Rule-Based Reasoning using State Space Search

Dieter De Paepe, Ruben Verborgh, and Erik Mannens

Ghent University – iMinds
Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
dieter.depaepe@ugent.be

Abstract. Semantic Web reasoners are powerful tools that allow the extraction of implicit information from RDF data. This information is reachable through the definition of ontologies and/or rules provided to the reasoner. To achieve this, various algorithms are used by different reasoners. In this paper, we explain how state space search can be applied to create a backward-chaining rule-based reasoner. By using the OWL-profiles (especially OWL 2 RL), DL-based reasoning becomes possible. State space search is an approach used in the Artificial Intelligence domain that solves problems by modeling them as a graph and searching (using diverse algorithms) for solutions within this graph. State space search offers inherent proof generation and the ability to plug in different search algorithms to determine the characteristics of the reasoner such as: speed, memory or ensuring shortest proof generation.

Keywords: rule-based reasoning, state space search

1 Introduction

Semantic reasoners are used to infer new knowledge from existing knowledge. Ideally, a verifiable proof can be provided for each inference. Despite the importance of proofs for the *trust* component in the Semantic Web Stack[1] envisioned by Tim Berners-Lee, proof generation is only supported by very few reasoners (such as EYE [10] and cwm [3]).

Algorithms used by current reasoners are: Euler path detection (EYE), tableaux (Pellet [6]) and RETE (FuXi [1]). This paper describes how *state space search* [8], a concept from Artificial Intelligence in which problems are solved by exploring a state graph, can be applied to reasoning. Using different approaches is important since it allows reasoners to be truly independent, once again aiding to establish the trust component in the Semantic Web Stack.

By plugging different search algorithms into state space search, the characteristics of the reasoner can be tweaked. This includes speed and memory usage, but also the ability to generate the shortest proof, a property not directly supported by any other reasoner. For applications such as RESTdesc [9], where proofs are used to compose web services, the length of a proof plays an important role.

[1] https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

2 Context and Related Work

State space search transforms a problem context into a directed *state graph*, where each vertex represents a certain state of the context being modeled and the edges are determined by what is considered a valid transition between states. In this graph, both the starting situation and solution are represented by one or more vertices. This graph may or may not be materialized for a specific problem. *Path finding* is one of the more well-known applications of state space search.

The process of solving the problem consists of finding a path between the starting vertex and one of the solution vertices. This is done by creating a *search tree* that explores the state graph. The search tree and the state graph are separate concepts, as the search tree tracks information about the solution being constructed. In the example of path finding the state graph coincides with the graph being navigated where each vertex represents a location. The search tree tracks how each node was reached (one path may be better than the others).

The performance of state space search depends on the algorithm used to construct the search tree. Depending on the requirements, a suitable algorithm may be selected. Well-known algorithms include: A*, IDA* [5], SMA* [7] and (stochastic) beam search [11].

Although any form of backward-chaining reasoning could be considered as a form of state space search, it has not been examined as such in current literature to the best of our knowledge.

3 Rule-Based Reasoning using State Space Search

Rule-based reasoners differ from ontological reasoners since they are configured to only use a specified set of rules rather than the OWL 2 specification. Luckily, big parts of OWL 2 can still be used for rule-based reasoning through OWL 2 RL, a subset of OWL 2 that can be expressed in rules.

In rule-based reasoners, data and rules are both provided as input. Listing 1.1 and 1.2 list example rules and data in Notation3 (N3) [4].

```
1 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
2 @prefix : <http://www.example.com/>.
3 {?x a :CoolPerson} => {?x :has :sunglasses}.              # Rule 1
4 {?x a ?Sub. ?Sub rdfs:subClassOf ?Super} => {?x a ?Super}. # Rule 2
```

Listing 1.1. Example rules in N3.

```
1 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
2 @prefix : <http://www.example.com/>.
3 :MovieStar  rdfs:subClassOf :CoolPerson.
4 :Researcher rdfs:subClassOf :CoolPerson.
5 :alice a :Researcher.
```

Listing 1.2. Example facts in N3.

Using state space search, we can design a backward-chaining rule-based reasoner that allows us to prove given statements. Each node in the search tree contains a number

of statements that are yet to be confirmed. Starting from the root node which contains the statement to be proven, the children of each node represent the different ways to confirm a remaining statement from its parent. This can be done by finding the statement in the known facts, or by applying a rule that proves the statement given that its prerequisites are satisfied, thereby introducing new statements to be proven. A solution is found when a node is reached where no remaining statements are to be proven. The path from the root node to the solution acts as the corresponding proof. Figure 1 demonstrates an example search tree.

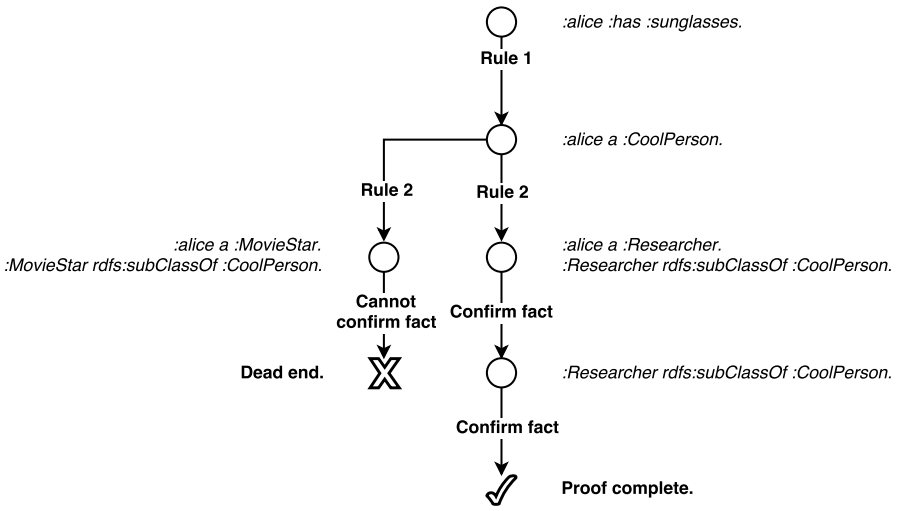

Fig. 1. Search tree for the given data and rules, proving that `:alice :has :sunglasses`. Each node lists the remaining facts to be proven. Statements are resolved in order in this example.

In more realistic examples, the search tree might contain millions of nodes. The chosen search algorithm determines how much of the search tree will be constructed before a solution is found.

4 Advantages of State Space Search Reasoning

By allowing to change the search algorithm, it is possible to tweak the characteristics of the reasoner. For example, it would be possible to guarantee that the shortest possible proof is found by using a variant of the A* algorithm. If memory is the most important factor, a variant of depth first search could be used. This choice does not have to be made in advance, it could be specified through configuration for each reasoner run.

Because multiple search algorithms are supported, it is easy to tweak the performance of the reasoner to specific use cases. It is also possible to create context agnostic heuristics to further improve these search algorithms by analyzing the common patterns of previously generated proofs. This could be similar to the approach used by Arndt et al. [2], where specialized rules are generated in an intermediate step to speed up reasoning by 75%.

Finally, the usage of different approaches to reasoning is not just interesting from an academic point of view. It is a vital part of the trust component in the Semantic Web Stack, where exchangeable proofs should be checked by independent reasoners using different algorithms.

5 Conclusions

In this paper, we explained our proposal to apply state space search to create a backward-chaining rule-based semantic reasoner. Using different algorithms is important to enable trust, but using state space search offers other specific advantages: 1) it is trivial to construct proofs during reasoning due to the nature of state space search; 2) the ability to switch between different search algorithms to determine the characteristics of the reasoner, such as guaranteeing the generation of the shortest proof; 3) the potential gain of automatically updating search heuristics based on previous queries.

References

1. Fuxi 1.4: A python-based, bi-directional logical reasoning system for the semantic web, https://code.google.com/archive/p/fuxi/
2. Arndt, D., De Meester, B., Bonte, P., Schaballie, J., Bhatti, J., Dereuddre, W., Verborgh, R., Ongenae, F., De Turck, F., Van de Walle, R., Mannens, E.: Improving OWL RL reasoning in N3 by using specialized rules. In: Tamma, V., Dragoni, M., Gonçalves, R., Ławrynowicz, A. (eds.) Ontology Engineering: 12th International Experiences and Directions Workshop on OWL. Lecture Notes in Computer Science, vol. 9557, pp. 93–104. Springer (Apr 2016), http://dx.doi.org/10.1007/978-3-319-33245-1_10
3. Berners-Lee, T.: cwm, https://www.w3.org/2000/10/swap/doc/cwm.html
4. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A Logical Framework For the World Wide Web. Theory and Practice of Logic Programming 8(03), 249–269 (2007), http://arxiv.org/abs/0711.1533
5. Korf, R.E.: Depth-first iterative-deepening. Artificial Intelligence 27(1), 97–109 (Sep 1985), http://linkinghub.elsevier.com/retrieve/pii/0004370285900840
6. Parsia, B., Sirin, E.: Pellet : An OWL DL Reasoner. Artificial Intelligence pp. 1–2 (2000)
7. Russell, S.: Efficient memory-bounded search methods. In: Proceedings of the 10th European Conference on Artificial Intelligence. pp. 1–5. ECAI '92, John Wiley & Sons, Inc., New York, NY, USA (1992), http://dl.acm.org/citation.cfm?id=145448.145476
8. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson, third edn. (2010)
9. Verborgh, R., Arndt, D., Van Hoecke, S., De Roo, J., Mels, G., Steiner, T., Gabarró Vallés, J.: The pragmatic proof: Hypermedia API composition and execution. Theory and Practice of Logic Programming (2016), http://arxiv.org/pdf/1512.07780v1.pdf
10. Verborgh, R., De Roo, J.: Drawing conclusions from linked data on the web: The EYE reasoner. IEEE Software 32(3), 23–27 (2015)
11. Wang, F., Lim, A.: A stochastic beam search for the berth allocation problem. Decision Support Systems 42(4), 2186 – 2196 (2007), http://www.sciencedirect.com/science/article/pii/S016792360600090X, decision Support Systems in Emerging Economies