Action Theories over Generalized Databases with Equality Constraints

ID: 1109

Abstract

Situation calculus action theories allow full first-order expressiveness but, typically, restricted cases are studied such that projection or progression become decidable or first-order, respectively, and computationally feasible. In this work we focus on KBs that are specified as generalized databases with equality constraints, thus able to finitely represent complete information over possibly infinite number of objects. First, we show that this form characterizes the class of definitional KBs and provide a transformation for putting KBs in this form that we call *generalized fluent DB*. Then we show that for action theories over such KBs, the KBs are closed under progression, and discuss how this view exposes some differences with existing progression methods compared to DB update. We also look into the temporal projection problem and show how queries over these theories can be decided based on an induced transition system and evaluation of local conditions over states. In particular, we look into a wide class of generalized projection queries that include quantification over situations and prove that it is decidable under a practical restriction. The proposed action theories are to date the most expressive ones for which there are known decidable methods for computing both progression and generalized projection.

Introduction

The situation calculus is a logical language designed for reasoning about actions and change (McCarthy and Hayes 1969). A basic action theory (BAT) (Reiter 2001) is a well-studied type of theory in this language that essentially consists of a first-order knowledge base (KB) that describes the initial state of a given domain, and a set of first-order axioms that specify how the properties of the domain change under the effects of named actions.

There are two important reasoning problems that are studied in the context of variants of the BATs, namely temporal *projection* and temporal *progression*. The problem of projection is about *predicting* whether a condition would hold or not in the resulting KB if a series of actions were to be performed in the initial KB. Projection is formed as an entailment question that works on the logical description of the KB and the axioms that specify how the KB may evolve, and in fact does not rely on or require altering the initial description of the KB. On the other hand, the problem of progression is that of *updating* the KB when some named action is executed by replacing the initial description with one that reflects the changes of the action.

If we think of a BAT as a database which also features some specified operations (or actions) that alter the data, solving the projection problem corresponds to answering a query over the state of the database after some of these operations are consecutively performed, while the progression problem is to provide a concrete representation of the resulting database state. It becomes then clear that these two problems are closely related. In particular, progression can be used as a way to solve the projection problem in the following way: first update the database according to the operations in question and then answer the query.

Nonetheless, this view is only helpful when the KB is a database. Solving projection and progression becomes very tricky in the general case when instead of a regular database we have an unrestricted first-order knowledge base, and unrestricted first-order specifications for the effects of actions. In fact, as far as progression is concerned, for the general case it has been shown that second-order logic may be required to capture the updated KB (Lin and Reiter 1997; Vassos and Levesque 2013). This is due to a "hidden" second-order axiom that is included in all studied action theories that inductively characterizes the set of legal situations.

Similarly, projection is essentially an entailment question over a (limited) second-order theory. Fortunately though, by means of *regression* (Pirri and Reiter 1999) the projection query can be reduced to one that specifies the weakest precondition with respect to the initial situation, thus reducing projection to a first-order theorem proving task over the description of the initial KB. The regression technique works for any query that refers to a specified sequence of named actions and is otherwise unrestricted. The decidability (and feasibility) of projection then relies on the form of the initial KB and the type of projection queries in question.

However, very few cases have been studied in the literature such that projection is decidable and practical, namely *(i)* the case when the KB is a regular database as we discussed above (Reiter 1992), *(ii)* the case when the KB is an open-world database of a particular form, in which case a sound and sometimes complete method for projection is specified (Liu and Levesque 2005), *(iii)* the case of a modified version of the situation calculus language built using a two-variable fragment of first-order logic (Gu and Soutchan-

ski 2007), in which case projection is decidable, and, more recently, *(iv)* the case of bounded action theories that require that in all models and in every situation there is a fixed upperbound on the number of positive atomic facts (De Giacomo, Lespérance, and Patrizi 2012), in which case a generalized version of projection is proven to be decidable.

Notably, the case when the KB has the form of a *generalized database with constraints* has not been investigated (Kanellakis, Kuper, and Revesz 1995). This form of database allows to specify relations with possibly *infinitely many tuples*, using constraints, and captures complete knowledge via the closed-world assumption. A number of different types of constraints have been studied, e.g., equality constraints, showing tractability results for query answering over such databases for some of the cases.

Similarly, some results identify cases where a first-order progression can be effectively computed (Vassos and Patrizi 2013), but typically do not investigate what this implies for projection. For example, in the notable case of theories with *local-effect* actions (Vassos, Lakemeyer, and Levesque 2008; Liu and Lakemeyer 2009) it is guaranteed that finitely many ground atoms may be affected by an action, which can be effectively identified, and one can progress the KB by forgetting the truth value for these atoms and using the effect axioms to set their new value. Nonetheless, as the initial KB is otherwise unrestricted, this result alone does not provide a way to solve the projection problem. In fact, it could very well be the case that solving an entailment question over the initial KB is undecidable but progression over a local-effect action is still computable.

In this work we aim at providing a special type of action theories that admit decidable methods for computing *both* progression and projection. We appeal to the use of a form of generalized database with equality constraints as the initial KB and show how existing intuitions from database theory provide solutions for solving projection and progression.

We then investigate richer forms of projection that may refer to more than one possible evolution of the initial KB, e.g., capturing invariants of the form "after execution of α condition ϕ always holds" and specify a condition that ensures we can answer such queries by means of an appropriate transition system that abstracts the infinite tree of situations. This is similar in spirit to the work in (De Giacomo, Lespérance, and Patrizi 2012) but extended to account for fluents with possibly infinite extensions.

To the best of our knowledge these action theories are to date the most expressive ones with an infinite domain and possibly infinite extensions for fluents for which there are known decidable methods for computing both progression and generalized projection.

Situation calculus basic action theories (BATs)

The situation calculus as presented by Reiter (2001) is a three-sorted first-order language $\mathcal{L}$ with equality (and some limited second-order features). The sorts are used to distinguish between actions, situations, and objects.

A *situation* represents a world history as a sequence of actions. The constant S_0 is used to denote the initial situation where no actions have occurred. Sequences of actions are built using the function symbol *do*, such that $do(a, s)$ denotes the successor situation resulting from performing action a in situation s. Actions need not be executable in all situations, and the predicate $Poss(a, s)$ states that action a is executable in situation s. We will typically use a to denote a variable of sort action and α to denote a term of sort action, and similarly s and σ for situations.

A *relational fluent* is a predicate whose last argument is a situation, and thus whose value can change from situation to situation. We do not consider *functional fluents* as well as non-fluent predicates and functions, but we note that they can be represented as relational fluents with some extra axioms. We also assume a finite number of fluent and action symbols, $\mathcal{F}$ and $\mathcal{A}$, and an infinite number of constants $\mathcal{C}$.

Often we need to restrict our attention to sentences in $\mathcal{L}$ that refer to a particular situation. For example, the initial knowledge base (KB) is a finite set of sentences in $\mathcal{L}$ that do not mention any situation terms except for S_0. For this purpose, for any situation term σ, we define $\mathcal{L}_\sigma$ to be the subset of $\mathcal{L}$ that does not mention any other situation terms except for σ, does not mention *Poss*, and where σ is not used by any quantifier (Lin and Reiter 1997). When a formula $\phi(\sigma)$ is in $\mathcal{L}_\sigma$ we say that it is *uniform in* σ (Reiter 2001).

Also, we will use $\mathcal{L}^2$ to denote the second-order extension of $\mathcal{L}$ that allows predicate variables that take arguments of sort situation. $\mathcal{L}^2_\sigma$ then denotes the second-order extension of $\mathcal{L}_\sigma$ by predicate variables with arguments of sort situation.

We will be dealing with a specific kind of $\mathcal{L}$-theory, the so-called *basic action theory* $\mathcal{D}$ which has the following form:[1]

$$\mathcal{D} = \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_0 \cup \Sigma, \text{ where:}$$

1. $\mathcal{D}_{ap}$ is a set of action precondition axioms (APs), one for each action function symbol $A_i \in \mathcal{A}$, of the form $Poss(A_i(\vec{x}), s) \equiv \Pi_i(\vec{x}, s)$, where $\Pi_i(\vec{x}, s)$ is in $\mathcal{L}_s$.
2. $\mathcal{D}_{ss}$ is a set of successor state axioms (SSAs), one per fluent symbol $F_i \in \mathcal{F}$, of the form $F_i(\vec{x}, do(a, s)) \equiv \Phi_i(\vec{x}, a, s)$, with $\Phi_i(\vec{x}, a, s) \in \mathcal{L}_s$. SSAs characterize the conditions under which F_i has a specific value at situation $do(a, s)$ as a function of situation s and action a.
3. $\mathcal{D}_{una}$ is the set of unique-names axioms for actions: $A_i(\vec{x}) \neq A_j(\vec{y})$, and $A_i(\vec{x}) = A_i(\vec{y}) \supset \vec{x} = \vec{y}$, for each pair of distinct action symbols A_i and A_j in $\mathcal{A}$.
4. $\mathcal{D}_0$ is uniform in S_0 and describes the initial situation.
5. Σ is a set of domain independent foundational axioms which formally define legal situations and an ordering by means of symbol $\sqsubseteq$, also using a second-order inductive axiom in $\mathcal{L}^2$.

We will typically restrict our attention to *standard interpretations*, where equality is identity, and there is a bijection between the set of constants and the domain of discourse. This restriction can be captured by a set of axioms $\mathcal{E}$ consisting of the axioms of equality and the set of sentences $\{c_i \neq c_j | c_i, c_j \in \mathcal{C}, i \neq j\}$ (Levesque 1998).

Generalized databases and query evaluation

Kanellakis *et al* (1995) study the generalization of relational databases using constraints of various forms that concisely

[1] For readability we often omit the leading universal quantifiers.

represent a possibly infinite set of tuples. The so-called *generalized databases* are first-order interpretations (finitely) represented as constraints on the tuples that each relation contains. Such databases are obtained by including in each relation the (possibly infinite) set of tuples that satisfy the corresponding constraints. Various classes of constraints are considered, such as equalities and real polynomial inequalities. Here we focus on equality constraints.

Let us recall some basic definitions from (Kanellakis, Kuper, and Revesz 1995) and present them in the context of the situation calculus language $\mathcal{L}$ we specified, restricted only to the sub-language of the constants $\mathcal{C}$, equality, variables of sort object, and the typical logical connectives.

Definition 1. An *equality constraint* is any literal formula of the form $x\theta y$ or $x\theta c$, where $c \in \mathcal{C}$ and θ is $=$ or $\neq$. A *generalized k-tuple* over variables $x_1, \ldots, x_k$ is a finite conjunction $\psi = \varphi_1 \wedge \cdots \wedge \varphi_\ell$ of equality constraints whose all variables are free and among $x_1, \ldots, x_k$. A *generalized relation of arity k* is a finite set $R = \{\psi_1, \ldots, \psi_q\}$, of generalized k-tuples over $x_1, \ldots, x_k$. The *formula corresponding to a generalized relation R* is then simply the disjunction $\psi_1 \vee \cdots \vee \psi_q$. We will use ϕ_R to denote the quantifier-free formula corresponding to relation R.

Generalized relations represent possibly infinite relations over the domain of sort objects of $\mathcal{L}$. In detail, let $R = \{\psi_1, \ldots, \psi_q\}$ be a generalized relation of arity k, and ϕ_R the corresponding formula to this relation. Then, R is associated with the k-ary relation $\{\vec{c} \mid \vec{c} \in \mathcal{C}^k,\ \mathcal{E} \models \phi_R(\vec{c})\}$.

Intuitively, the way generalized relations specify standard relations is by constraints of the form: "R contains only tuples whose components satisfy either equality constraint 1 or ... or equality constraint q". It is easy to see that since equality constraints can in fact assign values to all variables in a tuple, any finite relation can be represented as a generalized relation by simply listing all of its finitely many tuples. On the other hand, it can also be seen that infinite relations exist that are not captured by generalized relations.

The notion of generalized relation extends naturally to databases: a *generalized database* is a finite set of generalized relations (Kanellakis, Kuper, and Revesz 1995). Differently from standard settings in databases, however, as we noted above, generalized databases represent in general infinite relations. As a consequence, answers to queries are in general infinite and they cannot be represented by means of finite relations as in regular databases. Nonetheless, it turns out that query answers over generalized databases can be represented as generalized relations with constraints, thus providing a closed representation system.

The first trick needed to show this is to observe that we can characterize the answer to a query by replacing the occurrences of relation atoms in the query by the formulas corresponding to the relations of the generalized database. Let $\varphi(\vec{x})$ be a first-order query over the relation symbols $R_1, \ldots, R_n$ and D a generalized database over the same relations. Let $\varphi[R_1/\phi_{R_1}, \ldots, R_n/\phi_{R_n}](\vec{x})$ be the first-order formula in $\mathcal{L}$ that is the result of replacing every occurrence of R_i in φ by ϕ_{R_i}. This formula, denoted here as $\varphi'(\vec{x})$, is then a finite representation of the answer to query φ over D.

The second trick needed is to observe that $\varphi'(\vec{x})$ can be represented as a a finite set of generalized tuples that characterize the isomorphism types of regular tuples in the answer of the query. Kanellakis *et al.* (1995) specify a procedure that first builds all the (finitely many) generalized tuples ψ over $\vec{x}$ using only the constants mentioned in $\varphi'(\vec{x})$, and then checks which of these are consistent with $\varphi'(\vec{x})$. The set of the ones that are consistent is a generalized relation that (finitely) represents the answer to $\varphi(\vec{x})$ over D.

Kanellakis *et al.* (1995) also show that following this procedure the answer to a first-order query over a generalized database (with equality constraints) is computable in LOGSPACE data complexity. Thus, generalized databases constitute a notable case of infinite databases for which an effective procedure exists to answer queries. In the following sections we exploit this to address two fundamental problems of action theories in the situation calculus, namely progression and projection, in the case that the knowledge base (KB) is a generalized database with equality constraints.

BATs with generalized fluent databases

Reiter (2001) investigates the case that the initial knowledge base (KB) is a *definitional theory* with respect to the fluent atoms in S_0, in the sense that for each fluent there is a definitional axiom that characterizes the truth clue for all atoms in S_0 as follows: $\bigwedge_{F_i \in \mathcal{F}} \forall \vec{x}_i.F(\vec{x}_i, S_0) \equiv \phi_i(\vec{x}_i)$, where $\phi_i(\vec{x}_i)$ is an unrestricted first-order formula that mentions no situations. We will call $\phi_i(\vec{x}_i)$ the *definition* for F_i.

In the case studied in (Reiter 2001) the underlying language $\mathcal{L}$ also includes non-fluent predicates that can be mentioned in the definitions for fluents. A KB then that includes definitional axioms also for all non-fluent predicates is called a *closed initial database*, while one that includes such axioms only for fluents is called *relatively complete* (Lin and Reiter 1997). A closed initial KB always represents complete information about fluents, while relatively complete ones may also capture incomplete information by means of non-fluent predicates mentioned in the definitions for fluents and additional axioms about theses predicates in the KB, e.g., $\forall \vec{x}.F(\vec{x}, S_0) \equiv P(\vec{x})$, $P(c) \vee P(d)$.

As it is typical in the literature, we assume that our language $\mathcal{L}$ does not include non-fluent predicates. Any "static" relation can represented using a fluent with a successor state axiom that copies the initial value to all situations, so that any complications that arise from dealing with incomplete information for non-fluent predicates is treated uniformly in terms of fluents. Under this assumption, closed initial databases and relatively complete KBs collapse to the same form. We will refer to these KBs as *definitional KBs*.

Definitional KBs in $\mathcal{L}$ capture *complete* information for fluents under the assumption of the unique-name axioms for constants and axioms for equality in $\mathcal{E}$. For example the following axiom states that there are exactly two atoms true for $In(x_1, x_2, S_0)$, namely $In(box, it_1, S_0)$ and $In(box, it_2, S_0)$:

$$\forall x \forall y (In(x, y, S_0) \equiv (x{=}box \wedge (y{=}it_1 \vee y{=}it_2))).$$

Nonetheless, the definition for a fluent can be any unrestricted first-order formula built over the constants in $\mathcal{C}$ and

equality, for example it could have the following form that implies an infinity of ground atoms that are true in S_0:

$$\forall x \forall y (\mathit{In}(x, y, S_0) \equiv (x \neq \mathit{box} \wedge (y = \mathit{it}_1 \vee y = \mathit{it}_2))).$$

Note that this definition can be rewritten as a formula that corresponds to a generalized relation, by distributing over the disjunction. More complicated definitions can be formed also using quantification, for example:

$$\forall x \forall y (\mathit{In}(x, y, S_0) \equiv (\exists z (z \neq \mathit{box}) \wedge (y = \mathit{it}_1 \vee y = \mathit{it}_2))).$$

Here though, as for the general case as well, the quantification over objects in the definition is not very helpful as it evaluates to true due to the axioms in $\mathcal{E}$, reducing to a formula that can be rewritten again to a generalized relation.

As it turns out, any definitional KB can be rewritten as a generalized database with equality constraints. This rewriting is possible because first-order theories of equality admit quantifier elimination. Next, we present a recursive procedure that allows us to transform a generic first-order formula ϕ in $\mathcal{L}$ that is situation-free, into a logically equivalent one (under the assumption of $\mathcal{E}$) that is quantifier-free. This will allow us to transform any definitional KB into one specified in terms of generalized relations, and appeal to the evaluation methods for generalized databases for handling projection and progression.

The transformation is inspired by that proposed by Libkin to prove the "natural-active" collapse of first-order formulas over relational databases (Grädel et al. 2005). We follow the same ideas but adapt and extend the approach accordingly so that it works also when a database may have relations with infinite elements.

Definition 2. Let ϕ be a first-order formula in $\mathcal{L}$ that is situation-free, and let C be the finite set of constants occurring in it. Without loss of generality assume that no variable occurs quantified in the scope of different quantifiers (if needed, the variable can be renamed). For convenience we use the standard symbols $\top$ and $\bot$, always interpreted as true and false, respectively. The transformation $T[\phi]$ is as follows ($[v/v']$ stands for the syntactic replacement of the symbol v by v'):

- $T[\phi] = \phi$, for the case that ϕ is $x = y$, $x = c$ or $x = x$;
- $T[\neg\psi] = \neg T[\psi]$;
- $T[\phi_1 \vee \phi_2] = T[\phi_1] \vee T[\phi_2]$;
- $T[\exists y \psi] = \bigvee_{c \in C} T[\psi][y/c] \vee \bigvee_{x_i \in \vec{x}} T[\psi][y/\vec{x}_i] \vee T_y[T[\psi]]$;

where T_y is as follows:

- $T_y[x = y] = T_y[y = c] = T_y[\bot] = \bot$;
- $T_y[y = y] = T_y[\top] = \top$;
- $T_y[\psi] = \psi$, for ψ any other atomic formula;
- $T_y[\neg\psi] = \neg T_y[\psi]$;
- $T_y[\psi_1 \vee \psi_2] = T_y[\psi_1] \vee T_y[\psi_2]$.

The interesting case is the last bullet of the definition of T. Intuitively, $T[\phi]$ is a rewriting of ϕ where the values that y can assume through the existential quantification are grouped into: those occurring as constants in ϕ (first disjunct); those assigned to the components of $\vec{x}$ (second disjunct); and those not falling into any of these classes (third disjunct). Since the number of values in the former two groups is finite, the quantification in these cases can be replaced by a finite disjunction. As to the latter disjunct, $T_y[\psi]$ is the result of replacing the atomic sub-formulas of $T[\psi]$ where y occurs, with the result of their evaluation (using $\top$ for true and $\bot$ for false), obtained by *assuming y different from all the constants and the other free variables of ψ*. Since $T[\psi]$ is quantifier-free, under this assumption and the axioms in $\mathcal{E}$, any atomic sub-formula containing y, which is of the form $t = t'$, with t and t' either variables (one of which is y) or constant symbols, can be replaced by $\top$ or $\bot$, without affecting the semantic of $T[\psi]$. Notice that T_y removes all the occurrences of y from $T[\psi]$.

The following theorem makes precise in what way the transformed formula is equivalent to the original one.

Theorem 1. *Let ϕ be a first-order formula in $\mathcal{L}$ that is situation-free. For every $\vec{c} \in \vec{C}$ the following holds:*

$$\mathcal{E} \models \phi(\vec{c}) \text{ iff } \mathcal{E} \models T[\phi](\vec{c}).$$

The proof is done by induction on the structure of ϕ following the intuition described above. As a corollary to Theorem 1 we have the following.

Corollary 2. *Let $\phi(\vec{x})$ be a first-order situation-free formula in $\mathcal{L}$. There exists a quantifier-free first-order formula $\phi'(\vec{x})$ such that for every $\vec{c} \in \vec{C}$, $\mathcal{E} \models \phi(\vec{c})$ iff $\mathcal{E} \models \phi'(\vec{c})$.*

We identify basic action theories over generalized databases as follows.

Definition 3. A set $\mathcal{D}_0$ of first-order sentences uniform in S_0 is a *generalized fluent database (GFDB)* iff it has the form $\bigwedge_{F_i \in \mathcal{F}} \forall \vec{x}_i . F(\vec{x}_i, S_0) \equiv \phi_i(\vec{x}_i)$, where $\phi_i(\vec{x}_i)$ is a formula that corresponds to a generalized relation over $\vec{x}$, i.e., is a disjunction of conjunctions of equality constraints. A basic action theory $\mathcal{D}$ is a *basic action theory over a generalized fluent database (BAT-GFDB)* iff it also includes the set of axioms $\mathcal{E}$ and $\mathcal{D}_0$ is a generalized fluent database.

Using the previous result we can transform any definition formula in a definitional KB to a quantifier-free formula and then to a DNF, thus putting the KB into the form of a generalized fluent database.

Corollary 3. *Let ϕ be a definitional KB. There exists a generalized fluent database ϕ' such that $\mathcal{E} \models \phi \equiv \phi'$.*

As discussed in the previous section for such KBs there is also a decidable procedure for evaluating queries. Note also that equivalence of generalized fluent databases can also be decided, formed as an appropriate query. With these tools available we now proceed to show how solutions for progression and projection can be obtained for BAT-GFDBs.

Progression of BAT-GFDBs

The progression of a basic action theory is the problem of *updating* the initial KB so that it reflects the current state of the world after some actions have been performed, instead of the initial state of the world. In other words, in order to

do a one-step progression of the BAT $\mathcal{D}$ with respect to the ground action α we need to replace $\mathcal{D}_0$ in $\mathcal{D}$ by a suitable set $\mathcal{D}_\alpha$ of sentences uniform in $do(\alpha, S_0)$ so that the original theory $\mathcal{D}$ and the theory $(\mathcal{D} - \mathcal{D}_0) \cup \mathcal{D}_\alpha$ are equivalent with respect to how they describe the situation $do(\alpha, S_0)$ and the situations in the future of $do(\alpha, S_0)$.

In a seminal paper Lin and Reiter (1997) gave a model-theoretic definition for the progression $\mathcal{D}_\alpha$ of $\mathcal{D}_0$ wrt α and $\mathcal{D}$ that achieves this goal. Finding such a $\mathcal{D}_\alpha$ is a difficult task and it has been shown that second-order logic may be required in the general case (Lin and Reiter 1997; Vassos and Levesque 2013). Nonetheless there are many special cases where a first-order progression can be effectively computed. In fact, for the definitional KBs, and as a result also for the special case of generalized fluent databases, there is a very simple way to progress.

Theorem 4. *(Lin and Reiter 1997) Let $\mathcal{D}_0$ be $\bigwedge_{F_i \in \mathcal{F}} \forall \vec{x_i}.F(\vec{x_i}, S_0) \equiv \phi_i(\vec{x_i})$, and for all $F_i \in \mathcal{F}$, let $\mathcal{D}_{ss}$ include an SSA of the form $F_i(\vec{x_i}, do(a, s)) \equiv \Phi_i(\vec{x_i}, \alpha, S_0)$. For each $F_i \in \mathcal{F}$, let $\Phi'_i(\vec{x_i}, \alpha, S_0)$ be the sentence obtained by replacing every occurrence of fluent atoms $F_j(\vec{o}, S_0)$ in $\Phi_i(\vec{x_i}, \alpha, S_0)$ by $\phi_j(\vec{o})$. Then, $\mathcal{D}_\alpha : \bigwedge_{F_i \in \mathcal{F}} \forall \vec{x_i}.F_i(\vec{x_i}, do(a, s)) \equiv \Phi'_i(\vec{x_i}, \alpha, S_0)$ is a progression of $\mathcal{D}_0$ with respect to α and the theory $\mathcal{D}$.*

Observe that this is very similar to the first trick we discussed when we reviewed the work on generalized databases and query evaluation (Kanellakis, Kuper, and Revesz 1995), where we replaced the occurrences of relation atoms in the query by the formulas corresponding to the relations of the generalized database. It is interesting to look into how this method works when $\mathcal{D}_0$ is a generalized fluent database, that will illustrate how the second trick can also be of use.

Note that since each $\Phi_i(\vec{x_i}, \alpha, S_0)$ in the SSAs is in general unrestricted, e.g., may include quantifiers, $\mathcal{D}_\alpha$ is not guaranteed to be in the form of a generalized fluent database even though $\mathcal{D}_0$ is. The point in using a form like the generalized fluent database is that it allows us to perform query evaluation using the methods and existing technologies in constraint databases instead of performing more general theorem proving. Therefore, we want progression to preserve the form of $\mathcal{D}_0$. The method of Theorem 4 does well in preserving the form of a *definitional KB* but does not preserve the form in the case of a generalized fluent database.

This is how the second trick becomes useful. The idea is to consider generalized tuples as the "base" formulas that we use to express any generalized fluent relation. This is similar to a regular database where we would update $\mathcal{D}_0$ into a $\mathcal{D}_\alpha$ such that for every fluent a finite list of tuples is specified. Corollary 3 then provides a way to transform the resulting $\mathcal{D}_\alpha$ of Theorem 4 into the form of a generalized fluent database by means of the procedure of Definition 2.

Theorem 5. *Let $\mathcal{D}$ be a BAT over a generalized fluent database and α a ground action. Then there exists a first-order progression $\mathcal{D}_\alpha$ of $\mathcal{D}_0$ wrt α and $\mathcal{D}$ that is in the form of a generalized fluent database.*

As a consequence, we can *iteratively* progress a BAT-GFDB and express the state corresponding to any ground situation as a generalized fluent database.

Finally, since every definitional KB can be expressed as a generalized fluent database, this analysis also illustrates a subtle detail about the way we understand progression. Both a progression $\mathcal{D}_\alpha$ according to Theorem 4 and a progression $\mathcal{D}'_\alpha$ according to Theorem 5 qualify as logically correct progressions of $\mathcal{D}_0$ and are logically equivalent (under the assumption of $\mathcal{E}$). Nonetheless, $\mathcal{D}_0$ is more of a *logical specification of the changes* that need to be made due to action α and $\mathcal{D}'_\alpha$ more of a *materialized update* of these changes.

Another way to look at it is that the progression procedure of Theorem 4 is purely syntactic (linear to the size of $\mathcal{D}_0$) and does not involve any form of evaluation; in a sense, the fluents are not updated to a new truth value but, rather, the new truth values are still specified with respect to the initial situation. As there are to date no implemented systems for performing progression, this difference is less easy to identify because as far as the logical specification is concerned $\mathcal{D}_0$ is a well-behaved progression. In practice though, we would expect that unless a method that goes along the lines of Theorem 5 is followed, the progressed theory would in fact look much similar to the original theory $\mathcal{D}$, and queries over the current state would perform similar to using a *regression* technique over the original theory $\mathcal{D}$.

Projection over BAT-GFDBs

The *(simple) projection problem* is the task of *predicting* whether a condition holds at a particular time in the future after a series of ground actions have been executed (Reiter 2001). The following is a straightforward result.

Theorem 6. *Let $\mathcal{D}$ be a BAT-GFDB, $\alpha_1, \ldots, \alpha_n$ a sequence of ground actions, and $\phi(s)$ a first-order formula uniform in s. Then determine whether or not the following holds is decidable: $\mathcal{D} \models \phi(do(\alpha_n, \cdots do(\alpha_1, S_0)))$.*

This is not a new result and can be proven by means of regression and the fact that $\mathcal{E}$ is decidable. Our previous analysis also shows that simple projection queries over a BAT-GFDB can be decided also by means of iteratively progressing $\mathcal{D}_0$ wrt $\alpha_1, \ldots, \alpha_n$ according to Theorem 5 and then evaluating the query over the resulting generalized fluent database following the method described in (Kanellakis, Kuper, and Revesz 1995). Depending on the type of queries, and the frequency that actions occur, either approach may be preferred under conditions.

We now proceed to show a major result about the decidability of richer projection queries over BAT-GFDBs that may also quantify over future situations.

Generalized projection

A *generalized* version of the projection problem is when ϕ is may refer to any number or combination of future situations. For example, $\mathcal{D} \models \forall s(do(\alpha, S_0) \sqsubseteq s \supset \psi(s))$ states that after executing action α condition $\psi(s)$ will hold in all situations s in the future, essentially saying that it will always remain true. We will look into a type of such queries that is similar to $\mathcal{L}^F_\sigma$ defined in (Vassos and Levesque 2013).

We consider the language $\mathcal{L}_p$ of *generalized projection queries* φ. $\mathcal{L}_p$ is defined on top of the language $\mathcal{L}_n$, whose formulas ϕ are as follows: $\phi := x = c \mid x = y \mid F(\vec{x}, s) \mid$

$F(\vec{x}, \sigma) \mid \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi$, for F a fluent symbol, c a constant, and σ a ground situation term. The formulas φ of $\mathcal{L}_p$ are defined as: $\varphi := \phi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists s.\sigma \sqsubseteq s \wedge \varphi$, where $\phi \in \mathcal{L}_n$ is any formula uniform in s or in a ground situation term σ, and whose free variables (if any) are only of sort situation.

We will also consider a non-trivial class of BAT-GFDBs, defined below.

Definition 4. Let $\mathcal{D}$ be a BAT-GFDB and B a natural number. A ground situation term σ is said to be *constant-bounded by B in $\mathcal{D}$*, C-bounded by B for short, iff for every fluent $F_i \in \mathcal{F}$ the definition of F_i in $\mathcal{D}_0$ mentions at most B distinct constant symbols. $\mathcal{D}$ is said to be *constant-bounded by a finite bound B*, C-bounded by B for short, iff every ground situation term of $\mathcal{D}$ that is executable is also C-bounded by B.

Intuitively, this means that any state associated with some executable situation of $\mathcal{D}$ can be fully represented by a generalized database, mentioning only a bounded number of constants (possibly different in each state).

For this class of action theories, we prove that entailment of generalized projection queries is decidable.

Theorem 7. *Given a BAT-GFDB $\mathcal{D}$ that is C-bounded by some B and a generalized projection query φ that is a sentence in $\mathcal{L}_p$, it is decidable to check whether $\mathcal{D} \models \varphi$.*

The rest of this section is dedicated to prove this result. We start by introducing auxiliary notions and results.

Definition 5. A *(labelled) transition system over generalized fluent databases* (for an action theory $\mathcal{D}$), *GFDB-TS* (for $\mathcal{D}$) for short, is a tuple $T = (Q, q_0, \rightarrow, L)$, where:

- Q is the GFDB-TS's (nonempty) set of nodes[2];
- $q_0 \in Q$ is the GFDB-TS's initial node;
- $\rightarrow \subseteq Q \times Act \times Q$ is the GFDB-TS's transition relation, for Act the set of all ground action terms of $\mathcal{D}$; we interchange the notations $(q, \alpha, q') \in \rightarrow$ and $q \xrightarrow{\alpha} q'$;
- L is the GFDB-TS's labelling function, associating each node q with a generalized fluent database $L(q)$.

We associate each ground situation term $\sigma = do([\alpha_1, \ldots, \alpha_n], S_0)$ with the node q_σ s.t. $q_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{n-1}} q_\sigma$, if it exists.

Notice that in Definition 5, the $L(q)$ labeling of a generic node q is a GFDB and so is uniform in S_0 as required by the corresponding definition. While this is necessary for technical reasons, such GFDBs are not to be intended as defining or referring to the "original" initial situation. Instead, they should be intuitively understood as defining the state of the situation obtained by executing, from S_0, the ground actions labeling a path from q_0 to q, while also moving the "S_0 point of reference" to be the current situation.

Besides the standard semantics of $\mathcal{L}_p$ over action theories, we define a semantics over GFDB-TSs that will be useful later to prove our results.

[2]We use *node* instead of *state* to avoid confusion with the states associated with situations in action theories.

Definition 6. Given a GFDB-TS T, an $\mathcal{L}_p$ formula φ, and a node q of T, we define when T *satisfies φ at node q*, written $T, q \models \varphi$, as follows:

- for $\varphi = \phi \in \mathcal{L}_n$, $T, q \models \varphi$, iff
  - ϕ is uniform in s, i.e., of the form $\phi(s)$, and $\mathcal{E}, L(q) \models \phi(S_0)$, i.e., treated as a local query over node q; or
  - ϕ is uniform in σ, q_σ exists, and $T, q_\sigma \models \phi[\sigma/s]$, i.e., reduced to the previous case as a a unique base case;
- the semantics of the connectives $\neg$, $\wedge$ is as standard;
- $T, q \models \exists s.\sigma \sqsubseteq s \wedge \varphi$, for $\sigma = do([\alpha_1, \ldots, \alpha_n], S_0)$, if for some $\sigma' = do([\alpha_1, \ldots, \alpha_n, \ldots, \alpha_m], S_0)$ s.t. $m \geq n$, it is the case that $q_{\sigma'}$ is defined and $T, q_{\sigma'} \models \varphi[s/\sigma']$;

When φ is a sentence, T is said to *satisfy* φ, written $T \models \varphi$ iff $T, q_0 \models \varphi$.

Notice again the use of S_0 as a placeholder for any situation associated with q.

Every BAT-GFDB $\mathcal{D}$ implicitly induces an infinite GFDB-TS that is essentially the situation tree annotated with actions as arc labels, and generalized fluent databases as descriptions of the states associated with situations (by Theorem 5 every state of a BAT-GFDB can be faithfully represented in this way).

Definition 7. The *induced GFDB-TS* of a BAT-GFDB $\mathcal{D}$ is the GFDB-TS $T_\mathcal{D} = (Q, q_0, \rightarrow, L)$ (over $\mathcal{D}$), s.t.:

- Q is the set of all $\mathcal{D}$'s ground situation terms;
- $q_0 = S_0$;
- $q \xrightarrow{A(\vec{c})} q'$ iff $q' = do(A(\vec{c}), q)$;
- $L(q)$ is a generalized database such that:
  - if $q = q_0$ then $L(q) = \mathcal{D}_0$;
  - if $q \neq q_0$ and there exists q' s.t. $q' \xrightarrow{A(\vec{c})} q$, then $L(q')$ is the progression of $L(q)$ w.r.t. $A(\vec{c})$ and $\mathcal{D}$, where the situation term $do(A(\vec{c}), q)$ is replaced by S_0.

The first result we have shows that, as far as generalized projection queries are concerned, the induced GFDB-TS can be used as an alternative representation of $\mathcal{D}$.

Lemma 1. *Let $\mathcal{D}$ be a BAT-GFDB and $T_\mathcal{D}$ the corresponding induced GFDB-TS. Then, for any generalized projection query φ that is a sentence in $\mathcal{L}_p$, we have that*

$$\mathcal{D} \models \varphi \textit{ iff } T_\mathcal{D} \models \varphi.$$

Proof By induction on the structure of φ. Notice that the semantics of Definition 6 considers all the (infinitely many) ground situation terms of $\mathcal{D}$, i.e., all nodes of $T_\mathcal{D}$. □

Thus, one can check whether $\mathcal{D} \models \varphi$ using $T_\mathcal{D}$. Obviously, this does not imply decidability straightforwardly. Indeed, as both the situation terms and the state space of $T_\mathcal{D}$ are in general infinite, one cannot, e.g., simply follow a recursive procedure based on the inductive definition of satisfaction (such a procedure would not terminate).

We next show how to circumvent this probllem when dealing with a BAT-GFDB that is C-bounded by a bound B. To this end, we show how to construct, starting from $\mathcal{D}$ and φ, a finite GFDB-TS $\hat{T}_{\mathcal{D},\varphi}$ that is indistinguishable from $T_\mathcal{D}$, by φ.

Fix a finite set $H \subseteq \mathcal{C}$ of constants, such that $\mathcal{C}_\mathcal{D} \cup \mathcal{C}_\varphi \subseteq H$ and $|H| \geq B \cdot |\mathcal{F}| + |\mathcal{C}_\mathcal{D} \cup \mathcal{C}_\varphi| + N_\mathcal{A}$, where: $\mathcal{C}_\mathcal{D}$ and $\mathcal{C}_\varphi$ are the set of constants respectively occurring in $\mathcal{D}$ and φ, and $N_\mathcal{A}$ is the largest number of parameters in the action types of $\mathcal{D}$.

The construction of $\hat{T}_{\mathcal{D},\varphi}$ is shown in Algorithm 1. We use $Progr(\mathcal{D}_0, A(\vec{c}))$, to refer to the result of progressing an initial theory $\mathcal{D}_0$ w.r.t. a ground action $A(\vec{c})$, which we assume to be a GFDB (see Theorem 5). The symbol $\equiv_\mathcal{E}$ represents logical equivalence between theories, under $\mathcal{E}$.

Algorithm 1 (Constructs $\hat{T}_{\mathcal{D},\varphi}$)

```
1: procedure BUILDT̂(D, φ)
2:   Q := {q0};
3:   → := ∅;
4:   L(q0) := D0;
5:   Front := {q0};
6:   while Front ≠ ∅ do
7:     for all q ∈ Front do
8:       Front := Front \ {q};
9:       for all A(h⃗) s.t. A ∈ A, h⃗ ∈ H⃗ and E, L(q) ⊨
     Poss(A(h⃗), S0) do
10:        P := Progr(L(q), A(h⃗))[do(S0, A(h⃗))/S0];
11:        if ¬∃q' ∈ Q s.t. P ≡E L(q') then
12:          Q := Q ∪ q', for q' a fresh node;
13:          L(q') := P;
14:          Front := Front ∪ {q'};
15:        else
16:          let q' ∈ Q be s.t. L(q') ≡E P;
17:        end if
18:        → := → ∪ (q, A(h⃗), q');
19:      end for
20:    end for
21:  end while
22:  return (Q, q0, →, L);
23: end procedure
```

The procedure inductively builds a GD-TS for the theory $\mathcal{D}$, by applying, at every step, all the executable actions obtained from the action types of $\mathcal{D}$ and the constants in H. Applying an action $A(\vec{h})$ consists in progressing (line 10) the labelling DB of the current node q (initially q_0) w.r.t. $A(\vec{h})$, provided it is executable according to the labeling $L(q)$ (line 9), and replacing, in the obtained progression, the situation term $do(A(\vec{h}), S_0)$ by S_0. If the obtained progression P is not logically equivalent (under $\mathcal{E}$) to any GFDB labeling some node of (the current) Q, then a fresh node q' is added to Q, with labeling $L(q) = P$ (lines 11–14); if instead some node q' exists with $L(q')$ logically equivalent to P, then q' is simply retrieved from Q (line 16), and no new node is added. In either case, a transition from q to q' under the executed action is added to $\rightarrow$ (line 14). Every time a fresh node is added to Q, it is stored in the set $Front$, which represents the nodes of Q that needs to be expanded. Initially, $Front$ contains only q_0. The algorithm returns when $Front$ is empty.

Lemma 2. *Algorithm 1 terminates on any C-bounded BAT-GFDB $\mathcal{D}$ and generalized projection query φ.*

Proof Direct consequence of the following facts: H is finite, thus so is the set of ground action terms; checking whether $\mathcal{E}, L(q) \models Poss(A(\vec{h}), S_0)$ is decidable as $L(q)$ is a GFDB, for which entailment is decidable; checking whether $P \equiv_\mathcal{E} L(q)$ is decidable, P and $L(q)$ being GFDBs; for a given (finite) set of fluents $\mathcal{F}$ and a finite set H of constants, there exist only finitely many equivalence classes of logically equivalent (under $\mathcal{E}$) GFDBs that can be defined using only constants from H. In particular, the last observation guarantees that only finitely many distinct nodes can be generated, and thus that $Front$ is eventually empty. □

The following result, together with Lemma 1, proves that one can use the finite-state $\hat{T}_{\mathcal{D}\varphi}$, instead of the infinite $T_\mathcal{D}$, to check whether $\mathcal{D} \models \varphi$.

Lemma 3. *For any BAT-GFDB $\mathcal{D}$ C-bounded by some bound B and generalized projection query φ that is a sentence in $\mathcal{L}_p$, we have that:*

$$T_\mathcal{D} \models \varphi \textit{ iff } \hat{T}_{\mathcal{D},\varphi} \models \varphi.$$

Proof We make use of the following definition. Given two GFDBs $\mathcal{D}_0$ and $\mathcal{D}'_0$, and a set $C \subseteq \mathcal{C}$ of constants, we say that $\mathcal{D}_0$ and $\mathcal{D}'_0$ are *C-indistinguishable*, written $\mathcal{D}_0 \approx_C \mathcal{D}'_0$, if there exists a bijection $\gamma : \mathcal{C}_\mathcal{D} \cup C \rightarrow \mathcal{C}_{\mathcal{D}'} \cup C$ that is the identity on C, s.t. for the theory $\mathcal{D}''_0$ obtained from $\mathcal{D}_0$ by renaming all of its constants c as $\gamma(c)$, it is the case that $\mathcal{D}''_0 \equiv_\mathcal{E} \mathcal{D}'_0$. Intuitively $\mathcal{D} \approx_C \mathcal{D}'$ means that $\mathcal{D}$ and $\mathcal{D}'$ are logically equivalent up to renaming of the constants not mentioned in C.

Let $T_\mathcal{D} = (Q, q_0, \rightarrow, L)$, $\hat{T}_{\mathcal{D},\varphi} = (\hat{Q}, \hat{q}_0, \hat{\rightarrow}, \hat{L})$, and $C_{\mathcal{D},\varphi} = \mathcal{C}_\mathcal{D} \cup \mathcal{C}_\varphi$. We next prove that: (*) for any $q \in Q$ and $\hat{q} \in \hat{Q}$ s.t. $L(q) \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q})$, $T_\mathcal{D}, q \models \varphi$ iff $\hat{T}_{\mathcal{D},\varphi}, \hat{q} \models \varphi$. Since $L(q_0) \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q}_0)$ (as, by Algorithm 1, $L(q_0) = \hat{L}(\hat{q}_0)$), this implies that $T_\mathcal{D}, q_0 \models \varphi$ iff $\hat{T}_{\mathcal{D},\varphi}, \hat{q}_0 \models \varphi$, i.e., $T_\mathcal{D} \models \varphi$ iff $\hat{T}_{\mathcal{D},\varphi} \models \varphi$.

The proof of (*) is by induction on the structure of φ. Consider first the case of $\varphi = \phi \in \mathcal{L}_n$. If ϕ is uniform in s, it is sufficient to prove that $\mathcal{E}, L(q) \models \phi$ iff $\mathcal{E}, \hat{L}(\hat{q}) \models \phi(S_0)$. This comes as a consequence of the fact that $L(q)$ and $\hat{L}(\hat{q})$ are logically equivalent up to renaming of constants *not mentioned* in ϕ, therefore the models of $L(q)$ and $\hat{L}(\hat{q})$ have exactly the same relational structure but some (finitely many) objects, not referred by any constant in ϕ, renamed. Thus, such models are not distinguishable by ϕ.

If ϕ is uniform in σ, we observe that all of the constants occurring in σ, i.e., in some of its action terms, are in $C_{\mathcal{D},\varphi}$, thus also in the set of constants H used in Algorithm 1. Therefore, all action terms occurring in σ are considered in the construction of $\hat{T}_{\mathcal{D},\varphi}$. Now, it can be proven that: (**) whenever $L(q) \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q})$, for any action term $A(\vec{h})$ s.t. $\vec{h} \in C_{\mathcal{D},\varphi}$, there exists q' s.t. $(q, A(\vec{h}), q') \in \rightarrow$ iff there exists $\hat{q}'$ s.t. $(\hat{q}, A(\vec{h}), \hat{q}') \in \hat{\rightarrow}$ and $L(q') \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q}')$. This is a consequence of the facts that: $L(q) \models Poss(A(\vec{h}))$ iff

$\hat{L}(\hat{q}) \models Poss(A(\vec{h}), S_0)$ (the proof is similar to the case of ϕ uniform in s), and that querying $L(q)$ and $\hat{L}(\hat{q})$, which are $C_{\mathcal{D},\varphi}$-indistingushable, with a formula whose constants are in $C_{\mathcal{D},\varphi}$ (as is the case of the right-hand side of SSAs in $\mathcal{D}$), produces $C_{\mathcal{D},\varphi}$-indistinguishable results. Therefore, by iterative applications of (**), starting from q_0 and $\hat{q}_0$ and using the actions in $\sigma = do([\alpha_1, \ldots, \alpha_n], S_0)$, we have that there exists $q_\sigma \in Q$ iff there exists $\hat{q}_\sigma \in \hat{Q}$ s.t. $\hat{L}(\hat{q}_\sigma) \approx_{C_{\mathcal{D},\varphi}} L(q_\sigma)$. Then, since $\phi[\sigma/s]$ is uniform in s, by the inductive hypothesis, we have that $T_\mathcal{D}, q_\sigma \models \phi[\sigma/s]$ iff $\hat{T}_{\mathcal{D},\varphi}, \hat{q}_\sigma \models \phi[\sigma/s]$.

The case of boolean connectives is straightforward.

For the case of $\varphi = \exists s.\sigma \sqsubseteq s \wedge \varphi'$, a generalization of (**) above can be proved, which states that: (***) whenever $L(q) \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q})$, for any action term $A(\vec{c})$ s.t. $\vec{c} \in \mathcal{C}$, there exists q' s.t. $(q, A(\vec{c}), q') \in \rightarrow$ iff there exists $\hat{q}'$ s.t. $(\hat{q}, A(\vec{h}), \hat{q}') \in \hat{\rightarrow}$ and $L(q') \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q}')$, for $\vec{h}$ obtained by an appropriate renaming of the constants in $\vec{c}$, according to a bijection consistent with some of the bijections witnessing $L(q) \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q})$, and that is the identity on $C_{\mathcal{D},\varphi}$. The cardinality constraint guarantees H to contain enough values to obtain $\vec{h}$ as above. In turn, (***) implies that, for $\sigma = do([\alpha_1, \ldots, \alpha_n], S_0)$, a node $q_{\sigma'} \in Q$ exists, with $\sigma' = do([\alpha_1, \ldots, \alpha_n, \ldots, \alpha_m], S_0)$ ($m \geq n$), iff there exists a node $\hat{q}_{\sigma''} \in \hat{Q}$, with $\sigma'' = do([\alpha_1, \ldots, \alpha_n, \ldots, \alpha''_m], S_0)$, s.t. $L(q_{\sigma'}) \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q}_{\sigma''})$. Moreover, by the induction hypothesis, we have that $T, q_{\sigma'} \models \varphi[s/\sigma']$ iff $\hat{T}_{\mathcal{D},\varphi}, \hat{q}_{\sigma''} \models \varphi[s/\sigma'']$, which concludes the proof. □

To complete the proof of Theorem 7, it remains to show that checking whether $\hat{T}_{\mathcal{D},\varphi} \models \varphi$ is decidable. Notice that this is not a trivial consequence of the node-finiteness of $\hat{T}_{\mathcal{D},\varphi}$, in that the semantics of generalized projection queries over GFDB-TS requires, in general, to consider all executable situation terms of $\mathcal{D}$, which can be infinitely many.

Lemma 4. *Given a C-bounded BAT-GFDB $\mathcal{D}$ and a generalized projection query φ that is a sentence in $\mathcal{L}_p$, checking whether $\hat{T}_{\mathcal{D},\varphi} \models \varphi$ is decidable.*

Proof To perform the check, we use the following recursive procedure (the case of boolean connectives $\neg$, $\wedge$ and $\vee$, omitted for brevity, is as standard):

1: **procedure** CHECK$\hat{T}(q, \varphi)$
2: **if** $\varphi = \phi \in \mathcal{L}_n$ and ϕ is uniform in s **then**
3: **return** $\hat{T}_{\mathcal{D},\varphi}, q \models \varphi$;
4: **end if**
5: **if** $\varphi = \phi \in \mathcal{L}_n$ and ϕ is uniform in σ **then**
6: **if** q_σ does not exist in Q **then**
7: **return** $false$;
8: **else**
9: **return** CHECK$\hat{T}(q_\sigma, \varphi[\sigma/s])$;
10: **end if**
11: **end if**
12: **if** $\varphi = \exists s.do([\alpha_1, \ldots, \alpha_n], S_0) \sqsubseteq s \wedge \phi$ **then**
13: **for all** paths $q_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} q_{n+1} \xrightarrow{\alpha_{n+1}} \cdots \xrightarrow{\alpha_{m-1}} q_m$ s.t. in the suffix $q_{n+1} \xrightarrow{\alpha_{n+1}} \cdots \xrightarrow{\alpha_{m-1}} q_m$, no node occurs more than once **do**
14: **if** CHECK$\hat{T}(q_m, \varphi[s/\sigma']) == true$, for $\sigma' = do([\alpha_1, \ldots, \alpha_{m-1}], S_0)$ **then**
15: **return** $true$;
16: **end if**
17: **end for**
18: **return** $false$;
19: **end if**
20: **end procedure**

Procedure Check$\hat{T}(q_0, \varphi)$ terminates, and returns $true$ iff $\hat{T}_{\mathcal{D},\varphi} \models \varphi$. Termination is a consequence of the recursive structure of the procedure, the fact that $\hat{T}_{\mathcal{D},\varphi}, q \models \varphi$ is decidable in the base case, i.e., $\varphi \in \mathcal{L}_n$ and uniform in s, as $L(q)$ is a generalized fluent database, and checking whether $L(q) \models \tilde{\varphi}(S_0)$ is decidable, and that the paths to consider are only finitely many. Soundness is straightforward, as the procedure essentially explores a finite subset of the situation terms (represented as paths) needed to check whether $\hat{T}_{\mathcal{D},\varphi} \models \varphi$, in the most general case (quantification over s).

As to completeness, the case of interest is when a quantified situation occurs (line 12). In this case, observe that, due to the finiteness of Q, the paths considered at line 13 visit all possible nodes q_m that can be reached by considering all ground situation terms σ' s.t. $\sigma \sqsubseteq \sigma'$. Moreover, it can be shown that for any two situation terms σ' and σ'' s.t. $q_{\sigma'} = q_{\sigma''}$, it is the case that $\hat{T}_{\mathcal{D},\varphi}, q_{\sigma'} \models \varphi[s/\sigma']$ iff $\hat{T}_{\mathcal{D},\varphi}, q_{\sigma'} \models \varphi[s/\sigma'']$. Therefore, by considering only the (finely many) paths of line 13, one covers indeed all the cases obtained by considering all possible ground situation terms that are successors of $\sigma = do([\alpha_1, \ldots, \alpha_n], S_0)$, as the semantics of Definition 6 requires. □

Lemmas 1, 2, 3 and 4 prove, together, Theorem 7.

Conclusions and future work

In this paper we looked into situation calculus action theories over generalized fluent databases with equality constraints, making a connection between work in the situation calculus and work in constraint query languages. In particular we contributed with the following:

- We showed that generalized fluent databases with equality constraints characterize the class of definitional KBs without non-fluent predicates (Reiter 2001), and introduced a transformation that can be used to bring any definitional KB into the form of a generalized fluent database.
- We showed that generalized fluent databases are well-behaved in the sense that their form is preserved under progression and that compared to the existing approach to progressing a definitional KB, the proposed progression method is closer to the notion of database update and can help bringing the computational benefits that are intended for such an operation.
- We showed that simple projection queries over BAT-GFDBs are decidable and can be answered by means of evaluation of generalized databases and an induced transition systems.

- We extended the notion of boundedness (De Giacomo, Lespérance, and Patrizi 2012) to represent infinitely many facts with exceptions and showed that generalized projection for a wide class of sentences is decidable.

To the best of our knowledge BAT-GFDBs with bounded unknowns are to date the most expressive situation calculus action theories with an infinite domain and possibly infinite extensions for fluents, for which computing progression and generalized projection is decidable.

For future work we want to consider other constraints, and look into controlled ways to express incomplete information similar to the extensions of proper KBs in (Liu, Lakemeyer, and Levesque 2004) and (De Giacomo, Lespérance, and Levesque 2011). We believe that the second approach, which looks into a form of named nulls, and, similarly, previous work on bounded unknowns (Vassos and Patrizi 2013), can be used to include a practical form of incomplete information in the generalized fluent databases and the action theories over them.

Finally, we intend to investigate how our results can be used to build a practical implementation for a variant of the Golog family of high-level agent programming languages (Levesque et al. 1997; De Giacomo et al. 2009).

References

De Giacomo, G.; Lespérance, Y.; Levesque, H. J.; and Sardina, S. 2009. *Multi-Agent Programming: Languages, Tools and Applications*. Springer. chapter IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents, 31–72.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2011. Efficient reasoning in proper knowledge bases with unknown individuals. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, 827–832. AAAI Press.

De Giacomo, G.; Lespérance, Y.; and Patrizi, F. 2012. Bounded situation calculus action theories and decidable verification. In Brewka, G.; Eiter, T.; and McIlraith, S. A., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference (KR)*.

Grädel, E.; Kolaitis, P. G.; Libkin, L.; Marx, M.; Spencer, J.; Vardi, M. Y.; Venema, Y.; and Weinstein, S. 2005. *Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Gu, Y., and Soutchanski, M. 2007. Decidable reasoning in a modified situation calculus. In *In: Proceedings of International Joint Conference on AI, (IJCAI 2007*, 1891–1897.

Kanellakis, P. C.; Kuper, G. M.; and Revesz, P. Z. 1995. Constraint query languages. *Journal of Computer and System Sciences* 51(1):26–52.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.

Levesque, H. J. 1998. A completeness result for reasoning with incomplete first-order knowledge bases. In *Proceedings of KR-1998, Sixth International Conference on Principles of Knowledge Representation and Reasoning*.

Lin, F., and Reiter, R. 1997. How to progress a database. *Artificial Intelligence* 92(1-2):131–167.

Liu, Y., and Lakemeyer, G. 2009. On first-order definability and computability of progression for local-effect actions and beyond. In *Proc. IJCAI-09*, 860–866.

Liu, Y., and Levesque, H. J. 2005. Tractable reasoning with incomplete First-Order knowledge in dynamic systems with Context-Dependent actions. In *Proceedings of the 19th international joint conference on Artificial intelligence*, IJCAI'05. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Liu, Y.; Lakemeyer, G.; and Levesque, H. J. 2004. A logic of limited belief for reasoning with disjunctive information. In *Proc. KR-04*, 587–597.

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence* 4:463–502.

Pirri, F., and Reiter, R. 1999. Some contributions to the metatheory of the situation calculus. *Journal of the ACM* 46(3):261–325.

Reiter, R. 1992. The projection problem in the situation calculus: A soundness and completeness result, with an application to database updates. In *In Proceedings First International Conference on AI Planning Systems*, 198–203.

Reiter, R. 2001. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Vassos, S., and Levesque, H. J. 2013. How to progress a database III. *Artificial Intelligence* 195:203–221.

Vassos, S., and Patrizi, F. 2013. A Classification of First-Order Progressable Action Theories in Situation Calculus. In Rossi, F., ed., *Proc. of IJCAI'13*.

Vassos, S.; Lakemeyer, G.; and Levesque, H. J. 2008. First-order strong progression for local-effect basic action theories. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR-08)*, 662–272.