University of Catania

Department of Matemathics and Computer Sciences

XXXIV Ph.D. in Computer Science (International)

Carmelo Fabio Longo

A Family of Reactive-Cognitive Architectures based on Natural Language Processing, as Decision-Making Helpers for IoT, in the Closed- or Open-World Assumption

Doctoral Thesis

University advisor: Prof. Corrado Santoro

Academic Period 2018 - 2021

"I've seen things you people wouldn't believe. Attack ships on fire off the shoulder of Orion. I watched C-beams glitter in the dark near the Tannhauser gate. All those moments will be lost in time, like tears in rain..."

Roy Batty, Blade Runner

Abstract

This thesis investigates and proposes different solutions in the scope of cognitive architectures leveraging natural language processing. Although the number of existing cognitive architectures has reached a significant number (several hundred), many of them have been mainly used as research tools so far, and none of them has been specifically designed for the Internet of Things. On the other hand, nowadays companies producing vocal assistants aim more at increasing their pervasiveness than at improving their native reasoning, due to their inability of combining facts with rules in order to infer new knowledge and help the user in decision-making tasks. Such a motivations led to the design of CASPAR, a novel architecture for instantiating *cognitive agents* based of natural language. Such agents are not based on clouds and do not require any semantic training, plus they are able of deduction on facts and axioms in first-order logic inferred directly from natural language. A case-study is provided to show the effectiveness of the approach, in cases of direct commands and routines subordinated also by a meta-reasoning in a conceptual space, by parsing utterances with promising real-time performances. The lack of CASPAR to obtain results in the presence of non-unifying clauses and to manage large knowledge bases as well, led afterwards to the design of AD-CASPAR, capable of both abduction as pre-stage of deduction and of being also an interesting tool for instantiating *thinking* Telegram chatbots. Finally, the chance to make CASPAR suitable for different scenarios in the open-world assumption, led to the design of SW-CASPAR, which is also a valid tool for the task of learning ontologies serialized in OWL 2. Such ontologies respond to the specifications of LODO, which is a fondational ontology aiming to fill the gap between natural language and human-like fashioned reasoning.

Acknowledgements

My acknowledgement, first of all, goes to my Ph.D. advisor and friend, Prof. Corrado Santoro, who supported me even after eighteen years elapsed from my degree in Computer Science. Under his guide, what I previously saw far and almost unreachable, became feasible and close at hand.

I would like also to express my gratitude to Prof. Giovanni Gallo, who made it possible the collaboration between the University of Catania and the previous company I worked for, in order to develop applications for the Pepper robots and start my journey towards the Ph.D course.

I would like to manifest my gratitude to Prof. Fabrizio Messina, who involved me often in academic experiences and he was always ready to assist me in whatever I needed, even for a coffee.

I would like to thank also Prof. Daniele Francesco Santamaria, Prof. Marianna Nicolosi Asmundo and Prof. Domenico Cantone, for their contribution to my knowledge of the Semantic Web.

I want to thank my friend Prof. Ilaria Frana, for her support concerning linguistic science.

I would kindly thank also Prof. Valeria Seidita and Prof. Giuseppe Vizzari, who have accepted being reviewers of this thesis and are part of the friendly community of the WOA[1], which I have begun to know even before the Ph.D course.

Finally, I want to thank myself for not doubting my chances and accomplish such a academic experience, showing that it is never too late to grow and improve one's condition through study and will.

[1]**W**orking **O**n **A**gents workshops.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation reports the research activities carried on during the Ph.D. program in Computer Science I attended at the University of Catania - Departments of Mathematics and Computer Science. My research area is mainly concerned on agents leveraging natural language processing, in order to fulfill deductive tasks and execute related plans. Possible applications spans from Robotics to Internet of Things, or whatever other scenario where parsing from natural language to logical forms is required to perform deductive/reactive operations.

The motivation to begin the journey of such a Ph.D. course was born with my past experience with the robot humanoid Pepper[1] and also with my passion for Philip K. Dick literature. I clearly remember the exciting moment in which I took two Peppers out of the boxes, in the company where I worked before the Ph.D. course. I gave the robots the names *Roy* and *Pris*, like the two charming androids from Philip K. Dick's 1968 novel *Do Androids Dream of Electric Sheep?*[2]. As I learned the interesting mobility of the two robots, thanks to their great number of junctions, sensors and interfacing features, I realized also they were just going to be a couple of complex toys for the price[3] of a subcompact car, unless they acquired some kind on human-like deductive capability. Pepper is sold together with Choregraphe, a visual programming software developed by SoftBank Robotics, known for its interactive interface and ease of use. But the chance given by Choregraph of creating even complex behaviour routine, involving movement, image recognition and sounds, is far away from what I meant above as *human-like deductive capabilities*.

[1]https://www.softbankrobotics.com/emea/en/pepper

[2]The book served as the primary basis for the 1982 *Blade Runner* movie.

[3]Now on sale for €9900.

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. In light of this, as we can see in detail in the next chapters, I can affirm that the framework I designed, namely CASPAR, can be considered cognitive architecture as well. Such a statement complies also with Thagard[1] definition of cognitive architecture: "a general proposal about the representations and processes that produce intelligent thought.". Nonetheless, CASPAR has been first presented in a workshop[2] on agents, then published in a good impact factor journal[3], while its two evolutions AD-CASPAR and SW-CASPAR were presented in international conferences[4, 5] as well, which constitutes a solid proof about their contribution to the state-of-the-art.

Considering the limitations of nowadays artificial intelligence systems, I wanted to investigate deeply the scientific literature, in order to find out actually how far the reality is from the imagination. After that, I pursued the road of natural language processing as gateway to the *thought-shape* in human-like fashion, in order to put an agent in condition of creating new knowledge starting from an ontology made of facts and rules, then fulfilling the task of decision-making helper with reactive interaction with environment. This activity led first to the birth of CASPAR, a scalable architecture able to instantiate intelligent agents based on natural language processing, then AD-CASPAR and SW-CASPAR. The former is able of abduction as pre-stage of deduction; the latter is focused on reasoning over the Semantic Web. The common core of these three architectures is a production rule system leveraging semantic dependencies, in order to create meta-structures that permit to infer logical forms with ease. Such operations are carried out by specific modules embedded in a *Belief-Desire-Intention*[6] framework, whose design got inspiration from the Bratman[7] theory on practical human reasoning.

In Section 1.1 of this introduction, after a brief overview about the historical motivations and features of the first cognitive architectures, it will be anticipated in a nutshell what actually the contribution of CASPAR is. Such a contribution will be made more explicit in the rest of the thesis; Section 1.2 introduces briefly the implications occurring whether we consider *consciousness* as possible variable among others, in the measure of how much it could influence agent's behavior; Section 1.3 introduce the foundation of the deep linkage between language and cognition;

Section 1.4 concerns the issues of having a knowledge base (ontology) inferred from utterances in natural language, as arbitrary description of a domain given by a speaker, when such a knowledge base has to be used for reasoning purposes; finally, Section 1.5 provides a brief outline about the chapters ahead.

1.1 Artificial Cognition

In the scope of cognitive architectures, the first pioneers who wondered how the brain works attempted a reverse-engineer of such a complex biological system. Worth of mention is the Unified Theory of Cognition[8], with the purpose of providing comprehensive insights for the design of a cognitive system. Although interesting, it didn't convince widely researchers in the behaviour science, or maybe it was too ambitious. Instead, many of them focused on characterizing the mechanisms of small parts of the brain, down to single interactions between neuronal cells.

There are several impressive examples of large-scale, single-cell neural simulations; among the best known there is Izhikevich's simulation[9], which is on the scale of human cortex, running approximately 100 billion neurons. Modha's Cognitive Computation project[10] at Almeda Labs is on the scale of cat cortex (1 billion neurons) but is faster than Izhikevich's simulations, although both share largely random connectivity in their networks. Markram's simulation[11] takes in account of approximately 1 million neurons with a deeper biological detail than respect to other simulations, reflecting accurately connectivity, synaptic and neural dynamics. Despite a growing interest in these simulations, none of these past models have demonstrated perceptual, cognitive of motor functions. According to the authors of this [12] survey, focusing on neuronal-cell interactions make it unclear what kinds of inputs the model should have and how to interpret its output.

In 1963 Newell and Simon presented a program which they called *General Problem Solver* (GPS)[13], which was the first in a line of explanations of human cognitive performance relying on production systems. Historically, production systems has been the most influential approach towards cognitive systems building. The general interest in GPS led to the development of other cognitive architectures, all of which with production systems as their core: Soar[14], EPIC[15] and ACT[16] were among the best known. Although their early success, such architectures was not suitable to interact with fast environments with difficult-to-predict dynamics of the world. As a result, in Robotic, in order to control low-level behaviour of robots, differential equations, statistics and signal processing are preferred rather than production systems. On the other hand, it remain unclear how to use these mathematical method for characterizing high-level cognitive behaviour, like language, complex planning, deductive reasoning, etc. Hence, the presence of a gap in our understanding of real cognitive systems appears evident.

There are many other cognitive architecture implementing human-like behaviors, beyond the ones cited above: according to this [17] recent survey, the number of existing cognitive architectures has reached several hundred, but most of the existing surveys do not reflect this growth and instead focus on a handful of well-established solutions. In such a plethora, the aim of CASPAR, as we will see in Chapter 2, is to fill the above gap making usage of meta-reasoning, in order to subordinate fast-dynamic real-world perception and actions with high-level cognition encoded in a conceptual space.

Another important distinction among artificial cognitive systems is between "natural/cognitive/biological" inspired systems and "machine" oriented systems. The former tend to imitate (at different levels of abstraction) the behaviour of systems existing in nature. While machine oriented ones, without taking any inspiration from nature, tackle computational challenges posed by the problem itself. Worth of mentioning is what reported in [18], where authors state that "the quest for *artificial flight* succeeded when the Wright brothers and others stopped imitating birds and started using wind tunnels and learning about aerodynamics". This example, also highlighted by Lieto in his recent book[19], is relevant to realize that "functional resemblance" in term of generated output (i.e. the ability to fly in this case) between organism and machines, is not sufficient for explaining that function through a model capable of reproducing the essential features of that organism. Still Lieto, in his book, proposes the *Minimal Cognitive Grid*, a pragmatic methodological tool to rank the different degrees of structural accuracy of artificial systems in order to project and predict their explanatory power. Such a metric will be used in Chapter 5 to evaluate all three novel cognitive architectures introduced in this dissertation.

Another example where full functional resemblance is different than respect what was experimental achieved, although inspired, is the technique called *Back-Propagation*, which is the keystone of the deep learning revolution and sub-symbolic approaches. It is known that bio-electric information flow in humans brain's neurons is unidirectional and without any backward interaction, unlike in the design of modern artificial neural networks. Drawbacks and trade-off between the usage of symbolic and sub-symbolic approaches are widely addressed here [20]. Regarding to the architectures described in this thesis, as it will shown ahead, they take advantages of both symbolic and sub-symbolic approaches, because one of the adopted component integrates a neural dependency parser, whose outcomes are afterward parsed with a production rule system; therefore, in a certain sense, CASPAR and its derivations can be considered hybrids solutions, floating between such two main widespread approaches (sub-symbolic and symbolic).

1.2 The Issue of Consciousness

In a research activity concerning cognitive architectures, each of which reflects some aspect of human cognitive capabilities, for a comprehensive analysis it does make sense to consider the foggier, but which might influence the human behaviour as well: the consciousness, also known as *self*. But before going further, what definition to provide for such a word, in the scope of Artificial Intelligence? Instead of providing whatever kind of definition of consciousness, one could refuses utterly the idea of *self* as mental substance and consider it a mere illusion. According to Metzinger[21], through our experience we develop models of the *self*, which are just information processes of the brain. However, since we have not free access to these mental processes, we tend to presuppose the presence of an entity at the base of our own model of the *self*. Thus, such entity is characterized as the *self*. From such a point of view, the consciousness would exist only when we ask ourselves about it. On the other hand, if we don't want to refuse the idea of a *self* and keep arguing about it, together with its definition, since this is an exact science thesis, we could identify the neural correlates of consciousness and quantify them, forcing us abandoning philosophical theories to delve into neuroscience.

James[22] affirms that a conscious system in human-like fashion should have a kind of awareness about both environment and its internal status, together with memory and executive functions. Intuitively we can deduce also that the most such features the most the system's deductive capabilities should increase as well, in a human-like fashion; but the thing is that, while it has been easy to deduce that such features are necessary conditions for a consciousness, no one has yet provided strong proofs concerning sufficient ones. A comprehensive explaining of consciousness would require solving what Chalmers[23] calls the *hard problem of consciousness*. According to Chalmers, the easy problem of consciousness is explaining *how* the brain generates the behavior associated with consciousness. In contrast, the hard problem requires a theory to address the question of *why* any physical process generates (or is) consciousness.

The widely discussed Tononi's Integrated Information Theory (IIT)[24] attempts to identify the essential properties of consciousness (axioms) and, from there, infers the properties of physical systems that can account for it (postulates). Based on the postulates, it permits in principle to derive, for any particular system of elements in a state, whether it has consciousness, how much, and which particular experience it is having. IIT postulates that consciousness is equal to integrated information Φ, which is the total amount of information that is exchanged (integrated) between distinct parts of the brain. The author of [25] argues that IIT is just a theory of protoconsciousness, which fails in its stated goal of quantifying consciousness and it is vulnerable to one of the strongest arguments in philosophy of mind: the Chalmers Principle of Organizational Invariance[23, 26]. Differently from IIT, which attributes an emergent consciousness to a system as long as its distinct parts exchanges data, the Chalmers Principle start from two thought-experiments, namely *fading/dancing qualia*, addressing the question about what sort of physical systems can give rise to conscious experience. His suggestion is that when experience arises from a physical system, it does so in virtue of the system's functional organization.

Among detractors of IIT there is also Aaronson[27], who argued that something as uncomplicated as a network of XOR gates arranged in an n x n square grid could generate $\sqrt{n}\ \Phi$. By expanding the size of n, a network of XOR gates could be created with arbitrarily large values of Φ. Nonetheless, Aaronson's vision of consciousness is closer to the type of subjective experience humans and animals when awake, which guides their behavior and which they lack when asleep or anesthetized; thus, it is clear that he is not using the word consciousness in the same way as Tononi. It seems IIT being a theory of a much more general form of subjective experience and, hence, measures protoconsciousness rather than consciousness. Rosenberg[28] suggests protoconscious experiences have no mind associated with them and that “the experiences we might attribute to noncognitive systems do not contain ‘little pains’ or ‘little specks of blue’ but instead have some kind of qualitative character very alien to us”.

In light of above, as far as we know, no one has yet the magic recipe for creating a non-simulated artificial consciousness, unless being a chatbot with the aim to pass the well-known Turing test [29], i.e., to fool an interlocutor about who is talking with. Therefore I agree with [30] and other detractors that it cannot be used as test base to evaluate *conscious* intelligent systems. The most important objection is because it only refers to the manifest behaviour of a given system, while no claim can be made about the internal mechanism that have led to that behaviour. Another objection is about its excessive anthropocentrism, because it targets explicit human-like thinking, so cannot be used to provide a universal criterion for attributing intelligence. A further problem concerns also the subjective evaluation of the interrogator, since different human interrogators, indeed, might judge in a different way the same behaviour.

Summing up, after such a brief overview, in my humble opinion a *third person* description of a phenomenon related to a strictly *first person* point of view is not feasible, for the same reason that only lovers should talk about love: by an outside observer who is not a lover, love can only be seen as something that either positively or negatively affects mood and consequently behavior; therefore, without any chance and will of detecting the subjective-qualitative aspects of such mental processes. What remains is somehow a step backwards in cognitive studies as in the early days of psychology, when most theories of the mind were generated by the method of introspection, that is sitting and thinking very carefully, so as to discern the components of cognitive processing. But in those times, different people introspected different things, so there was a crisis in “introspectionist” psychology. This crisis was resolved by “behaviorist” psychologists who simply disallowed introspection. The only relevant data for understanding cognitive systems were data that could be inferred from the “outside” (that is, from behavior).

Hence, at this point it make sense to introduce the concept of *relative consciousness*, that is more related to the level of *self* which *appears* to the evidence of an observer. Moreover, a test to evaluate such a relative consciousness in a cognitive system, should measure its deductive/abductive capabilities and the consequent level of (auto)awareness as well. That's what I am interested in more, in the scope of this thesis. The relative consciousness could be measured also in human beings, putting in the middle of the interest just the external effects of their *supposing feelings* and their relation with the environment, which is an alternative and more constructive way to endorse Metzinger theory. Furthermore, such a vision of consciousness it is strictly related to the system's functional organization, being a direct consequence of the latter, therefore it doesn't not contrast the above cited Principle of Organizational Invariance.

In light of the above, my personal insight about consciousness is that it might *emerge* from a structural chaos made of unrelated with each other information, on the basis of needs aimed to avoid uncomfortable conditions. Nevertheless, such pieces of information will take their part in a functional organization made of specific *patterns*, whom can be formalized in production rules, in a sort of *reinforcement learning* aimed to avoid already seen uncomfortable scenarios, by favoring instead those leading to satisfactory outcomes. Such kind of reactive learning will produce also an auto-awareness leading to a semi-continuous verification of being in a optimal general (or particular one) condition. In a hyperbole of humanistic achieved knowledge, such a process will lead an agent potentially to enquire himself also about consciousness itself, similarly to any sentient being, while the issue would be otherwise discarded in order to privilege more instinctive drives.

1.3 The role of Language in Cognitive Science

In this subsection it will briefly highlighted the relations between language and conscious experience as mental process. The psychologist Euan Macphail[31] affirms that language and self awareness are necessary for consciousness; thus, both newborns and animals should be involved in such a lack. Kock[32] argues strongly against this, as IIT endorser, but starting from the arbitrary assumption that experience and consciousness are the same thing, embracing somehow the thesis of panexperientialism[4]. Beyond that, there are also proofs about which the ability of language is not necessary for conscious experience: worth of mention are aphasia[5] cases[33, 34] of patients, in a temporary inability of language usage, which kept making experience of all feelings linked to incoming sensory stimulus. On the other hand, during a human being growing, in normal conditions the language assumes a fundamental role in order to encode in semantic structures what otherwise would be cloudy feelings mapped in some region of the brain. In particular, the language contributes deeply in the way we do experience of the world and in our sense of *self* as narrative center of past and present.

Jaynes[35] provides a palentology of consciousness, where the language assume a key role in both human beings mnemonic ability and the acquired skill of narrativize memories in definite structure. According to the author, such a narrativization was born as encoding of past event relations and epic poems. Before that, the usage of written language was primarily a tool for compilation of inventories, record stocks and property exchanges.

In a much-quote passage, Whorf[36] wrote: "We dissect nature along lines laid down by our native language. The categories and types that we isolate from the world of phenomena we do not find there because they stare every observer in the face; on the contrary, the world is presented in a kaleidoscopic flux of impressions which has to be organized by our minds - and this means largely by the linguistic system of our minds. We cut nature up, organize it into concepts, and ascribes significances as we do, largely because we are party to an agreements to organize it in this way - an agreement that holds throughout our speech community and is codified in the pattern of our language. The agreement is, of course, an implicit and unstated one, *but its terms are absolutely obligatory*; we cannot talk at all except by subscribing to the organization and classification of data which the agreement decrees."

In light of above, we can consider language as the more external substrate of cognitive information useful to retrace backwards an inferred meaning of thoughs and concept. That's because for the novel cognitive architectures introduced in this thesis, instead of a reverse-engineering of the brain, I chose to start from words

[4]The Panexperientialism claims that everything has experience.

[5]Language disorder given by a limited cerebral damage, usually but not always in the left cortical hemisphere.

as bricks to builds concepts from utterances, by leveraging also ontologies to acquire commons sense knowledge of known semantic structures. The drawback in such a process is that concepts inferred from natural language might be arbitrarily interpreted and are subject to ambiguities. For instance, the utterance *Roy goes to the red room with a light* can be interpreted in at least two distinct ways: one interpretation would be *Roy goes to the red room whose light is turned on*, while another interpretation is *Roy goes to the red room while holding a light*. In the next subsection such biases are discussed, for the task of ontology learning from natural language.

1.4 The Issue of Natural Language Ontology

The task of modelling an ontology reflecting a domain, by the means of a description in natural language of the domain itself, implies several bias that must be considered. Ontology learning from natural language can be accomplished in three ways: manual modelling, leveraging the effort of experts of the domain; cooperative modelling, where most or all the task are supervised by experts; semi-automatic modelling, where the ontology construction process is performed automatically with limited intervention by users or experts. According the authors of [37], full automatic modelling by a system is still a significant challenge and it is not likely to be possible. When sentences are not specifically well-formed, such a task can be quite hard, because of all possible semantic ambiguities of idioms and their arbitrary descriptive nature of the world, which can induce morphologically distinct sequences of words to express the same concept.

The author of [38] reports that "Natural Language Ontology is a branch of both methaphysics and linguistic semantic. Its aims to uncover the ontological categories, notions and structures which are implicit in the use of natural language, that is, the ontology that a speaker accept when using a language". But such an acceptance implies several issues to address for an ontology learning system designer. For instance, for a speaker would be quite natural and simple to express that a certain object does not exits. In a *closed-world assumption* we can limit to not assert such a concept, or at least to retract the representation of it from a knowledge base. But how to operate in an *open-world assumption*, in order to let participate such information in a reasoning process in human-like fashion? How to keep consistency when an information and its complement are possibly present (like it could happen in text given by a speaker) both in an ontology? In linguistic science, intentional object as *nonexistent* are considered particularly problematic[39]; for instance, given an ontology A representing a domain of existing entities and another ontology B representing a description in natural language of the same domain: we cannot definitively say that A and B are equivalent, due to a possible introduction in B of entities which not exist but are functional to the arbitrarily descriptive use of words present in the source text of B.

Let us consider now the following two sentences: *Robert walked down the street* and *Robert had a walk down the street*; how to infer that they express the same concept in a decision process? In the second sentence we are in presence of the so-called *deverbal nominalization* of the verb *walk* versus a noun expressing the action of walking. So we are forced to create somehow a bridge between the two sentences, in order to achieve the same result when both are participating in a reasoning process. Similarly, let us consider also the simple verbal phrase: *the friends are happy* and the snipplet: *happy friends*. As for the former, we are in presence of a *deadjectival nominalization*, because the adjective *happy* become the object of a copular[6]verb (be), thus the subject (friends) has the property denoted by the adjective; so, how to express, by means of an ontology, that the two pieces of information lead the same concept?

Pinker[40] describes an hypothetical language *of the mind*, the so-called *mentalese*, which should stand above all ambiguities, universally homogeneous across languages. He affirms also that whatever language people speak is hopelessly unsuited to serve as our internal medium of computation, because of five different type of issues: *ambiguity*, *lack of logic precision*, *coreferencing*, *deixis* and *synonymy*. As for ambiguity, generally is when the same word might have different meaning depending on the context. In 2.3.4 it is shown how CASPAR addresses such a issue with its disambiguation strategy. As for lack of logical precision, let us consider the example devised by the computer scientist Drew McDermott:

[6]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.

Ralph is an elephant.
Elephants live in Africa.
Elephant have tusks.

In this case an inference-making device would deduce: *Ralph lives in Africa* and *Ralph have tusks*. The issue is that Africa is the common place where all elephants live in, but Ralph's tusks are his own and the device *doesn't* know that, because the distinction is nowhere to be found in any of the statements. The third issue called "co-reference"[7] concerns the resolution of pronouns referencing nouns in the scope of the same discourse. The state-of-the-art comprises several tools acting as coreferencers: one of these I have been testing is Neuralcoref[8], which can work also on different level of greediness depending on domain. The forth issue comes from those aspects of language that can only be interpreted in the context of a conversation or text - what linguistic call "deixis". Considering article like *a* and *the*, what is the difference between *killed a policeman* and *killed the policeman*? Only that in the second sentence it is assumed that some specific policeman was mentioned earlier or is salient in the context. Thus in isolation the two phrases are synonymous, but in the following context (the first from an actual newspaper article) their meaning are completely different:

A policeman's 14-year-old son, apparently enraged after being disciplined for a bad grade, opened fire from his house, *killing a policeman* and wounding three people before he was shot dead.

A policeman's 14-year-old son, apparently enraged after being disciplined for a bad grade, opened fire from his house, *killing the policeman* and wounding three people before he was shot dead.

Outside of a particular conversation or text, then, the words *a* and *the* are quite meaningless. The fifht issue, which is the synonymy, is when different sentences refer to the same event and therefore licence many of the same inferences, similarly

[7] Also called *anaphora resolution*.
[8] https://github.com/huggingface/neuralcoref

to deverbal and deajectival nominalization. For instance, considering thr following sentences:

Sam sprayed paint into the wall.
Sam sprayed the wall with paint.
Paint was sprayed into the wall by Sam.
The wall was sprayed with paint by Sam.

in all four cases one can conclude that wall has paint on it. But despite the four distinct arrangement of words mean the same thing, no simple processor would infer implicitly such a conclusion.

Those seen above and many other issues related to the natural language ontology, for the sake of shortness, are described here [38], which can be a good start point in order to fill the gap between a domain and an arbitrary description of it given by a speaker.

1.5 Thesis Outline

The content of Chapter 2 is the outcome of more of two years of study and experiments in the field of natural language processing and reasoning. The way that led me to the design of CASPAR, whose related article has been recently included in a issue of the journal *Engineering Applications of Artificial Intelligence* published by Elsevier, was full of non-trivial choices: the information representation, the reasoner, the container where to embed all the interacting modules, etc. The final result is a scalable framework for instantiating agents based on natural language processing, i.e., vocal assistants operating in Internet of Things scenarios. The scalability is meant for both descriptiveness of knowledge bases and for the interfacing features with the environment.

Chapter 3 is referred to the first of the two evolution of CASPAR, namely AD-CASPAR, presented in the *4th Workshop on Natural Language for Artificial Intelligence* co-located with the *19th International Conference of the Italian Association for Artificial Intelligence*. AD-CASPAR inherited all its predecessor features, plus the chance of abductive reasoning as pre-stage of deduction. This process, unlike deductive reasoning, yields one or more plausible conclusions but does not positively verify them, expressing also uncertainty levels. Furthermore, AD-CASPAR uses abductive outcomes as input to attempts deduction and the CASPAR's so-called *nested reasoning* described in Subsection 2.3.5. Such a strategy is useful in order to optimize processing in presence of large knowledge bases, focusing only to more relevant clauses and discarding the others.

Chapter 4 concerns the second evolution of CASPAR, namely SW-CASPAR, which has been presented in *22st Workshop "From Objects to Agents" (WOA 2021)* and selected for a special issue of the journal *Intelligenza Artificiale*[9]. Such a architecture inherits all its predecessor features, but with the difference that the meta-reasoning is achieved in the open-world assumption, by leveraging the Semantic Web.

Chapter 5 argues the overall evaluation of the CASPAR's cognitive architectures family with a pragmatic methodological tool to rank the different degrees of structural accuracy of artificial systems in order to project and predict their explanatory power.

Chapter 6 sums up the overall conclusions about strengths and weaknesses of the CASPAR's cognitive architectures family, plus a perspective about continuing such a research.

Finally, Chapter 7 is about all works published during my Ph.D. course, where I figure as an author.

[9]https://www.iospress.com/catalog/journals/intelligenza-artificiale

Chapter 2

CASPAR: Cognitive Architecture System Planned and Reactive

2.1 Introduction

The introduction of the Internet of Things (IoT), in the last years, has changed the lifestyle of thousands of people, thanks to a large number of sensors and actuators, interconnected with each other, that makes the diversity of the available systems bounded only by the imagination of the designer. Any person or sensor might trigger a proper device or group of them, at any moment and under whatever conditions, giving the feeling the home is somehow *alive* and capable of autonomous behaviour. This 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 domotic case, *vocal assistants*, working together with the rest of the compatible devices, assume an important role; firstly, because it's quite comfortable triggering devices behaviours, scheduling reminders or requesting general culture informations (e.g., like news, biography of known people, etc.) by means of the voice; secondly, because many of nowadays users grew up with the cultural heritage of science fiction literature and filmography, where vocal assistants at the service of humans, performing complex reasoning tasks, make human being's workflow easier. We can think of HAL 9000 in "2001: a Space Odyssey", or the more recent "OS 1" (self-named: "Samantha") of the movie: "Her". So, in one word, because it's cool.

Nowadays, companies producing vocal assistants, aim more at increasing their pervasiveness than at improving their native reasoning capabilities. Amazon, for example, after the success of the Echo line (Echo Dot, Echo Plus, Echo Show, etc.), recently proposed to put its vocal assistant Alexa in our cars (Echo Car) and even on top of a pair of glasses (with the announced product baptized as Echo Frames), while no news is available about the improvement of their reasoning capabilities. Additional features can be included by developers with the so-called "Skills" or leveraging external cloud services like IFTTT[1], but we are far away from what we mean: with "reasoning capabilities", we can intend not only the ability to infer the proper association *command* → *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. To let vocal assistants help us in our cognitive processes using a form of logic reasoning, we must provide them with facts and rules to be combined together, or more simply, we must give assistants the basis and the options to freely and implicitly extract all they need in order to do that, from texts in natural language. The latter was the start point of this research, whose goal has been achieved by subordinating direct command Reasoning with a Meta-Reasoning in the conceptual space.

The concept of Meta-Reasoning in the cognitive sciences is often associated to the utterance: *Thinking about Thinking*, which expresses only marginally what in this work is proposed.

Except the well known [41] cloud-based vocal assistants, other kind of solutions [42–44] are based on neural models exclusively trained on the domotic domain, or exploit chat engines [45, 46] whose understanding skills are strictly depending on syntax. This makes the range of their capabilities quite limited.

In light of the above considerations, our aim is the design of a cognitive architecture which we called CASPAR, based on Natural Language Processing (NLP), that make it possible the implementation of *cognitive* agents able to transcend the available ones in performing reasoning activities. Such agents could be used for both domotic purposes and any other kind of applications involving common deductive processes based on natural language. A first overview of such an architecture has been provided here [2], in the course of a conference on agents.

As a further motivation, we have to highlight that, as claimed in [17], cognitive architectures have been mainly used as research tools so far, and very few of them have been developed outside of academia; plus, none of them has been specifically

[1] `https://ifttt.com/`.

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.

As shown in the rest of this Section, the proposed cognitive architecture is made of four main components: (i) *Translation Service*, a Natural Language Processing (NLP) pipeline which converts natural language utterances in logical axioms; (ii) *Reactive Reasoner*, a multi-frame system expressed in a prolog-like semantic provided by the BDI framework Phidias [47], with the role of parser for both cognitive and reactive purposes; (iii) *Cognitive Reasoner*, a module implementing novel algorithms for asserting/querying knowledge bases composed by clauses possibly nested inside one another; (iv) *Smart Environment Interface*, an interface providing connection between the agent engine and the so-called *Smart Home* representing the outside world.

Although cognitive architectures should be distinguished from models that implement them, we can affirm that our architecture can be used as domotic agent *as is*, after the definitions of both the involved entities and the I/O interfaces. However, at the same time, our architecture can be used as a basis for the integration of extra features not included in our original prototype.

This chapter is structured as follows: Section 2.2 describes the state of the art of related literature; Section 2.3 shows in detail the semantic notation used and all the architecture's main components and underlying modules; Section 2.4 shows how to put in practice what theoretically explained in prior sections, providing also an experimental evaluation of real-time performances;

A Python implementation of CASPAR is also provided for research purposes in a Github repository[2].

2.2 Related works

Since the framework proposed in this chapter wants to be a sort of *crossover* between cognitive architectures and IoT, in order to instantiate *cognitive agents*, this section is focused on those works addressing the common issue of NLP plus IoT and logical deduction.

[2] `http://www.github.com/fabiuslongo/pycaspar`

In [48] a theoretical comparison between three of the most popular cognitive architecture is presented, which inspired several subsequent works: SOAR, CLARION, LIDA. In particular, SOAR is the oldest cognitive architecture developed. Originally coined by Newell and his colleagues, it was an acronyms for **S**tate **O**perate **A**nd **R**esult, which synthesized its main theme: the fact that cognitive tasks are represented by symbolic problem spaces containing a series of states. Such spaces are searched by production rules grouped into operators. This heuristic search driven behaviour was inherited directly from the GPS system. Exactly as in GPS, SOAR accomplishes problem solving by selecting, given a certain goal state to reach, appropriate operators able to reduce the *symbolic distance* between the goal state and current state. Although interesting, since it is was directly inspired by the cognitive psychology research, such an approach cannot be suitable to support real-time applications aiming at human-fashioned timing responses. Instead, for the latters, more vertical and focused approaches are required, than the one considering the *hope* of reducing randomly the symbolic distance from a goal. In any case, the architectures of this dissertation got also inspired by SOAR for two reasons: firstly, the usage of production rules; secondly, for the derivation of subgoals of the original ones, when the formers cannot be achieved[3], through a constrained run-time expansion of the knowledge base.

The authors of [49] present a computational model called MoralIDM, which integrates multiple AI techniques to model human moral decision-making, by leveraging a two-layer inference engine which takes in account of prior cases decisions and a knowledge base of a formal representation of moral qualitative-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) into predicate calculus.

The authors of [50] present a system called LUCIA built on Embodied Construction Grammar (ECG) and the SOAR cognitive architecture, which combines cognitive linguistics with known properties of human language processing, in order to process simple instructions for making reasoning. Furthermore, they propose the evaluation of computational models that are able to perform language comprehension using methods that approximate properties of human language processing; to

[3]A particular state which is defined as *impasse*.

this aim they introduce ten cognitive criteria. Its easy to demonstrates that Caspar, with a proper tuning, fulfills largely those criteria thanks to its native mapping between semantic and ontologies, as with the one (number seven) related to the selection of meanings according to contexts; in Subsection 2.3.4 it is shown how such a issue has been addressed.

In [51] the authors describe three different spoken dialog system, one of them based on the architecture FORR for the task of ordering books from the public library by phone. All three dialog systems are based on a local Speech-to-Text engine called PocketSphinx which is notoriously less performing than cloud-based systems[52]. This leads to a greater struggle to reduce the bias between user's requests and results.

The architecture DIARC[53] has been designed for addressing the issue of recognize morally and socially charged situations in human-robot collaborations. Although it exploits several known resources for NLP (such as Sphinx, Verbnet and Framenet), it has been tested only on trivial examples in order to trigger robot reactions, using an own symbolic representation of both known and perceived facts.

In general, we can say cognitive architectures[50, 54–56] leveraging NLP are limited in both in their domain of application, because they work only on small subsets of the idiom in exam, and also in terms of the syntactic structures recognition.

In the scope of IoT, as far as we know, there is not any IoT framework in the state-of-the-art for the implementation of *true* cognitive agents making usage of natural language as Caspar does. An agent based on natural language might also simulates different moods, showing a sort of human-like consciousness or reacting on specific utterances (like the commercial ones Alexa or Google); but currently they are no capable of combining acquired knowledge in order to infer new one. Apart such a important distinction, the IoT paradigm is rapidly growing with the new development, design, and integration of technology. As a result, several frameworks have been developed to fulfill some of the new requirements and needs.

The authors of [57] present a technical comparison of IoT frameworks engaged in the current industrial context that provide SoS solutions to Industry 4.0 issues.

In the scope of home applications, there are several frameworks aspiring to be competitive to the well-kwown commercial ones. One of them worth to be mentioned is Wit.ai[4], an open NLP platform which can be used via Web or as cloud service, allowing developers to build bot/conversational applications. Wit.ai provides an interface and an API to perform training of human conversations; the objective is to have a platform able to parse incoming messages (voice or text) into a structured data. The process is based on intents and entities: an intent is simply what the user intends to do (e.g. change Temperature), while entities are variables containing details of the user's task.

Mycroft[5] is another IoT framework built to be an open source answer to the commercial ones, whom instead are considered as *black boxes* because they allow some flexibility to create new Skills, but still usually strictly controlled. Mycroft gives also the user the chance to use different known Speech-to-Text engines either locals (Sphinx, DeepSpeech) or clouds (Google). Despite the considerable amount of packages for the framework installation, the user is still bound to logon to its web portal either for pairing devices or get new Skills.

Other surveys[58–60] reflect the remarkable advances and the interest around the paradigm of the IoT, although they are mostly focused on protocols, sensors and other technological aspects whom are on lower layers of processing than NLP.

2.3 The Architecture

We baptized our architecture as CASPAR: **C**ognitive **A**rchitecture **S**ystem **P**lanned **a**nd **R**eactive; the name summarizes its two main features. The core of CASPAR processing is managed by Phidias [47], a *Belief-Desire-Intention*[6] framework able to give Python programs the ability of performing logic-based reasoning (in the Prolog style); moreover, it lets developers write reactive procedures, i.e., pieces of program that can promptly respond to environment events.

In Fig. 2.1, all interacting components are depicted, each filled with a distinct colour. The main component of this architecture, namely the *Reactive Reasoner* (central box in Fig. 2.1), acts as ”core router” by delegating operations to other components, and providing all needed functions to make the whole system fully operative. The agent's Knowledge Base (KB) is divided into two distinct parts

[4]https://wit.ai/
[5]https://mycroft.ai/

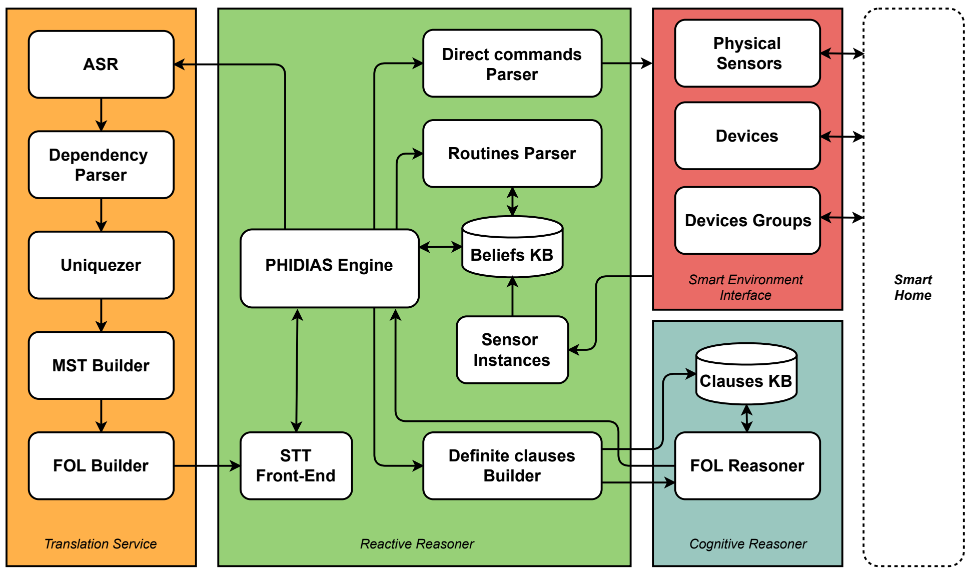


Figure 2.1: The Software Architecture of CASPAR.

operating separately, whom 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 affect on; the latter contains conceptual information not perceived by agent's sensors, but on which we want the agent makes 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 First Order Logic (FOL) definite clauses, and it can be interrogated providing answer to any query (*True* or *False*).

The two KBs represent, somehow, two different kind of human being memory: the so called *procedural memory* or *implicit memory*[61], 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 one another in a very reactive Decision Making process, as shown in Subsection 2.4.3.

In Fig. 2.2 it is shown a simplified data flow schema in CASPAR. Each sensor in the upper ovals can get information either from a microphone (in the case of

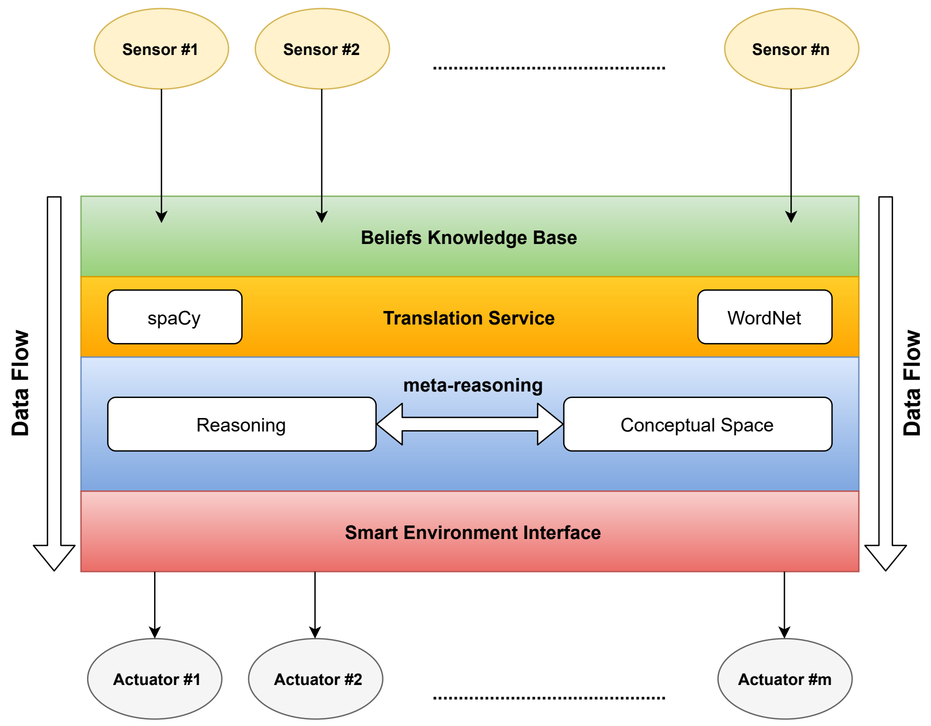


Figure 2.2: The Data Flow Schema in CASPAR.

vocal command) or from whatever other kind of device able of capturing physical quantities from environments. Each information coming from sensors is translated in specific beliefs and stored in the Beliefs KB. In the case a belief is not related to an utterance, the former will be sent directly to the Smart Environment Interface, without passing through the Translation Service, where libraries for piloting devices actuators are invoked. Otherwise, in the case of vocal commands, beliefs containing text of utterances will pass thought the Translation Service, in order to produce logical forms by leveraging the dependency parser *spaCy*[6] and the lexical resource *WordNet*[62]. Depending on the utterances content, vocal commands might be also subordinated by meta-reasoning in the conceptual space; otherwise, specific beliefs producted on the basis of logical forms, will trigger their related plan without involving meta-reasoning but directly the interaction with devices. Such a data flow will be explained in details in the next subsections.

In Section 1.2 it was briefly addressed the topic of consciousness, whose definition in this work has to be considered in the measure of how much it can affect

[6]https://spacy.io/

on agent's behaviour. Since CASPAR agents-derived behaviour is affected by meta-reasoning in conceptual spaces, we can consider consciousness just a further *beliefs producer* Sensor[7] as the upper ovals in Fig. 2.2, which gives its contribution to the current conceptual space. For what concerns *how* such beliefs would be asserted, and of what cognitive processes they may be the result, it is out of the scope of this work. I can just affirm that such architecture, in the presence of many sensors coming from the equivalent of the human senses, it would generate high values of Φ as Integrated Information, which in Section 1.2 we identified a necessary condition for the emergence of the consciousness of a system. Nonetheless, being made of production rules and FOL clauses, CASPAR does not contradict the Chalmers Principle of Organizational Invariance.

2.3.1 The Translation Service

This component (left box in Fig. 2.1) is a pipeline of modules with the task of taking a sound stream in Natural Language, provided by a software interface called *Sensor Instance* within the Reactive Reasoner, then translating it in a FOL expression. The central module of such a component, and the only one which changes across languages, is the so-called *Macro Semantic Builder*, which is a production rules system that *capture* utterances semantic starting from their dependencies, in order to create a meta-structure that facilitates logical form extractions.

The details of the modules and the related process is dealt with in the following; however, before proceeding with the modules illustration, next it is shown a background about the semantic notation produced by this component.

Information Representation

The choice of FOL representation has been made because of well-known algorithms of Unification (for comparing formulas) and reasoning like Backward-Chaining (for querying a KB), whose features we are aware, as we are confident due to the large amount of theory behind.

The Translation Service generates logical expressions inheriting the shape from the event-based formal representation of Davidson[63], with *from-one-to-three*-arity

[7]Although operating internally to the agent's engine.

predicates, labeled using lemmas and the so-called "Part-Of-Speech" (POS) tags; the latters give a shallow terms typization, useful for disambiguation/role distinction. So, every term `t` of this representation can be as follows:

$$\texttt{t:= x | L:POS(t) | L:POS(t}_1\texttt{, t}_2\texttt{, t}_3\texttt{) | L:POS(t}_1\texttt{, t}_2\texttt{)}$$

where `x` is a variable bound either to universal or existential quantifier, `L` is a lemma, `POS` is a Part-of-Speech tag. Also implication symbols are included in such a notation, when a group of predicates subordinate the remaining ones.

For this semantic notation, we took into account also the analysis done in [64] about the so-called *slot allocation*, which indicates specific policies about entity's location inside each predicate, depending on verbal cases. Moreover, terms with only one argument are used for both noun representation and quality modificators (adjectives and adverbs), which will share one argument to link them together.

Each verb and its relation with subjects and object are represented by a three arguments predicate: the first argument, defined as *davidsonian* variable, identifies uniquely the related action linked to the verb semantically described by the label, while the other two arguments represent subject and object. The latters order will be inverted in the presence of passive verbs, morphologically described by specific dependencies as *nsubjpass* and *agent*, whom implicitly summarize the *agent-driven* Parsons[65] approach (usually called "neo-Davidsonian"); in case of intransitive or imperative verbal phrases, respectively, subject or object slots will be left empty.

Adverbs are represented by one-arity terms, whose arguments will be equal to the davidsonian variable of the verb which are referred to.

Finally, prepositions are represented by two-arguments predicates; the first argument is either a davidsonian or common variable, while the second argument is the object of the preposition.

Here are some examples for each of the different cases. Let our utterance be the following one:

$$\textit{Robert drinks wine} \tag{2.1}$$

This case is represented by the conjunction of the following predicates (the existential operator is omitted):

$$\texttt{Robert:NNP(x}_1\texttt{)} \wedge \texttt{wine:NN(x}_2\texttt{)} \wedge \texttt{drink:VBZ(e}_1\texttt{, x}_1\texttt{, x}_2\texttt{)}$$

The predicate `Robert:NNP`(x_1) represents the fact that x_1 has `Robert` as lemma and `NNP` as POS. The same concept is applied to `wine:NN`(x_2). While, the predicate `drink:VBZ`(e_1, x_1, x_s) represents the fact that x_1 and x_2 are part of a verbal phrase with `drink` as lemma, `VBZ` as POS and e_1 as identificator of the action described by such entities interaction; for the rest of the paper we will refer to such predicates as *actions*.
Let us now suppose to add some additional informations to the utterance:

Robert slowly drinks good wine

In this case the two following predicates will be joined to the previous formula:

$$\texttt{slowly:RB}(e_1) \land \texttt{good:JJ}(x_2)$$

which are modificators both of the verbal phrase (adverb) and its object (adjective). In detail, `slowly:RB`(e_1) means that an adverb as *slowly* is applied to the unique action identified by e_1 and `good:JJ`(x_2) means that an adjective as *good* is applied to all predicates having x_2 as argument.
Let us suppose to apply a further modification to the utterance, in order to include a verbal preposition as well:

Robert slowly drinks good wine in the living room

still, three more predicates will join to the conjuction:

$$\texttt{in:IN}(e_1, x_3) \land \texttt{living:NN}(x_3) \land \texttt{room:NN}(x_3)$$

Some dependencies permit us to extract even interactions between different verbal phrases like the following:

Robert knows that Barbara drinks wine

in this case two actions are linked together by the means of davidsonian variable as follows:

$$\texttt{know:VBZ}(e_1, x_1, e_2) \land \texttt{Robert:NNP}(x_1) \land \texttt{drink:VBZ}(e_2, x_2, x_3) \land \\ \texttt{Barbara:NNP}(x_2) \land \texttt{wine:NN}(x_3)$$

As we can see, the object of the action labeled with `know:VBZ` is a davidsonian variable representing another action (`drink:VBZ`). Even non-davidsonian variables can be *actions-crossing*, like in the following example:

$$\textit{The man called Robert is a good man} \tag{2.2}$$

In this case the representation is:

$$\texttt{man:NN}(x_1) \wedge \texttt{Robert:NNP}(x_3) \wedge \texttt{call:VBN}(e_1, x_1, x_3) \wedge \texttt{man:NN}(x_2) \wedge \texttt{good:JJ}(x_2) \wedge \texttt{be:VBZ}(e_2, x_1, x_2)$$

Here the two actions labeled with `call:VBN` and `be:VBZ` share the same subject x_1.

Automatic Speech Recognition

This module allows a machine to understand the user's speech and convert it into a series of words through a computer program, thus creating a kind of natural and mutual communication [66–68].

Although it is actually encapsulated inside a Sensor Instance, whom will be shown in detail in Subsection 2.4.1, we decided to represent it at the top of the Translation Service, because it embodies the very beginning of the pipeline which will produce the final formula to be sent to the Reactive Reasoner.

Dependency Parsing

This module aims at extracting the semantic relationships between all words in a utterance coming from the ASR. Such relationships, which are also called *dependencies*, in this work play a key role in FOL expressions building. In the last years we had a significant advancement in developing fast and accurate dependency parser, for many idioms and developing languages, together with its employment in NLP applications. In [69] the authors present a comparative analysis of ten leading statistical dependency parsers on a multi-genre corpus of English. The study of the outcomes of these dependency parsers helped us to fill the gap between utterances in natural language and the machine information representations used for this work.

In the following subsections it will be shown how to build the so-called *Macro Semantic Table* proposed in this paper, i.e., a meta-structure between dependencies and FOL expressions, resuming in a canonical shape all semantic relations of a single utterance, from which it is possible to extract FOL expressions with ease.

Entities Uniquezation

When an utterance is interpreted and transformed into a set of words, according to the phrase itself, one or more words can be duplicated: as an example, in the sentence (2.2) the word "man" appears two times, however its role in the semantic, as well as its dependency with other terms, is different according to the place in which the word appears. For this reason, each word that appears more than one time must be properly identified. This is the task of this module, that aims at renaming these duplicated entities to ensure the correctness of the next pipeline module (Macro Semantic Table) outcome, whose data structures need a distinct reference to each entity coming from the dependency parser.

Since dependency parsers basically are classifiers trained on recognizing relations between words, they will never work properly whether such words are concatenated with numbers in the body of a sentence. In light of this, we cannot expect from a dependency parser the same outcome after having concatenated an assigned number (occurrence in the utterance) to every word.

While it is a trivial task to give a proper enumeration to every dependency entity when any word is not repeated, we cannot say the same in presence of more occurrences of the same word. Considering the dependencies shape, each of them is represented as it follows:

```
dependency(governor, dependent)
```

where `dependency` is a tag remarking a semantic relation between the two entities in the round brackets; `governor` and `dependent` are words within the sentence, which they always will be distinct with each other except for the ones related to the `ROOT` dependency. If we look at the dependencies structure, it will be easy to give a proper number to all dependents, because they follow the same words order in the body of each sentence. For instance, considering the dependencies of the sentence (2.2):

```
det(man,the)
nsubj(be,man)
acl(man,call)
```

```
oprd(call,Robert)
ROOT(be,be)
det(man,a)
amod(man,good)
attr(be,man)
```

the first basic renaming step will give the following result, where all dependents have been concantenated with their actual occurrency in the body of the sentence:

```
det(man,the01)
nsubj(be,man01)
acl(man,call01)
oprd(call,Robert01)
ROOT(be,be01)
det(man,a01)
amod(man,good01)
attr(be,man02)
```

At this point, the question would be: what about governors? How can we give the proper enumeration, for example, of every occurrency of governors whose value is `man`? In other words, the governor inside `acl` or `amod` or wherever `man` can be found, has to be renamed as `man01` or `man02`? The solution to such a issue comes from a special feature of the dependency parser which we used for this work. Such a tool, which is spaCy[70], together with the dependencies gives information about all entities offset within the body of the utterances, then linking together entities and offsets we'll obtain in any case unique entities within every dependency.

The contribution of this module is mandatory to ensure the correctness of the Macro Semantic Table building, which we will see in detail in the next subsection.

Macro Semantic Table builder

The purpose of this module, which is made of a system of production rules, is to build a novel semantic structure called *Macro Semantic Table* (MST) summarizing in a canonical shape all the semantic features of a sentence, in order to derive FOL expressions as described in subsection 2.3.1.

MSTs can only represent verbal phrases and are built through a matching of each dependency with a production rule, taking into account of languages diversity and dependency tagset. The same MST might be generated starting from two or more distinct set of dependencies, in a non-deterministic fashion-way. With a good knowledge of any idiom, the MST Builder can be shaped in order to create MST starting from different languages/dependency tags in a flexible way. Worth of mentioning it is that the first version of this module, for the sake of experimentation, was attempted by using classical procedural code and loops, but it was very difficult to handle in order to cover all dependencies. Moreover, as it can be inspected in the Github code[8], the current version not only is very compact and straightforward, but permits also to scale with ease on the final outcome's expressiveness.

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

$$\texttt{MST(u) = \{ACTIONS, VARLIST, PREPS, BINDS, COMPS, CONDS\}}$$

where:

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

All tuples inside such lists are populated with variables and labels whose indexing is considered disjoint between distinct lists, although there are significant relations which will be discussed ahead.

All dependencies we will take in exam for the rest of this chapter are part of the Clear NLP tagset[71] coupled with the Penn Treebank Part-of-Speech[72].

The VARLIST list

This list contains tuples representing either nouns or adverbs, defined as:

```
(var, lemma:POS)
```

[8] https://github.com/fabiuslongo/pycaspar/blob/master/mst_builder.py

where `var` is an assigned variable, `lemma:POS` is the lemma and Part-of-Speech either noun or adverb; the same value of `var` might have place either in `ACTIONS` or `PREPS`.

The ACTIONS list

Each tuple into `ACTIONS` have a cohesive role among other predicates, representing a natural language verbal phrase. Each tuple contains four entities:

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

where `lemma:POS` is the lemma and Part-of-Speech of a verb, `dav` an assigned davidsonian variable and `subj/obj` are variables in some tuples of `VARLIST`.

A new action is created by production rules matching with tags as: *nsubj*, *csubj*, *nsubjpass*, or implicitly (whether not already existing) in the case of: *ccomp*, *acl*, *relcl*, *dobj*, *xcomp*.

Whenever an `ACTION` is going to be created, the davidsonian and subject/object index counters (related to the same distinct sentence) get incremented on distinct counting threads, in order to populate all its initially emply slots and `VARLIST` with informations carried by the dependency in exam. Otherwise their values will remain an unknown default value (question mark) in case of other lexical forms (intransitive, imperative, passive).
For instance, considering the sentence (4.2), as *uniquezed* dependencies we will have:

```
nsubj(drink01, Robert01)
ROOT(drink01, drink01)
dobj(drink01, wine01)
```

At runtime, after the first dependency (*nsubj*) analysis, the corresponding MST will be:

$$\texttt{ACTIONS = [(drink01:VBZ, } e_1\texttt{, } x_1\texttt{, } x_2\texttt{)]}$$
$$\texttt{VARLIST = [(}x_1\texttt{, Robert01:NNP), (}x_2\texttt{, ?)]}$$

The `ROOT` tags will not be parsed. After the parsing of the third dependency *dobj*, we'll obtain a further information about the action's object, then `VARLIST` will change as it follows:

VARLIST = [(x_1, Robert01:NNP), (x_2, wine01:NN)]

Dependecies as *ccomp*, *acl* and *relcl*, might carry additional informations about interactions between distinct actions, which we'll see later for creating one another merged predicates.

The PREPS list

This list is made of tuples containing informations carried by the *prep* and *pobj* dependencies. The general schema is:

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

where `lemma:POS` is the lemma and Part-of-Speech of a preposition: the second argument can be either a davidsonian variable or a normal one, `obj` a variable present in a tuple of `VARLIST` as well. For instance, let the sentence be:

Robert drinks wine in the living room

and its dependencies:

```
nsubj(drink01, Robert01)
ROOT(drink01, drink01)
dobj(drink01, wine01)
prep(drink01, in01)
det(room01, drink01)
compounds(room01, living01)
pobj(in01, room01)
```

As a rule matches with the dependency *prep*, we still not have enough informations to finalize completely a tuple in `PREPS` until *pobj* is reached, due their connection by means of the word `in01`, but we can create a *pending* tuple inside `VARLIST` as it follows:

ACTIONS = [(drink01:VBZ, e_1, x_1, x_2)]

VARLIST = [(x_1, Robert01:NNP), (x_2, wine), (x_3, ?)]

PREPS = [(in01:IN, e_1, x3)]

with the reference of x_3 still undetermined.
As the last dependency *pobj* is parsed, which carries the value `room01` for x_3, the preposition can be finalized as it follows:

```
VARLIST = [(x1, Robert01:NNP), (x2, wine01:NN), (x3, room01:NN)]
                 PREPS = [(in01:IN, e1, x3)]
             COMPS = [(room01:NN, living01:NN)]
```

Due to the presence of the *compound* dependency, we take the chance to spend some words about the lists `COMPS`. The base concept is quite simple: all multi words nouns will share the same argument, which is this list purpose. Each couple in it contains a *referred* noun as first argument (that is within some couple in `VARLIST` as well) and the *referring* nouns as second argument.

The BINDS list

This list, whose name is the contraption of `BINDINGS`, is made of tuples as it follows:

$$(\texttt{label:POS}_1,\ \texttt{label:POS}_2)$$

where $\texttt{label:POS}_1$ is a tuple in `VARLIST` and $\texttt{label:POS}_2$ a new one encountered in a parsed dependency. Such a list is modeled by rules matching with the following dependencies group: *amod*, *poss*, *nummod*, *nmod*, *appos*, *quantmod*, with the purpose of storing information about adjectives and additional adverbs.
For instance, let the utterance in exam be:

Robert is a nice, calm and good fellow (2.3)

and its dependencies:

```
nsubj(drink01, Robert01)
ROOT(be01, be01)
det(fellow01, a01)
amod(fellow01, nice01)
punct(nice01, ,:,)
conj(nice01, calm01)
```

```
cc(calm01, and01)
conj(calm01, good01)
attr(be01, fellow01)
```

In this case, as dependencies carrying informations about modifiers we have *amod* and two occurrencies of *conj*; but they cannot be used as they come and put inside `BINDS`, unless they are *rectified*, i.e., shaped in the form *(referred, referent)*, where *referred* is a known label within one couple in `VARLIST` and *referent* a new label during the parsing, in order to obtain the following result:

$$\texttt{VARLIST = [(x}_1\texttt{, Robert01:NNP), (x}_2\texttt{, fellow01:NN)]}$$

```
BINDS = [(fellow01:NN, nice01:NN), (fellow01:NN, calm01:JJ),
                (fellow01:NN, good01:JJ)]
```

Considering the sentence (2.3) dependencies, we take also the chance to introduce the interaction between the last dependency *attr* with the MST: together with *acomp*, it assumes the role of value giver for a variable in `VARLIST`, previously instantiated with some action. Furthermore, in this case the dependency *conj* has the role of *modificator extensor*, then it will populate the `BINDS` list as well, which usually is fed also by another dependencies group made of: *advmod*, *neg*, *npadvmod*.

The COMPS list

As anticipated during the `PREPS` analysis, this list is made of tuples as it follows:

$$\texttt{(lemma:POS}_1\texttt{,lemma:POS}_2\texttt{)}$$

where $\texttt{label:POS}_1$ have place within a tuple in `VARLIST` due to some already parsed dependency, while $\texttt{label:POS}_2$ is a new entity. Its population is fulfilled by a production rule matching with the dependency *compound*.

The CONDS list

This list contains davidsonian variables whose actions will be used to create predicates subordinating other ones in the FOL expression. It is modeled by a production rule matching with the dependency *advmod*, whether the entity is an explicit conditional word like *when*, *if* or *while*; otherwise the entity will be put in `BINDS` as modificator (adverb/adjective). For instance, let the utterance be:

When the sun shines strongly, Robert is happy (2.4)

in this case the related MST will be:

$$\texttt{ACTIONS = [(shine01:VBZ, } e_1\texttt{, } x_1\texttt{, } x_2\texttt{), (be01:VBZ, } e_2\texttt{, } x_3\texttt{, } x_4\texttt{)]}$$
$$\texttt{VARLIST = [(}x_1\texttt{, sun01:NN), (}x_2\texttt{, ?), (}x_3\texttt{, Robert01:NNP),}$$
$$\texttt{(}x_4\texttt{, happy01:JJ)]}$$
$$\texttt{CONDS = [}e_1\texttt{]}$$

As for the question mark coupled with the variable x_2 inside `VARLIST`, since there was not any dependencies having changed its value, the verbal action related to `shine01:VBZ` is considered intransitive.

The FOL Builder

This module aims to build FOL expression starting from the MST structures seen in the prior subsection.
Since (virtually) all approaches to formal semantics assume the Principle of Compositionality[9], formally formulated by Partee[73], every semantic representation can be incrementally built up when constituents are put together during parsing. By virtue of this, it is possible to build FOL expressions straightforwardly starting from a MST, which summarize all semantic features in a sentence.

Now we will focus on each list within the MST and their role in the FOL expressions building. For each tuple `(var, lemma:POS)` in `VARLIST` the following predicate[10] will be created:

`lemma:POS(var)`

which represents a noun like `tiger:NN(`x_1`)` or `Robert:NNP(`x_1`)`, etc. `var` can be also a davidsonian variable when `POS` has the value of `RB`. In such cases the tuples represent adverbs like `Hardly:RB(`e_1`)` or `Slowly:RB(`e_2`)`, etc.

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

[10] In this subsection the enumeration coming from the Uniquezer is omitted.

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 like:

`be:VBZ`(e_1, x_1, x_2) or `shine:VBZ`(e_2, x_3, x_4), etc.

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 respectively in a tuple of `ACTIONS` or `VARLIST`, while obj is a variable in a tuple of `VARLIST` as well. Such predicates will represent verbal/noun prepositions. For instance, considering the sentence:

Robert travels in Italy

starting from `(In:IN, e1, x3)` the following predicate will be created:

`In:IN`(e_1, x_3)

together with:

`Travel:VBZ`(e_1, x_1, __)
`Robert:NNP`(x_1)
`Italy:NNP`(x_3)

Similarly, *The State of Alabama* will be represented as:

`State:NNP`(x_1)
`Of:IN`(x_1, x_2)
`Alabama:NNP`(x_2)

For each tuple (`lemma:POS`$_1$,`lemma:POS`$_2$) in `COMPS`, whose first entity `lemma:POS`$_1$ is in a tuple of `VARLIST` as well, a predicate will be created as it follows:

`lemma:POS`$_2$(`var`$_{nn}$)

where $\texttt{var}_{nn}$ is the variable of a the tuple in VARLIST with $\texttt{lemma:POS}_1$ as second entity. In case of multi-word nouns, each further noun over the first of them in VARLIST will be encoded within COMPS. For instance, considering the multi-word noun: *Barack Hussein Obama*, it will be encoded as:

$$\texttt{Barack:NNP(x}_1\texttt{)} \wedge \texttt{Hussein:NNP(x}_1\texttt{)} \wedge \texttt{Obama:NNP(x}_1\texttt{)}$$

where first $\texttt{Barack:NNP(x}_1\texttt{)}$ will be extracted from VARLIST, then the rest from COMPS.

The list BINDS is reserved to create future predicates representing adjectives with almost the same function of COMPS, with the slight difference that $\texttt{var}_{nn}$ might be either in VARLIST or COMPS. For instance, *good man* will be represented as:

$$\texttt{Man:NN(x}_1\texttt{)} \wedge \texttt{Good:JJ(x}_1\texttt{)}$$

As for CONDS, it contains davidsonian variables whose actions (plus their related predicates) subordinate all other predicates in the corresponding FOL expressions. For instance, considering the sentence (2.4), such expressions would be an implication like it follows:

$$\forall\ \texttt{e}_1\ \texttt{LEFT\_HAND\_SIDE(e}_1\texttt{)} \implies \exists\ \texttt{e}_2\ \texttt{RIGHT\_HAND\_SIDE(e}_2\texttt{)}$$

where $\texttt{LEFT\_HAND\_SIDE(e}_1\texttt{)}$ incorporates the following predicates with x_1 as universal bound variable:

$$\texttt{shine:VBZ(e}_1\texttt{, x}_1\texttt{, \_\_)} \wedge \texttt{sun:NN(x}_1\texttt{)}$$

and $\texttt{RIGHT\_HAND\_SIDE(e}_2\texttt{)}$ incorporates:

$$\texttt{be:VBZ(e}_2\texttt{, x}_3\texttt{, x}_4\texttt{)} \wedge \texttt{Robert:NNP(x}_3\texttt{)} \wedge \texttt{happy:JJ(}x_4\texttt{)}$$

with x_3 and x_4 as existential bound variables.

It appears clear that whether a davidsonian variable is found inside CONDS, its related actions (together with all related predicates) will migrate to the left-hand side as subordinating conditions of the overall logical expression.

In the next subsections we will see how to obtain *nested* formulas, which means to substitute all action's variables with their own subject/object related predicates and eliminating the not more useful davidsonian variables, in order to obtain definite clauses.

Since the MST Builder is made of production rules taking in account of relations (dependencies) between words, as long as such relations are treated properly by some rule, the accuracy of the conversion from natural language to logical form can be clearly considered equal to the accuracy of the dependency parser. As for the latter employed for this work's case-study, which is spaCy[70], in the author's website[11] it is reported to have a state-of-the-art accuracy of 90% for all trained models available for the English idiom. Moreover, out of the accuracy, in this comparison [69] spaCy is also recommended for its speed, hence it can be considered an optimal choice in the scope of NLP real-time applications, as the one proposed in this chapter.

2.3.2 The Reactive Reasoner

This component (central box in Fig. 2.1) has the central task of letting other component communicate with each other; it also embed modules as the Speech-To-Text (STT) Front-End, IoT Parsers (Direct commands Parser and Routines Parser), Sensor Instances and Definite Clauses Builder. The Reactive Reasoner incorporates also the Beliefs KB, which supports both Reactive and Cognitive Reasoning.

The Reactive Reasoner processes the FOL expressions provided by the Translation Service by delegating the parsing work to its modules, depending on the required operation. Among such modules there is the Direct commands Parser, which has the task of extracting from an IoT command what needed (operations and parameters) in order to select a specific plan and execute all related operations.
When subordinating conditions within the IoT command are detected, the parsing work is delegated to the Routines Parser, which will schedule operations execution accordingly to other parameters provided by the Sensor Instances. In the case-study of Section 2.4 direct commands and routines examples are provided.

In the field of IoT, the parsing of natural language commands so far has been addressed mostly by encoding utterances and training neural models with their features. Such a strategy has been one of the most favorite and often even performative, although quite *vertical* to be used only on specific domains. As for this paper approach, the idea of less specific agents is preferred, which, of course, requires an

[11]https://spacy.io/models/en#en_core_web_lg

utterances recognizer able to work on a wider domain. That's because the modules of the Reactive Reasoner work on logical representations of natural language commands. In any case a training is needed, but in this work is provided at the level of semantic word relations given by a pre-trained dependency parser; even if, in order to leverage such a technology, as we have seen in the prior subsections, several operations and choices must be accomplished before.

In this work, a production rules system is used as reactive tool to trigger proper plans in the presence of specific asserted beliefs. In [74] we have shown the effectiveness of this approach leveraging the BDI framework Profeta[75], even with a shallower analysis of the semantic dependencies, as well as an operations encoding via WordNet in order to make the operating agent multi-language and multi-synonimous.

Furthermore, together with direct commands, next we will also see how to infer routines from natural language, involving entities and devices; the same kind of operations which, in commercial products, one can obtain only by operating on a complex menu system within a proprietary apps.

The core of CASPAR processing is managed by Phidias[47]; the latter, as Profeta successor, is able to give Python programs the ability of performing logic-based reasoning (in the Prolog style); plus, it let developers to write reactive procedures, i.e. pieces of program that can promptly respond to environment events.

An important module of the Reactive Reasoner is the Definite Clauses Builder, which is responsible of combining FOL expression predicates with common variables, via a production rules system, in order to produce *nested* definite clauses (made possible of composite terms). Considering the sentence (2.4) and its related FOL expression producted by the Translation Service, the Definite Clauses Builder, taking in account of the Part-of-Speech of each predicate, will produce the following 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 requires a 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 Part-of-Speech tags and applying the following hierarchy to every noun expression as it follows:

$$\texttt{IN(JJ(NN(NNP(x))), t)} \tag{2.5}$$

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; 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.

2.3.3 The Cognitive Reasoner

This component (right bottom in Fig. 2.1) allows an agent to assert/query the Clauses KB with nested definite clauses, built by the Definite Clauses Builder described in the prior subsection.
Beyond the nominal FOL reasoning made by the Backward Chaining algorithm, this framework exploits also another class of FOL expressions here defined as *assignment rules*. We refer to a class of rules of the type "P is-a Q" implicitly entailed, 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:

$$\forall x \; \texttt{Robert(x)} \implies \texttt{Man(x)} \tag{2.6}$$

but before that, if we create FOL expressions like those shown in the previous subsections, we must do a premise: the introduction of such rules in a KB can be possible only by implicitly 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. In other terms, in a conceptual domain the predicate `P(x)` means that `x` *has the property of being* `P`, which is different than `L:POS(x)`, i.e. `x` *is a gramatical term identified by the lemma* `L` *and the Part-of-Speech* `POS`.
Basically we need a medium to give additional meaning to our predicates, which is provided by WordNet. This will allow us to make logical reasoning in a conceptual space, by means of the following functions:

$$F_I : P_S \longrightarrow P_C \tag{2.7}$$

$$F_{Args(F_I)} : X_S^n \longrightarrow Y_C^n \tag{2.8}$$

F_I is the *Interpreter Function* between the space of all semantic predicates P_S which can be yield by the `MST` sets and the space of all conceptual predicates P_C having a synset as label; it is not injective, because a single semantic predicate might have multiple correspondences in the codomain, depending on the context incorporating the lemmatized word (which is part of the predicate label in the domain).
$\mathrm{F}_{Args(F_I)}$ is between domain and codomain (both with arity equal to n) of all predicate's arguments of F_I.
For instance, considering the FOL expression of *Robert is a man*:

$$\texttt{be:VBZ(e}_1\texttt{, x}_1\texttt{, }x_2\texttt{)} \wedge \texttt{Robert:NNP(x}_1\texttt{)} \wedge \texttt{man:NN(}x_2\texttt{)} \tag{2.9}$$

after an analysis of *be*, we find this lemma within the WordNet synset encoded by `be.v.01` and defined by its gloss as: *have the quality of being* (referred to a subject and an object/adjective). 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 the functions 2.7 and 2.8, the expression 2.9 can be rewritten as[12]:

$$\texttt{be.v.01\_VBZ(d}_1\texttt{, y}_1\texttt{, y}_2\texttt{)} \wedge \texttt{Robert\_NNP(y}_1\texttt{)} \wedge \texttt{man.n.01\_NN(y}_2\texttt{)}$$

where VBZ indicates the present tense of be.v.01, Robert_NNP(y_1) means that y_1 identify the person *Robert*, and man.n.01_NN(y_2) means that y_2 identify *an adult person who is male (as opposed to a woman)*.
Considering the meaning of be.v.01_VBZ and that *be* is considered a *copular verb*, i.e., an intransitive verb but identifying a subject with an object, it does make sense also to rewrite the previous expression as:

$$\forall \texttt{y Robert\_NNP(y)} \implies \texttt{man.n.01\_NN(y)} \tag{2.10}$$

Having such a rule in a KB means that we can implicitly admit additional clauses having man.n.01_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(y) becomes argument of man.n.01_NN(y) as it follows:

$$\texttt{man.n.01\_NN(Robert\_NNP(y))} \tag{2.11}$$

which agrees with the hierarchy of 2.5 as outcome of the Definite Clauses Builder.

It is relevant to say that an utterance might contain more than one occurrence of the lemma *be*, so the following question arises: what to do in those cases? The policy is to check only the label of the so-called *utterance verb driver* among available ones, which is provided by the dependency *ROOT*.
We explain better considering another utterance as example:

Robert is a guy who is fine

In such case there are two occurrences of the verb *be*, but only the first will be classified as *ROOT* by the dependency parser. In light of this, the corresponding assignment rule will be the following:

$$\texttt{Robert\_NNP(y)} \implies \texttt{be.v.01\_VBZ(guy.n.01\_NN(y), fine.s.04\_JJ(y4))}$$

[12]In the new conceptual domain, the semicolon between lemmas and POS is replaced by the underscore.

For a proper synset choice, the reader is referred to the subsection 2.3.4.

As with the *assignment rules*, intuitively, the so-called *causal rules* could also be defined, when in presence of verbs like *induce*, *cause*, *entail*, etc. as *ROOT* entity. In those cases, the FOL expression could also be splitted into two hand sides as *antecedent* and *consequent*. Such rules, might be inferred regardless the idiom, by checking the belonging of the lemma to the synset *induce.v.02*, whose gloss is defined as: *cause to do; cause to act in a specific manner*. A comprehensive result could also be obtained by checking the kinship of hyperonym as *induce.v.02*, which will include the policy also the verbs *entails*, *imply*, *mean*, *lead*, etc. Unfortunately, in the scope of this work a causal rule not always can be inferred *as is*, unlike assignment rules, unless to deal with a *pure causal* KB, because there is some aspect to take in account to hold the overall coherence.
For instance, let the sentence be the following one:

smoking causes lung cancer

whose FOL expression is:

$$\texttt{cause:VBZ(e1, x1, x2)} \wedge \texttt{smoking:NN(x1)} \wedge \texttt{lung:NN(x2)} \wedge \texttt{cancer:NN(x2)}$$

About the American Lung Association, it has been estimated that active smoking is responsible for close to 90 percent of lung cancer cases. In light of such an estimate, we cannot represent realistically the prior utterance like it follows:

$$\texttt{smoking:NN(x1)} \implies \texttt{lung:NN\_cancer:NN(x2)}$$

without expressing the *domain-dependent* uncertainty level as well (which could also be zero). Such a issue could be addressed by introducing a fuzzy knowledge representation, which has been out of the scope of this research so far.

In order to make logical reasoning with the Backward Chaining algorithm, we need to obtain definite clauses from our representation, either to feed or query the KB. As claimed in [18], 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. In our Clauses KB no single literal will ever be negative due to the *closed-world assumption*, since negations (treated like whatever adverb) when detected and

related to *ROOT* dependency, are considered as polarity inverter of verbal phrases; so, in this case, the *tell* operation 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).

Name	Synset
$\text{GLOSS}_{l,S}$	$\text{argmax}_{Synset}\text{sim}(\text{S, Synset.gloss}_l)$
$\text{EXAMPLES}_{l,S}$	$\text{argmax}_{Synset}\text{sim}(\text{S, Synset.example}_l)$
$\text{BEST}_{l,S}$	$\text{argmax}_{Synset}\text{max}(\text{GLOSS}_{l,S}, \text{EXAMPLES}_{l,S}))$
$\text{AVERAGE}_{l,S}$	$\text{argmax}_{Synset}\text{avg}(\text{GLOSS}_{l,S}, \text{EXAMPLES}_{l,S}))$
$\text{COMBINED}_{l,S}$	$\text{argmax}_{Synset}\text{sim}(\text{S, concat(Synset.gloss}_l, \text{Synset.example}_l))$

Table 2.1: Type of metrics of an unsupervised *naive* synset choice, for a lemma l of the sentence S.

2.3.4 Synsets Selection and Word Sense Disambiguation

A proper synset choice for each predicate's label fulfills the task of Word Sense Disambiguation, because in the case of the usage of the same lemma in two sentence expressing different meaning for it, the difference is implicit for a human being; but not for a machine. So, to let an agent differentiates two predicates expressing different meaning, taking in account of the context, the usage of synsets (properly selected) can be a suitable solution. A synset expresses exactly a meaning of a word, described by its gloss and conveyed through its lemmas.

The task of synset choice in this work is strictly related with the so-called *Common-Sense Conceptual Categorization.* Different psychological studies[76–78] started by Rosch results[13] achieved in the mid-1970s, suggest that people can use either of prototypes or exemplars in organization tasks. In this work it was taken in account the second categorization type, which in literature is known as "exemplar-based categorization". A more prototype-alike categorization might also be achieved

[13]Rosch's result indicates that conceptual knowledge is organized in our mind in term of prototypes.

by considering WordNet hypernym instead of synsets, which would give a more general meaning to each predicate's label.

In order to achieve a proper synset choice, we considered the insight about which in a common sense words embedding, the "bag of words" (context) used for the gloss/examples related to a synset comprising a lemma, should be vectorially closer, among all synsets comprising that lemma, to another bag of words making effective usage of such a lemma in a similar context. In light of this, we had interesting results by exploiting an unsupervised naive strategy, taking in account of different combinations of doc2vect[79] similarity (sim) between the sentence in exam and the informations within the related WordNet synsets, thus considering one of the following options for each lemma l of the sentence S as in Table 2.1, depending on the semantic richness of the domain. The same information have been exploited in other works as well, based on vectorial similarity involving synset glosses and mentioned in a survey[80] on Word Sense Disambiguation.
For instance, considering $\texttt{EXAMPLES}_{l,S}$ from the Table 2.1, the target word *bass* and the following two sentences:

He likes to eat a bass

He likes to play the bass

In the first case, it is easy to verify that the selected synset for *bass* will be: `Sea_bass.n.01`, whose gloss is: *the lean flesh of a saltwater fish of the family Serranidae.*
In the second case, the selected synset will be: `Bass.n.07`, whose gloss is: *the member with the lowest range of a family of musical instruments.*

Instead, for semantic richer domains, could be better to use $\texttt{COMBINED}_{l,S}$ from the Table 2.1. For instance, considering the target word *bank* and the following sentences:

Three masked men stolen all cash money from the bank

In this case the selected synset will be:
`depository_financial_institution.n.01`, whose gloss is: *a financial institution that accepts deposits and channels the money into lending activities.*

The boy leapt from the bank into the cold water

Here the selected sysnet will be: `bank.n.01`, whose gloss is: *sloping land (especially the slope beside a body of water).*

For this first prototype of Caspar, we chose to follow the above approach also because the common sense words embedding used to compute the metrics in Table 2.1 is contained in the same model exploited by the dependency parser (spaCy); indeed, such a choice favors times performances over all.
We haven't accomplished an accuracy analysis of this kind of disambiguation, due to its dependence on the domain and especially for the presence/absence of the fields `examples` in WordNet, whom sometimes are not populated; a possible strategy to address such a lack could be to integrate other lexical resources (ConceptNet, VerbNet, etc.). But considering that also the best performative state-of-the-art approach not reaches the 100% of accuracy, since even slight differences in the synset choice might lead also to unsuccessful reasoning, we thought to a strategy to address such a issue, which in this work is defined as *Grounded Meaning Context.* In the scope of a distinct session, during a normal conversation, it is most likely that words with same lemmas are considered to have the same meaning. In light of this, by setting in the Caspar configuration file a parameter named `GMC_ACTIVE` to the value of *True*, it is possible to set all labels with the same lemma to the first encountered synset value of them, in the scope of the same session.

In order to achieve a lighter clause processing/reasoning, Caspar gives the also chance to restrict the synset choice only to a specific subset of Part-of-Speech, by changing a specific parameter in the configuration file.

2.3.5 Nested Reasoning and Clause Conceptual Generalizations

The aim of the Cognitive Reasoner is to query a KB made of *nested* definite 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 the already seen *assignment rules* for derivation of new knowledge.

The Backward Chaining algorithm, as is, in presence of clauses differing one another only in the arguments, when the latters are not variables and could be replaced consistently with the KB, cannot be effective. To achieve such a goal, when required clauses are not present in the KB but deductible by proper arguments substitutions, 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 it will be shown how, by expanding properly the KB at assertion-time, the reasoning itself can be achieved acceptably.

In order to obtain the above goal, in this work two novel algorithms called *Nested-Tell* and *Nested-Ask* are proposed, whom extend the radius of the nominal Backward Chaining through an expansion of the KB with new clauses generated by arguments substitutions at assertion-time.

For instance, considering a KB made, at most, of one-level[14] nested predicates, like the following:

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

querying such a KB with $\mathtt{P_3(H_3(x))}$ using the Backward Chaining algorithm will return *False*, because there are not any unifiable literals present, neither as consequent of a clause. Instead, by applying the substitution related to $\mathtt{H_3(x)} \implies \mathtt{F_3(x)}$, we can also query the KB with $\mathtt{P_3(F_3(x))}$ which is present as consequent of the first clause; that's what CASPAR achieves by means of *Nested-Ask* (algorithm 3).
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 $\mathtt{P_1(G_1(x_1))}$ and $\mathtt{P_2(G_2(x_2))}$. Such literals, although not initially present in the KB as asserted clauses, can be obtained by replacing their argument leveraging other clauses in the same KB, in order to assert what is required for making the Backward Chaining return *True*; that's what the *Nested-Tell* (algorithm 2) achieves, implicitly asserting the following clauses together with $\mathtt{P_1(F_1(x_1))}$ and $\mathtt{P_2(F_2(x_2))}$:

[14]Supposing a zero-level nested predicate be $\mathtt{P(x)}$.

$$\mathtt{P_1(F_1(x_1))} \implies \mathtt{P_1(G_1(x_1))}$$
$$\mathtt{P_1(F_1(x_1))} \implies \mathtt{P_2(G_2(x_2))}$$

Since we cannot know in advance what a future successful *nested* reasoning requires, along with the previous clauses also the so defined *Clause Conceptual Generalizations* can be asserted:

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

where the antecedent of the implication is unchanged to hold the quality of the rule, while $\mathtt{F_1(x_1)}$, $\mathtt{F_2(x_1)}$, $\mathtt{F_3(x_3)}$, as satisfiability contributors of respectively $\mathtt{P_1(F_1(x_1))}$, $\mathtt{P_2(F_2(x_2))}$, $\mathtt{P_3(F_3(x_3))}$, are assumed asserted together with the latters. In other terms, the predicates: $\mathtt{P_1}$, $\mathtt{P_2}$, $\mathtt{P_3}$ can be considered as *modifiers* of respectively $\mathtt{F_1}$, $\mathtt{F_2}$, $\mathtt{F_3}$.
A generalization of the implication's antecedent is possible only through a weaker assertion of the entire expression, by changing $\implies$ with $\land$ as it follows:

$$\mathtt{G_1(x_1)} \land \mathtt{G_2(x_2)} \land \mathtt{F_3(x_3)}$$

which is not admitted as definite clause in this KB, being not a single positive literal. In any case, the existence of the triple (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 (2.12)

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

```
Hard(Shine(Sun(x1), __)) ⟹ Slowly(Drink(Barbara(x3),
                    Fresh(Lemonade(x4))))
```

Considering as *modifiers* adjectives, adverbs and prepositions, following the schema in Table 2.2 where: `A=Applied`, `NA=Not Applied`, all the clauses generalizations

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

Table 2.2: Generalization scheme related to the sentence (2.12).

(corresponding to the first three rows of the table, while the forth is the initial clause) can be asserted as it follows:

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

where the adverb *Hard*, being common part of all the antecedents composition, is always *Applied* to hold the quality of the rules, while the consequent shape will range on all possible variations of its modifiers, which will be 2^n with n=#modifiers.

Although in the previous example the number of generalizations is equal to 4, in general might be quite higher: it has been observed, after an analysis of more text corpus from the Stanford Question Answering Dataset[81], 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 number 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 changing[15] the policies of selective inclusion/exclusion of modifiers categories (adjectives, adverbs or prepositions).

In any case, as shown in the case-study of Section 2.3, the synergy between generalization assertions and Nested-Tell gives the chance to Nested-Ask to achieve a successful reasoning on a wider range of cases, whom otherwise would not be achieved with the nominal Backward Chaining.

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 this

[15]Check out config.ini in the Github repository.

approach sacrifices space for a lighter reasoning, but we rely on three distinct points in favor of our choice:

1. An efficient indexing policy of the Clauses KB, in order to obtain 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 chapter we assume the assignment rules properly chosen, among the most likely ones which can get the query closer to a proper candidate. In the next chapter a custom balancing between two distinct Clauses KB working on different levels has been presented as possible solution: in the lower level (long-term memory) only clauses pertinent with the query will be searched, then put in the high one (short-term memory) for attempting a successful reasoning. Similar approaches have also been used with interesting outcomes in some of the widespread Cognitive Architectures[48].

The only issue which cannot be ignored is that a Clauses KB *expansion*, due to the Clause Conceptual Generalization, of course depends on the assertions order (as we will see better in Section 2.4). For giving an idea, the reader might imagine a Matryoshka toy with all its combining chances: only one combination will reduce all the pieces in one (the biggest doll), one within each other, while whatever other combination will left useful pieces out, whether not used in the proper sequence. A possible strategy to address such a issue, for a future work, could be to attempt a further expansions reconsidering already processed clauses (without assert the already existing ones) after the assertion of other possible related ones, selected under efficient heuristics, which might participate in the process. But such a strategy must be employed under proper constraints, in order to avoid a useless and excessive expansion of the KB.

In the next subsection we will see in detail the Nested-Tell and Nested-Ask algorithms, which we have briefly described in this subsection.

2.3.6 Algorithms

Algorithm 1 shows the pseudo-code of function *produce_clauses()* which produces a set of single positive literals derived from an initial literal (parameter *literal*), accordingly to a specific knowledge base (parameter *KB*). It is implemented as a recursive function and takes an additional parameter (*derived*) representing the set of new single positive literals progressively produced by the recursive calls. The algorithm goes through all the clauses in the knowledge base (line 4) and, for each argument in the input literal (line 5), it checks for possible unification (line 6). If the left hand side of a clause in the knowledge base can be unified with the considered argument, the latter can be replaced (in a copy of literal) with the correspondent right hand side of the input literal, producing a derived literal (line 8) which differ in the presence of the new standardized argument. The new literal is checked for possible unification with other in derived, to avoid similarities differing on variables name (line 11) and included in the list *derived* (line 15). Of course, in order to assure completeness of the algorithm, the same procedure needs to be invoked for the newly generated literal (line 16). This happens until no new arguments unification is found.

Algorithm 1 produce_clauses(*KB*, *literal*, *derived*)

Input: (i) *KB*: set of `definite clause`, (ii) *literal*: `single positive literal`, (iii) *derived*: set of `single positive literal`, should be empty in the first call.
Output: A set of `single positive literal`, derived from the literal.

```
 1: declare kb_clause: definite clause;
 2: declare arg, arg_std, new_literal: single positive literals;
 3: declare clause_unified: boolean flag;
 4: foreach kb_clause in KB do
 5:     foreach arg in ARGS(literal) do
 6:         if UNIFY(GET_LHS(kb_clause), arg) is not fail then do
 7:             arg_std ← standardize(arg);
 8:             new_literal ← REPLACE(literal, arg_std, GET_RHS(kb_clause));
 9:             clause_unified ← False
10:             foreach derivation in derived do
11:                 if UNIFY(derivation, new_literal) is not fail then do
12:                     clause_unified ← True;
13:                     break
14:             if clause_unified is False then do
15:                 derived ← derived ∪ new_literal;
16:                 produce_clauses(KB, new_literal, derived);
```

Algorithm 2 shows the pseudo-code of function *nested_tell()* which includes a new clause (parameter *clause*) into a knowledge base (parameter *KB*), together with all the other clauses, in the form of implications, that can be derived by it accordingly to the current content of the knowledge base. Note that the derived clauses should be computed only if the input clause is a single positive literal.

The algorithm checks if the input clause is an implication (line 3). If this is the case, then the clause should be inserted in the knowledge base without any further work (line 12). Otherwise, if the input clause is a single positive literal, all the derived clauses are obtained by invoking Algorithm 1 (line 5) and by using all the single positive literals in the returned set (lines 6) to generate new implications (line 8) in the case the two hand sides are not unifiable (line 7)[16]. In these implications, the left hand side is the initial clause (line 9) while the right hand side is one of the the returned literals (line 10).

Algorithm 2 nested_tell(*KB*, *clause*)

Input: (i) *KB*: set of `definite clauses`, (ii) *clause*: `definite clause`
Output: A new knowledge base (set of `definite clause`) in which the clause has been inserted together with all the other clauses derived by it

```
1:  declare derived: set of single positive literals;
2:  declare derived_clause: definite clause;
3:  if GET_LHS(clause) is $\emptyset$ then do
4:      derived $\leftarrow$ $\emptyset$;
5:      produce_clauses(KB, clause, derived);
6:      foreach derived_clause in derived do
7:          if UNIFY(clause, derived_clause) is fail then do
8:              declare new_clause: definite clause;
9:              SET_LHS(new_clause, clause);
10:             SET_RHS(new_clause, derived_clause);
11:             ASSERT(KB, new_clause);
12: ASSERT(KB, clause);
```

Finally, Algorithm 3 shows the pseudo-code of function *nested_ask()* which provides a domain extension to the classical Backward Chaining algorithm, when the latter fails. This algorithm takes into consideration all the possible candidates that can be derived from the original query (parameter *candidates*) through unifications and substitutions. If the original query or one of the derived candidates is found to be true, the function returns the set of substitutions that supports the proof. The function returns *False* otherwise.

The algorithm is implemented as a recursive function so parameter *candidates* should be set equal to the empty set when calling the function for the first time, then it goes through all the clauses in the knowledge base (line 5) and for each argument of the goal (line 6) it checks for possible unification with the left hand side

[16]The assertion of a clause having their handsides unifiable with each other would make the Backward Chaining algorithm hang.

of the considered clause (line 7). If a unification is possible, then a new candidate is produced from a copy of the goal by replacing its argument in exam with the corresponding standardized right hand side of the clause (line 9). If no unifiable candidates with the new one have been discovered so far (line 11, 12), the latter is inserted in the set of the candidates (line 16) and the classical Backward Chaining algorithm is invoked (line 17). If the reasoning is successful, then the function exits providing the related set of substitutions (line 19). Otherwise, a recursive call is performed passing the candidate as new goal. If all the clauses present in the knowledge base are considered without finding a possible derived candidate which is provable to be true, the function simply returns *False*.

Algorithm 3 nested_ask(*KB*, *goal*, *candidates*)

Input: (i) *KB*: set of `definite clause`, (ii) *goal*: `single positive literal`, (iii) *candidates*: set of `single positive literal`, should be empty in the first call.
Output: A ¡`single positive literal`, `substitution`¿ for the goal or for another candidate or false

```
 1: declare clause: definite clause;
 2: declare arg, candidate: single positive literal;
 3: declare result: substitution;
 4: declare clause_unified: boolean flag;
 5: foreach clause in KB do
 6:     foreach arg in ARGS(goal) do
 7:         if UNIFY(GET_LHS(clause), arg) is not fail then do
 8:             new_arg ← standardize(GET_RHS(clause));
 9:             candidate ← REPLACE(goal, arg, new_arg);
10:             clause_unified ← False
11:             foreach cand in candidates do
12:                 if UNIFY(cand, candidate) is not fail then do
13:                     clause_unified ← True;
14:                     break
15:             if clause_unified is False then do
16:                 candidates ← candidates ∪ candidate;
17:                 result ← Ask(KB, candidate);
18:                 if result is not false then do
19:                     return result;
20:                 return nested_ask(KB,candidate, candidates);
21: return false;
```

2.3.7 The Smart Environment Interface

This component provides a bidirectional interaction between the architecture and the outer world. As shown in Fig. 2.1 and with further details in the Section 2.4, 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 expressed in high level language. The latter should contain all

```python
class HotwordDetect(Sensor):
    def on_start(self):
       self.running = True
       print("\nStarting Hotword detection...")
       # put instantiation hotword code here

    def on_stop(self):
        print("\nStopping Hotword detection...")
        self.running = False

    def sense(self):
        while self.running is True:
           # -------------> put hotword detection code here <---------------
           # when right hotword is detected: self.assert_belief(HOTWORD_DETECTED("ON"))


class UtteranceDetect(Sensor):
    def on_start(self):
       self.running = True
       print("\nStarting utterance detection...")
       # instantiate hotword engine here

    def on_stop(self):
        print("\nStopping utterance detection...")
        self.running = False

    def sense(self):
        while self.running:
           # -------------> put utterance detection code here <---------------
           # when incoming new utterance detected: self.assert_belief(STT(utterance))
```

Figure 2.3: Python implementation of the Sensor Istances `HotwordDetect` and `UtteranceDetect`.

required functions for piloting each device in order to get the wanted behaviour, whose implementation in this work is left to the developer.

2.4 Case-study

In this prototype implementation of CASPAR, Python was chosen as developing language for two main factors: firstly, the availability of Phidias, which is Python-based; secondly, the increasing popularity of the language itself, which was ranked first in 2020 [82] with a share of 30.09% and a trend of +3.9%. In regard of the dependency parser, the spaCy [70] framework was exploited, which is written in Python as well and provides neural models of different sizes, trained on a good range of idioms. As hardware platforms, the framework has been tested in both a common Windows OS architecture (i5 6600K, 16GB ram) and a Raspberry Pi 4 (4Gb). The latter has lately become quite popular in the field of home automation, especially for the low price (on sale now for $55) and its interesting interfacing features.

```
# 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 on
+INTENT(X, "light", "kitchen", T) / lemma_in_syn(X, "switch.v.03") >>
    [exec_cmd("switch.v.03", "light", "kitchen", T)]
+INTENT(X, "light", Y, T) / lemma_in_syn(X, "switch.v.03") >> [show_line("Result: invalid location")]

# turn off
+INTENT(X, "light", "living room", T) / lemma_in_syn(X, "change_state.v.01") >>
    [exec_cmd("change_state.v.01", "light", "living room", T)]
+INTENT(X, "alarm", "garage", T) / (lemma_in_syn(X, "change_state.v.01") &
    eval_cls("Be(House(x1), Safe(x2))")) >> [exec("change_state.v.01", "alarm", "garage", T)]

# open
+INTENT(X, "door", "living room", T) / lemma_in_syn(X, "open.v.01") >>
    [exec_cmd("open.v.01", "door", "living room", T)]
+INTENT(X, "door", "kitchen", T) / lemma_in_syn(X, "open.v.01") >> [exec_cmd("open.v.01", "door", "kitchen", T)]
+INTENT(X, "door", Y, T) / lemma_in_syn(X, "open.v.01") >> [show_line("Result: invalid location")]

# specify, set, determine, define, fix, limit
+INTENT(X, "cooler", "bedroom", T) / lemma_in_syn(X, "specify.v.02") >>
    [exec_cmd("specify.v.02", "cooler", "bedroom", T)]
+INTENT(X, "cooler", Y, T) / lemma_in_syn(X, "specify.v.02") >> [show_line("Result: invalid location")]

# cut
+INTENT(X, "grass", "garden", T) / lemma_in_syn(X, "cut.v.01",) >> [exe_cmd("cut.v.01", "grass", "garden", T)]
+INTENT(X, "cut.v.01", "grass", Y, T) / lemma_in_syn(X, "cut.v.01",) >> [show_line("Result: invalid location")]

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

Figure 2.4: The Smart Environment Interface.

```
> +FEED("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))
-------------------------------------------
4 definite clauses added to Knowledge Base

> +FEED("Colonel West is American")

Be(Colonel_West(x1), American(x2))
Colonel_West(x) ==> American(x))
-------------------------------------------
2 definite clauses added to Knowledge Base

> +FEED("missiles are weapons")

Be(Missile(x1), Weapon(x2))
Missile(x) ==> Weapon(x)
-------------------------------------------
2 definite clauses added to Knowledge Base

> +FEED("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))
-------------------------------------------
16 definite clauses added to Knowledge Base

> +FEED("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))
-------------------------------------------
1 definite clauses added to Knowledge Base

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

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

Query: Be(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}
```

Figure 2.5: Caspar Clauses KB changes, after assertions.

For the sake of shortness, we will not report in this chapter the whole agent's code, referring the reader to the Github repository. Instead, we will give a limited overview about the application-specific modules which need modifications, in order to customize the architecture for the user's purposes.

Before going further in this section, an important thing to highlight is that CASPAR is also ready to exploit Phidias multiagent features, in order to send beliefs in the KB of other instances, even if they are microcontroller units (MCU) with very few resources (as reported in [47]).
Phidias gives also the chance to instantiate a service to fulfill the interoperability between heterogeneous systems, even when they use different native communication protocols. In the case of MCUs providing a small memory footprint, there will be no need to install the entire CASPAR framework, but just the Smart Environment Interface or a Sensor Instance (depending on the type of interaction), together with Phidias.

After such a premise, in the subsections ahead we will see in details the code of the Sensor Instances, how they work with some examples, finally a discussion about run-time performances for the two architectures as experimental evaluation.

2.4.1 The Sensor Instances

As shown in Figure 2.1, the connection of CASPAR with the outer world is fulfilled by the Smart Environment Interface, which exchanges data with its counterparts Sensor Instances within the Reactive Reasoner. Such Sensor Instances can be seen in detail in Fig. 2.3, namely *HotwordDetect* and *UtteranceDetect*, which are instances of the superclass *Sensor* provided by Phidias. The latter let each instance work asynchronously with respect to the main agent's engine.

The instance *HotwordDetect* has the task of waiting for a waking word which will pull out the agent from its idle state. The method `on_start(self)` at the line 2 has the task of initialize the Speech-to-Text (STT) engine; another method at line 7, namely `on_stop(self)`, has the task of stopping the activity of the Sensor Instance; at line 11, the method `sense(self)`, when running and a proper word is detected, will assert the belief `HOTWORD_DETECTED`. Such a Sensor Instance is specific-designed for STT engines trained on recognize few specific words, because they have to be fast without accessing continuously to possible external clouds for the detection and recognition of multi-words utterances. The latter task is fulfilled by the second Sensor Instance in Fig. 2.3 at the line 18, namely `UtteranceDetect`, whose working schema is similar to `HotwordDetect` and produces the belief `STT(X)` with `X` as detected multi-words utterance.

The reader can notice, by the commented lines 5, 13, 14 and 21, 29, 30, that we have left the developer the freedom to choose his preferred engines, for both hotwords [83, 84] and general utterances [52] recognition, among the recommended ones.

In a similar way, whatever other kind of Sensor can be instantiated by asserting the related beliefs inside the method `sense`, when specific physical parameters are detected. In this case, one or more rule must be added among the others in the Smart Environment Interface and related plans must be create for them, to make the agent deal with the new beliefs.

2.4.2 IoT Commands Processing

As the voice-related beliefs are asserted by Sensor Istances, the STT Front-End (within the Reactive Reasoner) will handle the user's utterance for extracting the related intentions together with associated parameters; but before that, the *HotwordDetect* sensor must be started. When a hotword is detected, *HotwordDetect* is stopped and, together with the assertion of the belief `WAKE(ON)`, *UtteranceDetect* is started: the latter will waits for fifteen[17] seconds for some input, otherwise it will return to its initial state.

On the agent's `WAKE(ON)` state, for this prototype the user can:

1. Give IoT commands/routines, which they will be parsed by the Direct Commands/Routines Parser.

2. Feed the Clauses KB, after giving the further hotword *listen*, this time managed by *UtteranceDetect*.

3. Query the previously populated Clauses KB, after giving the further hotword *reason*, this time managed by *UtteranceDetect*.

[17]This value can be changed in the config.ini file of this work github repository.

4. End the cognitive phase at once (started either with *listen* or *reason*), giving the keyword *done*.

5. Wait for a custom value in seconds and let the agent return to its idle state.

As an input utterance is processed by the Translation Service, whether its FOL expression contains an implication, it will be automatically classified as routine, parsed and stored in the Beliefs KB; otherwise, it will be treated as direct command.

The *Direct Command Parser* (within the Reactive Reasoner) has the task of combining specific beliefs containing common variables, with the final aim of triggering one of the production rules in the Smart Environment Interface in Fig. 2.4. For instance, let the direct command be:

Set the cooler at 27 degrees in the bedroom

In this case, the Translation Service will produce the following FOL expression:

$$\texttt{set:VB}(\texttt{d}_1,\ \texttt{\_\_},\ \texttt{x}_1) \land \texttt{cooler:NN}(\texttt{x}_1) \land \texttt{at:IN}(\texttt{d}_1,\ \texttt{x}_2) \land \texttt{27:CD}(\texttt{x}_2) \land \texttt{degree:NNS}(\texttt{x}_2) \land \texttt{in:IN}(\texttt{x}_2,\ \texttt{x}_3) \land \texttt{bedroom:NN}(\texttt{x}_3)$$

The Direct Command Parser, invoked by the Reactive Reasoner, will produce the following belief:

```
INTENT(set, cooler, bedroom, (at 26 degree))
```

which contains as arguments: operation type, subject, object and other parameters. Such a belief will trigger the rule in line 29 of Fig. 2.4, then the associated plan inside the square brackets will be executed, whose parameters will be sent to the devices by the means of the procedure *exec_cmd*; the latter is subclass of the base class *Action* of Phidias and has the following arguments:

```
specify.v.02, cooler, bedroom, (at 26 degree)
```

where the operation type is encoded by `specify.v.02`, being a synset code provided by WordNet.
The same result can be achieved using also words like: *specify*, *determine*, *define*, *fix* or *limit*, instead of *set*. This can be possible by subordinating the rule triggering with the so-called *Active Belief* `lemma_in_syn(X, Synset)`, which will be *True* only

when `X` is a lemma contained into the specified WordNet `Synset`. The usage of such a condition, after a proper entities definition, makes the rule system multi-synonimous and multi-language, because the WordNet synset codes are the same across different languages. Each entity except the operation type, of course, will be language-specific and the designer can choose to encode them either as single or group of devices.

As for the *Routine Parser*, it has the task of combining proper beliefs in order to produce commands that cannot be executed without the presence of other specific beliefs, which might be asserted by some Sensor Instance. For instance, let the command be:

Turn off the lights in the living room, when the temperature is 25 and the time is 12

In this case, the Translation Service will produce the following FOL expressions:

$$\texttt{be:VBZ}(d_2,\ x_3,\ x_4) \wedge \texttt{be:VBZ}(d_3,\ x_5,\ x_6) \wedge \texttt{temperature:NN}(x_3) \wedge \texttt{25:CD}(x_4) \wedge \texttt{time:NN}(x_5),\ \texttt{12:CD}(x_6) \Longrightarrow \texttt{turn:VB}(d_1,\ \_\_,\ x_1) \wedge \texttt{off:RP}(d_1) \wedge \texttt{light:NNS}(x_1) \wedge \texttt{in:IN}(d_1,\ x_2) \wedge \texttt{living:NN}(x_2) \wedge \texttt{room:NN}(x_2)$$

then the *Routines Parser*, invoked by the Reactive Reasoner, will produce the following beliefs:

```
COND(337538, be, temperature, 25)
COND(337538, be, time, 12)
ROUTINE(337538, turn, light, living room, off)
```

Such beliefs represent a routine and its related two conditionals, all linked together by the same code `337538` (a timestamp of the daytime in milliseconds). All routines and relatives conditionals reside in the Beliefs KB until some Sensor Instance produces beliefs which meet the conditionals themself (line 3 of Fig. 2.4). In detail, whether some sensor produces both the following beliefs:

```
SENSOR(be, temperature, 25)
SENSOR(be, time, 12)
```

then, the routine and its conditionals are retracted from the Beliefs KB and the following belief is asserted:

```
INTENT(turn, light, living room, off)
```

which will trigger the rule at line 17 of Fig. 2.4.

2.4.3 Reasoning and Meta-Reasoning

In this subsection we will show the outcomes of a command processing inferred from a natural language utterance, taking also part in a meta-reasoning process during a functional interaction between Beliefs KB and Clauses KB. But before introducing meta-reasoning, let us see how CASPAR processes a slightly rephrased KB (*Colonel West*) treated in [18], achieving also a successful reasoning on the related query. The main utterance of the *Colonel West* case is the following one:

It is a crime for an American to sell weapons to hostile nations

which was arbitrarily encoded as:

$$\texttt{American(x)} \land \texttt{Weapons(y)} \land \texttt{Sells(x, y, z)} \land \texttt{Hostile(z)} \implies \texttt{Criminal(x)}$$

The subjectivity of such encoding raise the issue of the so-called *tailorability*, making logical forms entailed from the same sentence potentially distinct one another. The natural language query is the following one:

West is a criminal?

which is encoded as the ground literal:

`Criminal(West)`

Now, if we asserted a CASPAR's representation of the main utterance whose subject is *crime*, in order to obtain a successful reasoning for the specified query which refers on *being a criminal*, we should also introduce an additional clause regarding the relation between *crime* and *criminal*! Since we want to avoid this, for the sake of shortness we allow ourselves to rephrase the main utterance as it follows:

When an American sells weapons to a hostile nation, that American is a criminal

In common spoken language, one would gladly use the pronoun *he*, instead of the second occurrency of *American* (without *that).* In such a case, before CASPAR's parsing, an anaphora Resolution is needed. Among open-source tools, as already mentioned in Section 1.4, we had interesting results by using NeuralCoref [85] especially for its fully integration with SpaCy.

In Fig. 2.5 all new clauses[18] producted after every assertion by using CASPAR are shown. Althought this prototype is designed to work as vocal assistant, one can alike verify the reasoning by asserting manually the belief `FEED` in the Phidias shell, instead of `STT` at line 30 of Fig. 2.3.
Since representing-query literal is the following:

$$\texttt{Be(Colonel\_West(x}_1\texttt{), Criminal(x}_2\texttt{))}$$

we can clearly see that there is no definite clause in Fig. 2.5 having as such a literal as right-hand side. Among all, the closer literal is at line 51, but it has $\texttt{American(x}_4\texttt{)}$ instead of $\texttt{Colonel\_West(x}_1\texttt{)}$ as first argument of `Be`; in such case the nominal Backward Chaining algorithm will return *False*.
Instead, by querying with the Nested-Ask algorithm, the following new candidate leading to a successful reasoning can be inferred by leveraging the clause at line 13:

$$\texttt{Be(American(x}_1\texttt{), Criminal(x}_2\texttt{))}$$

The rest of the work will be accomplished by the nominal Backward Chaining, by leveraging the clauses at lines 47, 37, 41.

Next, taking in exam a more IoT-relevant KB, we will show how a meta-reasoning will take part in a direct commands processing. Let the initial utterances be the following ones (supposing they are asserted after the agent start-up):

When an inhabitant is at home the house is safe.

Robert is an inhabitant.

The corresponding clauses produced by the Definite Clauses Builder will be:

$$\texttt{At(Be(Inhabitant(x1), \_\_), Home(x5))} \Longrightarrow \texttt{Be(House(x3), Safe(x4))}$$
$$\texttt{Be(Robert(x5), Inhabitant(x6))}$$
$$\texttt{Robert(x)} \Longrightarrow \texttt{Inhabitant(x)}$$

where the third clause expresses the same meaning of the second one, but in the shape of an assignment rule.
Lets suppose now a further clause asserted by another Sensor Instance, exchanging

[18]Part-of-Speech tags are obmitted

data with an image recognition system. Successful tests have been conducted by connecting a Camera Module V2 to a second Raspberry Pi4, than the main one with a full installation of CASPAR; In the former, only the Sensor Instance was running. After the face recognition of the camera, the Sensor Instance of the second device will assert a belief containing a natural language utterance, into the KB of the first one, related to the recognized person in the nearby of the home entrance:

`+FEED("Robert is at home")`

The Definite Clauses Builder, in synergy with the Nested-Tell algorithm, will assert the following further clauses:

```
Be(Robert(x20), __)
At(Be(Robert(x20), __), Home(x22))
At(Be(Robert(x20), __), Home(x22)) ⟹ At(Be(Inhabitant(v_1), __),
Home(x22)))
```

At this point, supposing to give the agent the command: *Turn off the alarm in the garage*, the rule at the line 19 of Fig. 2.4 will be triggered only after a successful evaluation of the following clause:

`Be(House(x3), Safe(x4))`

It is easy to verify how the Backward Chaining algorithm fulfills such a evaluation, in a KB composed by the previous two groups of clauses. The meta-reasoning related to such evaluation is possible thanks to a special belief native of Phidias (*Active Belief*), namely `eval_cls(Y)`, subordinating the plan together with the other conditionals and querying the Clauses KB with `Y`.

In the case of routines, the second device should assert also a further belief, together with `+FEED("Robert is at home")`, to let a time-subordinated routine become an effective intent and trigger the rule, as seen in Subsection 2.4.2. This further belief could be as the following one:

`SENSOR(be, person, detected)`

together, for instance, to a time-subordinated one as it follows:

`SENSOR(be, time, 22:00)`

after which, when the time is `22.00` the agent will take (in an autonomous way) the decision of executing the command, on the basis of the meta-reasoning.

2.4.4 Evaluation

In this subsection, we report the results of some experiments conducted on the two architectures mentioned at the beginning of this section, by taking the response times on three different scenarios. Although six distinct users have tested the framework, differentiated by age and gender but with results quite close one another, the following results were obtained by the first author of this paper which is not an English-native speaker. All three tables discussed in the following are made of 4 columns: Architecture, Detect Time (STT Detection Time, comprising the utterance pronunciation), Parsing Time (pure CASPAR processing) and Overall (Detect Time + Parsing Time). In every scenario, the STT detection is achieved by using the Google API cloud.

In Table 2.3, we can see only a slight difference between the performance of the two architectures, working on a direct command after the agent start-up: an average loss on the overall performance of -0,15% on the Raspberry over Windows i5-6600K. The difference become even more thin in Table 2.4 (-0,12%) and Table 2.5 (-0,11%), even if the related commands are subordinated by a meta-reasoning. The loss depends on both the STT service and the dependency parser; the latter remains in the cache after the first parsing and make the subsequent operations lighter, which is what happen in Tables 2.4, 2.5, due to the pre-population of the Clauses KB.

In light of the above, in general we can affirm that in a CASPAR-based IoT system the real-time performances will depend mostly on both the responsiveness of the actuators devices and the quality of the Internet connection (on which, in this experiment, relays the Translation Service); the latter, in this case-study, was 80/20 Mbit download/upload.

Architecture	Detect Time	Parsing Time	Overall
Win i5-6600K	3,802	1,675	5,477
	3,243	0,065	3,308
	3,050	0,089	3,139
	3,146	0,053	3,199
	3,099	0,062	3,161
Raspberry Pi4 B	4.101	2,017	6,118
	3,852	0,156	4,008
	3,640	0,194	3,834
	3,338	0,185	3,523
	3,212	0,333	3,545

Table 2.3: Realtime performances (in seconds) in the case of consecutive successful executions of the command: *turn off the light in the living room.*

Architecture	Detect Time	Parsing Time	Overall
Win i5-6600K	3,656	0,064	3,720
	3,215	0,059	3,274
	3,175	0,056	3,231
	3,375	0,074	3,449
	3,056	0,053	3,109
Raspberry Pi4 B	4,695	0,103	4,798
	3,554	0,059	3,613
	3,600	0,086	3,686
	3,527	0,052	3,579
	3,439	0,055	3,494

Table 2.4: Realtime performances (in seconds) in the case of consecutive successful executions of the command: *turn off the alarm in the garage*, subordinated by: *The house is safe* and a KB made of: *Robert is an inhabitant, Robert is at home, When an inhabitant is at home the house is safe.*

Architecture	Detect Time	Parsing Time	Overall
Win i5-6600K	3,068	0,072	3,140
	3,029	0,096	3,125
	2,942	0,086	3,028
	3,042	0,095	3,137
	3,044	0,069	3,113
Raspberry Pi4 B	4,523	0,059	4,582
	2,793	0,070	2,863
	3,253	0,055	3,308
	3,144	0,061	3,205
	3,810	0,060	3,870

Table 2.5: Realtime performances (in seconds) in the case of the command: *turn off the alarm in the garage*, subordinated by: *Colonel West is a criminal* and the KB in Fig. 2.5.

Chapter 3

AD-CASPAR: Abductive-Deductive evolution of CASPAR

3.1 Introduction

Among applications 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. 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 a utterance the intentions together with the related parameters, then to execute the wanted operation providing 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. The early shared aim for conversational chatbot systems was to pass the Turing test, hence to fool the user about his interlocutor; the state-of-art of such chatbot systems can be probed in the scope of the Loebner Prize Competition [86].

One of the most common platforms for building conversational chatbot is AIML [87] (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 [88] AIML and Chatscript [89] 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 [90], making usage of huge corpus of conversations to train generative models that, given an input, are able to generate answers. In general, all chatbots are not easily scalable without writing additional code or re-train models with fresh datasets. As for the latters, unfortunately neural networks (in particular, deep neural networks 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 more deep semantic level as a dependency parser, in order to build logical models of utterances in natural language, prevents much more such a drawback.

In this chapter, a cognitive architecture called AD-Caspar based on NLP and FOL reasoning is presented, as baseline platform for implementing scalable and flexible chatbots with both goal-oriented and conversational features; nevertheless, this architecture leverages question-answering techniques and it is able of combining facts and rules in order to infer new knowledge monotonically from its own knowledge base. This first prototype is not yet capable of instantiating chatbots with 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, raising an implicit horizontal scaling of acquired responsiveness on possible questions. After every parsed sentence, as we will show in the next sections, the system made of agent+knowledge base is able to act as a deductive database [91].

AD-Caspar inherits most of its features directly from its predecessor Caspar seen in Chapter 2. The additional features introduced in AD-Caspar are the usage of abduction as pre-stage of the deduction (that's why the presence of AD before Caspar), in order to make inference 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.

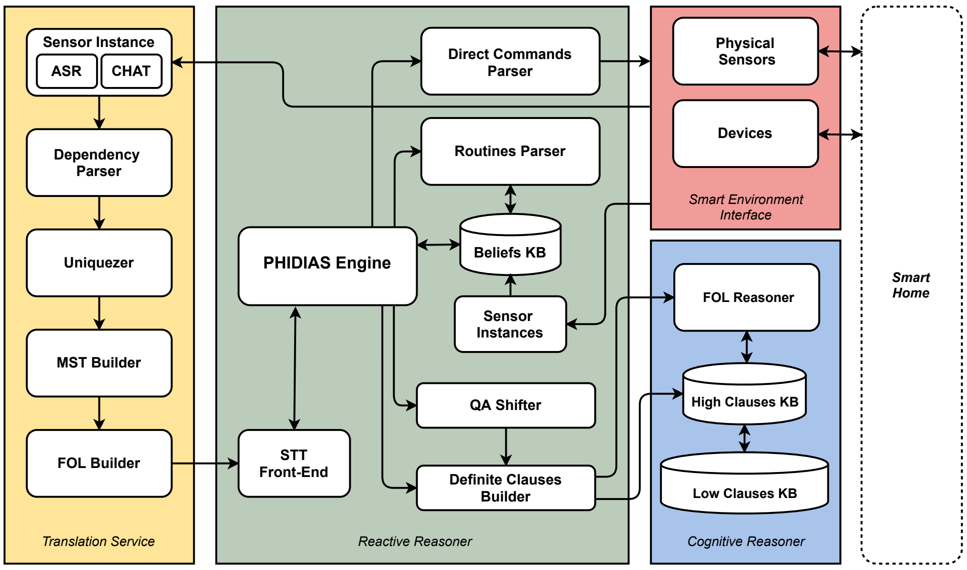


Figure 3.1: The Software Architecture of AD-CASPAR

This chapter is structured as follows: Section 3.2 shows in detail all the architecture's components and underlying modules; Section 3.3 shows how AD-CASPAR deals with polar and wh-questions; Section 3.5 is about a case-study where it is shown an instance of *thinking* Telegram chatbot. A Python prototype implementation of AD-CASPAR is also provided for research purposes in a Github repository[1].

3.2 The Architecture

In this section all interacting components of this architecture are explained. AD-CASPAR inherits all its features from CASPAR, plus specific modules for question-answering, and abductive reasoning supported by a NoSql engine to deal with large knowledge bases.

The main component of this architecture, namely the *Reactive Reasoner* (central box in Fig. 3.1), acts as "core router" by delegating operations to other components, and providing all needed functions to make the whole system fully operative. The Reactive Reasoner contains a further module than respect to CASPAR, which is called *QA Shifter*, having the task of recombining grammatical terms from a *question shape* to more possible *answer shapes*, as shown in Section 3.3.

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

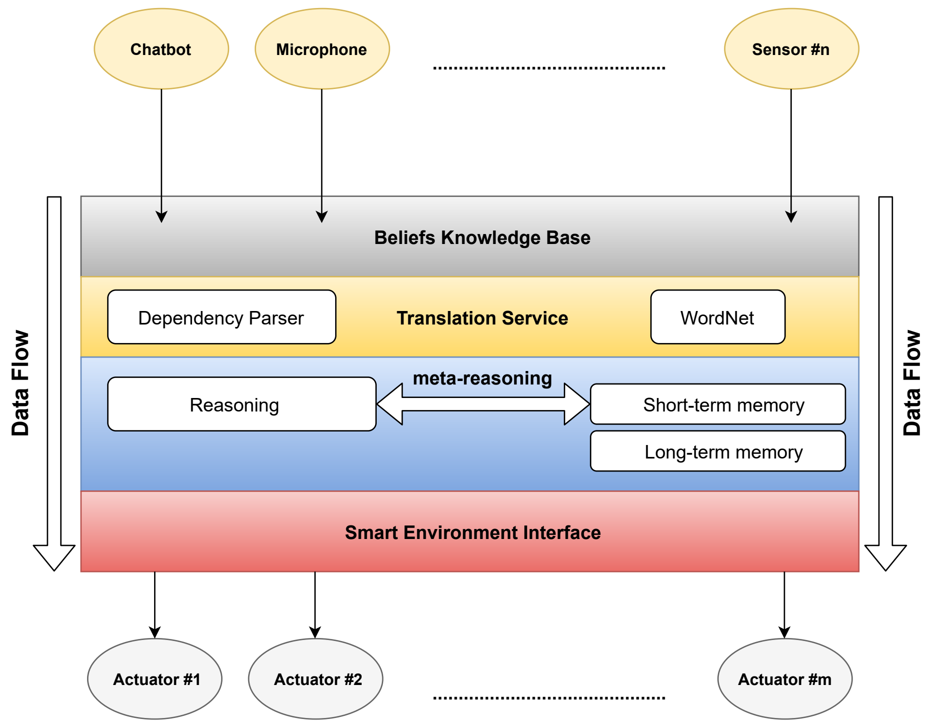


Figure 3.2: The Data Flow Schema in AD-CASPAR

As for AD-CASPAR, its Knowledge Base (KB) is divided into *Beliefs KB* and *Clauses KB*, where the latter is further divided in two distinct layers: *High Clause KB* and *Low Clauses Kb*. The totality of the knowledge is stored in the low layer, but the logical inference is achieved in the high one by means of the *FOL Reasoner*. Among all clauses, only the most relevant for the query in exam will be transferred from Low Clauses KB to High Clauses KB, taking into account of a confidence threshold based on specific features which will be discussed ahead. The two layers of the Clauses KB can be seen as *Short-Term Memory* (High Clauses KB) and *Long-Term Memory* (Low Clauses KB).

The *Translation Service* (left box in Fig. 3.1) shares the same features inherited from CASPAR and seen in Subsection 2.3.1, excepting for the *Sensor Instance* at the beginning of the pipeline, which can alternatively obtains utterances from a chat window as well. Finally, the *Smart Environment Interface* (upper right box in Fig. 3.1) is also entirely inherited from CASPAR.

In Figure 3.2 it is shown a simplified data flow schema in AD-CASPAR. Each sensor in the upper ovals can get information either from a chatbot, a microphone (in the case of vocal commands) or whatever other kind of device able of capturing physical quantities from the environment. Each information coming from sensors is translated in specific beliefs and stored in the Beliefs KB. In case a belief is not related to an utterance, the former will be sent directly to the Smart Environment Interface, without passing through the Translation Service, where libraries for piloting devices actuators are invoked. Otherwise, in the case of vocal commands, beliefs containing text of utterances will pass through the Translation Service, in order to produce logical forms by leveraging a dependency parser and the lexical resource WordNet. Depending on the utterances content, vocal commands might be also subordinated by meta-reasoning in the conceptual space, which is made of Short- and Long-term memory, before interacting with devices.

3.3 Question Answering

In this section it is shown how this architecture deals with question-answering. Differently from its predecessor, which works with a single-volatile Clauses KB, AD-Caspar can count on a two-layer Clauses KB: High Clauses KB and Low Clauses KB. Every assertion is made on both the layers, but the logical inference is made only on the High one. As for queries, whether a reasoning fails, the Low Clauses KB is used to populate the High one with relevance-based clauses, taking into account of common features between clause-query and clauses stored in the Low Clauses KB. Each record in the Low Clauses KB is stored in a NoSQL database and is made of three fields: nested definite clause, features vector and the sentence in natural language. The features vector is made of all labels composing a clause. For instance let's considering the following sentence:

Barack Obama became president of United States in 2009

In this case, the record stored in Low Clauses KB will be as it follows[2]:

- `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]`

[2]Considering the nested notation seen in the prior chapter including also POS tags.

`- Barack Obama became president of United States in 2009`

The abductive strategy of transfer from Low Clauses KB to High Clauses KB takes in account of a metric defined as $Confidence_c$ between all records in Low Clauses KB and a query:

$$Confidence_c = \frac{|\bigcap(Feats_q, Feats_c)|}{|Feats_q|} \tag{3.1}$$

where $Feats_q$ is the features vector extracted from the query, and $Feats_c$ is the features vector in a record stored in Low Clauses KB.
Once obtained the sorted list of all features occurrences, together with the related clauses, the most relevant clauses will be copied in the High Clauses KB. Such a operation is accomplished via the `aggregate_clauses_greedy` algorithm shown in Algorithm 4, which with a *greedy* euristic 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 transferred in the High Clauses KB, considering the wanted distance (3.1) from the query.

3.4 Algorithm

First, at line 1 of Algorithm 4, `aggregate_clauses_greedy` extracts the vector containing all the features of *clause*; 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 a value. For instance, considering the Table 3.1, having a query with the features `[a, b, c]`, the function `get_relevant_clauses_from_db` will create a list made of the following two tuples:

```
(3, [cls_4, cls_5])
(2, [cls_2, cls_3])
```

The rationale behind such function's output is that the first value of the each tuple is the size of the intersection between the features of the query and the features of Table 3.1, while the second value is a vector containing the clauses themselves having in common such a intersection size. At line 4, the first value of each tuple

```
1  aggr = db.clauses.aggregate([
2          {"$project": {
3              "value": 1,
4              "intersection": {"$size": {"$setIntersection": ["$features", features]}}
5          }},
6          {"$group": {"_id": "$intersection", "group": {"$push": "$value"}}},
7          {"$sort": {"_id": -1}},
8          {"$limit": 2}
9      ])
```

Figure 3.3: The Python Mongodb aggregate operator implementing the function `get_relevant_clauses_from_db` at line 2 of Algorithm 4.

Clauses	Features
`cls_1`	`[a, x, z, y]`
`cls_2`	`[a, b]`
`cls_3`	`[a, b, x, y]`
`cls_4`	`[a, b, c, d]`
`cls_5`	`[a, b, c, w]`

Table 3.1: A simple instance of Clauses and related Features.

is extracted and used in line 5 to calculate the confidence (3.1). 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 4, for the prototype in the Github repository the desired result has been achieved by leveraging the Mongodb aggregation operator, in a single fast and efficient database operation shown in Fig. 3.3. 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.

Algorithm 4 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`.

```
 1: ft ← extract_features(clause);
 2: aggr ← get_relevant_clauses_from_db(ft);
 3: foreach record in aggr do
 4:     occurrencies_found ← record.features_occurrencies
 5:     confidence ← occurrencies_found / size(ft)
 6:     foreach cls in record.clauses do
 7:         if cls not in aggregated and confidence ≥ threshold then do
 8:             aggregated.append(cls);
 9:             aggregate_clauses_greedy(cls, aggregated, threshold);
10: return aggregated
```

3.4.1 Polar Questions

Polar questions in the shape of nominal assertion (excepting for the question mark at the end) are transformed in 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?

can be distinguished by means the dependency `aux(said, Has)` and they 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.

3.4.2 Wh-Questions

Differently from polar questions, for dealing with wh-question we have to transform the question into assertions one can expect as likely answers. To achieve that, after an analysis of several types of questions for each category[3], 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 by `AUX` and `ROOT` dependencies. On the basis of the latters, all remaining chunks are obtained. In regard of likely answers composition, the module *QA Shifter* has the task of recombining the question chunks in a different order, depending on the idiom in exam, considering

[3]Who, What, Where, When, How

also the type of wh-question. Such a 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:

Who could be Biden?

In this case, the chunks sequence will be as it 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 the chunks in a different sequence, by adding also another specific word (Dummy), in order to compose a proper likely assertion like it follow:

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

At this point, by joining all words in such a sequence, a likely assertion to use as query will be the following:

Biden could be Dummy

The meaning of the keyword *Dummy* will be discussed next. In all verbal phrases where `ROOT` is a copular verb[4] (like *be*), i.e., a non-transitive verb but identifying the subject with the object (in the scope of a verbal phrases), the following sequence will also be considered as likely assertion.

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: *Biden is something.* But *something* surely is not what we are looking for as information, but *the elected president of United States* or something else. This means that, within the FOL expression of the query, *something* must be represented by a mere variable and not a ground term. In light of this, instead of *something*, this architecture uses the keyword *Dummy*: during the creation of

[4]The verbs for which we want to have such a behaviour can be defined by a parameter in a configuration file. For further details we refer the reader to the documentation in this work's Github repository.

a FOL expression containing such a word, the Translation Service will impose the Part-of-Speech `DM` to *Dummy*, whose parsing is not expected by the Clauses Builder, thus it will discarded. At the end of this process, as FOL expression of the query we'll have the following literal:

```
Be_VBZ(Biden_NNP(x1), x2)                                   (3.2)
```

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

```
Be_VBZ(Biden_NNP(x1), Of_IN(President_NN(x2), America_NNP(x3)))
```

querying with the (3.2) by using the Backward Chaining algorithm, the latter as result will return back a unifying substitution with the previous clause as it follows:

```
{v_41:  x1, x2:  Of_IN(President_NN(v_42), America_NNP(v_43))}   (3.3)
```

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 as response in natural language.

3.5 Case-study

In this section it is shown a simple instance of Telegram chatbot based on the AD-Caspar architecture, and how it deals with assertions and queries. The output of each interaction, which in this section is made of substitutions, has to be considered aimed to clarify the logical inference behind the logical processing. In a classical chatbot instance, such output should be replaced with proper snippets, custom answers or further questions, following possibly the schema of a dialog system. The Github repository reports detailed instructions about both installation and configuration of a base chatbot instance.

```

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 3.4: AD-Caspar Hi and Low Clauses KBs changes, after assertions.

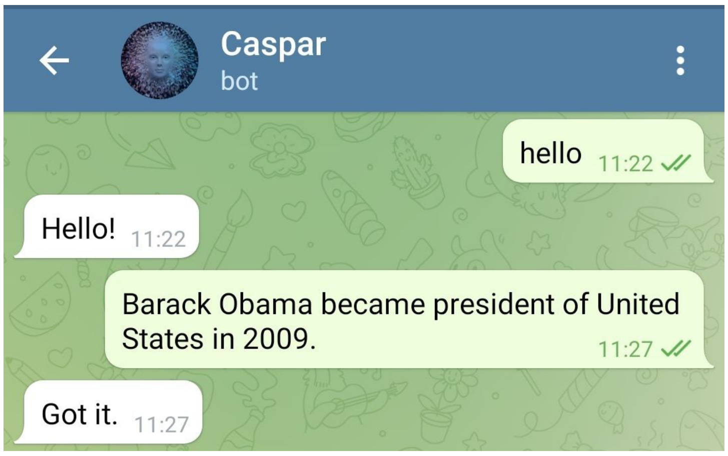

Caspar
bot
hello 11:22
Hello! 11:22
Barack Obama became president of United States in 2009. 11:27
Got it. 11:27

Figure 3.5: Starting a Telegram chat session with an instance of AD-Caspar.

After the agent is started, to begin a new session the user has to provide the keyword *hello* (Fig. 3.5), for making the agent to come out from its idle state, then choose the type of interaction in the chat window as it follows:

1. Assertion, with a dot at the end of sentence (Fig. 3.5).

2. Question, with a question mark at the end of sentence (Fig. 3.6).

3. Direct command or routine definition, without any dot or question mark at the end (Fig. 3.7).

In regard of the first interaction type, the agent's behaviour will be as like as CASPAR, having in addition the parallel population of the Low Clauses KB. As for the second type, its outcome will be a set of logical inferential substitutions containing composite literals (when not returning *False*) as answers to the questions. The outcome of the third type of interaction is the same seen for CASPAR in Chapter 2, considering also meta-reasoning; which means that a Telegram instance can be used also as Internet of Things agent as in Figure 3.7.

3.5.1 Asserting and Querying the Chatbot

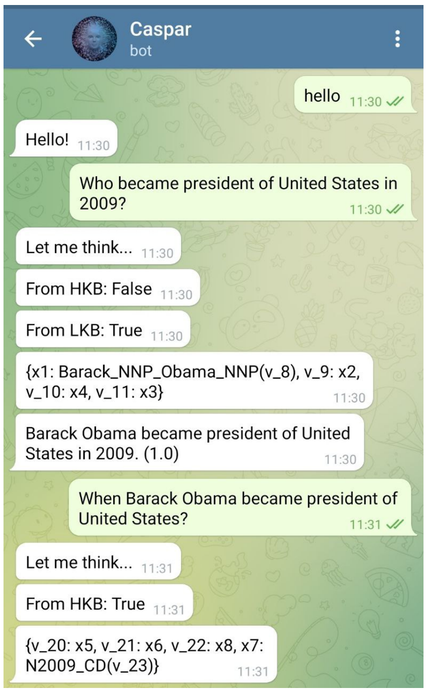
In Figure 3.4 it is shown the content of both High and Low Clauses KB, after the events of Figure 3.5, by means the AD-CASPAR native commands `hkb()` and `lkb()`.

In Figure 3.6 we can see how the chatbot is queried with *wh-questions*, giving back a substitution as result from the High Clauses KB `(From HKB: True)` containing a literal, which is a logic representation of a snippet-result in natural language. In this case the substitution of the variable `x1` is the literal representing the snippet *Barack Obama*, whose words are concatenated together to their Part-of-Speech[5], while `x7` is the representation of the number 2009.

After the chatbot rebooting[6], as we can see in Figure 3.6, the result will be extracted from the Low Clauses KB `(From LKB: True)` taking in account of the confidence threshold (3.1), because the High Clauses KB is still empty `(From HKB: False)`. Such a threshold, depending on the domain, can be changed by modifying the value of the parameter `MIN_CONFIDENCE` in `config.ini` (LKB Section).

[5]Optionally, Part-of-Speech can be excluded by a specific setting in the `config.ini`.

[6]The user has to provide again the word *hello* to start a new session, after the latter is expired.



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

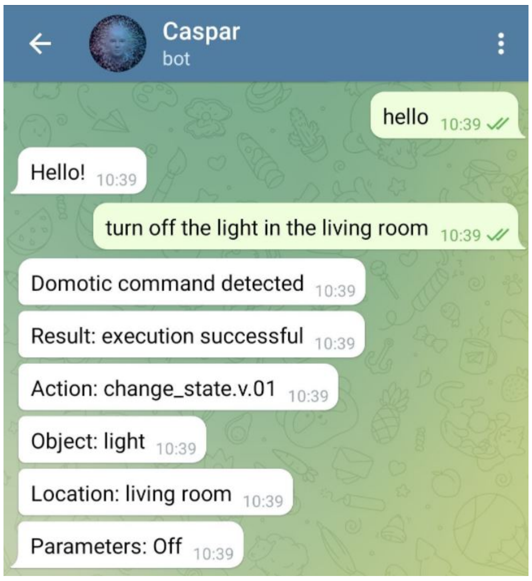
When Barack Obama became president of United States? 11:31

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

Figure 3.6: Querying with *who* and *when* questions, after chatbot rebooting and Low Clauses KB fed.



hello 10:39

Hello! 10:39

turn off the light in the living room 10:39

Domotic command detected 10:39

Result: execution successful 10:39

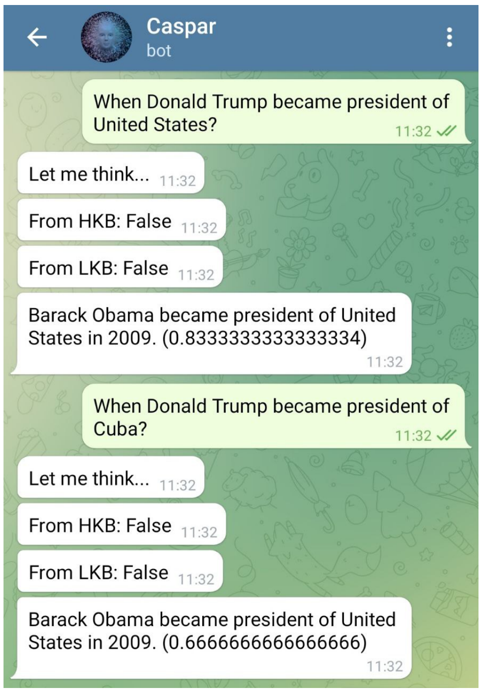
Action: change_state.v.01 10:39

Object: light 10:39

Location: living room 10:39

Parameters: Off 10:39

Figure 3.7: AD-Caspar execution of a Internet of Thing command in a Telegram chat session.



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

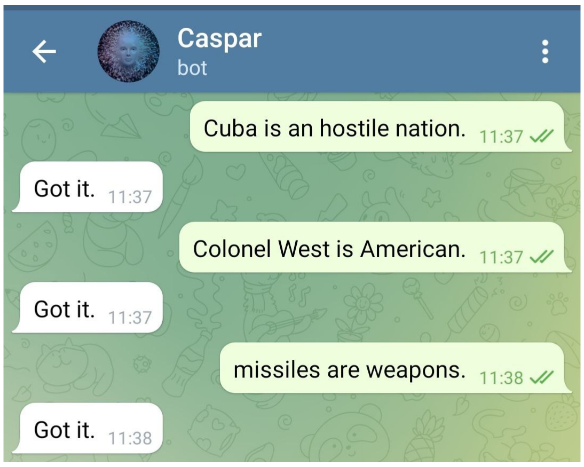
Figure 3.8: Abductive results with confidence threshold = 0.6, after querying with *when* questions.

3.5.2 Failing Queries

In the bot closed-world assumption, the agent can give back only answers unifying with the content of its own knowledge base, otherwise it will return *False*. Optionally, with the value of the parameter `SHOW_REL` set to *True* in the `config.ini`, the closest results can be shown together with their confidence (Fig. 3.8).

3.5.3 Nested Reasoning

Since AD-Caspar inherits most features from its predecessor Caspar, among them there is the one which triggers the expansion of the Clauses KB (both High and Low), due to the assertion of both assignment rules and clause conceptual generalization. Such a behaviour permits what in the prior chapter was defined as *Nested Reasoning*. The *expanded* knowledge base in Fig. 2.5 can be achieved also in a Telegram chatbot: in Figure 3.9 and Fig. 3.10 there are all chatbot interactions required to feed the so-called *Colonel West* case's KB; in Figure 3.11 a successful reasoning is shown, considering the case in which all required clauses are within the High Clauses KB



Cuba is an hostile nation. 11:37

Got it. 11:37

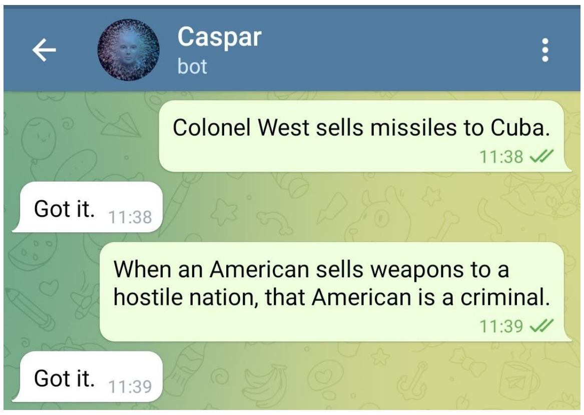
Colonel West is American. 11:37

Got it. 11:37

missiles are weapons. 11:38

Got it. 11:38

Figure 3.9: *Colonel West* KB assertions (part. 1).



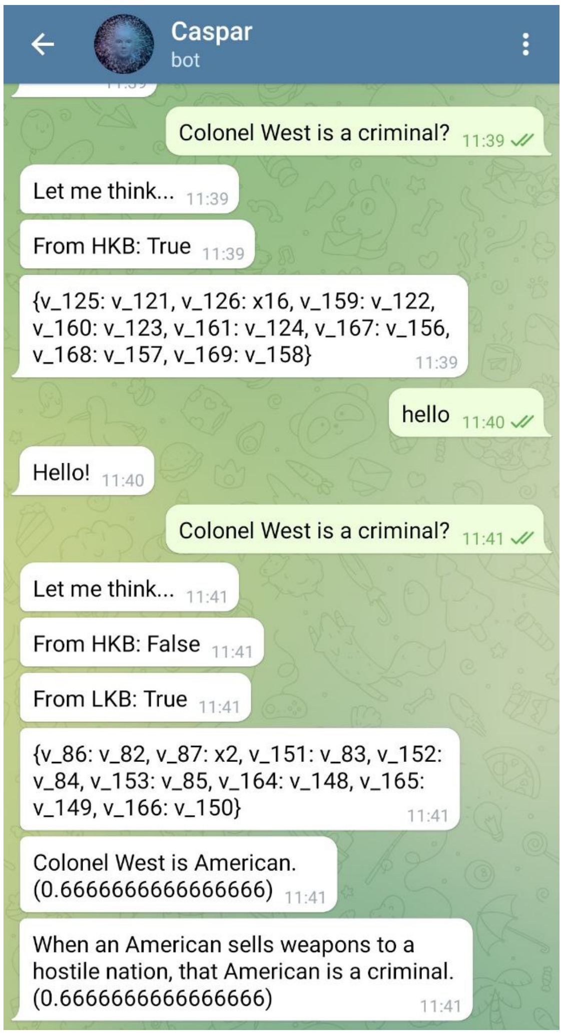
Colonel West sells missiles to Cuba. 11:38

Got it. 11:38

When an American sells weapons to a hostile nation, that American is a criminal. 11:39

Got it. 11:39

Figure 3.10: *Colonel West* KB assertions (part. 2).



```
Colonel West is a criminal? 11:39
Let me think... 11:39
From HKB: True 11:39
{v_125: v_121, v_126: x16, v_159: v_122,
v_160: v_123, v_161: v_124, v_167: v_156,
v_168: v_157, v_169: v_158} 11:39
hello 11:40
Hello! 11:40
Colonel West is a criminal? 11:41
Let me think... 11:41
From HKB: False 11:41
From LKB: True 11:41
{v_86: v_82, v_87: x2, v_151: v_83, v_152:
v_84, v_153: v_85, v_164: v_148, v_165:
v_149, v_166: v_150} 11:41
Colonel West is American.
(0.6666666666666666) 11:41
When an American sells weapons to a
hostile nation, that American is a criminal.
(0.6666666666666666) 11:41
```

Figure 3.11: A successful reasoning on the *Colonel West* KB before and after chatbot rebooting, getting clauses through abduction (with confidence threshold = 0.6) from the Low Clauses KB.

`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

Table 3.2: Realtime cognitive performances (in seconds) of a Telegram chatbot engine based on AD-Caspar, in the case of the question: *Colonel West is American?* and three distinct KBs containing the clauses in Fig. 2.5 (with confidence threshold = 0.6).

(`From HKB: True`); otherwise, after the chatbot rebooting, in the same picture it is shown how the agent gets from the Low Clauses KB all needed clauses through an abductive step with confidence threshold equal to 0.6, then how it achieves a successful reasoning after the transferring of the clauses in the High Clauses KB from the Low one (`From HKB: False, From LKB: True`).

3.5.4 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 an average reasonable response time. In light of above, to address such a issue, since a chatbot relays on the internet, its realtime performances depends firstly on the bandwidth of the internet connection, secondly on the chatbot engine. 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 3.2. The hardware platform the chatbot has been tested on is Intel i7-8550U 1.80Ghz with

8GB RAM. The first column `Knowledge Base` of Table 3.2 refers to three distinct KBs, each with different size, containing the clauses seen in Fig. 2.5. Such clauses are asserted starting from the already seen (in Section 2.3) five sentences related to the *Colonel West* case, namely:

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

Together with each sentence's related clause, assignment rules and clause conceptual generalizations seen in Section 2.3 are also asserted, making possible the expansion of useful clauses from 5 up to 25, thanks to Algorithm 1 and Algorithm 2 seen in Subsection 2.3.6. The first KB, namely *West25*, contains exactly such 25 clauses, while *West104* and *West303* contain either such clauses, but respectively plus 78 and 278 random unrelated clauses. The column `HKB` of Table 3.2 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 (3.1). The third column `LKB+HKB` of Table 3.2 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 value in the first row, which comprises the Mongodb access time and the filling of the High Clauses KB with clauses coming from the Low one, via the `aggregate_clauses_greedy` algorithm. 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 knowledge base. The average timings in the bottom row show how the first row's value is amortized, by compensating the loss with the gain achieved by reasoning on a fewer number of clauses than respect the initial content of all knowledge bases in exam (*West25*, *West104* and *West303*). Intuitively it is expected such bias to be increased for larger knowledge bases, which demonstrates the effectiveness of such approach for two distinct tasks: firstly, to deals with larger knowledge bases considering only the most relevant clauses in the reasoning process; secondly, to permit abduction as pre-stage of deduction in order to give back closer results also in presence of non-successful reasonings.

Chapter 4

SW-CASPAR: Semantic Web based translation of CASPAR

4.1 Introduction

In Chapter 2 we have seen the novel architecture CASPAR and how it can be used to instantiate agents with both cognitive and reactive features. Such agents are able to reason in a process subordinated by a further level of reasoning, namely *meta-reasoning*, in a conceptual space, whose content is made of facts and axioms in first-order logic with the closed-world assumption. An important variation of such architecture, which would make it suitable for different scenarios and represents the main motivation behind this chapter, consists in reasoning over shared *ontologies* in the open-world assumption, i.e., by leveraging the *Semantic Web*.
Ontologies are formal, explicit specification of a shared conceptualization [92]. The Semantic Web, with all its layers and frequent updates, can be considered the very backbone of nowadays ontologies.

The closed-world assumption applies when a system has complete information, like many database applications. On the contrary, the open-world assumption applies when a system has incomplete information. For example, consider a patient's clinical history system. If the patient's clinical history does not include a particular allergy, it would be incorrect to state that the patient does not suffer from that allergy; it is unknown if the patient suffers from that allergy, unless more information is given to disprove the assumption.

On the basis of the above a variation of CASPAR, namely SW-CASPAR, was designed. The latter is capable of parsing IoT commands from natural language, subordinating them by a meta-reasoning over the Semantic Web. An important contribution in the development of such architecture came from the Owlready[93] library, which gives also the chance of reasoning in the *local closed world*[1] as reported in [94].

SW-Caspar relies on the support of a module called *Ontology Builder*, which dynamically creates domain-legacy ontologies serialized in OWL 2, in order to allow for meta-reasoning. Nonetheless, it can can be used also as an alternative stand-alone tool for creating ontologies, in substitution of other state-of-the-art tools, for several reasons that will be clarified in later on. A Python prototype implementation of SW-Caspar is also available as a Github repository.[2]

This chapter is structured as follows: Section 4.2 describes the state of the art of related literature; Section 4.3 shows in detail all the architecture's components and underlying modules; Section 4.4 illustrates the strategy applied for the task of the Ontology Learning; finally, Section 4.5 depicts a case-study of an agent working on a health scenario, making usage of both reasoning and meta-reasoning in the Semantic Web.

4.2 Related works

The architecture explained in this chapter inherits all the *cognitive* IoT features from its predecessor Caspar, so the reader is referred to Chapter 2 for more details about. Hence, this section is focused more on the scope of ontology learning from natural language, which is the main additional contribute of this chapter.

The disruptive growing of textual data on the web, coupled with an increasing trend to promote the semantic web, have made the automatic ontology construction from text a very promising research area. However, manual construction of ontologies is time consuming as well as an extremely laborious and costly process. For this reason, several approaches have been designed to automatize the ontology learning from text, each with different level of human interaction. Such approaches can be divided into two categories: linguistic based and machine learning approaches. Among linguistic based approaches, the authors of [95] use semantic templates and

[1]Owlready reasoning capability in the *local closed world* is limited to a set of individuals and classes.

[2]http://www.github.com/fabiuslongo/sw-caspar

lexico-syntactic patterns such as "NP is type NP" to extract hypernym and meronym relations. Although it is well known that these approaches have reasonable precision but low recall [37]. In order to achieve terms extraction, [96] leverages POS tagging to assign parts-of-speech to each word and a rule-based sentence parser. However many words are ambiguous, so this approach will lead possibly to a low accuracy, without a valid disambiguation strategy. SW-Caspar approach, although similar, makes usage also of a performative disambiguation module described in details in Subsection 2.3.4, extracting also conditional-word based axioms. The authors of [97] make use of a dependency parser to map syntactic dependencies into semantic relations. Such approaches are useful for terms and concepts extraction and also for relations discovery, even though they need to cooperate with other algorithms and/or rules for better performance.

As for machine learning approaches, the system ASIUM [98] adopts agglomerative clustering for taxonomy relations discovery, even if axioms only express subsumption relationship (IS-A) between unary predicates and concepts. The system OntoLearn [80] extracts only taxonomic relations, taking in account of hypernyms from WordNet. The system HASTI [99] builds automatically ontologies from scratch, using logic-based, linguistic-based and statistical-based approaches. It is one of the few systems which try to learn axioms using inductive logic programming, even though they are very general. Furthermore, such a system has the limitation that each intermediate node, in the conceptual hierarchy, has at most two children. Worth of mentioning there is also Text-to-Onto [100], which builds taxonomic and non-taxonomic relations that make use of data mining and natural language processing. Other approaches [101–103] are also interesting, although either they have limitations on the composition of learned concepts or they generate too many hypotheses, making the involved calculation unmanageable.

Besides a large analysis of the state-of-the-art, the authors of [104] discuss on reasons and techniques about the usage of deep neural networks in the Ontology Learning. In these cases, neural network are often hard to train, although in many cases they give better results by using large, domain-related datasets.

4.3 The Architecture

In this section all interacting components of this architecture are explained, which are also depicted in Fig. 4.1 each filled with distinct colours.

SW-Caspar inherits most of Caspar features, apart the *Ontology Builder* instead of the Definite Clause Builder within the *Reactive Reasoner* (central box in Fig. 4.1), and the *Ontology* in OWL 2 instead of the Clauses KB, which is integrated together with a reasoner (Pellet [105]) within the *Cognitive Reasoner* (bottom right in Fig. 4.1). As already pointed out in the introduction, the architecture's name derives directly from its predecessor, namely **S**emantic **W**eb-**C**ognitive **A**rchitecture **S**ystem **P**lanned **a**nd **R**eactive, which synthesized its main theme: the transposition of Caspar in the Semantic Web.

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.

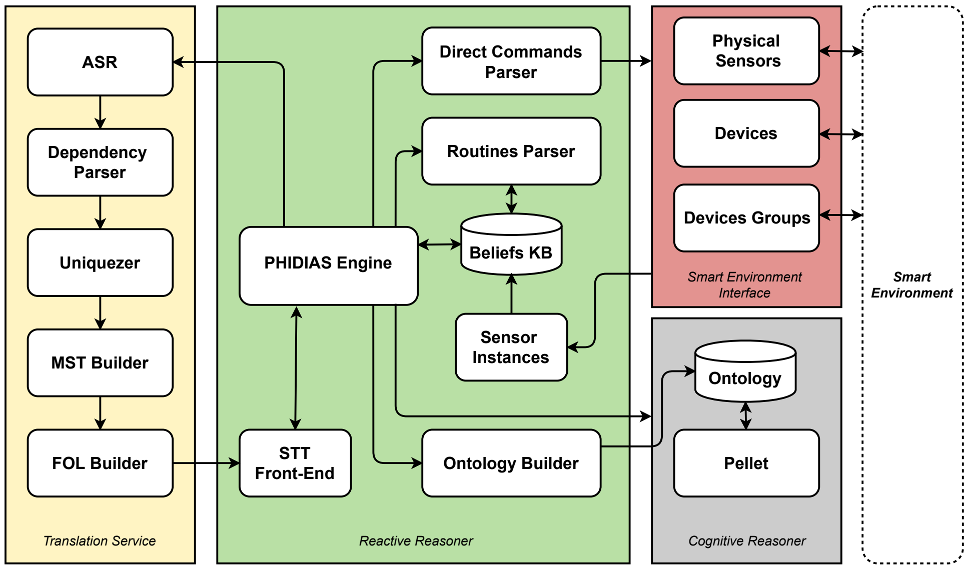


Figure 4.1: The Software Architecture of SW-Caspar.

This architecture's knowledge base (KB) is divided into two distinct parts operating separately, which we will distinguish as *Beliefs KB* and *Ontology*: the former contains information about physical entities which affect and are affected on the agent, whereas the latter contains conceptual information not perceived by agent's sensors, but on which the agent can make inference with the open-world assumption.

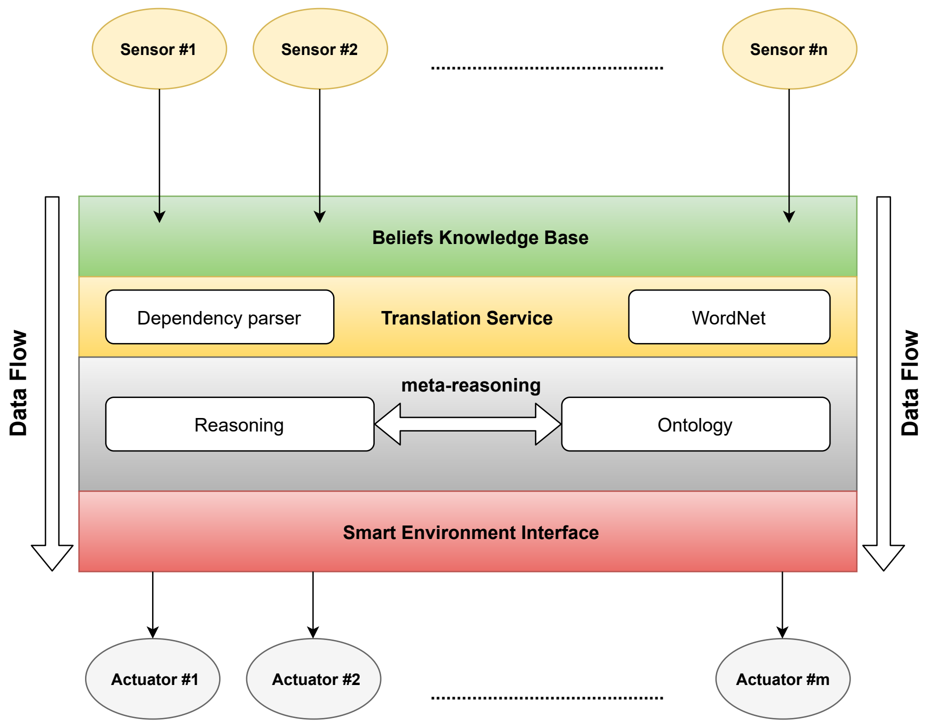


Figure 4.2: The Data Flow Schema of SW-CASPAR

The Beliefs KB provides exhaustive cognition about what the agent could expect as input data coming from the outside world; as the name suggests, such a KB is fed by specific beliefs that can - in turn - activate related plans in the agent's behaviour; Ontology is defined by triples in OWL 2, and is fed by the Ontology Builder within the Reactive Reasoner.

The two KBs represent, somehow, two different kinds of human being memory: the so called *procedural memory* or *implicit memory*[106], made of thoughts directly linked to concrete and physical entities, and 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.

In Fig. 4.2 it is shown a simplified data flow schema of SW-CASPAR. Each sensor in the upper ovals can get information either from a microphone (in the case of vocal commands) or from whatever other kind of device able of capturing physical quantities from the environment. Each information coming from sensors is translated in specific beliefs and stored in the Beliefs KB. In case a belief is not related to an utterance, the former will be sent directly to the Smart Environment Interface, without passing through the Translation Service, where libraries for piloting devices actuators are invoked. Otherwise, in the case of vocal commands, beliefs containing text of utterances will pass through the Translation Service, in order to produce logical forms by leveraging the dependency parser and the lexical resource WordNet. Depending on the utterances content, vocal commands might be also subordinated by meta-reasoning in the conceptual space, before interacting with devices.

4.3.1 The Translation Service

This component (left box in Fig. 4.1) 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 [63]. The reader is referred to Subsection 2.3.1 for a detailed description of such a component.

4.3.2 The Reactive Reasoner

This component (central box in Fig. 4.1) has the task of letting other modules communicate with each other; it also includes additional modules such as the Speech-To-Text (STT) Front-End (which transforms information coming from other modules in beliefs as well as for CASPAR), IoT Parsers (Direct Command Parser and Routine Parser), Sensor Instances, and Ontology 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 [47], which gives Python programs the ability to perform logic-based reasoning (in Prolog style) and lets developers write reactive procedures, i.e., pieces of programs that can promptly respond to environment events. For further detail about such a component, readers are referred to Section 2.3.

4.3.3 The Smart Environment Interface

This component (upper right box in Fig. 4.1) provides a bidirectional interaction between the architecture and the outer world. In [74] we have shown the effectiveness of this approach by leveraging the Phidias predecessor Profeta [75], even with

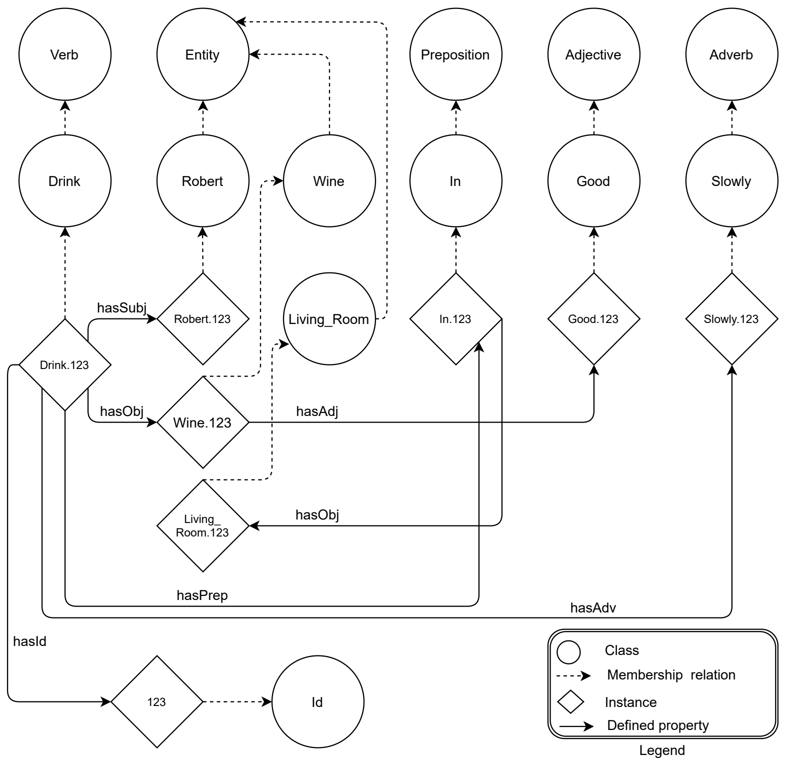


Figure 4.3: A simple instance of LODO ontology related to the sentence 4.2.

a shallower analysis of the semantic dependecies, as well as an operation encoding via WordNet in order to make the operating agent multi-language and multi-synonymous. For more details, the reader is referred to Subsection 2.3.7.

4.3.4 The Cognitive Reasoner

This component (bottom right box in Fig. 4.1) allows an agent to implicitly invoke the Pellet reasoner at runtime, in order to achieve a meta-reasoning subordinating agent's IoT commands. This component comprises the Ontology as well, which is fed by the Ontology Builder within the Reactive Reasoner.

4.4 The Ontology Learning

Differently from most approaches to ontology learning from text, this time we give up the idea that whatever such approach will be it will suffer from the biases related to the ontology of natural language described in Section 1.4. In light of this, firstly we create a FOL representation directly linked to the linguistic features of the sentences in exam, which is provided by the Translation Service component; secondly, whether required, we provide the ontology-specific SWRL [107] rules, whom extend the OWL 2 expressiveness by adding Horn-like axioms. Such rules, as we can see next in this chapter, will contribute to fill the gap between the ontology itself and the expected reasoning in human-like fashion. The core module of the Translation Service, as depicted in Fig. 4.1, is the MST Builder described in Section 2.1. The Translation Service translates a text in natural language into a FOL expression, inheriting the shape from the event-based formal representation of Davidson [63]. Thus, having such a FOL expression, the module STT Front-End, taking in account of the Part-of-Speech which are parts of each label's predicates, will assert specific beliefs triggering the production rules system of the Ontology Builder. The latter has the task of physically creating the OWL 2 domain-legacy ontology containing the triples representing all verbal phrases and their satellite semantic parts (nouns, adjectives, prepositions and adverbs). For its direct derivation from Davidson notation, in this work we defined such a family of ontologies as **L.O.D.O.** (**L**inguistic **O**riented **D**avidsonian **O**ntology). The latter can be considered a foundational ontology, i.e., a specific type of ontology designed to model high-level and domain independent categories about the real world.

The general schema of LODO is quite straightforward. We define *regular verbal phrase* a set of triples in OWL 2 made by the following classes and their instances:

- *Verb*. Each instance of this class represents what comes as verbal phrase in the Davidsonian notation, from the Translation Service. Each individual has the following object properties: *hasId*, having the values of a unique timestamp; *hasSubject* representing the verb subject in the domain of *Entity*; *hasObject*, representing the verb object in the domain of either *Entity* or *Verb* (in the case of embedded verbal actions). Another property, namely *isPassive*, possibly indicates whether a verbal action is passive or not.

- *Id*. Each instance of this class represents a unique timestamp related to a verbal actions. It takes the value of the object property *hasId* from some instance of *Verb*. As in temporal logic, such value can be useful to deal with inconsistency cases: the higher is the Id, the more valid is the related instance of *Verb*, even with such an instance has the property *hasAdverb* equal to the value of *Not*[3]; to be noticed that a proper SWRL axiom could be also used to invalidate such obsolete individuals, in order to let them not participate in a reasoning process. Furthermore, taking into account of the Part-of-Speech, it can also be introduced an object property *hasTime* of such instance, in order to express the tenses of the verbal actions (Present, Past Tense, Past Participle, Gerund) than respect the timestamp.

- *Entity.* Each instance of this class represents an entity referenced by the object property either *hasSubject* or *hasObject.* Compound nouns are possible concatenated in order to form a single individual, of expressed through the property *hasCpmd* referencing other entities.

- *Adjective.* Each instance of this class takes the values of the object property *hasAdj* of some instance of *Entity.*

- *Preposition.* Each instance of this class represents a preposition and it is referenced by the object property *hasPrep* of some instance of either *Verb* or *Entity.* Moreover, each of such instance has the object property *hasObject* referencing some instance of *Entity*

- *Adverb.* Each instance of this class represents an adverb and has the values of the object property *hasAdv* of some instance of *Verb.*

Together with such taxonomic and non-taxonomic relations, LODO comprises also a group of axioms (or part of them) implicitly created by SW-Caspar, with the aim of increasing the chances of reasoning/graph matching. Such axioms are summarized as follows:

- *Assignment Rules.* Such rules are implicitly asserted in the presence of a copular verb Cop representing expression (possibly identified also by its synset), which will be useful in order to assign the class membership of the verb's object to its subject. Formarly, the following expression:

$$\mathsf{Subject{:}POS}(\mathsf{x}_1) \land \mathsf{Cop{:}POS}(\mathsf{e}_1, \mathsf{x}_1, \mathsf{x}_2) \land \mathsf{Object{:}POS}(\mathsf{x}_2)$$

[3]Negations are treated as whatever adverb.

where each predicate has its own POS tag. Such a expression will trigger the assertion of the following SWRL rule (omitting the POS which can be also included in classes labelling):

$$Subject(?x) \rightarrow Object(?x) \tag{4.1}$$

- *Legacy Rules.* Such rules are implicitly asserted together with the assignment rules, to let a copular verb's subject inherit both adjectives and prepositions properties of the verb's object. Formally, considering (4.1), the corresponding legacy rule will be the following:[4]

$$Subject(?x2),\ Object(?x1),\ hasAdj(?x1,\ ?x3),\ Adjective(?x3) \rightarrow hasAdj(?x2,\ ?x3)$$

- *Deadjectival Rules* (optional). In the presence of an instance of *Adjective*, Such rule assert a new deadjectivated instance of the latter as new membership of the adjective related noun. Formally:

$$Entity(?x1),\ hasAdj(?x1,\ ?x2),\ Adjective(?x2) \rightarrow Entity(?x2)$$

- *Deverbal Rules* (in progress of development). In the presence of an instance of *Verb*, such rules assert a new deverbalized instance of the latter having the same entities of the former.

- *Implicative Copular Rules.* Such rules take in account of implicative axioms, possibly coming from the Translation Service in FOL Davidsonian notation and containing a single copular verbs in the implication's head. They are useful to infer new memberships of the initial sentence subject, which must be present also in the body. The production rule of the Ontology Builder for such rules assertion matches with the following pattern:

$$\mathsf{Subject}(\mathsf{x}_{body}) \wedge \ldots \Longrightarrow \mathsf{Subject}(\mathsf{x}_{subj}) \wedge \mathsf{Object}(\mathsf{x}_{obj}) \wedge \mathsf{Cop}(\mathsf{e}_{cop},\ \mathsf{x}_{subj},\ \mathsf{x}_{obj})$$

[4]Similarly for preposition, by changing *hasAdj* with *hasPrep*.

As shown above, the label Subject must be in both left and right-hand side of the FOL expression; otherwise, in order to replace possible pronouns that invalidate the pattern, an anaphora resolution pre-processing could be required before the Translation Service pipeline in Fig. 3.1. Cop is the label of a copular verb which will be absorbed, permitting the formal assertion of the following pattern:

$$Subject(?x_{obj}),\ \ldots \rightarrow Object(?x_{obj})$$

Any other implicative FOL expression with non-copular verb in the head will be discarded, due to the non-monotonic features of SWRL.

- *Value Giver Statements* (optional). Such a statement contributes to give a value to a data property *hasValue* related to a specified individual, which is parsed by the Ontology Builder by matching the following pattern of beliefs:

  GND(FLAT, X, Y), ADJ(FLAT, X, "Equal"), PREP(FLAT, X, "To", S),
  VALUE(FLAT, S, V)

  The first argument (FLAT) of all beliefs is for distinguishing non-implicative expressions from implicative ones, and also either right- or left-hand side. The belief GND is related to a ground term with label Y coming from a FOL expression, which corresponds to a couple of class-individual in the ontology. The beliefs ADJ and PREP specify a lexical content (*Equal* and *To*) among their arguments, while VALUE specifies the value that must be given to the individual corresponding to label Y. The property *hasValue* might be involved in comparison operations in the composition of a SWRL axiom.

- *Values Comparison Conditional* (optional). Such conditionals are parsed from sentences similarly to the *Value Giver Statement*, but they will take place within the body of implicative copular rules.

For instance, considering the following sentence:

Robert slowly drinks good wine in the living room. (4.2)

The Translation Service will give the following FOL expression as result:

$$\text{Robert:NNP}(x_1) \wedge \text{wine:NN}(x_2) \wedge \text{drink:VBZ}(e_1, x_1, x_2) \wedge \text{slowly:RB}(e_1) \wedge \text{good:JJ}(x_2) \wedge \text{in:IN}(e_1, x_3) \wedge \text{living:NN}(x_3) \wedge \text{room:NN}(x_3)$$

Then, the STT Interface on the basis of POS and arguments cardinality will produce the following set of beliefs:

PREP(FLAT, e1, In, x3), ACTION(ROOT, FLAT, Drink, e1, x1, x2), GND(FLAT, x3, Living), GND(FLAT, x1, Robert), GND(FLAT, x2, Wine), GND(FLAT, x3, Room), ADV(FLAT, e1, Slowly), ADJ(FLAT, x2, Good)

Since implicative copular rules can be applied only one time for sentence, the label ROOT as first argument of the belief ACTION has the aim of distinguishing the main verbal action from possible other verbs in the same sentence. The final step is done by the Ontology Builder (within the Reactive Reasoner in Fig. 4.1), whose production rules will match in a specified order such beliefs, in order to interface with the Owlready libraries and to create the OWL 2 ontology.
In Fig. 4.3 such an ontology is depicted. The classes in the upper level (*Verb*, *Entity*, *Preposition*, and *Adverb*) are meant to be subclasses of *Things*; the remaining ones in the circles are subclasses of the former. The diamond shaped boxes are individuals whose label contains also a reference of verbal action's Id (the value *123* is not indicative). The latter is also an individual itself, being instance of the class *Id*.
Due to the presence of the adjective *good* related to the individual *wine*, optionally one might activate the SW-Caspar deadjectival generation rule; then, invoking an OWL reasoner, the individual *wine* will achieve the new deadjectivated membership *good*.

4.5 Case-Study

In this section, first of all it will be shown how the Ontology Builder module of SW-Caspar deals with the same KB seen in Subsection 2.4.3, namely the *Colonel West* case KB, which is achieved starting from the following sentences:

- *Colonel West is American.*
- *Cuba is a hostile nation.*

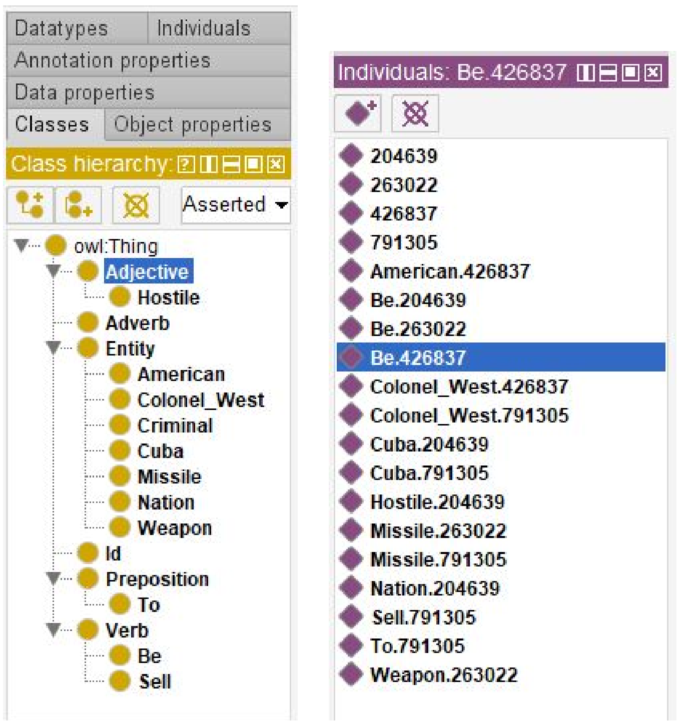


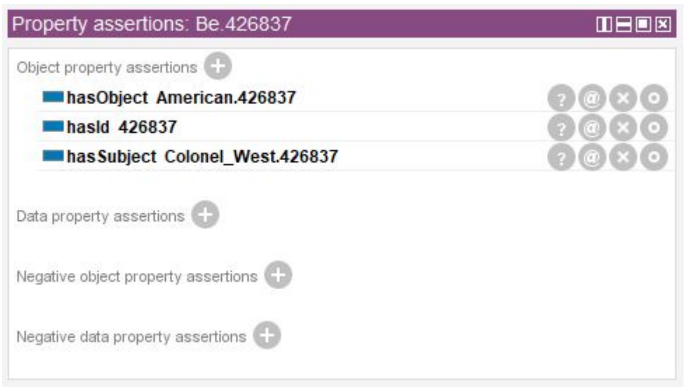
Figure 4.4: The LODO taxonomic relations and instances from the *Colonel West* KB.

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

In Fig. 4.4 all classes and instance are shown in the Protégé environment. In particular, Fig. 4.5 shows all non-taxonomic relations related to the individual `Be.426837`: `hasSubject`, `hasObject` and `hasId`. In Fig. 4.6 all generated axioms are shown, while in Fig. 4.7 two new inferred[5] membership related to the individual `Colonel_West.791305` are shown, which are the representations of *Colonel West is American* and *Colonel West is a criminal*. The latter is the sentence-query already seen in Subsection 2.4.3.

Next, a simple case of ontology building and reasoning/meta-reasoning is presented, showing how IoT agents based on SW-Caspar are able to parse natural

[5]With the reasoner Hermit or Pellet.


Property assertions: Be.426837

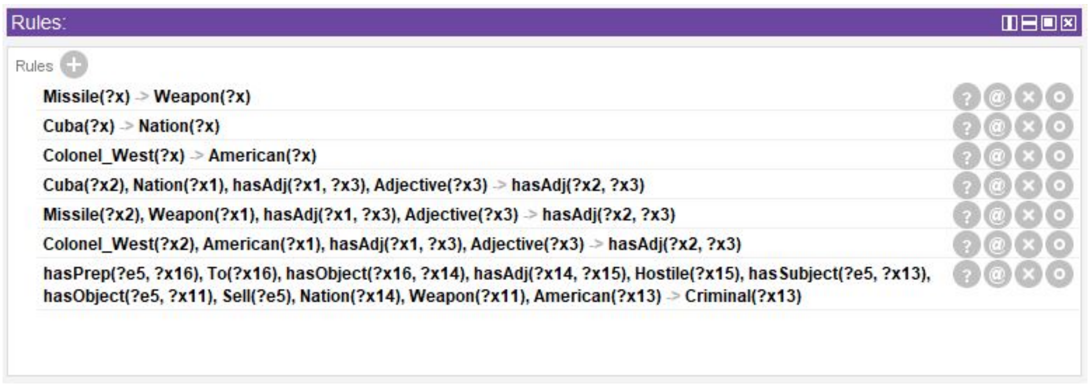
Object property assertions
- hasObject American.426837
- hasId 426837
- hasSubject Colonel_West.426837

Data property assertions

Negative object property assertions

Negative data property assertions

Figure 4.5: The LODO non-taxonomic relations related to the individual `Be.426837` from the *Colonel West* KB.


Rules:

Rules
- Missile(?x) -> Weapon(?x)
- Cuba(?x) -> Nation(?x)
- Colonel_West(?x) -> American(?x)
- Cuba(?x2), Nation(?x1), hasAdj(?x1, ?x3), Adjective(?x3) -> hasAdj(?x2, ?x3)
- Missile(?x2), Weapon(?x1), hasAdj(?x1, ?x3), Adjective(?x3) -> hasAdj(?x2, ?x3)
- Colonel_West(?x2), American(?x1), hasAdj(?x1, ?x3), Adjective(?x3) -> hasAdj(?x2, ?x3)
- hasPrep(?e5, ?x16), To(?x16), hasObject(?x16, ?x14), hasAdj(?x14, ?x15), Hostile(?x15), hasSubject(?e5, ?x13), hasObject(?e5, ?x11), Sell(?e5), Nation(?x14), Weapon(?x11), American(?x13) -> Criminal(?x13)

Figure 4.6: The LODO rules from the *Colonel West* KB.

language commands and reason about their execution in the open-world assumption. Inspired by [94], where the author shows a use-case consisting in reasoning on drugs contraindications in presence of food intolerances, in this case-study we mainly focus on clinical information of patients and known issues of drugs.

We suppose to provide a hospital with a (semi-)automatized drug distribution system based on natural language recognition, or even to build a robot performing such a task. In such a scenario, we suppose to define one or more agents assisting doctors in the decision-making task related with the administration of drugs, on the basis of known drug's issues and clinical picture of patients. In order to address such a task, we extended the Smart Environment Interface of SW-Caspar with the following set of two production rules given by Phidias,[6] which considers a known

[6]For the sake of the shortness we consider a simplified form of rules.

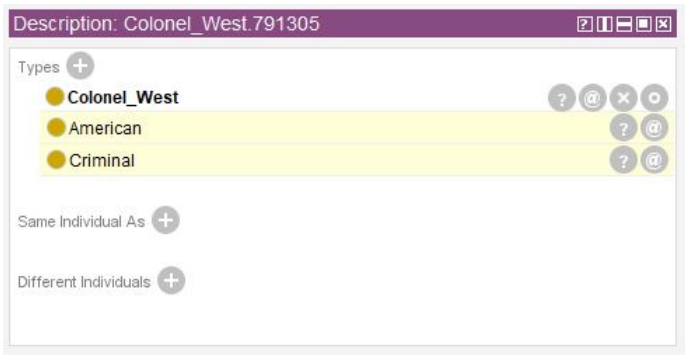


Figure 4.7: Inferred LODO membership after reasoning from the *Colonel West* KB.

contraindication of the drug *Rinazina* for hypertensive patients:

$$+\mathsf{INTENT}(\text{"Rinazina"}, \mathsf{T})\ /\ \mathsf{eval\_sem}(\mathsf{T}, \text{"Hypertensive"}) >> [\mathsf{say}(\text{"Nope. Patient is hypertensive"})] \tag{4.3}$$

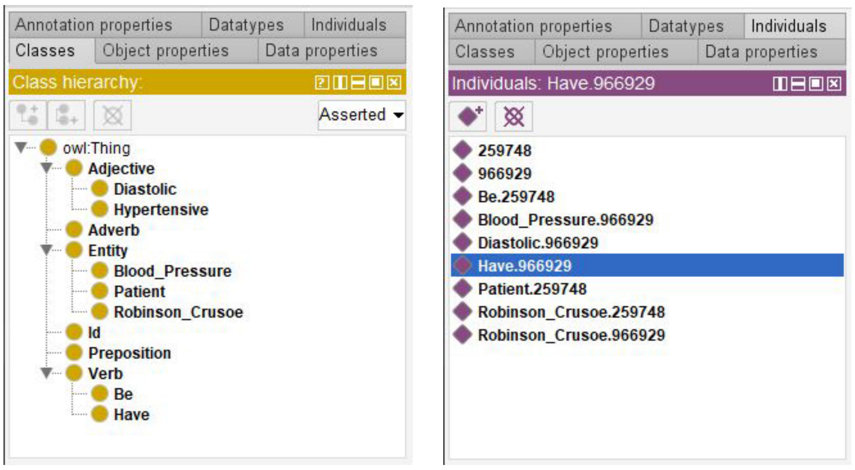
$$+\mathsf{INTENT}(\text{"Rinazina"}, \mathsf{T}) >> [\mathsf{exec\_cmd}(\text{"Rinazina"}, \mathsf{T}), \mathsf{say}(\text{"Execution successful"})] \tag{4.4}$$

As seen in Fig. 2.4, each production rule (which begins with +) is expressed in a Prolog-like sintax, where the left hand-side encompasses a belief we want the rule to match with (INTENT), conditioned by other beliefs (in this case the Active Belief eval_sem); the right-hand side encompasses in square brackets the plan to execute when the rule is triggered.
We now generate the domain-legacy ontology exploiting LODO with all the required information about the patient *Robinson Crusoe* and his health disorders starting from the following sentences:

- *Robinson Crusoe is a patient.*
- *Robinson Crusoe has diastolic blood pressure equal to 150.*
- *When a patient has diastolic blood pressure greater than 140, the patient is hypertensive.*

The first sentence is parsed as both regular verbal phrase and assignment rule, whose related classes and individuals are shown in Fig. 4.8. A unique timestamp is adopted for all the elements of the same verbal phrase.

Together with classes and individuals, the corresponding legacy rules are also asserted (first and third rule of Fig. 4.10) allowing *Robinson Crusoe* to inherit all the features of the individual *patient* in analogous way as the speaker's knowledge flow in the scope of the same discourse. The developer might also provide customized IRI,[7] either manually or automatically, by means of possible pre-compiled association tables.



```
owl:Thing
  Adjective
    Diastolic
    Hypertensive
  Adverb
  Entity
    Blood_Pressure
    Patient
    Robinson_Crusoe
  Id
  Preposition
  Verb
    Be
    Have
```

```
259748
966929
Be.259748
Blood_Pressure.966929
Diastolic.966929
Have.966929
Patient.259748
Robinson_Crusoe.259748
Robinson_Crusoe.966929
```

Figure 4.8: The LODO taxonomic relations and instances of the case-study

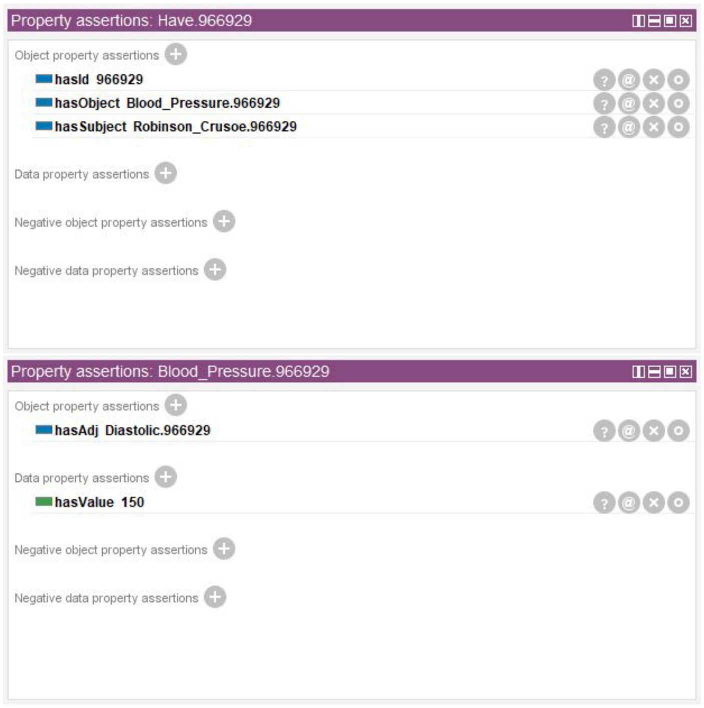
As shown in Fig. 4.10, the second sentence is parsed as regular verbal phrase containing also a value giver statement (depicted in Fig. 4.9).

The third sentence is parsed by the Translation Service as FOL expression containing an implication such as:

$$\mathsf{Have{:}VBZ}(e_1, x_1, x_2) \wedge \mathsf{Blood{:}NN}(x_2) \wedge \mathsf{Patient{:}NN}(x_1) \wedge \mathsf{Pressure{:}NN}(x_2) \wedge \mathsf{Than{:}IN}(x_2, x_5) \wedge \mathsf{Diastolic{:}JJ}(x_2) \wedge \mathsf{Great{:}JJR}(x_2) \wedge \mathsf{140{:}CD}(x_5) \Longrightarrow \mathsf{Patient{:}NN}(x_3) \wedge \mathsf{Hypertensive{:}JJ}(x_4) \wedge \mathsf{Be{:}VBZ}(e_2, x_3, x_4)$$

In the presence of such a expression, the Ontology Builder will asserts an implicative copular rule containing a values comparison conditional (the second entry in Fig. 4.10), without creating any individuals linked together by the same timestamp. At the end of the ontology building process, the agent is ready to parse a command containing the following text:

[7]Internationalized Resource Identifier


Property assertions: Have.966929

Object property assertions

hasId 966929

hasObject Blood_Pressure.966929

hasSubject Robinson_Crusoe.966929

Data property assertions

Negative object property assertions

Negative data property assertions

Property assertions: Blood_Pressure.966929

Object property assertions

hasAdj Diastolic.966929

Data property assertions

hasValue 150

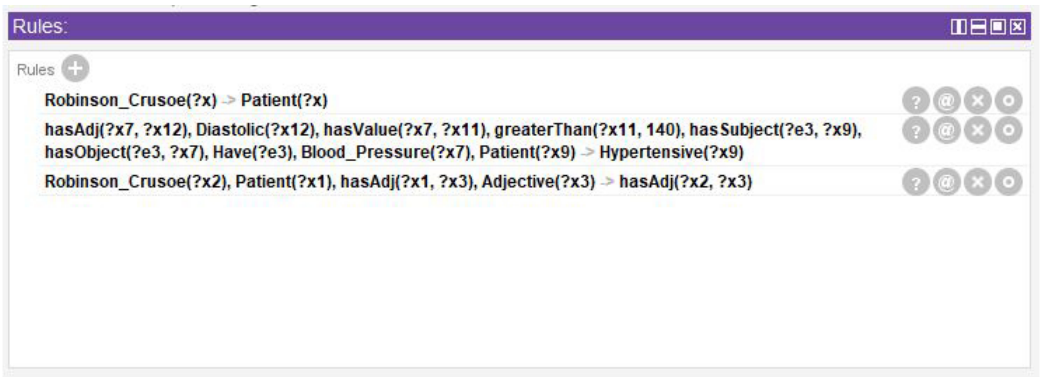
Negative object property assertions

Negative data property assertions

Figure 4.9: The LODO non-taxonomic relations of the case-study

Give Rinazina to Robinson Crusoe

As the agent invokes the Pellet reasoner and checks for the membership of *Robinson Crusoe* to the class *Hypertesive* (as well as in Fig. 4.11 with Protégé), after a successful meta-reasoning of the Active Belief eval_sem, the production rule 4.3 will match with the content of Belief KB and the command will be discarded with an objection message from the agent. Otherwise, the production rule 4.4 will match and the command will be executed without any objection message. Of course, the meta-reasoning could involve also more complex queries expressed in SPARQL language and even in local closed world.



Rules:

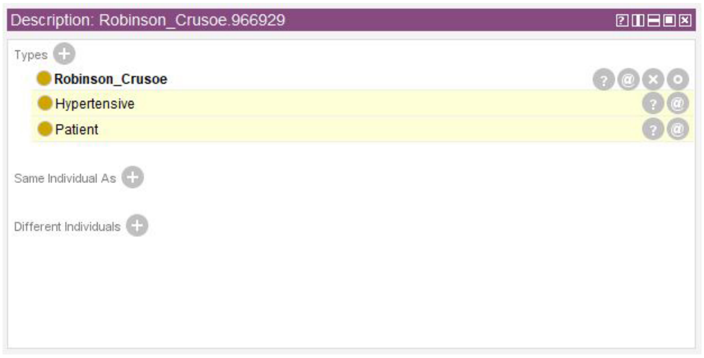
Rules

Robinson_Crusoe(?x) -> Patient(?x)

hasAdj(?x7, ?x12), Diastolic(?x12), hasValue(?x7, ?x11), greaterThan(?x11, 140), hasSubject(?e3, ?x9), hasObject(?e3, ?x7), Have(?e3), Blood_Pressure(?x7), Patient(?x9) -> Hypertensive(?x9)

Robinson_Crusoe(?x2), Patient(?x1), hasAdj(?x1, ?x3), Adjective(?x3) -> hasAdj(?x2, ?x3)

Figure 4.10: The LODO rules of the case-study



Description: Robinson_Crusoe.966929

Types

Robinson_Crusoe

Hypertensive

Patient

Same Individual As

Different Individuals

Figure 4.11: Inferred LODO membership after reasoning.

Chapter 5

Overall Evaluation

This chapter is focused on a overall evaluation of the cognitive architectures reported in the prior chapters, especially whether literally they can be considered *cognitive systems*. In general, to evaluate a new model, in the case its performances can be quantified somehow, a comparison is usually made with gold standards or other model's scores from the state-of-the-art (accuracy, precision, response times, etc.). Such an evaluation was achieved in the case of CASPAR and AD-CASPAR considering timings related to specific functions, even if without taking in account of any gold standard but a reasonable human-fashioned response time. In the case of a more general evaluation, without considering just any specific functions, it is required to shift to wider evaluations encompassing more and different criteria. In the scope of of biological inspired artificial systems, several general criteria have been proposed to characterize the design of biologically plausible models. In this regard, the roboticist Barbara Webb[108] identified a list of dimensions for the characterization of different design aspects of bio-inspired models, which are:

- **Biological relevance**: it indicates if a computational model can be considerate *close* to a given biological system taken as a source of inspiration.
- **Level**: it indicates how much the model takes in account of internal structures, than respect to its biological counterpart.
- **Generality**: it indicates how many different biological systems can be represented by a model.
- **Abstraction**: it indicates how many details are explicit in the artificial model, than respect to the natural systems taken as a source of inspiration. It

shouldn't be confused with the "level" dimension, since a more abstract model could also be more complex than the corresponding lower-level brain model.

- **Structural accuracy**: it estimates the similarity between the mechanisms underlying the behaviour of an artificial model with respect those of the target biological system.
- **Performance match**: it estimates the performances of the model than respect the ones of the target biological system.
- **Medium**: it refers to the physical medium used to implement the model.

In his recent book[19], Lieto proposes a much more synthetic list of elements that subsumes some of Webb's dimensions and that, additionally, can be applied not only to biological inspired systems but also to cognitively inspired ones. Such a list is defined as *Minimal Cognitive Grid* and encompasses the following evaluation criteria:

- **Functional/Structural Ratio**: this dimension synthesizes and subsumes the "biological relevance" and "structural accuracy" individuated by Webb by enabling, in principle, the possibility of performing both a quantitative and qualitative comparison between different artificial systems (whether they are cognitively inspired or not). The lower the ratio, the better.
- **Generality**: this features aim at evaluating how general is a given system/architecture and whether it can be used to simulate a set of cognitive functions and not just a narrow one.
- **Performance match**: this dimension involves a direct comparison between natural and artificial systems in terms of the obtained results for specific or general tasks. Along this line, Lieto proposes two additional specific requirements that refers to such an aspect:
    1. The analysis of system errors (which, in human-like artificial systems, should be similar to those committed by humans).
    2. The execution time of the tasks (which, again, should converge toward human performances).

Another interesting proposal within the field of cognitive AI and cognitive modelling research is represented by the so-called *Standard Model of Mind*[1], based on an abstraction from some of the most adopted cognitive architectures, namely SOAR[14], ACT-R[109] and SIGMA[110]. This abstraction is based on the consensus reached in the community over decades of research, which has been grouped into different levels of analysys:

1. **Structure and Processing Mechanisms**: it concerns that processing in human-like architecture is assumed to be based on a small number of task-independent modules, and should support both serial and parallel information processing mechanisms. All three architectures taken as source of inspiration converge towards the necessity of distinguishing between a Long-Term Declarative Memory and a Procedural one, as well as a control interface for the mutual interaction between the Procedural Module, Perception/Motor modules and Declarative Memory.

2. **Memory and Content**: in this case the main point of convergence regards the integration of hybrid symbolic-subsymbolic representations.

3. **Learning Processes**: in this case, the elements of convergence regard the following assumptions:

   (i) All types of long-term knowledge should be learnable from a human-like architecture.

   (ii) Learning is seen as an incremental process typically based on some form of a backward flow of information through internal representations of past experiences.

   (iii) Learning over times scales is assumed to arise from the accumulation of learning over short-term experiences.

4. **Perception and Motor Mechanisms**: perceptual and motor modules are assumed to be modality specific (e.g. auditory, visual, etc.) and associated with specific buffers for the access to the working memory.

[1]Later on called *Common Model of Cognition*.

In the next subsections, the Minimal Cognitive Grid (MCG) and the Standard Model of Mind (SMM) will be used to evaluate the novel cognitive architectures proposed in this dissertation.

5.1 MCG Evaluation

In this section, all *Minimal Cognitive Grid* criteria are used to evaluate the novel cognitive architectures seen in the prior chapters, on the basis of which it is evident that such a family can be considered made of cognitive systems.

5.1.1 Functional/Structural Ratio

The architecture of CASPAR encompasses some features which also come from different assumptions about mental or brain processes. The first assumption concerns the BDI framework Phidias on which CASPAR has been built, whose concept is inspired by Bratman[7] theory on practical human reasoning.

Another assumption language-based comes from the component Translation Service (left box in Fig. 2.1, 3.1 and 4.1), which fulfills the function of the brain areas that are responsible for understanding words and sentences. Such brain areas are mainly located in two regions, in the left side of the brain, they are connected by nerves, and together they form a network that provides the hardware for language. The Translation Service is also responsible of modelling the conceptual space used for the meta-reasoning, where each predicates's label can also be a WordNet synset selected with a Disambiguation technique explained in Subsection 2.3.4. Composite common sense representations relies on the Principle of Compositionality formally formulated by Partee[73], as seen in Subsection 2.3.1, where *"The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined."*. Such a representation is built in its basic components on *exemplar-based*[111] categorization, because it chooses the best candidate among all synsets related examples, optionally using also other fields of synsets records or various combinations of them.

The cerebellum functions, which in human brain plays an important role in motor control, are simulated by the Smart Environment Interface (Fig. 2.1) working together with the Beliefs KB.

In the case of AD-Caspar, the two layers of the Clauses KB simulate *Short Term Memory* (High Clauses KB) and *Long Term Memory* (Low Clauses KB), in order to achieve abduction as pre-stage of deduction and to deals with large knowledge base.

In light of the above, it is clear that Caspar and its derivative architectures have both functional and structural features.

5.1.2 Generality

As explained in Subsection 2.3.2, for what concerns language the base assumption of Caspar[2] is to be as general as possible, because instead of working with a limited set of recognized sentences related to distinct functions, the integration of the dependency parser permits higher levels of expressiveness, which is required to build domain-related conceptual spaces.

5.1.3 Performance match

As for possible errors produced by instances of Caspar and its derivative architectures, as explained in Subsection 2.3.1, as long as dependencies are treated properly by some production rule in the MST Builder, the accuracy of the conversion from natural language to logical form can be clearly considered equal to the accuracy of the dependency parser, which in the case of spaCy is 90% for the English idiom. The usage of out-of-commons-sense utterances, might lead to unexpected logical form which would make reasoning failed.

In regard of time performances, in Fig. 2.5 it was shown how Caspar reaches very promising timing performances, on devices with limited hardware resources like Raspberry. In the case of AD-Caspar, Table 3.2 shows also good timing performances featuring a balancing between Low and High Clauses KB when KB size increases.

[2]Which is in common also with both AD-Caspar and SW-Caspar.

5.2 SMM Evaluation

In this section, it is addressed a comparative evaluation between convergence criteria related to the *Standard Model of Mind* and CASPAR's family, even if, as the authors themselves admit, such a model is currently underspecified. In any case, such a model represents a starting point, a platform for developing high-profile research in both AI and (computational) Cognitive Science.

5.2.1 Structure and Processing Mechanisms

In regard of such criteria, CASPAR and its derivative architectures are endowed of asynchronous sensors instances which permit both parallel and serial information processing mechanisms (Fig. 2.3). Furthermore, as seen in Section 2.1, such cognitive architectures are endowed also with a KB divided in Declarative Memory and Procedural one, and a specific interface (the Smart Environment Interface seen in Section 2.3.7) for direct interaction with Perception/Motor modules.

5.2.2 Memory and Content

The working memory of CASPAR's family has a native integration of hybrid symbolic-subsymbolic representations. The subsymbolic representation comes from the neural dependency parser (in this case spaCy), which is trained for the classification of common-sense semantic relations (dependencies) between words in natural language utterances. In regard of the symbolic representation, it is achieved in the shape of beliefs, each of whom interact with a system of production rules, and in the shape of clauses in first-order logic. As explained in Section 2.2, such two group of symbolic information can interact with each other in a (meta-)reasoning process.

5.2.3 Learning Processes

As for such a criteria, apart CASPAR which has not a long-term memory but just a volatile one, for both AD-CASPAR and SW-CASPAR all types of long-term knowledge is learnable from a human-like architecture, which is made of the pipeline Translation Service. The learning process, which at this stage of the designs don't take in account of tout court *machine learning* and similar ones approach[3], is based on information (in the form of clauses) provided by past interactions based on natural language. Moreover, especially for AD-Caspar whose Clauses KB is splitted in two layers representing Short- and Long-Term memory, learning over times scales is arised from the accumulation of learning over short-term experiences.

5.2.4 Perception and Motor Mechanisms

All cognitive architectures derived from Caspar support fully any kind of interaction with perceptual and motor modules, through the Sensor Instances, which will have their specific buffers for the access to the working memory. As long as there will be production rules taking in account of beliefs related to such external perceptions, specific plan containing high-level code for reactive interaction with the environment will be triggered.

[3]Although Sensor Instances can be fully compatible with them.

Chapter 6

Conclusions

In this research, the road of natural language processing was chosen and pursued, as a means for achieving mutual understanding between human and machine, with the aim of decision-making coadiuvation. The first result comes with the design of the cognitive architecture CASPAR, described in Chapter 2, from which it is possible to instantiate *cognitive agents* capable of both reactive and cognitive reasoning. It works by using a knowledge base divided into two distinct parts (Beliefs KB and Clauses KB) which can also interact one another in a meta-reasoning process. In particular, the more the Clauses KB increases, the more CASPAR's cognitive features improve, due to an implicit and native capability of inferring monotonically additional axioms from its own KB. Thanks to novel algorithms leveraging existing knowledge or producted by themself, CASPAR is able to transcend the limit of the known Backward Chaining algorithm given by the nested semantic notation, which is also as highly descriptive as compact. Moreover, such a architecture is able to parse complex direct IoT commands and routines, letting the user to easily customize its own interfacing with environments, with whatever Speech-to-Text engine, at the condition that logical expressions producted by CASPAR will be coherent as long as parsed natural language utterances have syntactic structure known by the dependency parser; which means that *out of common sense* expressions or particular idiomatic ones could lead to unexpected results, depending on the datasets the dependency parser has been trained on (which can be considered an *acceptable* limit for this architecture).

In complex production rule systems, when these rule bases grow in size, they possible become very hard to understand and maintain, due to their *flat* nature in which basically it is hard to define or identify modules. To address such a issue, CASPAR was divided in three distinct hierarchical levels of production rules: the first and second level[1], which are very compact, are related to the semantic of language and does not never grow in size depending on domain. The third level[2], that potentially grows together with domains and functional properties of instantiated agents, is related to the interaction with environments. In this level, all rules might be parameterized in a very extensive way through the usage of Phidias *Active Beliefs*, including also possible interaction with databases in order to give values to such beliefs only when requested, and then make the operating range of matching chances at run-time expanded. In this way, since all production rules are loaded into the agent memory at start-up, by limiting the rule base size through parametrization, the main inner Phidias loop will surely work faster than respect to consider each combination of couples device-command explicitly stated inside rules, therefore addressing efficiently rule bases size increase. Indeed, such a rule parametrization should also minimize accesses to database.

Since a successful reasoning achieved with CASPAR pass through only unifying predicates (via the Backward Chaining algorithm), the natural evolution is the one which takes in account of closer results as well, in cases of non-successful reasoning: that's what was achieved with AD-CASPAR. In Chapter 3 such a cognitive architecture is described, which inherits all CASPAR features plus the chance of abductive reasoning as pre-stage of deduction and the inclusion of a Telegram chatbot. Furthermore, AD-CASPAR is able to rephrase wh-questions into likely assertions one can expect as likely answer, thanks to a production rule system which leverages also a dependency parser. The combination of Translation Service/Definite Clause Builder and QA Shifter makes the chatbot proposed in 3.5 easily scalable on the knowledge we want it to deals with, because the user has to provide just new sentences in natural language at runtime, like in a normal conversation. The Python prototype proposed is at a very early stage, although it is already able to reason on a wide range of knowledge bases. For an extensiver usage, a dialog system must be still designed, in order to extract snippet-result in natural language from the substitutions shown in Fig. 3.6 and Fig. 3.11, or give back custom answers on the basis of results, optionally (re-)asking the user a question. A further add-on, for future works, might be the design of an additional module inspired to the human

[1]MST Builder and Definite Clauses Builder.
[2]The Smart Environment Interface.

hippocampus, to let the agent spontaneously link together knowledge for relevance in order to enhance such a dialog system.

One of the weakness of CASPAR, which is fully described at the end of Subsection 2.3.5, is that distinct sequences of assertions give rise to distinct Clauses KB expansions, due to the Clause Conceptual Generalization. That's will not happen for what concerns the third cognitive architecture described in Chapter 4, which is SW-CASPAR, because its conceptual KB is made of triples and it makes reasoning over the Semantic Web, even if rules expressiveness is below[3] than respect to what achieved with FOL nested clauses asserted by instances of CASPAR. Moreover, the choice of replacing the Clauses KB with an ontology made of triples in OWL 2 makes SW-CASPAR suitable for different scenarios in the open-world assumption.

The role of decision-making helper can be fulfilled, by an agent, only when the latter uses a language compliant with what a human counterpart can make usage of, in order to receive instructions and give back understandable feedback. Whether such instructions are extracted from natural language, we have seen (in Section 1.4) different biases from the proper understanding of utterances' meaning. Such a gap between words sequences and meanings can be partially corrected with a set of specific corrective rules, which possible lead two (or more) morphological distinct utterances expressing the same meaning to have equivalent role in a deductive process. Such rules are expressed formarly with the novel fondational ontology LODO, described in Section 4.4, through which it is possible to correct ambiguities of natural language in order to build an ontology representing meaning sentences. Indeed, ambiguities of natural language ontologies can be addressed in more depth, in order to include additional rules in LODO, thus leading to a more human-fashioned reasoning.

In light of above, the results from each chapter demonstrate that one can choose to employ either CASPAR or AD-CASPAR or SW-CASPAR for the design of cognitive-reactive system based on natural language processing, depending on the domain and the intended usage. Furthermore, their specific *onion-shape* designs, give also the chance to integrate modules into one another with ease, in order to create different combinations of interactive features.

[3]Due to the non-monotonic features of SWRL.

Finally, in Chapter 5 two different group of criteria from the state-of-the-art have been used, in order to evaluate the Caspar's family in the scope of cognitive architectures. The evaluation analysis showed that Caspar is a powerful starting point, endowed of functional and structural cognitive features, for the design of computational cognitive models integrating different types of perception, especially reasoning from natural language and (meta-)reasoning.

Chapter 7

Publications

- C. F. Longo, C. Santoro, D. F. Santamaria, M. N. Asmundo, and D. Cantone. "SW-CASPAR: Reactive-Cognitive Architecture based on Natural Language Processing for the task of Decision-Making in the Open World Assumption". In: *22st Workshop "From Objects to Agents" (WOA 2021)*. 2021.
- C. F. Longo, F. Longo, and C. Santoro. "Caspar: Towards decision makinghelpers agents for IoT, based on natural language and first order logic reasoning". In: *Engineering Applications of Artificial Intelligence* 104 (2021), Elsevier, p. 104269. issn: 0952-1976.
- C. Santoro, 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 (NL4AI2020)* co-located with the *19th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2020)*. 2020.
- C. F. Longo, F. Longo and C. Santoro. "A Reactive Cognitive Architecture based on Natural Language Processing for the task of Decision-Making using a Rich Semantic". In: *21st Workshop "From Objects to Agents" (WOA2020)*. 2020.
- F. D'Urso, C. F. Longo, and C. Santoro. "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.

- D. Cantone, C. F. Longo, M. Nicolosi-Asmundo, D. F. Santamaria, and S. Santoro. “Towards an Ontology-Based Framework for a Behavior-Oriented Integration of the IoT”, in *Proceedings of the 20th Workshop From Objects to Agents*, 26-28 June, 2019, Parma, Italy, CEUR Workshop Proceeding Vol. 2404, pp. 119–126, 2019.

- C. F. Longo, C. Santoro, and 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.

- C. F. Longo, C. Santoro. ”A Python-based Assistant Agent able to Interact with Natural Language”. In: *19st Workshop ”From Objects to Agents” (WOA2018)*. 2018.

- D. Cantone, C. F. Longo, M. Nicolosi-Asmundo, D. F. Santamaria, and S. Santoro. “Ontological Smart Contracts in OASIS: Ontology for Agents, Systems, and Integration of Services”, in *Proceedings of the XIV International Symposyium on Intelligent Distributed Computing (IDC 2021).*

Bibliography

[1] P. Thagard. "Critical thinking and informal logic: Neuropsychological perspectives." In: *Informal logic, 51* (2011), pp. 152–170. DOI: https://doi.org/10.22329/il.v31i3.3398.

[2] C. S. Carmelo Fabio Longo Francesco Longo. "A Reactive Cognitive Architecture based on Natural Language Processing for the task of Decision-Making using a Rich Semantic". In: *21st Workshop "From Objects to Agents" (WOA 2020)*. 2020.

[3] C. F. Longo, F. Longo, and C. Santoro. "Caspar: Towards decision making helpers agents for IoT, based on natural language and first order logic reasoning". In: *Engineering Applications of Artificial Intelligence* 104 (2021), p. 104269. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2021.104269. URL: https://www.sciencedirect.com/science/article/pii/S0952197621001160.

[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] C. F. Longo, C. Santoro, D. F. Santamaria, M. N. Asmundo, and D. Cantone. "SW-CASPAR: Reactive-Cognitive Architecture based on Natural Language Processing for the task of Decision-Making in the Open World Assumption". In: *22st Workshop "From Objects to Agents" (WOA 2021)*. 2021.

[6] A. Rao and M. Georgeff. "BDI agents: From theory to practice". In: *Proceedings of the first international conference on multi-agent systems (ICMAS-95)*. San Francisco, CA. 1995, pp. 312–319.

[7] M. E. Bratman. *Intentions, Plans and Practical Reason.* Harvard University Press, 1987.

[8] A. Newell. *Unified theories of cognition.* Cambridge, MA: Harvard University Press., 1990.

[9] G. M. Izhikevich E. M. Edelman. “Large-scale model of mammalian thalamocortical systems.” In: *Proceedings of the National Academy of Sciences of the United States of America, 105(9).* 2008, pp. 3593–3598.

[10] D. S. Ananthanarayanan R. Modha. “Anatomy of a cortical simulator.” In: *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC ’07, (p. ˜1), New York.* ACMPress, 2007.

[11] H. Markram. “The blue brain project.” In: *Nature reviews. Neuroscience, 7(2)* (2006), pp. 153–160.

[12] H. de Garis, C. Shuo, B. Goertzel, and L. Ruiting. “A world survey of artificial brain projects, Part I: Large-scale brain simulations.” In: *Neurocomputing, 74(1–3)* (2010), pp. 3–29.

[13] A. Newell and H. A. Simon. *GPS, a program that simulates human thought.* Computers and thought. New York: McGraw-Hill., 1976.

[14] P. S. Rosenbloom, J. E. Laird, and A. Newell. *The Soar papers: Research on integrated intelligence.* Cambridge, MA: MIT Press., 1993.

[15] D. E. Kieras D. E. Meyer. “An overview of the EPIC architecture for cognition and performance with application to human-computer interaction.” In: *Human-Computer Interaction, 4(12)* (1997), pp. 391–438.

[16] J. R. Anderson. *The architecture of cognition.* Cambridge, MA: Harvard University Press, 1983.

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

[18] P. N. Stuart J. Russel. “Artificial Intelligence: A Modern Approach”. In: Pearson, 2010. Chap. 9.3.

[19] A. Lieto. *Cognitive Design for Artificial Minds.* Routledge, 2021. Chap. 3.

[20] A. Darwiche. "Human-Level Intelligence or Animal-Like Abilities?" In: *Communications of the ACM* 61 No. 10 (2018), pp. 56–67. DOI: 10.1145/3271625.

[21] T. Metzinger. *Being No One: The Self-Model Theory of Subjectivity.* MIT Press, 2003.

[22] J. W. *The principles of psychology.* Cambridge, MA: Harvard University Press, 1983.

[23] C. D. *The Conscious Mind: In Search of a Fundamental Theory.* New York: Oxford University Press, 1996.

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

[25] M. A. Cerullo. "The Problem with Phi: A Critique of Integrated Information Theory". In: *PLOS Computational Biology* (2015). DOI: 10.1371/journal.pcbi.1004286.

[26] C. D. *Absent qualia, fading qualia, dancing qualia.* Metzinger T. (Ed.), Conscious experience (pp. 309–328). Imprint Academic, 1995.

[27] S. Aaronson. *Giulio Tononi and Me: A Phi-nal Exchange.* Available at http://www.scottaaronson.com/blog/?p=1823. 2014.

[28] R. G. *A Place for Consciousness: Probing the Deep Structure of the Natural World.* Oxford: Oxford University Press, 2004.

[29] A. Turing. "Computing machinery and intelligence". In: Mind, 1950, pp. 433–60.

[30] F. Bianchini. *Turing and the evaluation of intelligence.* Isonomia: Online Philosophical Journal of the University of Urbino:1-18, 2014.

[31] E. M. Macphail. *The evolution of consciousness.* Oxford University Press, 1998. DOI: https://doi.org/10.1093/acprof:oso/9780198503248.001.0001.

[32] C. Koch. *The Feeling of Life Itself: Why consciousness can't be computated.* Massachussets Institute of Technology, 2019. ISBN: 9780262042819.

[33] J. B. Taylor. *My Stroke of Insight: A Brain Scientist's Personal Journey.* London: Hodder Stoughton, 2008, p. 183. ISBN: 978-0-340-98048-4.

[34] R. M. Lazar, R. S. Marshall, G. D. Prell, and J. Pile-Spellman. "The experience of Wernicke's aphasia". In: *Neurology* (2000). DOI: https://doi.org/10.1212/WNL.55.8.1222.

[35] J. Julian. *The Origin of Consciousness in the Breakdown of the Bicameral Mind.* Mariner Books/Houghton Mifflin Company, 2000.

[36] J. Carrol. *Language, Thought and Reality: Selected writings of Benjamin Lee Whorf.* Massachussets Institute of Technology, 1956.

[37] A. Browarnik and O. Maimon. "Ontology Learning from Text". In: *The First International Conference on Big Data, Small Data, Linked Data and Open Data.* 2015.

[38] F. Moltmann. *Natural Language Ontology.* Mar. 2017. DOI: 10.1093/acrefore/9780199384655.013.330. URL: https://oxfordre.com/linguistics/view/10.1093/acrefore/9780199384655.001.0001/acrefore-9780199384655-e-330.

[39] S. G. Williams. "Ontology, Identity and Modality By Peter van Inwagen, Cambridge University Press, 2001, pp. 261." In: *Philosophy* 79.2 (2004), pp. 335–342. DOI: 10.1017/S0031819104120305.

[40] S. Pinker. *The Language Instinct.* William Morrow and Company, 1994. ISBN: 0-688-12141-1.

[41] V. Kepuska and G. Bohouta. *Next- Generation of Virtual Personal Assistants (Microsoft Cortana, Apple Siri, Amazon Alexa and Google Home).* 2018.

[42] I. H. Heesij Jeon Hyung Rai Oh and J. Kim. "An Intelligence Dialog Agent for the IoT Home". In: *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence.* 2016.

[43] P. E.V., M. M.S., R. A.Y., V. L.S., K. M.V., and P. S.V. "Investigation and Development of the Intelligent Voice Assistant for the Internet of Things Using Machine Learning". In: *in Moscow Workshop on Electronic and Networking Technologies.* 2018.

[44] B. S. Mahnoosh Mehrabani Srinivas Bangalore. “Personalized Speech Recognition for Internet of Things”. In: *IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015.

[45] R. H. Rohan Kar. “Applying Chatbots to the Internet of Things: Opportunities and Architectural Elements”. In: *International Journal of Advanced Computer Science and Applications 7(11)* (2016). DOI: 10.14569/IJACSA.2016.071119.

[46] S. J. Cyril Joe Baby Faizan Ayyub Khan. “Home Automation using IoI and a Chatbot using Natural Language Processing”. In: *IEEE International Conference on Innovation in Power and Advanced Computing Technologies.* 2017.

[47] F. D’Urso, C. F. Longo, and C. Santoro. “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.

[48] D. F. Lucentini and R. R. Gudwin. “A Comparison Among Cognitive Architectures: A Theoretical Analysis”. In: *2015 Annual International Conference on Biologically Inspired Cognitive Architectures*. 2015.

[49] K. F. Morteza Dehghani Emmett Tomai and M. Klenk. “An Integrated Reasoning Approach to Moral Decision-Making”. In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. 2018.

[50] J. E. L. Peter Lindes. “Toward Integrating Cognitive Linguistics and Cognitive Language Processing”. In: *Proceedings of the 14th International Conference on Cognitive Modeling*. 2016.

[51] J. G. Susan L. Epstein Rebecca J. Passonneau and T. Ligorio. “The Role of Knowledge and Certainty in Understanding for Dialogue”. In: *Advances in Cognitive Systems: Papers from the 2011 AAAI Fall Symposium (FS-11-01).* 2011.

[52] V. Këpuska and G. Bohouta. “Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx)”. In: *Int. Journal of Engineering Research and Application* Vol. 7, Issue 3, (Part -2) (March 2017), pp. 20–24.

[53] J. K. Matthias Scheutz Paul Schermerhorn and D. Anderson. “First steps toward natural human-like HRI”. In: *Auton Robot* (2007), 22, 411–423 (2007). DOI: https://doi.org/10.1007/s10514-006-9018-3.

[54] P. Bustos, L. J. Manso, J. P. Bandera, A. Romero-Garcés, L. V. Calderita, R. Marfil, and A. Bandera. “A Unified Internal Representation of the Outer World for Social Robotics”. In: *Robot 2015: Second Iberian Robotics Conference*. Ed. by L. P. Reis, A. P. Moreira, P. U. Lima, L. Montano, and V. Muñoz-Martinez. Cham: Springer International Publishing, 2016, pp. 733–744. ISBN: 978-3-319-27149-1.

[55] P. Lison and G.-J. Kruijff. “Salience-driven Contextual Priming of Speech Recognition for Human-Robot Interaction”. In: *Frontiers in Artificial Intelligence and Applications*. Vol. 178. 2008. ISBN: 978-1-60750-355-2.

[56] T. A. Vasanth Sarathy Jason R. Wilson and M. Scheutz. “Enabling Basic Normative HRI in a Cognitive Robotic Architecture”. In: *2nd Workshop on Cognitive Architectures for Social Human-Robot Interaction 2016 (CogArch4sHRI 2016)*. 2016.

[57] C. Paniagua and J. Delsing. “Industrial Frameworks for Internet of Things: A Survey”. In: *IEEE SYSTEMS JOURNAL* (2020). DOI: https://doi.org/10.1109/JSYST.2020.2993323.

[58] P. Ray. “A survey on Internet of Things architectures”. In: *Journal of King Saud University – Computer and Information Sciences* (2016). DOI: http://dx.doi.org/10.1016/j.jksuci.2016.10.003.

[59] J. D. Hasan Derhamy Jens Eliasson and P. Priller. “A Survey of Commercial Frameworks for the Internet of Things”. In: *2015 IEEE 20th Conference on Emerging Technologies and Factory Automation (ETFA)* (2015). DOI: https://doi.org/10.1109/ETFA.2015.7301661.

[60] S. S. Sabry, N. A. Qarabash, and H. S. Obaid. “The Road to the Internet of Things: a Survey”. In: *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*. 2019, pp. 290–296. DOI: 10.1109/IEMECONX.2019.8876989.

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

[62] G. A. Miller. “WordNet: A Lexical Database for English”. In: *Communications of the ACM Vol. 38, No. 11: 39-41*. 1995.

[63] D. Davidson. “The logical form of action sentences”. In: *The logic of decision and action*. University of Pittsburg Press, 1967, pp. 81–95.

[64] S. Anthony and 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.

[65] T. Parsons. “Events in the Semantics of English: A Study in Subatomic Semantics”. In: MIT Press, 1990.

[66] X. Huang and L. Deng. “An Overview of Modern Speech Recognitiong”. In: *Indurkhya/Handbook of Natural Language Processing C5921*$_C$01. Microsoft Corporation, 2009, pp. 339–344.

[67] R. Rajan Mehla Mamta. “Automatic Speech Recognition: A Survey”. In: *International Journal of Advanced Research in Computer Science and Electronics Engineering* Volume 3, Issue 1 (January 20147), pp. 20–24.

[68] A. D. Saliha Benkerzaz Youssef Elmir. “A Study on Automatic Speech Recognition”. In: *Journal of Information Technology Review* Volume 10, Number 3 (August 2019).

[69] 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, pp. 387–396.

[70] H. Matthew. *spaCy: Industrial-Strength Natural Language Processing*. https://spacy.io. 2017.

[71] ClearNLP. *Clear NLP Tagset*. URL: https://github.com/clir/clearnlp-guidelines.

[72] L. D. Consortium. *Treebank-3*. URL: https://catalog.ldc.upenn.edu/LDC99T42.

[73] B. H. Partee. "Lexical Semantics and Compositionality". In: vol. 1. Lila R. Gleitman and Mark Liberman editors, 1995, pp. 311–360.

[74] C. F. Longo, C. Santoro, and 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.

[75] L. Fichera, F. Messina, G. Pappalardo, and C. Santoro. "A Python Framework for Programming Autonomous Robots Using a Declarative Approach". In: *Sci. Comput. Program.* 139 (2017), pp. 36–55. DOI: 10.1016/j.scico.2017.01.003. URL: https://doi.org/10.1016/j.scico.2017.01.003.

[76] J. D. Smith and J. P. Minda. "Prototypes in the mist: The early epochs of category learning". In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 24(6) (1998), pp. 1411–1436. DOI: 10.1037/0278-7393.24.6.1411.

[77] B. C. Malt. "An on-line investigation of prototype and exemplar strategies in classification". In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 15(4) (1989), pp. 539–555. DOI: 10.1037/0278-7393.15.4.539.

[78] M. J. Smith J.D. Murray M.J.Jr. "Straight talk about linear separability". In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 23(3) (1997), pp. 659–680. DOI: 10.1037/0278-7393.23.3.659.

[79] T. M. Quoc Le. "Distributed Representations of Sentences and Documents". In: *Proceedings of the 31st International Conference on Machine Learning, Beijing, China*. 2014.

[80] R. Navigli. "Word Sense Disambiguation: A Survey". In: *ACM Computing Surveys* Vol. 41, No. 2, Article 10 (2009). DOI: 10.1145/1459352.1459355.

[81] stanford. *The Stanford Question Answering Dataset SQuAD2.0*. 2018. URL: https://rajpurkar.github.io/SQuAD-explorer/.

[82] PYPL. *PopularitY of Programming Language.* http://pypl.github.io/PYPL.html. 2020.

[83] Snowboy. *Snowboy, a hotword detection engine.* URL: https://https://snowboy.kitt.ai.

[84] Picovoice. *Porcupine.* URL: https://https://github.com/Picovoice/porcupine.

[85] Huggingface. *NeuralCoref 4.0: Coreference Resolution in spaCy with Neural Networks.* URL: https://github.com/huggingface/neuralcoref.

[86] H. Loebner. *The Loebner Prize.* Available at https://www.ocf.berkeley.edu/~arihuang/academic/research/loebner.html.

[87] A. fondation. *Artificial Intelligence Markup Language.* Available at http://www.aiml.foundation/.

[88] H. Madhumitha.S Keerthana.B. "Interactive Chatbot Using AIML". In: *Int. Jnl. Of Advanced Networking Applications* Special Issue (2019).

[89] B. Wilcox. *Chatscript.* Available at https://github.com/ChatScript/ChatScript.

[90] Q. V. L. Ilya Sutskever Oriol Vinyals. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems* 27 (2014).

[91] J. H. Kotagiri Ramamohanarao. "An Introduction to Deductive Database Languages and Systems". In: *The International Journal of Very Large Data Bases* Journal, 3, 107-122 (1994).

[92] D. Oberle, N. Guarino, and S. Staab. *What is an ontology?* Handbook on Ontologies, 2nd edition. Springer, 2009.

[93] L. Jean-Baptiste. *Owlready2 0.31.* 2021. URL: https://pypi.org/project/Owlready2/.

[94] J.-B. Lamy. "Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies". In: *Artificial Intelligence in Medicine* 80 (2017), pp. 11–28. ISSN: 0933-3657. DOI: https://doi.org/10.1016/j.artmed.2017.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0933365717300271.

[95] M. Finkelstein-Landau and E. Morin. “Extracting semantic relationships between terms: supervised vs. unsupervised methods”. In: *International Workshop on Ontological Engineering on the Global Information Infrastructure, Dagstuhl Castle, Germany.* 1999.

[96] A. S. “Part-of-Speech Tagging and Partial Parsing”. In: *Young S., Bloothooft G. (eds) Corpus-Based Methods in Language and Speech Processing. Text, Speech and Language Technology, Springer, Dordrecht* vol 2 (1997), pp. 501–518. DOI: https://doi.org/10.1007/978-94-017-1183-8_4.

[97] P. Gamallo, M. Gonzalez, A. Agustini, G. Lopes, and V. S. de Lima. “Mapping Syntactic Dependencies onto Semantic Relations”. In: *Proceedings of the ECAI Workshop on Machine Learning and Natural Language Processing for Ontology Engineering.* 2002.

[98] D. Faure and C. Nedellec. “Knowledge acquisition of predicate argument structures from technical texts using machine leraning: The system ASIUM”. In: *Knowledge Acquisition, Modelling and Management* (1999), pp. 329–334.

[99] A. A. B. Mehrnoush Shamsfard. “Learning ontologies from natural language texts”. In: *Int. J. Human-Computer Studies 60* vol 60 (2004), pp. 17–63.

[100] A. Maedche and S.Staab. “The text-to-onto ontology learning environment”. In: *Software Demonstration at ICCS-2000-Eight International Conference on Conceptual Structures.* 2000.

[101] S. Gillani and A. Kő. “ProMine: A Text Mining Solution for Concept Extraction and Filtering”. In: *Corporate Knowledge Discovery and Organizational Learning: The Role, Importance, and Application of Semantic Business Process Management.* Ed. by A. Gábor and A. Kő. Cham: Springer International Publishing, 2016, pp. 59–82. ISBN: 978-3-319-28917-5. DOI: 10.1007/978-3-319-28917-5_3.

[102] M. R. U. Hahn and S. Schulz. “MediSynDiKATe–design considerations for an ontology-based medical text understanding system”. In: *Proceedings of the AMIA Symposium (p. 330), American Medical Informatics Association.* 2000.

[103] K. Z. E. Drymonas and E. Petrakis. “Unsupervised ontology acquisition from plain text: The OntoGain system”. In: *Natural Language Processing and Information System* (2010), pp. 277–287.

[104] H. Y. Al-Aswadi Fatima N. Chan and K. H. Gan. “Automatic ontology construction from text: a review from shallow to deep learning trend”. In: *Artificial Intelligence Review* (2020).

[105] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. “Pellet: A practical OWL-DL reasoner”. In: *Web Semantics* 5.2 (2007), pp. 51–53. ISSN: 1570-8268.

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

[107] World Wide Web Consortium. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. 2004. URL: http://www.w3.org/Submission/SWRL/.

[108] B. Webb. “Can robots make good models of biological behaviour?” In: *Behavioral and Brain Sciences* 24.6 (2001), pp. 1033–1050. DOI: 10.1017/S0140525X01000127.

[109] F. E. Ritter, F. Tehranchi, and J. D. Oury. “ACT-R: A cognitive architecture for modeling cognition”. In: *Wiley interdisciplinary reviews. Cognitive science* 10.3 (May 2019), e1488. ISSN: 1939-5078. DOI: 10.1002/wcs.1488. URL: https://doi.org/10.1002/wcs.1488.

[110] P. S. Rosenbloom. “The Sigma cognitive architecture and system”. In: (2013).

[111] R. M. Nosofsky. “Formal Approaches in Categorization: The generalized context model: an exemplar model of classification”. In: 2011.