Plan Synthesis for Knowledge and Action Bases*

Diego Calvanese, Marco Montali, Fabio Patrizi
Free Univ. of Bozen-Bolzano, Italy
lastname@inf.unibz.it

Michele Stawowy
IMT Lucca, Italy
michele.stawowy@imtlucca.it

Abstract

We study plan synthesis for a variant of Knowledge and Action Bases (KABs), a rich, dynamic framework, where states are description logic (DL) knowledge bases (KBs) whose extensional part is manipulated by actions that possibly introduce new objects from an infinite domain. We show that plan existence over KABs is undecidable even under severe restrictions. We then focus on state-bounded KABs, a class for which plan existence is decidable, and provide sound and complete plan synthesis algorithms, which combine techniques based on standard planning, DL query answering, and finite-state abstraction. All results hold for any DL with decidable query answering. We finally show that for lightweight DLs, plan synthesis can be compiled into standard ADL planning.

1 Introduction

Recently, there has been an increasing interest in formalisms that integrate static structural knowledge, as expressed, e.g., in description logics (DLs) [Baader *et al.*, 2003], with action-based mechanisms, to capture a domain's evolution over time. Combining these two aspects into a single logical system is known to be difficult and easily leading to undecidability, even for simple forms of inference about system dynamics, when the logics used are rather limited [Wolter and Zakharyaschev, 1999; Gabbay *et al.*, 2003]. To overcome these restrictions, a recent line of work by Bagheri Hariri *et al.* [2013b] has introduced Knowledge and Action Bases (KABs). Such systems rely on Levesque's functional approach [Levesque, 1984] to operate over a full-fledged DL knowledge base (KB), by means of actions: actions evolve the system by querying the current state using logical inference (ASK operation), and then using the derived facts to assert new knowledge in the resulting state (TELL operation).

A prominent feature of KABs is that actions allow one to incorporate into the KB external input provided by fresh objects taken from an infinite domain. This gives rise, in general, to an infinite-state system, in which reasoning is undecidable. Nevertheless, decidability of verification of first-order temporal properties has been obtained for KABs that are *state-bounded* [Bagheri Hariri *et al.*, 2013a; 2014], i.e., in which the number of objects in each single state is bounded a-priori, but is unbounded along single runs, and in the whole system.

Here, we study the problem of planning, specifically plan existence and synthesis [Ghallab *et al.*, 2004; Cimatti *et al.*, 2008], for knowledge-intensive dynamic systems over infinite domains. For doing so we consider a variation of KABs, termed *Explicit-Input KABs* (eKABs), more suited for our purposes, in which the input-related information for an action is made explicit in its signature, and not hidden in its conditional effects. In fact, eKABs can be considered as a concrete instantiation of the more abstract framework of Description Logic-based Dynamic Systems (DLDS) [Calvanese *et al.*, 2013b], and inherit its possibility of abstracting away the specific DL formalism used to capture the underlying KB.

We show that, in line with previous work on planning in rich settings [Erol *et al.*, 1995], plan existence is undecidable even for severely restricted eKABs. We prove decidability in PSPACE data complexity of such problem, for state-bounded eKABs, by combining techniques based on standard planning, DL query answering, and finite-state abstractions for DLDSs. Our work is similar in spirit to the one by Hoffmann *et al.* [2009], as combining a rich knowledge-based setting with the possibility of incorporating external input. However, to the best of our knowledge, our setting is the first one where planning is decidable without severe restrictions, in particular on how the external input is handled. We present a sound and complete algorithm, returning plans that represent templates for the original synthesis problem, defined over an infinite domain. We then concentrate on eKABs based on lightweight DLs of the *DL-Lite* family [Calvanese *et al.*, 2007b; 2009] and show that, in this case, plan synthesis can also be tackled by compilation into standard ADL, which can then be processed by any off-the-shelf ADL planner [Pednault, 1994].

2 Description Logic Knowledge Bases

Let Δ be a countably infinite universe of objects, acting as standard names [Levesque and Lakemeyer, 2001]. A (DL)

*We acknowledge the support of: Rip. Diritto allo Studio, Università e Ricerca Scientifica of Prov. Autonoma di Bolzano–Alto Adige, project VERISYNCOPATED; the EU FP7 large-scale integrating project Optique (FP7-IP-318338); and the UniBZ CRC projects KENDO and OnProm.

knowledge base (KB) $\langle T, A\rangle$ consists of a TBox T, capturing the intensional knowledge on the domain of interest, and an ABox A, capturing the extensional knowledge: T is a finite set of universal, first-order (FO) assertions based on concepts (unary predicates) and roles (binary relations); A is a finite set of *assertions*, or *facts*, i.e., atomic formulas $N(d)$ and $P(d, d')$, with N a concept name, P a role name, and $d, d' \in \Delta$. By $\text{ADOM}(A)$ we denote the set of objects occurring in A. For details on DLs, we refer to [Baader *et al.*, 2003]. To maintain a clear separation between the intensional and the extensional levels, we disallow *nominals* (concepts interpreted as singletons) in T.

We adopt the standard FO semantics for DL KBs. When $\langle T, A\rangle$ is *satisfiable*, i.e., admits at least one model, we also say that A is *T-consistent*. A KB $\langle T, A\rangle$ *logically implies* an ABox assertion α, written $\langle T, A\rangle \models \alpha$, if every model of $\langle T, A\rangle$ is also a model of α. Two ABoxes A_1 and A_2 are *logically equivalent modulo renaming wrt. a TBox T*, if they imply the same assertions, up to object renaming. When this holds, we write $A_1 \cong^h_T A_2$, where h denotes the bijection that renames the objects occurring in A_1 into those in A_2.

We use queries to extract information from a KB. A *union of conjunctive queries (UCQ)* q over a KB $\langle T, A\rangle$ is a FO formula $\bigvee_{1\leq i\leq n} \exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i})$, where $conj_i(\vec{x}, \vec{y_i})$ is a conjunction of equality assertions and atoms mentioning concept/role names of T, with variables from $\vec{x} \cup \vec{y_i}$ and objects as terms. A *boolean* UCQ is one without free variables. By $\text{ANS}(q, T, A)$ we denote the set of *(certain) answers* of a query q over a KB $\langle T, A\rangle$, consisting of all the substitutions θ of q's free variables with objects from $\text{ADOM}(A)$, s.t. $q\theta$ holds in every model of $\langle T, A\rangle$. We also consider the *EQL-Lite* extension of UCQs [Calvanese *et al.*, 2007a], referred to as *ECQ*s. The language of ECQs over a TBox T is:[1]

$$Q \;::=\; [q] \mid \neg Q \mid Q_1 \wedge Q_2 \mid \exists x.Q,$$

where q is a UCQ. The *certain answers* $\text{ANS}(Q, T, A)$ *of an ECQ Q over* $\langle T, A\rangle$ are obtained by first computing, for each atomic ECQ $[q]$, the certain answers of q, and then composing the obtained answers through the FO constructs in Q, with existential variables ranging over $\text{ADOM}(A)$. Hence $[q]$ acts as a *minimal knowledge operator*, and negation and quantification applied over UCQs are interpreted epistemically [Calvanese *et al.*, 2007a]. Under this semantics, ECQs are *generic*, in the sense of *genericity* in databases [Abiteboul *et al.*, 1995]: a query evaluated over two logically equivalent ABoxes returns the same answer, modulo object renaming.

As customary, we consider DLs for which query answering (and hence the standard reasoning tasks of KB satisfiability and logical implication) is decidable. In our examples, we use $\mathcal{ALCQIH}$ [Baader *et al.*, 2003]. We also consider lightweight DLs of the *DL-Lite* family, in particular *DL-Lite*$_\mathcal{A}$ [Calvanese *et al.*, 2009], for which checking KB satisfiability and answering ECQs over a KB are both *FO-rewritable* [Calvanese *et al.*, 2007a; 2009]. The latter means that every ECQ Q expressed over a *DL-Lite*$_\mathcal{A}$ TBox T can be effectively rewritten into a FO query $rew(Q, T)$ s.t., for every ABox A, the certain answers $\text{ANS}(Q, T, A)$ can be computed by *evaluating* $rew(Q, T)$ over A seen as a database under the *closed-world assumption*. Also, T-consistency of A can be checked by evaluating over A an ECQ Q^T_{unsat} (depending on T only).

[1] We consider ECQs that are *domain-independent* and $\langle T, A\rangle$-range restricted [Calvanese *et al.*, 2007a].

3 Explicit-input Knowledge and Action Bases

We define Explicit-input Knowledge and Action Bases (eKABs) parametrically wrt a DL $\mathcal{L}$. An $\mathcal{L}$-eKAB $\mathcal{K}$ is a tuple $\langle \mathcal{C}, \mathcal{C}_0, T, A_0, \Lambda, \Gamma\rangle$, where:

- $\mathcal{C} \subseteq \Delta$ is the (possibly infinite) *object domain* of $\mathcal{K}$;
- $\mathcal{C}_0 \subset \mathcal{C}$ is a *finite* set of *distinguished objects*;
- T is an $\mathcal{L}$-TBox;
- A_0 is an $\mathcal{L}$-ABox all of whose objects belong to $\mathcal{C}_0$;
- Λ is a finite set of *parametric actions*;
- Γ is a finite set of *condition-action rules*.

When the specific DL $\mathcal{L}$ is irrelevant, we omit it.

A *parametric action* α has the form $a(\vec{p}) : \{e_1, \ldots, e_n\}$, where: a is the action's *name*, $\vec{p}$ are the *input parameters*, and $\{e_1, \ldots, e_n\}$ is the finite set of *conditional effects*. Each *effect* e_i has the form $Q_i \rightsquigarrow \textbf{add } F_i^+, \textbf{del } F_i^-$, where:

- Q_i is the *effect condition*, i.e., an ECQ over T, whose terms can be action parameters $\vec{p}$ (acting as variables), additional free variables $\vec{x}_i$, or objects from $\mathcal{C}_0$;
- F_i^+ and F_i^- are two sets of atoms over the vocabulary of T, with terms from $\vec{p} \cup \vec{x}_i \cup \mathcal{C}_0$.

Given an action α and a substitution θ assigning objects of $\mathcal{C}$ to $\vec{p}$, $\alpha\theta$ denotes the *action instance* of α obtained by assigning values to $\vec{p}$ according to θ. We write $\alpha(\vec{p})$ to explicitly name the input parameters of α. The ABox resulting from the *application* of an action instance $\alpha\theta$ on an ABox A, denoted $\text{DO}(\alpha\theta, A, T)$, is the ABox $(A \setminus A^-_{\alpha\theta}) \cup A^+_{\alpha\theta}$, where:

- $A^+_{\alpha\theta} = \bigcup_{i\in\{1,\ldots,n\}} \bigcup_{\sigma\in \text{ANS}(Q_i\theta, T, A)} F_i^+\sigma$;
- $A^-_{\alpha\theta} = \bigcup_{i\in\{1,\ldots,n\}} \bigcup_{\sigma\in \text{ANS}(Q_i\theta, T, A)} F_i^-\sigma$.

Importantly, $\alpha\theta$ is *applicable* to A only if $\text{DO}(\alpha\theta, A, T)$ is consistent with T.

A *condition-action rule (CA rule)* γ_α for an action α has the form $Q_\alpha(\vec{x}) \mapsto \alpha(\vec{p})$, where Q_α is an ECQ mentioning only objects from $\mathcal{C}_0$ and whose free variables $\vec{x}$ come from $\vec{p}$. We assume that each action α in Λ has exactly one corresponding condition-action rule γ_α in Γ (multiple rules can be combined into a single disjunctive rule). The ECQ Q_α is used to constrain the set of action instances potentially applicable to a certain Abox A. Specifically, let θ be a parameter substitution for $\vec{p}$, and let $\theta[\vec{x}]$ denote the parameter substitution obtained by projecting θ on $\vec{x}$. Then, θ is a *$\mathcal{K}$-legal parameter substitution in A for* α, if:

- $\theta : \vec{p} \to \mathcal{C}$;
- $\theta[\vec{x}] \in \text{ANS}(Q_\alpha, T, A)$;
- $\text{DO}(\alpha\theta, A, T)$ is T-consistent.

When this is the case, $\alpha\theta$ is a *$\mathcal{K}$-legal action instance in A*. Any variable in $\vec{p} \setminus \vec{x}$ is also referred to as an *external input parameter* of α. Note that, when present, such parameters

are not constrained by Q_α and can be assigned to any object, including fresh ones not occurring in $\text{ADOM}(A)$.

The *semantics* of eKABs is defined in terms of transition systems (TSs) with states and transitions labeled by ABoxes and action instances, respectively [Calvanese *et al.*, 2013b]. A TS Υ is a tuple $\langle \mathcal{C}, T, \Sigma, s_0, abox, \rightarrow \rangle$, where:

- $\mathcal{C}$ is the *object domain*;
- T is a TBox;
- Σ is a (possibly infinite) *set of states*;
- $s_0 \in \Sigma$ is the *initial state*;
- $abox$ is the *labeling function*, mapping states from Σ into T-consistent ABoxes with terms from $\mathcal{C}$ only;
- $\rightarrow \subseteq \Sigma \times L \times \Sigma$ is a labeled *transition relation*, with L the (possibly infinite) set of labels.

We write $s \xrightarrow{l} s'$ as a shortcut for $\langle s, l, s' \rangle \in \rightarrow$.

An eKAB $\mathcal{K} = \langle \mathcal{C}, \mathcal{C}_0, T, A_0, \Lambda, \Gamma \rangle$ *generates* a TS $\Upsilon_\mathcal{K} = \langle \mathcal{C}, T, \Sigma, s_0, abox, \rightarrow \rangle$, where:

- $abox$ is the identity function;
- $s_0 = A_0$;
- $\rightarrow \subseteq \Sigma \times L_\mathcal{K} \times \Sigma$ is a labelled transition relation with $L_\mathcal{K}$ the set of all possible action instances;
- Σ and $\rightarrow$ are defined by mutual induction as the smallest sets s.t. if $A \in \Sigma$, then for every $\mathcal{K}$-legal action instance $\alpha\theta$ in A, we have that $\text{DO}(\alpha\theta, A, T) \in \Sigma$ and $A \xrightarrow{\alpha\theta} \text{DO}(\alpha\theta, A, T)$.

In general, $\Upsilon_\mathcal{K}$ is infinite, if $\mathcal{C}$ is so.

Example 1. *An eKAB $\mathcal{K} = \langle \mathcal{C}, \mathcal{C}_0, T, A_0, \Lambda, \Gamma \rangle$ is used to support decision making in a company, where employees work on tasks. $\mathcal{C}$ contains infinitely many objects. T is expressed in $\mathcal{ALCQIH}$, and contains the following axioms:*

- *$Eng \sqsubseteq Emp$ and $Tech \sqsubseteq Emp$ (engineers and technicians are employees);*
- *$\exists hasTask^- \sqsubseteq Emp$ and $\exists hasTask \sqsubseteq Task$ ($hasTask$ role links employees to tasks),*
- *similar axioms can be used to express that $worksIn$ links employees to company branches;*
- *$Emp \sqsubseteq (= 1\, worksIn)$ (employees work in exactly one one branch);*
- *$hasResp \sqsubseteq (\leq 1\, Emp)$ and $hasResp \sqsubseteq hasTask^-$ (a task may have at most one responsible, among the employees associated to the task);*
- *$\exists hasResp^- \sqsubseteq \neg Tech$ (technicians are not task responsible).*

As for the dynamic aspects, to model that a new engineer can be hired in a branch provided that the planning agent does not know whether there already exists an engineer there, $\mathcal{K}$ contains the rule

$$Branch(b) \wedge \neg[\exists x. Eng(x) \wedge worksIn(x, b)] \mapsto \mathsf{HireEng}(x, b)$$

where:

$$\mathsf{HireEng}(e, b) : \{\mathsf{true} \rightsquigarrow \textbf{\textit{add}}\, \{Eng(e), worksIn(e, b)\}\}$$

A similar action $\mathsf{HireTech}(t, b)$ can be used to hire a technician. Rule

$$Task(t) \wedge Emp(e) \mapsto \mathsf{MakeResp}(t, e)$$

states that an employee can be made responsible of a task, where

$$\mathsf{MakeResp}(t, e) : \left\{ \begin{array}{r} hasResp(t, p) \rightsquigarrow \textbf{\textit{del}}\{hasResp(t, p)\} \\ \mathsf{true} \rightsquigarrow \textbf{\textit{add}}\{hasResp(t, e)\} \end{array} \right\}$$

removes the previous task responsible, if any, and makes the selected employee the new responsible. Finally, rule

$$Emp(e) \mapsto \mathsf{Anon}(e)$$

models that an employee can be anonymized, where action

$$\mathsf{Anon}(e) : \{worksIn(e, b) \rightsquigarrow \textbf{\textit{del}}\{worksIn(e, b)\}\}$$

models anonymization by removing the explicit information on the branch to which the selected employee belongs.

Note that there is a complex interplay between the TBox and the dynamic component of $\mathcal{K}$. E.g., the last TBox axioms forbids to hire a technician and make it responsible for a task. However, notice that this is not explicitly forbidden in the condition-action rule that defines the (potential) applicability of the MakeResp *action.* ∎

4 eKAB planning

Let $\mathcal{K}$ be an eKAB and $\Upsilon_\mathcal{K}$ its generated TS. A *plan for* $\mathcal{K}$ is a finite sequence $\pi = \alpha_1\theta_1 \cdots \alpha_n\theta_n$ of action instances over $L_\mathcal{K}$. The plan π is *executable* on $\mathcal{K}$ if there exists a (unique) run $\rho = A_0 \xrightarrow{\alpha_1\theta_1} A_1 \xrightarrow{\alpha_2\theta_2} \cdots \xrightarrow{\alpha_n\theta_n} A_n$ of $\mathcal{K}$. We call ρ the run *induced* by π, and A_n the *final state* of ρ. An *eKAB planning problem* is a pair $\langle \mathcal{K}, G \rangle$, with $\mathcal{K}$ an eKAB and G, the *goal*, a boolean ECQ mentioning only objects from $\mathcal{C}_0$. A plan π for $\mathcal{K}$ *achieves* G, if $\text{ANS}(G, A_n, T) = \mathsf{true}$, for A_n the final state of the run induced by π.

Example 2. *Given the eKAB in Example 1, a goal G could express the intention to have an engineer and a technician working for a given task* t*, provided that, for privacy reasons, it is not known to the planning agent whether the two work in the same branch. Formally:*

$$\begin{array}{l} \exists e_1, e_2. Tech(e_1) \wedge Eng(e_2) \\ \qquad \wedge hasTask(e_1, \mathtt{t}) \wedge hasTask(e_2, \mathtt{t}) \\ \qquad \wedge \neg[\exists b. worksIn(e_1, b) \wedge worksIn(e_2, b)] \end{array}$$

where negation is in fact interpreted epistemically, in accordance with the semantics of ECQs. ∎

We first consider *plan existence*, i.e., the problem of checking whether, for a planning problem $\langle \mathcal{K}, G \rangle$, an executable plan π for $\mathcal{K}$ exists that achieves G.

Theorem 4.1. *Let $\mathcal{L}$ be a DL for which answering ECQs*[2] *is in* CONP *in data complexity. Then plan existence for $\mathcal{L}$-eKABs with a finite object domain is decidable in* PSPACE *in the size of the object domain.*

[2]For a significant class of DLs, UCQ-query answering is known to be in CONP in data complexity [Glimm *et al.*, 2008; Ortiz *et al.*, 2008; Calvanese *et al.*, 2013a]. ECQs inherit this result, since they simply combine the certain answers returned by the embedded UCQs with the evaluation of the FO operators present in the query.

When the object domain is infinite, plan existence is undecidable even by considering severely limited eKABs. The following result is similar in spirit to the undecidability result by Erol *et al.* [1995] in the classical planning setting but with an infinite object domain. Also, a similar result was presented in [Zarrieß and Claßen, 2015], in a knowledge-based setting that is radically different from ours, since in their case the TBox is considered only in the initial state, while subsequent states may violate its assertions. This makes the proof technique significantly more complex in our case.

Theorem 4.2. *Plan existence for eKABs with an infinite object domain is undecidable, even if the goal is a ground CQ, and the input eKAB has: (i) an empty TBox; (ii) actions/rules employing UCQs only.*

We next attack undecidability by focusing on infinite-domain eKABs that are *state-bounded* [Bagheri Hariri *et al.*, 2013a; Calvanese *et al.*, 2013b; Belardinelli *et al.*, 2014]. Specifically, an eKAB $\mathcal{K}$ is *b-bounded* if its generated TS $\Upsilon_{\mathcal{K}} = \langle T, \Sigma, s_0, abox, \rightarrow \rangle$ is s.t. for every state $s \in \Sigma$, we have $|\text{ADOM}(abox(s))| \leq b$, that is, every state (or ABox) of $\Upsilon_{\mathcal{K}}$ contains at most b distinct objects. Note that a b-bounded eKAB still has, in general, infinitely many states. Indeed, from an infinite $\mathcal{C}$, one can obtain infinitely many distinct ABoxes, each containing a bounded number of objects.

For state-bounded eKABs, we have the following result, which follows from the verification results by Calvanese *et al.* [2013b]. The result implies that, for state-bounded eKABs, checking plan existence is not more difficult than in the standard setting of propositional planning [Bylander, 1994].

Theorem 4.3. *Plan existence over state b-bounded eKABs is decidable in* PSPACE *in the bound b.*

We observe that while boundedness is undecidable in general (by reduction to checking whether a Turing Machine uses a bounded number of cells on a given input), checking whether an eKAB is bounded *for a given bound* is decidable, as proven in [De Giacomo *et al.*, 2014], although in the Situation Calculus framework.

5 Plan Synthesis

We now consider plan synthesis for eKABs. We first introduce a technique based on classical planning [Brachman and Levesque, 2003]. A *classical planning domain* is a triple $\mathcal{D} = \langle S, A, \rho \rangle$, where S is a finite set of *states*, A is a finite set of *actions*, and $\rho : S \times A \rightarrow S$ is a *transition function*. Domain states are propositional assignments, and actions are operators that change them, according to ρ. A *classical planning problem* is a triple $\mathcal{P} = \langle \mathcal{D}, s_0, G \rangle$, where $\mathcal{D}$ is a planning domain, $s_0 \in S$ is the *initial state*, and $G \subseteq S$ is a set of *goal states*. A solution to $\mathcal{P}$ is a finite sequence of actions, called *plan*, that takes $\mathcal{D}$ from s_0 to a goal state, as a result of the changes made by the sequence of actions.

A problem $\mathcal{P}$ induces a labelled graph $\mathcal{G} = \langle N, E \rangle$, where $N = S$ and E contains a (labelled) edge $s \xrightarrow{a} s'$ iff $s' = \rho(s, a)$. Essentially, planning algorithms amount to searching (in an effective way) a path in $\mathcal{G}$ from s_0 to a *goal* state.

Algorithm 1 Forward planning algorithm schema

```
1: function FindPlan(Prob)
2: input: A planning problem Prob
3: output: A plan that solves Prob, or fail if there is no solution
4:     V := ∅                                   ▷ Global set of visited states
5:     return FwSearch(Prob,initialState(Prob),ϵ)
6: function FwSearch(Prob, s, π)
7: input: A planning problem Prob, the current state s, and the
   sequence π of actions that led to s
8: output: A plan that solves Prob, or fail if there is no solution
9:     if s ∈ V then return fail                ▷ Loop!
10:    V := V ∪ {s}
11:    if holds(goal(Prob), s) then return π
12:    for all ⟨a, s'⟩ ∈ successors(Prob, s) do
13:        π_n := FwSearch(Prob, s', π · a)
14:        if π_n ≠ fail then return π_n
15:    return fail
```

Algorithm 1 shows the schema of a basic algorithm that synthesizes a plan through a forward search on the induced graph. The auxiliary functions $\text{INITIALSTATE}(Prob)$, $\text{GOAL}(Prob)$, $\text{HOLDS}(G, s)$, and $\text{SUCCESSORS}(Prob, s)$ are self-explicative. Note that the schema above abstracts from the specific interpretation of states, that is, it is applicable no matter how states and actions are represented, once the functions above are instantiated on the case at hand. For instance, for a classical planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, G \rangle$ with $\mathcal{D} = \langle S, A, \rho \rangle$, we have $\text{INITIALSTATE}(\mathcal{P}) = s_0$, $\text{GOAL}(\mathcal{P}) = G$, $\text{HOLDS}(G, s) = \text{true}$ iff $s \in G$, and $\text{SUCCESSORS}(\mathcal{P}, s) = \{\langle a, s' \rangle \mid s' = \rho(s, a)\}$.

Next, we show how this schema can be lifted to handle plan synthesis for eKABs. The lifting we propose results in an algorithm that is *correct*, i.e., it

(i) terminates,

(ii) preserves plan existence, and

(iii) produces proper plans, i.e., if it does not fail, the returned result corresponds to a proper solution to the input planning problem.

We stress that while we present the lifting on Algorithm 1, the same approach applies to any other, possibly optimized, algorithm. Indeed, the lifting strategy is agnostic wrt how the search space is traversed.

5.1 Plan Synthesis for eKABs

We consider the two classes of eKABs for which decidability of plan existence has been established in Section 4.

eKABs with Finite Domain. This case differs from classical planning in that eKABs have ABoxes as states, and they provide an implicit representation of successor states in terms of actions and condition-action rules. The former aspect requires to replace propositional entailment with ECQ query answering when checking whether the goal has been reached. The latter requires to use DO to compute a state's successors. Specifically, given an eKAB planning problem $\mathcal{E} = \langle \mathcal{K}, G \rangle$, where $\mathcal{K} = \langle \mathcal{C}, \mathcal{C}_0, T, A_0, \Lambda, \Gamma \rangle$, with finite $\mathcal{C}$, Algorithm 1 can be instantiated as follows:

- $\text{INITIALSTATE}(\mathcal{E}) = A_0$;

Algorithm 2 Plan synthesis for state-bounded eKABs.

function FINDPLAN-EKABSB($\mathcal{E}, b$)
input: An eKAB planning problem $\mathcal{E} = \langle \mathcal{K}, G \rangle$, where $\mathcal{K} = \langle \mathcal{C}, \mathcal{C}_0, T, A_0, \Lambda, \Gamma \rangle$ is b-bounded and $\mathcal{C}$ is infinite
output: A plan that solves $\mathcal{E}$, or fail if there is no solution
  $n := max\{k \mid$ there is $\alpha \in \Lambda$ with k parameters$\}$
  pick $\widehat{\mathcal{C}}$ s.t. $\begin{cases} \mathcal{C}_0 \subseteq \widehat{\mathcal{C}} \subset \mathcal{C} \\ |\widehat{\mathcal{C}}| = b + n + |\mathcal{C}_0| \end{cases}$ ▷ Abstract dom.
  $\widehat{\mathcal{K}} := \langle \widehat{\mathcal{C}}, \mathcal{C}_0, T, A_0, \Lambda, \Gamma \rangle$
  return FINDPLAN-EKABFD($\langle \widehat{\mathcal{K}}, G \rangle$)

- GOAL$(\mathcal{E}) = G$;
- HOLDS$(\mathcal{E}, A) =$ ANS(G, T, A);
- SUCCESSORS$(\mathcal{E}, A)$ returns the set of pairs $\langle \alpha\theta, A' \rangle$, where
  - $\alpha \in \Lambda$,
  - θ is a $\mathcal{K}$-legal parameter substitution in A for α, and
  - $A' =$ DO$(\alpha\theta, A, T)$.

We call the resulting algorithm FINDPLAN-EKABFD. By this instantiation, the search space of FINDPLAN-EKABFD is exactly $\Upsilon_{\mathcal{K}}$, which is finite, so being the eKAB domain. Thus, we have:

Theorem 5.1. FINDPLAN-EKABFD *is correct.*

Plan Synthesis for State-Bounded eKABs. We now consider state-bounded eKABs over infinite object domains. In this case, the search space is potentially infinite, thus FINDPLAN-EKABFD is not readily applicable, as termination is not guaranteed. To tackle this problem, instead of visiting the original, infinite, search space, we work on a finite-state abstraction of the eKAB. We first argue that the execution semantics of eKABs has two properties:

- it is driven by ECQ-query answering, which is generic (cf. Section 2);
- all allowed configurations of external input parameters are considered when applying an action.

These imply that eKABs are *generic DLDSs* in the sense of Calvanese *et al.* [2013b], which, together with state-boundedness, allows us to apply the same abstraction technique used by Calvanese *et al.* [2013b]. In particular, reachability is preserved if the infinite-state TS induced by the input eKAB is shrunk into a finite-state TS over a finite, but sufficiently large, object domain. We leverage this technique as the core of the FINDPLAN-EKABSB procedure shown in Algorithm 2, which reduces the original planning problem over an infinite search space to a classical planning problem.

It can be checked that all correctness conditions are satisfied: *(i)* is a consequence of Theorem 5.1; *(ii)* holds because the finite-state abstraction preserves reachability; *(iii)* holds because the search space of the finite-state abstraction is contained into that of the original eKAB. Thus, we have:

Theorem 5.2. FINDPLAN-EKABSB *is correct.*

Example 3. *Consider again the eKAB $\mathcal{K}$ in Example 1, together with goal G in Example 2. Notice that $\mathcal{K}$ is state-bounded, as the only way to increase the number of objects present in the system is by hiring someone, but the number of hirings is in turn bounded by the number of company branches (which is fixed once and for all in the initial state). Now assume that the initial state indicates the known existence of a main and a subsidiary branch for the company, as well as the fact that task* t *has an assigned technician from the main branch:*

$$A_0 = \{Branch(\texttt{main}), Branch(\texttt{sub}), Tech(\texttt{123}), worksIn(\texttt{123}, \texttt{main}), hasTask(\texttt{123}, \texttt{t})\}$$

A possible plan leading from A_0 to a state where G holds is

$$\pi_1 = \mathsf{HireEng}(\texttt{452}, \texttt{sub})\ \mathsf{MakeResp}(\texttt{t}, \texttt{452})$$

This plan achieves G by hiring an engineer in a different branch from that of 123*, with whom the engineer shares task* t*. Observe, again, the interplay between the actions and the TBox: while no action explicitly indicates that* 452 *has task* t*, this is implicitly obtained from the fact that such an engineer is made responsible for* t*.*

Another, quite interesting plan achieving G is:

$$\pi_2 = \mathsf{HireEng}(\texttt{521}, \texttt{main})\ \mathsf{MakeResp}(\texttt{t}, \texttt{521})\ \mathsf{Anon}(\texttt{521})$$

In this case, an engineer is hired in the same branch of technician 123*, and made responsible for task* t*. These two actions do not suffice to achieve G, since the planning agent knows that the two employees work in the same branch. This knowledge is somehow "retracted" by anonymizing the hired engineer: after the execution of* Anon(521)*, the planning agent still knows that* 521 *must work in some branch (this is enforced by a dedicated TBox axiom), but does not know which one, thus satisfying also the (epistemic) negative part of the goal.* ∎

5.2 Plan Templates and Online Instantiation

FINDPLAN-EKABSB returns plans with ground actions mentioning only objects in $\widehat{\mathcal{C}}$, as those in $\mathcal{C} \setminus \widehat{\mathcal{C}}$ are not used in $\widehat{\mathcal{K}}$. Such plans can be regarded as templates from which we can obtain regular plans for $\mathcal{K}$. This is a consequence of genericity, which yields that a plan keeps achieving a goal even under consistent renaming of the objects it mentions.

To formalize this intuition, we recast the notion of equality commitment [Calvanese *et al.*, 2013b] in this setting. Let B be a set of objects, and I a set of external input parameters. An *equality commitment (EC) H* over a *finite* set $S \subseteq B \cup I$ is a partition $\{H_1, \ldots, H_n\}$ of S s.t. each H_j contains at most *one* object from B. A substitution $\theta : I \to B$ is *compatible* with H if:

- for every pair of parameters $i_1, i_2 \in I$, we have that $i_1\theta = i_2\theta$ iff i_1 and i_2 belong to the same H_j;
- whenever H_j contains an object $d \in B$, then $i\theta = d$ for every $i \in H_j$.

Intuitively, θ is compatible with H if it maps parameters from the same class into the same object, and parameters from different classes into distinct objects. Finally, given a finite set $B' \subset B$ and a substitution $\theta : I \to B$, fix an ordering $O = \langle d_1, \ldots, d_n \rangle$ over the set $B_{cur} = B' \cup$ IM(θ) of the objects that are either mentioned in B' or assigned to I by θ. The EC H *induced by θ over B'* (under O) is the partition $H = \{H_1, \ldots, H_n\}$, s.t., for $j \in \{1, \ldots, n\}$:

Algorithm 3 Online instantiation of a plan template.

procedure ONLINEEXEC($\mathcal{K}, \pi$)
input: An eKAB $\mathcal{K} = \langle \mathcal{C}, \mathcal{C}_0, T, A_0, \Lambda, \Gamma \rangle$, and a plan $\pi = \alpha_1\theta_1 \cdots \alpha_m\theta_m$
  $A_o := A_0$ ▷ old, effective state
  $A_n := A_0$ ▷ current, effective state
  $h : \mathcal{C}_0 \to \mathcal{C}_0$ s.t. $h(d) = d$ for each $d \in \mathcal{C}_0$ ▷ cur. bijection
  for $k \in \{1, \ldots, m\}$ **do**
    $H :=$ eq. commitment induced by θ_i over ADOM(A)
    pick θ'_k that is compatible with $h(H)$ ▷ Agent choice
    $A'_o := \text{DO}(\alpha_i\theta_k, A_o, T)$
    $A'_n := \text{DO}(\alpha_i\theta'_k, A_n, T)$
    $h_n : \mathcal{C}_0 \cup \text{ADOM}(A_o) \to \mathcal{C}_0 \cup \text{ADOM}(A_n)$ s.t.
$$\begin{cases} h_n(d) = d, & \text{for } d \in \mathcal{C}_0 \\ h_n(d) = h(d), & \text{for } d \in \text{ADOM}(A'_o) \cap \text{ADOM}(A_o) \\ h_n(\theta_k(i)) = \theta'_k(i), & \text{for } i \text{ parameter of } \alpha_k \\ & \text{s.t. } \theta_k(i) \notin \text{ADOM}(A_o) \end{cases}$$
    $h := h_n$, $A_o := A'_o$, $A_n := A'_n$

- H_j contains d_j iff $d_j \in B'$, i.e., objects mentioned by θ but not present in B' are discarded;
- H_j contains $i \in \text{DOM}(\theta)$ iff $\theta(i) = d_j$, i.e., all parameters mapped to the same, j-th object are included in the j-th equivalence class.

Example 4. *Let $B = \{d_1 \ldots, d_5\}$, $B' = \{d_1, d_2, d_3\}$, $I = \{i_1, \ldots, i_4\}$, and $\theta : I \to B$, s.t.: $\theta(i_1) = d_1, \theta(i_2) = \theta(i_3) = d_5, \theta(i_4) = d_4$. The EC induced by θ over $B \cup \{d_4, d_5\}$ is $H = \{H_1, \ldots, H_5\}$, where: $H_1 = \{d_1, i_1\}, H_2 = \{d_2\}, H_3 = \{d_3\}, H_4 = \{i_4\}, H_5 = \{i_2, i_3\}$. Notice that $H_i \subseteq B$.* ∎

We can now state the following key result.

Lemma 5.3. *Let $\mathcal{K} = \langle \mathcal{C}, \mathcal{C}_0, T, A_0, \Lambda, \Gamma \rangle$ be an eKAB, and A_1, A'_1 ABoxes over T, s.t. $A_1 \cong^h_T A'_1$ for some object renaming h. Let $\alpha(\vec{p}, \vec{i})$ be an action in Λ with external input parameters $\vec{i}$, and θ a $\mathcal{K}$-legal substitution in A_1 for α. Let H be the equality commitment induced by θ over ADOM(A_1), and $H' = h(H)$ the equality commitment obtained from H by renaming each $d \in \text{ADOM}(A_1)$ as $h(d) \in \text{ADOM}(A_2)$. If θ' is a parameter substitution for $\vec{p}$ and $\vec{i}$ compatible with H', then:*

- *θ' is a $\mathcal{K}$-legal parameter for α in A'_1;*
- *$\text{DO}(\alpha\theta, A_1, T) \cong^{h'}_T \text{DO}(\alpha\theta', A'_1, T)$, where h' extends h as follows: for every parameter i of α s.t. $\theta(i) \notin \text{ADOM}(A_1)$, we have $h'(\theta(i)) = \theta'(i)$.*

Intuitively, Lemma 5.3 states that, modulo object renaming consistent with a parameter substitution that induces the same equality commitment, the same action can be applied to two logically equivalent ABoxes. Furthermore, such action induces the same update, modulo renaming of the objects mentioned in the two ABoxes and the involved parameters. In Algorithm 3, we exploit this result to build, in an online fashion, a plan for $\mathcal{K}$ starting from one for $\widehat{\mathcal{K}}$. This provides the freedom of dynamically choosing which actual objects to use when actions are executed, provided that the choice induces the same equality commitment induced by the parameter substitution in the original plan. By Lemma 5.3, we obtain:

Theorem 5.4. *Let $\mathcal{E} = \langle \mathcal{K}, G \rangle$ be an eKAB planning problem. If π is a plan that achieves G, then* ONLINEEXEC$(\mathcal{K}, \pi)$ *is guaranteed to achieve G for each possible choice.*

Example 5. *Consider plan π_2 of Example 3. This plan can be lifted online by the planning agent as follows: when hiring the engineer, the planning agent can freely inject a* fresh *employee identifier in place of* `521`*, provided that the chosen identifier is then consistently used in the subsequent actions. In other words, π_2 acts as a footprint for the infinite family of plans of the form*

$$\mathsf{HireEng}(Id, \mathtt{main})\ \mathsf{MakeResp}(\mathtt{t}, Id)\ \mathsf{Anon}(Id)$$

where Id is selected on-the-fly when the planning agent executes HireEng*, and is s.t. it is different from all the objects present in A_0 (this reconstructs the same equality commitment as in the case of* `521`*). All such infinite plans are guaranteed to achieve G.* ∎

6 Plan Synthesis for Lightweight eKABs

We consider plan synthesis for state-bounded eKABs over the lightweight DL *DL-Lite*$_\mathcal{A}$ [Calvanese *et al.*, 2009], and devise a technique based on compilation into ADL planning [Pednault, 1994; Drescher and Thielscher, 2008].

6.1 ADL Planning

An *ADL planning problem* is a tuple $\langle \mathcal{C}, \mathcal{C}_0, \mathcal{F}, \mathcal{A}, \varphi, \psi \rangle$, where:

- $\mathcal{C}$ is a *finite object domain*;
- $\mathcal{C}_0 \subseteq \mathcal{C}$ is the set of *initial objects*;
- $\mathcal{F}$ is a finite set of *fluents*, i.e., predicates whose extension can vary over time;
- $\mathcal{A}$ is a finite set of *ADL operators*;
- φ is the *initial state*, i.e., a conjunction of ground literals using predicates in $\mathcal{F}$ and objects in $\mathcal{C}_0$; and
- ψ is the *goal description*, i.e., a closed FO formula using predicates in $\mathcal{F}$ and objects in $\mathcal{C}_0$.

Each ADL operator in $\mathcal{A}$ is a tuple $\langle \text{N}, \vec{x}, \rho(\vec{x}), \varepsilon(\vec{x}) \rangle$, where:

- N is the *name*;
- variables $\vec{x}$ are the *parameters*;
- $\rho(\vec{x})$ is the *precondition*, i.e., a FO formula over $\mathcal{F}$, and whose terms are quantified variables, objects in $\mathcal{C}_0$, and parameters $\vec{x}$;
- $\varepsilon(\vec{x})$ is the *effect*, i.e., the universal closure of a FO conjunction built from admissible components, inductively defined as follows:
  - Fluent literals over $\mathcal{F}$ and whose terms are variables, objects in $\mathcal{C}_0$, or parameters in $\vec{x}$, are admissible.
  - If $\phi_1(\vec{y_1})$ and $\phi_2(\vec{y_2})$ are admissible, then $\phi_1(\vec{y_1}) \wedge \phi_2(\vec{y_2})$ and $\forall \vec{y_1}.\phi_1(\vec{y_1})$ are also admissible.
  - Let $\phi_1(\vec{y_1})$ be a FO formula over $\mathcal{F}$, whose terms are quantified variables, objects in $\mathcal{C}_0$, variables $\vec{y_1}$, or parameters $\vec{x}$. If $\phi_2(\vec{y_2})$ is admissible and does not contain any occurrence of $\to$ nor $\forall$, then $\phi_1(\vec{y_1}) \to \phi_2(\vec{y_2})$ is also admissible. This is used to tackle so-called *ADL conditional effects*.

6.2 Translation Procedure

We now define a syntactic, modular translation procedure LEKAB2ADL that takes as input a *DL-Lite*$_\mathcal{A}$-eKAB planning problem $\mathcal{E} = \langle \mathcal{K}, G \rangle$ with $\mathcal{K} = \langle \mathcal{C}, \mathcal{C}_0, \mathcal{T}, A_0, \Lambda, \Gamma \rangle$, and produces a corresponding ADL planning problem $\mathcal{P}_\mathcal{K} = \langle \mathcal{C}_\mathcal{K}, \mathcal{C}_0, \mathcal{F}_\mathcal{K}, \mathcal{A}_\mathcal{K}, \varphi_\mathcal{K}, \psi_G \rangle$.

Object domain. As in Section 5, if $\mathcal{C}$ is finite, so is $\mathcal{C}_\mathcal{K}$. If instead $\mathcal{C}$ is infinite but $\mathcal{K}$ is b-bounded, we fix $\mathcal{C}_\mathcal{K}$ to contain $\mathcal{C}_0$ plus $n + b$ objects from $\mathcal{C}$.

Fluents. $\mathcal{F}_\mathcal{K}$ is obtained by encoding concept and role names in $\mathcal{T}$ into corresponding unary and binary fluents. It also contains two special nullary fluents: $ChkCons$, distinguishing *normal execution modality* from *check consistency modality* of $\mathcal{P}_\mathcal{K}$, and $Error$, marking when the consistency check fails.

Operators. $\mathcal{A}_\mathcal{K}$ is obtained by transforming every action in Λ, with its condition-action rule in Γ, into an ADL operator: each action's effect produces a conditional effect in the ADL operator, and the condition-action rule its precondition.

Since in ADL, FO formulae are directly *evaluated* over FO structures (without ontological reasoning), we have to suitably consider the contribution of $\mathcal{T}$. Indeed (cf. Section 3), $\mathcal{T}$ is used both during query answering and to check whether the ABox resulting from an action instance is $\mathcal{T}$-consistent. We tackle both problems by relying on *DL-Lite*$_\mathcal{A}$'s *FO rewritability* of both ECQs and $\mathcal{T}$-consistency checks (cf. Section 2). To embed such checks into ADL, we force $\mathcal{A}_\mathcal{K}$ to alternate between two phases: the *normal execution modality*, where a "normal" ADL operator is applied, mirroring the execution of an action instance of $\mathcal{K}$; and the *check consistency modality*, where a special ADL operator checks if the obtained state is $\mathcal{T}$-consistent. Alternation is realized by toggling the fluent $ChkCons$, and activating $Error$ when the consistency check fails, thus blocking operator application.

Technically, consider an action $\alpha = a(\vec{p}) : \{e_1, \ldots, e_n\}$ in Λ and its corresponding condition-action rule $Q_\alpha(\vec{x}) \mapsto a(\vec{p})$ in Γ. Let $\vec{z} = \vec{p} \setminus \vec{x}$. We produce a corresponding ADL operator $\langle a, \vec{p}, \rho_\alpha(\vec{p}), \varepsilon_\alpha(\vec{p}) \rangle$. Its precondition $\rho_\alpha(\vec{p})$ corresponds to the FO formula $rew(Q_\alpha(\vec{x}), \mathcal{T}) \wedge \neg ChkCons \wedge \neg Error$, which leaves the external input parameters $\vec{z}$ unconstrained. The operator effect $\varepsilon_\alpha(\vec{p})$ is the FO conjunction of the translation of $e_1, \ldots, e_n$. Each $e_i = Q_i(\vec{p}, \vec{x}_i) \rightsquigarrow$ **add** F_i^+, **del** F_i^- generates the following conjunct:

$$rew(Q_i(\vec{p}, \vec{x}_i), \mathcal{T}) \rightarrow \bigwedge_{P_j(\vec{p}, \vec{x}_i) \in F_i^+} P_j(\vec{p}, \vec{x}_i) \wedge \bigwedge_{P_k(\vec{p}, \vec{x}_i) \in F_i^-} \neg P_k(\vec{p}, \vec{x}_i) \wedge ChkCons$$

Finally, the special ADL operator used to check $\mathcal{T}$-consistency is $\langle check, \emptyset, \rho_{chk}, \varepsilon_{chk} \rangle$, where $\rho_{chk} = ChkCons$ just checks whether the check flag is on, while ε_{chk} takes care of toggling the check flag, as well as of triggering the error flag if an inconsistency is detected:

$$\varepsilon_{chk} = \neg ChkCons \wedge (Q^\mathcal{T}_{\text{unsat}} \rightarrow Error)$$

Initial state specification. $\varphi_\mathcal{K}$ is by constructing the conjunction of all facts contained in A_0: $\varphi_\mathcal{K} = \bigwedge_{P_i(\vec{o}) \in A_0} P_i(\vec{o})$.

Goal description. The goal description ψ_G is obtained from goal G in $\mathcal{E}$ as $\psi_G = rew(G, \mathcal{T}) \wedge \neg ChkCons \wedge \neg Error$.

The two-fold contribution of $\mathcal{T}$ is taken care of, as in the operators, by rewriting G (via $rew(G, \mathcal{T})$) and ensuring that the ending state is $\mathcal{T}$-consistent (by requiring the absence of the consistency check and error flags in ψ_G).

We close by considering the following algorithm, called FINDPLAN-LEKABADL:

1. take as input a *DL-Lite*$_\mathcal{A}$-eKAB planning problem $\mathcal{E}$;
2. translate $\mathcal{E}$ into an ADL planning problem using LEKAB2ADL;
3. invoke an off-the-shelf ADL planner;
4. if the planner returns fail, return fail as well;
5. if the planner returns a plan π, filter away all $check$ operators from π, and return it as a result.

Theorem 6.1. FINDPLAN-LEKABADL *is correct.*

7 Conclusion

We have studied plan synthesis over eKABs, rich dynamic systems operating over a full-fledged DL KB. We have shown that checking plan existence is undecidable even under severe restrictions on the eKAB of interest. Nonetheless, we have demonstrated that, under the state-boundedness assumption for eKABs, plan existence is PSPACE in data complexity. In this setting, we have presented sound, complete and terminating algorithms for plan synthesis. We have then studied plan synthesis in the case where the eKAB is equipped with a lightweight DL of the *DL-Lite* family. In particular, we have proposed a technique to compile plan synthesis for state-bounded, lightweight eKABs into standard ADL planning.

To test the feasibility of our proposal for knowledge-intensive planning, we ran a preliminary empirical evaluation of the framework of Section 6. We considered a *DL-Lite*$_\mathcal{A}$-eKAB over the domain of Example 1, translated it into ADL, and fed it as input via PDDL to an off-the-shelf planner (we used FastDownward[3]). We varied the difficulty by increasing the bound on the object domain, thus affecting the number of ground atoms for the planner. With 9 domain elements, resulting in ∼2 000 atoms, a plan was found in under 1s. For 30 elements (∼300 000 atoms) a plan was found in 120s, while for 35 elements (∼1 200 000 atoms) the planner timed out. Although preliminary, we consider these results positive, given the complexity of the setting and the absence of optimizations, which we left for future work. In this respect, a promising line of research is to combine our approach with that of [Fan *et al.*, 2012]. There, the authors study a knowledge-intensive variant of Golog operating over an infinite object domain, and where incomplete knowledge is captured using *proper KBs* instead of DLs. In particular, they propose a semi-decidable technique to handle progression and query evaluation. In spite of the key differences between that approach and ours, their approach is reminiscent to the abstraction technique we adopt towards decidability. We intend to leverage this similarity with the aim of understanding if, and how, the optimization strategies proposed in [Fan *et al.*, 2012] can be lifted to our setting.

[3] http://www.fast-downward.org/

References

[Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[Bagheri Hariri *et al.*, 2013a] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. Verification of relational data-centric dynamic systems with external services. In *Proc. of PODS 2013*, pages 163–174, 2013.

[Bagheri Hariri *et al.*, 2013b] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. Description logic Knowledge and Action Bases. *J. of Artificial Intelligence Research*, 46:651–686, 2013.

[Bagheri Hariri *et al.*, 2014] Babak Bagheri Hariri, Diego Calvanese, Alin Deutsch, and Marco Montali. State-boundedness in data-aware dynamic systems. In *Proc. of KR 2014*. AAAI Press, 2014.

[Belardinelli *et al.*, 2014] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. Verification of agent-based artifact systems. *J. of Artificial Intelligence Research*, 51:333–376, 2014.

[Brachman and Levesque, 2003] Ron J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann, 2003.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

[Calvanese *et al.*, 2007a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of IJCAI 2007*, pages 274–279, 2007.

[Calvanese *et al.*, 2007b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[Calvanese *et al.*, 2009] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. Ontologies and databases: The *DL-Lite* approach. In Sergio Tessaris and Enrico Franconi, editors, *RW 2009 Tutorial Lectures*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009.

[Calvanese *et al.*, 2013a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. *Artificial Intelligence*, 195:335–360, 2013.

[Calvanese *et al.*, 2013b] Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. Verification and synthesis in description logic based dynamic systems. In *Proc. of RR 2013*, volume 7994 of *LNCS*, pages 50–64. Springer, 2013.

[Cimatti *et al.*, 2008] Alessandro Cimatti, Marco Pistore, and Paolo Traverso. Automated planning. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 841–867. Elsevier, 2008.

[De Giacomo *et al.*, 2014] Giuseppe De Giacomo, Yves Lespérance, Fabio Patrizi, and Stavros Vassos. Progression and verification of situation calculus agents with bounded beliefs. In *Proc. ofAAMAS'14*, 2014.

[Drescher and Thielscher, 2008] Conrad Drescher and Michael Thielscher. A fluent calculus semantics for ADL with plan constraints. In *Proc. of JELIA 2008*, volume 5293 of *LNCS*, pages 140–152. Springer, 2008.

[Erol *et al.*, 1995] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1–2):75–88, 1995.

[Fan *et al.*, 2012] Yi Fan, Minghui Cai, Naiqi Li, and Yongmei Liu. A first-order interpreter for knowledge-based golog with sensing based on exact progression and limited reasoning. In *Proc. of AAAI 2012*. AAAI Press, 2012.

[Gabbay *et al.*, 2003] Dov Gabbay, Agnes Kurusz, Frank Wolter, and Michael Zakharyaschev. *Many-dimensional Modal Logics: Theory and Applications*. Elsevier, 2003.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning – Theory and Practice*. Elsevier, 2004.

[Glimm *et al.*, 2008] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *J. of Artificial Intelligence Research*, 31:151–198, 2008.

[Hoffmann *et al.*, 2009] Jörg Hoffmann, Piergiorgio Bertoli, Malte Helmert, and Marco Pistore. Message-based web service composition, integrity constraints, and planning under uncertainty: A new connection. *J. of Artificial Intelligence Research*, 35:49–117, 2009.

[Levesque and Lakemeyer, 2001] Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.

[Levesque, 1984] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23:155–212, 1984.

[Ortiz *et al.*, 2008] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. of Automated Reasoning*, 41(1):61–98, 2008.

[Pednault, 1994] Edwin P. D. Pednault. ADL and the state-transition model of action. *J. of Logic and Computation*, 4(5):467–512, 1994.

[Wolter and Zakharyaschev, 1999] Frank Wolter and Michael Zakharyaschev. Temporalizing description logic. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems*, pages 379–402. Studies Press/Wiley, 1999.

[Zarrieß and Claßen, 2015] Benjamin Zarrieß and Jens Claßen. Verification of knowledge-based programs over description logic actions. In *Proc. of IJCAI'15*, 2015.