In order to obtain such a goal, CASPAR extends the radius of the nominal Backward-Chaining through the expansion of the Clauses KB with new knowledge generated starting from arguments substitutions on copies of specific clauses already asserted before.

For instance, let us consider a KB made at most of one-level[4] composite predicates as follows:

[4]Supposing a zero-level composite predicate be `P(x)`.

$$
\begin{gathered}
P_1(G_1(x_1)) \land P_2(G_2(x_2)) \implies P_3(F_3(x_3)) \\
P_1(F_1(x_1)) \\
P_2(F_2(x_2)) \\
F_1(x) \implies G_1(x) \\
F_2(x) \implies G_2(x) \\
H_3(x) \implies F_3(x)
\end{gathered}
$$

Querying such a KB with $P_3(H_3(x))$, for instance, using the Backward-Chaining algorithm, it will return *False* because there are neither any unifiable literals present nor as consequent of a clause. Instead, by exploiting $H_3(x) \implies F_3(x)$, we can also query the KB with $P_3(F_3(x))$ which is present as consequent of the first clause and it is surely satisfied together with $P_3(H_3(x))$: that's what we define as *Nested Reasoning*.

Now, to continue the reasoning process, we should check about the premises of such clause, which is made of the conjunction of two literals, namely $P_1(G_1(x_1))$ and $P_2(G_2(x_2))$. The latters, although not initially asserted, can be obtained starting by argument substitution on copies of other clauses from the same KB. Such a process is achieved by implicitly asserting the following clauses together with $P_1(F_1(x_1))$ and $P_2(F_2(x_2))$:

$$
\begin{gathered}
P_1(F_1(x_1)) \implies P_1(G_1(x_1)) \\
P_1(F_1(x_1)) \implies P_2(G_2(x_2))
\end{gathered}
$$

Since we cannot know in advance what a future successful reasoning requires, considering all possible nesting levels, along with the previous clauses also the so-called *Clause Conceptual Generalizations* will be asserted:

$$
\begin{gathered}
P_1(G_1(x_1)) \land P_2(G_2(x_2)) \implies F_3(x_3) \\
F_1(x_1) \\
F_2(x_1)
\end{gathered}
$$

where the antecedent of the implication is unchanged to hold the quality of the rule, while $F_1(x_1)$, $F_2(x_1)$, $F_3(x_3)$, as satisfiability contributors of respectively $P_1(F_1(x_1))$, $P_2(F_2(x_2))$, $P_3(F_3(x_3))$, are assumed asserted together with the latters. In other terms, the predicates: P_1, P_2, P_3 can be considered as *modifiers* of respectively F_1, F_2, F_3.

A generalization considering also the antecedent of the implicative formula is possible only through a weaker assertion of the entire formula itself, by changing $\implies$ with $\land$ as it follows:

$$\exists\ x_1,\ x_2,\ x_3 \mid G_1(x_1) \land G_2(x_2) \land F_3(x_3)$$

which is not admitted as definite clause, being not a single positive literal. In any case, the mutual existence of x_1, x_2, x_3 which satisfies such a conjunction, is already subsumed by the implication.

After such a theoretic premise, let's make a more practical example considering the following natural language utterance:

When the sun shines hard, Barbara drinks slowly a fresh lemonade

Table 1
Clause Generalizations Constituents Table (A=Applied, NA=Not Applied)

Hard	Slowly	Fresh
A	NA	NA
A	A	NA
A	NA	A
A	A	A

The corresponding definite clause will be (omitting the POS tags for the sake of readability):

$$\texttt{Hard(Shine(Sun(x1), \_\_))} \implies \texttt{Slowly(Drink(Barbara(x3),}\ \texttt{Fresh(Lemonade(x4))))}$$

Considering as *modifiers* adjectives, adverbs and prepositions, following the schema in Table 1: all the clauses generalization (corresponding to the first three rows of the table, while the forth is the initial clause) can be asserted as it follows:

$$\texttt{Hard(Shine(Sun(x1), \_\_))} \implies \texttt{Drink(Barbara(x3), Lemonade(x4)))}$$
$$\texttt{Hard(Shine(Sun(x1), \_\_))} \implies \texttt{Slowly(Drink(Barbara(x3), Lemonade(x4)))}$$
$$\texttt{Hard(Shine(Sun(x1), \_\_))} \implies \texttt{Drink(Barbara(x3), Fresh(Lemonade(x4)))}$$

As said before, the antecedent (whether existing) of all generalizations remains unchanged to hold the quality of the triggering condition, while the consequent shape will range on all possible variations of its modifiers, which will be 2^n with n as number of modifiers. Here the adverb *Hard*, being common part of all the antecedents composition, is always *Applied*.

Although in such a case the number of generalizations is equal to 4, in general it might be quite higher: it has been observed, after an analysis of more text corpus from the Stanford Question Answering Dataset[29], that the average number of modifiers in a single non-implicative utterance is equal to 6. In such cases the number of generalizations would be equal to 64, but greater numbers of modifiers would make the parsing less tractable, considering also arguments analysis for possible substitutions. In order to limit such a phenomenon, depending on the domain, CASPAR gives the chance to limit the number of generalizations by a specific parameter which modifies the policies of selective inclusion/exclusion of modifiers categories (adjectives, adverbs or prepositions).

In such a scenario, of course, the more the combinatorial possibilities, the more the number of clauses in the Clauses KB. It will appear clear, for the reader, that this approach sacrifices space for a lighter reasoning, but we rely on a three distinct points in favor of our choice:

1. An efficient indexing policy of the Clauses KB, for a fast retriving of any clause.
2. The usage of the class *Sensor* of Phidias for every clauses assertion, which works asynchronously with respect to the main production rules system, will make the agent immediately available after every interrogation without any latency, while additional clauses will be asserted in background.

3. We point to keep the Clauses KB as small as possible, in order to limit the combinatorial chances. In this paper we assume the assignment rules properly chosen among the most likely which can get the query closer to a proper candidate. As future works, a reasonable balancing between two distinct Clauses KB working on different levels might be a good solution: in the lower level (long-term memory) only clauses pertinent with the query will be searched, then put in the higher one (short-term memory) for attempting a successful reasoning. Similar approaches have been used with interesting outcomes in some of the widespread Cognitive Architectures[8].

As result evaluation, we consider a slightly rephrased KB (*Colonel West*) treated in [28], showing how CASPAR is able to make a successful reasoning for a question requiring a non-trivial deduction. Although this architecture is designed to work as vocal assistant, one can alike verify the reasoning by asserting manually the same belief STT asserted by the Sensor Instance as shown in Listing 2 in the Appendix. In light of this, after each assertion (lines 1, 8, 13, 18, 30) the new asserted clauses are shown, and it appears clear how the agent *expands* the Clauses KB considering generalizations and argument substitutions. After the query is given (line 45), is shown how the nominal Backward-Chaining algorithm is not enough for achieving a successful reasoning, while it happens using the Nested Reasoning.

In Section 3.3 we have also shown how a direct command or routine can be subordinated by a clause. Although in the example (see lines 12-13 of Listing 1 in the Appendix) the production rule contains the representation of *An inhabitant is at home*, even a clause involving the Nested Reasoning might trigger such a rule; for instance, a simple toy scenario could include a facial recognizer among the domotic devices, which obtains information about known/unknown faces when someone is detected in the environment. Such a recognition could generate a clause representing (for instance) *Robert is at home*, which, combined with another clause representing *Robert is an inhabitant*, will produce the representation of *An inhabitant is at home*; the latter will trigger the production rule (related to a direct command or routine) that will turn off the alarm in the garage. This will not happen whether a thief or a domestic animal is detected, thus it provides indeed a valid example about how Beliefs KB and Clauses KB interact with each other, in a non-trivial process of deduction.

5. Conclusions and Future Work

In this paper, we have presented the design of a cognitive architecture called CASPAR able to implements agents capable of both reactive and cognitive reasoning. Nevertheless we want to mark a way towards a comprehensive strategy to make deduction on Knowledge Bases whose content is parsed directly from natural language. This architecture works by using a Knowledge Base divided into two distinct parts (Beliefs KB and Clauses KB), which can also interact with each other in decision-making processes. In particular, as long as the Clauses KB increases, its cognitive features improve due to an implicit and native capability of inferring combinatorial rules from its own Knowledge Base. Thanks to the Nested Reasoning and the Clause Conceptual Generalizations, CASPAR is able to transcend the limit of the known Backward-Chaining algorithm due to the nested semantic notation; the latter is as highly descriptive as compact. Furthermore, agents based on such an architecture are able to parse complex direct IoT commands and routines, letting the user customize with ease his own Smart Environment Interface and Sensors, with whatever Speech-to-Text engine.

As future works, we want to test CASPAR capabilites with other languages than english and evaluate other integrations, like Abductive Reasoning and Argumentations. Even chatbots applications can take advantage of this architecture's features.

Finally, we want to exploit Phidias multiagent features by implementing standardized communication protocols between agents and exploit other ontologies as well.

References

[1] V. Këpuska, G. Bohouta, Next-generation of virtual personal assistants (Microsoft Cortana, Apple Siri, Amazon Alexa and Google Home), in: 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), 2018, pp. 99–103.

[2] H. Jeon, H. R. Oh, I. Hwang, J. Kim, An Intelligent Dialogue Agent for the IoT Home, in: AAAI Workshops, 2016. URL: https://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12596.

[3] E. V. Polyakov, M. S. Mazhanov, A. Y. Rolich, L. S. Voskov, M. V. Kachalova, S. V. Polyakov, Investigation and development of the intelligent voice assistant for the Internet of Things using machine learning, in: 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT), 2018, pp. 1–5.

[4] M. Mehrabani, S. Bangalore, B. Stern, Personalized speech recognition for Internet of Things, in: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015, pp. 369–374.

[5] R. Kar, R. Haldar, Applying Chatbots to the Internet of Things: Opportunities and Architectural Elements, International Journal of Advanced Computer Science and Applications 7 (2016). URL: http://dx.doi.org/10.14569/IJACSA.2016.071119. doi:10.14569/IJACSA.2016.071119.

[6] C. J. Baby, F. A. Khan, J. N. Swathi, Home automation using IoT and a chatbot using natural language processing, in: 2017 Innovations in Power and Advanced Computing Technologies (i-PACT), 2017, pp. 1–6.

[7] I. Kotseruba, J. K. Tsotsos, 40 years of cognitive architectures: core cognitive abilities and practical applications, Artificial Intelligence Review (2018) Rev 53, 17–94 (2020). doi:https://doi.org/10.1007/s10462-018-9646-y.

[8] D. F. Lucentini, R. R. Gudwin, A comparison among cognitive architectures: A theoretical analysis, in: 2015 Annual International Conference on Biologically Inspired Cognitive Architectures, 2015.

[9] T. Giulio, Consciousness as integrated information: A provisional manifesto, The Biological bulletin (2008) 215(3), 216–242.

[10] S. Epstein, R. Passonneau, J. Gordon, T. Ligorio, The role of knowledge and certainty in understanding for dialogue, in: AAAI Fall Symposium Series, 2011. URL: https://www.aaai.org/ocs/index.php/FSS/FSS11/paper/view/4179.

[11] V. Këpuska, G. Bohouta, Comparing speech recognition systems (microsoft api, google api and cmu sphinx), Int. Journal of Engineering Research and Application Vol. 7, Issue 3, (Part -2) (March 2017) 20–24.

[12] M. Dehghani, E. Tomai, K. Forbus, M. Klenk, An integrated reasoning approach to moral decision-making, in: Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3, AAAI'08, AAAI Press, 2008, p. 1280–1286.

[13] M. Scheutz, P. Schermerhorn, J. Kramer, D. Anderson, First Steps toward Natural Human-like HRI, Auton. Robots 22 (2007) 411–423. URL: https://doi.org/10.1007/s10514-006-9018-3. doi:10.1007/s10514-006-9018-3.

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

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

[16] L. D. Consortium, Treebank-3, 2017. URL: https://catalog.ldc.upenn.edu/LDC99T42.

[17] X. Huang, L. Deng, An Overview of Modern Speech Recognitiong, Microsoft Corporation, 2009, pp. 339–344.

[18] R. Rajan Mehla, Mamta, Automatic speech recognition: A survey, International Journal of Advanced Research in Computer Science and Electronics Engineering Volume 3, Issue 1 (January 20147) 20–24.

[19] A. D. Saliha Benkerzaz, Youssef Elmir, A study on automatic speech recognition, Journal of Information Technology Review Volume 10, Number 3 (August 2019).

[20] 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.

[21] S. Anthony, J. Patrick, Dependency based logical form transformations, in: SENSEVAL-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text, 2015.

[22] ClearNLP, Clear nlp tagset, 2015. URL: https://github.com/clir/clearnlp-guidelines.

[23] B. H. Partee, Lexical Semantics and Compositionality, volume 1, Lila R. Gleitman and Mark Liberman editors, 1995, p. 311–360.

[24] 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.

[25] 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.

[26] L. Fichera, F. Messina, G. Pappalardo, C. Santoro, A python framework for programming autonomous robots using a declarative approach, Sci. Comput. Program. 139 (2017) 36–55. URL: https://doi.org/10.1016/j.scico.2017.01.003. doi:10.1016/j.scico.2017.01.003.

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

[28] P. N. Stuart J. Russel, Artificial Intelligence: A Modern Approach, Pearson, 2010.

[29] stanford, The stanford question answering dataset squad2.0, 2018. URL: https://rajpurkar.github.io/SQuAD-explorer/.

In this appendix, a simple instance of Smart Environment Interface is provided (Listing 1) together with an example of how Clauses Knowledge Base changes, after assertions (Listing 2).

```
# Routine conditionals management
+SENSOR(V, X, Y) >> [check_conds()]
check_conds() / (SENSOR(V, X, Y) & COND(I, V, X, Y) & ROUTINE(I, K, J, L, T)) >> [-COND(I, V, X, Y), +START_ROUTINE(I), check_conds()]
check_conds() / SENSOR(V, X, Y) >> [-SENSOR(V, X, Y)]

# Routines execution
+START_ROUTINE(I) / (COND(I, V, X, Y) & ROUTINE(I, K, J, L, T)) >> [show_line("routine not ready!")]
+START_ROUTINE(I) / ROUTINE(I, K, J, L, T) >> [-ROUTINE(I, K, J, L, T), +INTENT(K, J, L, T), +START_ROUTINE(I)]

# turn off
+INTENT(X, "light", "kitchen", T) / lemma_in_syn(X, "change_state.v.01") >> [exec_cmd("change_state.v.01", "light", "kitchen", T)]
+INTENT(X, "alarm", "garage", T) / (lemma_in_syn(X, "change_state.v.01") &
    eval_cls("At_IN(Be_VBP(Inhabitant_NN(x1), __), Home_NN(x2))")) >> [exec("change_state.v.01", "alarm", "garage", T)]

# any other commands
+INTENT(V, X, L, T) >> [show_line("Result: failed to execute the command: ", V)]
```

Listing 1: A simple instance of Smart Environment Interface

```
> +STT("Nono is an hostile nation")

Be(Nono(x1), Nation(x2))
Be(Nono(x1), Hostile(Nation(x2)))
Nono(x) ==> Nation(x)
Nono(x) ==> Hostile(Nation(x))

> +STT("Colonel West is American")

Be(Colonel_West(x1), American(x2))
Colonel_West(x) ==> American(x))

> +STT("missiles are weapons")

Be(Missile(x1), Weapon(x2))
Missile(x) ==> Weapon(x)

> +STT("Colonel West sells missiles to Nono")

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))

>+STT("When an American sells weapons to a hostile nation, that American is a criminal")

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

>+STT("reason")

Waiting for query...

> +STT("Colonel West is a criminal")

Reasoning..............

Query: Be_VBZ(Colonel_West(x1), Criminal(x2))

 ---- NOMINAL REASONING ---

Result: False

 ---- NESTED REASONING ---

Result:  {v_211: v_121, v_212: x2, v_272: v_208, v_273: v_209, v_274: v_210, v_358: v_269, v_359: v_270, v_360: v_271}
```

Listing 2: CASPAR Clauses Knowledge Base changes and reasoning, after assertions