Making DL-Lite Planning Practical (Abstract)*

Stefan Borgwardt[1], Jörg Hoffmann[2], Alisa Kovtunova[1], and Marcel Steinmetz[2]

[1] Institute of Theoretical Computer Science, TU Dresden, Germany
firstname.lastname@tu-dresden.de
[2] Saarland University, Saarland Informatics Campus, Germany
lastname@cs.uni-saarland.de

AI planning is a well-investigated framework for describing the evolution of system states through actions [4]. Action preconditions are first-order logic formulas, which are evaluated over states, i.e. finite sets of facts. Action effects either add or remove facts from the current state. Formulas are evaluated using a closed-domain, closed-world semantics, i.e. the domain is fixed a priori and facts that are not contained in a state are assumed to be false. The aim is to find a *plan*, i.e. a sequence of (grounded) actions that satisfy a goal formula.

Knowledge representation formalisms are a natural way to introduce global constraints on permissible states; however, they usually interpret first-order formulas over arbitrary models, instead of just one model with a fixed domain. *DL-Lite Explicit-Input Knowledge and Action Bases (eKABs)* [3] were proposed to combine classical planning with state axioms formulated in DL-Lite [2], while allowing the creation of an unbounded number of new objects and interpreting action preconditions under open-world semantics. Due to first-order rewritability of DL-Lite, eKABs can be translated into the classical planning language PDDL. In theory, this allows the use of off-the-shelf planning systems. However, an initial evaluation [3, 10] using the Fast Downward (FD) planning platform [5] showed poor performance on a simple hand-crafted domain, with the planner being unable to solve even trivial problem instances. In our experiments, the problem also appeared in Fast Forward (FF) [7].

Our Contribution We find that the bottleneck lies in the DNF transformations used to compile the PDDL tasks into the planners' internal (grounded) representations. These naïve transformations are applied in-situ on complex formulas, causing a worst-case exponential blow-up on non-DNF input. Here we investigate two PDDL pre-compilations that enable polynomial DNF transformations.

The first pre-compilation (Ne), proposed by Nebel [9], uses auxiliary predicates to represent each (already grounded) complex sub-formula ϕ: P_ϕ represents the truth value of ϕ in the current state; and P'_ϕ is designed to be true iff ϕ has already been evaluated. Additional ground actions are introduced to determine the value of P_ϕ provided that P'_ψ is true for all sub-formulas ψ of ϕ. This evaluation is repeated after every regular action, which increases the plan length.

We therefore propose a second pre-compilation (DP) that avoids this overhead by employing PDDL *derived predicates* [6, 11], which specify a rule-based update

* Abstract of a short paper accepted at KR 2021. This work is supported by the DFG grant 389792660 (TRR 248) (see https://perspicuous-computing.science).

of auxiliary predicates that is applied at every planning state. Every complex subformula ϕ is replaced by a new predicate P_ϕ that obeys the update rule $P_\phi \leftarrow \phi$. This results in a set of rules that is equivalent to a non-recursive, and therefore stratified, Datalog program with negation [1].

Table 1. Per-domain aggregated statistics. (a) Number of instances solved within resource limits. (b) Number of instances that passed the planners' PDDL pre-processing.

		(a) # solved						(b) # PDDL processed					
		FF			FD			FF			FD		
Domain	#	O	Ne	DP	O	Ne	DP	O	Ne	DP	O	Ne	DP
Robot	20	**20**	1	**20**	4	1	**20**	**20**	**20**	**20**	4	**20**	**20**
TaskAssign	20	1	1	15	3	1	**20**	1	10	15	3	10	**20**
Cats	20	14	14	**20**	14	11	**20**	14	**20**	**20**	14	**20**	**20**
Elevator	20	12	0	**20**	**20**	0	**20**	12	**20**	**20**	**20**	**20**	**20**
TPSA	15	7	5	5	**14**	4	5	7	5	5	**14**	4	5
VTA	15	**15**	6	**15**	**15**	4	13	**15**	6	**15**	**15**	4	13
VTA-Roles	15	5	4	5	**15**	0	5	6	4	5	**15**	0	5
Assembly	30	0	0	24	**30**	0	**30**	9	10	24	**30**	4	**30**
GridPlacement	20	5	1	17	6	2	**20**	5	**20**	**20**	6	**20**	**20**
Miconic	30	9	3	**13**	9	2	9	11	27	19	14	18	**30**
$\sum$	205	88	35	154	130	25	**162**	100	142	163	135	120	**183**

Experiments We compare the pre-compilations against the original PDDL files (O) using the planners FF and FD 20.06. We developed a set of benchmarks, comprising the previous eKAB tasks [3,10] (Robot, TaskAssign), new eKAB domains (Cats, Elevator), and benchmarks adapted from prior work on planning with propositional background ontologies [8] (TPSA, VTA). We also investigate existing PDDL benchmark domains with minor modifications (Assembly, Miconic), as our pre-compilation may be useful on any planning domain with complex preconditions, and a showcase domain (GridPlacement) enforcing challenging DNF transformations. The benchmarks[3] and pre-compilers[4] are available online.

As indicated by the O columns in Table 1, PDDL processing indeed constitutes the main bottleneck: In almost every unsolved instance, the planners failed already during PDDL processing. Part (b) shows that DP reduces the overhead of PDDL input handling and thereby increases overall performance in nearly every domain. TPSA and VTA are the only exceptions were the pre-compilation turned out detrimental. Besides an absence of complex conditions, in these domains the actions assign predicates to previously unbound objects. Therefore, by increasing the number of objects, the grounding and translation sizes grow drastically.

In contrast, the number of instances that could be solved by FF and FD after the Ne pre-compilation drops substantially in every domain. Apart from a blow-up of the file size, the Ne pre-compiler obfuscates the planning task's original structure, which leads both planners' searches into serious troubles.

In future work, we plan to design tools tailored to eKABs over DL-Lite and more expressive ontology languages, leveraging e.g. PDDL derived predicates.

[3] `https://gitlab.perspicuous-computing.science/m.steinmetz/pddl-dllite-benchmarks.git`

[4] `https://gitlab.perspicuous-computing.science/a.kovtunova/moreflags2.git`

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Dl-Lite: Tractable description logics for ontologies. In: Veloso, M.M., Kambhampati, S. (eds.) Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference. pp. 602–607. AAAI Press / The MIT Press (2005), http://www.aaai.org/Library/AAAI/2005/aaai05-094.php
3. Calvanese, D., Montali, M., Patrizi, F., Stawowy, M.: Plan synthesis for knowledge and action bases. In: Kambhampati, S. (ed.) Proc. of the 25th Int. Joint Conf. on Artificial Intelligence (IJCAI'16). pp. 1022–1029. AAAI Press (2016), https://www.ijcai.org/Abstract/16/149
4. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
5. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research **26**, 191–246 (2006). https://doi.org/10.1613/jair.1705, https://doi.org/10.1613/jair.1705
6. Hoffmann, J., Edelkamp, S.: The deterministic part of IPC-4: An overview. Journal of Artificial Intelligence Research **24**, 519–579 (2005). https://doi.org/10.1613/jair.1677
7. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14**, 253–302 (2001). https://doi.org/10.1613/jair.855
8. Hoffmann, J., Weber, I., Scicluna, J., Kacmarek, T., Ankolekar, A.: Combining scalability and expressivity in the automatic composition of semantic web services. In: 8th International Conference on Web Engineering (ICWE'08) (2008). https://doi.org/10.1109/ICWE.2008.8
9. Nebel, B.: On the compilability and expressive power of propositional planning formalisms. Journal of Artificial Intelligence Research **12**, 271–315 (2000). https://doi.org/10.1613/jair.735
10. Stawowy, M.: Plan Synthesis in Explicit-input Knowledge and Action Bases. Ph.D. thesis, IMT School for Advanced Studies Lucca, Italy (2016). https://doi.org/10.6092/imtlucca/e-theses/208
11. Thiebaux, S., Hoffmann, J., Nebel, B.: In defense of PDDL axioms. Artificial Intelligence **168**(1–2), 38–69 (2005). https://doi.org/10.1016/j.artint.2005.05.004

Making DL-Lite Planning Practical

Stefan Borgwardt[1] , **Jörg Hoffmann**[2] , **Alisa Kovtunova**[1] , **Marcel Steinmetz**[2]
[1]Institute of Theoretical Computer Science, TU Dresden, Germany
[2]Saarland University, Saarland Informatics Campus, Germany
{stefan.borgwardt,alisa.kovtunova}@tu-dresden.de, {hoffmann,steinmetz}@cs.uni-saarland.de

Abstract

Planning in the presence of background ontologies is a topic of long-standing interest in AI. It combines the problems of (1) belief update complexity and (2) state space combinatorics. DL-Lite offers an attractive solution to (1), with belief updates possible at the ABox level. Indeed, it has been shown that DL-Lite planning can be compiled into the commonly used planning language PDDL. Yet that compilation was previously found to be infeasible for off-the-shelf planning systems. Here we analyze the reasons for this problem, and find that the bottleneck lies in the planner pre-processes, in particular in the naïve DNF transformations used to compile the PDDL input into the planners' internal representations. Consequently we design a PDDL pre-compiler realizing a polynomial DNF transformation. We leverage a particular PDDL language feature ("derived predicates") to avoid the need for excessive control structure. Our pre-compiler turns out to be quite effective: the previous bottleneck disappears, and experiments on a broad range of benchmarks demonstrate the first practical technology for DL-Lite planning.

1 Introduction

AI planning is a well-investigated framework for describing the evolution of system states through the application of actions (Ghallab, Nau, and Traverso 2004). Action preconditions are first-order logic formulas, which are evaluated over states, i. e. finite sets of facts. Action effects either add or remove facts from the current state. Formulas are evaluated using a closed-domain, closed-world semantics, i. e. the domain is fixed a priori and facts that are not contained in a state are assumed to be false. Knowledge representation formalisms are a natural way to introduce global constraints on permissible states. However, they usually interpret first-order formulas over arbitrary models with arbitrary domains, instead of just one model with a fixed domain.

Early work on open-world, open-domain formalisms for constraining possible states quickly noticed the so-called ramification problem (Ginsberg and Smith 1988), i. e. an action that makes a fact true also has to take care of satisfying the state axioms, possibly requiring to add or remove other facts, which may also involve new objects. One approach to deal with this problem is to not view states as finite interpretations, but as sets of formulas interpreted in an open-world fashion. In this way, implicit knowledge about the world added by the state axioms does not have to be made explicit in the states themselves. This means that action preconditions are also evaluated under open-world semantics, while action effects do not directly modify a single interpretation, but instead operate on a set of formulas.

DL-Lite Explicit-Input Knowledge and Action Bases (eKABs) (Calvanese et al. 2016) combine classical planning with state axioms formulated in the description logic DL-Lite (Calvanese et al. 2005), while allowing the creation of an (a priori) unbounded number of new objects and interpreting action preconditions under open-world semantics. A translation of eKABs into classical PDDL planning problems was proposed, which in theory allows the use of off-the-shelf planning systems. However, an initial evaluation (Calvanese et al. 2016; Stawowy 2016) using the Fast Downward (FD) planning platform (Helmert 2006) showed poor performance on a simple hand-crafted domain, with the planner being unable to solve even trivial problem instances.

Here we analyze the reasons for this problem, which appears also in Fast Forward (FF) (Hoffmann and Nebel 2001). We find that the bottleneck lies in the DNF transformations used to compile the PDDL input into the planners' internal representations. These naïve transformations are applied in-situ without introducing auxiliary predicates, causing a worst-case exponential blow-up on non-DNF input. Here we consider PDDL pre-compilations that enable polynomial DNF transformations. However, in a straightforward approach (Nebel 2000), the introduction of auxiliary predicates and actions comes with substantial overhead for truth-value evaluation of sub-formulas during planning, forcing the planner to re-evaluate the relevant formulas in between the applications of regular actions. We show that this overhead can be avoided by employing PDDL *derived predicates* (Hoffmann and Edelkamp 2005; Thiebaux, Hoffmann, and Nebel 2005), which specify a rule-based update of auxiliary predicates that is applied at every planning state.

We contribute a broad set of DL-Lite planning benchmarks, comprising the previous eKAB tasks (Calvanese et al. 2016; Stawowy 2016), some new eKAB domains, as well as benchmarks adapted from prior work formulating semantic web service composition as planning with propositional background ontologies (Hoffmann et al. 2008). Our pre-compiler turns out to be quite effective: the bottleneck disappears, and our experiments demonstrate the first practically viable technology for DL-Lite planning.

2 Background

As our optimizations work on the level of classical planning, we introduce the main relevant notions here, and then shortly illustrate eKABs and the original translation.

Planning Languages.

We consider the "level 1" fragment of PDDL 2.1 (Fox and Long 2003), excluding numeric fluents and temporal actions, but including ADL expressions (Pednault 1989). A *PDDL task* is a tuple $\Pi = \langle \mathcal{P}, \mathcal{A}, \mathcal{O}, I, G \rangle$ consisting of *predicate symbols* $\mathcal{P}$, *action schemas* $\mathcal{A}$, *objects* $\mathcal{O}$, the *initial state* I, and the *goal* G. All sets must be finite. A *fact* is a ground first-order atom formulated using $\mathcal{P}$ and $\mathcal{O}$. I is a set of facts that are true initially, and all facts not contained in I are assumed to be false initially. The goal G is a closed first-order formula over $\mathcal{P}$ and $\mathcal{O}$. An action schema $a \in \mathcal{A}$ is a triple $\langle \overrightarrow{x_a}, \mathsf{pre}_a, \mathsf{eff}_a \rangle$ with *parameters* (variables) $\overrightarrow{x_a}$, the *precondition* pre_a, and a set of *effects* eff_a. pre_a is a first-order formula over $\mathcal{P}$ and $\mathcal{O}$ whose free variables are restricted to $\overrightarrow{x_a}$. Each effect $e \in \mathsf{eff}_a$ is a tuple $\langle \overrightarrow{y_e}, \mathsf{cond}_e, \mathsf{add}_e, \mathsf{del}_e \rangle$ where $\overrightarrow{y_e}$ is a tuple of variables, cond_e is the *effect condition*, a first-order formula over $\mathcal{P}$ and $\mathcal{O}$ whose free variables are in $\overrightarrow{x_a} \cup \overrightarrow{y_e}$, the *add list* add_e is a set of positive first-order literals with free variables only in $\overrightarrow{x_a} \cup \overrightarrow{y_e}$, and the *delete list* del_e is a set of such negative literals. The instantiation of the parameters of an action schema a with objects $\overrightarrow{o}$ from $\mathcal{O}$ yields a *ground action* $a(\overrightarrow{x_a}/\overrightarrow{o})$, also written $\underline{a}$ if $\overrightarrow{o}$ is not important.

A *state* s is an interpretation of $\mathcal{P}$ over the closed-world universe $\mathcal{O}$, represented by convention as the set of facts that are true. Formulas are evaluated on states according to standard first-order semantics, where quantifiers range over the objects $\mathcal{O}$. A ground action $\underline{a}$ is *applicable* in s if $s \models \mathsf{pre}_{\underline{a}}$. The result of this application is the state $s[\![\underline{a}]\!]$ with $p \in s[\![\underline{a}]\!]$ iff (1) there exists an effect $e \in \mathsf{eff}_{\underline{a}}$ with instantiation $\overrightarrow{o_y}$ such that $s \models \mathsf{cond}_{e(\overrightarrow{y_e}/\overrightarrow{o_y})}$ and $p \in \mathsf{add}_{e(\overrightarrow{y_e}/\overrightarrow{o_y})}$, or (2) $p \in s$ and it holds for all effects $e \in \mathsf{eff}_{\underline{a}}$ and instantiations $\overrightarrow{o_y}$ that $s \not\models \mathsf{cond}_{e(\overrightarrow{y_e}/\overrightarrow{o_y})}$ or $\neg p \notin \mathsf{del}_{e(\overrightarrow{y_e}/\overrightarrow{o_y})}$. The application of a sequence of ground actions is defined accordingly. A sequence of ground actions $\pi = \underline{a_1}, \dots, \underline{a_n}$ is a *plan* of Π if π is applicable in I and $I[\![\pi]\!] \models G$.

Our pre-compiler furthermore makes use of the PDDL 2.2 (Hoffmann and Edelkamp 2005) language feature *derived predicates* (a.k.a. *axioms*). The task Π then additionally defines a set $\mathcal{P}_{\text{der}}$ of derived predicates disjoint from $\mathcal{P}$, and a set $\mathcal{R}_{\text{der}}$ of rules r of the form $P_r(\vec{x}) \leftarrow \mathsf{body}_r(\vec{x})$ where body_r is a formula with free variables $\vec{x}$ and $P_r \in \mathcal{P}_{\text{der}}$. PDDL 2.2 prescribes that (1) $\mathcal{P}_{\text{der}}$ may occur in conditions (pre, cond, G, body) but not in effects; and that (2) derived predicates do not occur negated in rule bodies (i. e. the negation normal form of every body_r does not contain negated atoms using $\mathcal{P}_{\text{der}}$). These restrictions could be relaxed in principle (e. g. (2) can be replaced with rule stratification (Thiebaux, Hoffmann, and Nebel 2005)), but the choice in PDDL 2.2 is to focus on a simpler case for practical considerations. The semantics is straightforward: in every state, all derived facts (groundings of $\mathcal{P}_{\text{der}}$) are first set to false, then the rules $\mathcal{R}_{\text{der}}$ are applied up to the (unique) fixed point.

DL-Lite eKABs. The light-weight description logic DL-Lite (Calvanese et al. 2005; Poggi et al. 2008) allows formulating state constraints over unary and binary predicates. A *TBox* is a finite set of axioms $B_1 \sqsubseteq B_2$ or $B_1 \sqsubseteq \neg B_2$, where B_1, B_2 are either unary predicates or of the form $\exists R$ for a binary predicate R; the latter corresponds to the formula $\exists y.R(x, y)$. For example, $\mathsf{Employee} \sqsubseteq \exists \mathsf{worksFor}$ says that every employee needs to work in some department, and $\mathsf{ElectronicEngineer} \sqsubseteq \neg \mathsf{SoftwareDeveloper}$ expresses that electronic engineers cannot be software developers at the same time.

A *DL-Lite eKAB* $\langle \mathcal{P}, \mathcal{A}, \mathcal{O}, \mathcal{T}, I, G \rangle$[1] (Calvanese et al. 2016) extends a PDDL task by a DL-Lite TBox $\mathcal{T}$, a possibly infinite set of objects $\mathcal{O}$, the fact that states (including the initial state) are viewed under the open-world assumption, and a modified syntax for actions $\mathcal{A}$ and the goal G. Action preconditions, effect preconditions, and the goal are now specified as *extended conjunctive queries (ECQs)* (Calvanese et al. 2007; Calvanese et al. 2016), which are a kind of first-oder formulas whose atoms are *conjunctive queries (CQs)*, i. e. existentially quantified conjunctions of atoms. The conjunctive queries are evaluated using the open-world semantics of DL-Lite, but their results (i. e. certain answers) are then interpreted under an epistemic semantics to allow a combination of open- and closed-world conditions. For example, consider the TBox $\{\mathsf{Employee} \sqsubseteq \exists \mathsf{worksFor}\}$, the state $\{\mathsf{Employee}(a)\}$, and the preconditions $\phi_1(x) = \neg[\exists y.\mathsf{worksFor}(x, y)]$ and $\phi_2(x) = \neg\exists y.[\mathsf{worksFor}(x, y)]$, where the conjunctive queries are indicated with square brackets. Then ϕ_1 does not apply to object a, because all employees are known to work for some department (even if the specific department is unknown), but $\phi_2(a)$ is true since no particular y is known for which $\mathsf{worksFor}(a, y)$ holds.

Another major difference to PDDL is the fact that actions are only applicable if they do not yield an inconsistent state s, i. e. such that $\mathcal{T} \cup s$ has no models. For example, the action with precondition $\mathsf{Employee}(x)$ and effect $\mathsf{SoftwareDeveloper}(x)$ would cause an inconsistency when applied to object a in state $\{\mathsf{ElectronicEngineer}(a)\}$ considering the TBox $\{\mathsf{ElectronicEngineer} \sqsubseteq \mathsf{Employee}$, $\mathsf{ElectronicEngineer} \sqsubseteq \neg\mathsf{SoftwareDeveloper}\}$.

The DL-Lite eKAB to PDDL Compilation. In (Calvanese et al. 2016), a translation from *state-bounded* DL-Lite eKABs to equivalent PDDL tasks is presented. Its main ingredients are (i) a bound on the number of objects in each state (Calvanese et al. 2013; Calvanese et al. 2016), (ii) a translation of ECQs into first-oder formulas under closed-world semantics (Poggi et al. 2008; Calvanese et al. 2007; Calvanese et al. 2016), and (iii) an additional predicate and action that checks consistency of a state.

Since PDDL does not support TBoxes, the translation (ii) effectively compiles the TBox into the ECQ conditions, thereby simulating open-world query answering by a closed-world formula. Essentially, every atom is replaced by the disjunction of all its implicants; for example $\mathsf{Employee}(x)$ becomes $\mathsf{Employee}(x) \vee \mathsf{ElectronicEngineer}(x)$ to simu-

[1]The syntax is slightly adapted for compatibility with PDDL.

late the fact that electronic engineers are considered to be employees ($\mathsf{ElectronicEngineer} \sqsubseteq \mathsf{Employee}$). Thus, every CQ becomes a disjunction of CQs that enumerates all possible combinations of atom implicants.

Step (iii) uses a formula that describes every possible inconsistent state. The set of all axioms involving negation, e. g. $\mathsf{ElectronicEngineer} \sqsubseteq \neg\mathsf{SoftwareDeveloper}$, is first reformulated into a disjunction of CQs of the form $\mathsf{ElectronicEngineer}(x) \wedge \mathsf{SoftwareDeveloper}(x)$, which describe basic inconsistent situations. By applying the translation (ii) to these CQs, the resulting formula includes all possible ways in which inconsistencies can be generated.

3 Pre-Compiling the eKAB PDDL

We next explain the main practicality bottleneck of eKAB planning with state-of-the-art planning systems, and then describe our fix adding an additional *pre-compilation* step which transforms the eKAB-generated PDDL to a simpler form of PDDL more tailored towards these systems.

The Problem: Planner Pre-Processors. Observe that, depending on the size and complexity of the ontology, the formulas generated from the TBox can become quite complicated. First, the generated disjunctions of CQs in (ii) can become large if there are many different ways to entail the original atoms. The formula generated in (iii) can be exponentially large in the size of the TBox, since it needs to cover all possible interactions between positive and negative axioms. This is not necessarily a problem per se, if the TBox is small. Yet the structure of these formulas is problematic for state-of-the-art planners like FD and FF, which handle only a rather small fragment of formulas effectively.

The core reason for this is the gap between the PDDL input and the internal representation used for planning: the latter is fully grounded and restricts all conditions (pre, cond, G) to conjunctions of atoms. A *pre-processor* takes care of the transformation from PDDL to this representation, enumerating ground facts and actions and applying a number of syntactic transformations (Gazen and Knoblock 1997). This pre-processor is very similar in both FF and FD. In particular, all formulas are first grounded and then transformed to DNF, where the DNF transformation is done naïvely, i. e. in-situ without introducing auxiliary predicates. Now, while the translation of an individual CQ in eKAB may be a DNF formula, such formulas can be arbitrarily nested inside other logical constructors, and during grounding (which replaces universal/existential quantifiers with enumerative conjunction/disjunction) the formulas grow even larger. These formulas can then be quite costly to translate into DNF naïvely. Indeed, our experiments clearly confirm that this has been the main bottleneck in planning on eKAB models.

Pre-Compilation 1: Non-Naïve DNF Transformation. An obvious answer to this problem is to use a non-naïve DNF transformation instead, introducing auxiliary predicates to represent sub-formulas. Indeed, it has long been known that propositional (i. e. ground) formulas can be compiled away at the cost of an increase in plan length (Nebel 2000). We are not aware of any implementation of this approach, but we experiment with it here. The compilation requires to introduce two auxiliary predicates for every ground sub-formula ϕ: P_ϕ represents the truth value of ϕ in the current state; and P'_ϕ is designed to be true iff ϕ has already been evaluated. Actions are introduced to evaluate ϕ provided P'_ψ is true for all sub-formulas ψ of ϕ; all regular action preconditions include P'_ϕ for the precondition and all effect conditions (this last bit is a small innovation here, going beyond the above-cited compilation which does not handle conditional effects); the effects of all regular actions set all P'_ϕ facts to false, thus forcing the planner to re-evaluate the relevant formulas in the next step.

Pre-Compilation 2: Getting Rid of the Overhead. Beyond this known result, we leverage derived predicates to compile formulas away *without* an increase in plan length. This is actually quite simple, and in principle (see below) works also at the original (non-grounded) PDDL level. For each complex sub-formula $\phi(\vec{x})$ occurring anywhere in the planning task, we introduce a new predicate $P_\phi(\vec{x})$ along with the derivation rule $r = P_\phi(\vec{x}) \leftarrow \phi(\vec{x})$, and we replace ϕ with P_ϕ everywhere except in the body of the new rule r. This results in a set of rules that is equivalent to a non-recursive, and therefore stratified, Datalog program with negation (Abiteboul, Hull, and Vianu 1995).

Implementation of our Pre-Compilations. As previously discussed, PDDL 2.2 does not actually allow stratified Datalog with negation. For this and other pragmatic reasons, we decided to rely partly on the pre-processor of FF (Hoffmann and Nebel 2001). Namely, we let that pre-process ground the eKAB-derived PDDL task, let it transform all formulas to negation normal form, and let it translate negative literals $\neg p$ to new atoms $not{-}p$. At that point, we take over and overwrite FF's original (naïve) DNF transformation. When using derived predicates, this results in negation-free rules compliant with PDDL 2.2. We then let FF continue with the remainder of its pre-process, and output the resulting representation into a PDDL file, which can then be given to any off-the-shelf planner.

4 Experiments

We empirically compare the performance of Fast Forward (FF) (Hoffmann and Nebel 2001) and Fast Downward 20.06 (FD) (Helmert 2006) using the derived-predicate based pre-compiler (denoted DP) with Nebel's (2000) pre-compilation (denoted Ne) and the unprocessed original PDDL files (denoted O). FF was run with standard parameters. For FD, we chose the configuration that most closely resembles FF's search procedure. All experiments were run on a computer with an Intel Core i5-4590 CPU@3.30GHz processor, and run time and memory cutoffs of 600 seconds and 8 GBs.

Benchmarks. We contribute a benchmark collection for DL-Lite planning, consisting of 125 instances from a variety of sources. A detailed description can be found in the supplementary material. We include two scalable eKAB benchmark domains, *Robot* and *TaskAssign*, used in prior

		(a) # solved						(b) # PDDL processed						(c) PDDL processing time						(d) pre-compiler time	
		FF			FD			FF			FD			FF			FD				
Domain	#	O	Ne	DP	O	Ne	DP	O	Ne	DP	O	Ne	DP	O	Ne	DP	O	Ne	DP	Ne	DP
Robot	20	**20**	1	**20**	4	1	**20**	**20**	**20**	**20**	4	**20**	**20**				15.03	20.20	**10.09**	0.08	0.08
TaskAssign	20	1	1	15	3	1	**20**	1	10	15	3	10	**20**				0.81	27.17	**0.12**	34.13	36.34
Cats	20	14	14	**20**	14	11	**20**	14	**20**	**20**	14	**20**	**20**	34.41	0.03	**0.00**	68.34	39.76	**0.14**	0.07	0.07
Elevator	20	12	0	**20**	**20**	0	**20**	12	**20**	**20**	**20**	**20**	**20**	60.20	0.01	**0.00**	0.35	20.51	**0.29**	0.14	0.13
TPSA	15	7	5	5	**14**	4	5	7	5	5	**14**	4	5	**0.00**	14.05	0.05	**1.42**	352.93	1.83	24.60	24.53
VTA	15	**15**	6	**15**	**15**	4	13	**15**	6	**15**	**15**	4	13	**0.00**	10.12	0.34	**1.32**	351.18	304.23	2.63	0.13
VTA-Roles	15	5	4	5	**15**	0	5	6	4	5	**15**	0	5	2.92	25.26	**0.29**				49.88	49.15
Assembly	30	0	0	24	**30**	0	**30**	9	10	24	**30**	4	**30**	**0.00**	6.58	0.03	**0.22**	383.73	0.55	2.31	1.31
GridPlacement	20	5	1	17	6	2	**20**	5	**20**	**20**	6	**20**	**20**	41.35	0.23	**0.01**	49.80	29.06	**7.01**	0.03	0.02
Miconic	30	9	3	**13**	9	2	9	11	27	19	14	18	**30**				56.61	75.71	**0.33**	0.32	0.05
$\sum$	205	88	35	154	130	25	**162**	100	142	163	135	120	**183**	27.47	6.32	**0.09**	28.23	92.95	**18.61**	6.28	6.18

Table 1: Per-domain aggregated statistics. Best results are highlighted in **bold**. (a) Number of instances solved within resource limits. (b) Number of instances that passed the planners' PDDL processing step. (c) Average PDDL processing time (seconds) on instances successfully processed by all configurations, considering FF and FD individually. Instances are ignored if the processing time was less than 1 second in all configurations. (d) Average time (seconds) for the pre-compilations.

work (Calvanese et al. 2016; Stawowy 2016). We extend these with additional larger instances. We adapt the *VTA* (virtual travel agency) and *TPSA* (VOIP request) benchmarks from semantic web-service composition (Hoffmann et al. 2008) to the eKAB framework, also adding a third domain *VTA-Roles* where we included a more interesting ontology. Finally we created two new eKAB domains, *Cats* and *Elevator*, inspired by standard planning benchmarks. In addition we run experiments on PDDL benchmark domains, outside the eKAB context, as our pre-compilation techniques may be useful on any planning domain with complex action pre- and effect conditions. We used *Assembly* and *Miconic* with minor modifications, and we created a showcase domain, *GridPlacement*, specifically designed to contain challenging DNF transformations. The benchmarks[2] and pre-compilers[3] are available online.

Discussion. Table 1 gives a summary of the results. As indicated by the comparison of the O columns in (a) and (b), PDDL processing indeed constitutes the bottleneck in this benchmark collection. In almost every instance that was not solved, the planners have been terminated already during the construction of their internal planning task representation.

Table 1 (b) and (c) show that the DP pre-compilation can successfully reduce the overhead of both planners' PDDL input handling in nearly every domain. The only exception is in web-service composition, were the pre-compilation turned out detrimental, especially for FD. Besides an absence of complex conditions, in the web-service domains the actions assign predicates to previously unbound objects in the world. Therefore, by increasing the number of objects, the grounding and translation sizes grow drastically.

The PDDL processing advantages carry over to overall performance. FF and FD solve a significantly larger fraction of the DP pre-compiled instances than unprocessed ones.

In contrast, both planners could not exploit the previous pre-compilation approach, Ne, effectively. There are two reasons. First of all, the encoding of the formula evaluations resulted in a considerable blow-up of the input files. This overhead affects both planners' PDDL processing, and partly explains the difference between the Ne and DP columns in Table 1 (b) and (c). Comparing both planners, FF deals with the Ne results much more efficiently. While FF can process substantially more instances after the Ne pre-compilation than before, FD actually processes less. For one, it should be noted that, due to the particular compiler implementation, the structure of the generated PDDL files already correlate with FF's internal task representation to some extent. Moreover, FD's internal planning task representation differs fundamentally from that of FF. However, deriving this representation requires additional analysis steps that scale with the number of actions (yet not with derived predicates, which are treated separately). The comparatively large number of auxiliary actions introduced by the compilation significantly impedes FD's performance even in the smallest instances, which is reflected in the large average run times reported in Table 1 (c). The second and even more critical issue of the Ne pre-compiler is the obfuscation of the planning task's original structure. As alluded by Table 1 (a), both planners' searches run into serious troubles. The number of instances that could be solved by FF and FD after the pre-compilation drops substantially in every domain.

5 Conclusion

We demonstrated that planning with DL-Lite eKABs can be made feasible through pre-compilations producing PDDL more digestible for state-of-the-art planners. While planning with DL state axioms has been of long-standing interest, this is the first practical integration we are aware of.

This positive result paves the way for future research expanding this integration. Possible directions include the design of planning tools tailored to eKABs, as well as eKABs over more expressive ontology languages leveraging e.g. PDDL derived predicates.

[2] https://gitlab.perspicuous-computing.science/m.steinmetz/pddl-dllite-benchmarks.git

[3] https://gitlab.perspicuous-computing.science/a.kovtunova/moreflags2.git

Acknowledgments

This work is supported by DFG grant 389792660, TRR 248 (`https://perspicuous-computing.science`).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2005. Dl-Lite: Tractable description logics for ontologies. In Veloso, M. M., and Kambhampati, S., eds., *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, 602–607. AAAI Press / The MIT Press.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. EQL-Lite: Effective first-order query processing in description logics. In Veloso, M. M., ed., *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI'07)*, 274–279.

Calvanese, D.; De Giacomo, G.; Montali, M.; and Patrizi, F. 2013. Verification and synthesis in description logic based dynamic systems. In Faber, W., and Lembo, D., eds., *Proc. of the 7th Int. Conf. on Web Reasoning and Rule Systems (RR'13)*, Lecture Notes in Computer Science, 50–64. Springer.

Calvanese, D.; Montali, M.; Patrizi, F.; and Stawowy, M. 2016. Plan synthesis for knowledge and action bases. In Kambhampati, S., ed., *Proc. of the 25th Int. Joint Conf. on Artificial Intelligence (IJCAI'16)*, 1022–1029. AAAI Press.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Gazen, B. C., and Knoblock, C. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proceedings of the 4th European Conference on Planning (ECP'97)*, 221–233.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Ginsberg, M. L., and Smith, D. E. 1988. Reasoning about action I: A possible worlds approach. *Artificial Intelligence* 35:165–195.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research* 24:519–579.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J.; Weber, I.; Scicluna, J.; Kacmarek, T.; and Ankolekar, A. 2008. Combining scalability and expressivity in the automatic composition of semantic web services. In *8th International Conference on Web Engineering (ICWE'08)*.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

Pednault, E. P. D. 1989. ADL: exploring the middle ground between STRIPS and the situation calculus. In Brachman, R. J.; Levesque, H. J.; and Reiter, R., eds., *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89). Toronto, Canada, May 15-18 1989*, 324–332. Morgan Kaufmann.

Poggi, A.; Lembo, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *Journal on Data Semantics* X:133–173.

Stawowy, M. 2016. *Plan Synthesis in Explicit-input Knowledge and Action Bases*. Ph.D. Dissertation, IMT School for Advanced Studies Lucca, Italy.

Thiebaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1–2):38–69.

Making DL-Lite Planning Practical: Supplementary Material

Benchmark Description

Our collection of benchmarks consists of a total of 205 instances from varied sources, including existing ones as well as ones that we created.

Original eKAB benchmarks: We considered the scalable eKAB benchmark *Robot* used in prior work [Calvanese et al., 2016, Stawowy, 2016]. In this domain, a robot is positioned on a grid without knowing its exact position. The goal of the robot is to reach a specific cell. The ontology describes relations between rows and columns, e. g. $\mathsf{BelowOf6} \sqsubseteq \mathsf{BelowOf7}$ and $\mathsf{AboveOf1} \sqsubseteq \neg\mathsf{BelowOf1}$. In the original work, the Fast Downward[1] planner (FD) reached a 30-minutes timeout on a 7×7-grid. However, for the experiments reported in this paper, FD using our pre-compilation managed to solve an instance of size 22×22 in less than 2 minutes.

TaskAssign was also presented in [Calvanese et al., 2016] as a running example. In our paper, we also used some of its axioms in examples. The goal of this benchmark is to hire two electronic engineers for a company, among other personnel. The ontology describes disjointness relations of different job positions. For our experiments, we scale up the original problem by increasing the number of already hired people as well as the number of objects in the world.

Own eKAB domains: We created a new eKAB domain *Cats*, inspired by the well-known classical planning benchmark *BT* (see [Petrick and Bacchus, 2002] for a description). In this domain, there is a set of packages connected by a functional binary predicate $\mathsf{contains}$ either to a Cat or to a Bomb. No Cat is a Bomb, both of them are Objects and only a $\mathsf{Package}$ can $\mathsf{contain}$ an Object. The actions $\mathsf{release}$ and dunk can be performed on a package to release a cat or disarm a bomb from the package. The task is to find a plan such that only disarmed objects remain in the packages.

In addition, a scalable eKAB domain called *Elevator* was developed inspired by *Miconic* [Koehler and Schuster, 2000], a classical planning benchmark which we also consider in our experiments. An elevator can move up and down between floors to serve passengers according to their origins and destinations. Here, the ontology axioms establish, inter alia, that $\mathsf{Boarded}$ and Served are subcategories of $\mathsf{Passenger}$ and

[1] `http://www.fast-downward.org/`

each Floor is connected to exactly one other Floor by the functional binary predicate next.

Web-service composition: We adapted already existing web-service composition benchmarks [Hoffmann et al., 2008] to the eKAB framework: *VTA*, *VTA-Roles*, and *TPSA*. In the original *VTA* task, a travel agency must book a travel itinerary including a Flight, a Hotel and a CarRental. The ontology distinguishes different types of requests, e. g. FlightRequest $\sqsubseteq$ Flight and FlightRequest $\sqsubseteq \neg$CarRentalRequest. The only modification we made is setting up an order for choosing new, unbound objects from the world. This little trick simplifies grounding not only for our pre-compilation, but also for internal grounding in FF and FD. Next, since the *VTA* axioms in the original planning task are relatively simple, for *VTA-Roles* we used the expressiveness of $DL\text{-}Lite$ in eKAB to create a more elaborate ontology with binary predicates that connect all requests like FlightRequest or AirportShuttle to the Trip they were made for.

In *TPSA*, when a Service is requested, a business process for preparing a contract, a hardware and a product activation must be developed by a planner. In this domain, we also incorporated binary predicates as well as an assignment order on the level of an ontology.

In all web-service domains, *TPSA*, *VTA*, and *VTA-Roles*, most of the actions assign predicates to new objects in the world. Therefore, by increasing the number of objects (even with introduced assignment orders), the grounding and pre-compilation sizes grow drastically. That fact together with an absence of complex conditions in the domain actions results in a decreased performance when using our pre-compilation.

All of the previously described benchmarks are created specifically in order to extend planning with background knowledge expressed by an ontology. Thus, the problem instances for *Robot*, *TaskAssign*, *Cats*, *Elevator*, *TPSA*, *VTA* and *VTA-Roles* have two representations: the eKAB syntax and its translation[2] into PDDL. The benchmarks in the following paragraph do not have any ontology as a part of the domain description.

PDDL: We included two benchmark domains from the planning literature with complex action pre- and effect conditions: *Assembly*, and *Miconic*. We made minor modifications, introducing additional actions to make some predicates non-static. This prevents the planners from simplifying the conditions in a simple pre-processing step. In *Assembly*, given available parts and tools, a planner must find a valid way to assemble a bracket by setting assemble- and remove-orders and then implement this plan with respect to committed resources. *Miconic* resembles *Elevator*, but it does not have any ontology and it considers also some categories of passengers like attendant or never alone; the domain assumes that passengers may have conflicts with each other and an access to the elevator can also be blocked.

Additionally, we created a showcase domain, *GridPlacement*, specifically designed to contain challenging DNF transformations. In contrast to *Robot*, here a robot starts from a known position of the grid, it has a fixed weight capacity and there are some

[2]see `https://github.com/stawo/ekabPlanner`

objects with given weights scattered on a grid. By pickup, drop and move, the robot must transport the objects to specific coordinates on the grid.

References

[Calvanese et al., 2016] Calvanese, D., Montali, M., Patrizi, F., and Stawowy, M. (2016). Plan synthesis for knowledge and action bases. In Kambhampati, S., editor, *Proc. of the 25th Int. Joint Conf. on Artificial Intelligence (IJCAI'16)*, pages 1022–1029. AAAI Press.

[Hoffmann et al., 2008] Hoffmann, J., Weber, I., Scicluna, J., Kacmarek, T., and Ankolekar, A. (2008). Combining scalability and expressivity in the automatic composition of semantic web services. In *8th International Conference on Web Engineering (ICWE'08)*.

[Koehler and Schuster, 2000] Koehler, J. and Schuster, K. (2000). Elevator control as a planning problem. In Chien, S. A., Kambhampati, S., and Knoblock, C. A., editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pages 331–338. AAAI.

[Petrick and Bacchus, 2002] Petrick, R. P. A. and Bacchus, F. (2002). A knowledge-based approach to planning with incomplete information and sensing. In Ghallab, M., Hertzberg, J., and Traverso, P., editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pages 212–222. AAAI.

[Stawowy, 2016] Stawowy, M. (2016). *Plan Synthesis in Explicit-input Knowledge and Action Bases*. PhD thesis, IMT School for Advanced Studies Lucca, Italy.