A Description Logics Based Cognitively Inspired Tool for Knowledge Generation Via Concept Combination

Antonio Lieto[1], Gian Luca Pozzato[2,**] and Gioele Tallone[2]

[1]*Dipartimento di Scienze Politiche e della Comunicazione/DISPC, Università degli Studi di Salerno, 84084 Fisciano (SA), Italy*

[2]*Dipartimento di Informatica, Università degli Studi di Torino, 10149, Turin, Italy*

Abstract

In this work we continue our investigation on tools for the dynamic generation of novel knowledge by exploiting a recently introduced extension of a Description Logic of typicality able to combine prototypical descriptions of concepts. Given a goal expressed as a set of properties, in case an intelligent agent cannot find a concept in its initial knowledge base able to fulfill all these properties, our system exploits the Description Logic $\mathbf{T}^{\mathrm{CL}}$ in order to find two concepts whose creative combination satisfies the goal. The knowledge base of the agent is then extended by the prototype resulting from the concept combination, and the combined concept represents the solution for the initial goal. In addition, we show how the tool we propose can be employed in the field of cognitive architectures in order to overcome situations like the *impasse* in SOAR by extending the possible options of its subgoaling procedures.

Keywords

Description Logics, Cognitive Architectures, Dynamic Knowledge Generation, Nonmonotonic reasoning

1. Introduction

A challenging problem in Artificial Intelligence concerns the capability of an intelligent agent to achieve its goals when its knowledge base does not contain enough information to do that. In this line of research, existing goal-directed systems usually implement a re-planning strategy in order to tackle the problem. This is systematically performed by either an external injection of novel knowledge or as the result of a communication with another intelligent agent [1].

In this work, we continue our investigation started with the works [2, 3, 4, 5], in which we propose an alternative approach, consisting in a dynamic and automatic generation of novel knowledge obtained through a process of commonsense reasoning. The idea is as follows: given an intelligent agent and a set of *goals*, if it is not able to achieve them from an initial knowledge base, then it tries to dynamically generate new knowledge by *combining* available information. Novel information will be then used to extend the initial knowledge base in order to achieve the goals. As an example, suppose that an intelligent agent is aware of the facts that, normally, coffee contains caffeine and is a hot beverage, that the chocolate with cream is normally sweet and has a taste of milk, whereas Limoncello is not a hot beverage (normally,

CILC 2024: 39th Italian Conference on Computational Logic, June 26-28, 2024, Rome, Italy

*Corresponding author.

alieto@unisa.it (A. Lieto); gianluca.pozzato@unito.it (G. L. Pozzato); gioele.tallone@edu.unito.it (G. Tallone)

https://docenti.unisa.it/024406/home (A. Lieto); https://www.di.unito.it/~pozzato (G. L. Pozzato)

0000-0002-8323-8764 (A. Lieto); 0000-0002-3952-4624 (G. L. Pozzato)

it is served chilled). Both coffee and Limoncello are after meal drinks. Cold winters in Turin suggest to have a hot after-meal drink, also being sweet and having taste of milk. None of the concepts in the knowledge base of the agent are able to achieve the goal on their own, however, the combination between coffee and chocolate with cream provides a solution.

In this paper we introduce EDIFICA, which stands for "ExtenDIble & FlexIble concept Combination Architecture", a tool extending GOCCIOLA [3] following this approach in the context of Description Logics (from now on, DLs for short). DLs are one of the most important formalisms of knowledge representation and are at the base of the languages for building ontologies such as OWL. In this respect, we exploit the Description Logic $\mathbf{T}^{\mathrm{CL}}$, recently introduced in order to account for the phenomenon of concept combination of prototypical concepts [6]. The logic $\mathbf{T}^{\mathrm{CL}}$ relies on the logic of typicality $\mathscr{ALC} + \mathbf{T}_{\mathbf{R}}^{R_A C_L}$ [7], whose semantics is based on the notion of rational closure, as well as on the DISPONTE semantics of probabilistic DLs [8], and is equipped with a cognitive heuristic used by humans for concept composition. In this logic, typicality inclusions of the form $p :: \mathbf{T}(C) \sqsubseteq D$ are used to formalize that "we believe with degree p about the fact that typical Cs are Ds". As in the distributed semantics, this allows us to consider different scenarios containing only some typicality inclusions, each one having a suitable probability. Such scenarios are then used to ascribe typical properties to a concept C obtained as the combination of two concepts, revising the initial knowledge base with the addition of typical properties of C. In the example, the revised knowledge base provided by the logic $\mathbf{T}^{\mathrm{CL}}$ contains typical properties of the combination of coffee and chocolate with cream, which suggests to consider a beverage corresponding to the famous Turin drink known as *Bicerín* (little glass), made by coffee, chocolate and cream.

EDIFICA tries to tackle the main criticisms and drawbacks of GOCCIOLA, namely:

- GOCCIOLA randomly selects concepts to be combined; as an alternative, EDIFICA selects such candidates starting from the properties of the goal to be fullfilled;
- GOCCIOLA is able to combine only two concepts at a time, whereas EDIFICA has no longer this limitation;
- EDIFICA implements a more sophisticated mechanism in order to choose the list of concepts to be combined among all the candidates;
- EDIFICA adopts a "smart" mechanism in generating scenarios in the logic $\mathbf{T}^{\mathrm{CL}}$, for instance by discarding inconsistent ones, rather than generating all of them by means of a "brute force" mechanism as in GOCCIOLA.

Moreover, EDIFICA offers an *open* and modular architecture, that allows the user to provide and add specific and suitable mechanisms. In addition, we also show how the proposed tool can be employed in the field of cognitive architectures in order to overcome situations like the *impasse* in the well-established architecture SOAR [9] by extending the possible options of its subgoaling procedures.

The plan of the paper is as follows. In Section 2 we briefly recall the DL for concept combination $\mathbf{T}^{\mathrm{CL}}$. In Section 3 we provide a formal description of the problem of dynamic knowledge generation in the context of the logic $\mathbf{T}^{\mathrm{CL}}$. In Section 4 we recall the main features of the ancestor of EDIFICA, namely the system GOCCIOLA, and we show that it is a promising candidate to tackle such a problem by means of some paradigmatic examples. In Section 5 we describe the

novel system EDIFICA, by emphasizing how it improves the existing GOCCIOLA. In Section 6 we show how EDIFICA can be employed in the field of cognitive architectures by extending the options of their subgoaling procedures. We conclude in Section 7 with some pointers to future issues.

2. The Logic $\mathbf{T}^{\text{CL}}$: a Description Logic of Typicality for Concept Combination

In [6] we have introduced a nonmonotonic Description Logic of typicality called $\mathbf{T}^{\text{CL}}$ (typicality-based compositional logic). This logic combines two main ingredients. The first one relies on the DL of typicality $\mathcal{ALC} + \mathbf{T}^{RACL}_{\mathbf{R}}$ introduced in [7], which allows to describe the *protoype* of a concept. In this logic, "typical" properties can be directly specified by means of a "typicality" operator $\mathbf{T}$ enriching the underlying DL, and a TBox can contain inclusions of the form $\mathbf{T}(C) \sqsubseteq D$ to represent that "typical Cs are also Ds". As a difference with standard DLs, in the logic $\mathcal{ALC} + \mathbf{T}^{RACL}_{\mathbf{R}}$ one can consistently express exceptions and reason about defeasible inheritance as well. For instance, a knowledge base can consistently express that "normally, athletes are fit", whereas "sumo wrestlers usually are not fit" by $\mathbf{T}(\mathit{Athlete}) \sqsubseteq \mathit{Fit}$ and $\mathbf{T}(\mathit{SumoWrestler}) \sqsubseteq \neg \mathit{Fit}$, given that $\mathit{SumoWreslter} \sqsubseteq \mathit{Athlete}$. The semantics of the $\mathbf{T}$ operator is characterized by the properties of *rational logic* [10], recognized as the core properties of nonmonotonic reasoning. $\mathcal{ALC} + \mathbf{T}^{RACL}_{\mathbf{R}}$ is characterized by a minimal model semantics corresponding to an extension to DLs of a notion of *rational closure* as defined in [10] for propositional logic: the idea is to adopt a preference relation among $\mathcal{ALC} + \mathbf{T}^{RACL}_{\mathbf{R}}$ models, where intuitively a model is preferred to another one if it contains less exceptional elements, as well as a notion of *minimal entailment* restricted to models that are minimal with respect to such preference relation. As a consequence, $\mathbf{T}$ inherits well-established properties like *specificity* and *irrelevance*: in the example, the logic $\mathcal{ALC} + \mathbf{T}^{RACL}_{\mathbf{R}}$ allows us to infer $\mathbf{T}(\mathit{Athlete} \sqcap \mathit{Bald}) \sqsubseteq \mathit{Fit}$ (being bald is irrelevant with respect to being fit) and, if one knows that Hiroyuki is a typical sumo wrestler, to infer that he is not fit, giving preference to the most specific information.

As a second ingredient, we have considered a distributed semantics similar to the one of probabilistic DLs known as DISPONTE [8], allowing to label inclusions $\mathbf{T}(C) \sqsubseteq D$ with a real number between 0.5 and 1, representing its degree of belief/probability. In a slight extension of the above example, we can express a degree of belief in the typicality inclusions about athletes and sumo wrestlers: we believe with a probability of 80% that, normally, athletes are fit whereas sumo wrestlers are not; furthermore, we believe that athletes are usually young with a higher degree of 95%. This is formalized by the following knowledge base: (1) $\mathit{SumoWrestler} \sqsubseteq \mathit{Athlete}$; (2) $0.8 :: \mathbf{T}(\mathit{Athlete}) \sqsubseteq \mathit{Fit}$; (3) $0.8 :: \mathbf{T}(\mathit{SumoWrestler}) \sqsubseteq \neg \mathit{Fit}$; (4) $0.95 :: \mathbf{T}(\mathit{Athlete}) \sqsubseteq \mathit{YoungPerson}$. We consider eight different scenarios, representing all possible combinations of typicality inclusion: as an example, $\{((2), 1), ((3), 0), ((4), 1)\}$ represents the scenario in which (2) and (4) hold, whereas (3) does not. We equip each scenario with a probability depending on those of the involved inclusions: the scenario of the example, has probability 0.8×0.95 (since 2 and 4 are involved) $\times (1 - 0.8)$ (since 3 is not involved) $= 0.152 = 15.2\%$. Such probabilities are then taken into account in order to choose the most adequate scenario describing the prototype of the combined concept.

The logic $\mathbf{T}^{\mathrm{CL}}$ seems to be a promising candidate in order to tackle the problem of concept combination. Combining the typical knowledge of pre-existing concepts is among the most creative cognitive abilities exhibited by humans. This generative phenomenon highlights some crucial aspects of the knowledge processing capabilities in human cognition and concerns high-level capacities associated to creative thinking and problem solving. Dealing with this problem requires, from an AI perspective, the harmonization of two conflicting requirements that are hardly accommodated in symbolic systems (including formal ontologies [11]): the need for a syntactic and semantic compositionality (typical of logical systems) and that one concerning the exhibition of typicality effects. According to a well-known argument [12], in fact, prototypes are not compositional. The argument runs as follows: consider a concept like *pet fish*. It results from the composition of the concept *pet* and of the concept *fish*. However, the prototype of *pet fish* cannot result from the composition of the prototypes of a pet and a fish: e.g. a typical pet is furry and warm, a typical fish is grayish, but a typical pet fish is neither furry and warm nor grayish (typically, it is red).

Given a knowledge base $\mathscr{K} = \langle \mathscr{R}, \mathscr{T}, \mathscr{A} \rangle$, where $\mathscr{R}$ is the set of standard (rigid) inclusions of $\mathscr{ALC}$, $\mathscr{T}$ is the set of typicality inclusions, and $\mathscr{A}$ is the ABox, and given two concepts C_H and C_M occurring in $\mathscr{K}$, the logic $\mathbf{T}^{\mathrm{CL}}$ allows defining a prototype of the compound concept C as the combination of the HEAD C_H, the dominant element in the combination, and the MODIFIER C_M. Typical properties to ascribe to the combination of C_H and C_M are inclusions of the form $\mathbf{T}(C) \sqsubseteq D$ (or, equivalently, $\mathbf{T}(C_H \sqcap C_M) \sqsubseteq D$), whose intuitive meaning is that "D is a typical property of C combining C_H and C_M", and are obtained by considering blocks of scenarios with the same probability, in decreasing order starting from the highest one. We first discard all the inconsistent scenarios, then:

- we discard those scenarios considered as *trivial*, consistently inheriting all the properties from the HEAD from the starting concepts to be combined. This choice is motivated by the challenges provided by task of common-sense conceptual combination itself: in order to generate plausible and creative compounds it is necessary to maintain a level of surprise in the combination. Thus both scenarios inheriting all the properties of the two concepts and all the properties of the HEAD are discarded since prevent this surprise;
- among the remaining ones, we discard those scenarios inheriting properties from the MODIFIER in conflict with properties that could be consistently inherited from the HEAD;
- if the set of scenarios of the current block is empty, i.e. all the scenarios have been discarded either because trivial or because preferring the MODIFIER, we repeat the procedure by considering the block of scenarios having the immediately lower probability.

Remaining scenarios are those selected by the logic $\mathbf{T}^{\mathrm{CL}}$. The ultimate output of our mechanism is a knowledge base in the logic $\mathbf{T}^{\mathrm{CL}}$ whose set of typicality properties is enriched by those of the compound concept C. Given a scenario w satisfying the above properties, we define the properties of C as the set of inclusions $p :: \mathbf{T}(C) \sqsubseteq D$, for all $\mathbf{T}(C) \sqsubseteq D$ that are entailed from w in the logic $\mathbf{T}^{\mathrm{CL}}$. The probability p is such that:

- if $\mathbf{T}(C_H) \sqsubseteq D$ is entailed from w, that is to say D is a property inherited either from the HEAD (or from both the HEAD and the MODIFIER), then p corresponds to the degree of belief of such inclusion of the HEAD in the initial knowledge base, i.e. $p : \mathbf{T}(C_H) \sqsubseteq D \in \mathscr{T}$;

- otherwise, i.e. $\mathbf{T}(C_M) \sqsubseteq D$ is entailed from w, then p corresponds to the degree of belief of such inclusion of a MODIFIER in the initial knowledge base, i.e. $p \ : \ \mathbf{T}(C_M) \sqsubseteq D \in \mathcal{T}$.

The knowledge base obtained as the result of combining concepts C_H and C_M into the compound concept C is called *C-revised* knowledge base, and it is defined as follows:

$$\mathcal{K}_C = \langle \mathcal{R}, \mathcal{T} \cup \{p \ : \ \mathbf{T}(C) \sqsubseteq D\}, \mathcal{A} \rangle,$$

for all D such that either $\mathbf{T}(C_H) \sqsubseteq D$ is entailed in w or $\mathbf{T}(C_M) \sqsubseteq D$ is entailed in w, and p is defined as above. In [6] we have shown that reasoning in the logic $\mathbf{T}^{\text{CL}}$ remains in the same complexity class of standard $\mathcal{ALC}$ Description Logics, namely that reasoning in $\mathbf{T}^{\text{CL}}$ is ExpTime-complete.

3. Exploiting the Logic $\mathbf{T}^{\text{CL}}$ for Knowledge Generation Via Concept Combination

We exploit the logic $\mathbf{T}^{\text{CL}}$ in order to tackle the following problem: given a knowledge base $\mathcal{K}$ in the Description Logic $\mathbf{T}^{\text{CL}}$, an intelligent agent has to achieve a goal $\mathcal{G}$ intended as a set of concepts $\{D_1, D_2, \ldots, D_n\}$. More precisely, the agent has to find a *solution* for the goal, namely a concept C such that, for all properties D_i, it holds that either $\mathcal{K} \models C \sqsubseteq D_i$ or $\mathcal{K} \models \mathbf{T}(C) \sqsubseteq D_i$ in the logic of typicality $\mathcal{ALC} + \mathbf{T}_{\mathbf{R}}^{RACL}$. If $\mathcal{K}$ does not contain any solution for the goal, then the agent tries to generate a *new concept* by combining two existing ones C_1 and C_2 by means of the logic $\mathbf{T}^{\text{CL}}$: C is then considered a solution for the goal if, considering the $(C_1 \sqcap C_2)$-revised knowledge base $\mathcal{K}_C$ extending $\mathcal{K}$, we have that, for all properties D_i, it holds that either $\mathcal{K}_C \models C \sqsubseteq D_i$ or $\mathcal{K}_C \models \mathbf{T}(C) \sqsubseteq D_i$ in the logic of typicality $\mathcal{ALC} + \mathbf{T}_{\mathbf{R}}^{RACL}$.

This is formally defined as follows:

Definition 1. *Given a knowledge base $\mathcal{K}$ in the logic $\mathbf{T}^{CL}$, let $\mathcal{G}$ be a set of atomic concepts $\{D_1, D_2, \ldots, D_n\}$ called* goal. *We say that a concept C is a* solution *to the goal $\mathcal{G}$ if either:*

- *for all $D_i \in \mathcal{G}$, either $\mathcal{K} \models C \sqsubseteq D_i$ or $\mathcal{K} \models \mathbf{T}(C) \sqsubseteq D_i$ in the logic $\mathbf{T}^{CL}$*

or

- *C corresponds to the combination of two concepts C_1 and C_2 occurring in $\mathcal{K}$, i.e. $C \equiv C_1 \sqcap C_2$, and the C-revised knowledge base $\mathcal{K}_C$ provided by the logic $\mathbf{T}^{CL}$ is such that, for all $D_i \in \mathcal{G}$, either $\mathcal{K}_C \models C \sqsubseteq D_i$ or $\mathcal{K}_C \models \mathbf{T}(C) \sqsubseteq D_i$ in the logic $\mathbf{T}^{CL}$.*

Let us conclude this section by formalizing the example of the Introduction.

Example 2. *In the example of the Introduction, suppose that $\mathcal{K}$ contains the information that, normally, coffee contains caffeine and is a hot beverage; moreover, we have that the chocolate with cream is normally sweet and has a taste of milk, whereas Limoncello is not a hot beverage (normally, it is served chilled). Both coffee and Limoncello are after meal drinks. We can represent these information as follows:*

$0.9 \;::\; \mathbf{T}(\mathit{Coffee}) \sqsubseteq \mathit{AfterMealDrink}$
$0.8 \;::\; \mathbf{T}(\mathit{Coffee}) \sqsubseteq \mathit{WithCaffeine}$
$0.85 \;::\; \mathbf{T}(\mathit{Coffee}) \sqsubseteq \mathit{HotBeverage}$
$\mathit{Limoncello} \sqsubseteq \mathit{AfterMealDrink}$
$0.9 \;::\; \mathbf{T}(\mathit{Limoncello}) \sqsubseteq \neg \mathit{HotBeverage}$
$0.65 \;::\; \mathbf{T}(\mathit{ChocolateWithCream}) \sqsubseteq \mathit{Sweet}$
$0.95 \;::\; \mathbf{T}(\mathit{ChocolateWithCream}) \sqsubseteq \mathit{TasteOfMilk}$

Cold winters in Turin suggest to have a hot after-meal drink, also being sweet and having taste of milk. We can then define a goal $\mathcal{G}$ as

$$\mathcal{G} = \{\mathit{AfterMealDrink}, \mathit{HotBeverage}, \mathit{Sweet}, \mathit{TasteOfMilk}\}.$$

None of the concepts in the knowledge base represent a solution for the problem. However, the combination between the concepts Coffee and ChocolateWithCream represents a solution. Indeed, the revised knowledge base obtained by exploiting the logic $\mathbf{T}^{CL}$ to combine these concepts allows the agent to extend its knowledge with the following typicality inclusions:

$0.9 \;::\; \mathbf{T}(\mathit{Coffee} \sqcap \mathit{ChocolateWithCream}) \sqsubseteq \mathit{AfterMealDrink}$
$0.85 \;::\; \mathbf{T}(\mathit{Coffee} \sqcap \mathit{ChocolateWithCream}) \sqsubseteq \mathit{HotBeverage}$
$0.65 \;::\; \mathbf{T}(\mathit{Coffee} \sqcap \mathit{ChocolateWithCream}) \sqsubseteq \mathit{Sweet}$
$0.95 \;::\; \mathbf{T}(\mathit{Coffee} \sqcap \mathit{ChocolateWithCream}) \sqsubseteq \mathit{TasteOfMilk}$

providing a solution for the goal corresponding to the famous Turin drink known as Bicerín *(little glass).*

4. The Ancestor of the System EDIFICA: the System GOCCIOLA

In this Section we describe GOCCIOLA, a preliminary implementation of a system able to extend the knowledge of an agent in order to fulfill a set of properties representing the goal that the agent wants to achieve. GOCCIOLA is implemented in Python and its current version, along with the files for the examples presented in this paper, are available at http://di.unito.it/gocciola. The architecture of the system GOCCIOLA is shown in Figure 1.

As an example, let us consider the goal: *object, cutting, graspable*, in other words our agent is looking for an object being graspable and which is able to cut. The initial knowledge base is formalized in the language of the logic $\mathbf{T}^{\mathrm{CL}}$ and it is stored in a suitable file. Rigid properties, holding for all individuals of a given class, are stored as pairs object-property, whereas typical properties are formalized as triples object-property-probability. We have considered an extension with probabilities of a portion of the ontology opencyc[1]. As an example, the concept *Vase* is stored as follows (on the right the corresponding knowledge base in $\mathbf{T}^{\mathrm{CL}}$):

[1] https://github.com/asanchez75/opencyc.

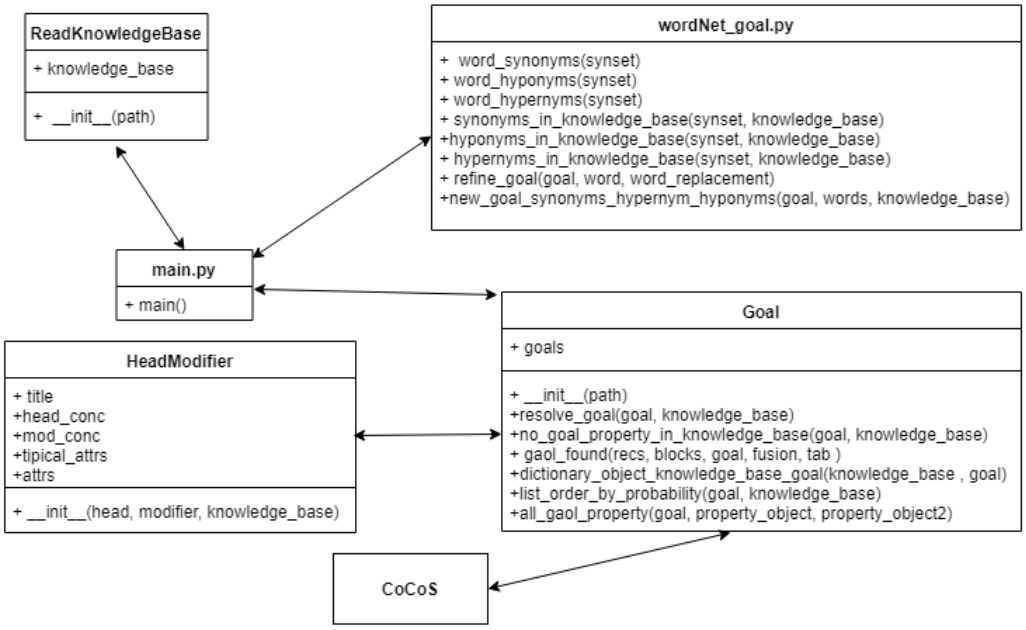

Figure 1: The architecture of the system GOCCIOLA.

vase, object	$Vase \sqsubseteq Object$
vase, high convexity	$Vase \sqsubseteq HighConvexity$
vase, ceramic, 0.8	$0.8 \ :: \ \mathbf{T}(Vase) \sqsubseteq Ceramic$
vase, to put plants, 0.9	$0.9 \ :: \ \mathbf{T}(Vase) \sqsubseteq ToPutPlants$
vase, to contain objects, 0.9	$0.9 \ :: \ \mathbf{T}(Vase) \sqsubseteq ToContainObjects$
vase, graspable, 0.9	$0.9 \ :: \ \mathbf{T}(Vase) \sqsubseteq Graspable$

To run GOCCIOLA, the user has to invoke the Python interpreter on the file *main.py*, which consults the initial knowledge base and the goal to achieve (classes *ReadKnowledgeBase* and *Goal*, respectively). First of all, the method *resolve_goal* checks whether the knowledge base contains a concept C immediately satisfying it, i.e. exhibiting all the concepts of the goal either as rigid or typical properties: in this case, the system is done, and GOCCIOLA ends its computation by suggesting such a concept C as the solution. In case C does not exist, the system GOCCIOLA tries to extend the knowledge base of the agent by looking for at least two concepts, C_1 and C_2, whose combination via $\mathbf{T}^{\mathrm{CL}}$ generates a concept C' satisfying the goal. More in detail:

- GOCCIOLA computes a list of concepts of the initial knowledge base satisfying at least a property of the goal. As an example, suppose that the following inclusions belong to the knowledge base:

  $Spoon \sqsubseteq Graspable$
  $0.85 \ :: \ \mathbf{T}(Spoon) \sqsubseteq \neg Cutting$
  $0.9 \ :: \ \mathbf{T}(Vase) \sqsubseteq Graspable$
  $Vase \sqsubseteq Object$

Both *Vase* and *Spoon* are included in the list of candidate concepts to be combined;

- for each item in the list of candidate concepts to be combined, GOCCIOLA computes a rank of the concept as the sum of the probabilities of the properties also belonging to the goal, assuming a score of 1 in case of a rigid property. In the example, *Vase* is ranked as $0.9+1=1.9$, since both *Graspable* and *Object* are properties belonging to the goal: for the former we take the probability 0.9 of the typicality inclusion $\mathbf{T}(Vase) \sqsubseteq Graspable$, for the latter we provide a score of 1 since the property $Vase \sqsubseteq Object$ is rigid. Concerning the concept *Spoon*, GOCCIOLA computes a rank of 1: indeed, the only inclusion matching the goal is the rigid one $Spoon \sqsubseteq Graspable$;
- GOCCIOLA checks whether the concept obtained by combining the two candidate concepts with the highest ranks, C_1 and C_2, is able to satisfy the initial goal. GOCCIOLA computes a double attempt, by considering first C_1 as the HEAD and C_2 as the MODIFIER and, in case of failure, C_2 as the HEAD and C_1 as the MODIFIER.

In order to combine the two candidate concepts C_1 and C_2, GOCCIOLA exploits the system CoCoS [13], a tool generating scenarios and choosing the selected one(s) according to the logic $\mathbf{T}^{\text{CL}}$. The current version of the system is implemented in Pyhton and exploits the translation of an $\mathscr{ALC}+\mathbf{T}_{\mathbf{R}}^{RACL}$ knowledge base into standard $\mathscr{ALC}$ introduced in [14, 7] and adopted by the system RAT-OWL [15]. CoCoS makes use of the library `owlready2` [2] that allows one to rely on the services of efficient DL reasoners, e.g. the HermiT reasoner. GOCCIOLA also exploits WordNet sysnsets in order to extend its search space in case of a failure. In detail, if the goal contains properties not belonging to the initial knowledge base, GOCCIOLA looks for hypernyms or hyponyms in order to rewrite such properties.

5. The System EDIFICA

As already mentioned in the Introduction, we have extended the system GOCCIOLA in order to tackle the main limitations affecting the systems, namely:

- in order to achieve a given goal, GOCCIOLA randomly chooses concepts to be combined;
- GOCCIOLA is able to deal only with combinations of two concepts at a time;
- GOCCIOLA randomly chooses the two initial candidates to be combined in order to fulfill a goal;
- in order to combine concepts C_1 and C_2, the underlying concept combination mechanism based on CoCoS [13] generates all 2^n possible scenarios, where n is the sum of the number of typicality inclusions of the form $p :: \mathbf{T}(C_1) \sqsubseteq P$ and the number of typicality inclusions of the form $p :: \mathbf{T}(C_2) \sqsubseteq P$.

In this section we describe the main features of EDIFICA (https://github.com/JoddyJordan/EDIFICA-Code) and we try to point out how it significantly improves both the performance and the results proposed by its ancestor GOCCIOLA.

The goal of EDIFICA is the same of GOCCIOLA: find a concept combination based on the generation of scenarios of possible properties able to achieve a given goal. The main difference

[2] https://pythonhosted.org/Owlready2/

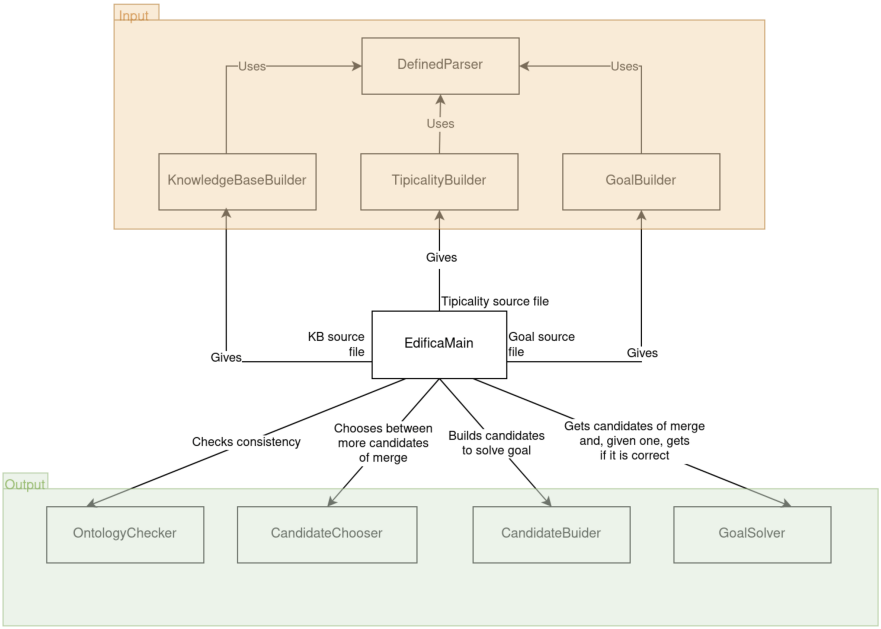

Figure 2: The architecture of the system EDIFICA.

is that EDIFICA is a tool born with an architecture that can be extended by the user. This architecture allows us to add a bunch of new features that are not present in GOCCIOLA. These new features are:

- the possibility for the user to customize the input flows: information about rigid properties, information about typical properties and information about the goal to solve;
- the possibility for the user to define a suitable module for the extraction of candidates for goal-resolution, in other words concepts to be combined for finding a solution for the initial problem;
- the possibility for the user to define a chooser of the best candidate (i.e. a set of concepts) for the goal;
- the possibility for the user to adopt a combination of an arbitrary number of concepts, with additional heuristics for the concept combination.

Furthermore, EDIFICA implements a more efficient scenario generation, in particular it implements a scheme of analysis of the properties involved in the concept combination in order to avoid the generation of inconsistent scenarios: in this way, not all 2^n scenarios are generated and – only later – discarded, they are simply not generated.

The *open* architecture of EDIFICA is shown in Figure 2: such an architecture includes a set of interfaces, each expressing a function of the system; moreover, a user can add new interfaces, i.e. new functions, or re-define existing ones.

In order to better understand how EDIFICA improves GOCCIOLA, let us conclude this section by a running example. Suppose both the systems have the following knowledge base:

$AlcoholicBeverage \sqsubseteq Beverage$
$KioskBeverage \sqsubseteq Beverage$
$HotBeverage \sqcap RoomTemperatureBeverage \sqsubseteq \bot$
$Coffee \sqsubseteq KioskBeverage$
$0.75 \;::\; \mathbf{T}(Coffee) \sqsubseteq HotBeverage$
$0.75 \;::\; \mathbf{T}(Coffee) \sqsubseteq CompanionBeverage$
$Cola \sqsubseteq KioskBeverage$
$0.75 \;::\; \mathbf{T}(Cola) \sqsubseteq ExtraSweetBeverage$
$Whisky \sqsubseteq AlcoholicBeverage$
$0.75 \;::\; \mathbf{T}(Whisky) \sqsubseteq RoomTemperatureBeverage$
$0.75 \;::\; \mathbf{T}(Whisky) \sqsubseteq CompanionBeverage$
$Rum \sqsubseteq AlcoholicBeverage$
$0.75 \;::\; \mathbf{T}(Rum) \sqsubseteq RoomTemperatureBeverage$

Let us consider both the systems to solve the following goal:

$$\{AlcoholicBeverage, KioskBeverage\}$$

EDIFICA starts by finding the candidates for the goal resolution. A simple but effective mechanism is by exploring every combination of subclasses of each single concept in the goal. The result are the following candidates:

$$\{Cola, Rum\}, \{Cola, Whisky\}, \{Coffee, Rum\}, \{Coffee, Whisky\}$$

Subsequently, EDIFICA orders the candidates by a preference-function. This function is based on two criteria: first, we order the candidates by an increasing number of concepts belonging to the candidate; second, we order the candidates with same length by a decreasing number of typical properties involved in such a candidate. The first candidate in this example is then:

$$\{Coffee, Whisky\}$$

As mentioned, in addition to the above described "intelligent" choice of the candidate concepts to be combined, EDIFICA also implements a smart generation of scenarios: rather than generating all of them as done by GOCCIOLA, EDIFICA checks two kind of relations – conflict relations and coordination relations – among typical properties. In our example, the candidate involves four typical properties, so there are 16 possible scenarios; EDIFICA checks:

- Conflict relation: two concepts in the body of the typical properties are disjoint. For instance, *HotBeverage* and *RoomTemperatureBeverage* are disjoint: we cannot have a beverage which is at the same time hot and room temperature;
- Coordination relation: two concepts in the body of the typical properties can be coordinated to be coherent: the concept *CompanionBevarage* occurs two times which means that, if in a scenario one is true, in the other one it has to be also true and vice versa.

After this analysis, EDIFICA uses a recursive algorithm that fixes at every step the truth value for the choice of a specific typical property and updates the other typical properties by using the conflict and the coordination relations. In our example, only 6 scenarios are so generated, rather than $2^4 = 16$. If each candidate scenario is inconsistent, EDIFICA will discard the candidate and pick the next list of concepts from the remaining ones.

6. Compliance and Extension of SOAR procedures

We conclude this work by showing how the proposed system EDIFICA can integrate, and extend, the classical subgoaling mechanism embedded in a cognitive architecture like SOAR [9], with a particular reference to the *impasse* mechanisms developed in such architecture. In our opinion, this compliance represents an important aspect to point out since SOAR is one of the most mature cognitive architectures and has been used by many researchers worldwide during the last 30 years in the field of cognitive modelling and intelligent systems. This system was considered by Allen Newell a candidate for a Unified Theory of Cognition [16] and still represents an important pillar in the effort of building a general integrated model of cognition [17]. This system adheres strictly to Newell and Simon's physical symbol system hypothesis [18] which states that symbolic processing is a necessary and sufficient condition for intelligent behavior. One of the main themes in SOAR is that all cognitive tasks can be represented by problem spaces that are searched by production rules grouped into operators. These production rules are fired in parallel to produce reasoning cycles. From a representational perspective, SOAR exploits symbolic representations of knowledge (called chunks) and use pattern matching to select relevant knowledge elements. Basically, when a production rule matches the contents of declarative (working) memory, then the rule fires and the content from the declarative memory (called Semantic Memory in SOAR) is retrieved.

Such type of knowledge structures, however, are usually heavily used to perform standard logical reasoning and, as a consequence, are strongly biased towards a "classical" conceptualisation of knowledge in terms of necessary or sufficient conditions and are not equipped with commonsense representational and reasoning knowledge components [3]. If a problem (an *impasse* in SOAR terms) arises due to the fact that certain knowledge is lacking, resolving this impasse automatically becomes the new goal (and this process is known as *subgoaling*). This new goal becomes a subgoal of the original one, which means that once the subgoal is achieved, control is returned to the main goal. The subgoal has its own problem space, state and possible set of operators. Whenever the subgoal has been achieved it passes its results to the main goal, thereby resolving the impasse. Learning is keyed to the subgoaling process: whenever a subgoal has been achieved, new knowledge is added to the knowledge base to prevent the impasse that produced the subgoal from occurring again (this learning process is known as *chunking*). If an impasse occurs because the consequences of an operator are unknown, and in the subgoal these consequences are subsequently found, knowledge is added to SOAR,Äôs memory about the consequences of that operator. An important feature in SOAR concerns the fact that it can also use external input as part of its impasse resolution process, therefore new knowledge can

[3]This problem arises despite the fact that the chunks in SOAR can be represented as a sort of frame-like structures containing some commonsense (e.g. prototypical) information. We remind to [19] for details analysis on such issue.

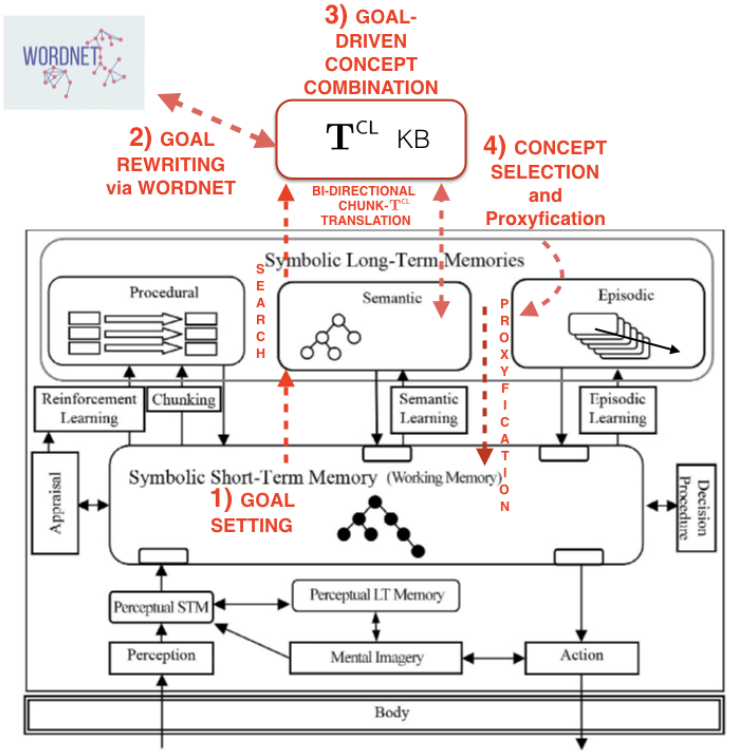

Figure 3: The SOAR Cognitive Architecture. In red, the compliance with EDIFICA in order to extend the classical subgoaling procedure.

extended the Semantic Memory of SOAR and can be incorporated into the learned rules. In this context, the proposed system can be integrated, and can extend, the SOAR subgoaling procedure as illustrated in Figure 3. The process of bi-directional translation between a chunk-like representation and the language of $\mathbf{T}^{\mathrm{CL}}$ can be provided as introduced in [20] and implemented in [15], where a typicality-based property is translated into a standard Description Logic knowledge base (corresponding to a chunk-based symbolic representation in SOAR). In particular, the overall approach is compliant with the idea of a goal-directed contextual activation of concepts obtained via a process of knowledge "proxyfication" [21] from the long-term memory to the short term memory of a cognitive agent (already employed in knowledge-based systems like DUAL-PECCS [22], integrated with different cognitive architectures, including SOAR [23, 24]). A final element emerging from the described compliance consists in the fact that that the output of the new subgoaling procedure (i.e. the novel concept dynamically generated in the KB and made available in the working memory of the architecture to solve the original goal) can be used in the SOAR learning mechanism known as *chunking*, which converts the obtained concept used to solve the goal at hand in the procedural memory of the system in order to avoid to perform *ex-novo* the same reasoning cycle in case the agent encouters againg the same goal to solve.

From a more implementative point of view, the above mentioned integration between the

Semantic Memory in SOAR (SMEM) and our hybrid KB can be obtained as follows: SMEM is accessed through two dedicated working memory channels, called `^command` and `^result`. In particular, `^command` is the branch of the working memory buffer where the GOAL setting takes places. In case the goal cannot be satisfied, this kind of request, instead of launching a standard search in the SOAR SMEM, can use our system to select which concepts can be potentially combined to extend the available knowledge and to solve the goal in hand. This kind of connection can be done by modifing the SOAR kernel and by creating novel RHS (Right Hand Side) functions able to launch our system, and its $\mathbf{T}^{\mathrm{CL}}$ knowledge base in order to take advantage of the reasoning procedure presented in the above sections. The result of this process will produce in output a novel prototype-based representation that is can be used to solve and goal. Such result can be stored in the `^result` channel, the branch of the SOAR working memory buffer devoted to acquiring the output from the external modules. Once the result of our system is "proxyfied" and the goal is solved, it can then be used in the chunking mechanism of the architecture.

7. Conclusions

We have presented EDIFICA, a tool implementing a cognitive architecture whose aim is to dynamically extend a Description Logics knowledge base by exploiting conceptual combination. EDIFICA extends GOCCIOLA by trying to tckle its main criticisms, namely a random choice of the concepts to be combined in order to achieve a goal, a "brute force" approach in the generation of possible scenarios and the limitation to solutions obtained by combining only two concepts. Moreover, the architecture of EDIFICA is open and modular and can therefore easily be personalized by the user.

EDIFICA, as well as GOCCIOLA, relies on CoCoS, a tool for combining concepts in the logic $\mathbf{T}^{\mathrm{CL}}$. In future research, we aim at studying the application of optimization techniques in [25, 26] in order to improve the efficiency of CoCoS and, as a consequence, of EDIFICA.

In future research, we aim at extending EDIFICA in order to provide a partial solution, satisfying a proper subset of the initial goals. Moreover, we are currently planning to evaluate the results proposed by EDIFICA by suitable experiments involving humans. Similarly to what done with GOCCIOLA, we have the objective of testing EDIFICA by asking it to solve some well-established and paradigmatic examples from the literature, by considering a knowledge base extending opencyc and involving humans. As mentioned in [27], there is no benchmark test available for this kind of task on both human participants and artificial systems: therefore, we aim at testing our system by comparing our results with the ones for the OROC system [27], to the best of our knowledge, the first one proposing a proof-of-concept procedure for the evaluation of concept composition, by considering the same goals they used as testbed.

References

[1] D. W. Aha, Goal reasoning: Foundations, emerging applications, and prospects., AI Magazine 39 (2018).

[2] E. Chiodino, A. Lieto, F. Perrone, G. L. Pozzato, A goal-oriented framework for knowledge invention and creative problem solving in cognitive architectures, in: G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), volume 325 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2020, pp. 2893–2894. URL: https://doi.org/10.3233/FAIA325. doi:10.3233/FAIA200440.

[3] A. Lieto, F. Perrone, G. L. Pozzato, E. Chiodino, Beyond subgoaling: A dynamic knowledge generation framework for creative problem solving in cognitive architectures, Cognitive Systems Research 58 (2019) 305–316. doi:10.1016/J.COGSYS.2019.08.005.

[4] A. Lieto, G. L. Pozzato, F. Perrone, E. Chiodino, Knowledge capturing via conceptual reframing: A goal-oriented framework for knowledge invention, in: M. Kejriwal, P. A. Szekely, R. Troncy (Eds.), Proceedings of the 10th International Conference on Knowledge Capture, K-CAP 2019, Marina Del Rey, CA, USA, November 19-21, 2019, ACM, 2019, pp. 109–114. URL: https://doi.org/10.1145/3360901. doi:10.1145/3360901.3364422.

[5] A. Lieto, G. L. Pozzato, F. Perrone, A dynamic knowledge generation system for cognitive agents, in: 31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019, IEEE, 2019, pp. 676–681. URL: https://ieeexplore.ieee.org/xpl/conhome/8970220/proceeding. doi:10.1109/ICTAI.2019.00099.

[6] A. Lieto, G. L. Pozzato, A description logic of typicality for conceptual combination, in: M. Ceci, N. Japkowicz, J. Liu, G. A. Papadopoulos, Z. W. Ras (Eds.), Foundations of Intelligent Systems - 24th International Symposium, ISMIS 2018, Limassol, Cyprus, October 29-31, 2018, Proceedings, volume 11177 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 189–199. doi:10.1007/978-3-030-01851-1_19.

[7] L. Giordano, V. Gliozzi, N. Olivetti, G. L. Pozzato, Semantic characterization of Rational Closure: from Propositional Logic to Description Logics, Artificial Intelligence 226 (2015) 1–33. doi:10.1016/j.artint.2015.05.001.

[8] F. Riguzzi, E. Bellodi, E. Lamma, R. Zese, Reasoning with probabilistic ontologies, in: Q. Yang, M. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, 2015, pp. 4310–4316. URL: http://ijcai.org/proceedings/2015.

[9] J. Laird, The Soar cognitive architecture, MIT Press, 2012.

[10] D. Lehmann, M. Magidor, What does a conditional knowledge base entail?, Artificial Intelligence 55 (1992) 1–60. doi:http://dx.doi.org/10.1016/0004-3702(92)90041-U.

[11] M. Frixione, A. Lieto, Towards an extended model of conceptual representations in formal ontologies: A typicality-based proposal., J. UCS 20 (2014) 257–276.

[12] D. N. Osherson, E. E. Smith, On the adequacy of prototype theory as a theory of concepts, Cognition 9 (1981) 35–58.

[13] A. Lieto, G. L. Pozzato, A. Valese, CoCoS: a typicality based concept combination system, in: P. Felli, M. Montali (Eds.), Proceedings of the 33rd Italian Conference on Computational Logic, Bolzano, Italy, September 20-22, 2018, volume 2214 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018, pp. 55–59. URL: https://ceur-ws.org/Vol-2214/paper6.pdf.

[14] L. Giordano, V. Gliozzi, N. Olivetti, G. L. Pozzato, Minimal model semantics and rational

closure in description logics, in: T. Eiter, B. Glimm, Y. Kazakov, M. Krötzsch (Eds.), Informal Proceedings of the 26th International Workshop on Description Logics, volume 1014 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2013, pp. 168–180.

[15] L. Giordano, V. Gliozzi, G. L. Pozzato, R. Renzulli, An efficient reasoner for description logics of typicality and rational closure, in: A. Artale, B. Glimm, R. Kontchakov (Eds.), Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017, volume 1879 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2017. URL: http://ceur-ws.org/Vol-1879/paper25.pdf.

[16] A. Newell, Unified theories of cognition, Harvard University Press, 1994.

[17] J. E. Laird, C. Lebiere, P. S. Rosenbloom, A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics., Ai Magazine 38 (2017).

[18] A. Newell, H. A. Simon, Computer science as empirical inquiry: Symbols and search, Communications of the ACM 19 (1976) 113–126.

[19] A. Lieto, C. Lebiere, A. Oltramari, The knowledge level in cognitive architectures: Current limitations and possible developments, Cognitive Systems Research 48 (2018) 39–55.

[20] L. Giordano, V. Gliozzi, N. Olivetti, G. L. Pozzato, ALC+T: a preferential extension of description logics, Fundamenta Informaticae 96 (2009) 341–372. doi:10.3233/FI-2009-185.

[21] A. Lieto, A computational framework for concept representation in cognitive systems and architectures: Concepts as heterogeneous proxytypes, Procedia Computer Science 41 (2014) 6–14.

[22] A. Lieto, D. P. Radicioni, V. Rho, A common-sense conceptual categorization system integrating heterogeneous proxytypes and the dual process of reasoning, in: In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, AAAI Press, 2015, pp. 875–881.

[23] A. Lieto, D. P. Radicioni, V. Rho, Dual peccs: a cognitive system for conceptual representation and categorization, Journal of Experimental & Theoretical Artificial Intelligence 29 (2017) 433–452.

[24] A. Lieto, D. Radicioni, V. Rho, E. Mensa, Towards a unifying framework for conceptual represention and reasoning in cognitive systems, Intelligenza Artificiale 11 (2017) 139–153.

[25] M. Alberti, E. Bellodi, G. Cota, F. Riguzzi, R. Zese, cplint on SWISH: probabilistic logical inference with a web browser, Intelligenza Artificiale 11 (2017) 47–64. doi:10.3233/IA-170106.

[26] E. Bellodi, E. Lamma, F. Riguzzi, R. Zese, G. Cota, A web system for reasoning with probabilistic OWL, Journal of Software: Practice and Experience 47 (2017) 125–142. doi:10.1002/spe.2410.

[27] A.-M. Olteţeanu, Z. Falomir, Object replacement and object composition in a creative cognitive system. towards a computational solver of the alternative uses test, Cognitive Systems Research 39 (2016) 15–32.