Towards Ontology-Mediated Planning with OWL DL Ontologies

Tobias John[1], Patrick Koopmann[2]

[1]University of Oslo, Gaustadalléen 23B, 0316 Oslo, Norway
[2]Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands

Abstract
While classical planning languages make the closed-domain and closed-world assumption, there have been various approaches to extend those with DL reasoning, which is then interpreted under the usual open-world semantics. Current approaches for planning with DL ontologies integrate the DL directly into the planning language, and practical approaches have been developed based on first-order rewritings or rewritings into datalog. We present here a new approach in which the planning specification and ontology are kept separate, and are linked together using an interface. This allows planning experts to work in a familiar formalism, while existing ontologies can be easily integrated and extended by ontology experts. Our approach for planning with those ontology-mediated planning problems is optimized for cases with comparatively small domains, and supports the whole OWL DL fragment. The idea is to rewrite the ontology-mediated planning problem into a classical planning problem to be processed by existing planning tools. Different to other approaches, our rewriting is data-dependent. A first experimental evaluation of our approach shows the potential and limitations of this approach.

Keywords
Planning, OWL Ontologies, Description Logics

1. Introduction

We present a new formalism to integrate OWL ontologies into planning problems, together with a first practical technique for automated planning for such ontology-mediated planning problems. Different to existing approaches, our formalism keeps the ontology component and the planning component separate from each other. Our practical implementation is optimized for planning problems with small domains, and is a first technique for automated planning that supports full OWL.

Both planning and ontologies are commonly used in approaches to develop autonomous robots [1, 2]. The motivation for the present work comes from planning problems for autonomous underwater vehicles (AUVs). Such robots are often used for inspection tasks, e.g. of underwater infrastructure such as pipelines or oil platforms, as well as for mapping of the sea floor [3], but eventually they should also be able to complete more complex missions that include manipulation tasks [4]. The robots need to be able to work autonomously, because their operation area is very remote and without a connection to a human operator. Even recovering the vehicle in case of a problem is a difficult and time consuming task. Therefore, the mission

DL 2023: 36th International Workshop on Description Logics, September 2–4, 2023, Rhodes, Greece
0000-0001-5855-6632 (T. John); 0000-0001-5999-2583 (P. Koopmann)

plans for such vehicles should be as robust as possible, which includes that the robots have some understanding of the domain they operate in. This domain knowledge is not specific to planning, and would thus be ideally formalized in an ontology that can also be used in other contexts of AUVs, such as configuring them, or recognizing unexpected situations [5]. For example, such an ontology might define a concept of ProtectedAnimal, based on the concept of Animal and having a position that is located in a NatureProtectionArea. Using such an ontology, the robot would then be able to understand when it needs to keep a larger distance to an animal in order not to disturb it. Ontologies are an ideal framework to represent such domain knowledge, and there are existing ontologies for the underwater domain, such as the SWARMS ontology [6, 7]. However, if we want to use such an ontology in connection with planning, we need a planning framework that can make use of the ontology.

In this paper, we propose a general framework to connect planning problems with OWL ontologies, and a technique to compute plans for such problems. Using this framework, we can create a planning domain that interacts with the ontology to generate plans that take its domain knowledge into account. Similar to [4], we use the ontology to model the environment. But additionally, we model actions of the robot that manipulate the objects and the relations between objects in the environment, e.g. that the robot opens or closes a valve.

Using ontologies to support planning is not a new idea, and has been investigated for decades. An overview about early works in which ontologies are used to infer implicit information about planning states can be found in [8]. Different approaches have since then been used to model planning domains, actions, and even planning problems using ontologies, but also to use ontologies to generate planning problems, in domains as diverse as kitting and assembly [9, 10], semantic web service decomposition [11, 12], robotics [13], train depot management [14] and manufacturing [15]. These approaches usually depend on a static ontology that is used to generate specifications for the planner, while the actions of the planning specifications cannot modify the ontology.

In [16, 17], actions can use DL concepts in the preconditions and postconditions of an action, which then operate on the models of an OWL ontology. A downside of letting actions directly operate on the models is that it is not trivial to determine the implicit consequences of an action, that is, to ensure that after executing an action on a model, we obtain an interpretation that is still a model of the ontology. This problem is also known as the ramification problem.

The ramification problem is avoided in approaches where actions do not operate on models, but on the knowledge base itself. This is the case with the *Knowledge Action Bases* (KABs) and *extended Knowledge Action Bases* (eKABs) introduced in [18, 19], which combine DL knowledge bases with actions that can add facts to and remove them from the knowledge base. Here, every state in the planning domain corresponds to a DL knowledge base, and pre-conditions of actions can query implicit information entailed in the current state via DL reasoning. The idea is that what is known about the world in each system state is represented using facts of a knowledge base, interpreted as potentially incomplete under the open-world assumption, and any implicit consequences of an action are accessed only through reasoning with the ontology. Existing approaches to plan with eKABs practically rely on rewriting the eKABs into planning problems in pure PDDL [20] or its extension with derived predicates, i.e. axioms [21], so that a standard planning system such as Fast-Downward planner [22] can be used. The limits of such an approach are investigated in [23], where the underlying ontology can be expressed in the description logic Horn-$\mathcal{ALCHOIQ}$, which roughly corresponds to the Horn-fragment of OWL DL without complex object property axioms. While Horn-$\mathcal{ALCHOIQ}$ is quite expressive, there are many properties useful for planning that cannot be expressed (see Section 3 for a simple example). To our knowledge, no research in this direction considers more expressive ontology languages.

Our approach is close to that of eKABs, but goes beyond existing approaches: 1) Rather than integrating actions and knowledge, we strive for a separation of the representation formalisms, and 2) using a domain-dependent rewriting approach, we are able to support the full OWL DL 2 syntax as defined in [24] (including SWRL rules [25]).

The aim of 1) is to have a presentation format that is tailored towards the specific needs and skills of knowledge engineers and planning experts. In particular, in our framework, we favor a strong separation of concerns, with the planning specification encoded in standard PDDL, and the domain knowledge encoded in a separate OWL ontology. The connection between the two is established via an interface that links statements in the planning language to OWL axioms. This way, existing OWL ontologies can be easily integrated, and PDDL experts do not need to learn another knowledge representation formalism.

Our solution to 2) is inspired by a technique for ontology-mediated probabilistic model checking presented in [26, 27], which uses a similar separation of concerns as our approach, but with a simpler representation of states using propositional logic. This allows us to support ontologies that go beyond Horn, and are thus able to use many naturally occurring constructs such as disjunction (e.g. to express that a valve must be either open or closed), or at-most constraints (e.g. to express how many objects an AUV can carry). Similar to the work in [28], we use justifications [29] to determine which elements of a planning state are relevant to an action to be executed. However, while the authors of [28] are interested in explaining pre-conditions in an action for a singular state, we use justifications to determine conditions on all possible states.

This paper extends our work on defining Ontology-Mediated planning as presented in [30] by a first evaluation. We demonstrate with the implementation that our method is capable of dealing with complex planning problems but that there are also planning domains where existing methods are superior.

2. Preliminaries

We recall the relevant notions regarding planning with PDDL. We assume the reader is familiar with the basics of OWL and description logics (DLs). For an introduction into OWL and description logics, we refer to [31]. We further assume standard knowledge of first-order logic, and use $\models$ to express entailment between theories and satisfaction in models. We call a formula $P(\vec{t})$ *atom*, which is *ground* if $\vec{t}$ contains only constants.

2.1. PDDL Planning Specifications

We consider the common syntax and semantics as introduced in [20, 32] and described in detail in [33]. A *PDDL planning specification* $\mathbf{P}$ is a tuple $\langle D, P\rangle$ that contains a *domain* $D = \langle \mathcal{P}, \mathcal{A}, \mathcal{D}\rangle$ and a *problem* $P = \langle O, I, G\rangle$. Here, $\mathcal{P}$ is a finite set of predicate names, $\mathcal{A}$ a finite set of actions,

$\mathcal{D}$ a finite set of derivation rules, O a finite set of objects, I is an initial state and the goal G is a first-order formula with predicates from $\mathcal{P}$. A *state* is a finite set of ground atoms over $\mathcal{P}$ and O, interpreted as first-order interpretation; an *action* is a tuple $a = \langle V, \mathsf{pre}, \mathsf{eff}\rangle$ where V is a vector of variables, pre is the *precondition* (a first-order formula with predicates from $\mathcal{P}$ and free variables from V) and $\mathsf{eff} = \langle \mathsf{add}, \mathsf{del}\rangle$ is the *effect*. Both add and del are finite sets of atoms over predicates from $\mathcal{P}$ using variables from V and constants from O. If neither pre nor eff contain variables from V or $V = \emptyset$, we call a a *ground action*.

Derivation rules are of the form $p(V) \leftarrow \phi(V)$, where V is a vector of variables, $p \in \mathcal{P}$, and ϕ is a first-order formula over the predicates in $\mathcal{P}$ with free variables V and constants from O. We often call derivation rules just *rules* and $\phi(V)$ the *body* of a rule. If a predicate p occurs on the left hand side of a rule, it is called a *derived predicate*. Derived predicates are neither allowed to occur negatively in a derivation rule, nor are they allowed to occur in an effect of an action. For a finite set of atoms s, we define $\mathcal{D}(s)$ as the least fix point over the possible applications of some rules from $\mathcal{D}$ to the atoms in s, i.e., we apply the rules from $\mathcal{D}$ exhaustively and add the derived ground atoms until no more rules can be applied.

Let $a = \langle V, \mathsf{pre}, \mathsf{eff}\rangle$ with $\mathsf{eff} = \langle \mathsf{add}, \mathsf{del}\rangle$ be an action and $\theta : V \mapsto O$ a variable assignment. We denote by $\theta(a)$ the ground action obtained by replacing each $x \in V$ in a with $\theta(x)$. A ground action is *applicable* in a state s iff $\mathcal{D}(s) \models \mathsf{pre}$, that is, the precondition is evaluated over the atoms in the state and the entailed derived atoms. The result of applying the action a on s is then denoted $s(a)$, defined as $s(a) := (s \setminus \mathsf{del}) \cup \mathsf{add}$, i.e., all atoms are deleted and added according to the effect. A *plan* π is now a sequence $a_1 \ldots a_n$ of ground actions that generates a sequence of states $s_0 \ldots s_n$ such that 1) $s_0 = I$ is the initial state of the planning problem, 2) for each $i \in \{1, \ldots, n\}$, a_i is applicable in s_{i-1} and $s_i = s_{i-1}(a_i)$, and 3) the goal is reached: $\mathcal{D}(s_n) \models G$.

There are many extensions to PDDL, for example conditional effects. The described components are the ones necessary for our framework but it can also be used with such extensions.

3. The Framework

We capture our framework formally via *ontology-mediated planning specifications*. At the heart of those is the notion of *ontology-enhanced states*, which combine a PDDL state with an OWL ontology.

Definition 1 (Ontology-Enhanced State). *An* ontology-enhanced state *is a tuple* $q = \langle P_q, \mathcal{O}_q\rangle$, *where* P_q *is a set of atoms called the* planner perspective of q, *and* $\mathcal{O}_q$ *is a set of OWL axioms called the* OWL perspective of q.

The idea is that each state has a *planner perspective*, on which the planner directly operates, and on which preconditions and effects of actions are evaluated and executed, respectively. The planner perspective of an ontology-enhanced state is, as for classical planning problems, a set of ground atoms, where predicates of arbitrary arity may occur. On the other side, there is the *OWL perspective* of the ontology-enhanced state, which corresponds to an OWL ontology, i.e. a set of OWL axioms, and from which implicit entailments can be derived using reasoning. The two perspectives are linked via an interface: which axioms are in the OWL perspective

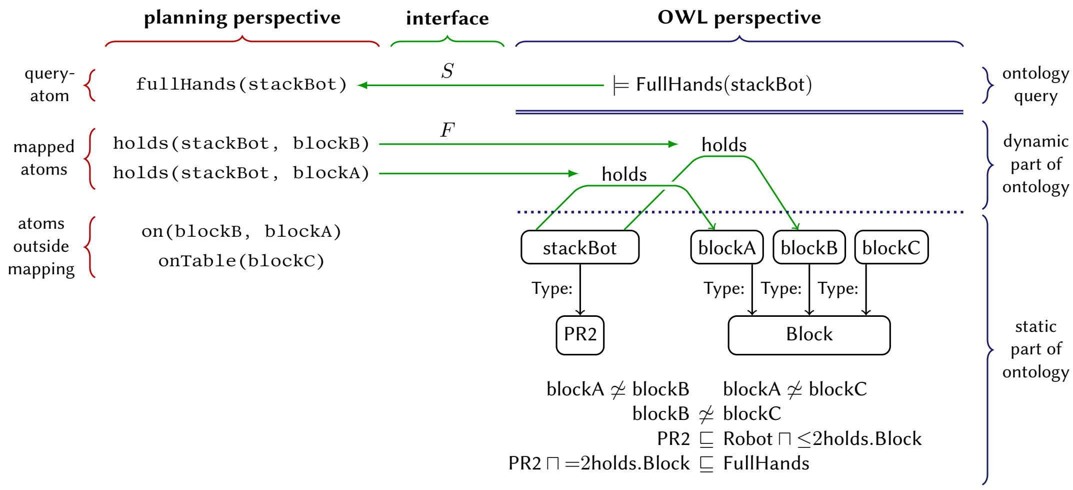

Figure 1: Example of ontology based planning. The interface maps ontology queries to planning predicates and atoms in the planning perspective to ABox atoms. The static part of the ontology contains information about instances (ABox) as well as general axioms (TBox). The connections between the two perspectives via the fluent (F) and query (S) interface are shown in green.

depends on the atoms in the planner perspective. There is however also a static part, which we call the static ontology, that describes time-independent information (such as class definitions and general domain knowledge), which is obtained from an external OWL file and has no direct correspondence in the planner perspective. The planner perspective can access implicit information from the OWL perspective using *query predicates*. Specifically, whether a query-atom is active in the planner perspective depends on what can be derived from the OWL perspective of the state. Before we give the formal definition of how this works, we illustrate this idea with an example.

Example 1. *An example of an ontology-enhanced state is depicted in Figure 1. The scenario is inspired from the classical blocksworld planning example. In contrast to the classical problem where the robot has only one hand, we use an OWL ontology to specify the type of the robot and infer its number of hands. In the example, the stacking robot is a* PR2 *robot [34] that can hold two blocks at a time, and if it holds two blocks, it becomes an instance of* FullHands*. While relatively simple, those cardinality constraints already go beyond the expressivity of Horn-$\mathcal{ALCHOIQ}$, the most expressive DL currently supported by existing implementations for eKABs (see Section 1). The planner perspective of the state is shown on the left, and the OWL perspective is shown on the right. The interface is in the middle. If the atom* `holds(stackBot, blockA)` *becomes true in the planner perspective, this is reflected in the ontology perspective as an OWL axiom expressing a corresponding relation between the two individuals* stackBot *and* blockA*. Using the static ontology, we can infer that* stackBot *is an instance of the OWL class* FullHands*, because the* holds *relation is true for two different blocks. This is reflected by the entailed OWL axiom* FullHands(stackBot)*. We also have a query predicate* `fullHands`*, which corresponds to a query over instances of the OWL class* FullHands*. Since we can infer from the OWL perspective that* stackBot *is an instance*

```
OBJECT    stackBot   -> stackBot
OBJECT    blockA     -> blockA
OBJECT    blockB     -> blockB
OBJECT    blockC     -> blockC

PREDICATE holds(_,_) -> holds
```

(a) Example of a fluent interface.

```
PREDICATE: fullHands
VARIABLES: ?r
TYPE_SPECIFICATION:
   Robot(?r)
QUERY:
   FullHands(?r)
```

(b) Example of a query specification.

Figure 2: Interface specification using the syntax of our implementation.

of FullHands*, the atom* `fullHands(stackBot)` *becomes true in the planner perspective of the state.*

The central notion of this paper is that of an *ontology-mediated planning specification*, which consists of the following three components:

1. the *PDDL component* **P**, which is a PDDL planning specification consisting of a domain and a problem,
2. the *static ontology* $\mathcal{O}$, which is an OWL ontology specifying the static knowledge, that is, it contains axioms whose truth cannot be affected by actions, and
3. the *interface* that specifies how the two perspectives of an ontology-enhanced state should be linked. The interface itself consists of two parts:
   a) the fluent interface, and
   b) the query interface.

The *fluent interface* maps objects, unary and binary predicates used in the planner perspective to the named individuals, OWL classes and OWL properties that are used in the OWL perspective. An example of how this looks like for our implementation is shown in Figure 2a. In the context of this paper, it is convenient to see the fluent specification simply as a partial function F that assigns to some of the predicates and objects X in the planning specification an IRI $F(X)$. We require F to be inverse functional, that is, F^{-} is also a function. We lift F in a straight-forward way to atoms by setting $F(P(t_1, \ldots, t_n)) = F(P)\big(F(t_1), \ldots, F(t_n)\big)$ if it is defined.

The *query interface* is a set of *query specifications* $S = \langle p_S, V_S, T_S, Q_S \rangle$, which each consist of four components:

1. p_S is the *query predicate*,
2. V_S is a vector of *query variables*, whose number corresponds to the arity of p_S,
3. the *type specification* T_S assigns to each variable $x \in V_S$ an OWL class expression specifying its *static type*, and
4. the *query* Q_S is a set of OWL axioms using variables from V_S as place holders for individual names.

An example of how this looks like for our implementation is shown in Figure 2b. Note that in the type specification, we can only assign one class expression to each variable, while variables may occur in arbitrary ways in the query. The static types are used to restrict the set of named individuals that can be assigned to a variable: candidates for a variable $x \in V_S$ are individual names a for which the static ontology entails $a : T_S(x)$, that is, which are an instance of the class expression assigned to x via T_S. For the specification in Figure 2b, $V_s = (?\mathrm{r})$ and ?r can be associated with instances of the class Robot. For a given static ontology $\mathcal{O}$ and query specification S, we thus have a set $\Theta(S, \mathcal{O})$ of *legal assignments* $\theta : V_S \to \mathsf{Ind}(\mathcal{O})$ of variables to individual names in $\mathcal{O}$. Finally, Q_S specifies the OWL query that the query predicate p_S stands for. For a given assignment $\theta \in \Theta(S, \mathcal{O})$, $\theta(Q_S)$ denotes the set of OWL axioms obtained by replacing each variable $x \in V_S$ in Q_S by $\theta(x)$. In the present example, for the assignment $\theta(?\mathrm{r}) = \mathsf{stackBot}$, we would have $\theta(Q_S) = \{\ \mathsf{fullHands(stackBot)}\ \}$.

We have now all ingredients to define ontology-mediated planning specifications.

Definition 2. *An* ontology-mediated planning specification *is a tuple $\langle \mathbf{P}, \mathcal{O}, F, \mathbf{S}\rangle$, where* ***P*** *is a PDDL planning specification consisting of a planning domain and a planning problem, $\mathcal{O}$ is an OWL ontology called the* static ontology*, F is a fluent interface, and $\mathbf{S}$ is a set of query specifications called the* query interface*.*

An ontology-mediated planning specification determines when an ontology-enhanced state is compatible for that specification. In particular, a state $q = \langle P_q, \mathcal{O}_q\rangle$ is *compatible* to an ontology-mediated planning specification $\mathbf{OP} = \langle \mathbf{P}, \mathcal{O}, F, \mathbf{S}\rangle$, where $\mathcal{D}$ are the derivation rules in $\mathbf{P}$, iff:

C1 P_q is a set of atoms over predicates and constants occurring in $\mathbf{P}$,

C2 $\mathcal{O} \subseteq \mathcal{O}_q$ (the static ontology is always part of the OWL perspective),

C3 for every atom $\alpha \in \mathcal{D}(P_q)$ for which $F(\alpha)$ is defined, $F(\alpha) \in \mathcal{O}_q$

C4 $\mathcal{O}_q$ contains no axioms that are not required due to Conditions **C2** and **C3**

C5 for every query specification $S = \langle p_S, \langle x_1, \ldots, x_n\rangle, T_S, Q_S\rangle \in \mathbf{S}$ and $\theta \in \Theta(S, \mathcal{O})$, if $F^-(\theta(x_i))$ is defined for each variable x_i and $\mathcal{O}_q \models \theta(Q_S)$, then

$$p_S(F^-(\theta(x_1)), \ldots, F^-(\theta(x_n))) \in P_q.$$

Given an ontology-mediated planning specification $\mathbf{OP} = \langle \mathbf{P}, \mathcal{O}, F, \mathbf{S}\rangle$ and a state P in the corresponding planning domain, we define the *extension ext*$(P, \boldsymbol{OP})$ *of P according to* ***OP*** as follows. Let 1) P' be the set of atoms in P that are not over query predicates, 2) $\mathcal{O}_q$ the set of axioms required to satisfy Conditions **C2** and **C3** based on the atoms in P', and 3) P_q the extension of P' by all atoms over query predicates that are required to satisfy Condition **C5** for the ontology $\mathcal{O}_q$. Then, $\mathrm{ext}(P, \mathbf{OP}) = \langle P_q, \mathcal{O}_q\rangle$.

Example 2. *Consider the example in Figure 1 where $\alpha =$ `holds(stackBot, blockA)` and $\beta =$ `holds(stackBot, blockB)` with $\alpha, \beta \in P_q$ and F is defined as in Figure 2a and* S *as in Figure 2b. Then, according to* **C3***, the axioms from the mappings $F(\alpha) =$* holds(stackBot, blockA) *and* $F(\beta) = \text{holds(stackBot, blockB)}$ *are part of* $\mathcal{O}_q$*. Using the static part of* $\mathcal{O}_q$*, which states that* stackBot *is a PR2 robot and* blockA *is different from* blockB*, we can infer that* $\mathcal{O}_q \models \{\text{FullHands(stackBot)}\}$*. Using* $\theta = \{(?\mathrm{r} \mapsto \text{stackBot})\}$*,* F *and* S *from Figure 2b, we can apply **C5** to determine that* `fullHands(stackBot)` $\in P_q$*.*

It remains to define the semantics of actions and plans on ontology-mediated planning specifications. Fix an ontology-mediated planning specification $\mathbf{OP} = \langle \mathbf{P}, \mathcal{O}, F, \mathbf{S} \rangle$. Let a be a ground action with precondition pre and effect eff $= \langle \text{add}, \text{del} \rangle$. Let q be an ontology-enhanced state. We say that a is *applicable* on q iff $\mathcal{D}(P_q) \models$ pre. The result of *applying* a *on* q is then denoted by $q(a)$ and defined as $q(a) = \text{ext}(P_q(a), \mathbf{OP})$. We can now define *plans for* ***OP*** similarly as we did for planning specifications: Namely, a plan is a sequence $a_1 \ldots a_n$ of actions that generates a sequence $q_0 q_1 \ldots q_n$ of ontology-enhanced states s.t.

1. $q_0 = \text{ext}(I, \mathbf{OP})$, where I is the initial state of the PDDL planning problem in $\mathbf{OP}$,
2. for each $i \in \{1, \ldots, n\}$, $q_i = q_{i-1}(a_i)$,
3. for each $i \in \{1, \ldots, n\}$, a_i is applicable on q_{i-1}, and
4. $\mathcal{D}(P_{q_n}) \models G$, where $\mathcal{D}$ are the derivation rules of the planning domain, and G is the formula describing the goal of the planning problem.[1]

4. Solving Ontology-Mediated Planning Problems in Practice

Semantically, our approach is very related to that of eKABs introduced in [19]. eKABs do not offer a differentiation between OWL perspective and planner perspective. Instead, actions operate directly on OWL axioms, which can be directly referenced to both pre-conditions and post-conditions of the actions. We conjecture that it is always possible using simple transformations to translate an eKAB with a finite domain into an ontology-mediated planning problem. In the other direction, we can translate ontology-mediated planning problems into eKABs by replacing atom predicates by the corresponding OWL class and OWL properties, and replacing query atoms by the corresponding queries. It is thus in theory possible to use an eKAB planner to compute plans for ontology-mediated planning problems. However, existing implementations for eKAB planning have limitations regarding the supported OWL fragment. The general idea of these approaches is to take the eKAB planning specification, and translate it into a PDDL specification that can then be used by a standard PDDL planner. Those techniques focus on the *planning domain*, that is, the obtained rewritings are independent of the planning problem. The approach presented in [19, 35] only supports *rewritable DLs*, which would correspond to the OWL fragment OWL-QL. The approach presented in [23] goes further by using derivation rules, which allows to encode Horn-$\mathcal{ALCHOIQ}$ via a known translations of such ontologies into datalog programs. Horn-$\mathcal{ALCHOIQ}$ roughly corresponds to the Horn fragment of OWL

[1] Note that we allow the plan to go through states whose OWL perspective is inconsistent. If this is not wanted, an easy way to avoid this would for example be to use a query predicate to detect such states, and to adapt the preconditions of all actions so that they are not applicable in inconsistent states. As a consequence, a goal state can never be reached from such a state.

$$\texttt{inconsistent} \leftarrow \begin{pmatrix} & \texttt{holds(stackBot, blockA)} \\ \wedge & \texttt{holds(stackBot, blockB)} \\ \wedge & \texttt{holds(stackBot, blockC)} \end{pmatrix}$$

$$\begin{aligned} \texttt{fullHands(stackBot)} \leftarrow & \texttt{inconsistent} \vee \\ & \begin{pmatrix} & (\texttt{holds(stackBot, blockA)} \wedge \texttt{holds(stackBot, blockB)}) \\ \vee & (\texttt{holds(stackBot, blockA)} \wedge \texttt{holds(stackBot, blockC)}) \\ \vee & (\texttt{holds(stackBot, blockB)} \wedge \texttt{holds(stackBot, blockC)}) \end{pmatrix} \end{aligned}$$

Figure 3: Computed Derivation rules resulting for the example from Figure 1.

DL. For DLs that are not Horn, a translation into datalog is generally not possible, since datalog is itself a Horn logic. The same applies to rewriting into derivation rules, if those are supposed to be defined independently of the objects of the planning problem. Therefore, in order to support full OWL DL, we need to take into account also the planning problem. Specifically, our approach directly iterates over the possible assignments for each query predicate. This allows us to develop a more generic approach that does not restrict the ontology language, as long as a reasoner for it is available.

The basic idea is to construct a derivation rule for each query predicate, which determines for each valid variable assignment a set of conditions that can be evaluated directly on the planner perspective of a state. The details on how we construct these derivation rules in practice can be found in the extended version of this paper [36].

Example 3. *Figure 3 depicts the generated derived predicates for our running example. We introduce the atom* `inconsistent`*, which captures the states in the ontology perspective that are inconsistent. The atom is used in the derivation rule for every query atom. In our example, the static ontology states that every individual from the class* PR2 *is only allowed to hold at most two blocks. Using the fluent interface, we can determine the combination of atoms in the planning perspective that would lead to an ontology that would violate this constraint. There is only one derivation rule for the query-predicate* `fullHands` *as the only possible variable mapping is* $?\mathrm{r} = \texttt{stackBot}$ *because* stackBot *is the only individual with the static type* Robot. *The query atom is true if the OWL perspective is inconsistent or if the* stackBot *holds exactly two different blocks.*

5. Evaluation

Implementation. We implemented our method of compiling ontology-mediated planning specifications into PDDL specification with derivation rules.[2] We use the standard formats PDDL and TTL for the planning specification and the ontology respectively, and we use our own text-based formats for the fluent and query interface. Our compilation algorithm relies on an extensive computation of justifications, for which we used a modified version of the blackbox justification algorithm implemented in the OWL-API [38], together with the OWL reasoning

[2]The source files and scripts to reproduce the evaluation can be obtained online [37].

Table 1
Results grouped by domain. Computation times are in seconds and the median over the commonly solved instances (except times marked with "*", those refer to the median for the instances solved by this method). Best results are marked in bold (where applicable).

Domain	#	# axioms in T-Box	# solved HORN	# solved OM	# compiled HORN	# compiled OM	planning time HORN	planning time OM	compilation time HORN	compilation time OM
Drones	24	17	**24**	4	**24**	4	8.2	**4.1**	**0.6**	917.5
Queens	30	17	**24**	19	**30**	20	3.0	**0.8**	**0.7**	146.5
Pipes	21	43	14	**19**	**21**	**21**	5.4	**0.2**	**0.7**	10.0
Blocksworld	21	5	—	13	—	21	—	4.8*	—	1.1*

system HermiT [39]. The computed derivation rules are added to the PDDL domain. We used the fast-downward planning system [22] with the heuristic A* for planning. We chose this heuristic because many of the more advanced heuristics have problems working with derivation rules.

For our evaluation, we compare our method to the eKAB method presented in [23]. We choose this competitor as it is the implementation that can deal with the most expressive DL fragment and performs best on existing benchmark domains [23]. We used a time limit of 1200s and a memory limit of 8GB. Both limits applied to compiling and planning individually.

Benchmarks. Our benchmark consists of instances from the domains used in [23], as well as some new domains. As to be expected, our method is at this stage not yet competetive on all domains, and in fact, on some of the domains used in [23] to evaluate the performance of eKABs based on rewritable and Horn DLs, our method almost always timed out. To have a more interesting picture, we focus here on the more complex domains from that paper ("Drones" and "Queens"), which surprisingly turned out also to be the more interesting ones for our approach, and present the other results in the extended version of this paper [36]. In particular, our benchmark set contained 54 instances from two of the most complex domains that were introduced in [23], to which we added two new domains with 39 instances. The existing instances are eKABs, which are based on the DL fragment Horn-$\mathcal{ALCHOIQ}$. We translated them manually to ontology-mediated planning specifications, which mainly involved specifying the interface. The domains "Pipes" and "Blocksworld" were created by us. "Pipes" is a complex domain describing a mission for an underwater robot in a 2D world. The world contains pipes, valves and tanks that can be connected to each other and that are located at different waypoints. The goal is to document damages of the pipe and to turn the valves such that no tank is connected to a damaged pipe segment. "Blocksworld" reflects the domain from our running example (see e.g. Example 1). It is inspired by the Blocksworld domain from the international planning competition 2000 [40]. This domain uses axioms that can not be captured by Horn-$\mathcal{ALCHOIQ}$.

Results. Table 1 provides a summary of our experiments. We call the method presented in this paper OM and the method presented in [23] HORN. OM was capable of handling some of the domains very well, while the performance on others is worse than HORN.

In general, OM had longer compilation times and shorter planning times compared to Horn. This simpler structure of the derivation rules generated by OM resulted in a faster search in the planning phase as each state could be evaluated faster, e.g. in the domain “Pipes” the planner could, on average, evaluate 11,000 states per second for Horn and 117,000 states per second for OM. Therefore, we expected OM to outperform Horn in cases where the planner needs to search in a huge state spaceand the number of fluents and queries is low. This is e.g. the case for the larger instances from the domain “Pipes”, which could be solved by OM but not by Horn.

The size of the ontology is in general not a problem for OM as the domain “Pipes”, which contains a larger T-Box than the other domains, could be compiled in rather short time. Similarly, increasing the expressiveness of the underlying DL does not seem have a negative effect, as all instances from the domain “Blocksworld” could be compiled within the provided bounds.

On the other hand, as the domain “Drones” shows, the performance on instances from the existing domains is often poor. As mentioned before, this picture was even worse with the other benchmarks from [23], on which our method almost always caused a timeout. One reason is that we need to map every atom from the planning perspective to the OWL perspective to describe equivalent instances to the eKAB instances. This results in many, often several hundred, fluents which again results in many explanations for an inconsistent ontology. As OM enumerates all the possibilities in the derivation rules, this is a problem and leads to a huge increase in compilation time. The detailed evaluation in the extended version of this paper shows that this can happen even in relatively small instances [36].

6. Conclusion

We proposed ontology-mediated planning specifications as a way to integrate OWL reasoning into planning. One objective was to find a formalism that allows for a separation of concerns, allowing to separate the specification of ontologies from the specification of planning problems and domains. This has the advantage that the ontology can be maintained by ontology experts, while the planning specification can be developed by planning experts, with the interface serving as the only connecting component. We developed a first practical method for computing plans for such planning problems, which relies on justifications. This technique allows us to be flexible with respect to the ontology language, with the result that our method supports the entirety of OWL DL, going beyond what is currently supported by implementations for the related frameworks of KABs and eKABs. Our evaluation shows that our method can outperform existing methods on some instances but is not competitive for most existing benchmark domains yet. In the future, we want to investigate optimizations of our approach, maybe combining it ideas of the other rewriting-based approaches, in order to obtain shorter compilation times.

Acknowledgments

Tobias John is part of the project REMARO that has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956200. Patrick Koopmann is supported by the German Research Foundation (DFG), grant 389792660 as part of TRR 248 – CPEC.

References

[1] A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Goncalves, M. K. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas, J. Olszewska, H. Nakawala, E. Pignaton, A. Gyrard, S. Borgo, G. Alenyà, M. Beetz, H. Li, A review and comparison of ontology-based approaches to robot autonomy, The Knowledge Engineering Review 34 (2019/ed). doi:10.1017/S0269888919000237.

[2] E. Karpas, D. Magazzeni, Automated planning for robotics, Annual Review of Control, Robotics, and Autonomous Systems 3 (2020) 417–439. doi:10.1146/annurev-control-082619-100135.

[3] A. Sahoo, S. K. Dwivedy, P. S. Robi, Advancements in the field of autonomous underwater vehicle, Ocean Engineering 181 (2019) 145–160. doi:10.1016/j.oceaneng.2019.04.011.

[4] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, M. Carreras, ROSPlan: Planning in the robot operating system, Proceedings of the International Conference on Automated Planning and Scheduling 25 (2015) 333–341. doi:10.1609/icaps.v25i1.13699.

[5] W. Dargie, Eldora, J. Mendez, C. Möbius, K. Rybina, V. Thost, A. Turhan, Situation recognition for service management systems using OWL 2 reasoners, in: IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM, IEEE Computer Society, 2013, pp. 31–36. doi:10.1109/PerComW.2013.6529452.

[6] X. Li, S. Bilbao, T. Martín-Wanton, J. Bastos, J. Rodriguez, SWARMs ontology: A common information model for the cooperation of underwater robots, Sensors 17 (2017) 569. doi:10.3390/s17030569.

[7] Z. Zhai, J.-F. Martínez Ortega, N. Lucas Martínez, P. Castillejo, A rule-based reasoner for underwater robots using OWL and SWRL, Sensors 18 (2018) 3481. doi:10.3390/s18103481.

[8] Y. Gil, Description logics and planning, AI Magazine 26 (2005) 73–73. doi:10.1609/aimag.v26i2.1814.

[9] S. Balakirsky, Z. Kootbally, T. Kramer, A. Pietromartire, C. Schlenoff, S. Gupta, Knowledge driven robotics for kitting applications, Robotics and Autonomous Systems 61 (2013) 1205–1214. doi:10.1016/j.robot.2013.04.006.

[10] Z. Kootbally, C. Schlenoff, C. Lawler, T. Kramer, S. K. Gupta, Towards robust assembly with knowledge representation for the planning domain definition language (PDDL), Robotics and Computer-Integrated Manufacturing 33 (2015) 42–55. doi:10.1016/j.rcim.2014.08.006.

[11] M. Klusch, A. Gerber, M. Schmidt, Semantic web service composition planning with OWLS-Xplan., in: T. R. Payne, V. A. M. Tamma (Eds.), Agents and the Semantic Web, Papers from the 2005 AAAI Fall Symposium, AAAI Press, 2005, pp. 55–62. URL: https://www.aaai.org/Library/Symposia/Fall/2005/fs05-01-008.php.

[12] Z. Ďurčík, J. Paralič, Transformation of ontological represented web service composition problem into a planning one, Acta Electrotechnica et Informatica 11 (2011). doi:10.2478/v10198-011-0014-y.

[13] Y. Al-Safi, V. Vyatkin, An ontology-based reconfiguration agent for intelligent mechatronic

systems, in: V. Mařík, V. Vyatkin, A. W. Colombo (Eds.), Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2007, pp. 114–126. doi:10.1007/978-3-540-74481-8_12.

[14] H. Louadah, E. Papadakis, T. L. McCluskey, G. Tucker, P. Hughes, A. Bevan, Translating ontological knowledge to PDDL to do planning in train depot management operations, 36th Workshop of the UK Planning and Scheduling Special Interest Group (2021).

[15] S. Borgo, A. Cesta, A. Orlandini, A. Umbrico, Knowledge-based adaptive agents for manufacturing domains, Engineering with Computers 35 (2019) 755–779. doi:10.1007/s00366-018-0630-6.

[16] M. Milicic, Complexity of planning in action formalisms based on description logics, in: Proc. of LPAR 2007, volume 4790 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 408–422. doi:10.1007/978-3-540-75560-9_30.

[17] B. Zarrieß, J. Claßen, Verification of knowledge-based programs over description logic actions, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI, AAAI Press, 2015, pp. 3278–3284. URL: http://ijcai.org/Abstract/15/462.

[18] B. B. Hariri, D. Calvanese, M. Montali, G. D. Giacomo, R. D. Masellis, P. Felli, Description logic knowledge and action bases, Journal of Artificial Intelligence Research 46 (2013) 651–686. doi:10.1613/jair.3826.

[19] D. Calvanese, M. Montali, F. Patrizi, M. Stawowy, Plan synthesis for knowledge and action bases, Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI, AAAI Press, Palo Alto, California, 2016, p. 8. URL: http://www.ijcai.org/Abstract/16/149.

[20] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL the planning domain definition language, Technical Report (1998).

[21] J. Hoffmann, S. Edelkamp, The deterministic part of IPC-4: An overview, Journal of Artificial Intelligence Research 24 (2005) 519–579. doi:10.1613/jair.1677.

[22] M. Helmert, The fast downward planning system, Journal of Artificial Intelligence Research 26 (2006) 191–246. doi:10.1613/jair.1705.

[23] S. Borgwardt, J. Hoffmann, A. Kovtunova, M. Krötzsch, B. Nebel, M. Steinmetz, Expressivity of planning with horn description logic ontologies, Proceedings of the AAAI Conference on Artificial Intelligence 36 (2022) 5503–5511. doi:10.1609/aaai.v36i5.20489.

[24] B. Parsia, B. Motik, P. Patel-Schneider, OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition), W3C Recommendation, W3C, 2012. URL: http://www.w3.org/TR/owl2-syntax/.

[25] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A semantic web rule language combining OWL and RuleML, W3C Member submission 21 (2004) 1–31. URL: http://www.w3.org/Submission/SWRL/.

[26] C. Dubslaff, P. Koopmann, A.-Y. Turhan, Enhancing probabilistic model checking with ontologies, Formal Aspects of Computing (2021). doi:10.1007/s00165-021-00549-0.

[27] C. Dubslaff, P. Koopmann, A. Turhan, Ontology-mediated probabilistic model checking, in: W. Ahrendt, S. L. T. Tarifa (Eds.), Integrated Formal Methods - 15th International Conference, IFM, volume 11918 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 194–211. doi:10.1007/978-3-030-34968-4_11.

[28] I. Gocev, S. Grimm, T. A. Runkler, Explanation of action plans through ontologies,

in: H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, R. Meersman (Eds.), On the Move to Meaningful Internet Systems. OTM Conferences, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 386–403. doi:10.1007/978-3-030-02671-4_24.

[29] M. Horridge, Justification based explanation in ontologies, Ph.D. thesis, University of Manchester, UK, 2011. URL: http://www.manchester.ac.uk/escholar/uk-ac-man-scw:131699.

[30] T. John, P. Koopmann, Planning with ontology-enhanced states using problem-dependent rewritings, in: Planning and Ontology Workshop (PLATO), ICAPS, CEUR Workshop Proceedings, CEUR-WS.org, 2023. To appear.

[31] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017.

[32] V. Lifschitz, On the semantics of STRIPS, in: M. Georgeff, Lansky, Amy (Eds.), Reasoning about Actions and Plans, Morgan Kaufmann, San Mateo, CA, 1987, pp. 1–9.

[33] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, Journal of Artificial Intelligence Research 20 (2003) 61–124. doi:10.1613/jair.1129.

[34] J. Bohren, R. B. Rusu, E. Gil Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mösenlechner, W. Meeussen, S. Holzer, Towards autonomous robotic butlers: Lessons learned with the PR2, in: IEEE International Conference on Robotics and Automation, 2011, pp. 5568–5575. doi:10.1109/ICRA.2011.5980058.

[35] S. Borgwardt, J. Hoffmann, A. Kovtunova, M. Steinmetz, Making DL-Lite planning practical, in: M. Bienvenu, G. Lakemeyer, E. Erdem (Eds.), Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning, Hanoii, Vietnam, 2021, pp. 641–645. doi:10.24963/kr.2021/61.

[36] T. John, P. Koopmann, Towards ontology-mediated planning with owl dl ontologies (extended version), 2023. doi:10.48550/arXiv.2308.08200.

[37] T. John, P. Koopmann, Supplementary Material for the paper "Towards Ontology-Mediated Planning with OWL DL Ontologies", 2023. doi:10.5281/zenodo.8225315.

[38] M. Horridge, S. Bechhofer, The OWL API: A Java API for OWL ontologies, Semantic Web 2 (2011) 11–21. doi:10.3233/SW-2011-0025.

[39] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: An OWL 2 reasoner, Journal of Automated Reasoning 53 (2014) 245–269. doi:10.1007/s10817-014-9305-1.

[40] F. Bacchus, AIPS 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems, Ai magazine 22 (2001) 47–47.