Automated Planning with Ontologies under Coherence Update Semantics (Extended Abstract)*

Stefan Borgwardt[1] and Duy Nhu[1] Gabriele Röger[2]

[1] Institute of Theoretical Computer Science, Technische Universität Dresden, Germany
{stefan.borgwardt,hoang_duy.nhu}@tu-dresden.de
[2] Department of Mathematics and Computer Science, Universität Basel, Switzerland
gabriele.roeger@unibas.ch

Automated planning is a core area within Artificial Intelligence that describes the development of a system through the application of actions [9]. 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, i.e. absent facts are assumed to be false. A (ground) action is *applicable* if its precondition is satisfied in the current state w.r.t. an assignment of its variables. The objective is to select a sequence of applicable actions to reach the goal, called a *plan*. To facilitate expressive reasoning in the standard closed-world planning formalisms, logical theories under open-world semantics can be added to describe the possible interactions between objects of a domain of interest. Particularly, 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* [5]. 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 [5].

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

$$\mathsf{on\_block} \sqsubseteq \mathsf{on},\ \exists \mathsf{on\_block}^- \sqsubseteq \mathsf{Block},\ \mathtt{funct}\,\mathsf{on\_block},$$
$$\mathsf{on\_table} \sqsubseteq \mathsf{on},\ \exists \mathsf{on\_table}^- \sqsubseteq \mathsf{Table},\ \mathsf{Block} \sqsubseteq \neg\mathsf{Table},$$
$$\mathsf{Block} \equiv \exists \mathsf{on},\ \exists \mathsf{on\_block}^- \sqsubseteq \mathsf{Blocked},$$
$$\exists \mathsf{on\_block} \sqsubseteq \neg\exists \mathsf{on\_table},\ \mathsf{on\_block}(b_1, b_2),\ \mathsf{on\_table}(b_3, t)$$

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

* Full paper accepted at KR'25 [3,4]

($\exists \mathsf{on\_block}^- \sqsubseteq \mathsf{Blocked}$). On the other hand, we know that $\mathsf{on\_block}(b_1, b_3)$ cannot hold, since the $\mathsf{on\_block}$ relation is functional ($\texttt{funct}\,\mathsf{on\_block}$).

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 under epistemic semantics. In particular, the precondition requires that the ontology must entail that x is on top of y. Moreover, under the epistemic semantics, $\neg[\mathsf{Blocked}(x)]$ means that the ontology does not entail $\mathsf{Blocked}(x)$ (but not necessarily that $\neg\mathsf{Blocked}(x)$ is entailed). For example, the action is applicable for the substitution $x \mapsto b_1$, $y \mapsto b_2$, $z \mapsto b_3$, since $\mathsf{on\_block}$ is included in on and neither $\mathsf{Blocked}(b_1)$ nor $\mathsf{Blocked}(b_3)$ are entailed.

One property of eKABs is that action effects ignore implicit knowledge and only check whether the subsequent state is consistent with the TBox, which leads to the problems demonstrated by the next example.

Example 2.

(i) One effect of $\mathsf{move}(b_1, b_2, b_3)$ is to add $\mathsf{on\_block}(b_1, b_3)$ to the state (ABox), which would result in an inconsistent state, as argued previously, and therefore the action would not be applicable.
(ii) We could add the effect $\neg\mathsf{on}(b_1, b_2)$ to obtain a consistent state. However, since $\mathsf{on}(b_1, b_2)$ is not explicitly present in the state, this operation would not affect the state at all, and $[\mathsf{on}(b_1, b_2)]$ would continue to hold due to the presence of $\mathsf{on\_block}(b_1, b_2)$.
(iii) Moreover, even if we explicitly remove $\mathsf{on\_block}(b_1, b_2)$, we would lose the information that b_2 is a block.

This illustrates that, when executing actions, we have to take care of three types of implicit effects: adding a fact requires (i) removing any conflicting facts to ensure consistency, whereas removing a fact requires (ii) removing all stronger facts, and (iii) adding previously implied facts to avoid losing information. Addressing these challenges, the *coherence update semantics* was introduced for updating an ABox in the presence of a *DL-Lite* TBox, where the updated ABox can be computed with a non-recursive Datalog$^\neg$ program [6]. However, this semantics considers only single-step ABox updates, whereas, for planning, such implicit effects need to be considered for each action on the way to a goal.

Here, we consider *DL-Lite*$_{core}^{(\mathcal{HF})}$ [1] (simply *DL-Lite* in the following) and 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 introduce a novel compilation into PDDL with *derived predicates* by utilising the Datalog$^\neg$ programs for eKABs [2] and the coherence update semantics [6]. Moreover, we evaluate the feasibility of our approach and the overhead incurred compared to the original eKAB semantics in off-the-shelf planning systems.

In the following, we describe how the coherence update semantics deals with the implicit effects (i)–(iii). An *update* contains a set of *insertion* and *deletion* operations of ABox assertions. For instance, an update requesting the

deletion of $\mathsf{on}(b_1, b_2)$ and insertion of $\mathsf{on\_block}(b_1, b_3)$ can be represented by $\mathcal{U} = \{\mathit{del}(\mathsf{on}(b_1, b_2)), \mathit{ins}(\mathsf{on\_block}(b_1, b_3))\}$. The coherence update semantics [6] takes an ABox $\mathcal{A}$ and computes an updated ABox $\mathcal{A}'$ that differs from $\mathcal{A}$ as little as possible (*minimal change property*) and is unique up to equivalence w.r.t. $\mathcal{T}$.

Example 3. We express the effect of the action $\mathsf{move}(b_1, b_2, b_3)$ in Example 1 by the above update $\mathcal{U}$. To compute the effects of $\mathcal{U}$, a Datalog$^\neg$ program $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ is applied to an initial dataset containing the assertions from $\mathcal{A}$ as well as the translated *update requests* $\mathit{ins\_p\_request}(\boldsymbol{c})$ ($\mathit{del\_p\_request}(\boldsymbol{c})$) for each $\mathit{ins}(p(\boldsymbol{c}))$ ($\mathit{del}(p(\boldsymbol{c}))$ in $\mathcal{U}$ [6]. In our example, we obtain the initial facts $\mathsf{on\_block}(b_1, b_2)$, $\mathsf{on\_table}(b_3, t)$, $\mathit{del\_on\_request}(b_1, b_2)$ and $\mathit{ins\_on\_block\_request}(b_1, b_3)$.

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

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

However, the first rule has no effect, 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. (ii) from Example 2):

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

Additionally, adding $\mathsf{on\_block}(b_1, b_3)$ also ensures that $\mathsf{on\_block}(b_1, b_2)$ gets deleted (even if the request for deleting $\mathsf{on}(b_1, b_2)$ would be absent), since otherwise the functionality of $\mathsf{on\_block}$ would be violated (cf. (i)):

$$\mathit{del\_on\_block}(x, y) \leftarrow \mathsf{on\_block}(x, y),\ \mathit{ins\_on\_block\_request}(x, z),\ y \neq z$$

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 $\mathit{ins\_Block\_closure}(b_2)$ (cf. (iii)):

$$\mathit{ins\_Block\_closure}(x) \leftarrow \mathit{del\_on\_block}(y, x),\ \neg\mathsf{Block}(x),\ \neg\mathit{ins\_Block\_request}(x),\ \neg\mathit{del\_Block\_request}(x)$$

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

$$\mathit{ins\_Block}(x) \leftarrow \mathit{ins\_Block\_closure}(x), \neg\mathit{ins\_Table\_request}(x)$$

In summary, the above rules derive $\mathit{ins\_on\_block}(b_1, b_3)$, $\mathit{del\_on\_block}(b_1, b_2)$, and $\mathit{ins\_Block}(b_2)$ (cf. Example 2).

In addition, the program $\mathcal{R}^{\mathsf{u}}_{\mathcal{T}}$ checks whether the same tuple is requested to be added to $\mathsf{on\_block}$ and removed from on, as the coherence semantics forbids this:

$$\mathit{incompatible\_update}() \leftarrow \mathit{ins\_on\_block\_request}(x, y),\ \mathit{del\_on\_request}(x, y)$$

The constructed Datalog$^\neg$ program is non-recursive and polynomial in the size of the ontology [6].

For planning, we lift the coherence update semantics to apply it to all actions in a planning problem. Our ceKAB semantics combines the favourable behaviours of the epistemic eKAB semantics for action conditions and of the coherence update semantics for action effects. Apart from defining the semantics, our contributions are as follows.

A Polynomial Compilation Scheme for ceKABs. A compilation scheme translates a ceKAB planning task to a PDDL task s.t. a plan for the ceKAB exists iff a plan for the PDDL exists. Additionally, if the translation is polynomially bounded in the size of the eKAB task, then the compilation scheme is *polynomial*. We develop a polynomial compilation scheme by extending the known eKAB-to-PDDL compilation from [2]. Additionally, the size of the PDDL plans is linear in the size of the corresponding ceKAB plans.

Deciding Plan Existence for ceKABs. The *coherence plan existence* problem asks whether a plan exists for a *DL-Lite* ceKAB task. We study the complexity of the problem by means of a result by [8] on the *plan existence* problem for classical planning (PDDL without derived predicates), which is ExpSpace-complete. By our polynomial compilation scheme and a reduction in the other direction (PDDL-to-ceKAB), we can show that the same holds for the coherence plan existence problem.

Experimental Evaluation. We conduct a range of experiments to evaluate the feasibility of our compilation and its performance compared to the pure eKAB semantics [2]. Our benchmark collection consists of 159 instances, using the existing eKAB benchmarks for DL-Lite from [2], as well as the classical Blocks planning benchmark paired with an ontology. We modify some of the benchmarks to ensure that all benchmarks have plans under both eKAB and ceKAB semantics.

We use Downward Lab [12] to conduct experiments with the Fast Downward planning system [10]. Our main focus is satisficing planning using greedy best-first search [7] with the FF heuristic [11], as well as a more aggressive variant $\widetilde{\text{FF}}$ provided by Fast Downward, which provides less heuristic guidance, but is faster to compute. Considering an extreme case, we also experiment with the blind heuristic that simply assigns 1 to non-goal states and 0 to goal states.

On most of the benchmarks, we observe that $\widetilde{\text{FF}}$ significantly outperforms FF in terms of memory and CPU time, due to the combinatorial explosion in the computation of the FF heuristic. In many benchmark instances, heuristic search does not perform better than blind search, which indicates a weak support for derived predicates in the heuristics in general. Compared to the original eKAB-to-PDDL compilation [2], supporting coherence update semantics imposes extra strain on the planning system as it introduces more derived predicates to express the implicit effects.

In future work, we will try to extend ceKABs to support more expressive ontologies, and improve the planning performance 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.

References

1. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. Artif. Intell. Res. **36**, 1–69 (2009). https://doi.org/10.1613/JAIR.2820
2. Borgwardt, S., Hoffmann, J., Kovtunova, A., Krötzsch, M., Nebel, B., Steinmetz, M.: Expressivity of planning with horn description logic ontologies. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022. pp. 5503–5511. AAAI Press (2022). https://doi.org/10.1609/AAAI.V36I5.20489
3. Borgwardt, S., Nhu, D., Röger, G.: Automated planning with ontologies under coherence update semantics. In: Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning, KR 2025, November 11-17 (2025), to appear
4. Borgwardt, S., Nhu, D., Röger, G.: Automated planning with ontologies under coherence update semantics (2025), https://arxiv.org/abs/2507.15120
5. Calvanese, D., Montali, M., Patrizi, F., Stawowy, M.: 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. pp. 1022–1029. IJCAI/AAAI Press (2016), http://www.ijcai.org/Abstract/16/149
6. De Giacomo, G., Oriol, X., Rosati, R., Savo, D.F.: Instance-level update in DL-Lite ontologies through first-order rewriting. J. Artif. Intell. Res. **70**, 1335–1371 (2021). https://doi.org/10.1613/JAIR.1.12414
7. Doran, J.E., Michie, D.: Experiments with the graph traverser program. Proceedings of the Royal Society A **294**, 235–259 (1966)
8. Erol, K., Nau, D.S., Subrahmanian, V.S.: Complexity, decidability and undecidability results for domain-independent planning. Artif. Intell. **76**(1-2), 75–88 (1995). https://doi.org/10.1016/0004-3702(94)00080-K
9. Ghallab, M., Nau, D.S., Traverso, P.: Automated planning - theory and practice. Elsevier (2004)
10. Helmert, M.: The fast downward planning system. J. Artif. Intell. Res. **26**, 191–246 (2006). https://doi.org/10.1613/JAIR.1705
11. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. J. Artif. Intell. Res. **14**, 253–302 (2001). https://doi.org/10.1613/JAIR.855
12. Seipp, J., Pommerening, F., Sievers, S., Helmert, M.: Downward Lab. https://doi.org/10.5281/zenodo.790461 (2017)