Etalog – a natural-looking knowledge representation formalism[*,**]

Ivan Rygaev[1]

[1] Laboratory of Computational Linguistics,
A. A. Kharkevich Institute for Information Transmission Problems,
Russian Academy of Sciences, Moscow, Russia
irygaev@gmail.com

Abstract. In this paper we present Etalog – a formal knowledge representation language developed in the Laboratory of Computational Linguistics IITP RAS in scope of the semantic text analyzer SemETAP. Borrowing ideas from existing formalisms we created a formal language which is a) more expressive than Description Logic, b) more human-readable than Datalog and Turtle, c) allows capturing not only truth-conditional but also (a part of) communicative structure of the sentence.

Keywords: Semantics, Natural language understanding, Knowledge representation, Inference.

1 Introduction

The semantic text analyzer SemETAP, under development in the Laboratory of Computational Linguistics IITP RAS, is aiming at performing deep semantic analysis of natural language texts. Based on various linguistic and extra-linguistic resources (a combinatorial dictionary, an ontology, a fact base and an inference engine), SemETAP is able to build a semantic structure of the text and extend it with implicit knowledge using inference rules (Boguslavsky et al 2015, 2018).

Semantic structure is based on the ontology and is built in two steps. First a syntactic (dependency) tree of the sentence is produced, and then (roughly) words are replaced with semantic concepts from the ontology and syntactic relations – with semantic ones, which results in a set of binary predicates (triples) of the form `relation(node1,node2)` or a semantic graph.

Below is an example of a sentence, its syntactic tree, semantic graph and the corresponding set of triples:

---

[*] This paper presents the results of a joint effort of the team of the Laboratory of Computational Linguistics IITP RAS. The team includes I. Boguslavsky, L. Iomdin, A. Lazursky, S. Timoshenko, T. Frolova, V. Dikonov, E. Inshakova, V. Sizov and others. I would like to thank my colleagues for their wonderful collaboration.

[**] This work was supported by the RSF grant 16-18-10422, which is gratefully acknowledged.

(1) John sold an umbrella to Peter

Fig. 1. Syntactic tree of sentence (1)

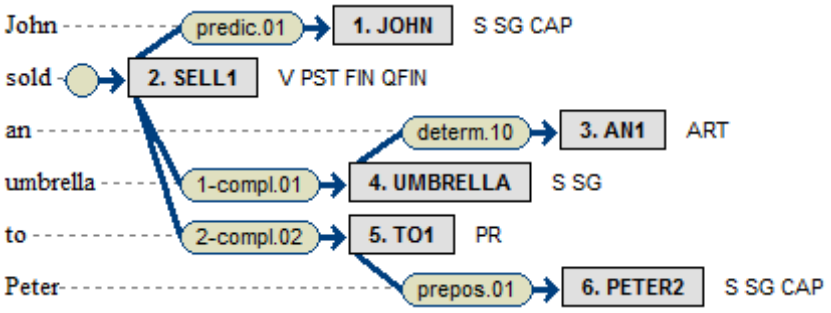
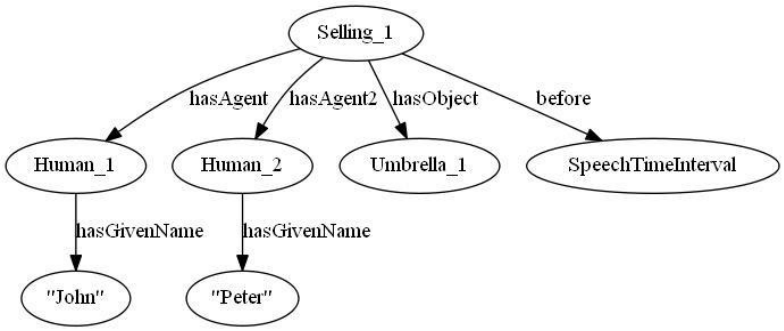


Fig. 2. (Simplified) semantic graph of sentence (1)




```
(2) hasGivenName (Human_1, "John")
    hasAgent (Selling_1, Human_1)
    hasAgent2 (Selling_1, Human_2)
    hasObject (Selling_1, Umbrella_1)
    hasGivenName (Human_2, "Peter")
    before (Selling_1, SpeechTimeInterval)
```

In the semantic structure, each node represents an individual of a certain class (semantic concept). There are implicit triples that assign a class to each individual, such as `isA(Human_1,Human)` or `Human(Human_1)`, but since the class name is encoded in the individual name these triples are omitted for the sake of simplicity.

A sale event could be represented as an n-place predicate: `Selling(Human_1, Umbrella_1, Human_2)`, but its reduction to binary predicates as suggested by neo-Davidsonian semantics (Parsons 1990, Higginbotham 2000), has a number of advantages: a) the event individual (`Selling_1`) allows attaching time, location and other adjuncts to the event; b) unexpressed arguments (such as selling price) can be safely ignored; c) the semantic structure resembles the syntactic one in a more straightforward way; d) it makes the semantic structure compatible with Semantic Web standards – RDF and OWL.

On the other hand a big list of unsorted triples is hard to read or type manually (when writing inference rules). Etalog was created to solve this issue. While maintaining the underlying structure as a set of triples it allows presenting it in such a way that simplifies reading and typing. Etalog is used for two purposes:

1. to write inference rules;
2. to represent the semantic structure of the sentence.

2 Language Features

Etalog emerged as a language for inference rules, most of which in our system are concept decomposition rules. Concept decompositions are similar to word definitions in the explanatory dictionary, but are written in formal language. They are created manually by linguists.

Originally both the premises and the conclusion of the rules were written as a list of triples. In our rules one variable usually participates in many relations, so the variable name had to be typed multiple times. A small typo in the variable name could lead to hard-to-identify logical errors. In the example below variables `?selling` and `?buying` are repeated 7 times each, other variables – 3 times each.

```
(3) Rule Selling: // Sale
     Selling (?selling) ->
     hasAgent(?selling,?seller), Agent(?seller), //seller
     hasAgent2(?selling,?buyer), Agent(?buyer), //buyer
     hasObject(?selling,?thing), Thing(?thing), //product
     hasPrice(?selling,?money), Money(?money), //money
     // Sale is defined through Purchase
     hasSyncEvent(?selling,?buying), Buying(?buying),
     hasAgent(?buying,?buyer),
     hasAgent2(?buying,?seller),
     hasObject(?buying,?thing),
     hasPrice(?buying,?money),
     hasSyncEvent(?buying,?selling).
```

To reduce the variable usage the following changes were made to the language:

1. Instead of functional PSO notation, triples are now written in SPO notation without parentheses: `?selling hasAgent ?seller`. This is similar to RDF/Turtle (Beckett et al 2014) and offers a closer resemblance to natural language. Unary predicates (class assignments) are also written without parentheses but they precede the subject: `Agent ?seller`.
2. Two or more consecutive triples with the same subject can be joined together without repeating the subject: `?selling hasAgent ?seller hasAgent2 ?buyer` is equivalent to `?selling hasAgent ?seller, ?selling hasAgent2 ?buyer`. Turtle also supports this feature but requires triples to be separated by a semicolon. In Etalog no special

separator is needed, which resembles the behavior of prepositions in natural language, where we can attach multiple prepositions to the same verb: *'go from Moscow to Kazan'*. This feature allows us to write a complex proposition with a single usage of the subject variable.

3. If the subject variable is used only once in the expression, it can be omitted altogether when preceded by a class name. `Buying hasAgent ?buyer` is equivalent to `Buying ?x hasAgent ?buyer`. The variable name is generated automatically for internal purposes and is not shown to the user.
4. Nested expressions are allowed. In the place of a variable it is possible to use a (bracketed) complex proposition in which this variable is a subject. `?selling hasAgent (Agent ?seller)` is equivalent to `?selling hasAgent ?seller, Agent ?seller`. Nested expressions also allow the variable name to be omitted, if it is not used anywhere else: `?selling hasAgent (Agent).`
5. Native support for inverse relations introduced. Relations of the form `hasX` (such as `hasAgent`, `hasObject`, etc.) can be used in the inverse form of `isXOf` (`isAgentOf`, `isObjectOf`, etc.) even if the inverse relation is not defined in the ontology. `?seller isAgentOf ?selling` is equivalent to `?selling hasAgent ?seller` but has `?seller` as a subject. So it can be combined with other triples where `?seller` is a subject into a complex proposition: `Agent ?seller isAgentOf (Selling hasObject (Umbrella))`. This resembles the relative clause construction in natural languages: *'An agent who sold the umbrella'*.

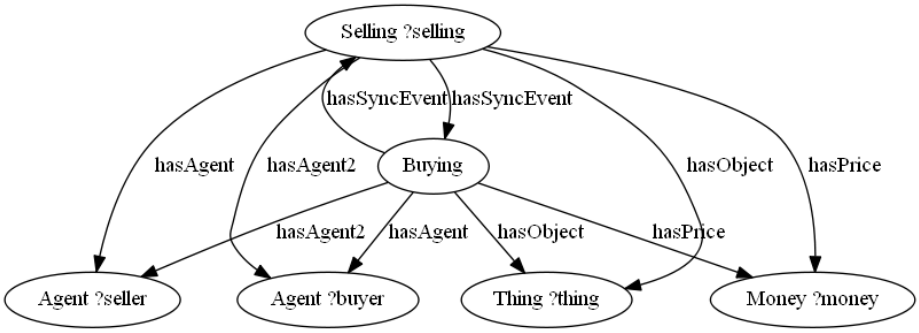
All these features greatly reduce the usage of variable names in Etalog expressions and make the Etalog text look more natural. Here is how the above rule will look like after all the features are applied:

```
(4) Rule Selling: // Sale
     Selling ?selling ->
     ?selling
       hasAgent (Agent ?seller) // seller
       hasAgent2 (Agent ?buyer) // buyer
       hasObject (Thing ?thing) // product
       hasPrice (Money ?money) // money
       hasSyncEvent
       ( // Sale is defined through Purchase
         Buying
           hasAgent ?buyer
           hasAgent2 ?seller
           hasObject ?thing
           hasPrice ?money
           hasSyncEvent ?selling
       ).
```

A semantic graph having a tree-like structure can be written in Etalog without mentioning any variable names whatsoever. This is similar to Description Logic (Baader et al 2003) and OWL Abstract Syntax (Patel-Schneider et al 2003). But unlike DL and OWL, which allow only tree-like expressions and do not make use of variables, in Etalog you can link different branches of the tree by introducing explicit variables and thus make a loop.

Loops are very useful in concept decomposition rules, where different parts of the decomposition are usually interconnected. For example, see below a semantic graph of rule (4). DL and OWL are not expressive enough to capture such links.

Fig. 3. Semantic graph of rule (4)



On the logical level Etalog is compatible with Datalog$^{+/-}$ (Calì et al 2011) and supports the same set of logical constructions, namely:

1. Conjunction
2. Implication
3. Implicit universal quantification in the antecedent
4. Implicit existential quantification in the consequent

Negation, disjunction, explicit quantification and modality are not supported at the level of language. There is some partial support for negation and modality in SemETAP semantic analyzer, but they are modelled as classes in the ontology, not as logical constructions of the language. Introducing full-fledged support for these features is constrained by RDF model (which has native support only for conjunction) and by the reasoner capabilities (adding these features can greatly increase the reasoning complexity).

3 Inference Rules Application Process

Another advantage of Etalog is that it hides technical details of the rule application from the linguist who creates the rule.

As we can see from the example above, concept decomposition almost always requires new variables to be present in the consequent which are not there in the antecedent. These variables represent various parts of the concept decomposition, in particular the arguments of an event.

Every situation of selling involves a seller, a buyer, a product and money. If anything is missing from the situation it cannot be called 'selling'. Nevertheless not all of these four arguments are necessarily instantiated in a sentence. In (1) there is no mentioning of money. In *'The car is sold'* three of the four arguments are missing. They may be missing in a sentence but they are still there in the situation the sentence describes. So, on the semantic level we have to restore all the missing arguments.

Rules are applied in the forward chaining manner known as *chase* (Benedikt et al 2017), i. e. they extend the existing semantic structure by adding new propositions and individuals to it. But we cannot just create new individuals for all new variables in the consequent. First we need to check if any of them are already there in the original semantic structure.

The logic of these checks can be complicated (see Rygaev 2017 for details) but the bottom-line is that an Etalog rule is internally split into a number of smaller implications, which are applied independently. Each such implication checks the existence of one or more individuals and creates them only if they are missing. For example, there will be a separate implication for each functional relation mentioned in the rule text.

So, for an Etalog rule it is possible that some individuals will be accommodated from the existing data while others will be added. This happens invisibly for the linguists who create rules in Etalog, so they can concentrate on the concept decomposition and ignore technical aspects of the rule application.

As we mentioned functional relations we have to note that Etalog rule compilation and application processes import certain data from an OWL ontology. This includes information about:

1. Class hierarchy
2. Inverse relations
3. Functional and inverse functional relations

Although all this information can be written in Etalog directly, it is much easier to include it in the ontology using OWL editors such as Protégé (Gennari 2003).

4 Semantic Structure Representation

Since the semantic structure of the sentence is also a set or triples, it is more than natural to represent it as an Etalog expression as well for better readability. But unlike inference rules, which are created manually, this task requires automatic generation of the Etalog expression from the set of triples.

Given that Etalog has substantial expressive power, we see that there are now many different ways to represent the same set of triples. So the question arises: how to choose the most appropriate expression? How to group nodes into complex propositions? Where to use nested expressions and inverse relations?

We came up with the following solution:

1. Do not invert any relations unless they are explicitly inverted in the original set of triples.

2. Identify the closest head for each individual in the structure. Starting from a node in the semantic graph and following the reverse direction of arrows identify the closest node which does not have any incoming arrows.
3. In case of loops when we return to the same node without finding a head – sort all individuals in the loop alphabetically and declare the first one to be the head.
4. Group all nodes by the closest head.
5. Express each group as a single complex proposition starting from the head (using the head as its subject).

Here is the Etalog expression for sentence (1)

```
(5) Selling #1
      hasObject (Umbrella #1)
      hasAgent (Human #1 hasGivenName "John")
      hasAgent2 (Human #2 hasGivenName "Peter")
      before SpeechTimeInterval
```

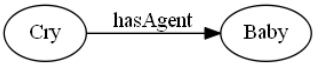
In this example we see another notation convention: a hash sign is used to shorten a variable name if it starts with the name of the class mentioned immediately before the variable. `Selling #1` is equivalent to `Selling ?selling_1`. Since all the individuals in the semantic structure obtain such naming, the new notation was introduced in order to exclude names duplication for better readability but still keep the class assignment explicit.

5 Communicative Dependencies

The expressiveness of the language allows capturing not only truth-conditional meaning of the sentence but some part of the communicative (information) structure as well. Although introducing such categories as theme/rheme or given/new will require additional means of expressing, Etalog is already capable of capturing communicative (referential) dependencies between nodes.

Igor Mel'čuk (2001) defines the communicative dependency in the following way: "In a semantic configuration (σ_1–σ_2), the semantic node (σ_2) is said to depend communicatively on the semantic node (σ_1) in a direct way, if this configuration can be reduced to (σ_1) (rather than to (σ_2)) such that the meaning conveyed is simply reduced but not distorted, the referent of (σ_1) remaining the same as that of the whole configuration (σ_1– σ_2)".

Fig. 4. Simple semantic configuration of two nodes (4)



For example, the referent of the semantic configuration in Fig. 4 can be either 'crying' (*the baby cries, crying of the baby*) or 'baby' (*the crying baby, the baby who is crying*) whereas the semantic dependency does not change ('baby' being the argument

of ‘crying’). The referent is determined by the communicative head of the configuration, not the semantic head. On the other hand the communicative head always corresponds to the syntactic head of a phrase when this configuration is expressed linguistically. “Comm-dependency is, so to speak, a way of 'foreseeing,' on the semantic level, the future syntactic dependencies” (Mel’čuk, ibid).

Using inverse relations we can express this configuration in Etalog either as `(Cry hasAgent (Baby))` or as `(Baby isAgentOf (Cry))`. So we can use Etalog syntax to represent the communicative dependencies between nodes while the underlying semantic structure (set of triples) remains the same.

(6) The writer burned the novel that he had written

```
(Burning #1
  hasAgent (Writer #1)
  hasObject
   (Novel #1
      isObjectOf
       (Writing #1
          hasAgent ?writer_1
       )
   )
)
```

(7) The novel was burned by the writer who had written it

```
(Burning #1
  hasObject (Novel #1)
  hasAgent
   (Writer #1
      isAgentOf
       (Writing #1
          hasObject ?novel_1
       )
   )
)
```

Sentences (6) and (7) describe the same situation but constitute different messages for the hearer. They answer different questions *(‘What did the writer do?’* and *‘Who burned the novel?’* respectively) and thus are communicatively different. In accordance to that, both Etalog expressions represent the same set of triples (the same truth-conditional meaning), but the syntax of the expressions mirrors the syntax of the sentences and thus reflects (some) communicative differences between them. They are organized in tree-like structures which resemble the syntactic trees of the original sentences.

On the one hand such semantic representation is language-independent (it contains no language-specific features of the original sentence). On the other hand it contains enough information to restore the syntax of the original sentence (to some extent). So it is a good candidate (at least better than just a set of triples) to serve as a language-independent interlingua to translate from one language to another.

6 Referring Expressions for Answers

Another application of Etalog is to present answers to questions. SemETAP is able to answer questions to a text. A natural language question is converted into a SPARQL query and is run against the semantic structure of the text. The query returns an individual which corresponds to a wh-word in the question.

Then we have to decide how to present it to the user. Ideally a natural language expression should be generated which allows the user to identify the object in the text. While we are not there yet, as an intermediate solution we present the answer expression in Etalog (Rygaev 2018). Here is an example:

(8) Text: After a pass by Kerzhakov into the penalty area Arshavin with a brilliant shot in the fall hammers the ball into the net.
Question: Which team does Arshavin play for?
Answer: `(FootballTeam isObjectOf (PlaysFor hasAgent (Human hasName "Kerzhakov")))`
Potential translation: The team which Kerzhakov plays for (The same team with Kerzhakov)

These Etalog expressions also capture not only truth-conditional meaning, but also the communicative (referential) structure of the answer. The expression is tree-like, with the found individual being the head of the phrase, and can serve as a template for the syntactic tree of the surface realization of the answer in natural language. Roughly what needs to be done is to replace ontological concepts with corresponding words and semantic relations with syntactic ones. This process is the opposite of the process of semantic analysis, which SemETAP is already capable to perform.

7 Conclusions

Etalog is a formal knowledge representation language, which is created to simplify reading and typing by linguists who work with it. It is based on the existing formalisms (Datalog$^{+/-}$, Description Logic, RDF/Turtle) and borrows a couple of ideas from natural language.

Etalog can express both *fact* knowledge (ABox in Description Logic terminology) as part of the semantic structure of the text and *rule* knowledge (TBox respectively) as a set of inference rules.

In addition, Etalog is able to capture some elements of communicative (referential) structure of the sentence, which makes it suitable to use as a language-independent interlingua for translation from one natural language to another.

8 References

1. Baader F., Calvanese D., McGuinness D.L., Nardi D., Patel-Schneider P.F. (2003). The Description Logic Handbook: Theory, Implementation, Applications. Cambridge University Press, Cambridge, UK, 2003.
2. Beckett D., Berners-Lee T., Prud'hommeaux E., Carothers G. (2014). RDF 1.1 Turtle. Terse RDF Triple Language. W3C Recommendation 25 February 2014. https://www.w3.org/TR/turtle/
3. Benedikt, M., Konstantinidis, G., Mecca, G., Motik, B., Papotti, P., Santoro, D., & Tsamoura, E. (2017). Benchmarking the chase. In Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (pp. 37-52). ACM.
4. Boguslavsky I.M., Dikonov V.G., Iomdin L.L., Lazursky A.V., Sizov V.G., Timoshenko S.P. (2015). Semantic Analysis and Question Answering: a System Under Development. Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference "Dialogue" (2015), p.62-79.
5. Boguslavsky I.M., Frolova T.I., Iomdin L.L., Lazursky A.V., Rygaev I.P., Timoshenko S.P. (2018). Semantic Analysis with Inference: High Spots of the Football Match. Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference "Dialogue" (2018), p.124-142.
6. Calì, A., Gottlob, G., Lukasiewicz, T., & Pieris, A. (2011). Datalog+/-: A family of languages for ontology querying. In Datalog Reloaded (pp. 351-368). Springer, Berlin, Heidelberg.
7. Gennari, J.H., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubézy, M., Eriksson, H., Noy, N.F. and Tu, S.W. (2003). The evolution of Protégé: an environment for knowledge-based systems development. International Journal of Human-computer studies, 58(1), pp.89-123.
8. Higginbotham J. (2000). On events in linguistic semantics. In: J. Higginbotham, F. Pianesi & A. Varzi (eds.). Speaking of Events. New York, Oxford: Oxford University Press, 49–79.
9. Mel'čuk, I. A. (2001). Communicative organization in natural language. John Benjamins Publishing Company.
10. Patel-Schneider P.F., Hayes P., Horrocks I. (2003). OWL Web Ontology Language. Semantics and Abstract Syntax. W3C Recommendation 10 February 2004. https://www.w3.org/TR/owl-semantics/
11. Parsons T. (1990). Events in the Semantics of English. A Study in Subatomic Semantics. Cambridge, MA: MIT Press.
12. Rygaev I. (2017), Rule-based Reasoning in Semantic Text Analysis. Proceedings of the Doctoral Consortium, Challenge, Industry Track, Tutorials and Posters @RuleML+RR 2017 hosted by International Joint Conference on Rules and Reasoning 2017 (RuleML+RR 2017).
13. Rygaev I.P. (2018). Referring Expression Generation for Question Answering and Graph Visualization. Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference "Dialogue" (2018), p.619-636.