Generating Commonsense Ontologies with Answer Set Programming

Stefan Jakob, Alexander Jahl, Harun Baraki and Kurt Geihs
Distributed Systems Department, University of Kassel, Wilhelmshöher Allee 71-73, Kassel, Germany

Keywords: Knowledge Representation and Reasoning, Knowledge-based Systems, Ontologies.

Abstract: The use of commonsense knowledge is essential for the interaction of humans and robots in a smart environment. This need arises from the way humans naturally communicate with each other, in which most details are usually omitted due to common background knowledge. To enable such communication with a robot, it has to be equipped with a commonsense knowledge representation that supports reasoning. Ontologies could be a suitable approach. However, current ontology frameworks lack dynamic adaptability, are monotonous, are missing negation as failure, and are not designed for huge amounts of data. This paper presents a new way to model ontologies based on a non-monotonic reasoning formalism. Our ontology modelling framework, called ARRANGE, allows for the automatic integration of graph-based knowledge sources to generate ontologies and provides corresponding tools. The presented experiments show the applicability of the generated ontologies and the performance of the ontology generation, the ontology reasoning, and the query resolution.

1 INTRODUCTION

Smart environments have become a part of our everyday life. In general, they are distributed IT systems that support humans in their daily chores by providing information via services or by physically supporting them via robots. A typical example is a Smart Home. It contains IoT devices, services either on local Edge devices or in the Cloud, different kinds of robots, and humans interacting with its components. Currently, robots for the Smart Home are mainly built for a single purpose, e. g., lawnmowers or vacuum cleaners. While these robots are particularly suited for their purpose, the focus in the field of service robots is set on the development of multi-purpose robots (Brady et al., 2015). To allow the safe cooperation of services, robots, and humans, key features like communication, knowledge representation and sharing, as well as reasoning, are needed. Since there is a vast amount of existing communication frameworks, the focus of this paper is set to knowledge sharing, knowledge representation, and reasoning. A straightforward solution for knowledge sharing would be the use of a central database in the smart environment. However, this would introduce a single-point-of-failure and a bottleneck to the system. Furthermore, the distance to the central database would increase the latency and could cause network congestions if too many parties access the central point simultaneously. Thus, a central database would not scale and not be suited for large-scale environments like Smart Cities. Therefore, a distributed multi-agent based knowledge registry has been proposed in (Jakob et al., 2020). In brief, the agents organise themselves in a B*-tree. While the inner nodes of the tree manage the structure of the tree, forward queries, and incorporate further nodes, the leaf nodes maintain the knowledge and answer queries. The distributed knowledge registry is based on a non-monotonic reasoning formalism, that supports efficient knowledge representation, storage, and reasoning.

A further challenge in a smart environment in which humans and robots interact is the use of commonsense knowledge and the symbolic representation of the environment. Humans rely on their commonsense knowledge to solve everyday tasks (Davis, 2014) and rely on symbols to abstract their environment. For example, a cup is a symbol for an object, which can hold liquids, is used for drinking, is often made of pottery, and has a handle. Hence, a service robot, which interacts with a human should have access to a source of commonsense knowledge, such as ConceptNet (Speer et al., 2017). These sources represent knowledge as a graph of symbols, that are connected by relations. However, these sources are typically not able to reason about the represented knowledge. We claim that the combination of non-monotonic reasoning with such symbols is particularly suited for communication between robots and humans. A non-monotonic approach allows for the

Jakob, S., Jahl, A., Baraki, H. and Geihs, K.
Generating Commonsense Ontologies with Answer Set Programming.
DOI: 10.5220/0010191905380545
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 538-545
ISBN: 978-989-758-484-8

dynamic adaptation of the knowledge during run-time, which is an important requirement in dynamic environments. For example, a robot has to its knowledge or retract derived knowledge if human provides further information.

The main contribution of this paper is, therefore, a non-monotonic ontology generation method which supports the dynamic incorporation of commonsense knowledge. Furthermore, the resulting ontology can be altered during run-time without any need to rebuild the complete ontology.

Section 2 introduces the non-monotonic ontology generation, which is evaluated in Section 3. Related works are discussed in Section 4. Section 5 summarises the paper and presents future work.

2 ASP ONTOLOGY MODELLING

This section presents the main contribution of this paper, which is the automatic generation of commonsense ontologies with Answer Set Programming (ASP) (Gelfond and Kahl, 2014). Section 2.1 shows the first step, namely the extraction of the commonsense knowledge from a hypergraph-based knowledge source like ConceptNet 5 (CN5) (Speer et al., 2017) and the translation into basic ontology rules. This is followed by a description of the ontology inference rules in Section 2.2 and the characterization of facets in Section 2.3. A graphical tool to adjust the resulting ontology is discussed in Section 2.4.

2.1 Ontology Extraction

The first step in the automatic generation of the commonsense knowledge is the extraction and translation of the knowledge from a hypergraph-based knowledge source. The general approach is to provide a root concept for the ontology and to traverse the hypergraph until no further edges adhere to a given set of stopping criteria. Algorithm 1 presents this automatic extraction and translation.

This algorithm receives a root concept c_r, a set of ontology relations R_o, a set of relations that represent synonyms R_s, a set of relations denoting properties R_p, and stopping criteria SC as input. In the first step, an adapted breadth-first search (BFS) is applied. During this step, the BFS starts at c_r and selects edges, which are annotated with the relations given in R_o and adhere to the stopping criterion SC_o. For example, a taxonomy could be created by only using the `IsA` relation of CN5 and relying on the edge weight of CN5 as a stopping criterion. By adjusting SC_o, edges could

Algorithm 1: Ontology Extraction.

Input : Root Concept c_r,
Set of Ontology Relations R_o,
Set of Synonymic Relations R_s,
Set of Property Relations R_p,
Set of Stopping Criteria SC
Output: ASP Commonsense Ontology o_{asp}

1 $<C, E>$:= adaptedBreadthFirstSearch(c_r, R_o, SC_o)
2 $E := E \cup$ getSynonyms(C, R_s, SC_s)
3 $E := E \cup$ getProperties(C, R_p, SC_p)
4 o_{asp} := translate(C, E)

be included into or excluded from the taxonomy. After the BFS, sets of all encountered concepts C and edges E are returned. Since synonyms are an important part of natural language, synonyms for all concepts in C are determined and the resulting edges that adhere to SC_s are added to E. One example is shown in Figure 1. `Pup` is a synonym for `puppy` and hence added in this step. In the third step, the properties of each concept are extracted using the relations given in R_p. After these steps, the gathered edges in E and concepts in C are translated into ASP.

```
1 #external cs_isA("dog","pet",136).
2 weight(136,668,0) :- cs_isA("dog",
    "pet",136).
3 #external cs_isA("puppy","dog",139).
4 weight(139,568,0) :- cs_isA("puppy",
    "dog",139).
```

Listing 1: Excerpt from the Commonsense Ontology.

Listing 1 shows an excerpt from the edges in Figure 1 translated into ASP. Each edge is represented by an External Statement and an auxiliary rule. External Statements are a feature provided by the ASP solver Clingo (Gebser et al., 2014), which enables the dynamic adaptation of the truth value of a predicate. Line 1 expresses that it is commonsense (`cs_`) that a `dog` is a `pet`. The External Statement contains a unique identifier, which is used to link it with the head of the auxiliary rule. This rule is used to represent the dynamically adaptable weight, for example, the edge weight in the case of CN5 since it can be seen as the reliability of the knowledge represented by the edge. As long as the External Statement is set to true, the weight predicate in the head can be derived. The weight predicate has three values. The first is the unique id. The second is the weight multiplied with 100 since ASP can only handle integer values. The last value is a timestamp, which will be used in the ontology inference rules presented in Section 2.2. If the External Statement is set to false, the weight and

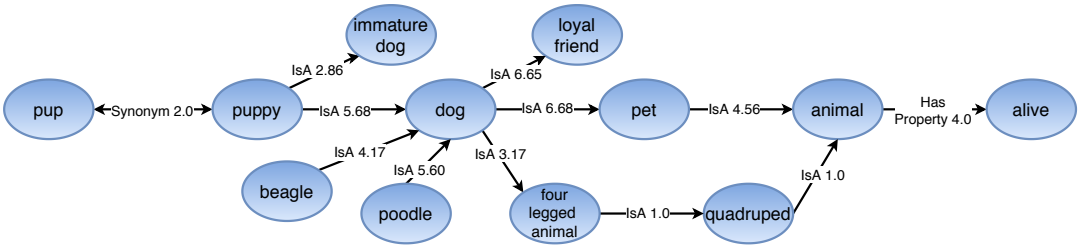

Figure 1: Excerpt from the CN5 Hypergraph.

the External itself will not be derived in the next solving step.

2.2 Ontology Inference Rules

After the generation of ASP rules representing the commonsense knowledge, inference rules are needed to create an ontology and allow the application of the commonsense knowledge to individuals. These inference rules are summarized in a second ASP program. It consists of six major parts: weight handling, commonsense propagation, determination of subsets, classification, facet handling, and Answer Set size reduction. Due to limited space, only excerpts from the weight handling, the commonsense propagation, classification, and Answer Set size reduction are shown.

Listing 2 presents an excerpt from these inference rules. Line 1 is used to determine the current weight for each translated edge based on the maximum timestamp (`MaxTS`). This enables the dynamical adaption of edge weights, for example, by human interaction. The use of a unique id allows mapping the resulting predicates to the corresponding edges. Line 2 and 3 are used to propagate commonsense knowledge. In Line 2, the predicate `is` is used to mark an initial classification of an individual (`Ind`), e. g., given by a human user or an image classification. If the initial classification fits a commonsense knowledge predicate via the `FromC` variable, further classifications can be derived. For example, considering the initial classification `is("rex","puppy")` and that it is commonsense knowledge that a `puppy` is a `dog` (Figure 1), it can be derived that `rex` is a `dog`, too. Line 3 shows further commonsense propagation, which is achieved by using already derived propagations and further commonsense knowledge. To stop this propagation, the stopping criteria are used. For example, the edge weights extracted from CN5. Since these weights represent the reliability of the edge, only those edges with an increasing weight one should be used for the propagation to achieve reliable results. Following lower or equal weights could result in an exhaustive traversal of the hypergraph and, thus, an impractical or cyclic classification. Line 4 and 5 are used to represent the final classification in a human and machine-readable format. Therefore, the rules like the one presented in Line 4 create an internal representation of the classification. Based on this internal classification, the rule in Line 5 derives the final classification annotated with the highest weight for each classification. The last part of this excerpt is the reduction of the Answer Set size. Therefore, the `#show` directive of Clingo is used, which limits its output to the respective predicates with the given arity. Thus, Line 6 reduces the output to all `classifiedAs` predicates, which have the arity three. Applying these rules on the commonsense knowledge presented in Figure 1 and the initial classification `is("rex","puppy")` will result in the following classification: `rex` is a `puppy`, a `dog`, a `loyal_friend`, and a `pet`.

```
1 currentWeight(Id,Weight,MaxTS) :-
      MaxTS = #max{TimeStep :
      weight(Id,_,TimeStep)},
      weight(Id,Weight,MaxTS).
2 isA(Ind,ToC,Weight):- is(Ind,FromC),
      cs_isA(FromC,ToC,Id),
      currentWeight(Id,Weight,MaxTS).
3 isA(FromC,ToC,Weight2):- isA(FromC,
      InterC,Weight1), cs_isA(InterC,
      ToC,Id), currentWeight(Id,
      Weight2,MaxTS),Weight1<Weight2.
4 classifiedAsInternal(FromC,ToC,
      MaxWeight) :- MaxWeight = #max{
      Weight : isA(FromC,ToC,Weight)},
      isA(FromC,ToC,_),is(FromC,_).
5 classifiedAs(FromC,ToC,MaxWeight):-
      MaxWeight = #max{ Weight :
      classifiedAsInternal(FromC,ToC,
      Weight)}, classifiedAsInternal(
      FromC,ToC,_).
6 #show classifiedAs/3.
```

Listing 2: Excerpt from the Inference Rules.

2.3 Facets

Facets are an important part during the creation of an ontology since they allow the definition of types, values, ranges, sub-properties, and domains. Hence, they provide the tools to further describe concepts and

their properties given in an ontology. To demonstrate the creation of facets, the ontology shown in Figure 1 is expanded with the edge stating that a `dog` has the property `coat_colour`. An example set of facets for the property `coat_colour` is shown in Listing 3.

```
1 #external facetOf("colour",
      "coat_colour").
2 #external typeOf("colour",
      "coat_colour","string").
3 #external valueRangeOf("colour",
      "coat_colour","0{black;grey;
      brown;white;brindle}1").
4 propertyViolation(Individual,
      "colour","coat_colour", "Too
      many Values") :- X = #count{
      Value : hasValue(Individual,"
      colour","coat_colour",Value,_)},
      hasValue(Individual,"colour","
      coat_colour",_,_), X > 1,
      valueRangeOf("colour",
      "coat_colour","0{black;grey;
      brown;white;brindle}1").
5 #external hasValue("rex","colour",
      "coat_colour","brown",0).
```

Listing 3: Example Facets.

Line 1 of this example is the actual definition of the facet. In this case, the property `coat_colour` has the facet `colour`, which has the type `string` as defined in Line 2. The range of this value can be restricted. This is achieved by adding Line 3 and the auxiliary rule in Line 4. Informally speaking, this External Statement defines, that the facet `colour` has to have at least `0` values and at max `1` value for this facet. Either a set of values or a range can be defined to further restrict the facet. The `colour` is restricted to the values inside the curly brackets. The auxiliary rule in Line 4 is used to check if the minimum and maximum cardinality of the values are satisfied by counting the respective values and comparing the count with the cardinalities. A restriction to the values is granted by a graphical user interface, which is presented in Section 2.4. The value of the `coat_colour` of `rex` is given alongside a timestamp in Line 5. Due to limited space, sub-property and domain facets are omitted in this example.

2.4 Graphical User Interface

To ease the generation and usage of an ASP ontology, ARRANGE (Answer set pRogRAmming oNtoloGy gEneration) provides a graphical user interface (GUI) as shown in Figure 2. It supports the automatic extraction of a commonsense knowledge ontology (Section 2.1), generates ontology inference rules (Section 2.2), and supports the creation of facets (Section 2.3).

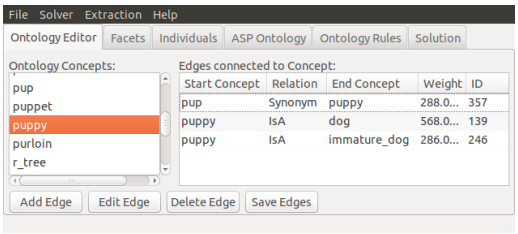

Figure 2: Graphical User Interface.

On the left side, a list of all concepts that are part of the ontology is given. By selecting one of these concepts, all edges connected to it are displayed. Here, new edges can be added, existing ones can be edited or deleted. Tabs for facets and individuals are available. The last three tabs provide an overview of the current ASP ontology, the inference rules, and the results of a solving step, which is available via the solver menu. The output of the solver is twofold. The first output is the Answer Set, which is a set of predicates and can be used for further reasoning processes. It contains all derivable facets, subset relations, and classifications. Using the example presented in the previous sections, the Answer Set includes, amongst others, the predicates `hasPropertyValue("rex","coat_colour", "brown")` and `classifiedAs("rex","pet",668)`. The second output is a graph representation of the Answer Set, which provides an overview of its predicates. The graph representing the Answer Set of the example used in the previous sections is shown in Figure 3. Blue arrows indicate facets, black arrows properties and green arrows subsets. Red arrows denote the classification, for example, `rex` is classified as a `dog` with a weight of 568.

3 EXPERIMENTS

The evaluation has been conducted on a Lenovo workstation equipped with an Intel®Core™ i7-7500U @ 2.70 GHz Dual-Core processor and 16 GB DDR4-2133 RAM running Ubuntu 18.04.4 with kernel version 4.15.0-112-generic. ConceptNet 5.7 and Clingo version 5.3.1 with gringo 5.3.1 and clasp 3.3.4 were used.

3.1 Ontology Extraction and Reasoning

The selection of the root concept as well as the set of stopping criteria (minimal weights when using CN5) applied in Algorithm 1 determine the size of the resulting ontology and the run time of its generation. Therefore, this section analyses three different

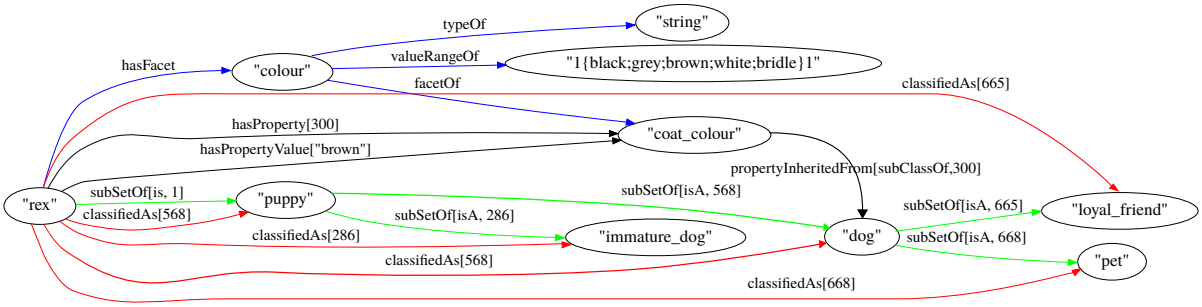

Figure 3: Graph Representation of the Classification.

root concepts originating from three different areas: `animal`, `person`, and `thing`. Additionally, different minimal weight sets are used.

The smallest ontologies have been extracted for a minimal weight of `2.5`. With CN5 as a knowledge source, this means that at least two verified sources were the origin of the edge. Furthermore, only the minimal weight for the `isA` relation is changed since it has the highest number of edges. The remaining relations have a fixed weight of `2.0`. The selected root concept has a big influence on the size of the extracted ontology since it determines in which area of the hypergraph the extraction starts. The concept `animal` has the highest number of edges (522), the concept `thing` has 196 edges, and `person` the lowest number of edges (17). Lowering the minimal weight has a major impact on the size of the ontology since more edges can be added. Hence, extensive parts of the hypergraph can be extracted. A minimal weight of `2.0` results in an identical size for all tested root concepts (95353 edges). Lowering the minimal weight even further results in an additional expansion of the ontology (201148 edges). Higher minimal weights are suited to create domain-specific ontologies while lower minimal weights will result in more general commonsense knowledge ontologies.

Besides the size of the ontology, the selection of the root concept, and the minimal weight affect the run time of the ontology generation. Again, the weight for the `isA` relation is adapted, while the remaining weights are fixed at `2.0`. The tests have been executed 20 times and the average has been determined. The average run time of the ontology generation mainly depends on the selected minimal weights. While small ontologies are generated in a few seconds (`animal` 142 s, `person` 4 s, `thing` 51 s), the generation for the medium size requires 8097 s for `animal`, 8199 s for `person`, and 8142 s `thing`. The generation of the largest ontologies takes roughly four hours (`animal` 13506 s, `person` 13341 s, `thing` 13134 s). Considering the sizes of these ontologies, the resulting run times are acceptable and show the viability of our approach. On average, the standard deviation of the measured run times is below 1 %. The generation of the ontology for the root concept `person` with a minimal weight of `2.5` has a standard deviation of 4 %, which is caused by its low run time.

The reasoning process of the presented framework is divided into seven distinct steps. These include the adding and grounding of the ontology edges, which are modelled as External Statements. Since Clingo assumes External Statements as false after they are added, they are set to true in the third step. Finally, the ontology is solved. The queries are ASP programs themselves, hence, they have to be added, grounded, and solved, too. In contrast to the ontology, they do not use External Statements. Table 1 presents the experimental results for different ontology sizes. The query used is the classification of `rex` as discussed in Section 2.2. Again, the values are averaged over 20 measurements.

The lowest run times were measured for a minimal weight of `2.5`. Lowering the minimal weight will result in larger ontologies as explained above. The measured run times increase linearly with the size of the measured ontologies, hence, the proposed ontology representation is scalable. Adding and grounding of the ontology have the highest impact on the run time of the ontology reasoning. This is caused by the number of ASP rules, which have to be considered. The run time of the External Statement assignment is almost neglectable since no reasoning is needed. The final solving step of the ontology reasoning has a low run time since no complex rules have to be considered during the ontology reasoning. In general, the run time of a query is lower than the run time of the ontology reasoning. This is caused by the number of rules that have to be considered in both processes. While the number of rules increases with the selected minimal weight and root concept, the number of rules for a query is fixed. Thus, the run time for adding a query is not influenced by the ontology size and is almost identical for all tested combinations. The grounding and solving steps of a query are affected by the size

Table 1: Run Time of Ontology Solving and Classification Query in [ms].

Min. Weight	2.5			2.0			1.0		
Concept	animal	person	thing	animal	person	thing	animal	person	thing
Add Ont.	98.7	9.3	47.6	15730.4	15616.4	15716.7	36300	37003	37020
Ground Ont.	78.9	11.4	30.5	14800.2	14679.5	14769.4	31525	31721.1	31823
Assign Ext.	1.4	0.15	0.6	256.5	247.7	262.9	523.9	520.9	521.7
Solve Ont.	3.9	0.5	1.5	1010.8	1000.3	1005.2	2257.5	2272.3	2271.6
Add Q.	11.6	13.5	11	11.8	11.7	11.8	12.7	13.1	13.1
Ground Q.	78.5	28.5	46.3	9299.6	9331.4	9350.1	19320.3	19566.8	19525.8
Solve Q.	3.5	0.3	1.1	743.1	735.8	740.2	1635.6	1650.1	1637.2

of the ontology. Again, their run times scale linearly with the size of the ontology. On average, the standard deviation of all measured run times is lower than 1 % for the largest ontologies and increase to roughly 2.5 % for the smaller ontologies, which is caused by the low run times. The only outlier is the standard deviation of query solving for the concept `Person` and a minimal weight of `2.5`. Since the measured run time is extremely low and thus can be influenced by any process running on the test system, the standard deviation is roughly 7 %.

3.2 Comparison to OWL

Let us compare the run time of OWL to the run time of an ASP ontology created by ARRANGE. We translate the well-known Pizza Ontology[1] into ASP. HermiT with version 1.4.3.456 and Protege 5.5.0 are used for OWL ontology reasoning. An individual with the class `margherita` is created in both formalisms and a classification query is applied. In both cases, new solver instances have been used for each measurement. The shown results are the average of 20 measurements. The ontology reasoning took 82.15 ms and query resolution 66.01 ms for the created ASP ontologies. In comparison, HermiT required 588.15 ms for the reasoning step and 154.55 ms for the query resolution. Hence, the reasoning process of the ASP ontology is roughly seven times faster than the OWL reasoning. On the one hand, this is caused by the underlying reasoning technique. While SAT-solvers, as used in ASP, provide fast results, the tableau algorithm of OWL is generally slower but provides better support in case of errors. The ASP query is twice as fast as the OWL description logic query. Both include the classification of an individual, as well as, all super- and sub-classes.

OWL is the de facto standard for knowledge representation in the Semantic Web and is widely used to model ontologies. OWL in its full specification is based on first-order logic and undecidable. Thus, most applications rely on one of the decidable subsets such as OWL DL. The different specifications of OWL have in common that no unique names are assumed, that the resulting ontologies are monotonous, and, that the open-world assumption holds. Since the unique name assumption is not given in OWL, distinct names can be given to one individual. Monotonous reasoning prevents the loss of already derived knowledge; thus, subsequent adaptations of the ontology may not contradict derived knowledge. The open-world assumption restricts the reasoning to statements that are explicitly modelled in the ontology. Default assumptions that may be overwritten by derived knowledge cannot be defined.

In contrast to OWL, ASP adheres to the unique name assumption. On the one hand, this prevents to model distinct individuals with the same name. On the other hand, it reduces the overall modelling effort, since explicitly stating that all occurrences of a name refer to the same individual is unnecessary. Additionally, ASP provides non-monotonous reasoning and supports the closed world assumption enabling the definition of defaults which is particularly useful when modelling commonsense knowledge. Finally, state-of-the-art ASP solvers like Clingo provide mechanisms to adapt an ontology dynamically and to enumerate all possible solutions. Hence, the usage of ASP is suited to model and reason about commonsense knowledge.

3.3 Applicability

The application of the ontologies extracted from CN5 provides good classifications as depicted in Figure 3. However, considering the excerpt from the CN5 hypergraph shown in Figure 1, the classification is missing the classes, e. g., `animal`. This is caused by the classification rules presented in Section 2.2, since they consider edges with increasing weight to prevent impractical classifications. To tackle this issue, ARRANGE provides two ways. The first is the adap-

[1] https://protege.stanford.edu/ontologies/pizza/pizza.owl (December 3, 2020)

tation of the corresponding weight with its GUI. It provides an overview of all edges that are part of the ontology and enables the adaptation of all weights during design time. The second option is applied by adding an additional rule that adjusts the weight of the corresponding edge. This rule has to be modelled like the Lines 2 or 4 of Listing 1 and has to have a higher timestamp than the current weight rule. For example, the weight of the edge between `pet` and `animal` shown in Figure 1 could be set to `7.0`. Subsequently, `rex` is classified as an `animal` and inherits the property `alive`.

4 RELATED WORK

The presented related work is divided into two main categories. The first one focusses on the extraction of ontologies from existing knowledge sources, while the second category focusses on ASP as ontology language and reflects upon tool support for the generation of ontologies. The general idea of an ontology is to provide a standardised terminology to represent knowledge which supports the collaboration of different parties. In comparison to other knowledge sources like databases, data warehouses, and knowledge graphs, ontologies support the definition of properties, the restriction of values, arbitrary logic constraints, and automatic reasoning. Furthermore, ontologies are declarative and do not rely on customized interpretations. They provide formal axioms and well-defined semantics, which is essential for the interaction of agents in heterogeneous and human-populated environments. However, the aforementioned knowledge sources can serve as a seed for the creation of an ontology, which is a reason for extensive research on automatic and semi-automatic converters.

A common approach to store and manage data is the use of databases or data warehouses. Databases and data warehouses contain tables, which consist of named columns and store data points in the corresponding rows. Columns of different tables can refer to each other and, thus, establishing relations between them. Relying on this scheme, ontologies can be generated. There are numerous approaches that realise this methodology using relational SQL databases and data warehouses (Zhou et al., 2010; Kiong et al., 2009; Al Khuzayem and McBRIEN, 2016; da Silva et al., 2016; El Idrissi et al., 2013). In general, these works build upon a set of predefined rules which are used to transform a database scheme into an ontology. This includes the translation of table names into ontology classes, field or column names into properties, and rows into individuals. Furthermore, semantics are added to the ontology. For example, Kiong et al. (Kiong et al., 2009) mark, amongst others, bridge tables, reference tables, and reference fields. Zhou et al. (Zhou et al., 2010) identify these references automatically and map them to the corresponding relations. Additionally, they set the cardinality of unique, nullable, and not-nullable properties accordingly. In (Al Khuzayem and McBRIEN, 2016), further features such as subclass relations and symmetric and reflexive properties are added.

Although the presented approaches achieve good results, the expressive power of the resulting ontologies and the flexibility of their methodology is limited. Databases usually comprise a small number of tables and, thus, only a few classes are generated. Hence, the number of resulting relations is restricted to a few references between tables, if they are explicitly stated. Furthermore, the generated ontology strongly depends on the design decisions for the database and will mainly consist of individuals instead of classes. ARRANGE, in comparison, generates ontologies consisting of a vast amount of classes based on a given hypergraph.

As opposed to these approaches, ARRANGE automatically extracts a commonsense ontology from a given hypergraph. Similar to OntoHarvester (Mousavi et al., 2013), it relies on a given seed class to start the ontology extraction but is able to derive subclasses based on the structure of the used hypergraph. ARRANGE supports the manual adaption of ontology parts during the design time and the run time of the ontology. Furthermore, ARRANGE enables reasoning, since it utilises ASP.

In general, numerous approaches exist that extract knowledge graphs, which are hypergraphs, from a given free text or from semi-structured knowledge (Paulheim, 2017). Many of these may be refined by crowds-based approaches or experts such as in case of CN5.

However, there is no axiomatic semantics and no reasoning support. Thus, it is beneficial to transfer the hypergraph to an appropriate knowledge representation. In (Krötzsch, 2017), Krötzsch lists requirements for such a knowledge representation. One of them is the existence of negation, which, for example, enables the definition of a class based on the absence of information. Description logics such as OWL do not provide negation as failure, are monotonic, and adhere to the open-world assumption. Krötzsch emphasises that description logics do not sufficiently address these requirements and recommends a declarative symbolic knowledge representation in the context of a computation paradigm instead of pure represen-

tation (Krötzsch, 2017). ASP with its non-monotonic reasoning capabilities, different kinds of negation, support of the closed world assumption, and its symbolic representation is a suitable formalism to tackle these issues. Hence, ARRANGE uses ASP to represent commonsense knowledge as ontologies, to support reasoning, and declarative programming.

OntoDLV (Ricca et al., 2009) uses an extension to basic ASP (OntoDLP) to model ontologies. For example, classes are declared by expanding predicate names with the key phrase `class`. OntoDLP supports the definition of individuals, relations, modules, and the creation of lists and sets. Besides these constructs, OntoDLP allows to model taxonomies by adding the keyword `isa` enabling the generation of a class based on inheritance and a set of attributes. Furthermore, OntoDLV provides a graphical modelling tool to support the creation of an ontology and allows the incorporation of OWL atoms.

In contrast to OntoDLV, ARRANGE does not rely on an extended version of ASP and uses the ASP-Core-2 standard. Additionally, External Statements provided by Clingo are used to create an ontology, which can be dynamically altered during run time.

5 CONCLUSIONS

In this paper, we have presented a framework to automatically extract ontologies from a hypergraph-based knowledge source like CN5. The resulting ontologies are formulated using the non-monotonic reasoning formalism ASP that supports dynamic adaptations of the ontology during run-time and the definition of defaults. The presented experiments proved that the combination of ARRANGE[2] with the commonsense knowledge source CN5 results in an adaptable and extensive commonsense knowledge ontology. The generation process itself is configurable and allows to extract different parts of the hypergraph.

Due to the size of the resulting ontologies, we plan in the future work to create an efficient distributed access and automatic distribution of individuals based on the ontology using the distributed and multi-agent-based knowledge management presented in (Jakob et al., 2020). This knowledge management will be evaluated in a search and rescue scenario, which incorporates several heterogeneous robots and UAVs. Furthermore, we extend the comparison to OWL by translating further ontologies.

[2]https://bitbucket.org/sjakob872/arrange/src/master/, (December 3, 2020).

REFERENCES

Al Khuzayem, L. and McBRIEN, P. (2016). OWLRel: Learning Rich Ontologies from Relational Databases. *Baltic Journal of Modern Computing*, 4(3):466.

Brady, G., Sterritt, R., and George, W. (2015). Mobile Robots and Autonomic Ambient Assisted Living. *Paladyn: Journal of Behavioral Robotics*, 6.

da Silva, T. O., Baião, F. A., and Revoredo, K. (2016). OntoDW: An Approach for Extraction of Conceptualizations from Data Warehouses. In *ONTOBRAS*, pages 83–94.

Davis, E. (2014). *Representations of Commonsense Knowledge*. Morgan Kaufmann.

El Idrissi, B., Baïna, S., and Baïna, K. (2013). Automatic Generation of Ontology from Data Models: A Practical Evaluation of Existing Approaches. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12. IEEE.

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2014). Clingo=ASP+Control: Extended Report. Technical report, Knowledge Processing and Information Systems.

Gelfond, M. and Kahl, Y. (2014). *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, Cambridge, USA.

Jakob, S., Jahl, A., Baraki, H., and Geihs, K. (2020). A Self-Organizing Multi-Agent Knowledge Base. Accepted for publication at IEEE ICWS 2020.

Kiong, Y. C., Palaniappan, S., and Yahaya, N. A. (2009). Health Ontology Generator: Design And Implementation. *IJCSNS*, 9(2):104.

Krötzsch, M. (2017). Ontologies for Knowledge Graphs? In *30th Int. Workshop on Description Logics*, volume 1879. CEUR-WS.org.

Mousavi, H., Kerr, D., Iseli, M., and Zaniolo, C. (2013). Ontoharvester: An Unsupervised Ontology Generator from Free Text. Technical report, InCSD Technical-Report 130003, UCLA.

Paulheim, H. (2017). Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic web*, 8(3):489–508.

Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., and Leone, N. (2009). OntoDLV: An ASP-based System for Enterprise Ontologies. *Journal of Logic and Computation*, 19(4):643–670.

Speer, R., Chin, J., and Havasi, C. (2017). ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 4444–4451.

Zhou, S., Ling, H., Han, M., and Zhang, H. (2010). Ontology Generator from Relational Database Based on Jena. *Computer and Information Science*, 3(2):263–267.