Default Reasoning on Top of Ontologies with dl-Programs

DAO Tran Minh and Thomas Eiter (Faculty Mentor)
Institute for Information Systems, Vienna University of Technology, Austria
Email: {dao,eiter}@kr.tuwien.ac.at

Abstract — *We study the usefulness of description logic programs (dl-programs) in implementing Reiter's default logic on top of a description logic knowledge base (DL-KB). To this end, we investigate transformations from default theories to dl-programs based on different established algorithms for computing default theory extensions, namely the select-defaults-and-check and the select-justifications-and-check algorithms. In each transformation, additional constraints are exploited to prune the search space based on conclusion-conclusion or conclusion-justification relations. The implementation was deployed as a new component in the dl-plugin for dlvhex, and evaluated with various experimental test ontologies, which showed promising results.*

I. Introduction

Default logic [1] has been one of the most prominent nonmonotonic reasoning formalisms due to its close relationship to common-sense reasoning. Ontologies are a key component of the future Semantic Web and reasoning on top also needs to handle default information. However, nonmonotonic reasoning with an ontology alone is not possible because description logics (DLs) which are used as ontology languages are monotonic. Therefore, incorporating default reasoning into ontologies is an interesting topic to take into account. The first attempt in this field was proposed in [2] which imposes some restrictions regarding decidability. Moreover, there has not been any implementation or further development on this approach since then. One of the reasons was the lack of support from research on integrating rules and ontologies at that time [3].

In this work, we investigate the problem of enabling default reasoning on top of ontologies specified by DL-KBs, based on an available mechanism that allows for integrating rules and ontologies, namely dl-programs [4]. In addition to our theoretical results, we provide a front-end as a new component in the dl-plugin [5] for the HEX-program solver dlvhex[1] [6], which implements dl-programs.

II. Embedding Defaults over Description Logics into dl-Programs

In [7], we present three transformations to embed default reasoning into dl-programs. Due to space constraints, we present here only one, viz. Υ, which we illustrate in a simple example rather than in full details.

[1] http://www.kr.tuwien.ac.at/research/systems/dlvhex/

Assume that we have a small DL-KB about birds and penguins:

$$L = \left\{ \begin{array}{c} Flier \sqsubseteq \neg NonFlier, Penguin \sqsubseteq Bird, \\ Penguin \sqsubseteq NonFlier, Bird(t) \end{array} \right\}$$

and a set of defaults:

$$D = \left\{ \frac{Bird(X) : Flier(X)}{Flier(X)} \right\}$$

Intuitively, we have a DL-KB in which flying and non-flying objects are distinguished. We know that penguins, which are birds, do not fly. We also know that birds normally fly, but this fact cannot be added into L because it will make L inconsistent. Having a bird named Tweety, t for short, and no more information, we would like to conclude that t flies; and later if we learn that t is a penguin, the opposite conclusion should hold. To make this possible, we add a set of defaults D as about on top of L. We call $\Delta = \langle L, D \rangle$ a *default theory over a DL-KB*. Through Υ, Δ is transformed to a dl-program $KB^{df}_{\Upsilon}(\Delta) = (L, P)$ where P consists of the following dl-rules:

```
cons_Flier(X) :-
  dom(X), not out_cons_Flier(X).
out_cons_Flier(X) :-
  dom(X), not cons_Flier(X).
in_Flier(X) :- cons_Flier(X),
  dom(X), DL[Flier+=in_Flier;Bird](X).
:- cons_Flier(X),
  DL[Flier+=in_Flier;-Flier](X).
:- not DL[Flier+=in_Flier;-Flier](X),
  out_cons_Flier(X).
dom(t).
```

These rules can be explained as follows: the first two guess if assuming an object can fly is consistent or not with the answer set, i.e., the agents' belief set. The third rule aims to find which objects can fly under the consistency assumption above, where the atom `DL[...]` incorporates the assumption on Fliers. The last two rules are two constraints: the first prevents cases in which we guessed that an object can fly is consistent to the answer set but in fact we conclude that it does not fly. Similarly, the second constraint kills all answer sets in which we guessed that an object cannot fly is consistent with the answer set but then we could not conclude that it does not fly.

With this dl-program, we have the desired answer set $I = \{dom(t), cons\_Flier(t), in\_Flier(t)\}$ which en-

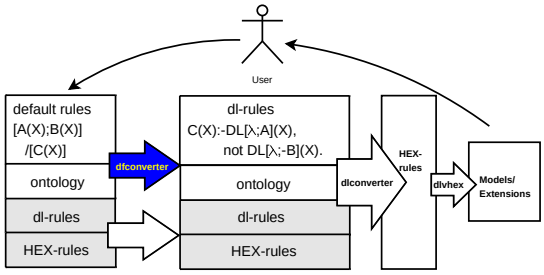

Figure 1: Strategy for implementing the df-converter

tails that t flies by default. If we now change to a new DL-KB $L' = L \cup \{Penguin(t)\}$, then from the dl-program $KB^{df}_{\Upsilon}(\Delta') = (L', P)$, t can no longer be inferred to be able to fly, and the corresponding answer set is $I' = \{dom(t), out\_cons\_Flier(t)\}$.

The other two transformations Π and Ω, the proof of correctness, and an optimizing technique using pruning rules are presented in [7].

III. Front-end Overview

The architecture of the front-end is shown in Figure 1. Users provide input, including a set of default rules along with a DL-KB in an OWL file, and then get the extensions of the default theory in terms of answer sets. Other optional inputs can be either dl-rules or low-level HEX-rules. Recall that dlvhex is a solver for HEX-programs. It receives input in terms of HEX-rules and returns results as answer sets to the user. As dlvhex has a dl-plugin, dl-programs are converted transparently into HEX-programs, the ordinary user is unaware of HEX-rules. However, this does not prevent experts from providing more sophisticated input such as constraints in terms of dl-rules or HEX-rules to reduce the search space and speed up the evaluation. In fact, we do provide a technique in which the user can specify more supportive information, i.e., typing predicates that helps our implementation gaining significant performance improvement. Details on this technique can be found in [7].

Utilizing the dl-plugin, we implement a df-converter whose input contains default rules accompanied with a DL-KB, optional dl-rules, or even HEX-rules. This converter transforms all the default rules into dl-rules based on different transformations, with the help of the DL-KB serving as the sources of individuals for the domain predicates to guarantee the safety condition [6]. Then, the transformed dl-rules, along with other input dl- and HEX-rules, are transferred to the dl-converter. The rest of the evaluation will be done by the dl-plugin and dlvhex. For more details on the implementation of the df-converter, we refer to the Master's Thesis [7] on which this abstract is based. The thesis also contains experimental results, and comparisons between different transformations, which reveals interesting tasks to improve the overall performance of the system.

IV. Future Work

Concerning our implementation of the front-end, some issues remain for future work. Firstly, we would like to investigate more sophisticated pruning rules depending on the structure of the default theory. Secondly, a closer look into particular kinds of default theories such as normal or semi-normal default should help to find more effective transformations.

As our work depends on dlvhex and the dl-plugin, the experimental results suggest the following tasks to increase the system performance.

A caching technique which is only available now for dl-atoms would give additional benefit if it can be implemented for cq-programs [8], an extended version of dl-programs, and already be deployed in the dl-plugin.

dlvhex is now using RacerPro as its only DL-reasoner. It would also be interesting to look at other possibilities interfacing dlvhex with different DL-reasoners such as KAON2 or Pellet, and then compare the results.

Currently, dlvhex takes all grounded dl-programs and computes the answer sets. Another challenging task would be to automatically classify the input and do only necessary rules grounding for a smaller search space.

References

[1] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

[2] F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Autom. Reasoning*, 14(1):149–180, 1995.

[3] G. Antoniou et al., Combining Rules and Ontologies: A survey. Technical Report IST506779/Linköping/I3-D3/D/PU/a1, Linköping University, Feb 2005.

[4] T. Eiter, G. Ianni, Th. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12-13):1495–1539, 2008.

[5] T. Krennwallner. Integration of Conjunctive Queries over Description Logics into HEX-Programs. Master's thesis, TU Wien, Oct 2007.

[6] R. Schindlauer. *Answer-Set Programming for the Semantic Web*. PhD thesis, TU Wien, Dec 2006.

[7] M. T. DAO. Default Reasoning on Top of Ontologies with dl-Programs. Master's thesis, TU Wien, Jun 2008.

[8] T. Eiter, G. Ianni, T. Krennwallner, R. Schindlauer. Exploiting Conjunctive Queries in Description Logic Programs. Annals of Mathematics and Artificial Intelligence. 2008.