From Natural Language to Argumentation and Cognitive Systems

Theodoros Mitsikas
School of Applied
Mathematical and Physical Sciences,
National Technical University of Athens,
mitsikas@central.ntua.gr

Nikolaos Spanoudakis
School of Production Engineering and Management,
Technical University of Crete,
nikos@science.tuc.gr

Petros Stefaneas
School of Applied Mathematical and Physical Sciences,
National Technical University of Athens,
petros@math.ntua.gr

Antonis Kakas
Department of Computer Science,
University of Cyprus,
antonis@ucy.ac.cy

Abstract

This paper reports on a program for the development of cognitive systems based on user specifications expressed in Natural Language in the form of high-level guidelines for the desired behaviour of those systems. Language analysis from existing NLP tools is used to generate scenarios based on the users' description of their preferences in the provision of the service by the system. These scenarios are combined with the *SoDA* Methodology to develop and generate an argumentation theory that captures the guidelines of operation given in the natural language description of the user. The intended architecture of a fully automated system that generates argumentation based cognitive systems from natural language is presented, together with a first analysis of the challenges posed and evaluation of progress in addressing them. This system is called *Gorgias-NL*, reflecting the central task of the approach to provide a natural language interface to the *Gorgias* argumentation system.

1 Introduction

We are today experiencing a dramatic increase in the interest of developing systems that have a human-like behaviour in providing personalized services to users. This is mainly driven by the market for systems that will be able to help people in their every day life in a *naturally intelligent* way. Such systems, fall under the area of Cognitive Systems (Langley, Laird, and Rogers 2009; Langley 2012; Michael et al. 2015) needing to interact with their users in a natural way and generally exhibit a behaviour that is cognitively compatible with that of human users without any technical knowledge, but rather relying on the *common sense knowledge* that people have and utilize through their use of *natural language*. In general, cognitive systems are thus envisaged to capture and learn their users' personal requirements through dialogues in natural language that are high-level and non-reliant on technical knowledge. The central challenge then, for building such systems, is to bridge the gap between Natural Language used by humans and Machine Languages through which these systems are built and executed on today's computers.

For example, we would like a user to be able to specify her/his guidelines for their own Cognitive Assistant for online shopping in the following manner:

> "The quality of food is very important for me. When possible try to economize. I like to eat organic food. I am not diabetic but I like to avoid sugary foods. I prefer not to eat red meat except for special occasions."

This would be followed by implicit evaluations of purchases suggested or made by the cognitive assistant, e.g.:

> "The fish last night was very good but not enough."

Such advanced expressions of specification forms the ultimate final goal of our work, namely, to be able to automatically capture these natural language specifications and their contextual meaning so that we can build automated systems whose operation is influenced or governed by these guidelines. This goal can be addressed, to a large extent, modularly in two stages. In the first stage the challenge is the linguistic extraction of (i) the *predicate relations* expressed in the natural language text and (ii) the *preference meaning* conveyed explicitly or implicitly in the text.

This preference information is relative amongst different entities at various levels, e.g. in the example above a preference of quality over economy is expressed as well as that of organic food over non-organic food. Terms like quality or economy of food do not have a full meaning until they are immersed into common sense knowledge in the context of the application discourse. Acquiring, organizing and using the relevant common sense knowledge forms a second part of our program for the development of these systems, outside the scope of this paper.

1.1 Argumentation for Cognitive Systems

The basis of our approach will be that of *argumentation* and its computational form as developed in Artificial Intelligence over the last two decades (see e.g. (Bench-Capon and Dunne 2007; Rahwan and Simari 2009) for an overall view). Argumentation will form the "middleware" between the natural language user input and the machine language on upon which the systems will be implemented. There are many reasons for choosing argumentation as the "technology basis", such as its natural link to human reasoning/thinking (Mercier and Sperber 2011) and the computational form that it exhibits in its dialectical process.

Specifically, our approach aims to produce, from the natural language user guidelines, an argumentation theory composed of arguments, a conflict relation between arguments and preference information that would give strength of some arguments over others. As mentioned above, we will separate the task into two phases. In the first phase we use standard Natural Language Processing (NLP) tools, such as that of Stanford Core NLP (Manning et al. 2014) to syntactically analyse the text sentences of the guidelines. This analysis is used to extract the symbolic predicate relations that the guidelines document contains. These are then used to form typical *application scenarios* in terms of collections of conditions under which various decisions or options are sanctioned. Hence, in the example above we may form a scenario where "a food with high quality sanctions the option to buy it" whereas another scenario would specify that "a food with a high price sanctions it the decision not to buy it".

Preference information is also based on the linguistic analysis of the text and patterns of expression in the text which contain implicitly the preferences of the user. The aim is to capture these through refined scenarios where the options sanctioned differ from those of the more general scenario. For example, we would have a general scenario where "food that contains red meat is sanctioned as not suitably to buy" but a refined scenario of "food that contains red meat for a special occasion is sanctioned as suitably to buy". Sometimes this would be clear from the text but not always. For example what is the preference of the user above for food of high-quality but also of high-price? Such combined scenarios could be put to the user for expressing more directly her/his preferences or could be kept "in mind" by the system waiting for further information on the possible user preference.

The consideration of combined application scenarios follows the *SoDA* (Software Development through Argumentation) Methodology (Spanoudakis, Kakas, and Moraitis 2016; 2017) for developing applications of argumentation. This methodology gives a principled and systematic way of considering preferences in scenarios and their refinements. This methodology then enables the automatic generation of an argumentation theory within the preference-based argumentation framework of the *Gorgias* system (Kakas, Mancarella, and Dung 1994; Dimopoulos and Kakas 1995; Kakas and Moraitis 2003) capturing the scenario information. The generated argumentation theory and implementation code is transparent to the user but can be executed directly under the *Gorgias* system to suggest choices under the user guidelines and to provide an explanation supporting these choices in terms of the argumentation process that makes them suitable choices.

Our approach is implemented in a system, called *Gorgias-NL*, whose aim is to pipeline together these different stages of analysis from the natural language text of the user to the *Gorgias* argumentation code that captures this.

1.2 Related Work

Our approach falls within the general effort of developing *human-like AI* systems that link directly with their human users in Natural Language. Such systems include proprietary personal assistants, e.g. Alexa, Cortana, Siri, and Google Assistant. Their development is driven by their direct link to the market and they use methods targeted to the specific needs of their application. Google Assistant[1] and Alexa[2] use phrase matching, which triggers the appropriate action or event. Siri[3] registers specific vocabulary to identify user's intents. Cortana analyses queries using bag-of-words models and a classifier, after eliminating noise (non-alphabetic and non-English words) and applying stemming (Elwany and Shakeri 2014). Similarly, in IBM's Watson, linguistic analysis of natural language input is carried out by two deep parsing components, an English Slot Grammar (ESG) parser and a predicate-argument structure (PAS), followed by pattern-based relation extraction (McCord, Murdock, and Boguraev 2012; Lally et al. 2012).

Other notable assistants are the open-source projects Lucida[4] and Mycroft[5]. Lucida uses OpenEphyra, a Java framework, as an internal parser to extract textual information and answer queries, plus a Wikipedia database stored in Lemur's Indri format. Mycroft's parsers take in a sentence and attempt to find user's intention by searching for matching words and phrases from registered intents and vocabulary.

In comparison, our approach rests on argumentation as a principled foundation for capturing human reasoning. The emphasis is not so much on the underlying learning technology for analysing natural language input but on the subsequent use of the information extracted. Our aim is to attempt to give this information a symbolic logical form, using argumentation as a basis for common sense logical reasoning, rather than a direct operational meaning. Also unlike many methods for Natural Language Understanding our linguistic analysis does not aim to translate directly the text into some formal logical (classical or non-classical) language but rather to form typical scenarios or partial models that capture the meaning of the text.

After a brief review of argumentation, the next section describes the extraction of the predicate relations expressed in the natural language text. It also gives the overall architecture of the *Gorgias-NL system* that implements our approach. The functionality of the system is illustrated through an example of scenario generation and argument extraction. The initial evaluation, challenges concerning open NLP problems, and the challenges emerging through our development and evaluation process are presented in Section 4. Finally, in Section 5 we conclude with a brief description of future work needed to complete our program of work.

2 Argumentation and SoDA Methodology

In this section we briefly review the basic theory of argumentation on which our approach rests and the $SoDA$ Methodology used for developing the argumentation code that capture the user's guidelines. This framework was proposed in (Kakas, Mancarella, and Dung 1994) and further developed later in (Kakas and Moraitis 2003) as a preference-based

[1] https://developers.google.com/actions/
[2] https://aws.amazon.com/lex/
[3] https://developer.apple.com/sirikit/
[4] http://www.lucida.ai/
[5] https://mycroft.ai/

argumentation framework with a view to support flexible decision making for autonomous agents.

Application requirements are captured via argumentation theories composed of arguments at different levels. Object level arguments support the possible decisions, or options, in the specific application domain, while first-level priority arguments express preferences on the object level arguments in order to resolve possible conflicts. Higher-order priority arguments are also used to resolve potential conflicts between priority arguments of the first or subsequent levels.

Argumentation theories are expressed in this framework in the language of rules and priorities on rules. An *argumentation theory* is a pair $(\mathcal{T}, \mathcal{P})$ whose sentences are rules of the form $L \leftarrow Body$, where $Body$ is a conjunction of positive or negative literals. Rules in $\mathcal{T}$ capture argument schemes for building **object level arguments**. On the other hand, rules in $\mathcal{P}$ represent argument schemes for building **priority arguments**. The head L of these rules has the general form, L =$prefer(rule1, rule2)$, where $rule1$ and $rule2$ are atoms naming two rules and $prefer$ refers to an (irreflexive) *higher priority* relation amongst the rules of the theory.

The semantics of an argumentation theory is defined via the standard notion of admissibility within an abstract argumentation framework $< Args, Att >$ associated to any given theory $(\mathcal{T}, \mathcal{P})$. In this the attack relation is defined through the conflicts between literals or between opposing priorities, prefer(r,r') and prefer(r',r) and a notion of relative strength for the attacking argument that is build through the priority rules that the arguments contain (see (Kakas and Moraitis 2003) for the details).

The simplicity of this argumentation framework facilitates the systematic *"Software Development for Argumentation"(SoDA)* methodology for developing applications of argumentation through a process of directly mapping the user or system requirements to an argumentation theory. The main feature of this methodology is the incremental consideration of scenarios in which the developer is asked to state the default preferred behaviour (i.e. preferred options) of the system and subsequently to consider refinements of the scenarios where this preference changes. The methodology concentrates on scenarios where conflicting options can occur and for each such case a process of refinement is initiated. A tool, called *Gorgias-B* (http://gorgiasb.tuc.gr), helps the developer to consider his/her application according to the *SoDA* methodology and offers a high-level environment through which the software code of the underlying argumentation theory is automatically generated.

The important characteristic of the methodology that stems from the declarative nature of the underlying argumentation framework is that the interaction with the developer is carried out in the high-level application language and hence no technical knowledge is needed by the developer. This is very important in our work as it indeed allows the users to express their requirements in Natural Language without the need to consider how this will be mapped into an argumentation theory or rules and priority rules.

3 Gorgias-NL Design and First Version

Our approach is based on extracting typical scenarios representing the user's preferences from the text document. A two step process is used. First, initial scenarios are extracted via a natural language analysis. Subsequently, the *SoDA* Methodology is applied, where the combined and refined scenarios are extracted. These scenarios constitute a complete argumentation software which implements the user's preferences.

3.1 Natural Language Understanding for Cognitive Assistants

A Cognitive Assistant must be capable of extracting the general and specific contexts via language analysis, without the need for detailed operational instructions. The user's language and vocabulary used in this domain consist of a subset of a Natural Language concerning everyday tasks.

The user will set the general guidelines and probably place an emphasis on distinct cases, possibly expressing exceptions to the guidelines s/he sets. This interaction has not necessarily an order or a time frame. New guidelines or exceptions can arise, as the user needs, the environment, and the cognitive system evolve. A guideline can contain conditions, exceptions, actions, or any combination of them.

The extraction of guidelines and preferences involves finding important words of a sentence and generating a knowledge representation corresponding to the user's guidelines. Combined with a representation of commonsense knowledge, the Cognitive System can reason over user preferences and user queries expressed using the same subset of Natural Language. These would typically provide information about the user's current needs and request a recommendation on what decision to take. The query, considers the current environment of the user and returns options that fall under the personal guidelines of the user.

3.2 Gorgias-NL Architecture

Gorgias-NL is based on a modular design, which is depicted in Fig. 1. All modules can function independently, and they communicate using simple specifications. Firstly, the user's text input concerning preferences and policies, is processed by the NLP module. The NLP module relies on state-of-the-art NLP software to obtain the language analysis for the given text input. Language analysis includes *coreference resolution*, i.e. determining possible expressions that refer to the same person, object or thing. At the current stage of implementation, those are performed by Stanford NLP, but any other software that outputs CoNNL-U data can be used with minor modifications.

Afterwards, the module responsible for scenario extraction uses the language analysis input to generate the preliminary scenarios. Although developed as a module for *Gorgias-NL*, it can be used independently as it generates conditions and options/actions from natural language. Within *Gorgias-NL*, the generated scenarios capture the user's general preferences.

Scenario generation is followed by application of the *SoDA* Methodology, to obtain the refined scenarios which capture the general user preferences. They form a consistent set of arguments, to be later used at the decision level.

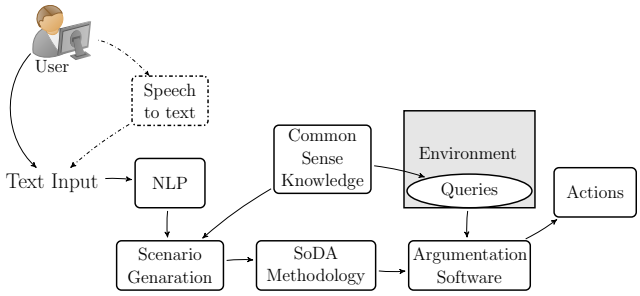

Figure 1: Design of the prototype system

The derived final argumentation software captures the preferred guidelines of operation of the user, while being consistent. Combined with the common sense knowledge, the environment, and user specific knowledge, queries can be answered or actions can be taken.

3.3 Scenario generation

For the scenario generation, the prototype is using a strict grammar-to-model parser. Syntactic structures are translated to predicates, along with their corresponding terms, in order to extract the first level symbolic information. Predicates derived from the root of the sentence, identify the option or the decision of the scenario. An overview of this is as follows:

Copular Verbs The predicate is the conditional clause, while the subject is a term (e.g. "the coupon **is** out-of-date" generates the predicate `out_of_date(coupon)`).

Auxiliary Verbs At this initial stage of our work we are assuming that auxiliary verbs do not form part of the predicate. Such verbs can qualify the predicate and its link to other information in the sentence but this would only be important if it makes a difference on a typical scenario.

Verbs The other verbs are predicates, while their respective subjects, objects, and noun modifiers are terms. In the domain of a "cognitive assistant", a greater focus needs to be given in imperative mood.

Adjectives Adjective phrases are predicates, and the corresponding noun is the term (e.g. "Pick up some **cheap** wine…" is `cheap(wine)`).

Adverbs Adverbs and adverb phrases can be terms (e.g. "Wake me up **tomorrow**" is `wake_me_up(tomorrow)`), or can be ignored.

Prepositions Prepositions are used as a variation of the predicate (e.g. "When I'm **at** work…" is `at_work`), or can be ignored.

Lemmatisation Predicates and terms are lemmatised. Exceptions to this include gerund, participles, and past tense.

Preference Several adverbs and prepositions can state explicitly a user preference (e.g. "**Normally** allow calls", "I prefer fish **over** meat").

This analysis can ignore some words of a sentence. While some information can be lost, it is not always essential. Similar to (McCord, Murdock, and Boguraev 2012), we are ignoring auxiliary verbs and words that introduce verb phrases (e.g. "that", if it is introducing a "that" clause). Forms of "be" are also ignored, e.g. if it is a copular verb. Determiners as possessive pronouns are taken into account. Articles, while they do not appear in the final form of scenarios, are used to determine if a term is a variable or a constant, together with the grammatical number of nouns.

The generated scenarios are of the form `<scenario_id, sentence_id, {Conditions}, Options>`. The notation is as follows: `scenario_id` is a number used to identify the scenario, `sentence_id` is a list of numbers corresponding to the natural language sentence(s) which the scenario refers to, `Conditions` is a list of zero or more predicates defining the context that constitutes the scenario, and `Options` is a list of one or more predicates corresponding to decision options and which are valid in the scenario.

To illustrate how scenarios are extracted from the guidelines text consider the following example concerning the management of discount offers. *"Normally, discard coupons. If a coupon is related to my wish list, save it, unless it is expensive. If a coupon offers a large discount, save it. Discard the coupons that are out-of-date."* After coreference resolution and language analysis performed by Stanford NLP, the scenario generation yields the following scenarios:

```
<1, 1, {}, discard(Coupon)>
<2, 2, {related_to(Coupon,wish_list)},
  save(Coupon)>
<3, 2, {expensive(Coupon),
  related_to(Coupon,wish_list)},
  neg(save(Coupon))>
<4, 3, {large(discount),
  offer(Coupon,discount)}, save(Coupon)>
<5, 4, {out_of_date(Coupon)},
  discard(Coupon)>
```

As shown above, each sentence can generate more than one scenario, e.g. if an exception is included. The latter part of the second sentence *"unless the coupon is expensive"* implies an additional scenario which includes the extra condition `expensive(Coupon)` and the unique option `neg(save(Coupon))`. This option is derived from the initial natural language sentence which does not provide an alternative option, thus the negation is used. For simplicity, we will assume that we have additional information to identify the two options of not saving and discarding.

3.4 Argument extraction from context scenarios

Given a set of primary scenarios extracted directly from the text as in the above example we then need to consider combinations of these scenarios to further capture the guidelines. The *SoDA* methodology helps us to consider such combinations in a structured and systematic way. For example, considering the combination of scenarios 2 and 5 we get a new combined scenario:

```
<6, {2,4}, {related_to(C,wish_list),
  out_of_date(C)}, {save(C), discard(C)}>
```

where initially both options are possible. At this stage, we consider the task of making a combined scenario more deterministic by trying to ascertain if the user has some (implicit) preference in such a scenario. For example, we can use pragmatic information that comes from the common sense knowl-

edge that typically out of date coupons to give preference to the discard option. This is a heuristic process that can also be captured through argumentation, namely using the common sense and practical information to support arguments for the various options in a combined scenario, as in the example above where we have a strong argument to discard. In cases, where it is not possible to decide in favour of a smaller set of preferred options in a combined scenario we can either ask the user for extra information on this or we can wait until some information that can help with this is acquired by the cognitive assistant. In our running example one possible outcome of this process could give the following combined scenarios.

```
<7, {3,4}, {large(discount),
  offer(C,discount), out_of_date(C)},
  discard(C)>
<8, {2,3}, {expensive(C),
  related_to(C,wish_list),
  large(discount), offer(C,discount)},
  {save(C), discard(C)}>
<9, {2,3,4}, {expensive(C),
  related_to(C,wish_list),large(discount),
  offer(C,discount), out_of_date(C)},
  discard(C)>
```

An algorithm is then applied that results in the generation of a $Gorgias$ argumentation theory from such context scenarios extracted from the text. This algorithm first constructs a *context graph* that puts scenarios in levels from the more general contexts to more specific ones. The *context graph* is a graph whose nodes correspond to scenarios. The nodes have a *name* (corresponding to a scenario), a set of *options* that are valid in the node's scenario and a *level* integer. Its root (first node that has only outgoing edges) is a node named *true*.

The algorithm to transform scenarios to such a graph starts from the "true scenario" at the root and then gradually adds scenarios from simple ones to more complex connecting with an edge those that are more specific contexts of others. Fig. 2 shows this initial graph for our example. Note that each node has its name on its label. For simplicity of representation, and since we only have two options, the valid option at each node is denoted by the colour of its background, white for *discard(C)* and grey for *save(C)*. Hence a note which is half white and half grey, e.g. node 4.1, designates that both options are possible. The context graph is refined in two passes. The

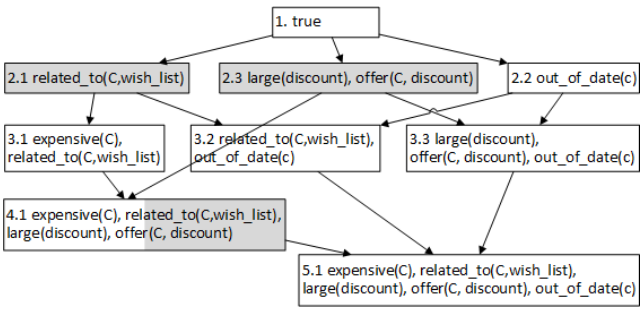

Figure 2: The graph of the generated scenarios

first one removes all edges that connect nodes whose source node and target node have the same set of options. The second pass removes all nodes that are not the root and there is no edge having them as targets. The final context graph for our example is shown in Fig. 3.

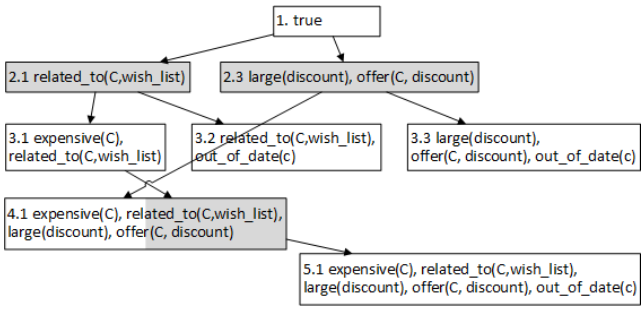

Figure 3: The refined context graph after the second step where all edges that connect same coloured nodes have been eliminated along with node 2.2 that becomes orphan

The algorithm then transforms the context graph into a structure called *threads* in the *SoDA* methodology. This considers any path from the root to a leaf for each pair of options and creates a corresponding thread by defining all the possible contexts - combinations of conditions - from the most general to the more specific, under which a decision between a pair of options is reached. In general, a thread, T^{ij}, is based on a pair of conflicting options O_i and O_j, and is a sequence, $\{P^k\}$ $(1 \leq k \leq n)$, where each P^k has the form, $P^k =< S^k, C^k_{ij}, C^k_{ji} >$, where S^k, C^k_{ij}, C^k_{ji} are sets of scenario conditions, such that $S^k = S^{k-1} \cup C^{k-1}_{ij} \cup C^{k-1}_{ji}$ (and hence $S^1 \subseteq S^2 \subseteq ... \subseteq S^n$). Here C^k_{ij} and C^k_{ji} are contextual conditions (possibly empty) such that O_i is **preferred over** O_j in the consistent scenario $S^k \cup C^k_{ij}$ and O_j is **preferred over** O_i in the consistent scenario $S^k \cup C^k_{ji}$.

Hence, in a thread based on two conflicting options we iteratively consider (consistent) refinements of an application scenario and at each such level, k, of iteration new *contextual* information is considered, denoted by C^k_{ij} or C^k_{ji}, under which one of the options is preferred over the other. Contextual information can be given for none of the preferences (in which case both C^k_{ij} and C^k_{ji} are equal to the empty set), for one or for both preferences between the two options O_i and O_j. In the special case of the contextual information not imposing any further refinement of the current scenario, i.e. when $C^k_{ij} = \{true\}$ (or $C^k_{ji} = \{true\}$), we have a **default preference** (at level k) of O_i over O_j (resp. O_j over O_i) and the thread stops.

Algorithm 1 shows the process of transforming the context graph to threads. Figure 4 shows the four threads for our example. The first thread T1 is given below, where it should be noted that the node with two valid options causes the T1 thread to fork creating T12 and T11:

for *each pair of conflicting options* **do**
  for *each path that contains these options* **do**
    Start a new thread **for** *each node starting from the root to the leaf* **do**
      add $P^{level} =< S^{level} = \{node\_name\}, C_{ij}^{level} = \{\}, C_{ji}^{level} = \{\} >$
      if *(O_i is valid in node))* **then**
        $C_{ij}^{level} = \{true\}$
      if *(O_j is valid in node))* **then**
        $C_{ji}^{level} = \{true\}$
      if $C_{ij}^{level} \neq \{\} \wedge C_{ji}^{level} \neq \{\}$ **then**
        This is the special case where both options are valid in this node. Fork the thread and make the $C_{ij}^{level} = \{true\}$ in the existing thread and $C_{ji}^{level} = \{true\}$ in the other thread
      for *each next_node starting from level towards the leaf* **do**
        if $(O_i \in option(next\_node)) \wedge C_{ij}^{level} = \{\}$ **then**
          $C_{ij}^{level} = \{name(next\_node) - name(node)\}$
          node = next_node

Algorithm 1: For generating the scenarios graph

$$
\begin{aligned}
&T^1(O_1 = discard(C), O_2 = save(C)) = \{\\
&P^0 =< S^0 = \{true\}, C_{12}^0 = \{true\},\\
&\quad C_{21}^0 = \{related\_to(C, wish\_list)\} >,\\
&P^1 =< S^1 = \{related\_to(C, wish\_list)\},\\
&\quad C_{12}^1 = \{expensive(C)\}, C_{21}^1 = \{true\} >,\\
&P^2 < S^2 = \{related\_to(C, wish\_list), expensive(C)\},\\
&\quad C_{12}^2 = \{true\},\\
&P^3 =< S^3 = \{related\_to(C, wish\_list),\\
&\quad expensive(C), large(discount), offer(C, discount)\},\\
&\quad C_{12}^3 = \{out\_of\_date(C)\}, C_{21}^3 = \{true\} >,\\
&P^4 =< S^4 = \{related\_to(C, wish\_list),\\
&\quad expensive(C), large(discount),\\
&\quad offer(C, discount), out\_of\_date(C)\},\\
&\quad C_{12}^4 = \{true\}, C_{21}^4 = \{\} >\}\\
&T^{12}(O_1 = discard(C), O_2 = save(C)) = \{\\
&P^3 =< S^3 = \{related\_to(C, wish\_list),\\
&\quad expensive(C), large(discount),\\
&\quad offer(C, discount)\}, C_{12}^3 = \{true\}, C_{21}^3 = \{\} >,\\
&T^{11}(O_1 = discard(C), O_2 = save(C)) = \{\\
&P^1 =< S^1 = \{related\_to(C, wish\_list)\},\\
&\quad C_{12}^1 = \{out\_of\_date(C)\}, C_{21}^1 = \{true\} >,\\
&P^2 =< S^2 = \{related\_to(C, wish\_list),\\
&\quad out\_of\_date(C)\}, C_{12}^2 = \{true\}, C_{21}^2 = \{\} >\}
\end{aligned}
$$

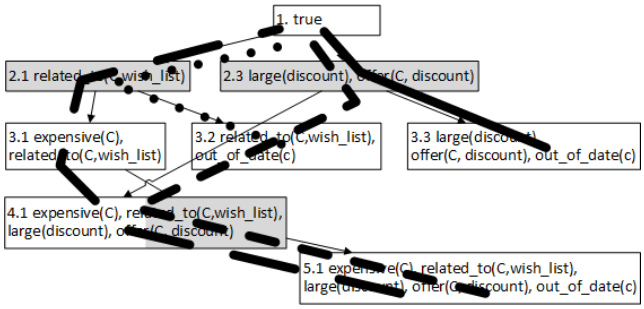

Figure 4: The threads depicted graphically in our context graph. T1 is the bold dashed line (with large dashes), T11 the forked thread from the previous one shown with the dotted line and T2 shown with the solid line. T22 forks from T2 (dashed line with small dashes). Note that T22 does not include node 3.2 (the arrow just passes over it in the figure)

From the threads we can automatically generate a corresponding *Gorgias* argumentation theory that captures the arguments for each of the options involved in a thread and their relative strength at each level of the thread. This is done by generating argument rules that support the priority of one argument rule over another at the previous level. The full details of this are omitted due to lack of space. They can be found in (Spanoudakis, Kakas, and Moraitis 2017).

The generated argumentation theory can be executed using the *Gorgias-B* tool (http://gorgiasb.tuc.gr) to reach decisions under given current circumstances. For our coupons example, given a specific coupon c_1 for which we know it offers a large discount *Gorgias-B* will return the decision *save(c1)* together with the explanation:

"*When [offer(c1, discount), large(discount)] choose save(c1)*"

based on the arguments that support this decision. If in addition we know that this coupon is out of date (e.g. as time passes) *Gorgias-B* will now return the decision *discard(c1)* together with the new explanation:

"*When [offer(c1, discount), out_of_date(c1), large(discount)] prefer discard(c1) over save(c1)*".

These explanations can now be transformed into natural language to be presented to the user if and when requested.

4 Evaluation and Challenges

Our initial evaluation sought to establish evidence for the feasibility of our approach, to uncover possible weaknesses in the implementation and to point out future challenges. We describe here briefly our plans for evaluation of our approach.

4.1 Evaluation

There is no widely accepted evaluation data or tests relevant to the domain of a cognitive assistant. In order to obtain a preliminary evaluation of the prototype system, we asked a group of users to express their preferences in a very short paragraph in natural language, on a task of their own choice.

The generated scenarios captured satisfactory the overall preferences of roughly 70% of the sentences expressed by the users. Examples of user preferences are: (i)*"Categorize work emails by their code in the subject. If I'm cced, put the email in the relevant folder. If the email is personal, read the subject of the email to me, except if I'm at a doctor."* and (ii) *"If I have a meeting with a client, make sure the appropriate documents according to the client are on my desktop. If I have a meeting with Mr. Doe interrupt us after 10 minutes with something urgent! Remind me to pick up my kids every day at 3:00 except from Tuesday, when you should remind me at 1:00."*. Examples like the last two sentences were only partially captured, due to duration, date and time mentions, which the prototype did not support during the evaluation stages.

We have used this evaluation exercise to help improve our natural language parser into scenarios. In future we will continue to perform such evaluation exercises, with larger groups of users, for early feedback on the system's performance.

We also note that the evaluation process has reveal interesting and useful information about how human users perceive cognitive systems/assistants. For example, initially no user expressed a high-level policy or an exception within a policy. Some users expressed simple if-then rules or simple commands (those are excluded from the above results). Other users, initially expressed sentences containing an excessive amount of non-essential information. More complex cases with exceptions, and without inessential information were expressed after an example was given, or after specific instructions to express themselves more freely, as in a conversation with an "intelligent" human. One example concerning a shopping assistant, specified by a user, is the sentence *"I would like you to inform me or alert me when you find through Google medium dresses for the summer in green or white colour."*, which our parser failed to capture. After a request to repeat the same guideline as it would be addressed to a human personal assistant, the user rephrased the sentence as *"Tell me if you find medium summer dresses in green or white."*, which the parser captured. This raises interesting questions about how people perceive an artificial cognitive assistant and how thus they behave when interacting with such a system.

4.2 Future Linguistic Challenges

Challenges concerning the interaction of the user with a cognitive system through automatic NLP include:

- Language analysis, coreference and ellipsis resolution of NLP software is not, yet, perfect. Furthermore, the analysis of complex sentences depends on the right punctuation of user's input. One example is the phrase *"If I have a meeting with a client, make sure the appropriate documents, according to the client, are on top of the desk"*. Depending on where the commas are, the language analysis differs, and the respectively generated scenarios do not necessarily capture the user's preferences.
- Language variability: Reasoning on the generated scenarios can be impossible without using the context, even for relatively simple inputs. Consider the following sentences: *Normally allow calls. If an advertiser calls, don't answer the call.* If the parser is strictly based on grammar and structures, the generated first scenario is `<1,1,{},allow(Call)>`. In the second sentence, the word "calls" is a verb, unlike the word "calls" in the first sentence which is a noun and the direct object. That will lead to different predicates and terms comparing e.g. to the sentence "Deny calls from advertisers". Moreover, the possible different predicates e.g. allow, answer and deny, don't answer, are the same, respectively, in the context of how to handle a phone call, given this simple policy.
- Communication between human and computer in a context of a Cognitive Personal Assistant requires identification of the type of a sentence. A given sentence can be a query (e.g. *"Where is my car?"*), a command (e.g. *"Turn off the lights."*), a fact (e.g. *"I parked my car at... "*), or part of a policy (e.g. *"Normally discard coupons."*). Identifying the type of the sentence is can benefit from work on question and intent classification.

Together with these challenges we also plan to explore other state of the art methods and tools for NLP such as neural networks with grammar-to-model training, or the use of Combinatory Categorial Grammars, e.g. openCCG [6] and compare their utility.

5 Conclusions and Future Work

We have presented the design of *Gorgias-NL*, an argumentation-based system for building cognitive personal assistants, together with its first implementation. *Gorgias-NL* is based on a modular design, which uses a NLP parser that generates scenarios from user guidelines and a module following the *SoDA* methodology to generate an argumentation theory capturing the guidelines.

We plan to continue the process of further development of the system together with an evaluation with human users so that we can test early the ecological validity of our design decisions. In a fully automated system we aim to allow the user to confirm or not its recommendations or if the user wishes to engage into a (structured) dialogue with the system in order to be persuaded or not about the recommendation of the cognitive assistant. This dialogue will form a new input for the system that could enrich the argumentation theory capturing the user's guidelines.

The most important future development is that of exploring the use of common sense knowledge in the approach and the design of our systems. Its use will be important at various levels. For example, it can be used in helping to understand implicit preferences of the user and thus resolve conflicts when we consider combinations of scenarios under the *SoDA* Methodology.

Common sense Knowledge will also be important when our cognitive systems, once build, are deployed in a real environment where they will need to decide on what options are best suited for their users. During their execution they will need to link the current "sensory information" that the system will be getting from its application environment with

[6] `http://openccg.sourceforge.net/`

the high-level concepts used to express the personal user guidelines. Common Sense Knowledge will help recognize which of these high-level concepts hold in any given current situation and hence which arguments apply in the current case of reasoning by the system. This recognition can be done as a subsidiary argumentation process on the beliefs of the system based on common sense arguments, sanctioning them from the sensory information framework (see (Diakidoy et al. 2014) where common sense knowledge is used in this way for narrative text comprehension). The *Gorgias* framework supports this two-level argumentation process, one for options and one for beliefs on which options are decided and hence this facilitates the integration of common sense knowledge uniformly in terms of argumentation. Nevertheless, the main challenge with common sense knowledge is acquiring, organizing, and accessing the relevant part of common sense knowledge when this needs to be used.

References

Bench-Capon, T. J. M., and Dunne, P. E. 2007. Argumentation in Artificial Intelligence. *Artificial Intelligence* 171(10-15):619–641.

Diakidoy, I.-A.; A.; Michael, L.; and Miller, R. 2014. Story Comprehension through Argumentation. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA)*, 31–42.

Dimopoulos, Y., and Kakas, A. C. 1995. Logic programming without negation as failure. In *Logic Programming, Proceedings of the 1995 International Symposium, Portland, Oregon, USA, December 4-7, 1995*, 369–383.

Elwany, E., and Shakeri, S. 2014. Enhancing Cortana User Experience Using Machine Learning. *Recall* 55(54.61):24–24.

Kakas, A., and Moraitis, P. 2003. Argumentation based decision making for autonomous agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, 883–890. New York, NY, USA: ACM.

Kakas, A. C.; Mancarella, P.; and Dung, P. M. 1994. The acceptability semantics for logic programs. In *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, 504–519.

Lally, A.; Prager, J. M.; McCord, M. C.; Boguraev, B. K.; Patwardhan, S.; Fan, J.; Fodor, P.; and Chu-Carroll, J. 2012. Question analysis: How Watson reads a clue. *IBM Journal of Research and Development* 56(3.4):2:1–2:14.

Langley, P.; Laird, J. E.; and Rogers, S. 2009. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research* 10(2):141 – 160.

Langley, P. 2012. The cognitive systems paradigm. In *Advances in Cognitive Systems*, 3–13.

Manning, C. D.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, S. J.; and McClosky, D. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, 55–60.

McCord, M. C.; Murdock, J. W.; and Boguraev, B. K. 2012. Deep parsing in Watson. *IBM Journal of Research and Development* 56(3.4):3:1–3:15.

Mercier, H., and Sperber, D. 2011. Why Do Humans Reason? Arguments for an Argumentative Theory. *Behavioral and Brain Sciences* 34(2):57–74.

Michael, L.; Kakas, A.; Miller, R.; and Turán, G. 2015. Cognitive Programming. In *Proceedings of the 3rd International Workshop on Artificial Intelligence and Cognition (AIC)*, 3–18.

Rahwan, I., and Simari, G. R. 2009. *Argumentation in Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st edition.

Spanoudakis, N. I.; Kakas, A. C.; and Moraitis, P. 2016. Applications of Argumentation: The SoDA Methodology. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands*, 1722–1723.

Spanoudakis, N. I.; Kakas, A. C.; and Moraitis, P. 2017. The SoDA Methodology and Gorgias-B Tool for Developing Argumentation Systems, AMCL-TR_01_2017. Technical report, Applied Mathematics and Computers Laboratory, Technical University of Crete, Chania, Greece.