A Short Report on Issues with DL-Learner

Tomáš Bisták

March 17, 2023

Introduction

In this document, we briefly discuss several issues with the latest implementation of DL-Learner we had to tackle throughout our experimentation. We also present our solutions to the problems outlined below. The updated version of DL-Learner to which we later refer is available on our GitHub[1] (the branch `v3`). Besides the changes described in the following text, we have made a couple of other modifications that we consider improvements rather than fixes, and for this reason we decided not deal with them in this report. Nevertheless, all these improvements are already included in the implementation provided in the branch `v3`.

1 The Extended Definition of ρ

After noticing some inexplicable differences in the accuracy of two logically equivalent descriptions reported by *OCEL*, we found out that extending the built-in ρ refinement operator according to the definition proposed in [1], Section 5.4, may lead to inconsistencies between the expected and the actual properties of the operator. More specifically, we think that the operation

$$\leq n\ r.D \rightsquigarrow \leq n\ r.\rho_A(D),$$

for $A = ar(r)$, any role r, concept D, and $n \in \mathbb{N}$[2], does not produce downward refinements. Moreover, we believe that in reality, this operation can be classified as an upward refinement operation if we assume that $\rho_A(D) = E$ for some $E \sqsubseteq D$. To support our claims, we also provide a proof of the latter.

[1] `https://github.com/mousetom-sk/DL-Learner`

[2] Here, we assume that zero is an element of $\mathbb{N}$ as well.

Proof. Let $\mathcal{L}$ be a description language which allows qualified number restrictions and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$. In order to prove that the operation in question is an upward refinement operation, it is now sufficient to show that for all concepts $D, E \in \mathbf{C}_\mathcal{L}$ such that $E \sqsubseteq D$, all roles $r \in \mathbf{R}_\mathcal{L}$, and any $n \in \mathbb{N}$, the proposition $\leq n\ r.D \sqsubseteq \leq n\ r.E$ holds, i.e. that for every model $\mathcal{I}$[3] of $\mathcal{K}$ it is also true that

$$\mathcal{I} \models \leq n\ r.D \sqsubseteq \leq n\ r.E. \tag{1}$$

Therefore, let D and E be arbitrary concepts in $\mathbf{C}_\mathcal{L}$ with $E \sqsubseteq D$, let $r \in \mathbf{R}_\mathcal{L}$ be an arbitrary role, and let n be a natural number. Let us further take an arbitrary model $\mathcal{I}$ of $\mathcal{K}$. As we aim to prove (1) which holds if and only if $(\leq n\ r.D)^\mathcal{I} \subseteq (\leq n\ r.E)^\mathcal{I}$, we first construct these sets based on the definition of the interpretation function[4]:

$$(\leq n\ r.D)^\mathcal{I} = \{a \in \Delta^\mathcal{I} \mid |\{b \in \Delta^\mathcal{I} \mid (a,b) \in r^\mathcal{I} \text{ and } b \in D^\mathcal{I}\}| \leq n\},$$
$$(\leq n\ r.E)^\mathcal{I} = \{a \in \Delta^\mathcal{I} \mid |\{b \in \Delta^\mathcal{I} \mid (a,b) \in r^\mathcal{I} \text{ and } b \in E^\mathcal{I}\}| \leq n\}.$$

Taking into account our premise that $E \sqsubseteq D$, we can also conclude that for any $b \in \Delta^\mathcal{I}$, it has to be true that if $b \in E^\mathcal{I}$, then $b \in D^\mathcal{I}$, since $E \sqsubseteq D$ iff $E^\mathcal{I} \subseteq D^\mathcal{I}$. Consequently, the inclusion

$$\{b \in \Delta^\mathcal{I} \mid (a,b) \in r^\mathcal{I} \text{ and } b \in E^\mathcal{I}\} \subseteq \{b \in \Delta^\mathcal{I} \mid (a,b) \in r^\mathcal{I} \text{ and } b \in D^\mathcal{I}\}$$

and the corresponding inequality

$$|\{b \in \Delta^\mathcal{I} \mid (a,b) \in r^\mathcal{I} \text{ and } b \in E^\mathcal{I}\}| \leq |\{b \in \Delta^\mathcal{I} \mid (a,b) \in r^\mathcal{I} \text{ and } b \in D^\mathcal{I}\}|$$

must be satisfied for any $a \in \Delta^\mathcal{I}$. Thus, when an individual $a \in \Delta^\mathcal{I}$ belongs to the extension of $\leq n\ r.D$, it has to be an element of $(\leq n\ r.E)^\mathcal{I}$ as well, because, given such an $a \in \Delta^\mathcal{I}$, from our previous observations it follows that

$$\begin{aligned} n &\geq |\{b \in \Delta^\mathcal{I} \mid (a,b) \in r^\mathcal{I} \text{ and } b \in D^\mathcal{I}\}| \\ &\geq |\{b \in \Delta^\mathcal{I} \mid (a,b) \in r^\mathcal{I} \text{ and } b \in E^\mathcal{I}\}|. \end{aligned}$$

This brings us to the desired conclusion that $(\leq n\ r.D)^\mathcal{I} \subseteq (\leq n\ r.E)^\mathcal{I}$. □

As a straight-forward remedy for the described problem, we suggest inverting the direction in which the fillers are refined in this case. Applying the proposed change should resolve this particular issue entirely, for the validity

[3] We will consider only finite interpretations.

[4] $|S|$ denotes the cardinality of a finite set S.

of $\leq n\ r.D \sqsubseteq \leq n\ r.E$, where $E \sqsubseteq D$, also implies that $\leq n\ r.E \rightsquigarrow \leq n\ r.D$ is an operation of specialisation.

We have already made the necessary modifications to the implementation of ρ by incorporating slightly adapted version of `OperatorInverter`'s `refine` method directly into `RhoDRDown`. For more details, see the method named `refineUpwards`.

Note on implementation: The try-catch block in the `refineUpwards` method is there to ensure that the previous configuration options are restored even if an error occurs in the process of refining the negated filler. We decided to take this precaution, because we experienced issues regarding concurrent access to a reasoner's internal data structures when disabling instance-based disjointness checks and sharing the same reasoner instance among multiple operators in the operator pool serving the workers utilised by *ParCEL* and *ParCELEx* algorithms. (We were only experimenting with `ClosedWorldReasoner`.) However, in order to properly address this problem, we had to eventually introduce `ConcurrentClosedWorldReasoner`.

Note on exact cardinality constraints: Inspecting the implementation of `RhoDRDown`, we came across a code section trying to implement undocumented rules to refine exact cardinality constraints. Since these were apparently neither downward nor upward refinement operations, we decided to comment this part out (despite the fact that exact cardinality constraints are currently not included in top-concept refinements).

Note on data properties: Studying the theoretical definition of the extended ρ operator, we realised that in this definition, the restrictions on numeric (double) data properties are also refined in the opposite direction. Nonetheless, the most recent version of DL-Learner we found on GitHub already implements the corrected definition of these refinement operations, and thus we suppose it is no longer an open problem.

2 Cardinality Constraints and the Closed World Reasoner

Unfortunately, redefining the ρ operator still did not fix all issues with accuracy. We noticed this while comparing the performance of *CELOE* and *OCEL* which could not agree on accuracy of concepts containing at-most restrictions because of using different ways to compute the coverage of a class expression. Whereas *CELOE* indirectly calls the `getIndividuals` method of the loaded reasoner (see `getCoverageCount` in `ReasoningUtils`), *OCEL* de-

pends on its `hasType` method. Closely examining `ClosedWorldReasoner`'s[5] implementation of these two methods, we came to the conclusion that the contradictory results were caused mostly by one simple, yet a few times repeated mistake in the process of determining whether an individual satisfies an at-most or at-least restriction. It consisted in forgetting to add the number of the already seen related individuals belonging to the extension of the filler to the number of the remaining related individuals when checking if it is still possible for the individual to be (or not to be) covered by the given expression (in the method `hasTypeImpl`, see the code for handling the cases in which `description` is `OWLObjectMinCardinality`, `OWLObjectMaxCardinality`, or `OWLObjectExactCardinality`). In `getIndividualsImpl`, on the other hand, there was a missing (or mispositioned) if statement in the section responsible for retrieving individuals satisfying an at-most restriction.

Side note On Universal-Quantification Semantics In General: The *Some-Only* and *NonEmpty* universal-quantification semantics are not always correctly treated by `ClosedWorldReasoner`, however, we have not addressed this issue, since we decided to use the *Standard* semantics (although, we have marked some of the problematic places with appropriate `TODO`s).

3 Multiple-Criteria Heuristic: Length Bonus Calculation

In order to improve the performance of *OCEL*, we also considered altering the default configuration of `MultiHeuristic` to better suit our needs. Looking more deeply into its implementation, we had a chance to learn how the length bonus is calculated - the method `getHeuristicLengthBouns`. We have to say that we deem this procedure a little suspicious mainly for the following two reasons. Firstly, the bonus does not accumulate while iterating through the expressions nested inside the one being evaluated. This may be due in part to the fact that the algorithm works only with a set of subexpressions, ignoring duplicates. Secondly, the aforementioned set provides no guarantee of a reasonable and stable traversal order. Hence, we decided to re-implement this method in such a way that the bonus is now calculated cumulatively, taking into account all the subexpressions (each instance of a repeated expression separately) and the actual length metric used during the search. You can find this new version of the `getHeuristicLengthBouns` method in our DL-Learner GitHub repository as well.

Note on implementation: In our implementation, we omitted the bonus

[5] `ClosedWorldReasoner` was the only reasoner involved in our experiments.

for data-property restrictions because we hold the opinion that all constituents of these expressions bear enough additional information w.r.t. their length, thereby eliminating the necessity to promote their use.

4 *ParCEL* and *ParCELEx* Unable to Reach Simple Refinements

Experimenting with the `useHasValueConstructor` option for the ρ operator, we discovered that *ParCEL* and *ParCELEx* algorithms were unable to reach refinements having the same length as the description they resulted from. This problem resided in the fact that the workers were accepting only the refinements which were longer than the current horizontal expansion of the parent node, which is always initialised to the length of the expression the node contains. We thus resolved this issue by modifying the method `refineNode` in the `ParCELWorkerAbstract` class (so that workers ask for refinements whose length is at most equal to the horizontal expansion) and also the corresponding if conditions in all worker implementations (inside their respective definitions of the method `run`). Please, keep in mind that we have tested only the correctness of `ParCELWorker`, `ParCELWorkerExV2`, and `ParCELWorkerExV12` after this change.

5 Other Detected Flaws and Improvements

5.1 Defining Start Class with *OCEL*

Defining a custom `startClass` was not properly supported by *OCEL*, since there was also a private member variable, named `startDescription`, whose value was, indeed, the one stored in the root of the search tree. Therefore, we removed the `startDescription` variable entirely, leaving only `startClass`.

5.2 Computing Applicable Object Properties for ρ

The set of applicable object properties is now computed with the help of the materialised representation of property domains - `opDomains`. For more details, see the method `comupteApp` in `RhoDRDown`.

5.3 Top-Concept Refinements: Constructing M for More Specific Domains

All existential-quantification expressions having nominals as fillers, i.e. `OWLObjectHasValue`, and all self restrictions, i.e. `OWLObjectHasSelf`, were stored in `m` instead of `mA`.

5.4 Calculating Accuracy for *ParCEL* and *ParCELEx*

When calculating accuracy, correctness, and completness (the method `getAccuracyAndCorrectness3` in `ParCELPosNegLP`), the algorithm tries to synchronise on `uncoveredPositiveExamples` in order to make a local copy. However, since the variable is not final, the underlying reference may change via a call to `setUncoveredPositiveExamples`. Even though, this actually never happens, we would rather remove this vulnerability and leverage the fact that when we first set this variable, only the reference to the object managed by the learner gets copied, so any change to that object will be automatically registered (and also any attempt to synchronise within `ParCELPosNegLP` will also prevent other threads from entering the critical sections inside `ParCELAbstract` or `ParCELExAbstract` methods). For more details, see the updated implementation of `setUncoveredPositiveExamples` and notice that the methods such as `newPartialDefinitionsFound` do not invoke it anymore.

Probably a more serious flaw we found was that instead of using the previously constructed local snapshot of `uncoveredPositiveExamples`, its "live" counterpart was used in the denominator of the formula for calculating completeness.

5.5 Minimising $C \sqcup \neg C$

Minimisation of expressions having the form $C_1 \sqcup C_2 \sqcup \cdots \sqcup C_n$, where $C_i \equiv \neg C_j$ for some $1 \leq i < j \leq n$, was not properly implemented in `OWLClassExpressionMinimizer`. More precisely, the algorithm only needed to know that $C_i \sqsubseteq \neg C_j$ was satisfied in order to deduce that $C_i \equiv \neg C_j$ (see the original and the corrected method `visit` for `OWLObjectUnionOf`).

5.6 `ParCELCoveredNegativeExampleComparator` Not Comparing Based on The Coverage of Negatives

This comparator's `compare` method was comparing the number of negative examples covered by `node1` to the number of positive examples covered by `node2` instead of comparing their coverage of negative examples.

References

[1] Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78:203–250, 2009.