State of the Art using Prolog

Jon-Paul Boyd

1st November 2017

Abstract Prolog is a declarative general-purpose logic programming language first conceived in the early 1970's by French computer scientist Alain Colmerauer. Its roots are in first order predicate calculus, and based on facts and rules. It has been widely used in artificial intelligence, especially within the interdisciplinary field of Natural Language Processing (NLP), providing applications with the ability to process human language due to its suitability in assembling human linguistics models for language search, determination and generation. This report presents a review of the state of the art using Prolog with emphasis on practical application within NLP over the last 5 years.

Keywords Prolog, natural language processing, applications of natural language processing

1. Introduction

Human language is the single greatest characteristic that differentiates human beings from other species. When we think we use language. When we organise ourselves, establish relationships, express beliefs, share knowledge and ideas, and communicate emotion we use language.

Natural Language Processing (NLP) looks to empower systems with the ability to analyse and understand the human language. By leveraging NLP, systems can help us organise our written word, derive meaning from our sentiments, engage in a meaningful conversation with us and perform actions upon our request.

In this decade of exponential data growth from web pages, documents, social media, entertainment and sensors, by humans and devices connected 24-7, in both the public and private enterprise domains, there is real opportunity for NLP to help us cope with, and make sense of, this flood of information.

The human language is easy for us to understand, our infants can meaningfully engage in discourse, and yet computer understanding of human language still presents many challenges. In this report, sections 2-6 highlight several recent examples of application of Prolog within an NLP discipline to solve some of these challenges. Section 7 briefly presents some recent Prolog innovation that is not NLP-specific, however they do support Prolog integration as an NLP component in common system architecture and software paradigms. Section 8 briefly considers today's popular AI programming languages. Section 9 provides a NLP classification of the application examples reviewed and closes with final thoughts.

2. Information Classification

According to WordPress statistics, 91.8 million new posts and 48.5 million new comments were made in the month of October 2017. Traditional techniques using word occurrence statistical analysis have been used to classify longer text documents, but for short texts like blogs, social media posts, tweets, news streams and online consumer feedback the word occurrence is too small to derive accurate classification.

A study on "*Classifying unlabelled short texts using a fuzzy declarative approach*" [1] proposes a classification method which could be used to mine customer sentiment on product feedback and understand political events and social movement in tweets. I find their solution particularly interesting as it doesn't require any prepared training sets, instead leveraging lexical databases like WordNet as a source of domain knowledge to enable classification. In essence the solution works by taking as input category name(s) from the user to derive a concept list of words from a lexicon or thesaurus like WordNet. Each word in the short text document along with words from the concept list are input as arguments to a proximity equation to estimate the relationship degree between the word pair, and hence an approximation of semantic closeness and therefore category classification can be made.

The solution was developed using Bousi~Prolog, an extension of the standard Prolog language, which provides for the required flexible search and proximity resolution via fuzzy logic. The authors tested their solution with sample short text datasets from news and web snippets (e.g. Reuters). Results were mixed according to the lexicon knowledge base/short text source pair. They note improved results with an ontology matched to the short text source. The classification results and proximity relation success rate is shown in the Appendix p.9.

I recognise the benefits in the generic framework of this solution that does not require timely and costly pre-prepared, supervised training sets. Given the nature of short text written expression, containing abbreviations such as NIMBY (Not In My Back Yard) and emoticons I would be interested to see how this noise can be transformed into useful information.

3. Information Extraction

In the study "*Accessing biological data as Prolog facts*" [2], the authors present a solution called Bio.db for working with biological data. Their solution models the interactions of proteins and genes as weighted graphs. Their solution provides an interesting feature in that they enable the user to easily switch between a variety of data sources, including Prolog quick load files and flavours of relational database (SQLite, RocksDB etc), so that the user can consume publicly available biological databases or their own self-compiled, self-service sources such as .csv files.

In Bio.db the native biological information is represented as Prolog facts, with the authors praising the suitability of Prolog in being able to readily express traditional relational database models, facilitating straight-forward data interrogation. String, a database of known and predicted protein-protein interactions is used in the study, with a focus on 8,548,002 interactions related to human proteins. With example queries such as returning all protein to protein interactions in response to gamma radiation, or a full gene population associated with gene ontology term GO:0046872 metal ion binding, the authors observe fast performance with their Prolog solution despite the data volume and complexity. See Appendix p.10 for example knowledge base query results and visualisation.

The authors argue that Prolog is a powerful language for building bioinformatics pipelines and that its role can be of crucial importance as biological data is increasingly needed to be viewed as knowledge both in the contexts of analysis and that of statistical inference and machine learning. *Prolog's knowledge representation credentials are highly relevant in this context. In addition, powerful interfaces to R such as Real allow Prolog to access the plethora of statistical libraries available in R. With the advances in modern Prolog systems in database integration and indexing technologies working with big data within Prolog is set to become an important application area for Prolog*. [2]

In this study we clearly see the selection of Prolog as the knowledge representation and query component in a modern application that includes R integration. Are we starting to see a resurgence of Prolog as a valid choice in a Big Data, medical and business analytics context?

The study "*Advances in integrative statistics for logic programming*" [3] continues the previous theme of integrating Prolog with R to deliver a hybrid software solution that combines the strengths of Prolog in knowledge representation and reasoning with the statistical inferencing, visualisation and vast open source, extensible features of R.

The well-established Real interface between Prolog and R enables R functions to be called using Prolog data. This study highlights more recent innovation with a new version called Real 2. This facilitates access to R code without any developer knowledge of R. Interaction with Real is via a single predicate ←/2, with the example below passing a Prolog list to R, applying the R Mean function to the Prolog data and getting the result back in a new Prolog variable.

```
? -a←[1, 2, 3], Mean←mean(a).

Mean = 2.0.
```

The example above clearly demonstrates how R can be simply and cleanly integrated into Prolog, making a strong case for this pairing solution into today's world of big data, business analytics, artificial intelligence, machine learning and bioinformatics. See Appendix p.11 - example knowledge base query, R function calling and visualisation.

In the work "*Learning and exploiting concept networks with ConNeKTion*" [4] a tool for concept network learning is introduced, to meet the challenge of automatically extracting useful information from document collections which continue to grow significantly. ConNeKTion takes a corpus of documents in natural language and builds a concept network, using nouns to indicate concepts and the network the interaction between the concepts. Properties are attributed to each concept from the associated verbs and adjectives. The network is formalised as a graph, with the nodes representing the concept nouns, and the edges the relationships expressing the grammatical structure of the text. The network graph can then be transformed into First-Order Logic (FOL) for query by Prolog.

I find several aspects of this solution compelling. First, the ConNeKTion solution was primarily designed to work without the need of supporting lexicons, to independently mine the document collection and solely use the content of the texts to build meaning. However, additional supporting taxonomy like WordNet or domain specific lexicons can be leveraged in an assisting capacity with supplemental enrichment during concept building where there is missing or poor-quality data. Second, with the use of algorithms they enrich the network with new nodes by building bridges between disjoint subgraphs, thereby establishing relationships between concepts that would otherwise be unknown due to lack of data or poor data quality. Third, with concept comparison and concept distance they can additionally enrich the network with generalisations. Fourth, keyword extraction is smarter, not based purely on single word occurrence alone but assessing each concept and its significance within the network. The tool is comprehensive with many features and an enlightening example of the power of representation of natural language in graph form to improve the understanding of and access to the underlying content, not only helping in making sense of what is there, but also what is missing. See Appendix p.12 for example of graphical representation of output and taxonomy building.

4. Computational Discourse

In the study "*Reasoning with Words: A First Approximation*" [5] the authors propose *a model of reasoning based on semantic relations among words and to incorporate it in the inference mechanism of a logic programming language.* The solution has two main features – the use of fuzzy logic with a weak unification resolution mechanism implemented by Bousi~Prolog, and the WordNet lexicon to provide a semantic relationship thesaurus.

For the first feature, weak unification, consider the example below and as provided by the authors on p.570.

```
% PROXIMITY EQUATIONS
physics ~math = 0.8.
physics ~chemistry = 0.8.
chemistry ~math = 0.6.

% FACTS
likes_teaching(john,physics).
likes_teaching(mary,chemistry).
has_degree(john,physics).
has_degree(mary,chemistry).

% RULES
can_teach(X,M):-has_degree(X, M),
likes_teaching(X, M).
```

If the above knowledge base was queried for who can teach maths with `?-can teach(X,math)`, the standard Prolog inference engine will derive false. However, just as we saw in the section 2 Information Classification example, the Bousi~Prolog implementation is used for its weak unification or proximity resolution capability, and derives X=john with 0.8 and X=mary with 0.6.

For the second feature regarding word semantic relationship, consider the following. *When two senses of two different words are identical, or nearly identical, we say the two senses are synonyms, for example couch/sofa, car/automobile. Antonyms by contrast are words with opposite meaning such as long/short, dark/light. One sense is a hyponym of another sense if the first sense is more specific, denoting a subclass of the other, for example car is a hyponym of vehicle, dog is a hyponym of animal and mango is a hyponym of fruit* [6] *(p.649-p650)*. These semantic relationships between words is provided by the WordNet thesaurus.

The combination of these two features deliver a reasoning mechanism that is more approximate and with a capability to assess by variation, and as an example could be integrated into a chatbot for more natural and more intelligent language discourse. See Appendix p.13 reasoning with synonymy and antonymy.

In the work "*An Emotion Understanding System*" [7] the authors remind us just how much information is conveyed in an emotion and how it influences human discourse. A conversation between two participants can evolve in different ways according to whether an emotion of happiness or sadness, joy or fear is exhibited or perceived. The perception of an emotion can help understand well-being, personality and mental state, willingness to cooperate or a reluctance to collaborate or move forward. The authors propose a solution for mining information from emotions to enable informed decision making and more natural user interaction by agents, based on perceived beliefs, motivations, standards, personality and cooperativeness.

In Prolog a knowledge base of emotions is defined as a set of rules which includes relations between the emotions and mental states. In discourse between a system agent and human user, the agent will perceive the user's emotional state by taking the user conversational input and querying the knowledge base to perceive emotions. This is a really interesting field of research, as what we a really talking about is cognitive empathy, and having this ability influences response in communicating with others. In the context of chatbots and conversation with humans, cognitive empathy can help chatbots understand if someone needs help, or there is a high likelihood of a positive purchasing decision, and negatively could be used to deceive.

The authors acknowledge there is still work to be done in both the consideration of intensity in emotions and refining the reasoning since they comment *Prolog gives answer "false" both to "really false" and "unknown", there should be a mechanism to distinguish between false conclusions and conclusions unable to judge*, which reminds me of the earlier examples where we've seen fuzzy logic for proximity resolution. See Appendix p.14 for example emotional model.

Now let's move on to argumentation in the context of computation discourse. *Argumentation, defined as "a verbal and social activity of reason aimed at increasing (or decreasing) the acceptability of a controversial standpoint for the listener or reader, by putting forward a constellation of propositions intended to justify (or refute) the standpoint before a rational judge*" [8]. *There are many uses and applications for automatic processing of the argumentative aspects of text, such as summarizing the argument of a complex court decision, helping a writer to structure an argument, and processing a large corpus of texts, such as blogs or consumer comments, to find arguments within it* [9].

In the work "*Ontology-based Argument Acquisition for Argumentative Agent*" [8] a method of argument acquisition based on domain ontology and Argument Ontology is proposed. A set of rules are derived from a domain ontology (DO) and expressed in OWL (W3C Web Ontology Language, a semantic Web language designed to represent rich and complex knowledge). A library such as Thea enables Prolog to parse the OWL ontology (authors don't indicate which library they use). The DO is then transformed into a Prolog knowledge base of facts and rules that form the initial argument set, and additionally help build counter-arguments. In a scenario where two agents participate in conversational discourse, they can understand and share arguments from the Prolog inference tree.

Argumentation is a very challenging intellectual and modelling problem, with arguments methods including the logical and the emotional and many arguments types such as the attack, counter, rebut and undercut, and. The discourse in which the arguments are being applied may be trivial or non-trivial, with different goals, such as defending or bargaining. An argument method and types may be more relevant in one scenario than another, with the argument approach in a court of law different to that within an online purchasing negotiation. See Appendix p.15 for example argument modelling.

The study "*A tool for Programming the behaviour of Intelligent Virtual Agents in Prolog*" [10] aims to evaluate Prolog as a means for defining intelligent virtual agent (IVA) behaviour. IVAs are defined as autonomous entities that can perceive their environment, interact with their environment and dynamically adjust their behaviour according to the environment state. These attributes facilitate enriching a virtual environment with improved realism and greater immersion due to more natural behaviours of actors within the world.

The authors extend the REVE platform used to build virtual worlds with Prolog as the component generating agent behaviour given its suitability for knowledge representation and intelligent reasoning. A continuous sense-decide-act cycle perceives the agent environment, decides what the agent should do next and act upon the next action, which could be simple, complex or random. The Prolog knowledge base can be dynamically updated with sensory data which may

influence further sense-decide-act cycles. The authors demonstrate that Prolog can very much be a relevant, valid and powerful component in building todays virtual reality (VR) environments. See Appendix p.16 for example architecture and behaviour predicates.

The study "*Towards Automatic Poetry Generation Using Constraint Handling Rules*" [11] covers the *implementation of an autonomous system capable of generating unique yet meaningful poetry using Constraint Handling Rules*. This avenue of research is particularly compelling as it looks to provision systems with the ability to express, innovate and create which comes naturally to humans. *One of the most expressive and creative uses of language is poetry, as it offers freedom, in terms of writing rules, and is highly based on human interpretation. A text needs to simultaneously satisfy three properties, to be considered a poem, namely poeticness (poetic structure), grammaticality (grammatical structure and syntax) and meaningfulness (semantic content).* [11]

The nature of poetry writing means the words and meaning come from within, from the soul and we don't look to replicate or borrow from other poetry. This is also the case in this solution which does not use existing poetry corpora. Only a custom lexicon, designed to simplify the reasoner with information suited to the implementation of Constraint Handling Rules (CHR) is used. This custom lexicon includes Part of Speech (POS) tags that identify whether a word is a noun, verb, conjunction etc, and facilitate ordering of the words in the poem.

A poem is incrementally generated, iteratively selecting only from possible words in the lexicon that satisfy constraints representing poeticness, grammaticality and meaningfulness. A rhythm matcher prunes the word list further, selecting words that rhyme for end of lines. CHR is used to ensure currently selected words are not the same as already chosen words during the iterative poem formulation cycle.

The authors evaluated the generated poems by posting them online with no indication they were of computer generated, for example http://allpoetry.com/poem/11642365 and http://allpoetry.com/poem/11642291, and received favourable feedback. Here is an example, titled Hope, which I believe demonstrates an acceptable outcome according to what makes a poem.

Friendship breathed or brightly trusted
Lovely marriage wedded and divorced
A wife touching after dearest lust
Firing over music and caress

5. Machine Translation

A study on "*An Adaptive Machine Translator for Multilingual Communication*" [12] explores the effectiveness of a machine translator using Lexical Functional Grammar (LFG) to model a generic, central framework with attributes (word sense, grammatical classification, gender etc) of given languages so that multilingual translation can transpire with only the linguistic attributes for each language needing to be defined centrally. If we assume that English and Spanish linguistic attributes are defined which enables English – Spanish translation, if we want to add English-French translation then only the linguistic attributes for French need to be additionally configured centrally.

There are two broad categories of MT systems: Example- Based Machine Translation (EBMT) and Statistical Machine Translation (SMT). The EBMT approach entails encoding a corpus of linguistic rules and data for the source and target languages supported by the MT system. At runtime, the source NL input is processed and analyzed against the encoded linguistic information, converted to equivalent representation in target language, and a translation is generated. In contrast the SMT approach trains the system on a large bilingual corpus of source-target pair probabilities, which is later consulted at runtime. SMT systems rely more on statistical pairing of words or phrases and less on grammatical or other types of linguistic data than EBMT systems. [12]

The authors opted for an EBMT-based solution to leverage the benefits of a lighter weight, descriptive model, with LFG attributes maintained by non-technical resources who did not have to be concerned with the inner technical workings of the system. This model choice also negated the need for a cured, prepared language translation corpus for each language pair. Prolog was used for the system implementation, matching well with LFG as a unification based grammar. As a software developer I can appreciate the generic framework, although I would have liked to have seen more detail on the outcome results of the solution, which the authors did not provide.

6. Speech Synthesis

Speech Synthesis is the artificial production of human speech. In the work "*Modelling Of Speech Synthesis Using Intelligent System Based On Prolog*" [13], the authors present an expert system for the Text To Speech (TTS) automatic reading of a text written in Standard Arabic, which unlike European languages is written right to left. Their aim is to develop Arabic text vocalising capability with proposed use cases in banking terminals and eLearning, to improve accessibility for the eyesight impaired and those with poor reading comprehension. These solutions enable delivery of services, comprehension and ultimately an improved quality of life.

The authors solution uses Prolog to model the domain specific lexicon knowledge base. This collection of phonetics covers the Arabic alphabet plus specific pronunciations (62 predicates modelling 28 consonants and 6 vowels). A transformation of the written graphemes to the phonemes and the chaining of the phonemes is made using a rule based system. It would have been useful if the authors elaborated further on how the grapheme to phoneme conversion was made. Specific detail on the architectural and integration components of their system, such as Prolog implementation and plugins to facilitate Prolog to Windows TTS, were missing.

Given the many varieties of Arabic dialect it would have been interesting to expand on observations from linguists on the final output of the system, and whether the speech was intelligible and natural. From a personal perspective I can understand how challenging the problem of converting text to intelligible, natural speech is – first introduced to TTS on a BBC Acorn Electron personal computer in 1994, my in-car navigation system from 2013 cannot pronounce French street names correctly when English spoken. There is still much work to be done.

7. Recent Related Innovations

No modern computer system consists of only one component. In the study "*Seamless cooperation of JAVA and PROLOG for rule based software development*" [14] proposes a connector architecture (CAPJA), simplifying the interfacing of JAVA with Prolog. This opens further possibilities for choosing the right tool for the job when architecting a system solution, for example the development of a rules based business decision framework, with the rule knowledge base represented and inferred within Prolog, and the connectivity, web integration and user interface managed by JAVA. Their solution allows the query of Prolog without the need for additional JAVA structures representing the Prolog terms.

In the study "*Logic Programming as a Service (LPaas) : Intelligence for the IoT*" [15] the authors explore the provisioning of Prolog as a cloud hosted AI component that can be leveraged to supply on-demand intelligence. This is a persuasive proposal – supplementing and enriching raw data from IoT devices, analysing and blending "at the scene" information with a Cloud-hosted knowledge base to monitor and act in a more intelligent, possibly bespoke and customised way. The authored solution interestingly can manage both dynamic and static knowledge bases, and stateful or stateless requests (e.g. going back to the next recursive iteration of a Prolog query response) depending upon requirement. They also define a LPaaS standard interface using standard communication protocols (e.g. REST over HTTP). *The Prolog engine is implemented on top of the tuProlog*. Is this another example of the possible resurgence of Prolog in choosing the right tool for the task within hybrid architectures for an AI use case scenario?

8. AI Programming Languages Today

During the research phase of this review it was particularly challenging to find journals specifically matching the demanded criteria of Prolog use in an NLP scenario that were published in the last 5 years. I don't have any dataset to substantiate the claim that today Python and Java are the most popular languages for A.I. development. However, in discussion with enterprise software vendor SAP regarding the development of a chatbot framework, I know the only languages they considered are Python and JAVA. Obviously the availability of experienced developers available to them with specific language experience weighs heavily in this consideration.

9. Conclusion

I believe the review clearly illustrates the strengths of Prolog in rapid prototyping of solutions to address complex NLP problems. What is encouraging to me personally is the recognition that Prolog is being included in modern hybrid systems architecture where a specific domain problem is best addressed with an existing logic programming inference engine. This makes me question whether large enterprise software vendors such as SAP who develop their own rules framework solutions with Java and SQL considered Prolog in the feasibility stage of software build. We have seen Prolog integrated with R for use within bioinformatics and I can see crossover into business analytics, and Prolog deployed to the cloud to provide distributed intelligence capability. We have also seen with several examples how Prolog can be used to model sophisticated understanding and dialogue, resulting in a more engaging and natural conversational experience for end users, and where this complexity requires a programmatic, context persistent approach (say over off-the-shelf SDK solutions from core.ai and api.ai).

There are still challenges to be overcome. There is no one single Prolog standard – in this review we have seen Bousi~Prolog and tuProlog used, making adoption more questionable. In my 25 years of software developer experience I've not met a single resource experienced in Prolog development, nor am I aware of any software supplier that can offer a Prolog develop service line to the largest of companies actively investing in A.I. solutions and who commonly offshore software development.

Table 1 Classification of reviewed applications

Application	Reference
Information Classification	[1]
Information Extraction	[2], [3], [4]
Computational Discourse	[5], [7], [8], [10], [11]
Machine Translation	[12]
Speech Synthesis	[13]

References

[1] F. P. Romero, P. Julian-Iranzo, A. Soto, M. Ferreira-Satler and J. Gallardo-Casero, "Classifying unlabeled short texts using a fuzzy declarative approach," *Language Resources and Evaluation,* vol. 47, pp. 151-178, 2012.

[2] N. Angelopoulos and J. Wielemaker, "Accessing biological data as Prolog facts," in *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming*, Namur, 2017.

[3] N. Angelopoulos, S. Abdallah and G. Giamas, "Advances in integrative statistics for logic programming," *International Journal of Approximate Reasoning,* vol. 78, pp. 103-115, 2016.

[4] F. Rotella, F. Leuzzi and S. Ferilli, "Learning and exploiting concept networks with ConNeKTion," *Applied Intelligence,* vol. 42, no. 1, pp. 87-111, 2014.

[5] C. Rubio-Manzano and P. Julian-Iranzo, "Reasoning with Words: A First Approximation," in *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE),* Beijing, 2014.

[6] D. Jurafsky and J. H. Martin, Speech and Language Processing, Pearson Education Inc., 2009.

[7] R. Qiao, H. He, J. Gu and M. Liu, "An Emotion Understanding System," in *2014 2nd International Conference on Systems and Informatics (ICSAI 2014)* , Shanghai, 2014.

[8] B. Liu, L. Yao and F. Liu, "Ontology-based Argument Acquisition for Argumentative Agent," in *2017 3rd International Conference on Information Management*, Cape Town, 2017.

[9] E. Cabrio, G. Hirst, S. Villata and A. Wyner, "Natural Language Argumentation: Mining, Processing, and Reasoning over Textual Arguments," in *Dagstuhl Seminar 16161*, Dagstuhl, 2016.

[10] G. Anastassakis and T. Panayiotopoulos, "A Tool for Programming the Behaviour of Intelligent Virtual Agents in Prolog," in *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, Corfu, 2015.

[11] A. e. Bolock and S. Abdennadher, "Towards automatic poetry generation using constraint handling rules," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, Salamanca, 2015.

[12] R. Lane and A. Bansal, "An Adaptive Machine Translator for Multilingual Communication," in *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Poznan, 2017.

[13] T. Hanane, H. Maamar and A. Hamid, "Modeling Of Speech Synthesis Using Intelligent System Based On Prolog," *Journal of Theoretical and Applied Information Technology,* vol. 76, no. 3, pp. 350-357, 2015.

[14] L. Ostermayer, "Seamless Cooperation of JAVA and PROLOG for Rule-Based Software Development," in *Doctorial Consortium@The 9th International Web Rule Symposium (RuleML)*, Berlin, 2015.

[15] R. Calegari, E. Denti, S. Marianit and A. Omicini, "Logic Programming as a Service (LPaaS): Intelligence for the IoT," in *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, Calabria, 2017.

Appendix

Information Classification - Classifying unlabelled short texts using a fuzzy declarative approach - classification results and proximity relation success rate

Table 3 Experimental results

Proximity relation	C (%)	W (%)	U (%)	P (%)	R (%)	F (%)
(A) News snippets experiment results						
Structural analogy	25	34	41	32	23	27
Contextual neighborhood	61	18	21	60	21	31
Wikipedia/WUP	64	23	13	70	60	65
Wordnet/WUP	60	25	15	67	57	62
BaseLine	22	6	72	76	20	31
(B) Web snippets experiment results						
Structural analogy	43	41	16	28	43	34
Contextual neighborhood	46	49	5	31	46	37
Wordnet/WUP	57	42	1	56	56	56
Wikipedia/WUP	72	27	1	73	72	75
BaseLine	17	13	70	54	17	25
(C) Reuters SHORTS experiment results						
Structural analogy	30	4	65	71	29	41
Contextual neighborhood	60	2	38	19	54	28
WordNet/WUP	85	7	8	91	76	83
Wikipedia/WUP	92	8	0	54	83	67
BaseLine	18	1	81	96	16	27
(D) Reuters-10 experiment results						
Structural analogy	17	25	58	22	16	18
Contextual neighborhood	55	29	16	15	49	23
WordNet/WUP	82	14	4	83	74	78
Wikipedia/WUP	78	22	0	78	70	74
YAGO/WUP	85	13	2	85	76	80
WordNet/synonymy	39	50	11	23	35	28
BaseLine	16	7	75	65	17	28

Classifying unlabelled short texts, page 174. Classification results are shown in Table 3 where we display the percentages of correct classifications (C), wrong classifications (W), unclassified documents (U), precision (P), recall (R), and their F measure (F) w.r.t. a set of categories in different experiments.

Information Extraction - Accessing biological data as Prolog facts - example knowledge base query results and visualisation

GO term	GO name	Population
GO:0000139	Golgi membrane	596
GO:0003674	molecular_function	751
GO:0004674	protein serine/threonine kinase activity	352
GO:0005524	ATP binding	1497
GO:0005575	cellular_component	489
GO:0006468	protein phosphorylation	456
GO:0010923	negative regulation of phosphatase activity	53
GO:0016021	integral component of membrane	3732
GO:0030424	axon	237
GO:0030425	dendrite	318
GO:0046872	metal ion binding	1988

Table 2: Gene ontology terms and associated GO term names for LMTK3. The third column shows the total of genes in the GO term.

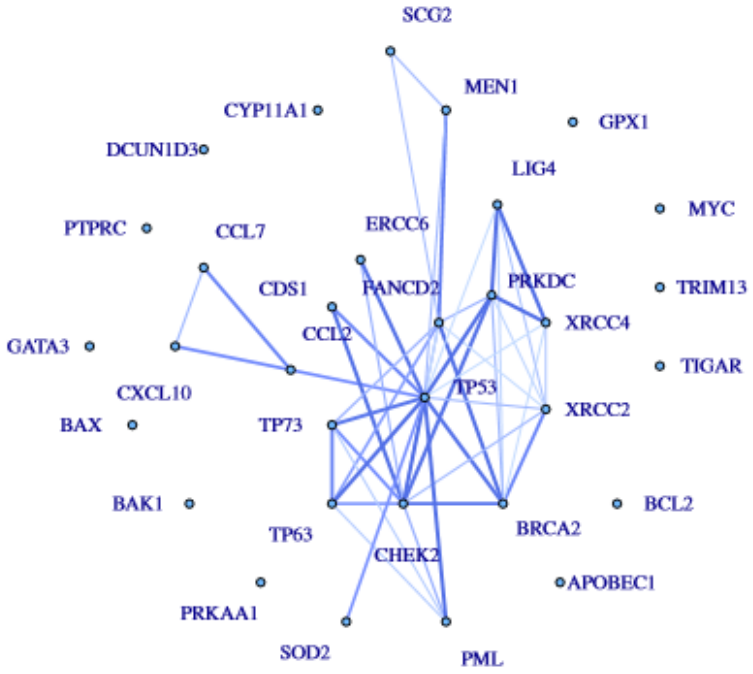

Figure 4: Gene ontology term: response to gamma radiation, GO:0010332. Edges are provided by the String database. The width and darkness of edge colour signify higher belief in the interaction being a real protein-protein interaction.

```
lmtk3_go :-
    map_gont_symb_gont('LMTK3',Gont),
    findall(Symb,map_gont_gont_symb(Gont,Symb),Symbs),
    map_gont_gont_gonm(Gont,Gonm),
    sort(Symbs,Oymbs),
    length(Oymbs,Len),
    fail.
```

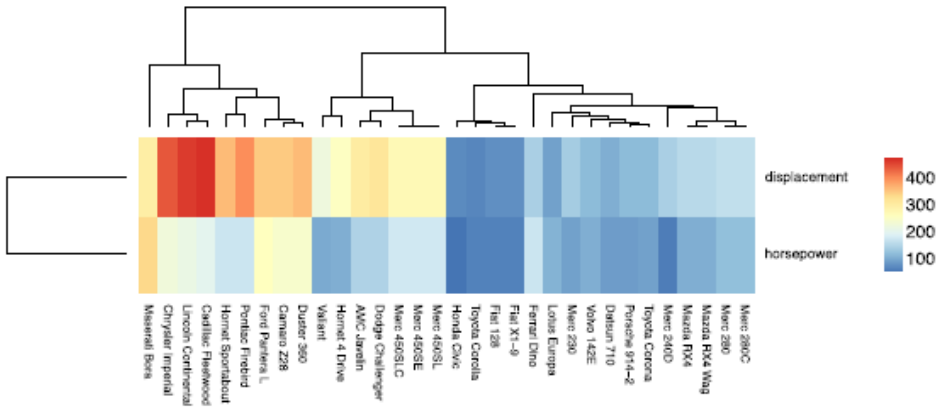

Fig. 5. Heatmap generation with aheatmap() from package NMF.

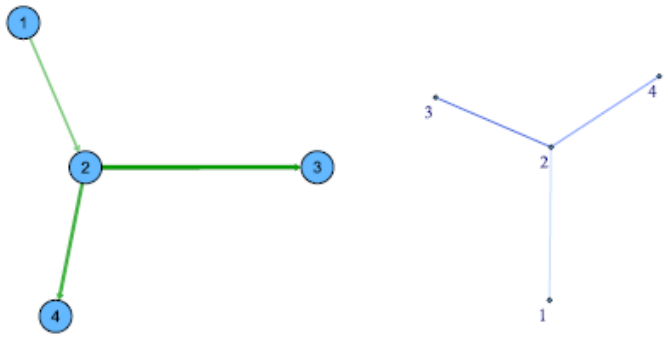

Fig. 6. Graphs generated by *wgraph_plot*/2. Left: plot uses default rendering with *qgraph()* call. Right: render changed to *igraph()* and a number of options specialised the output.

Heatmap drawing functions are ubiquitous in *R*. *b_real* provides an interface to the *aheatmap* function. In addition to some simple option mapping *aheatmap*/2 provides polymorphic support for the first argument which could be a matrix *R* variable or a Prolog representation of one. The following code uses the *mtcars* example dataset, from which it plots a heatmap of two variables: *hp* (horsepower) and *disp* (displacement) (Fig. 5).

```
?− MtC ← as.list(mtcars), memberchk(hp=HP,MtC),
   memberchk(disp=Disp,MtC), x ← [HP,Disp],
   rownames(x) ← c("horsepower","displacement"),
   ← aheatmap(x).
```

3.2.2. Weighted graphs

R has a number of plotting functions for drawing graphs formed of nodes and edges. Two of these are `igraph()` and `qgraph()`. The latter being based on the former with some extra options and facilities for grouping nodes. The Prolog pack *wgraph* provides a uniform Prolog interface to these two *R* libraries. A plot with the default renderings can be easily drawn from a list representing the graph connections and the weights on the edges:

```
?− G =[1−2:200, 2−3:400, 2−4:300],
   wgraph_plot(G,[ ]).
```

A set of Prolog options that control the choice of the drawing function and basic parameters of the graph, and which work irrespective of the drawing function can be provided in the second argument of *wgraph_plot*/2. In the following example `igraph()` is passed the size of nodes to use, the degree at which the node labels should be displayed and the distance of the label from the node edge. The resulting graph is shown in the RHS of Fig. 6.

```
?− G =[1−2:200, 2−3:400, 2−4:300],
   Opts =[plotter(igraph),
          label_distance(−1),
        label_degree(2),
```

Information Extraction - Learning and exploiting concept networks with ConNeKTion – Graphical representation of output and taxonomy building

expert system, that produces the following relationships between concepts:

attribute(C, A) attribute A describes concept C;
can(C, A) concept C may perform action A;
be(C, A) concept C may receive (be a passive subject of) action A;
is_a(C_1, C_2) concept C_1 is a subclass of concept C_2;
relationship(C_1, C_2) *relationship* holds between concepts C_1 and C_2.

where actions are expressed by verbs, and sentences involving verb 'to be' are used to obtain an initial sub-class structure for the taxonomy: e.g., "penguins are birds" yields *is_a*(*penguin*,*bird*). The expert system is a peculiarity of ConNeKTion, that provides additional high-level features about the textual content with respect to other approaches available in the literature.

A representational trick is adopted to treat indirect complements as direct ones, by embedding the corresponding preposition into the verb: e.g., "The cat jumped on the table" becomes *jump_on*(*cat*,*table*). Each arc in the graph is labeled with the frequency with which the associated relationship was found in the training corpus. This improves robustness, allowing to filter out all elements that do not pass a given level of reliability/frequency, and lays the basis for applying statistical reasoning on the graph. More precisely, the presence of *negation modifiers* for verbs in the syntactic tree of a sentence is used to track separately positive and negative forms: depending on the balance of frequency for these two forms, the system can infer whether a given relationship is occasional, typical, mandatory, prohibited, etc. for some concepts.

This yields the basic graph representation on which ConNeKTion works. Figure 1 shows the graph corresponding to the sentence "*Bell, based in Chicago, makes and distributes electronic, computer and building products*". Note that this part is completely incremental: any time new texts are available, they can be processed separately and integrated into the existing graph, updating the frequencies of existing nodes/edges and possibly adding new nodes and/or edges. It should also be noted that the learned graph, or portions thereof, can be directly mapped onto a First-Order Logic (FOL) description, that enables an application of several kinds of reasoning strategies on it. In particular, the neighboring subgraph of a node can be interpreted as a formal definition of the concept associated to that node: the wider the selected neighborhood, the more detailed the concept definition.

2.2 Taxonomy building

The taxonomic part of a concept network includes the class-subclass relationships according to which the set of available concepts is structured. The basic relation underlying taxonomies is *generalization*, for which we adopt a simple and operational definition:

Definition 1 (Generalization) [29] A concept G generalizes another concept C if anything that can be labeled as C can also be labeled as G, but not *vice-versa*.

Thus, generalization is fundamental, and provides many opportunities of enrichment and/or manipulation of the basic graph:

1. building taxonomic structures by (repeatedly) applying a generalization operator to identify existing, or create new, generalized concepts;
2. shifting the representation, by removing the generalized nodes from the graph and leaving just their generalization (that inherits all their relationships to other concepts);
3. extending the number of relationships between concepts belonging to the same connected component of the graph, or adding connections between disjoint components that open new (previously impossible) paths.

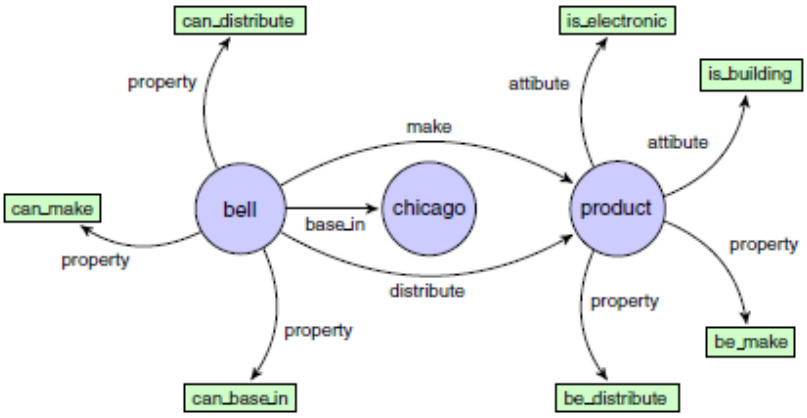

Fig. 1 Graphical representation of the output

quantify the relation between symbols. On the other hand, when symbols are interpreted as words it will be possible to model the semantic relations between them making use of a thesaurus $\mathcal{T}$. A word $w_1 \in \mathcal{T}$ is related to a word $w_2 \in \mathcal{T}$ with a degree α through a semantic relation (synonymy, antonymy or hypernymy) which is obtained from $\mathcal{T}$, what it is represented in Bousi∼Prolog by using a proximity equation "$w_1 \sim w_2 = \alpha$". So, words and their relations become part of the first order language alphabet and take part in the inference process naturally. This requires an inference rule based on the semantic relation that has been used to obtain the set of proximity equations between words. More formally, we will call *program*, Π, a set of first order horn clauses in which there is a set of symbols that we interpret as words. This program has an associated dictionary that we will call vocabulary of Π and that are composed of all those symbols (predicate, function or constant) present in the program and that are defined in a thesaurus $\mathcal{T}$.

Definition 1 (Vocabulary of a program Π*):* Given a program Π. The vocabulary of Π, denoted by $\mathcal{V}_\Pi$, is made of all the predicate, function or constant symbols in the program.

Definition 2 (Vocabulary of Π *and* $\mathcal{T}$*):* Given a program Π and a thesaurus $\mathcal{T}$. The vocabulary of Π and $\mathcal{T}$, denoted by $\mathcal{V}_\Pi^\mathcal{T}$, is made up of all the predicate, function or constant symbols in the program Π that are words in $\mathcal{T}$. That is, symbols which are interpreted as words in a thesaurus $\mathcal{T}$.

Example 3: Let Π be the program of Example 1 and a thesaurus $\mathcal{T}$ of the English language as WordNet. The vocabulary $\mathcal{V}_\Pi^\mathcal{T} = \{at, car, home, airport, driving\}$.

Now, each of the elements in $\mathcal{V}_\Pi^\mathcal{T}$, will be used by the thesaurus $\mathcal{T}$ in order to obtain the semantic relations: synonyms, antonyms and hypernyms. To establish the reasoning model, we have to define with precision the inference rules based on the semantic relations among words.

A. Reasoning with Synonymy

Reasoning with synonymy consists of using this semantic relation to infer. It is required to obtain inference rules based on this property. The deductive calculus based on synonymy has two rules. A direct inference rule that established that if I have A then I can infer B as long as A and B are synonyms; and an adaptation of the modus ponens rule that establishes that if I have A and B $\rightarrow$ C then I can infer C as long as A and B are synonyms. This is formally stated in the following definitions:

Definition 3 (Synonymy-based Direct Inference Rule): The synonymy-based direct inference rule states that from A it can be inferred as immediate consequence B, if A is a synonym of B.

Definition 4 (Synonymy-based Modus Ponens): The Modus ponens rule based on Synonymy states that from A and $B \rightarrow C$ it can be inferred as immediate consequence C if A is a synonym of B.

In order to simulate this reasoning, in our framework we have to compute the synonyms for each of the elements of the vocabulary of a program Π. Given a program Π, a thesaurus $\mathcal{T}$ and its associated vocabulary $cV_\Pi^\mathcal{T}$, then for all $w \in \mathcal{V}_\Pi^\mathcal{T}$, the set of synonyms of w extracted from $\mathcal{T}$ is denoted as $\mathcal{S}(w)_{\mathcal{V}_\Pi^\mathcal{T}} = \{s_1, \ldots, s_n\}$.

Example 4: Given the vocabulary of Example 3, for the thesaurus WordNet, it is possible to obtain a set of synonyms for each element of the vocabulary: $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(at) = \{helium, \ldots\}$; $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(car) = \{sedan, bike, motorcar, \ldots\}$; $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(home) = \{home\}$; $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(airport) = \{airdrome, heliport, aerodrome, \ldots\}$; $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(driving) = \{dynamic, impulsive, energetic, \ldots\}$.

Once the vocabulary of synonyms has been constructed, the proximity equations are created (synonym equations in this case) from the vocabulary of the program and its set of associated synonyms.

Definition 5 (Synonymy equations for Π*):* Given a program Π, a thesaurus $\mathcal{T}$ and its associated vocabulary $\mathcal{V}_\Pi^\mathcal{T}$. Then for all $w \in \mathcal{V}_\Pi^\mathcal{T}$ and its synonyms $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(w) = \{s_1, \ldots, s_n\}$, an entry $w \sim s_i = \alpha_i$ is created for each synonym s_i, with approximation degree α_i.

Example 5: Given a program $\Pi = \{loves(a, b)\}$, the thesaurus $\mathcal{T} = WordNet$ and the vocabulary of the program and the thesaurus $\mathcal{V}_\Pi^\mathcal{T} = \{loves\}$, we obtain the synonyms of the vocabulary, that is, $\mathcal{S}_{\mathcal{V}_\Pi^\mathcal{T}}(loves) = \{dotes\_on\}$. Then the synonym equation "$loves \sim dotes\_on = 1.0$" is created. Therefore, it could be asked if `a` dotes on `b`, launching the query "`?- dotes_on(a,b)`". Then, the system would respond "`Yes`", since `a` loves `b` and, by deduction based on synonymy, it also dotes on `b`.

B. Reasoning with Antonymy

Reasoning with antonymy consists of using this relation to obtain a type of reasoning based on this semantic relation. Following [11], antonymy is a phenomenon of the natural language which involves a pair of words (P, Q) with some important features [1]: (a) Having the pair (P, Q) then P is an antonym for Q and Q is an antonym for P, i.e. $P = aQ$ and $Q = aP$; (b) Every object x fulfills that "If x is aP then it is not the case that x is P", but it does not fulfill that "If x is not P then x is aP". For example, "the bottle is empty" means that it is not full, but "the bottle is not full" does not mean that it is empty, i.e. aP implies not P, but not P does not imply aP (with some exceptions).

We are going to handle antonymy as synonymy of opposite concepts. Consequently, the Synonymy-based Inference Rules are used but taking into account, in this case, that if P is an antonym of Q we have that P is a synonym of aQ, i.e. Q is the opposite of P. On the other hand, and from a syntactical point of view, it will be necessary to use a modifier of antonymy "not#" to provide information regarding the concepts that are antonyms. So, for example, if "cold" is antonym of "hot" then "cold" is synonym of

[1]For simplicity we only enunciate two of them, the necessary ones in order to define the inference rules based on this semantic relation, see [11] for more information.

Fig. 1. Emotion understanding system

evaluator to compare the perception P with the meta-model M.

The necessary and sufficient conditions of system S can understand emotions Es follows the general understanding mentioned above, which are: 1) S can access a meta-model (knowledge) M of emotions Es. 2) S can recognize the user's current emotion E to generate P. (The Emotion E may be expressed through text, expression, gesture, etc.. P is a perception of E by S with respect to M.) 3) S can deduce the user's information from the perception P by the knowledge M about Es.

Our Emotion understanding system shown in Fig. 1 consists of three components: meta-model, analyzer, and evaluator, which are under the same names as [5]. The meta-model M contains rule-like knowledge about emotions. The knowledge includes the relations between emotions and mental states, personalities, etc. The analyzer recognizes E to form a perception P of E. The evaluator interprets P with respect to the meta-model M, resulting an evaluation of the agent's beliefs, desires, standards and personalities, events happened, actions done by related agents, and cooperativeness. This work does not consider how the analyzer recognizes emotions, but focuses on the meta-model and the evaluator. The knowledge in the meta-model for recognition of emotions is also left out of account.

This system in our work functions differently from the system in article [5]. The latter is simulated in a multi-agent system in which agents can learn associations between actions and emotions of other agents to elicit target emotions in other agents. Their work is related to the component (VI) in the introduction. Our system however is used to excavate various information of other agents from their emotions, which involves components (II),(III),(IV),(IX). The system in the article [5] demonstrates relationship between emotions and actions by semantic graph. Our system adopts rules to convey the relationship between emotions and mental states, personalities, etc.

III. Meta-model about Emotion

The meta-model about emotions is a list of rules. Each rule adopts the form "*emotion* → *something*", which denotes "From the emotion, something can be inferred." These rules are classified into three categories: emotion to appraisal, emotion to personality, emotion to cooperativeness.

A. Emotion to Appraisal

OCC (Ortony, Clore, Collins) model [11] sorts emotions into event-based emotions, agent-based emotions and object-based emotions. This subsection considers the former two sorts of emotions. The symbols such as *Pre*, *Dis*, *Goal*, *AntiGoal* etc come from the paper [12], which added intensity to emotion formalization [13]. However, this work does not consider the intensity variables of emotions in the paper [12]. The new denotation $Done(\pi)$ represents execution of the action π.

1) Event-based Emotion: When an agent's emotion depends on some event, this event's consequence as desirable or undesirable with respect to its' goals can be reversely appraised. Event-based emotions are classified into well-being emotions, prospect-based emotions, fortunes of others emotions [13], [12].

Well-being Emotions If the agent u feels joy(distress), then he is fully convinced an event ψ as the target(anti-target).
$Joy_u\psi \rightarrow Bel_u\psi$
$Joy_u\psi \rightarrow Goal_u\psi$
$Distress_u\psi \rightarrow Bel_u\psi$
$Distress_u\psi \rightarrow AntiGoal_u\psi$

Prospect-based Emotions If the agent u feels hope (fear) then he believes an event ψ as the target(anti-target) to a certain extent. *Belh* means "don't entirely convince".
$Hope_u\psi \rightarrow Belh_u\psi$
$Hope_u\psi \rightarrow Goal_u\psi$
$Fear_u\psi \rightarrow Belh_u\psi$
$Fear_u\psi \rightarrow AntiGoal_u\psi$

If the agent u feels satisfaction(fearconfirmed), then he is confirmed about the prospect of an event ψ as target(anti-target).
$Satisfaction_u\psi \rightarrow Bel_uPBelh_u\psi$
$Satisfaction_u\psi \rightarrow Goal_u\psi$
$Satisfaction_u\psi \rightarrow Bel_u\psi$
$FearConfirmed_u\psi \rightarrow Bel_uPBelh_u\psi$
$FearConfirmed_u\psi \rightarrow AntiGoal_u\psi$
$FearConfirmed_u\psi \rightarrow Bel_u\psi$

If the agent u feels relief(disappointment), then he is disconfirmed about the prospect of an event $\neg\psi$, where ψ is target(anti-target). *Belq* means " believe a little at least".
$Relief_u\psi \rightarrow Bel_uPBelq_u\neg\psi$
$Relief_u\psi \rightarrow Goal_u\psi$

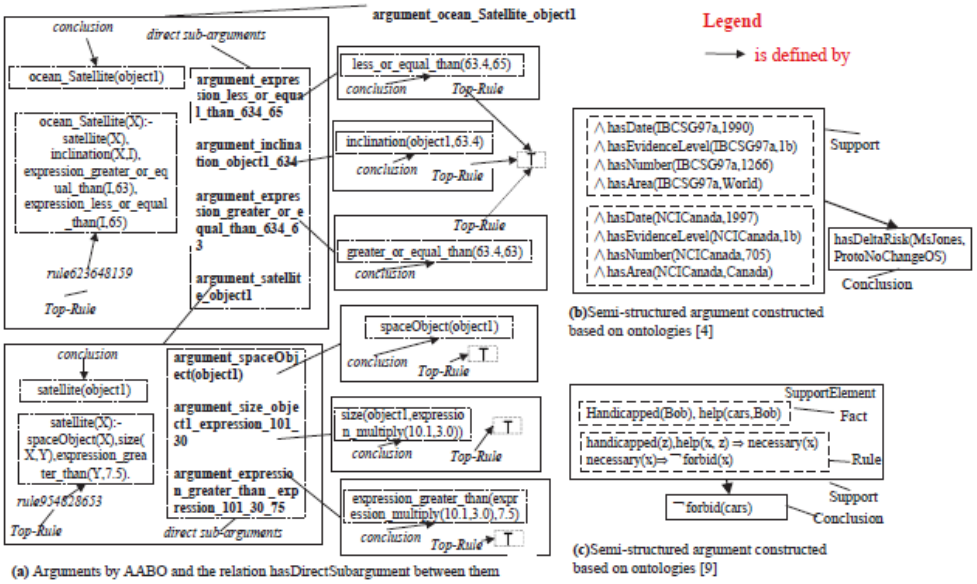

Figure 4. Arguments constructed based on ontologies.

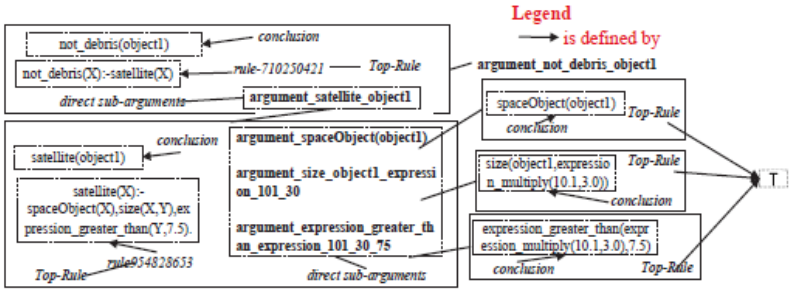

Figure 5. Argument containing classical negation constructed by AABO.

Although the attack relations between the arguments in DAT nodes are not specified in Figure. 3, their specific type can be deduced. The first 2 childs of the root node indicate that agent2 uses *argument_not_satellite_object1* (*arg13*) and *argument_not_ocean_Satellite_object1* (*arg19*) to attack *arg3* respectively. The conclusions of *arg3*, *arg13* and *arg19* indicate that *arg13* and *arg19* undercuts and rebuts *arg3* respectively. Similarly, other attack relations can be inferred. The attack arguments are all counter-arguments. So, the two agents are antagonistic. They try to make the opponent's arguments defeated and claim rejected using counter-arguments, which are built dynamically in the dialog. Unlike the counter-argument acquisition by detecting attacks [4, 5], the counter-arguments of the given arguments are built directly. The third child of the root node indicates that agent2 has no argument with the action "concede". This means that agent2 concedes the argument in the root node, i.e., *arg3* proposed by agent3 cannot be attacked by agent2 any more. So, agent3 wins, i.e., "ocean_Satellite(object1)" holds. So, *object1 is ocean satellite* is currently true.

AO is instantiated by the above arguments and their relatives constructed by agent3. Part of the instantiated AO expressed in Manchester Syntax is shown in Table II.

Inference information is in the arguments' components. E.g., *arg3* has the top rule *rule623648159*, four direct subarguments, and the conclusion *ocean_Satellite_object1*. *rule623648159* is defined by the following Prolog clause:

ocean_Satellite(X) :- satellite(X), inclination(X,I), expression_greater_or_equal_than(I,63), expression_less_or_equal_than(I,65).

Apart from enabling agents construct arguments autonomously in dialogs, AABO has two advantages over existing argument acquisition by deductive reasoning.

1) Facility of recording and reusing inference information of arguments

Figure. 3. indicates that the selected argument is constructed step-by-step by chaining the inference into a tree. The fully structured arguments reveal their inferences by the relationship *argument-subargument*. Figure. 4.(a) shows the *argument-subargument* relationships between an argument

TABLE I
PROLOG PREDICATES FOR THE DEFINITION OF AGENT BEHAVIOUR

Predicate, syntax	Semantics
initial_kb/1 *initial_kb*(*B*)	Unifies the agent's knowledge base with B.
sense/3 *sense*(*S*, *I*, *O*)	Unifies the agent's knowledge base with O after processing most recent sensory data S and current knowledge base I.
decide/2 *decide*(*I*, *O*)	Unifies the agent's knowledge base with O after committing to a course of action based on current knowledge base I.
act/3 *act*(*A*, *I*, *O*)	Unifies A with next action to be executed and agent's knowledge base with O after processing committed-to course of action in current knowledge base I.

virtual worlds as collections of virtual objects with physical properties, semantics and functionality [9], and (c) a model of perception and action for virtual agents [10].

In such an IVE, a typical application lifecycle would be as follows: First, the user starts the REVE Worlds system and creates a new virtual world by loading a VERL file through the system's user interface. Then, they start as many external behaviour-generation applications as required and connect them to REVE Worlds. After a connection with a behaviour-generation application is established, REVE Worlds creates a virtual body and yields control of it to the application, hence creating what is essentially a complete virtual agent. After they have been created, virtual agents repeatedly perceive the virtual world and act upon it as dictated by their behaviour-generation components according to data exchanged via the AIP protocol.

III. PROGRAMMING AGENT BEHAVIOUR IN PROLOG

A. Architecture

The presented extension to the REVE platform consists of a behaviour-generation component, referred to as *client* in the rest of this section, whose structure is shown in Figure 2.

As shown in Figure 2, the client consists of a communications layer handling AIP data exchange over TCP/IP, a control layer and a Prolog engine managing (i) the *agent KB* and (ii) four special-purpose predicates, referred as as *behaviour predicates*, which, along with additional developer code, constitute the Prolog engine's KB.

B. Behaviour predicates

The four behaviour predicates enable the specification of agent behaviour by the developer. As discussed in Section III-C, the first is used to initialize the agent KB while the remaining three drive the generation of the agent's behaviour by being continuously solved in a *sense-decide-act* cycle. They are listed in Table I.

As shown in Table I, the purpose of the *initial_kb/1* predicate is to initialize the agent KB. The predicate's semantics require the single argument *B* to be an unbound variable that shall be unified with the initial state of the agent KB whenever the Prolog engine attempts to solve a *initial_kb(B)* query. The

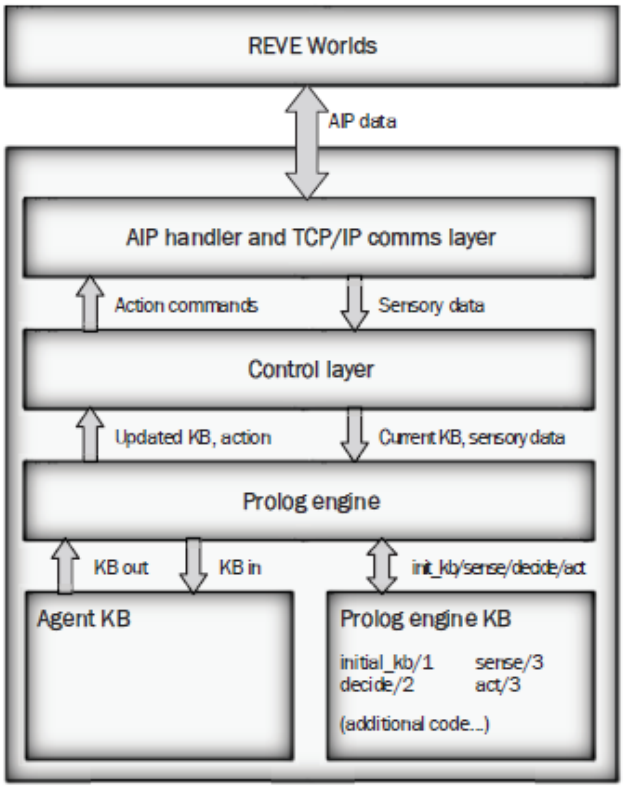

Fig. 2. Client architecture

predicate's implementation as well as the structure and initial contents of the agent KB are entirely up to the developer - no restriction whatsoever is imposed either by the REVE platform or the predicate's semantics: for instance, it can be a flat list of beliefs, a structured list reflecting a BDI KB, or else.

The purpose of the *sense/3* predicate is to process sensory data. The predicate's semantics require the first argument *S* to be the sensory data to be processed, the second argument *I* to be the current agent KB and the third argument *O* to be an unbound variable. Based on the implementation provided by the developer, whenever the Prolog engine attempts to solve a *sense(S, I, O)* query the predicate shall process sensory data and the current agent KB and shall unify the third argument with an updated agent KB reflecting the sensory data processing that took place. The structure and contents of the current and updated agent KB shall comply with the *initial_KB/1* predicate's semantics. Also, eventhough no such restriction is imposed by the predicate's semantics, the client's design specifies that the first argument shall be a list of symbolic sensory data according to the REVE representation's perception model (further discussed in Subsection III-D and by Anastassakis and Panayiotopoulos (2014) [10]). However, the design of the exact sensory data-processing mechanism and its outcome are entirely up to the developer: the sensory data can be added to the knowledge base, ignored or additionally