Hybridization of Description Logics and Logic Programming

Arun Raveendran Nair Sheela*,[1],[2],

[1] Clermont Auvergne University, France
[2] Thales Group
`Arun.RAVEENDRAN_NAIR_SHEELA@doctorant.uca.fr`
`arun.raveendran-nair-sheela@thalesgroup.com`

Abstract. Thales[3] aims to develop a virtual assistant to support pilots during flights, with a central component being the knowledge base. This knowledge base is built using a knowledge representation(KR) and reasoning system, encompassing various knowledge types including static and dynamic. However, existing KR systems present some limitations which include expressiveness and reasoning performance. The problem of expressiveness can be addressed by integrating two distinct KR concepts: Rules and Ontologies.

Ontologies offers a framework for formalizing concepts, properties, and relationships, whereas rules express knowledge through IF-Then constructs. Integrating these approaches enriches knowledge representation and reasoning systems in many ways and helps to achieve completeness. However, this integration poses challenges, such as the difficulty of aligning their semantics and addressing issues of decidability. This thesis focuses on defining a methodology to combine rules and ontologies to overcome these challenges and build an optimized reasoner to execute the reasoning tasks of the virtual assistant ensuring good performance.

Keywords: Description Logic · Rules · Logic Programming · Hybrid Knowledge Base .

1 Introduction

Artificial Intelligence(AI) exhibits various applications in the aviation industry such as virtual assistants. Deploying a virtual assistant during flights enhances pilot proficiency and ensuring the seamless execution of flight operations. In this context, Thales plans to construct a virtual assistant using a symbolic AI system, which falls under the domain of AI. The entirety of information within the virtual assistant is systematically represented in its knowledge base using a KR language. Utilizing this knowledge base with other components, the virtual assistant guides pilots through the execution of varied tasks, facilitated by a sophisticated reasoner.

Some use cases of this virtual assistant include fault detection and troubleshooting, providing suggestions to pilots for improving flight performance,

---

* Early Stage Ph.D - Work Started From September 2023.

[3] https://www.thalesgroup.com/

answering queries, and assisting in the decision-making process during critical situations. To execute these use cases, knowledge engineers represent contextual information along with instantaneous knowledge such as flight machinery data, meteorological data, and information from air traffic controllers in the knowledge base. This knowledge can be classified into three different types: static knowledge, which remains unchanged during flight; dynamic knowledge with infrequent updates and dynamic knowledge with periodic updates which changes during flights.

For the effective construction of a knowledge base, essential features include the Closed-World Assumption(CWA) means assuming false if something cannot be derived as true, beneficial for tasks like representing a list of airports, and the Open-World Assumption(OWA) means no conclusions is made from the absence of information, useful for use cases related to weather information. Also, we need to represent some procedural rules that help with fault detection and rectification. Additionally, Non-monotonic Reasoning, entails that new knowledge will invalidate the previously drawn conclusion, is required to perform reasoning over dynamic knowledge, which changes over time. A comprehensive solution to include all these features is to integrate two major KR concepts: Ontologies and Rules.

Ontologies are structured representations of knowledge within specific domains, organized hierarchically to facilitate effective knowledge management and representation[25]. Description Logic(DL)s, a decidable variant of first-order logic is a widely accepted ontology language[1]. DLs provides a systematic way to represent and reason about a domain knowledge by defining concepts, relationships between concepts, and constraints on these relationships.

Rules capture knowledge about the world by expressing the IF-Then relationship, There are many varieties of rule languages. But in general, they can be divided into Production systems and Logic Programming(LP) systems[10]. Production systems express knowledge using conditional statements with a specific control flow and the Logic Programming system uses a declarative style which means modelling what should be achieved instead of how to achieve it.

LP and DLs have many distinct features. So combining both approaches will enhance the capabilities of a knowledge representation and reasoning system. In [24], define several benefits of LP over DLs. LP facilitates the representation of non-tree-like relationships, offering a more flexible and dynamic approach to expressing complex connections within a knowledge domain and operates under the CWA and is non-monotonic. N-ary predicates , ability to represent integrity constraints, and exception modeling are some of the other benefits of LP. Furthermore, the availability of highly optimized reasoners enhances the efficiency of reasoning processes.

Contrarily, DLs offers a rich set of constructors for complex entity connections and allow reasoning over abstract relationships without requiring concrete instances, known as Terminological Reasoning. Reasoning with DL is based on OWA and it is monotonic which means new knowledge doesn't invalidate the previously drawn conclusion. Various DLs can be defined based on the combination of different constructors offering flexibility to choose a language that matches the required expressivity. Moreover, DLs enable Open-domain Reasoning, accommodating an infinite number of anonymous individuals within their reasoning framework.

Overall, LPs are suited for data-centric reasoning tasks like query answering, while DLs stand out for their terminological reasoning capabilities and hierarchical representation style. Despite the benefits, there are several challenges we need to face while combining DLs and LP due to inherent differences in the semantics of both approaches [13],[27]. This includes managing default assumptions such as CWA and OWA, monotonicity, treatment of equality, domain, and negation choices(Weak Negation for LP and Explicit Negation for DLs). Furthermore, decidability is another concern during this reconciliation. However, several works are being published that address the aforementioned challenges and give solutions to solve them(explained in section 2.3).

2 State of the Art

This section is split into three parts, covering the languages and reasoners for LP systems and DLs separately, and also delving into works that combine both approaches.

2.1 Reasoning in Description Logics

The current standard for ontologies is the Web Ontology Language(OWL), rooted in DLs. OWL offers a range of variants offering diverse constructors, each with distinct computational complexities tailored for specific use cases. At one end, there's the most expressive variant, OWL Full[30], allowing for rich and complex modeling but undecidable, OWL 2 introduced more tractable variants, such as OWL 2 RL (Rule Language), OWL 2 EL (Existential Language), and OWL 2 QL (Query Language)[8]. Listing 1.1 defines a small Description Logic(DL) with a Terminological Box(TBox) that delineates the abstract relationship among classes and properties, and an Assertional Box(ABox) specifies instances of these classes and properties. Equivalent OWL syntax of this is mentioned in Listing 1.2.

Listing 1.1. Example DL ontology

```
<!-- Axioms/Terminological Box/TBox -->
TBox = {Parent subClass Human }.
<!-- Assertional Box/ABox/Instances -->
ABox = {Parent(individual_a)}.
#Reasoning Tasks
<!- Subsumption Checking ->
?- subClass(Human,Parent). - Result: Yes.
<!- Instance Checking ->
?- Human(individual_a). - Result:Yes.
<!- Instance Retrivel ->
?- Human(X). - Result:individual_a.
```

Listing 1.2. Equivalent OWL Syntax

```
<!-- Axioms/Terminologies/TBox -->
<owl:Class rdf:about="#Human"/>
<owl:Class rdf:about="#Parent">
  <rdf:subClassOf rdf:resource="#Human"/>
</owl:Class>
<owl:Class rdf:about="#Father">
  <rdf:subClassOf rdf:resource="#Parent"/>
</owl:Class>
<!-- Assertions/ABox/Instances -->
<owl:NamedIndividual rdf:about="#individual_a">
  <rdf:type rdf:resource="#Human"/>
</owl:NamedIndividual>
```

In OWL languages, key reasoning tasks include consistency checking ensures logical coherence, instance checking verifying individual membership in specified concepts, subsumption checking for assessing concept hierarchy relationships and instance retrieval involves extracting relevant instances based on defined criteria. Several DL reasoning methodologies, such as Tableau-based algorithms, Classification, Rule-based DL reasoning, First-order Theorem prover-based DL reasoning, and DLE framework-based reasoning, are commonly employed to execute the mentioned reasoning tasks. Tableau-based reasoning is a method that constructs proof trees to ascertain the consistency of DL ontologies employed by well-known reasoners such as Pellet[15] and HermiT[31]. ELK is a reasoner that utilizes classification, the process of determining the subsumption relationships between concepts within an ontology by applying inference rules[38]. Additional reasoning methods include rule-based DL reasoners by taking advantage of rule engines to perform efficient reasoning over very large ABox(RDFox), by using first-order theorem provers(Surnia[18], Hoolet[5] using Vampire Theorem Prover) and Description Logic Entailement(DLE) framework by combining the benefits of tableau algorithm for effective TBox reasoning and rule engines for ABox reasoning[23](DLE-Jena).

2.2 Reasoning in Logic Programming

Different LP systems include Answer Set Programming (ASP)[22], General Logic Programming[29], F-logic[20], and Constraint Logic Programming[34]. Notably, these declarative languages primarily focus on query answering as the principal reasoning task, except ASP, which encompasses tasks such as brave and cautious reasoning, and answer set checking. ASP focused on generating answer sets, while other languages often utilize a backward chaining approach,attempting to prove the query by tracing back from known evidences.

Listing 1.3. Example of a Logic Programming System

```
<!-- Rule: All birds can fly. -->
bird(X) -> fly(X).
<!-- Fact:Tweety is a bird. -->
bird(Tweety).
<!- Query Answering. ->
fly(X).
<!- Result ->
Tweety.
```

For General Logic Programming, reasoning engines such as SWI-Prolog, XSB Prolog, and B-prolog are commonly employed. In ASP, dedicated reasoning engines such as Clingo, DLV, and S-Models and F-logic relies on engines like Florid and Flora2.

2.3 Combining DLs and LP System

Numerous research endeavors have explored the amalgamation of DLs and LP, leading to a categorization of below integration methods based on the intended semantics of the combined language, its expressivity, and the nature of the interaction between them. Broadly, the integration strategies can be distilled into two overarching approaches based on the semantics of the integrated languages: the Homogeneous Approach and the Hybrid Approach[14],[11].

Homogeneous Integration utilized a unified language with a single semantics that embedded both DLs and rules. Based on the expressivity, we can further classify Homogeneous integration into monotonic and quasi-monotonic approache(also called Full Integration or Embedding Approach).

Monotonic approach is mainly focused on first-order semantics and employs only open-world assumption. Semantic Web Rule Language(SWRL)[19], a union of DLs and Horn Logic is considered as a language of this approach. But in general, SWRL is considered undecidable, so various tractable forms were invented which include Description Logic Program(DLP)[6], Description Logic Rules[21], DL Safe Rules[21], and Existential Logic Program(ELP)[21]. There are mainly three ways to construct a reasoner for this approach including using a first-order theorem prover, an OWL reasoner, or a rule engine. In all these cases, we need to use a translator to convert either rules or DLs or both to the language accepted by the reasoner.

Quasi-monotonic approach extent the above approach by adding non-monotonic features by utilizing the extensions of first-order logics such as circumscription, default logic, and defeasible logic. In this approach, both open-world and closed-world assumptions are employed to execute the reasoning tasks. MKNF+[9] and Hybrid MKNF[3], based on auto-epistemic logic and DR-logic[7] based on defeasible logic are some of the existing languages for this approach.Also, Open-Answer Set Programming(OASP) is another approach that can be considered in this category[32]. We can construct a reasoner for this approach by using a rule engine by guaranteeing the semantics of the language. NoHr Reasoner is a query answering tool based on Hybrid MKNF, which uses XSB Prolog as the backend and uses a direct translator for OWL to LP conversion[36]. NoHr supports the tractable forms of OWL such as OWL 2 RL, OWL 2 EL, and OWL 2 QL. Also, there exists a prototype implementation of DR-Prolog, which uses the same XSB Prolog as the backend.

In **Hybrid Integration**, two distinct semantics coexist, one dedicated to DLs and the other tailored for LP. Based on how DL and LP interact, we can further classify the hybrid integration into two types, Loose and Tight Integration[12].

In **Loose Integration**, the DL and LP components are treated as distinct entities that share knowledge by exchanging the logical consequences derived from each other. DL-Program, as described in [4], employs a loose integration approach. Several implementations such as NLP-DL[35], a web interface using Racer and DLV, DReW System[17] utilizing DLV and F-Logic#[33] appointing onto-broker engines(both f-logic and DL), are built on DL-programs. DLV-Hex, built upon the Hex system, can also be regarded as a reasoner for DL-Programs."

In **Tight Integration**, the interaction between LP and the DL part occurs at the level of the semantics of both components. In this approach, a shared model for both LP and DL is established and computed by deriving a model for DL under first-order semantics. Subsequently, the model of LP is computed by employing either well-founded semantics or stable-model semantics after eliminating the DL atoms by utilizing the initially derived DL model. So for each DL model, there exists a corresponding LP model. DL-Log under stable model semantics[26] and HD-Rules under well-founded semantics[37] are some of the existing languages that use tight integration. Also, there exists a prototype reasoner for HD-Rules as a web interface that uses Pellet and XSB Prolog as the backend.

In addition to this, within Hybrid Integration, there is a third approach that blends the characteristics of loose and tight coupling which can be called **Flexible Integration**. Resilient Logic Program is an example of this kind of approach[28].

3 Problem Statement

The research questions outlined below delineate the distinct areas explored in this research:

- **RQ1 - How to combine DLs with LP system?** This research question will be addressed by exploring four sub-research questions defined below:
  - **RQ1.1 - What functional and non-functional requirements does the application domain expect from the integrated KR system?** Functional requirements address expressivity, information flow, the default assumptions for a rule, and DL predicates, emphasizing theoretical aspects. Non-functional requirements define characteristics such as query-answering performance, scalability, efficiency, and real-time reasoning performance, predominantly at the reasoner implementation level.
  - **RQ1.2 - Which kind of knowledge is better suited for modeling with DL, conversely, which one is more effectively represented through LP?** Carefully evaluating specific criteria is crucial when deciding whether to employ rules, Description Logic (DL), or both to model a particular use case.
  - **RQ1.3 - What are the existing methods to integrate DLs with LP?** The question aims to identify existing frameworks and, based on these, comprehend the general methods for integrating rules and DL.
  - **RQ1.4 - Is one of these existing methods sufficient for this application domain?** Identify a suitable framework that satisfies the functional and non-functional requirements of the virtual assistant. If the current formalisms prove inadequate, then the following questions must be addressed:
    - **RQ1.4.1 -** Why existing methods are not sufficient?
    - **RQ1.4.2 -** What improvements are needed?
    - **RQ1.4.3 -** Should we need to work on the theoretical aspects or just reasoner optimization?
- **RQ2 - How to define the semantics of a language that integrates rules and DL to fulfill the functional requirements of this application domain?** The answer to this research question depends upon RQ1(specifically, SRQ4) to identify or invent suitable semantics to combine LP and DL. This research question also aims to study the decidability and tractability of the language and discusses the constraints to enable it.
- **RQ3 - How to perform essential reasoning tasks such as query answering?** This question aims to build a reasoner that adheres to the defined declarative semantics of the language.

RQ3.1 - What architecture and algorithms are most appropriate for this? Two approaches exist for constructing a reasoner for the integrated language: a single rule reasoner or a coupled two-reasoner setup. And when it come to algorithms, options include there forward chaining and backward chaining systems. So identify an effective methodology to construct the reasoner is pivotal.

RQ3.2 - Is it possible to utilize existing tools for implementation, and if so, which one is the most suitable? Identifying the most efficient tool for implementing the operational semantics to execute the required reasoning task.

RQ3.3 - How to optimize the execution of reasoning tasks to achieve optimal results? Optimization techniques at both the algorithmic and implementation levels are explored to meet the non-functional requirements of the application domain.

4 Research Methodology

This research is structured into three distinct phases to address the aforementioned research questions: the State of the Art Phase, Experimentation Phase, and Evaluation phase as illustrated in Figure 1. The diagram also specifies the priority assigned to each research question.

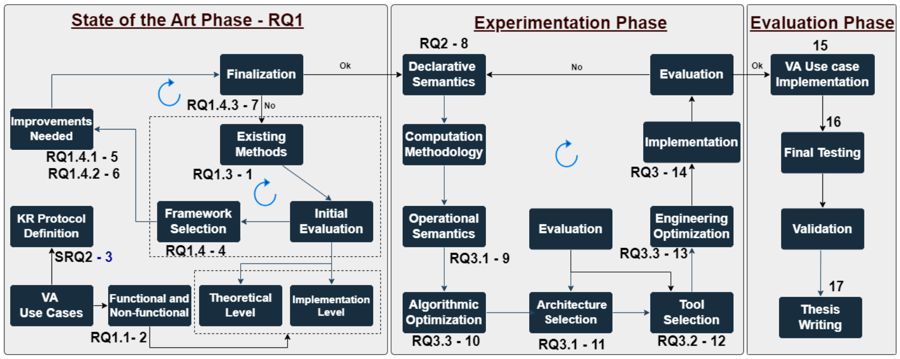

Fig. 1. Research Methodology

5 Evaluation Plan

We need to evaluate the result of this thesis theoretically by implementing the use cases to ascertain the fulfillment of the functional requirements. Key parameters for assessment include **Expressivity**, **Information Flow**, **Domain**, and **Negation choices for DL and LP Predicates**. As requirements evolve, additional parameters will be integrated.

Besides this, we need a quantitative evaluation to analyze the non-functional requirements. We decided to use existing OWL benchmarks incorporating a rule base with it for this. We utilize OWL2Bench, a dataset capable of generating a DL knowledge base and an extension of University Ontology Benchmark (UOBM) with a TBox encompasses four OWL languages(OWL 2 RL, OWL 2 QL, OWL 2 EL, OWL 2 DL) with all possible constructors and an ABox generator of variable size[16]. Selected parameters to evaluate non-functional requirements include **Query Answering Performance**, **Scalability**, **Query Result Comparison** using the Jaccard Similarity index[2], **Real-time Reasoning Performance and Timeliness**, and **Efficiency**. If existing benchmarks proves inadequate, there is a plan to develop an in-house benchmark that integrates both DL and LP to perform the evaluation.

6 Preliminary Result

Answering **RQ1.1**, the functional requirement involves defining a Knowledge Representation language capable of expressing various virtual assistant use cases, where certain scenarios require evaluation under an open-world assumption and others under a closed-world assumption. Additionally, there is a need for bi-directional information exchange between these cases. Also, the KR system must be able to capture and reason with the real-time data generated in flight. However, the expressivity and specific reasoning tasks required for operational use cases of the virtual assistant are yet to be defined. Addressing non-functional requirements, the KR system must exhibit good reasoning performance, scalability, real-time reasoning capabilities, and efficiency in terms of resource consumption.

RQ1.3 is addressed in section 2.3 and Table 1 illustrates the general features of methods to combine LP and DLs (Row 1) , including semantics (Row 2), the interaction between LP and DL components (Row 3), default assumptions (Row 4), the domain consideration (Row 5), and a review of existing works (Row 6).

Homogeneous - Monotonic	Homogeneous - Quasi-Monotonic	Hybrid - Loose Integration	Hybrid - Tight Integration
First-order(FO) Semantics	FO Non-Monotonic Semantics	DL under FO Semantics, Rules under NM semantics	DL under FO Semantics, Rules under NM semantics
Intralingual	Intralingual	Logical Consequences	Model-Based
Only OWA	OWA and CWA for Rules and DL atoms	OWA for DL atoms, CWA for Rule atoms	OWA for DL atoms, CWA for Rule atoms
Both rule and DL atoms are open	Open and closed atoms are allowed in both rules and DL	DL atoms are open, Rule atoms are closed	DL atoms are open, Rule atoms are closed
SWRL	Hybrid MKNF	DL-Program	DL-Log, HD-Rules

Table 1. Different ways to integrate Rules and DL

Answer to **RQ1.4**, The Monotonic approach, while a straightforward methodology, lacks certain features essential for the application domain, such as the closed-world assumption. The Quasi-monotonic approach is often considered undecidable, requiring significant restrictions in language expressivity to achieve decidability. Moreover, building a reasoner for Quasi-monotonic that captures the complex semantics is challenging. Also, Knowledge engineers may find it difficult to understand the integrated language, exemplified by the definition of Hybrid MKNF semantics involving operators namely K and Not. In the realm of Hybrid tight integration, the necessity to identify all models of DL, each corresponding to a grounded LP program, introduces complexity. So initially we planned to start working with a Loose integration approach, which is a very simple way to integrate rules and DL which also satisfies the initial functional requirements needed for the application domain. Furthermore, we can use very expressive syntax on both sides. DL-program which is a promising work based on loose integration and also, there are some prototype reasoners available. We planned to start our work with DL-programs, test it and understand the limitations, and subsequently decide on improvements or explore alternative integration types.

7 Conclusion and Future Work

In the above sections, we outlined our research plan to execute this thesis and provided an overview of our initial results. As a first step, we identified the initial requirements needed for this application domain and identified existing methods to combine LP and DLs. From the list, we chooses one method which is assumed to be sufficient for this application domain by considering the initial requirements. Our choosen framework is DL-Program.

Our next step is to perform evaluations on DL-program concerning identified functional and non-functional requirements. Then recognize the improvement needed for the approach to apply in this context or consider pursuing an alternative approach. Also, illustrate a protocol for knowledge engineers to select either LP or DL while modeling a use case.

Acknowledgments. This research is jointly funded by Thales and National Association of Research and Technology(ANRT),France. I express my sincere thanks to my supervisors: Christophe REY(Clermont Auvergne University), Florence DE GRANCEY(Thales), Victor CHARPENAY(Ecole des Mines de Saint-Ãtienne), and Farouk TOUMANI(Clermont Auvergne University) for their guidance and assistance thus far.

References

1. The Description Logic Handbook: Theory, Implementation and Applications
2. Jaccard index. `https://en.wikipedia.org/wiki/Jaccard_index`
3. et al, M.K.: Local closed world reasoning with description logics under the well-founded semantics. Artificial Intelligence **175**(9), 1528–1554 (2011)
4. et al, T.E.: Combining answer set programming with description logics for the semantic web. Artificial Intelligence **172**(12), 1495–1539 (2008)
5. Bechhofer, S.: Hoolet (2003), `https://owl.man.ac.uk/hoolet/`
6. Benjamin Grosof., e.a.: Description logic programs: combining logic programs with description logic. In: The Web Conference (2003)
7. Bikakis Antonis., e.a.: The dr-prolog tool suite for defeasible reasoning and proof explanation in the semantic web. In: Artificial Intelligence: Theories, Models and Applications. pp. 345–351 (2008)
8. Boris Motik., e.a.: Owl 2 web ontology language profiles. In: W3C Recommendation (11 December 2012), `https://www.w3.org/TR/owl2-profiles/`
9. Boris Motik., e.a.: Reconciling description logics and rules **57**, 30–62 (2010)
10. Crina Grosan., e.a.: Chapter 7 - Rule Based Expert System, pp. 149–181. `https://doi.org/10.1007/978-8-642-21004-4`
11. Drabent odzimierz., e.a.: Hybrid Reasoning with Rules and Ontologies (2009)
12. Drabent WÅodzimierz., e.a.: Hybrid Reasoning with Rules and Ontologies, pp. 1–49 (2009)

13. Eiter Thomas., e.a.: Rules and Ontologies for the Semantic Web, pp. 1–53 (2008). `https://doi.org/10.1007/978-3-540-85658-0_1`
14. Eiter Thomas., e.a.: Rules and ontologies for the semantic web. vol. 5224, pp. 1–53 (2008). `https://doi.org/10.1007/978-3-540-85658-0_1`
15. Evren Sirin., e.a.: Pellet: A practical owl-dl reasoner. Journal of Web Semantics
16. Gunjan Singh., e.a.: Owl2bench: A benchmark for owl 2 reasoners. In: International Workshop on the Semantic Web (2020)
17. Guohui Xiao., e.a.: The drew system for nonmonotonic dl-programs. In: China Semantic Web Symposium (2012)
18. Hawke, S.: Surnia. In: W3c page (2003), `https://www.w3.org/2003/08/surnia/`
19. Ian Horrocks., e.a.: Semantic web rule langauge. In: W3C (21 May 2004)
20. Kifer, M., Lausen, G.: F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In: ACM SIGMOD Conference (1989)
21. Krotzsch, M.: Description logic rules. In: Phd thesis (2010)
22. Lifschitz, V.: Answer set programming. Answer Set Programming (2019), `https://api.semanticscholar.org/CorpusID:61146927`
23. Meditskos., e.a.: Combining a dl reasoner and a rule engine for improving entailment-based owl reasoning. In: The Semantic Web - ISWC 2008
24. Motik Boris., e.a.: Can owl and logic programming live together happily ever after? In: The Semantic Web - ISWC 2006. pp. 501–514
25. NEUHAUS, F.: On the Definition of Ontology, pp. 1–10 (Proceedings of the Joint Ontology Workshops 2017)
26. Rosati, R.: Dl+log: Tight integration of description logics and disjunctive datalog. pp. 68–78 (01 2006)
27. Rosati Riccardo., e.a.: Integrating Ontologies and Rules: Semantic and Computational Issues, pp. 128–151. `https://doi.org/10.1007/11837787_5`
28. Sanja Lukumbuzya., e.a.: Resilient logic programs: Answer set programs challenged by ontologies. In: The Thirty-Fourth Conference on Artificial Intelligence 2020
29. Schulze, J.: Handbook of logic in artificial intelligence and logic programming (2016), `https://api.semanticscholar.org/CorpusID:63249517`
30. Sean Bechhofer., e.a.: Owl 2 web ontology language profiles. In: W3C Recommendation (10 February 2004), `https://www.w3.org/TR/owl-ref/`
31. Shearer Rob., e.a.: Hermit: A highly-efficient owl reasoner. vol. 432 (01 2008)
32. Stijn Heymans., e.a.: Open answer set programming for the semantic web. Journal of Applied Logic
33. Stijn Heymans., e.a.: F-logic: Loosely coupling f-logic rules and ontologies. 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology **1**, 248–255 (2010)
34. Thom W. Frhwirth., e.a.: Constraint logic programming - an informal introduction. In: Logic Programming Summer School (1992)
35. Thomas Eiter., e.a.: Nlp-dl (2007), `https://www.mat.unical.it/ianni/swlp/`
36. Vedran Kasalica., e.a.: Nohr: An overview. Artificial Intelligence pp. 1–7 (2020)
37. WÅodzimierz Drabent., e.a.: Hd-rules: A hybrid system interfacing prolog with dl-reasoners. In: Applications of Logic Programming to the Web (2007)
38. Yevgeny Kazakov ., e.a.: Elk reasoner: Architecture and evaluation. In: International Workshop on OWL Reasoner Evaluation (2012)