Semantic Reasoning with Differentiable Graph Transformations

Alberto Cetoli
QBE Europe
London, UK
Email: alberto.cetoli@uk.qbe.com

***Abstract*—This paper introduces a differentiable semantic reasoner, where rules are presented as a relevant set of graph transformations. These rules can be written manually or inferred by a set of facts and goals presented as a training set. While the internal representation uses embeddings in a latent space, each rule can be expressed as a set of predicates conforming to a subset of Description Logic.**

Keywords–Semantic Reasoning, Semantic Graphs, Graph Transformations, Differentiable Computing.

I. INTRODUCTION

Symbolic logic is the most powerful representation for building interpretable computational systems [1]. In this work, we adopt a subset of Description Logic [2] to represent knowledge and build a semantic reasoner, which derives new facts by applying a chain of transformations to the original set.

In the restricted context of this paper, knowledge can be expressed in predicate or graph form, interchangeably. Thus, semantic reasoning can be understood as a sequence of graph transformations [3], which act on a subset of the original knowledge base and sequentially apply the matching rules.

In this paper, we show that rule matching can be made differentiable by representing nodes and edges as embeddings. After building a one-to-one correspondence between a sequence of rules and a linear algebra expression, the system can eventually train the embeddings using a convenient loss function. The rules created in this fashion can then be applied during inference time.

Our system follows the recent revival of hybrid neuro-symbolic models [1], combining insights from logic programming with deep learning methods. The main contribution of this work is to show that reasoning over graphs is a learnable task. While the system is presented here as a proof of concept, we show that differential graph transformations can effectively learn new rules by training nodes, edges, and matching thresholds through backpropagation.

In Section II we describe in detail the fundamentals of our reasoner, with working examples shown in Section III. Section IV reviews specific connections with prior works and finally a few remarks in Section V conclude the paper. The relevant code can be found at [4].

II. PROBLEM STATEMENT

The system presented here is a semantic reasoner inspired by the early STRIPS language [5]. It creates a chain of rules that connects an initial state of *facts* to a final state of inferred predicates. Each rule has a set of pre- and post-conditions, expressed here using a subset of Description Logic (DL). In the following, we restrict our DL to Assertional Axioms (ABox). Thus, each fact can be represented as a set of predicates, or - equivalently - as a graph with matching rules as described below.

A. Rules as graph transformations

We use a predicate form to represent facts, rules, and intermediate states, as shown in Figure 1. For example, the semantics for "Joe wins the election in the USA" is captured in the following form

```
joe(a), win(a,b), election(b), in(b,c), USA(c)
```

In the prior example, joe, $election$, and USA are nodes of the semantic graph, whereas win and in are convenient relations to represent the graph's edges.

The rules are specified with a MATCH/CREATE pair as below

```
MATCH person(a), win(a,b), election(b)
CREATE (a), be(a,b), president(b)
```

The MATCH statement specifies the pre-condition that triggers the rule, while the CREATE statement acts as the effect - or post-condition - after applying the rule. The result of applying this rule is shown in Figure 1, where a new state is created from the original fact. Notice that the name joe (which matches $person$) is propagated forward to the next set of facts.

By applying rules in sequence one builds an inferential chain of MATCH and CREATE conditions. After each rule the initial facts graph is changed into a new set of nodes and edges. This chain of graph transformations builds a path in a convenient semantic space, as shown in Figure 2. One of this paper's main result is to show that there is a one-to-one correspondence between the chain of matching rules and a chain of linear algebra operations.

B. Nodes and edges as embeddings

Both nodes and edges are represented as embeddings in a latent space. For convenience, in the current work the vocabulary of possible nodes matches the Glove 300dim dataset [6], whereas edges are associated random embeddings linked to the relevant ontology.

C. Matching nodes and edges

A rule is triggered if the pre-condition graph is a sub-isomorphism of the facts. Each node and edge of the pre-conditions has a learnable threshold value t. Two items match if the dot product between their embeddings is greater than a specific threshold. In the predicate representation, we make explicit these trainable thresholds by adding the symbol $>$ to the predicate's name. In this way, the rule in Section II-A becomes

```
MATCH person>0.6(a), win>0.7(a,b), election>0.6(b)
CREATE (a), be(a,b), president(b)
```

 ISBN: 978-1-61208-888-4

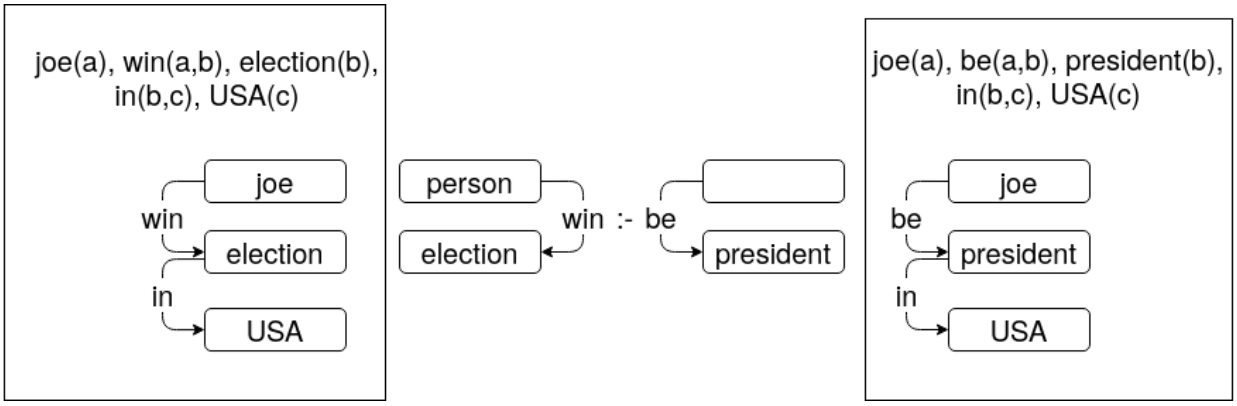

Figure 1. A matching rule example, as explained in II-A. The facts on the left are "transformed" into the ones on the right following the application of the relevant rule. This picture makes explicit the dual nature of the predicates/graph representation.

indicating that - for example - joe and $person$ would only match if their normalized dot product is greater than $t = 0.6$. In the Description Logic framework this is equivalent to an *individuality assertion*

$$joe \approx person \iff \text{embedding}(joe) \cdot \text{embedding}(person) > t$$

Matching facts and pre-conditions creates a *most general unifier* (MGU) that is propagated forward on the inference chain.

D. Creating a trainable path

During training, the final state is a *goal* of the system, as shown in Figure 2. The system learns how to create rules given a set of template *empty rules*, where the embeddings for each node and edge are chosen randomly. These templates are specified prior to the training using $*$ to indicate a random embedding, as in the following

```
MATCH *(a), *(a,b), *(b)
CREATE (b), *(b,d), *(d)
```

In the current state of development, the algorithm generates all possible paths - compatibly with boundary conditions - and then applies to each of them the training algorithm explained below. A more efficient method will be pursued in future works.

E. Training of the embeddings and thresholds

At every step of the inference chain the collection of predicates changes according to the order of transformations. At every step i, we employ a vector f_i that signals the truth value of each predicate. For computational reasons, the dimensions of this vector must be fixed in advance and set to the maximum size of the predicate set. The first value f_0 is a vector of ones, as every predicate in the knowledge base is assumed to be true.

At the end of the resolution chain there is a "goal" set of predicates, usually less numerous than the initial set of facts. A vector g indicates the truth conditions of the goal predicates. This vector - also of size n - contains a number of ones equal to the number of goal nodes and is zero otherwise. The application of a rule can then be described by two matrices: the similarity matrix S and the rule propagation matrix R.

A **similarity matrix** describes how well a set of facts matches the pre-conditions.

$$S_i = M_i \odot \text{Softmax}(P_i^T F_i - T_i) \tag{1}$$

Where P_i is the matrix with the pre-conditions's nodes as colums, F_i is the matrix with the fact nodes as columns at step i. M_i is the matrix of the matches, bearing value of 1 if two nodes match and vanishing otherwise. For example, if the first node of the pre-conditions matches the second node of the facts, the matrix will have value 1 at position (1, 0).

The matrix T_i is a bias matrix whose columns are the list of (trainable) thresholds for each predicate in the pre-conditions $T_i = \left[t_1^i, t_2^i, \ldots t_n^i\right]$. This bias effectively enforces the matching thresholds: A negative value as an argument to Softmax will lead to an exponentially small result after the operation.

All the matrices M, P, and F are square matrices $\in \mathcal{R}^{n \times n}$. Equation (1) is reminiscent of self-attention [7], with an added bias matrix T and a mask M.

A **rule propagation matrix** R puts into contact the left side of a rule with the right side. The idea behind R is to keep track of how information travels inside a single rule. In this work we simplify the propagation matrix as a fully connected layer with only one trainable parameter. For example, if the chosen size n is 4, a rule with three pre-conditional nodes and two post-conditional nodes has an R matrix as

$$R_i = w \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

where w is the "weight" of the rule. Given a first state f_0, the set of truth condition after n steps is

$$f_n = S_{n-1} \ldots R_1 S_1 R_0 S_0 f_0 \tag{3}$$

This final state f_n is compared against the goal's truth vector g to create a loss function.

The training of the relation embeddings follows the same sequence of operations as for the nodes. A set of truth vectors f^r

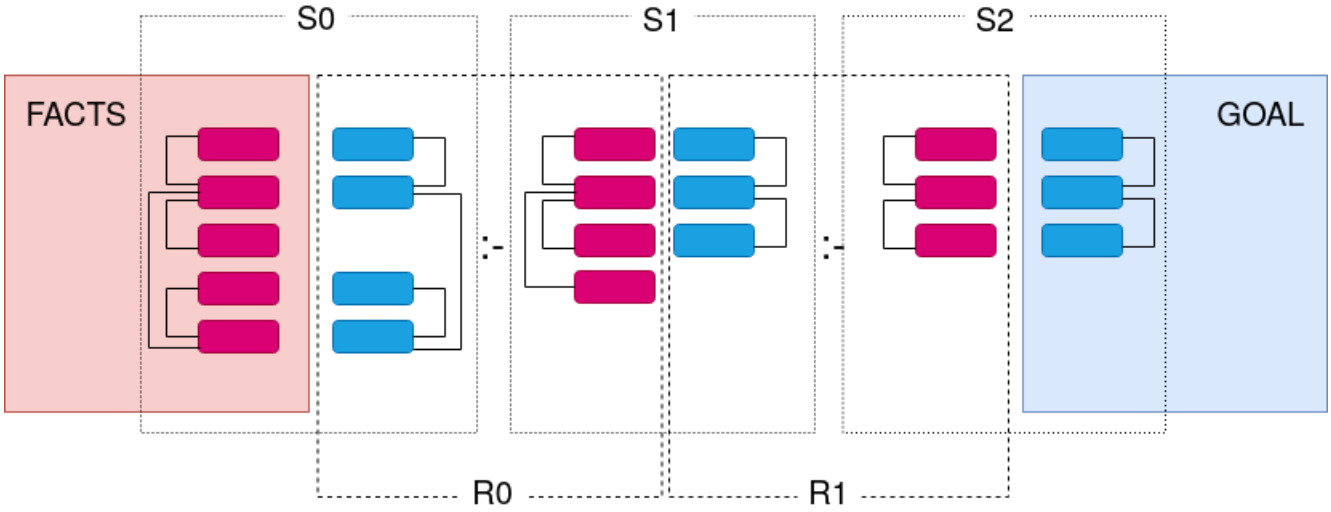

Figure 2. Example of a graph matching path. The facts on the left (represented as a graph of connected embeddings) are transformed through a chain of pre- and post-conditions into the goal on the right. This chain of rules is equivalent to a sequence of linear algebra operations, where the truth values of each predicated are propagated forward through a set of S and R matrices.

and states F^r is acted upon the relation similarity matrix

$$S_i^r = M_i^r \odot \text{Softmax}({P_i^r}^T F_i^r - t_i^r), \quad (4)$$

and the corresponding rule propagation matrix for relations R_i^r, leading to the final truth vector for relations

$$f_n^r = S_{n-1}^r ... R_1^r S_1^r R_0^r S_0^r f_0^r. \quad (5)$$

Following the example of the nodes, a goal vector for the relations is named g^r, containing the desired truth conditions for relations at the end of the chain.

The system learns the node and edge embeddings of the rules, while the initial facts and the goal are frozen during training. The system also learns the *matching thresholds* t and each rule's *weight* w. Following (3) and (5), the final loss function is computed as a binary cross entropy expression

$$\mathcal{L} = g \log(f_n) + g^r \log(f_n^r). \quad (6)$$

The system can in principle be trained over a set of multiple facts and goal pairs, in which case the loss function is the sum of all the pairs' losses. For simplicity, in this paper we limit the training to a single pair of facts and goal.

In order to avoid the Sussman anomaly, the same rule can only be used once in the same path.

III. Examples and Discussion

A. One-rule learning

As a toy example we want the system to learn that if someone is married to a "first lady", then this person is president. The facts are

```
person(a), spouse(a,b), person(b), be(a,c), first-lady(c)
```

and the goal is

```
person(a), profession(a,b), president(b)
```

Given the empty rule

```
MATCH *(a), *(a,b), *(b), *(a,c), *(c)
CREATE (b), *(b,d), *(d)
```

The system correctly learns the rule that connects the facts with the goal.

```
MATCH person>0.6(a), first-lady>0.6(b), person>0.6(c),
      be>0.63631916(a,b), spouse>0.6338593(a,c)
CREATE (b), president(d), profession(b,d)
```

While trivial, this is a fundamental test of the capacity of the system to learn the correct transformation. The matching thresholds have been clipped and cannot go below 0.6 in training.

While a successful result is almost guaranteed by choosing a rule that closely matches the boundary conditions, the system is proven capable of converging onto the correct embeddings and thresholds using just backpropagation.

B. Chained two-rule learning

While a single-rule transformation can be useful in a few edge cases, the real power of semantic reasoning comes from combining rules together. In this section we show - using another toy example - that the system can learn two rules at the same time. The simplified task is as in the following: to learn that "if a fruit is round and is delicious, then it is an apple." The facts are

```
fruit(a), be(a,b), round(b), be(a,c), delicious(c)
```

and the goal is

```
fruit(a), be(a,b), apple(b)
```

The system is given the two template rules to fit

```
MATCH *(a), *(a,b), *(b), *(a,c), *(c)
CREATE (b), and(b,c), (c)

MATCH *(a), and(a,b), *(b)
CREATE *(c), *(c,d), *(d)
```

Notice the "and" relations in the templates. These relations are frozen during training and constitute another constraint for the system to satisfy. In the end, our model learns the correct rules

```
MATCH fruit>0.6(a), round>0.6(b), delicious>0.6(c),
      be>0.6953449(a,b), be>0.6957883(a,c)
CREATE (b), (c), and(b,c)

MATCH round>0.6(a), delicious>0.6(b), and>0.9(a,b)
CREATE fruit(c), apple(d), be(c,d)
```

which satisfy the goal when chained.

Here, we forced the system to apply two rules since no single template would fit the boundary conditions. Of particular interest is the fact that the system learned the pre-conditions of the second rule $round > 0.6(a)$, $delicious > 0.6(b)$, $and > 0.9(a,b)$. This is not a trivial task, given that it started training with random embeddings and the only information about the correct values is the one propagated forward from the first rule.

IV. Related works

Neuro-symbolic reasoning has been an intriguing line of research in the past decades [8][9]. Some recent results make use of a Prolog-like resolution tree as a harness where to train a neural network [10], [11], [12], [13]. Our work is similar to theirs, but builds upon a STRIPS-like system instead of Prolog. A different approach employs a Herbrand base for inductive logic programming in a bottom-up solver [14].

Finally, one can see our method as a sequence of operations that create or destroy items sequentially. Each (differential) transformation brings forward a new state of the system made by discrete elements. These types of algorithms have already been investigated in the Physics community, for example in [15].

V. Conclusions

In this work we presented a semantic reasoner that leverages on differential graph transformations for rule learning. The system is built through a one-to-one correspondence between a chain of rules and a sequence of linear algebra operations. Given a set of facts, a goal, and a set of rules with random embeddings, the reasoner can learn new rules that satisfy the constraints. The rules are then written as a set of predicates with pre- and post-conditions, a more interpretable representation than embeddings and weights.

The system presented here is limited in speed and - as a consequence - volume of training data. This is mostly due to our path-creation algorithm, which generates all possible paths given a set of rules. A more efficient algorithm would employ a guided approach to path creation, similar to the method in [13]. A different and possibly novel efficiency gain could be found in a Monte Carlo method, where the path converges to the correct one through means of a Metropolis algorithm.

Using a more efficient algorithm the system would be able to leverage on a higher number of templates, thus making the system useful outside the set of toy examples presented here. In this scenario, another topic that needs addressing is how to best generate the templates from the available data.

Finally, an open question resides on whether the system is able to generalize, given multiple sets of facts and goals. This last inquiry will need a faster algorithm and will be pursued in a future work.

References

[1] A. d'Avila Garcez and L. C. Lamb, "Neurosymbolic AI: The 3rd Wave," arXiv e-prints, Dec. 2020, p. arXiv:2012.05876.

[2] M. Krötzsch, F. Simancik, and I. Horrocks, "A description logic primer," ArXiv, vol. abs/1201.4089, 2012.

[3] H. Ehrig, C. Ermel, U. Golas, and F. Hermann, Graph and Model Transformation. Berlin, Heidelberg: Springer-Verlag, 01 2015.

[4] "Source code," 2021. [Online]. Available: https://github.com/fractalego/dgt/tree/semapro2021

[5] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," Artificial Intelligence, vol. 2, no. 3, 1971, pp. 189–208. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370271900105

[6] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[8] A. S. d. Garcez, D. M. Gabbay, and K. B. Broda, Neural-Symbolic Learning System: Foundations and Applications. Berlin, Heidelberg: Springer-Verlag, 2002.

[9] A. Garcez, L. Lamb, and D. Gabbay, "Neural-symbolic cognitive reasoning," in Cognitive Technologies. Berlin, Heidelberg: Springer-Verlag, 2009.

[10] T. Rocktäschel and S. Riedel, "End-to-end differentiable proving," in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/b2ab001909a8a6f04b51920306046ce5-Paper.pdf

[11] P. Minervini, M. Bosnjak, T. Rocktäschel, and S. Riedel, "Towards Neural Theorem Proving at Scale," in ICML Workshop on Neural Abstract Machines and Program Induction (NAMPI), 2018.

[12] L. Weber, P. Minervini, J. Münchmeyer, U. Leser, and T. Rocktäschel, "NLProlog: Reasoning with weak unification for question answering in natural language," in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 6151–6161. [Online]. Available: https://aclanthology.org/P19-1618

[13] P. Minervini, M. Bosnjak, T. Rocktäschel, S. Riedel, and E. Grefenstette, "Differentiable Reasoning on Large Knowledge Bases and Natural Language," in Proceedings of the Association for the Advancement of Artificial Intelligence Conference on AI (AAAI), 2020.

[14] R. Evans and E. Grefenstette, "Learning explanatory rules from noisy data," J. Artif. Int. Res., vol. 61, no. 1, Jan. 2018, p. 1–64.

[15] A. W. Sandvik, "The stochastic series expansion method for quantum lattice models," in Computer Simulation Studies in Condensed-Matter Physics XIV, D. P. Landau, S. P. Lewis, and H.-B. Schüttler, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 182–187.

 ISBN: 978-1-61208-888-4