Experimental Results on Solving the Projection Problem in Action Formalisms Based on Description Logics

Wael Yehia,[1] Hongkai Liu,[2] Marcel Lippmann,[2] Franz Baader,[2] and Mikhail Soutchanski[3]

[1] York University, Canada
`w2yehia@cse.yorku.ca`
[2] TU Dresden, Germany
`lastname``@tcs.inf.tu-dresden.de`
[3] Ryerson University, Canada
`mes@scs.ryerson.ca`

Abstract. In the reasoning about actions community, one of the most basic reasoning problems is the projection problem: the question whether a certain assertion holds after executing a sequence of actions. While undecidable for general action theories based on the situation calculus, the projection problem was shown to be decidable in two different restrictions of the situation calculus to theories formulated using description logics.
In this paper, we compare our implementations of projection procedures for these two approaches on random testing data for several realistic application domains. Important contributions of this work are not only the obtained experimental results, but also the approach for generating test cases. By using patterns extracted from the respective application domains, we ensure that the randomly generated input data make sense and are not inconsistent.

1 Introduction

The situation calculus (SC) [7] is a popular many-sorted language for representing action theories. The *projection problem* is one of the most basic reasoning problems for action formalisms such as the SC. It deals with the question whether a certain formula holds after executing a sequence of actions from an initial state. Since the situation calculus encompasses full first-order logic, the projection problem for action theories formulated with it is in general undecidable. One possibility to restrict SC such that the projection problem becomes decidable is to use a decidable fragment of first-order logic instead of full first-order logic as underlying base logic [3, 5, 8]. Description Logics (DLs) [1] are well-suited for this purpose since they are well-investigated decidable fragments of first-order logic that have been used as knowledge representation formalisms in various application domains.

For the DL-based action formalism introduced in [3], it could indeed be shown that, in this restricted setting, the projection problem is decidable. Basically, the procedure for solving the projection problem developed in [3] works as follows. Using time-stamped copies of all relevant concept and role names in the input, one can describe the initial state and the subsequent states (obtained by executing the actions of a given sequence of actions one after the other) in an ABox and an (acyclic) TBox. Solving the projection problem is thus reduced to checking whether an appropriately time-stamped version of the assertion to be checked is entailed by this ABox w.r.t. this TBox.

For the SC-based action formalism introduced in [5, 8], the projection problem is solved by *regression*, as usual in situation calculus. In this approach, one transforms a query formula and an action sequence into a regressed formula. It is then checked whether the regressed formula is entailed from an incomplete description of the initial state and the unique name axioms. To obtain decidability, it is thus not sufficient to restrict the base logic (in which the initial state, the axioms, and the query formula are written) to a decidable fragment of first-order logic. One must also show, as done in [5, 8], that regression can be realised within this fragment, i.e., that the regressed formula belongs to this fragment.

Both approaches for solving the projection problem have been implemented. In this paper, we compare these implementations on random testing data for several realistic application domains. In addition to describing the obtained experimental results, this paper also sketches our approach for generating the test cases. By using typical patterns extracted from the respective application domains, we ensure that the randomly generated input data make sense and are not inconsistent.

The rest of the paper is organised as follows. After a short presentation of the two action formalisms and their respective approaches for solving the projection problem, the test case set-up is explained and the experimental results are presented. The paper concludes with a discussion of the results obtained and possible future work.

2 DL-Based Action Formalisms

In this section, we explain the two approaches for deciding the projection problem on an intuitive level. More details on the directly DL-based action formalism can be found in [3] and on the SC-based action formalism restricted to DLs in [5, 8]. We also discuss the differences between the two approaches.

We assume the reader to be familiar with basic notions of Description Logics (DLs), which can e.g. be found in [1]. In the following, we use action formalisms that are based on the Description Logic $\mathcal{ALCO}^U$, which extends the basic DL $\mathcal{ALC}$ with nominals, i.e., concepts interpreted as singleton sets (letter $\mathcal{O}$), and the universal role, which is interpreted as the binary relation that links any two domain elements (superscript U). In the following, the universal role will be denoted by the letter U.

Formulated for DL-based action formalisms, the *projection problem* deals with the following question: Given an initial ABox describing the initial state, a TBox describing the domain constraints, and a sequence of actions, does a query, i.e., an assertion, hold after executing the action sequence from the initial state? In the following, we call the approach for solving the projection problem introduced in [3] the *reduction approach* and the approach for solving the progression problem introduced in [5, 8] the *regression approach*.

In this paper, we will use examples from the *logistics domain* [4]. The domain represents a simplified version of logistics planning. Various objects, e.g. boxes, packages and luggage, can be transported by vehicles, such as airplanes and trucks, from one location to another. Locations, such as airports and warehouses, are located in cities.

2.1 The Reduction Approach

The action formalism [3] to which this approach applies, describes the possible initial states by an $\mathcal{ALCO}^U$-ABox and the domain constraints by an acyclic TBox. The restriction to acyclic TBoxes avoids anomalies caused by the so-called ramification problem (see [2, 6] for approaches that can deal with more general TBoxes). It remains to describe how actions are formalised. In contrast to the general setting introduced in [3], the actions used here are without occlusions,[4] which can be omitted since they are not required for the application domains used in our experiments. Basically, an action consists of a set of pre-conditions and a set of post-conditions. Both are given as ABox assertions. Pre-conditions must be satisfied for the action to execute and post-conditions describe what new assertions must be satisfied after the application of the action. Post-conditions can be conditional, i.e., the required change is only made in case the condition is satisfied. The following action, for instance, causes the airplane OurBoeing777 to fly from AirportJFK to AirportSIN transporting Box42, which is the only cargo:

$$\begin{aligned}\mathsf{fly}_1 := (\{&\mathsf{at}(\mathsf{OurBoeing777}, \mathsf{AirportJFK})\},\\ &\{\{\mathsf{loaded}(\mathsf{Box42}, \mathsf{OurBoeing777})\}/\neg\mathsf{at}(\mathsf{Box42}, \mathsf{NYC}),\\ &\quad\{\mathsf{loaded}(\mathsf{Box42}, \mathsf{OurBoeing777})\}/\mathsf{at}(\mathsf{Box42}, \mathsf{Singapore})\}).\end{aligned}$$

The pre-condition at(OurBoeing777, AirportJFK) ensures that this action is only applicable if OurBoeing777 is indeed at AirportJFK. The effect of this action, described by two conditional post-conditions, is that the box Box42 is no longer at NYC but at Singapore if it was loaded to OurBoeing777. This action is a simplified version of an action used in our experiments.

According to the semantics of DL actions defined in [3], nothing should change that is not explicitly required to change by some post-condition. It was shown in [3] that this action formalism can be seen as an instance of Reiter's situation calculus, i.e., there is a translation to SC.

[4] Occlusions describe which parts of the domain can change arbitrarily when an action is applied. Details about occlusions can be found in [3].

In [3], the projection problem for this action formalism is reduced to the consistency problem of an ABox w.r.t. an acyclic TBox. The central idea of the reduction is to create time-stamped copies of all concept, role, and individual names. Intuitively, A^0 is the concept A before executing any action, A^1 after executing the first action, A^2, after subsequently executing the second one, etc. Using such time-stamped copies, the effect that executing the actions has on named individuals, i.e., domain elements that correspond to an individual name, can be described using ABox assertions. Since post-conditions only state changes for named individuals, nothing should change for unnamed ones. This is expressed using an acyclic TBox.

Note that the reduction in [3] does not deal with the universal role, but integrating it into the reduction is not hard. In fact, its interpretation never changes, and it is not allowed to occur in post-conditions. We also applied some simple optimisations to the original reduction, which however proved to be quite valuable w.r.t. the time and memory consumption of our implementation.

2.2 The Regression Approach

This approach is based on Basic Action Theories (BATs) in the Situation Calculus (SC) [7]. In general, a BAT consists of pre-condition axioms (PAs) and successor state axioms (SSAs), which describe the effects and non-effects of actions; an initial theory, which describe the possible initial states; an situation independent acyclic terminology describing convenient definitions of the application domain; a set of domain independent foundational axioms; and axioms for the unique-name assumption (UNA). In SC, a sequence of actions is called a *situation*, and the empty sequence the *initial situation*. Applying actions can change the interpretations of certain predicates, which are called *fluents*. Fluents have an additional argument position to express the situation in which the fluent is considered. Given the initial theory, the situation determines which are the possible current states, i.e., how the fluents have been changed by executing the sequence of actions. A fundamental problem in reasoning about actions is the frame problem, i.e., how one compactly represents numerous non-effects of actions. Reiter's approach [7] for solving this problem is based on quantifying over action variables in SSAs. This approach is adapted to the context of BATs based on decidable description logics in [5, 8].

In the context of BATs in SC, *regression* is the act of converting the query formula (which should hold after executing the action sequence) to a formula that should hold in the initial situation by making use of the SSAs and the domain constraints. In the regression approach used in this paper, solving the projection problem in certain BATs based on a restricted first-order language, called $\mathcal{L}$, is reduced to solving the satisfiability problem in the Description Logic $\mathcal{ALCO}^U$. This is done by imposing precise syntactic constraints on Reiter's SSAs to guarantee that after doing the logically equivalent transformations required by regression, the resulting regressed formula remains in $\mathcal{L}$, if the projection query formula is in $\mathcal{L}$. Instead of defining this approach in detail, which is not possible due to space constraints, we illustrate it on an example. Detailed definitions can be found in [5, 8].

There is a PA axiom for each action that describes the pre-conditions for executing the action. For instance, the PA of the action fly looks as follows:

$$\begin{aligned}\mathsf{Poss}(\mathsf{fly}(z_1, z_2, z_3), S) = {} & \mathsf{airplane}(z_1) \wedge \mathsf{airport}(z_2) \wedge \mathsf{airport}(z_3) \wedge \\ & \mathsf{at}(z_1, z_2, S) \wedge (z_2 \neq z_3)\end{aligned}$$

Similarly, there is one SSA per fluent that describes the conditions under which actions make the fluent true or false. This is the SSA of the fluent loaded:

$$\begin{aligned}\mathsf{loaded}(x, y, \mathsf{do}(a, S)) = {} & \big(\exists z_1.a = \mathsf{load}(x, y, z_1) \wedge \mathsf{ready}(x, S)\big) \vee \\ & \big(\mathsf{loaded}(x, y, S) \wedge \neg(\exists z_1.a = \mathsf{unload}(x, y, z_1))\big)\end{aligned}$$

The domain constraints are basically acyclic TBox axioms, but written in the language $\mathcal{L}$. An example would be the following:

$$\mathsf{object}(x) \equiv \mathsf{box}(x) \vee \mathsf{luggage}(x)$$

The initial theory and the query are $\mathcal{L}$ sentences that are restricted such that they can be transformed into $\mathcal{ALCO}^U$-assertions. The following is an example of an initial theory $\mathcal{D}_{S_0}$ and a query W.

$$\begin{aligned}\mathcal{D}_{S_0} &= \mathsf{truck}(T_1) \wedge \mathsf{box}(B_1) \wedge \mathsf{location}(L_1) \wedge \forall x.(\mathsf{box}(x) \supset \mathsf{loaded}(x, T_1)) \\ W &= \mathsf{loaded}(B_1, T_1, \mathsf{do}([\mathsf{load}(B_1, T_1, L_1), \mathsf{unload}(B_1, T_2, L_1)], S_0))\end{aligned}$$

Note that the action sequence is here viewed to be part of the query, whereas for the reduction approach it is given separately.

2.3 Comparison of the Two Approaches and Restrictions

Both approaches solve the projection problem for an action formalism that is based on the DL $\mathcal{ALCO}^U$. The action sequence is in both cases a list of ground actions, i.e., they do not contain free variables. However, the actions considered in the regression approach are more general than the ones in the reduction approach. In fact, the actions of the latter approach are so-called local-effect actions, which can only effect changes for *named* individual (i.e., ones that are explicitly named by a constant symbol), whereas global-effect actions, which are allowed in the former approach, can also effect changes for unnamed individuals. For example, the action fly in the regression approach moves all objects loaded to the airplane from one city to the other, independent of whether these objects have a name or not. Its approximation in the reduction approach moves only objects that are explicitly mentioned by their name in the description of the action. Though this approximation may in principle lead to different answers for the projection problem, this never happened in our experiments.

Another difference are the languages in which the queries are formulated. In the reduction approach, only instance queries are considered, whereas the other approach formulates queries in a restricted first-order language, with explicit variables and (unguarded) quantification. However, for the queries considered in our experiments, we were able to use the universal role to bridge this gap.

Regarding the complexity of the projection problem, regression may take up to double exponential time and space, whereas the reduction approach takes polynomial space. However, this is the worst-case complexity of the approaches. The more expressive regression approach is assumed to hit this bound only in non-practical cases.

In the next section, we show how these approaches behave in practice by evaluating them on randomly generated input data that are modelled on patterns occurring in applications.

3 Test Case Generation and Experimental Results

In order to evaluate the two approaches for solving the projection problem, we generated testing data inspired by several application domains. Of course, the representation of the input data is different for the two approaches. Basically, we generated the data for the regression approach, and then translated them into the format required by the reduction approach. As already mentioned, global-effect actions are approximated by local effect actions that make changes to all the named individuals. Otherwise, the translation preserves the semantics of the action theory. To increase readability for DL-literate readers, we describe the translated input data rather than the SC-based ones. Also, we use the size of the translated input data as size of the input.

In general, a test case is divided into a fixed part and a varying part. The fixed part consists of descriptions of all available actions and an acyclic TBox describing the domain. The varying part consisting of the initial ABox, the query to be asked, and the action sequence.

Subsequently, we sketch one domain, the *logistics domain*, to give an impression about what kind of projection problems can be expressed and solved by the formalisms.[5] The examples used above already gave an impression of how the domain looks like. It represents a simplified version of logistics planning. Various objects, e.g. boxes and luggage, can be transported by vehicles, such as airplanes and trucks, from one location to another.

More precisely, the fixed part is as follows. The acyclic TBox describes general facts about the domain. It contains the following axioms:

$$\begin{aligned}
\mathsf{Object} &\equiv \mathsf{Box} \sqcup \mathsf{Luggage}\\
\mathsf{Vehicle} &\equiv \mathsf{Truck} \sqcup \mathsf{Airplane}\\
\mathsf{Truck} &\equiv \mathsf{TransportTruck} \sqcup \mathsf{MailTruck} \sqcup \mathsf{RecyclingTruck}\\
\mathsf{Location} &\equiv \mathsf{Airport} \sqcup \mathsf{Garage} \sqcup \mathsf{Street} \sqcup \mathsf{Warehouse}
\end{aligned}$$

[5] See `http://www.cse.yorku.ca/~w2yehia/dl2012_results.html` for further domains.

The initial ABox contains incomplete knowledge about the initial state. It states static facts like Airplane(OurBoeing777), Airport(AirportJFK), Box(Box42), Truck(OurTruck), City(NYC), inCity(OurGarage, NYC), etc. In addition, the initial ABox contains dynamical knowledge like at(OurTruck, AirportJFK), Ready(Box42), loaded(Box127, OurBoeing777), which means that OurTruck is at AirportJFK, Box42 is ready to be loaded, and Box127 is loaded into OurBoeing777.

The action sequences are built using the atomic actions load, unload, fly, and driveTruck, instantiated with appropriate individuals. The action load loads an object into a vehicle at a location, e.g. load(Box42, OurBoeing777, AirportJFK) is an instance of this action. It is executable if both Box42 and OurBoeing777 are at AirportJFK, i.e., at(Box42, AirportJFK) and at(OurBoeing777, AirportJFK) hold, and Box42 is ready to be loaded, i.e., we have Ready(Box42). The action unload is defined analogously for unloading an object from a vehicle at a location. Executing the action fly means that an airplane flies from one airport to another one, e.g. fly(OurBoeing777, AirportJFK, AirportSIN). This action is executable if we have at(OurBoeing777, AirportJFK), and the two airports are not the same, which is the case in the example. The action driveTruck is similar. It is used to drive a truck from one location to another one. An example for such an action would be driveTruck(OurTruck, OurGarage, AirportJFK). Executing the action is only possible if OurGarage and AirportJFK are at the same city.

On the DL side, queries are instance queries, i.e., concept assertions of the form $C(a)$ where C is a $\mathcal{ALCO}^U$-concept description built using the concept names and role names introduced above and a is an individual name such as Box42, AirportJFK, OurBoeing777, etc.

To properly test our implementations, we obviously need some sort of automated generation of input data, but due to the non-trivial expressivity of the language at hand, it is hard to generate useful test cases. We could not find any precedence for such an attempt, i.e., generating random projection problem test cases, in the literature. For planning domains [4] there are some people working on test case generation, but they are dealing with STRIPS or some of its extensions, which are mostly at the propositional level. For such inexpressive formalisms, it easier to generate input data that make sense. Generating purely random $\mathcal{ALCO}^U$-ABoxes and instance queries usually yields meaningless queries or initial theories that are inconsistent or action sequences that are not executable.

We describe now more precisely how we generate the testing data. The TBox contains the axioms above plus some auxiliary axioms described below. The initial ABox contains both static knowledge and dynamical knowledge. For the static knowledge, it makes no sense to add incompleteness, because usually we know, for instance, that Box42 is a box. We omit, however, axioms stating facts like Box42 is not an airplane, i.e., ¬Airplane(Box42). The real incompleteness concerns the dynamical knowledge. We found it useful to generate more or less complete knowledge by generating a number of boxes, trucks, airplanes, airports, and cities together with static knowledge like in which city airports are located. We generate also assertions for the role names at, loaded and Ready for all individuals of the respective type. We add more incompleteness by removing assertions from the ABox and adding new assertions using the following transformation rules: a $\sqcup$-rule, and an $\exists$-rule. Applying the $\sqcup$-rule means to take two assertions $A(a)$, $B(a)$ and replacing them with the assertion $(A \sqcup B)(a)$. For reasons of simplicity and interchangeability of the formats read by our implementations, we introduce a new concept name Aux, and add the concept definition $\mathsf{Aux} \equiv A \sqcup B$ to the TBox, and the assertion $\mathsf{Aux}(a)$ to the ABox.[6] Applying the $\exists$-rule means to take a role assertion $r(a, b)$ and to replace it with $(\exists r.\top)(a)$. For example, $\mathsf{at}(\mathsf{Box127}, \mathsf{AirportSIN})$ could be replaced with $(\exists \mathsf{at}.\top)(\mathsf{Box127})$. Instead of knowing that $\mathsf{Box127}$ is at $\mathsf{AirportSIN}$, we now just know that $\mathsf{Box127}$ is somewhere, i.e., at a location. Again, we introduce a new concept name for $\exists \mathsf{at}.\top$, and add its definition to the TBox. Of course, it makes no sense to apply the rules too often since then the resulting ABox would be too incomplete. For this reason, we use them rather sparingly: roughly there are ten-times more individuals than the number of times these rules are applied.

The query and the action sequence are generated using (domain-specific) patterns. Building formulas from hand-made patterns occurring in real applications is one step towards generating realistic test cases, but it suffers from the fact that the generated formulas might be too similar, and thus the test cases do not provide the extensive coverage that random formula generation does. Thus, we mixed patterns with a bit of randomness. We developed ten patterns for the logistics domain. Some of these patterns can take input, and generate a random input if none is given. For example, a pattern might describe whether there are any boxes on a truck in some location. The input could be any combination of box, truck or location constants. If necessary, a missing constant can be randomly generated, or it could remain as a variable otherwise. The following formula Φ was generated from such a pattern, where the input was box_i and $\mathsf{location}_j$. The generated formula is translated to the $\mathcal{ALCO}^U$-assertion α with aux being a dummy individual.

$$\begin{aligned}
\Phi &= \exists y.(\mathsf{loaded}(\mathsf{box}_i, y) \wedge \mathsf{truck}(y) \wedge \mathsf{at}(y, \mathsf{location}_j)) \\
\alpha &= (\exists U.(\{\mathsf{box}_i\} \sqcap \exists \mathsf{loaded}.(\mathsf{Truck} \sqcap \exists \mathsf{at}.\{\mathsf{location}_j\})))(\mathsf{aux})
\end{aligned}$$

We use those simple but versatile patterns to generate formulas that replace the propositions in a randomly generated satisfiable formula of propositional logic in disjunctive normal form in order to obtain the query. Using this approach, it is ensured that we have more randomness at the propositional level and less randomness at the FOL level.

Finally, to generate random but executable action sequences, we used patterns again. But now a pattern is a generic description of a sequence of actions necessary to satisfy a goal. For instance, one action sequence in the logistics domain describes the process of gathering all known boxes in a particular city and transporting them to another city. The choice of the cities is random, and

[6] Thus, the TBox contains also abbreviations for concepts, where the abbreviation occurs only once in the ABox.

picking the transportation vehicle is random as well. On top of that, we compute this action sequence based on a random initial ABox. Of course, we could have tried to pick purely random actions, but then most of the generated action sequences would not have been executable (i.e., not all pre-conditions would be satisfied). Testing randomly generated action sequences for executability takes time and would result in having to throw away most of the generated sequences.

We now present the testing results we obtained for the logistics domain. The results were obtained on an Intel® Core$^{\text{TM}}$ 2 Duo E8400 CPU with a clock frequency of 3.00 GHz, and 4 GB of RAM. We used JVM version 1.7.0. Both implementations use HermiT 1.3.6 as underlying DL reasoner.

To make the results better comparable, we generated a couple of initial theories, and then manually removed individuals from them to create new smaller initial theories. Then, we generated a set of queries and action sequences for each initial theory. This way, we can measure the running time for solving the projection problem as the ABox varies in size, while the query and action sequence stay the same. Some of the testing results for about 700 test cases that we obtained can be found in Tab. 1. These results give an impression of the performance of the two approaches. We sketched how the action sequences and the queries look like. Empty cells in Tab. 1 mean that the program could not finish within two minutes, which was our cut-off time. To be useful in practice, an answer within two minutes is preferred. For the regression approach, we also measured the time for computing the regressed formula only, which is usually quite small (about 0.01 s). Thus, the main time for the regression approach is used for checking the consistency of the generated knowledge base using HermiT. One can observe that the running times very much depend on the action sequence and the query that is asked. For action sequences of length 5 or more, both approaches seem to be struggling due to overhead taken by HermiT. Thus, improving DL-systems will improve the running time of our approaches significantly. Note that the last test case in Tab. 1 describes a rather simple scenario, which is still too complex for both approaches. One can observe also that the size of the initial ABox affects the running time. Note that the length of the regressed formula is not affected by these variations. Also, in most cases, a query involving a value restriction on U causes a longer running time for the reduction approach.

All in all, the testing done in this paper should be considered as a first step. More testing and a careful inspection of the structure of the input is needed to give an answer to the question which approach performs better on which inputs.

4 Conclusion and Future Work

We have compared the implementations of two approaches for solving the projection problem for DL-based action formalisms. The main result of this comparison is twofold. First, it is clear that both approaches seem to be struggling when dealing with action sequences longer than five. It turned out that computing the regression formula can be done very quickly, and most of the time for the regression approach is consumed by the DL-reasoner. For the reduction approach, it is

Table 1. Testing results for the logistics domain

Action sequence/ query	Number of individuals in ABox	Time for reduction approach	Time for regression approach
load $(\exists U.(\exists \mathsf{loaded}.(\exists \mathsf{at}.\{\ell_1\} \sqcap \exists \mathsf{at}.\{\ell_2\} \sqcap \mathsf{Truck}) \sqcap \mathsf{Box}))(\mathsf{aux})$	12	0.57 s	8.51 s
	13	0.58 s	17.07 s
	14	0.58 s	34.51 s
	15	0.69 s	68.98 s
	16	0.59 s	—
	17	0.59 s	—
	40	0.64 s	—
driveTruck $(\exists U.(\exists \mathsf{loaded}.(\exists \mathsf{at}.\{\ell\} \sqcap \mathsf{Truck})) \sqcap \mathsf{Box})(\mathsf{aux})$	7	0.64 s	1.36 s
	8	0.58 s	8.50 s
	9	0.82 s	2.14 s
	10	0.55 s	4.75 s
	11	0.55 s	12.09 s
	12	0.56 s	15.15 s
	13	0.90 s	16.21 s
fly, fly $(\forall U.(\neg(\mathsf{Box} \sqcap \exists \mathsf{at}.\{\ell\}) \sqcup \mathsf{Ready}))(\mathsf{aux})$	13	2.24 s	1.6 s
unload, load, unload $(\exists U.(\mathsf{Box} \sqcap \exists \mathsf{loaded}.\mathsf{Truck}))(\mathsf{aux})$	13	0.92 s	1.69 s
	14	1.04 s	1.68 s
	15	0.96 s	1.64 s
unload, load, fly $(\exists U.(\{b\} \sqcap \exists \mathsf{loaded}.\mathsf{Truck}))(\mathsf{aux})$	12	1.75 s	4.66 s
	13	1.76 s	8.51 s
	14	1.85 s	18.57 s
	34	6.63 s	2.72 s
load, load, load, fly $(\forall U.(\neg(\exists \mathsf{at}.\{\ell\} \sqcap \mathsf{Box}) \sqcup (\exists \mathsf{loaded}.\{a\})))(\mathsf{aux})$	30	—	70.98 s
unload, load, load, unload, unload $(\forall U.(\neg(\exists \mathsf{at}.\{\ell\}) \sqcap \mathsf{Box}) \sqcup \mathsf{Ready})(\mathsf{aux})$	8	—	23.04 s
	9	0.56 s	—
	10	0.57 s	—
	11	—	—
	12	—	—
unload, load, load, load, fly, unload, unload, unload $(\exists U.(\{b\} \sqcap \exists \mathsf{loaded}.\mathsf{Truck}))(\mathsf{aux})$	26	—	—

also clear that long action sequences cannot be handled efficiently since, in the worst case, the initial ABox is copied for each point in time. Also, we observed that the length of the action sequence is not the only metric that affects the running time. The size of the initial ABox as well as the form of the query are also important. We expect that more testing will give a more accurate description on what what are the weak points of the approaches. Maybe also the actions, i.e., the size and structure of the pre-conditions and the post-conditions, need to be considered in more detail. For the reduction approach, the pre-conditions affect the runtime in our experiments, while for regression they do not. As a result of these investigations, we hope to locate the parts of the implementations that need to be optimised.

The second main result is that we gained some experience in automatically generating testing data for the projection problem. It is clear that generating good test cases is not a trivial task (comparable to generating good test cases in programming languages). Randomly generating test cases on a purely syntactic level without taking the semantics of action formalisms into account is problematic since the resulting initial ABoxes can easily be inconsistent, the action sequences not executable, etc. In these cases, projection does not make sense. To take the semantics into account, we followed a domain specific approach, and used patterns found in applications to generate data that make sense from a semantic point of view. Nevertheless, these are only first step towards semantically well-founded test case generation. More versatile approaches need to be developed as future work.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. Baader, F., Lippmann, M., Liu, H.: Using causal relationships to deal with the ramification problem in action formalisms based on description logics. In: Proc. of the 17th Int. Conf. on Logic for Programming, Artifical Intelligence, and Reasoning (LPAR-17). Lecture Notes in Computer Science, vol. 6397, pp. 82–96. Springer-Verlag (2010)
3. Baader, F., Lutz, C., Miličić, M., Sattler, U., Wolter, F.: Integrating description logics and action formalisms: First results. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI-05). AAAI Press (2005)
4. Bacchus, F.: The AIPS '00 planning competition. AI Magazine 22(3), 47–56 (2001)
5. Gu, Y., Soutchanski, M.: A description logic based situation calculus. Ann. Math. Artif. Intell. 58(1-2), 3–83 (2010)
6. Lin, F., Soutchanski, M.: Causal theories of actions revisited. In: Proc. of the 25th AAAI Conf. on Artificial Intelligence (AAAI-2011). pp. 235–240 (2011)
7. Reiter, R.: Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems. The MIT Press, Bradford Books, Cambridge, Massachusetts, USA (2001)
8. Soutchanski, M., Yehia, W.: Towards an expressive decidable logical action theory. In: Proc. of the 25th Int. Workshop on Description Logics (DL-2012) (2012)