Automated Planning with Ontologies under Coherence Update Semantics

Stefan Borgwardt[1] , **Duy Nhu**[1] , **Gabriele Röger**[2]

[1]Institute of Theoretical Computer Science, TU Dresden, Germany

[2]Department of Mathematics and Computer Science, University of Basel, Switzerland

{stefan.borgwardt, hoang_duy.nhu}@tu-dresden.de, {firstname.lastname}@unibas.ch

Abstract

Standard automated planning employs first-order formulas under closed-world semantics to achieve a goal with a given set of actions from an initial state. We follow a line of research that aims to incorporate background knowledge into automated planning problems, for example by means of ontologies, which are usually interpreted under open-world semantics. We present a new approach for planning with *DL-Lite* ontologies that combines the advantages of ontology-based action conditions provided by *explicit-input knowledge and action bases (eKABs)* and ontology-aware action effects under the *coherence update semantics*. We show that the complexity of the resulting formalism is not higher than that of previous approaches, and provide an implementation via a polynomial compilation into classical planning. An evaluation on existing and new benchmarks examines the performance of a planning system on different variants of our compilation.

1 Introduction

Automated planning is a core area within Artificial Intelligence that describes the development of a system through the application of actions (Ghallab, Nau, and Traverso 2004). A planning task is defined by an initial state, a set of actions with preconditions and effects on the current state, and a goal condition. States can be seen as finite first-order (FO) interpretations, and all conditions are specified by FO-formulas that are interpreted on the current state under closed-world semantics. The objective is to select a sequence of applicable actions to reach the goal.

In the literature, one can find many approaches to add expressive reasoning facilities to planning formalisms (Ahmetaj et al. 2014; De Giacomo, Favorito, and Silo 2024; Corrêa et al. 2024; John and Koopmann 2024). One approach is to use additional logical theories under open-world semantics to describe the possible interactions between objects of a domain of interest. Here, we are interested in *Description Logics* (*DLs*) and their application in reasoning about the individual states of a system. The main challenge is to reconcile the open-world nature of DLs and the closed-world semantics employed in classical planning.

Explicit-input Knowledge and Action Bases (*eKABs*) combine planning with the description logic *DL-Lite* (Calvanese et al. 2016). There, states (*ABoxes*) are interpreted using open-world semantics w.r.t. a *background ontology* (*TBox*) specifying intensional knowledge using *DL-Lite* axioms. The background ontology describes constraints on the state, and entails additional facts that hold implicitly. Such a planning problem can be compiled into the classical *planning domain definition language (PDDL)* using query rewriting techniques (Calvanese et al. 2016).

Example 1. *Consider the following axioms and facts in a blocks world:*

$\mathsf{on\_block} \sqsubseteq \mathsf{on},\ \exists \mathsf{on\_block}^- \sqsubseteq \mathsf{Block},\ \texttt{funct}\ \mathsf{on\_block},$

$\mathsf{on\_table} \sqsubseteq \mathsf{on},\ \exists \mathsf{on\_table}^- \sqsubseteq \mathsf{Table},\ \mathsf{Block} \sqsubseteq \neg\mathsf{Table},$

$\mathsf{Block} \equiv \exists \mathsf{on},\ \exists \mathsf{on\_block}^- \sqsubseteq \mathsf{Blocked},$

$\exists \mathsf{on\_block} \sqsubseteq \neg\exists \mathsf{on\_table},\ \mathsf{on\_block}(b_1, b_2),\ \mathsf{on\_table}(b_3, t)$

Implicitly, we know that b_2 *is blocked (*$\mathsf{Blocked}(b_2)$*) since* b_1 *is on* b_2 *(*$\mathsf{on\_block}(b_1, b_2)$*) and every block that has another block on top is blocked (*$\exists \mathsf{on\_block}^- \sqsubseteq \mathsf{Blocked}$*). On the other hand, we know that* $\mathsf{on\_block}(b_1, b_3)$ *cannot hold, since the* on_block *relation is functional (*`funct` on_block*).*

Consider now the action $\mathsf{move}(x, y, z)$ *that moves Block* x *from position* y *to* z*. Its precondition is* $[\mathsf{on}(x, y)] \wedge \neg[\mathsf{Blocked}(x)] \wedge \neg[\mathsf{Blocked}(z)]$*, where the atoms in brackets are evaluated w.r.t. the ontology axioms (epistemic semantics). For example, the action is applicable for the substitution* $x \mapsto b_1$*,* $y \mapsto b_2$*,* $z \mapsto b_3$*, since* on_block *is included in* on *and neither* $\mathsf{Blocked}(b_1)$ *nor* $\mathsf{Blocked}(b_3)$ *are entailed.*

However, one property of this formalism is that action effects operate directly on the state, ignoring implicit knowledge, and only check whether the subsequent state is still consistent with the TBox.

Example 2. *The effect of the ground action* $\mathsf{move}(b_1, b_2, b_3)$ *is to add* $\mathsf{on\_block}(b_1, b_3)$ *to the state. In the eKAB formalism, this would make the state inconsistent, as argued above.*

To obtain a consistent state, we could remove $\mathsf{on}(b_1, b_2)$*. However, since this fact is not explicitly present in the state (ABox), this operation would not affect the state at all and* $[\mathsf{on}(b_1, b_2)]$ *would continue to hold due to* $\mathsf{on\_block}(b_1, b_2)$*.*

Moreover, even if we explicitly remove $\mathsf{on\_block}(b_1, b_2)$*, we would lose the information that* b_2 *is a block, which means that we should add* $\mathsf{Block}(b_2)$ *as well.*

This illustrates that actions can cause three types of implicit effects: removing a fact requires (i) removing all

stronger facts and (ii) adding previously implied facts to avoid losing information, whereas adding a fact requires (iii) removing any conflicting facts to ensure consistency.

Addressing these problems, De Giacomo et al. (2021) introduced the *coherence update semantics* for updating an ABox in the presence of a *DL-Lite* TBox. In our example, adding $\mathsf{on\_block}(b_1, b_3)$ would automatically remove $\mathsf{on\_block}(b_1, b_2)$ (iii) and add $\mathsf{Block}(b_2)$ (ii). Similarly, removing $\mathsf{on}(b_1, b_2)$ would also remove $\mathsf{on\_block}(b_1, b_2)$ (i). Notably, the updated ABox can be computed with the help of a non-recursive Datalog$^\neg$ program.

De Giacomo et al. (2021) only considered single-step ABox updates. However, for planning, such implicit effects need to be taken into account for each action on the way to the goal. In this paper, we extend eKAB planning by applying the coherence update semantics to action effects. We investigate the complexity of the resulting formalism of *ceKABs (coherent eKABs)* and show that it is not higher than for classical planning. In fact, ceKAB planning tasks can be compiled into PDDL with *derived predicates* by utilizing Datalog$^\neg$ programs describing eKAB (Borgwardt et al. 2022) and coherence semantics (De Giacomo et al. 2021). Moreover, we evaluate the feasibility of our approach in off-the-shelf planning systems and the overhead incurred compared to the original eKAB semantics.

Full proofs of all results as well as the implementation and benchmarks are available in the supplementary material.

1.1 Closely Related Work

Liu et al. (2011) formalized instance-level updates for expressive DLs where the update result can be expressed in the same description logic. Ahmetaj et al. (2014) described integrity constraints and states of graph-structured data over an action language where actions insert and delete nodes/labels using expressive DLs. The eKAB formalism was optimized and extended to support all Datalog$^\neg$-rewritable Horn DLs, via a compilation into PDDL with derived predicates (Borgwardt et al. 2021; Borgwardt et al. 2022). A similar approach uses a black-box, justification-based algorithm to compile an ontology-mediated planning problem into classical planning, even for non-Horn DLs (John and Koopmann 2024).

2 Preliminaries

We introduce all relevant formalisms, including *DL-Lite*, PDDL, eKABs, and the coherence update semantics. For more details, we refer to the original papers.

2.1 The Description Logic *DL-Lite*$_{core}^{(\mathcal{HF})}$

We consider *DL-Lite*$_{core}^{(\mathcal{HF})}$ (Artale et al. 2009), which we simply call *DL-Lite* in the following.[1] Given disjoint sets $\mathbf{C}, \mathbf{R}, \mathbf{I}$ of *atomic concepts* A, *atomic roles* P, and *individuals* (*constants*) c, *(general) concepts and roles* are formed as follows:

$$\textit{basic role: } Q \longrightarrow P \mid P^- \qquad \textit{role: } R \longrightarrow Q \mid \neg Q$$
$$\textit{basic concept: } B \longrightarrow A \mid \exists Q \qquad \textit{concept: } C \longrightarrow B \mid \neg B$$

[1] Calvanese et al. (2016) and De Giacomo et al. (2021) used *DL-Lite*$_A$, which is *DL-Lite*$_{core}^{(\mathcal{HF})}$ extended with attributes.

For the *inverse role* P^-, we set $(P^-)^- := P$. A *TBox* (a.k.a. *ontology*) is a finite set of *concept inclusions* $B \sqsubseteq C$, *role inclusions* $Q \sqsubseteq R$, and *functionality axioms* $(\mathtt{funct}\ Q)$, where an inclusion is called *negative* if it contains $\neg$, and *positive* otherwise, and functional basic roles and their inverses are not allowed to occur on the right-hand side of any positive inclusions. An *ABox* is a finite set of *concept assertions* $A(c)$ and *role assertions* $P(c, c')$, where c, c' are individuals. A *knowledge base* (*KB*) $\mathcal{K}$ is a tuple $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. The *signature* of $\mathcal{K}$, $\mathcal{A}$, or $\mathcal{T}$ is the subset of symbols $\mathbf{C} \cup \mathbf{R} \cup \mathbf{I}$ used in it.

An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of a non-empty set $\Delta^\mathcal{I} \supseteq \mathbf{I}$ (the *domain* of objects under the *standard name assumption*) and an *interpretation function* $\cdot^\mathcal{I}$ that maps atomic concepts A to subsets $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and roles P to relations $P^\mathcal{I} \subseteq (\Delta^\mathcal{I})^2$. This function is extended to

$$(P^-)^\mathcal{I} = \{(o', o) \mid (o, o') \in P^\mathcal{I}\} \quad (\neg Q)^\mathcal{I} = (\Delta^\mathcal{I})^2 \setminus Q^\mathcal{I}$$
$$(\exists Q)^\mathcal{I} = \{o \mid \exists o'.\, (o, o') \in Q^\mathcal{I}\} \quad (\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$$

An interpretation $\mathcal{I}$ *satisfies* $B \sqsubseteq C$ if $B^\mathcal{I} \subseteq C^\mathcal{I}$; $Q \sqsubseteq R$ if $Q^\mathcal{I} \subseteq R^\mathcal{I}$; $\mathtt{funct}\ Q$ if $(o, o'), (o, o'') \in Q^\mathcal{I}$ implies $o' = o''$; $A(c)$ if $c \in A^\mathcal{I}$; and $P(c, c')$ if $(c, c') \in P^\mathcal{I}$. It is a *model* of a knowledge base $\mathcal{K}$ if $\mathcal{I}$ satisfies all axioms in $\mathcal{K}$. If every model of $\mathcal{K}$ satisfies an axiom α, then $\mathcal{K}$ *entails* α (written $\mathcal{K} \models \alpha$). $\mathcal{K}$ is *consistent* if it has a model, and an ABox $\mathcal{A}$ is *consistent with* a TBox $\mathcal{T}$ if $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent.

The *$\mathcal{T}$-closure* $cl_\mathcal{T}(\mathcal{A})$ of $\mathcal{A}$ w.r.t. $\mathcal{T}$ is the set of all assertions over the signature of $\langle \mathcal{T}, \mathcal{A} \rangle$ that are entailed by $\langle \mathcal{T}, \mathcal{A} \rangle$. Two ABoxes $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* w.r.t. $\mathcal{T}$ if $cl_\mathcal{T}(\mathcal{A}) = cl_\mathcal{T}(\mathcal{A}')$. The *deductive closure* $cl(\mathcal{T})$ of $\mathcal{T}$ is the set of all TBox axioms over the signature of $\mathcal{T}$ that are entailed by $\mathcal{T}$, and can be computed in polynomial time (Calvanese et al. 2007b; Artale et al. 2009).

Example 3. *For Example 1, we have*

$$\mathcal{T} = \{\mathsf{on\_block} \sqsubseteq \mathsf{on},\ \exists \mathsf{on\_block}^- \sqsubseteq \mathsf{Block},\ \dots\}$$
$$\mathcal{A} = \{\mathsf{on\_block}(b_1, b_2),\ \mathsf{on\_table}(b_3, t)\}$$
$$cl_\mathcal{T}(\mathcal{A}) = \{\mathsf{on}(b_1, b_2),\ \mathsf{Block}(b_1),\ \mathsf{Block}(b_2),\ \dots\}$$
$$cl(\mathcal{T}) = \{\exists \mathsf{on\_block} \sqsubseteq \mathsf{Block},\ \exists \mathsf{on} \sqsubseteq \neg \mathsf{Table},\ \dots\}$$

Queries and Datalog A *conjunctive query* (*CQ*) is an FO-formula of the form $q(\vec{x}) := \exists \vec{y}.\Phi(\vec{x}, \vec{y})$ where Φ is a conjunction atoms of the form $A(x)$ or $P(x, y)$. A *union of conjunctive queries* (UCQ) is a disjunction of CQs with the same free variables. Given a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ and a substitution θ of the variables in $\vec{x}$ by constants from $\mathcal{A}$, the *UCQ answering* problem is to decide whether $\langle \mathcal{T}, \mathcal{A} \rangle$ entails the ground UCQ $\theta(q)$, which we denote as $\mathcal{A}, \mathcal{T}, \theta \models q$. Assuming w.l.o.g. that $\mathcal{T}$ contains two disjoint atomic concepts $A \sqsubseteq \neg B$, the special CQ $\bot := \exists x.A(x) \wedge B(x)$ can be used to check consistency of $\langle \mathcal{A}, \mathcal{T} \rangle$.

In the following, we use *states* s instead of ABoxes, which are finite sets of ground atoms (*facts*) $p(\vec{c})$ over a finite set $\mathcal{P}$ of predicates of arbitrary arity. When reasoning in *DL-Lite*, we only consider the unary and binary atoms in a state. Further, we denote by $\mathbf{I}(s)$ the constants occurring in s. Calvanese et al. (2007a) introduced *extended conjunc-*

tive queries (ECQs) to combine open- and closed-world reasoning. An ECQ Q is constructed as follows:

$$Q \coloneqq p(\vec{x}) \mid [q(\vec{x})] \mid \neg Q \mid Q_1 \wedge Q_2 \mid \exists y.\, Q\,,$$

where p is a predicate (of any arity), $\vec{x}$ are terms (constants or variables) and $q(\vec{x})$ is a UCQ. Intuitively, $[q(\vec{x})]$ denotes the evaluation of $q(\vec{x})$ w.r.t. a KB $\langle \mathcal{T}, s\rangle$, whereas $q(\vec{x})$ (without brackets $[\cdot]$) is evaluated directly over the minimal model of the state s. The complete semantics of ECQs is as follows:

$s, \mathcal{T}, \theta \models p(\vec{x})$	iff $s \models p(\theta(\vec{x}))$
$s, \mathcal{T}, \theta \models [q]$	iff $s, \mathcal{T}, \theta \models q$
$s, \mathcal{T}, \theta \models \neg Q$	iff $s, \mathcal{T}, \theta \not\models Q$
$s, \mathcal{T}, \theta \models Q_1 \wedge Q_2$	iff $s, \mathcal{T}, \theta \models Q_1$ and $s, \mathcal{T}, \theta \models Q_2$
$s, \mathcal{T}, \theta \models \exists y.Q$	iff $\exists d \in \mathbf{I}(s).\ s, \mathcal{T}, \theta[y \to d] \models Q$

Except for $[q]$, this corresponds to the standard (closed-world) evaluation of FO-formulas.

A *Datalog$^\neg$ rule* (Abiteboul, Hull, and Vianu 1995) has the form $p(\vec{x}) \leftarrow \Phi(\vec{x}, \vec{y})$, where the *head* $p(\vec{x})$ is an atom and the *body* $\Phi(\vec{x}, \vec{y})$ is a conjunction of *literals*, i.e. atoms or negated atoms. A finite set of Datalog$^\neg$ rules (*program*) $\mathcal{R}$ is *stratified* if there is a partition $\mathcal{P}_1, \dots, \mathcal{P}_n$ of the set of predicates $\mathcal{P}$ appearing in $\mathcal{R}$ s.t. for each predicate $p_i \in \mathcal{P}_i$ and $p_i(\vec{x}) \leftarrow \Phi(\vec{x}, \vec{y}) \in \mathcal{R}$ ($i \in \{1, \dots, n\}$), the following holds: (1) If $p_j \in \mathcal{P}_j$ appears in $\Phi(\vec{x}, \vec{y})$, then $j \leq i$; (2) If $\neg p_j \in \mathcal{P}_j$ appears in $\Phi(\vec{x}, \vec{y})$, then $j < i$. We consider only stratified Datalog$^\neg$ programs.

For a state s and a program $\mathcal{R}$, $\mathcal{R}(s)$ denotes the minimal Herbrand model of $s \cup \mathcal{R}$, where all variables are implicitly unversally quantified. Let $\mathcal{T}$ be a TBox and $q(\vec{x})$ be an UCQ. Then, $\mathcal{T}$ and $q(\vec{x})$ are *Datalog$^\neg$-rewritable* if there is a Datalog$^\neg$ program $\mathcal{R}_{\mathcal{T},q}$ and a predicate P_q s.t., for every state s and every mapping θ of $\vec{x}$ to $\mathbf{I}(s)$, it holds that $s, \mathcal{T}, \theta \models q(\vec{x})$ iff $\mathcal{R}_{\mathcal{T},q}(s) \models P_q(\theta(\vec{x}))$. When using *DL-Lite* TBoxes, all UCQs can be rewritten in this way (Calvanese et al. 2007b; Eiter et al. 2012), even without negation or recursion. Moreover, Calvanese et al. (2007a) extended rewritability to ECQs as follows:

Proposition 1. *Given an ECQ Q, let $\mathcal{R}_{\mathcal{T},Q}$ be the disjoint union of $\mathcal{R}_{\mathcal{T},q}$ for all $[q(\vec{x})]$ in Q, and let the FO-formula $Q_\mathcal{T}$ be the result of substituting each $[q(\vec{x})]$ in Q with $P_q(\vec{x})$. Then, $s, \mathcal{T}, \theta \models Q(\vec{x})$ iff $\mathcal{R}_{\mathcal{T},Q}(s) \models Q_\mathcal{T}(\theta(\vec{x}))$, for all s, θ.*

Example 4. *In the blocks world example, the query $\bot$ can be rewritten into the following rules, among others:*

$$\begin{aligned}
P_\bot &\leftarrow \mathsf{on_block}(x, y), \mathsf{on_table}(x, z)\\
P_\bot &\leftarrow P_{\mathsf{Block}}(y), P_{\mathsf{Table}}(y)\\
P_\bot &\leftarrow \mathsf{on_block}(x, y), \mathsf{on_block}(x, z), y \neq z\\
P_{\mathsf{Block}}(x) &\leftarrow \mathsf{Block}(x)\\
P_{\mathsf{Block}}(x) &\leftarrow \mathsf{on_block}(x, y)\\
P_{\mathsf{Block}}(x) &\leftarrow P_{\mathsf{on}}(x, y)
\end{aligned}$$

The ECQ $[\mathsf{on}(x, y)] \wedge \neg[\mathsf{Blocked}(x)] \wedge \neg[\mathsf{Blocked}(z)]$ can be rewritten to $P_{\mathsf{on}}(x, y) \wedge \neg P_{\mathsf{Blocked}}(x) \wedge \neg P_{\mathsf{Blocked}}(z)$.

2.2 PDDL

We use PDDL 2.1 (Fox and Long 2003) with stratified derived predicates. Here, states are viewed under the closed-world assumption, i.e. atoms absent from a state s are treated as false. Let $\mathcal{P}_{der}$ be a finite set of *derived predicates*. In the following, all FO-formulas are constructed from $\mathcal{P} \cup \mathcal{P}_{der}$ unless explicitly stated otherwise. However, states only contain facts over $\mathcal{P}$, and hence action effects cannot add or remove facts over $\mathcal{P}_{der}$. An *action* is a tuple $(\vec{x}, \mathtt{pre}, \mathtt{eff})$ consisting of parameters, a precondition, and a finite set of effects. A *precondition* is an FO-formula with free variables from $\vec{x}$ and a *(conditional) effect* is a tuple $(\vec{y}, \mathtt{cond}, \mathtt{add}, \mathtt{del})$. Here, $\vec{y}$ are additional variables, $\mathtt{cond}$ is an FO-formula with free variables from $\vec{x} \cup \vec{y}$, and $\mathtt{add}$ and $\mathtt{del}$ are finite sets of atoms and negated atoms, respectively, containing free variables from $\vec{x} \cup \vec{y}$ and predicates from $\mathcal{P}$.

A PDDL *domain description* is a tuple $(\mathcal{P}, \mathcal{P}_{der}, \mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ is a finite set of actions and $\mathcal{R}$ is a finite set of FO-rules of the form $p(\vec{x}) \leftarrow \Phi(\vec{x})$, where $p \in \mathcal{P}_{der}$ and Φ is an FO-formula. The set $\mathcal{R}$ is also restricted to be *stratified*, which is defined similarly as for Datalog$^\neg$ programs by restricting the positive and negative occurrences of predicates in $\Phi(\vec{x})$ depending on the location of p in a suitable partition of $\mathcal{P}$. While not every stratified set of FO-rules has an equivalent stratified Datalog$^\neg$ program (Röger and Grundke 2024), the converse trivially holds. A PDDL *task* is a tuple $(\Delta, \mathcal{O}, \mathcal{I}, \mathcal{G})$, where Δ is a PDDL domain description, $\mathcal{O}$ is a finite set of constants including those used in Δ, $\mathcal{I}$ is an *initial state*, and $\mathcal{G}$ is the *goal*, a closed FO-formula. Here, $\mathcal{O}$ plays the same role as the set $\mathbf{I}$ for DLs, and for consistency we will use the notation $\mathcal{O}$ in both settings from now on.

Substituting the parameters in $a = (\vec{x}, \mathtt{pre}, \mathtt{eff})$ according to an assignment θ from $\vec{x}$ to constants in $\mathcal{O}$ yields the *ground* action $\theta(a)$. For a ground action, we omit $\vec{x}$ and write $(\mathtt{pre}, \mathtt{eff})$. Given a state s, the ground action $a = (\mathtt{pre}, \mathtt{eff})$ is *applicable in s* if $\mathcal{R}(s) \models \mathtt{pre}$, which is defined similarly to Datalog$^\neg$. The *application* of a to s then produces a new state $s[\![a]\!]$ that contains a fact α iff either

- there is $(\vec{y}, \mathtt{cond}, \mathtt{add}, \mathtt{del}) \in \mathtt{eff}$ and an assignment θ such that $\mathcal{R}(s) \models \theta(\mathtt{cond})$ and $\alpha \in \theta(\mathtt{add})$, or
- $\alpha \in s$ and, for all $(\vec{y}, \mathtt{cond}, \mathtt{add}, \mathtt{del}) \in \mathtt{eff}$ and θ with $\mathcal{R}(s) \models \theta(\mathtt{cond})$, it holds that $\neg\alpha \notin \theta(\mathtt{del})$.

Intuitively, an effect inserts and removes facts according to $\theta(\mathtt{add})$ and $\theta(\mathtt{del})$ whenever $\theta(\mathtt{cond})$ is true in $\mathcal{R}(s)$. If an action concurrently inserts and deletes α due to conflicting effects, the insertion takes precedence by definition.

A sequence of ground actions $\Pi = a_1, \dots, a_n$ yields a sequence of states s, $s[\![a_1]\!]$, $s[\![a_1]\!][\![a_2]\!]$, …, $s[\![a_1]\!] \dots [\![a_n]\!]$ if each a_i is applicable in the preceding state, in which case Π is called *applicable in s*. Π is a *plan* for the PDDL task $(\Delta, \mathcal{O}, \mathcal{I}, \mathcal{G})$ if Π is applicable in $\mathcal{I}$ and $\mathcal{R}(\mathcal{I}[\![\Pi]\!]) \models \mathcal{G}$. The *length* of this plan is $\|\Pi\| := n$.

Although derived predicates are often implemented by treating them as simple actions that can be applied after each normal action, they can be used to encode planning problems more succinctly. In general, derived predicates cannot

be compiled away without increasing either the size of the PDDL task or the length of the plans super-polynomially (Thiébaux, Hoffmann, and Nebel 2005).

2.3 eKABs

An *explicit-input knowledge and action base (eKAB) domain description* (Calvanese et al. 2016) is a tuple $(\mathcal{P}, \mathcal{A}, \mathcal{T})$, where $\mathcal{P}$ is a finite set of predicates, $\mathcal{A}$ a finite set of actions that use ECQs instead of FO-formulas, and $\mathcal{T}$ a TBox over the unary and binary predicates in $\mathcal{P}$. An *eKAB task* is a tuple $(\Delta, \mathcal{O}, \mathcal{O}_0, \mathcal{I}, \mathcal{G})$, where Δ is an eKAB domain description, $\mathcal{O}$ is a set of constants (potentially infinite), $\mathcal{O}_0$ is a finite subset of $\mathcal{O}$ that includes all constants used in Δ, the initial state $\mathcal{I}$ uses only constants from $\mathcal{O}_0$ and is consistent with $\mathcal{T}$, and the goal $\mathcal{G}$ is a closed ECQ using only constants from $\mathcal{O}_0$.

A ground action a is *applicable* in state s if $s, \mathcal{T} \models \mathtt{pre}$ and $\langle \mathcal{T}, s[\![a]\!] \rangle$ is consistent, where $s[\![a]\!]$ is defined as for PDDL, but replacing $\mathcal{R}(s) \models \theta(\mathtt{cond})$ by $s, \mathcal{T}, \theta \models \mathtt{cond}$. Plans Π are also defined as before, but the goal condition is now expressed as $\mathcal{I}[\![\Pi]\!], \mathcal{T} \models \mathcal{G}$. We use the same notation $[\![\cdot]\!]$ as for eKABs, since it is usually clear from the context which definition we are referring to.

As usual, we assume that all eKABs are *state-bounded*, i.e. there is a bound $b \in \mathbb{N}$ s.t. any state reachable from $\mathcal{I}$ contains at most b objects, to obtain manageable state transition systems (Calvanese et al. 2013; Giacomo et al. 2016). For simplicity, we can then assume that $\mathcal{O} = \mathcal{O}_0$ and denote both as $\mathcal{O}$. Additionally, as in (Borgwardt et al. 2022), we assume w.l.o.g. that the goal $\mathcal{G}$ consists of a single ground atom $g(\vec{c})$ (without brackets $[\cdot]$).

Example 5. *The* move *action from the introduction consists of the parameters* (x, y, z)*, precondition* $[\mathsf{on}(x, y)] \wedge \neg[\mathsf{Blocked}(x)] \wedge \neg[\mathsf{Blocked}(z)]$ *and effects*

$$((),\ [\mathsf{Block}(y)],\ \emptyset,\ \{\neg\mathsf{on\_block}(x, y)\}),$$
$$((),\ [\mathsf{Table}(y)],\ \emptyset,\ \{\neg\mathsf{on\_table}(x, y)\}),$$
$$((),\ [\mathsf{Block}(z)],\ \{\mathsf{on\_block}(x, z)\},\ \emptyset),$$
$$((),\ [\mathsf{Table}(z)],\ \{\mathsf{on\_table}(x, z)\},\ \emptyset),$$

which essentially remove $\mathsf{on}(x, y)$ *and add* $\mathsf{on}(x, z)$.

eKAB-to-PDDL Compilation A *compilation scheme* **f** (Thiébaux, Hoffmann, and Nebel 2005; Borgwardt et al. 2022) is a tuple of functions $(f_\delta, f_o, f_i, f_g)$ that translates an eKAB task $\mathcal{E} = (\Delta, \mathcal{O}, \mathcal{I}, \mathcal{G})$ to a PDDL task $F(\mathcal{E}) := (f_\delta(\Delta), \mathcal{O} \cup f_o(\Delta), f_i(\mathcal{O}, \mathcal{I}), f_g(\mathcal{O}, \mathcal{G}))$ s.t. a plan for $\mathcal{E}$ exists iff a plan for $F(\mathcal{E})$ exists, and f_i, f_g are computable in polynomial time. If the size of $f_\delta(\Delta)$ and $f_o(\Delta)$ is polynomial w.r.t. the size of Δ, we say that **f** is *polynomial*. If, or every plan Π for $\mathcal{E}$, there is a plan Π' for $F(\mathcal{E})$ s.t. $\|\Pi'\| \leq c \cdot \|\Pi\|^n + k$ where c, n, k are positive integers, then **f** *preserves plan size polynomially*. If $n = 1$ (and $c = 1$), then **f** preserves plan size *linearly* (*exactly*).

A state-bounded eKAB task $\mathcal{E} = ((\mathcal{P}, \mathcal{A}, \mathcal{T}), \mathcal{O}, \mathcal{I}, \mathcal{G})$ can be compiled into PDDL by viewing the Datalog$^\neg$-rewritings of all ECQs as FO-rules and additionally checking consistency via the rewriting of $\bot$.[2]

- Construct a copy $\mathcal{T}'$ of $\mathcal{T}$ and copies Q' of all ECQs Q in $\mathcal{E}$ by replacing each predicate p by p'.
- Define $\mathcal{R}' := \mathcal{R}_{\mathcal{T}'} \cup \{p'(\vec{x}) \leftarrow p(\vec{x}) \mid p \in \mathcal{P}\}$, where $\mathcal{R}_{\mathcal{T}'}$ is the disjoint union of $\mathcal{R}_{\mathcal{T}',\bot}$ and all $\mathcal{R}_{\mathcal{T}',Q'}$ for ECQs Q in $\mathcal{E}$ (cf. Proposition 1).
- Define $F(\mathcal{E}) := ((\mathcal{P}, \mathcal{P}', \mathcal{A}', \mathcal{R}'), \mathcal{O}, \mathcal{I}, \mathcal{G}')$, where $\mathcal{P}'$ contains all predicates from $\mathcal{R}_{\mathcal{T}'}$, $\mathcal{G}' := \neg P_\bot \wedge \mathcal{G}$, and $\mathcal{A}'$ is obtained from $\mathcal{A}$ by replacing all preconditions $\mathtt{pre}$ by $\neg P_\bot \wedge \mathtt{pre}'_{\mathcal{T}'}$ and effect conditions $\mathtt{cond}$ by $\mathtt{cond}'_{\mathcal{T}'}$.

The condition $\neg P_\bot$ checks consistency w.r.t. $\mathcal{T}$, as required by the semantics. For *DL-Lite* TBoxes, this is a polynomial compilation scheme that preserves plan size exactly (Calvanese et al. 2007a; Eiter et al. 2012; Borgwardt et al. 2022).

2.4 Instance-Level Coherence Update

An *(instance-level) update* for *DL-Lite* (De Giacomo et al. 2021) is a set $\mathcal{U}$ consisting of *insertions* $ins(P(\vec{c}))$ and *deletions* $del(P(\vec{c}))$, where $P(\vec{c})$ is an ABox assertion. For an update $\mathcal{U}$, the set of ABox assertions occurring in insertions (deletions) in $\mathcal{U}$ is denoted $A^+_\mathcal{U}$ ($A^-_\mathcal{U}$). Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a consistent *DL-Lite* ontology and $\mathcal{U}$ an update.

Definition 1. *An ABox* $\mathcal{A}'$ accomplishes the update *of* $\mathcal{K}$ *with* $\mathcal{U}$ *if* $\mathcal{A}' = \mathcal{A}'' \cup A^+_\mathcal{U}$ *for some maximal subset* $\mathcal{A}'' \subseteq \mathcal{A}$ *s.t.* $\mathcal{A}'$ *is consistent with* $\mathcal{T}$ *and* $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \beta$ *for all* $\beta \in A^-_\mathcal{U}$.

Intuitively, $\mathcal{A}'$ should differ from $\mathcal{A}$ as little as possible. Moreover, $\mathcal{U}$ cannot add and delete the same assertion (explicitly or implicitly), as described by the following result.

Proposition 2 (De Giacomo et al. 2021)**.** *An ABox* $\mathcal{A}'$ *accomplishing the update of* $\mathcal{K}$ *with* $\mathcal{U}$ *exists iff* $A^+_\mathcal{U}$ *is consistent with* $\mathcal{T}$ *and* $cl_\mathcal{T}(A^+_\mathcal{U}) \cap A^-_\mathcal{U} = \emptyset$*. Then,* $\mathcal{A}'$ *is unique.*

$\mathcal{U}$ is *compatible with* $\mathcal{T}$ if it satisfies these two conditions, which are independent of the ABoxes $\mathcal{A}, \mathcal{A}'$. To make the semantics independent of the syntax of $\mathcal{A}$, the following definition considers the $\mathcal{T}$-closure $cl_\mathcal{T}(\mathcal{A})$ instead of $\mathcal{A}$ itself.

Definition 2 (Coherence Update Semantics)**.** *If* $\mathcal{U}$ *is compatible with* $\mathcal{T}$*, the* result $\mathcal{K} \bullet \mathcal{U}$ of updating $\mathcal{K}$ with $\mathcal{U}$ *is* $\langle \mathcal{T}, \mathcal{A}'' \rangle$*, where* $\mathcal{A}''$ *is equivalent (w.r.t.* $\mathcal{T}$*) to the ABox* $\mathcal{A}'$ *that accomplishes the update of* $\langle \mathcal{T}, cl_\mathcal{T}(\mathcal{A}) \rangle$ *with* $\mathcal{U}$.

Since $\mathcal{A}''$ is unique up to equivalence w.r.t. $\mathcal{T}$, $\mathcal{K} \bullet \mathcal{U}$ is treated as unique in the following.

De Giacomo et al. (2021) construct a Datalog$^\neg$ program $\mathcal{R}^\mathsf{u}_\mathcal{T}$ to compute the effects of an update w.r.t. $\mathcal{T}$. For this, $\mathcal{A}$ and $\mathcal{U}$ are encoded into a single dataset $D_{\mathcal{U},\mathcal{A}}$ that contains all assertions from $\mathcal{A}$ as well as *ins_p_request*$(\vec{c})$ (*del_p_request*$(\vec{c})$) for each $ins(P(\vec{c}))$ ($del(P(\vec{c}))$) in $\mathcal{U}$. To check whether $\mathcal{U}$ is compatible with $\mathcal{T}$, $\mathcal{R}^\mathsf{u}_\mathcal{T}$ uses the predicate *incompatible_update*() that is derived when the conditions of Proposition 2 are violated. The derived predicates *ins_p*$(\vec{x})$ and *del_p*$(\vec{x})$ represent the final insertion and deletion operations needed to obtain $\mathcal{K} \bullet \mathcal{U}$ from $\mathcal{K}$.

This construction is correct in the sense that the resulting ABox $\mathcal{A}_{\mathcal{U},\mathcal{T}}$ is equivalent to $\mathcal{A}''$ from Definition 2, where $\mathcal{A}_{\mathcal{U},\mathcal{T}}$ is obtained from $\mathcal{A}$ by inserting (deleting) an assertion $P(\vec{o})$ iff $\mathcal{R}^\mathsf{u}_\mathcal{T}(D_{\mathcal{U},\mathcal{A}}) \models ins\_p(\vec{o})$ ($\mathcal{R}^\mathsf{u}_\mathcal{T}(D_{\mathcal{U},\mathcal{A}}) \models del\_p(\vec{o})$).

[2] This is a simplified version of the compilation from Borgwardt et al. (2022) that does not allow fresh objects in rewritings.

Proposition 3 (De Giacomo et al. 2021)**.** *$\mathcal{U}$ is compatible with $\mathcal{T}$ iff $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}(D_{\mathcal{U},\mathcal{A}}) \not\models$ incompatible_update(). In this case, for all assertions $P(\vec{o})$ over the signature of $\langle\mathcal{T},\mathcal{A}\rangle$, we have $\langle\mathcal{T},\mathcal{A}_{\mathcal{U},\mathcal{T}}\rangle \models P(\vec{o})$ iff $\mathcal{K}\bullet\mathcal{U} \models P(\vec{o})$.*

Example 6. *Following Example 5, we express the effect of* $\mathsf{move}(b_1,b_2,b_3)$ *by the update* $\mathcal{U} = \{del(\mathsf{on\_block}(b_1,b_2)), ins(\mathsf{on\_block}(b_1,b_3))\}$. *Using coherence update semantics, we do not have to distinguish the type of b_2 and can simply use $del(\mathsf{on}(b_1,b_2))$ instead, which yields the facts* $\mathsf{on\_block}(b_1,b_2)$, $\mathsf{on\_table}(b_3,t)$, *del_on_request*(b_1,b_2) *and ins_on_block_request*(b_1,b_3).

First, the program $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ translates the requests into direct insertions and deletions:

$$\begin{aligned} \textit{del\_on}(x,y) &\leftarrow \mathsf{on}(x,y), \textit{del\_on\_request}(x,y)\\ \textit{ins\_on\_block}(x,y) &\leftarrow \neg\mathsf{on\_block}(x,y),\\ &\qquad \textit{ins\_on\_block\_request}(x,y)\end{aligned}$$

The first rule has no effect, however, since $\mathsf{on}(b_1,b_2)$ *is not in the ABox. Instead, we have to remove* $\mathsf{on\_block}(b_1,b_2)$ *since* $\mathsf{on\_block} \sqsubseteq \mathsf{on} \in \mathcal{T}$ *(cf. (i) from Example 2):*

$$\textit{del\_on\_block}(x,y) \leftarrow \mathsf{on\_block}(x,y), \textit{del\_on\_request}(x,y)$$

Additionally, adding $\mathsf{on\_block}(b_1,b_3)$ *also ensures that* $\mathsf{on\_block}(b_1,b_2)$ *gets deleted, since otherwise the functionality of* on_block *would be violated (cf. (iii)):*

$$\begin{aligned}\textit{del\_on\_block}(x,y) \leftarrow\ & \textit{on\_block}(x,y),\\ & \textit{ins\_on\_block\_request}(x,z), y \neq z\end{aligned}$$

Finally, due to $\exists\mathsf{on\_block}^- \sqsubseteq \mathsf{Block} \in \mathcal{T}$*, the program retains the information* $\mathsf{Block}(b_2)$ *when* $\mathsf{on\_block}(b_1,b_2)$ *is deleted, by first deriving ins_block_closure*(b_2) *(cf. (ii)):*

$$\begin{aligned}\textit{ins\_block\_closure}(x) \leftarrow\ & \textit{del\_on\_block}(y,x), \neg\textit{Block}(x),\\ & \neg\textit{ins\_block\_request}(x),\\ & \neg\textit{del\_block\_request}(x)\end{aligned}$$

This is then translated into an insertion operation if there are no conflicting requests that would cause an inconsistency (recall that $\mathsf{Block} \sqsubseteq \neg\mathsf{Table} \in \mathcal{T}$*):*

$$\textit{ins\_block}(x) \leftarrow \textit{ins\_block\_closure}(x), \neg\textit{ins\_table\_request}(x)$$

In summary, the above rules derive ins_on_block(b_1,b_3)*, del_on_block*(b_1,b_2)*, and ins_block*(b_2).

In addition, the program $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ checks the conditions of Proposition 2, e.g. whether the same tuple is requested to be added to on_block *and removed from* on*:*

$$\begin{aligned}\textit{incompatible\_update}() \leftarrow\ & \textit{ins\_on\_block\_request}(x,y),\\ & \textit{del\_on\_request}(x,y)\end{aligned}$$

3 eKABs with Coherence Update Semantics

Since the construction of the coherence update semantics does not consider sequences of actions (De Giacomo et al. 2021), planning with the semantics was impossible as a single update might not achieve the goal property. To resolve this, we now define our new formalism, which we call *coherent eKABs (ceKABs)*. Syntactically, ceKAB domain descriptions and tasks $((\mathcal{P},\mathcal{A},\mathcal{T}),\mathcal{O},\mathcal{I},\mathcal{G})$ are exactly the same as for eKABs. However, the difference lies in the semantics of the action effects. First, we define the update $\mathcal{U}_a$ that is induced by an action a.

Definition 3. *Let $a = (\mathtt{pre},\mathtt{eff})$ be a ground action and s a state. If $s,\mathcal{T} \models \mathtt{pre}$, then the* associated update $\mathcal{U}_a$ *is the smallest set s.t. for each $(\vec{y},\mathtt{cond},\mathtt{add},\mathtt{del}) \in \mathtt{eff}$ and for each assignment θ with $s,\mathcal{T} \models \theta(\mathtt{cond})$:*

- *for all $\alpha \in \theta(\mathtt{add})$, we have $\alpha \in A^{add}_{\mathcal{U}_a}$; and*
- *for all $\neg\alpha \in \theta(\mathtt{del})$, we have $\alpha \in A^{del}_{\mathcal{U}_a}$.*

The ground action a is applicable *in s if $s,\mathcal{T} \models \mathtt{pre}$ and $\mathcal{U}_a$ is compatible with $\mathcal{T}$. The* application *of a to s is then the state $s[\![a]\!]$ for which $\langle\mathcal{T},s\rangle\bullet\mathcal{U}_a = \langle\mathcal{T},s[\![a]\!]\rangle$.*

As before, the operation $\langle\mathcal{T},s\rangle\bullet\mathcal{U}_a$ considers only the unary and binary predicates in s and ignores any predicates of higher arity. Moreover, recall that the resulting state $s[\![a]\!]$ is only unique up to equivalence w.r.t. $\mathcal{T}$ (see Definition 2). We again abuse the notation $[\![\cdot]\!]$ since the semantics we use is usually clear from the context.

In contrast to PDDL and eKABs, for ceKABs, an assertion may be contained in both $A^{add}_{\mathcal{U}_a}$ and $A^{del}_{\mathcal{U}_a}$, which could cause $\mathcal{U}_a$ to be incompatible with $\mathcal{T}$. Hence, actions need to be specified in such a way to avoid conflicting effects. Moreover, the compatibility requirement for $\mathcal{U}_a$ already ensures that $s[\![a]\!]$ is consistent with $\mathcal{T}$, and thus we do not need to check consistency of $\langle\mathcal{T},s[\![a]\!]\rangle$ as for eKABs.

The definition of $s[\![a]\!]$ can again be extended to sequences of ground actions as usual.

Definition 4. *A sequence of ground actions $\Pi = a_1,\ldots,a_n$ is a* coherence plan *for the ceKAB task $((\mathcal{P},\mathcal{A},\mathcal{T}),\mathcal{O},\mathcal{I},\mathcal{G})$ if $\mathcal{I}[\![\Pi]\!],\mathcal{T} \models \mathcal{G}$.*

We also define *coherence compilation schemes* as for eKABs, but using coherence plans instead of eKAB plans.

3.1 Compilation to PDDL

We now present a polynomial compilation scheme by extending the eKAB-to-PDDL compilation with the update rules $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ and introducing an additional action that implements the effects of each update in the state. We use the predicate *updating*() to indicate that an update is in progress. As in Section 2.3, we use the copy $\mathcal{T}'$ of $\mathcal{T}$ where all predicates are renamed, and correspondingly rename the ECQs Q occurring in conditions to Q'.

Definition 5. *Let $\mathcal{E} = ((\mathcal{P},\mathcal{A},\mathcal{T}),\mathcal{O},\mathcal{I},\mathcal{G})$ be a ceKAB task. The PDDL task $F(\mathcal{E}) := ((\mathcal{P},\mathcal{P}'_{\mathsf{u}},\mathcal{A}'_{\mathsf{u}},\mathcal{R}'_{\mathsf{u}}),\mathcal{O},\mathcal{I},\mathcal{G}'_{\mathsf{u}})$ is constructed as follows:*

(1) $\mathcal{R}'_{\mathsf{u}}$ contains $\mathcal{R}'$ from Section 2.3 and $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ from Section 2.4 as well as, for each predicate $p \in \mathcal{P}$, the rules

$$\begin{aligned}\textit{updating}() &\leftarrow \textit{ins\_p\_request}(\vec{x}),\\ \textit{updating}() &\leftarrow \textit{del\_p\_request}(\vec{x}),\end{aligned}$$

where $\vec{x}$ matches the arity of p.

(2) $\mathcal{P}'_{\mathsf{u}}$ contains all predicates from $\mathcal{R}'_{\mathsf{u}}$.

(3) For each action $a = (\vec{x},\mathtt{pre},\mathtt{eff}) \in \mathcal{A}$, $\mathcal{A}'_{\mathsf{u}}$ contains the request action *a' obtained by:*

- *peplacing $\mathtt{pre}$ by $\neg$updating() $\wedge\ \mathtt{pre}'_{\mathcal{T}'}$,*
- *for each effect $e = (\vec{y},\mathtt{cond},\mathtt{add},\mathtt{del}) \in \mathtt{eff}$,*
  - *replacing $\mathtt{cond}$ by $\mathtt{cond}'_{\mathcal{T}'}$,*

– *replacing* $\texttt{add}$ *by*

$$\{\textit{ins\_p\_request}(\vec{x}) \mid p(\vec{x}) \in \texttt{add}\} \cup$$
$$\{\textit{del\_p\_request}(\vec{x}) \mid \neg p(\vec{x}) \in \texttt{del}\} \textit{ and}$$

– *replacing* $\texttt{del}$ *by* $\emptyset$.

(4) $\mathcal{A}'_{\mathsf{u}}$ *contains* $a_{\textit{update}} = (\texttt{pre}_{\textit{update}}, \texttt{eff}_{\textit{update}})$*, where:*

- $\texttt{pre}_{\textit{update}} = \textit{updating}() \wedge \neg \textit{incompatible\_update}()$ *and*
- *for each predicate* $p \in \mathcal{P}$, $\texttt{eff}_{\textit{update}}$ *contains*
  – $(\vec{x},\ \textit{ins\_p}(\vec{x}),\ \{p(\vec{x})\},\ \emptyset)$,
  – $(\vec{x},\ \textit{del\_p}(\vec{x}),\ \emptyset,\ \{\neg p(\vec{x})\})$,
  – $(\vec{x},\ \textit{ins\_p\_request}(\vec{x}),\ \emptyset,\ \{\neg \textit{ins\_p\_request}(\vec{x})\})$ *and*
  – $(\vec{x},\ \textit{del\_p\_request}(\vec{x}),\ \emptyset,\ \{\neg \textit{del\_p\_request}(\vec{x})\})$.

(5) $\mathcal{G}'_{\mathsf{u}} = \neg \textit{updating}() \wedge \mathcal{G}$.

To show that this compilation scheme is correct, we first observe that the rewritings of the ECQs in conditions are not affected by the rules for the coherence update semantics. The full proof can be found in the supplementary material (Borgwardt, Nhu, and Röger 2025a).

Lemma 1. *Let* $Q(\vec{x})$ *be an ECQ and* $Q'_{\mathcal{T}'}(\vec{x})$ *be the FO-formula resulting from rewriting* Q' *w.r.t.* $\mathcal{T}'$*. Then, for any assignment* θ *and state* s *that is consistent with* $\mathcal{T}$*, we have*

$$s, \mathcal{T}, \theta \models Q(\vec{x}) \ \textit{ iff } \ \mathcal{R}'_{\mathsf{u}}(s) \models Q'_{\mathcal{T}'}(\theta(\vec{x}))$$

Proof sketch. This follows from the fact that the additional predicates introduced in $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ do not interfere with the Datalog$^{\neg}$ rules $\mathcal{R}_{\mathcal{T}',Q'} \subseteq \mathcal{R}'$ and the rewriting $Q'_{\mathcal{T}'}$ of Q' w.r.t. $\mathcal{T}'$. □

Using Lemma 1, we can show that the compilation scheme is correct.

Theorem 1. *Definition 5 describes a polynomial coherence compilation scheme from ceKABs to PDDL that preserves plan size linearly.*

Proof. The Datalog$^{\neg}$ program $\mathcal{R}'$ and the rewritings $Q'_{\mathcal{T}'}$ can be constructed in polynomial time since $\mathcal{T}$ is a *DL-Lite* TBox (Calvanese et al. 2007a; Eiter et al. 2012). The set $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ can also be constructed in polynomial time (De Giacomo et al. 2021) and the remaining constructions in Definition 5 are linear in the size of the input ceKAB.

We now prove that there is a coherence plan Π for $\mathcal{E}$ iff there is a plan Π' for $F(\mathcal{E})$.

$\Rightarrow$: Assume that there is a coherence plan $\Pi = \theta_1(a_1), \ldots, \theta_n(a_n)$ for $\mathcal{E}$, where θ_i ($i \in \{1, \ldots, n\}$) are assignments for grounding the action parameters, and let $s_0 = \mathcal{I}, s_1, \ldots, s_n$ be the resulting sequence of states of $\mathcal{E}$, where $s_n, \mathcal{T} \models \mathcal{G}$. Let Π' be a PDDL plan obtained as follows: First, we replace each action a_i with the corresponding action a'_i from Definition 5(2), using the same assignment θ_i. Second, we add the ground action $a_{\textit{update}}$ from Definition 5(3) after every action $\theta_i(a'_i)$. Then, the constructed sequence of ground actions Π' is of the form $\theta_1(a'_1), a_{\textit{update}}, \ldots, \theta_n(a'_n), a_{\textit{update}}$. Let $s_0^{\textit{update}} = \mathcal{I}, s'_1, s_1^{\textit{update}}, \ldots, s'_n, s_n^{\textit{update}}$ be the sequence of states resulting from Π'. We show that this sequence actually exists, i.e. every action is applicable in the corresponding state, and that, for every $i \in \{1, \ldots, n\}$, the state $s_i^{\textit{update}}$ of $F(\mathcal{E})$ is equivalent (w.r.t. $\mathcal{T}$) to the state s_i of $\mathcal{E}$, via induction on the length of Π.

$$\begin{array}{ccccccc} s_0 & , & s_1 & , \ldots , & & s_n \\ \| & & \| & & & \| \\ s_0^{\textit{update}} & , & s'_1, \ s_1^{\textit{update}} & & s'_n, & s_n^{\textit{update}} \end{array}$$

If $\|\Pi\| = 0$, then $s_0^{\textit{update}} = s_0 = \mathcal{I}$, and thus the claim holds. Assuming that the claim holds for $\|\Pi\| = n$, we now prove it for $\|\Pi\| = n + 1$.

By the induction hypothesis, $s_n^{\textit{update}}$ is equivalent (w.r.t. $\mathcal{T}$) to s_n, which does not contain any *request* predicates, and thus $\mathcal{R}'_{\mathsf{u}}(s_n^{\textit{update}}) \not\models \textit{updating}()$. Since $\theta_{n+1}(a_{n+1})$ is applicable in s_n, we further know that $s_n, \mathcal{T}, \theta_{n+1} \models \texttt{pre}$, where $\texttt{pre}$ is the precondition of a_{n+1}. Since $s_n^{\textit{update}}$ and s_n entail the same assertions, they also entail the same UCQs and ECQs, and thus, in particular, $s_n^{\textit{update}}, \mathcal{T}, \theta_{n+1} \models \texttt{pre}$. By Lemma 1 and the fact that the precondition of a'_{n+1} is $\neg \textit{updating}() \wedge \texttt{pre}'_{\mathcal{T}'}$, we obtain that $\theta_{n+1}(a'_{n+1})$ is applicable in $s_n^{\textit{update}}$.

We now consider the state s'_{n+1} resulting from applying $\theta_{n+1}(a'_{n+1})$ to $s_n^{\textit{update}}$. Due to Lemma 1, the request action $\theta_{n+1}(a'_{n+1})$ adds exactly the *request* atoms corresponding to the update $\mathcal{U}_{\theta_{n+1}(a_{n+1})}$ constructed as in Definition 3. We can assume w.l.o.g. that this actually adds or removes at least one ground atom, and thus $\mathcal{R}'_{\mathsf{u}}(s'_{n+1}) \models \textit{updating}()$. Moreover, since $\mathcal{U}_{\theta_{n+1}(a_{n+1})}$ is compatible with $\mathcal{T}$ and $\mathcal{R}'_{\mathsf{u}}$ contains $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$, by Proposition 3, $\neg \textit{incompatible\_update}()$ is true in $\mathcal{R}'_{\mathsf{u}}(s'_{n+1})$. Hence, the precondition of $a_{\textit{update}}$ is satisfied, rendering $a_{\textit{update}}$ applicable in s'_{n+1}.

Similarly, by Proposition 3, $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ derives exactly the facts for inserting and deleting ground atoms $p(\vec{c})$ (i.e. $ins\_p(\vec{c})$ or $del\_p(\vec{c})$) that are required for computing the result of the update $\mathcal{U}_{\theta_{n+1}(a_{n+1})}$ on $\langle \mathcal{T}, s_n^{\textit{update}} \rangle$. Since $cl_{\mathcal{T}}(s_n) = cl_{\mathcal{T}}(s_n^{\textit{update}})$, this result is equivalent to the result $\langle \mathcal{T}, s_{n+1} \rangle$ of $\mathcal{U}_{\theta_{n+1}(a_{n+1})}$ on $\langle \mathcal{T}, s_n \rangle$. Subsequently, the action $a_{\textit{update}}$ implements these changes on the atoms $p(\vec{c})$ and deletes all *request* atoms from s'_{n+1}, obtaining the state $s_{n+1}^{\textit{update}}$ that is equivalent to s_{n+1}, as claimed.

Finally, it follows from Lemma 1 and the above claim that the goal $\mathcal{G}'_{\mathsf{u}} = \neg \textit{updating}() \wedge \mathcal{G}$ is satisfied by $\mathcal{R}'_{\mathsf{u}}(s_n^{\textit{update}})$, i.e. Π' is a plan for $F(\mathcal{E})$. Incidentally, this shows that plan size is preserved linearly, namely by a factor of 2, since Π' has exactly twice the amount of actions as Π.

$\Leftarrow$: Let Π' be a plan for $F(\mathcal{E})$. We show that Π' must have the form $\theta_1(a'_1), a_{\textit{update}}, \ldots, \theta_n(a'_n), a_{\textit{update}}$ (for some assignments θ_i, $i \in \{1, \ldots, n\}$) and thus correspond to a plan $\Pi = \theta_1(a_1), \ldots, \theta_n(a_n)$ for $\mathcal{E}$ as above. Assuming again w.l.o.g. that each ground action $\theta_i(a'_i)$ actually adds at least one *request* atom, each such action causes $\textit{updating}()$ to become true, whereas $a_{\textit{update}}$ causes $\textit{updating}()$ to become false. Therefore, these kinds of actions must alternate. Moreover, since $\mathcal{I}$ does not contain any *request* atom, and $\mathcal{G}'_{\mathsf{u}}$ requires $\textit{updating}()$ to be false, the plan must begin with an action of the form $\theta_1(a'_1)$ and end with $a_{\textit{update}}$.

Let $s_0^{\textit{update}} = \mathcal{I}, s'_1, s_1^{\textit{update}}, \ldots, s'_n, s_n^{\textit{update}}$ and $s_0 = \mathcal{I}, s_1, \ldots, s_n$ be the sequences of states resulting from Π' and Π, respectively. As for the other direction, we show

that Π is applicable in s_0 and that any state s_i is equivalent (w.r.t. $\mathcal{T}$) to the state s_i^{update}, by induction on n. In the base case where $n = 0$, it is clear that both tasks share the same state $\mathcal{I} = s_0 = s_0^{update}$.

For the induction step from n to $n + 1$, assume that s_n is equivalent to s_n^{update} w.r.t. $\mathcal{T}$. Since $\theta_{n+1}(a'_{n+1})$ is applicable in s_n^{update}, we know that $\mathcal{R}'_{\mathsf{u}}(s_n^{update}) \models \mathtt{pre}'_{\mathcal{T}'}$, where $\mathtt{pre}$ is the precondition of a_{n+1}. By Lemma 1, we obtain $s_n^{update}, \mathcal{T}, \theta_{n+1} \models \mathtt{pre}$, and thus $s_n, \mathcal{T}, \theta_{n+1} \models \mathtt{pre}$. Since $\mathcal{U}_{\theta_{n+1}(a_{n+1})}$ is exactly characterized by the *request* atoms added to s'_{n+1} by $\theta_{n+1}(a'_{n+1})$ (see Definition 5) and a_{update} is applicable in s'_{n+1}, we know that $\mathcal{R}'_{\mathsf{u}}(s'_{n+1}) \not\models$ *incompatible_update*(). By Proposition 3, it follows that $\mathcal{U}_{\theta_{n+1}(a_{n+1})}$ is compatible with $\mathcal{T}$. This shows that $\theta_{n+1}(a_{n+1})$ is applicable in s_n.

Similarly, since the subsequent a_{update} action translates the *ins_p*$(\vec{c})$ and *del_p*$(\vec{c})$ atoms in $\mathcal{R}'_{\mathsf{u}}(s'_{n+1})$ into adding or deleting $p(\vec{c})$, respectively, we know that $\langle \mathcal{T}, s_{n+1}^{update} \rangle$ is equivalent to $\langle \mathcal{T}, s_n^{update} \rangle \bullet \mathcal{U}_{\theta_{n+1}(a_{n+1})}$, which is equivalent to $\langle \mathcal{T}, s_n \rangle \bullet \mathcal{U}_{\theta_{n+1}(a_{n+1})}$ by the inductive hypothesis, and thus to $\langle \mathcal{T}, s_{n+1} \rangle$ by Definition 3. This concludes the proof of the claim.

Finally, similar to the other direction, it follows from Lemma 1 and the fact that the final states of the two plans are equivalent that $\mathcal{G}$ is satisfied in the last state s_n. □

Extending this result to more expressive description logics is part of future work. However, we already know that, unless ExpTime$^{\text{NP}}$ = ExpTime, there cannot be a polynomial compilation scheme that preserves plan size polynomially for an extension of ceKABs based on Horn-$\mathcal{SROIQ}$ ontologies. The proof of the same result for eKABs (Borgwardt et al. 2022, Theorem 5) constructs a family of eKAB tasks containing a single action that adds a fresh nullary predicate $g()$ to the state. Since this predicate has no connection to the constructed ontologies, it does not matter which semantics we use for the action effects, and thus the result also holds for ceKABs.

Next, we address the *coherence plan existence* problem, which decides the following: Given a *DL-Lite* ceKAB task $\mathcal{E}$, is there a coherence plan Π' for $\mathcal{E}$? For this, we use a result of Erol, Nau, and Subrahmanian (1995) on the *plan existence* problem for classical planning (PDDL without derived predicates), which the authors showed to be ExpSpace-complete.

Theorem 2. *The coherence plan existence problem is* ExpSpace*-complete.*

Proof sketch. Membership holds since Definition 5 is a polynomial compilation scheme into PDDL with derived predicates, for which the problem in ExpSpace. This can be shown by a small adaptation of the proof of Erol, Nau, and Subrahmanian (1995) since evaluating derived predicates can be done in the same way as applying actions, and hence does not require more than exponential space.

Hardness is shown via a polynomial reduction from the plan existence problem for PDDL. Specifically, a PDDL task $\mathcal{E}'$ without derived predicates corresponds to a ceKAB $\mathcal{E}$ obtained by interpreting all FO-formulas as ECQs and adding an empty TBox $\mathcal{T}$. Additionally, one needs to prevent the case that action effects delete and insert the same fact, since the coherence update semantics forbids this. However, this can be avoided by adding conditional comparisons of affected facts. □

4 Experiments

We conducted a range of experiments to evaluate the feasibility of our compilation and its performance compared to the pure eKAB semantics. All code and additional benchmark data are available in the supplementary material (Borgwardt, Nhu, and Röger 2025b).

4.1 Implementation

We extend the existing implementation of the eKAB compilation[3] (Borgwardt et al. 2022), which uses *Clipper*[4] (Eiter et al. 2012) to rewrite the UCQs in ECQs. Since they have the same syntax, eKAB and ceKAB tasks are encoded in the same way, as a combination of a `.pddl` domain file, a `.pddl` problem file and a `.ttl` file for the ontology. Before the actual compilation, we compute the deductive closure $cl(\mathcal{T})$ using the Datalog engine *Nemo*[5] (Ivliev et al. 2024), based on the derivation rules of Borgida, Calvanese, and Rodriguez-Muro (2008). The set $cl(\mathcal{T})$ is then used to construct the update rules $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$. The output of the compilation is a standard PDDL task, consisting of two `.pddl` files without a TBox.

In addition to the compilation scheme from Definition 5 (variant *deriveUp*), we explored a logically equivalent variant of the compilation. Variant *setUp* replaces all the rules deriving *updating*() by directly adding/deleting *updating*() in action effects. Orthogonally to these variants, we also consider syntactically simpler conditions (in axiom bodies, action preconditions, action effects and the goal) constructed by means of a Tseitin transformation (Tseitin 1983).

4.2 Benchmarks

We adapted the classical Blocks benchmark and the eKAB benchmarks for DL-Lite from Borgwardt et al. (2022): Cats, Robot, Elevator, TPSA, VTA, VTA-Roles, and TaskAssign.

The Blocks instances are similar to our running example, except that there is a concept name Holding $\sqsubseteq$ Blocked for blocks that are currently held in the hand, and two actions pick-up and put-down instead of a single move action, which is closer to the original benchmark. We used the instances of the classical Blocks domain[6] to construct the problem files using a simple syntactic transformation.

Since the original instances of Cats and Robot do not have coherence plans, we used modified benchmarks that we denote by Cats* and Robot*. In Cats*, preconditions and ef-

[3] https://gitlab.perspicuous-computing.science/a.kovtunova/pddl-horndl

[4] https://github.com/ghxiao/clipper

[5] https://knowsys.github.io/nemo-doc/

[6] https://github.com/aibasel/downward-benchmarks/tree/master/blocks

fects are more explicit than in Cats, ensuring that packages are consistently disarmed or removed. Actions in the original Robot benchmark take advantage of the eKAB semantics, which allow insertion and deletion of the same facts, but insertion takes precedence. Since ceKABs do not allow this, Robot* explicitly makes the insertion and deletion effects disjoint. After these modifications, all benchmarks have plans under both eKAB and ceKAB semantics.

4.3 Evaluation

We used Downward Lab (Seipp et al. 2017) to conduct experiments with the Fast Downward planning system (Helmert 2006) on Intel Xeon Silver 4114 processors running at 2.2 GHz with a time limit of 30 minutes and a memory limit of 3 GiB per task. The compilation time was less than 1.2 s on all variants for the instance with the largest ontology file (Robot*).

For determining optimal plans, we would need an admissible heuristic, but the informative admissible heuristics in Fast Downward do not support derived predicates. For this reason, we focus on satisficing planning, using greedy best-first search (Doran and Michie 1966) with the FF heuristic (Hoffmann and Nebel 2001). While the original heuristic does not support derived predicates, either, Fast Downward implements two variants of supporting them. Both are based on the general idea to treat derived predicates like actions, which means that they do not have to be applied. However, the naive approach can lead to weak heuristic estimates if derived predicates are required to be false, e.g. in action preconditions or the goal. For this reason, the standard implementation uses a compilation that (only for the heuristic computation) adds additional derived predicates to capture the requirement that the derived predicate is false. Since this is infeasible for cyclic dependencies, the standard heuristic variant (FF) treats negative derived predicates in such cycles as undefined. As the transformation for the remaining derived predicates can still lead to a combinatorial explosion, Fast Downward also provides a more aggressive variant (FF), where all derived predicates can be made false for free. FF will typically provide better heuristic guidance, but at a higher computational cost.

To also consider the other extreme, we include experiments with the blind heuristic that assigns 1 to non-goal states and 0 to goal states. This heuristic is extremely fast to compute, but gives no guidance beyond detecting goal states. Since it is admissible, it can also be combined with the A* algorithm (Hart, Nilsson, and Raphael 1968) to obtain a configuration that guarantees optimal plans.

Comparing Heuristics Table 1 shows the number of solved instances (coverage) for the different configurations with greedy best-first search (GBFS). The first block shows the results for our basic implementation as in Definition 5. We observe that FF significantly outperforms FF, which runs out of memory on the Elevator and the VTA domain, as well as 13 of the Blocks instances. On the remaining Blocks tasks, as well as the unsolved instances of Robot* and TaskAssign, it runs out of time. Both can be attributed to the combinatorial explosion in the heuristic computation. If we look at the heuristic estimates for the initial states (as a proxy for heuristic guidance), we observe no significant advantage for FF on most domains but significantly better estimates on Cat* and Elevator, where FF has heuristic estimate 0, providing no guidance. In Elevator, the advantage does not outweigh the higher computational cost, but in Cat* the advantage translates into significantly faster solving times.

The different performance of FF on Cats* and Elevator may be due to the fact that their ontologies include functional roles. Although the Blocks domain also includes the functional role on_block, this may not play a large role in the planning tasks, since each action checks whether the target block is free before placing another block on top.

In the domains besides Cat* and Elevator, FF indeed provides heuristic guidance, which can also be seen from the numbers of expansions in comparison to the blind heuristic, shown in Figure 1 (left). The middle plot of the figure shows the resulting running times of the planner: for the Robot*, Blocks and TaskAssign domains, the better guidance translates into better overall performance. For VTA and VTA-Roles, the heuristic becomes more expensive to compute with increasing size of the instances, but the heuristic estimate for all initial states is 8, so we get a relatively weaker heuristic for higher computational cost. In terms of coverage, FF only has an advantage over blind search in Blocks and TPSA, but from the running time results it is to expect that, with harder tasks, it would also outperform it on Robot* and TaskAssign.

We also conducted experiments using A* with the blind heuristic, achieving the same coverage results as with GBFS but with the guarantee that these plans are optimal. Figure 1 (right) plots the optimal plan length against the one with GBFS and FF on the commonly solved tasks. On most domains, this satisficing configuration gives optimal plans with the two exceptions of Robot* and, most pronounced, Blocks, where the length can be more than 3 times the optimal length.

Variant for *updating* Predicate We now turn to the impact of variant *setUp* shown in the second block of the table. We do not include blind search because the coverage matches variant *deriveUp*. We observe that *setUp* has no impact on FF but leads to a significant improvement for FF: *updating*() is no longer a derived predicate that is required negatively in the goal and action preconditions, so it no longer causes a computational burden on the heuristic. It still is much worse than FF, in particular on the Blocks and VTA domains, but provides the best configuration so far on Elevator, solving all tasks of the domain.

Tseitin Transformation The middle block of Table 1 contains the analogue results with the additional Tseitin transformation. This modification leads to a slightly worse coverage with FF on some domains but provides an impressive boost for FF. It now solves almost all instances, and is only beaten by FF (without Tseitin) on Robot* by 3 instances. Variant *setUp* no longer leads to an advantage in coverage, and even has a detrimental effect on FF. Indeed, an analysis of the total solving times reveals that FF performs only better on Robot* and TaskAssign, but is significantly outper-

Coverage	*deriveUp* FF	*deriveUp* $\widetilde{\text{FF}}$	*deriveUp* blind	*setUp* FF	*setUp* $\widetilde{\text{FF}}$	*deriveUp* FF	*deriveUp* $\widetilde{\text{FF}}$	*deriveUp* blind	*setUp* FF	*setUp* $\widetilde{\text{FF}}$	eKAB FF	eKAB $\widetilde{\text{FF}}$
Blocks (34)	0	**34**	15	3	**34**	**34**	**34**	15	**34**	**34**	6	34
Cats* (20)	**20**	14	14	**20**	14	**20**	14	14	**20**	9	20	20
Elevator (20)	7	18	18	**20**	18	**20**	18	18	**20**	10	20	20
TPSA (15)	**15**	**15**	0	**15**	**15**	**15**	**15**	0	**15**	**15**	15	15
Robot* (20)	2	**20**	**20**	18	**20**	17	17	18	18	**20**	9	20
TaskAssign (20)	9	**20**	**20**	14	**20**	**20**	**20**	**20**	**20**	**20**	10	20
VTA (15)	0	**14**	**14**	5	**14**	**14**	**14**	**14**	**14**	**14**	2	15
VTA-Roles (15)	5	**14**	**14**	5	**14**	**14**	**14**	**14**	13	13	2	15
Sum (159)	58	149	115	100	149	**154**	146	113	**154**	135	84	159

Table 1: Number of solved instances with greedy best-first search, in the middle with the additional Tseitin transformation.

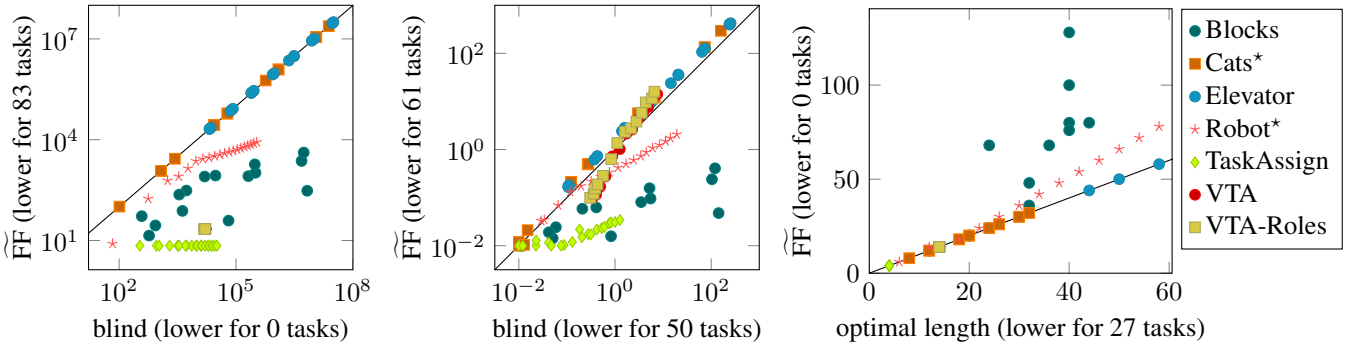

Figure 1: Analysis of $\widetilde{\text{FF}}$ (GBFS) on the *deriveUp* variant without Tseitin transformation. Comparison to blind (GBFS) in terms of expansions (left) and time (in seconds, middle) on the jointly solved tasks; resulting plan length compared to the optimal plan length (right).

formed by FF on Blocks, Cat* and Elevator.

Comparing ceKABs and eKABs The last block of Table 1 shows the coverage results for the analogous tasks generated with the eKAB-to-PDDL compilation by Borgwardt et al. (2022) without an additional Tseitin transformation. We see that also under that compilation, $\widetilde{\text{FF}}$ performs significantly better than FF. With a Tseitin transformation, both heuristics can solve all the benchmark tasks (not included in the table), which also the best configuration could not achieve on the ceKAB-to-PDDL compilation. We conclude that supporting coherence update semantics adds extra strain to the planning system.

5 Discussion and Conclusion

We have presented a novel combination of two semantics for planning with background ontologies: eKAB (epistemic) semantics for action conditions and coherence update semantics for single-step updates of *DL-Lite* ABoxes, taking into account implicit effects. We show that the resulting ceKAB semantics retains the favourable behaviour of both components, in particular allowing us to rewrite all operations into Datalog$^\neg$, and therefore into classical planning with derived predicates, in polynomial time. This also means that the plan existence problem remains in ExpSpace.

We conjecture that this would also hold for description logics other than *DL-Lite*$_{core}^{(\mathcal{HF})}$, as long as a suitable update semantics can be defined that can be expressed as a stratified Datalog$^\neg$ program. For example, in the Blocks domain it is natural to require that the transitive closure of on is irreflexive (Grundke, Röger, and Helmert 2024), which could be expressed in *DL-Lite*$_{core}^{(\mathcal{HF})^+}$ (Artale et al. 2009). Other possible extensions are to allow functional roles with subroles (e.g. on should be functional), nominals (e.g. Table should be a singleton set), or (restricted) Datalog rules like $\mathsf{on\_block}(x,y) \leftarrow \mathsf{on}(x,y) \wedge \mathsf{Block}(y)$ in the ontology. However, if the logic can express conjunction, it is unclear whether a suitable update semantics can be defined, since removing a conjunction involves a nondeterministic choice of which conjunct should be removed. Moreover, the coherence update semantics may not always be the most appropriate choice, so we will also study combined semantics that allow to switch, e.g. between eKAB and ceKAB semantics, for each insertion and deletion operation.

Lastly, in our evaluation we have seen many instances where heuristic search does not perform better than blind search, which indicates a weak support for derived predicates in the heuristic. This could be improved in the future by simplifying the Datalog$^\neg$ programs used in the compilations or by developing heuristics that better support the specific structure of the resulting derived predicates.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in grant 540204715 and by the Swiss National Science Foundation (SNSF) as part of the project "Practical Planning with Ontologies" (PPO).

References

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

Ahmetaj, S.; Calvanese, D.; Ortiz, M.; and Simkus, M. 2014. Managing change in graph-structured data using description logics (long version with appendix). *CoRR* abs/1404.4274.

Artale, A.; Calvanese, D.; Kontchakov, R.; and Zakharyaschev, M. 2009. The DL-Lite family and relations. *J. Artif. Intell. Res.* 36:1–69.

Borgida, A.; Calvanese, D.; and Rodriguez-Muro, M. 2008. Explanation in the DL-Lite family of description logics. In Meersman, R., and Tari, Z., eds., *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part II*, volume 5332 of *Lecture Notes in Computer Science*, 1440–1457. Springer.

Borgwardt, S.; Hoffmann, J.; Kovtunova, A.; and Steinmetz, M. 2021. Making DL-Lite planning practical. In Bienvenu, M.; Lakemeyer, G.; and Erdem, E., eds., *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, 641–645.

Borgwardt, S.; Hoffmann, J.; Kovtunova, A.; Krötzsch, M.; Nebel, B.; and Steinmetz, M. 2022. Expressivity of planning with horn description logic ontologies (technical report). *CoRR* abs/2203.09361.

Borgwardt, S.; Nhu, D.; and Röger, G. 2025a. Automated planning with ontologies under coherence update semantics (extended version).

Borgwardt, S.; Nhu, D.; and Röger, G. 2025b. Supplementary material to the paper "automated planning with ontologies under coherence update semantics". https://doi.org/10.5281/zenodo.16530898. Accessed: 2025-08-15.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007a. EQL-Lite: Effective first-order query processing in description logics. In Veloso, M. M., ed., *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 274–279.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007b. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reason.* 39(3):385–429.

Calvanese, D.; Giacomo, G. D.; Montali, M.; and Patrizi, F. 2013. Verification and synthesis in description logic based dynamic systems. In Faber, W., and Lembo, D., eds., *Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings*, volume 7994 of *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., *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 1022–1029. IJCAI/AAAI Press.

Corrêa, A. B.; De Giacomo, G.; Helmert, M.; and Rubin, S. 2024. Planning with object creation. In Bernardini, S., and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, 104–113. AAAI Press.

De Giacomo, G.; Oriol, X.; Rosati, R.; and Savo, D. F. 2021. Instance-level update in DL-Lite ontologies through first-order rewriting. *J. Artif. Intell. Res.* 70:1335–1371.

De Giacomo, G.; Favorito, M.; and Silo, L. 2024. LTLf goal-oriented service composition. In De Giacomo, G.; Fionda, V.; Fournier, F.; Ielo, A.; Limonad, L.; and Montali, M., eds., *Proceedings of the 3rd International Workshop on Process Management in the AI Era (PMAI 2024) co-located with 27th European Conference on Artificial Intelligence (ECAI 2024), Santiago de Compostela, Spain, October 19, 2024*, volume 3779 of *CEUR Workshop Proceedings*, 35–46. CEUR-WS.org.

Doran, J. E., and Michie, D. 1966. Experiments with the graph traverser program. *Proceedings of the Royal Society A* 294:235–259.

Eiter, T.; Ortiz, M.; Simkus, M.; Tran, T.; and Xiao, G. 2012. Query rewriting for Horn-SHIQ plus rules. In Hoffmann, J., and Selman, B., eds., *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, 726–733. AAAI Press.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.* 76(1-2):75–88.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.* 20:61–124.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Giacomo, G. D.; Lespérance, Y.; Patrizi, F.; and Vassos, S. 2016. Progression and verification of situation calculus agents with bounded beliefs. *Stud Logica* 104(4):705–739.

Grundke, C.; Röger, G.; and Helmert, M. 2024. Formal representations of classical planning domains. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 239–248. AAAI Press.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res.* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.* 14:253–302.

Ivliev, A.; Gerlach, L.; Meusel, S.; Steinberg, J.; and Krötzsch, M. 2024. Nemo: Your Friendly and Versatile Rule Reasoning Toolkit. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*, 21st International Conference on Principles of Knowledge Representation and Reasoning. International Joint Conferences on Artificial Intelligence Organization.

John, T., and Koopmann, P. 2024. Planning with OWL-DL ontologies. In Endriss, U.; Melo, F. S.; Bach, K.; Diz, A. J. B.; Alonso-Moral, J. M.; Barro, S.; and Heintz, F., eds., *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, 4165–4172. IOS Press.

Liu, H.; Lutz, C.; Milicic, M.; and Wolter, F. 2011. Foundations of instance level updates in expressive description logics. *Artif. Intell.* 175(18):2170–2197.

Röger, G., and Grundke, C. 2024. Negated occurrences of predicates in PDDL axiom bodies. In *Proceedings of the KI-2024 Workshop on Planning, Scheduling, Design and Configuration (PuK 2024)*.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artif. Intell.* 168(1-2):38–69.

Tseitin, G. S. 1983. On the complexity of derivation in propositional calculus. In Siekmann, J. H., and Wrightson, G., eds., *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Berlin, Heidelberg: Springer Berlin Heidelberg. 466–483.