Action Theories over Generalized Databases with Equality Constraints[*]

Fabio Patrizi and Stavros Vassos

Department of Computer, Control, and Management Engineering (DIAG)
Sapienza University of Rome
Rome, Italy
{patrizi,vassos}@dis.uniroma1.it

Abstract. In this work we focus on situation calculus action theories over *generalized databases with equality constraints*, here called GFDBs, which are able to finitely represent complete information over a possibly infinite number of objects. We contribute with the following: i) we show that GFDBs characterize the class of definitional KBs and that they are closed under progression; ii) we show that temporal projection queries are decidable for theories with an initial KB expressed as a GFDB, which we call GFDB-BATs; iii) we extend the notion of boundedness to allow for infinite objects in the extensions of fluents and prove that a wide class of generalized projection queries is decidable for GFDB-BAT under a restriction we call C-boundedness; iv) we show that checking whether C-boundedness holds for a given bound is decidable. The proposed action theories are to date the most expressive ones for which there are decidable methods for computing both progression and generalized projection.

Introduction

Situation calculus basic action theories (BATs) [13] are well-studied logical theories that consist of a first-order knowledge base (KB) which 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. Two important reasoning problems are studied in the context of variants of BATs: temporal *projection* and *progression*. Projection is about *predicting* whether a condition would hold in the resulting state if a series of actions were to be performed in the initial KB, while progression is about *updating* the KB by a new description that reflects the current state after actions have been performed.

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 then becomes clear

[*] The authors acknowledge the support of the EU Project FP7-ICT 318338 (OPTIQUE) and the Sapienza Award 2013 “Spiritlets” project.

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 we have an unrestricted first-order specifications for the KB and the effects of actions. 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 [9, 15]; a list of some special cases where it becomes first-order is studied in [16]. Similarly, a few cases have been studied such that projection is decidable, namely *(i)* the case when the KB is a regular database as we discussed above [14], *(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 [12], *(iii)* the case of a modified version of the situation calculus built using a two-variable fragment of first-order logic [6] in which case projection is decidable, and, more recently, *(iv)* the case of bounded theories that require that in all models and in every situation there is a fixed upperbound on the number of positive atomic facts [3].

Notably, the case when the KB has the form of a *generalized database with constraints* [7], which allows to specify relations with possibly *infinitely many tuples*, has not been investigated. In this work we show that for a special type of BATs whose KB is a generalized database with equality constraints projection is decidable and a first-order progression can always be computed. We then look into 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 also ensures decidability. To the best of our knowledge these BATs 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 a first-order progression and generalized projection.

Situation calculus basic action theories (BATs)

The situation calculus as presented by Reiter [13] 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. S_0 is used to denote the initial situation and 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 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. 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 [9]. When a formula $\phi(\sigma)$ is in $\mathcal{L}_\sigma$ we say that it is *uniform in* σ [13].

We will be dealing with a specific kind of $\mathcal{L}$-theory, the so-called *basic action theory (BAT)* $\mathcal{D}$ 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, one for each action function symbol $A_i \in \mathcal{A}$, of the form $Poss(A_i(\boldsymbol{x}), s) \equiv \Pi_i(\boldsymbol{x}, s)$, where $\Pi_i(\boldsymbol{x}, s)$ 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 $F_i(\boldsymbol{x}, do(a, s)) \equiv \Phi_i(\boldsymbol{x}, a, s)$, with $\Phi_i(\boldsymbol{x}, a, s) \in \mathcal{L}_s$. 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: $A_i(\boldsymbol{x}) \neq A_j(\boldsymbol{y})$, and $A_i(\boldsymbol{x}) = A_i(\boldsymbol{y}) \supset \boldsymbol{x} = \boldsymbol{y}$, for each pair of distinct 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 foundational axioms which formally define legal situations and an ordering by means of symbol $\sqsubseteq$, also using a second-order inductive axiom.

Finally, we will typically restrict our attention to the case that distinct constants are always interpreted into different objects. This unique-names restriction can be captured by a set of axioms $\mathcal{E}$ consisting of the axioms of equality and the set of sentences $\{c_i \neq c_j | c_i, c_j \in \mathcal{C}, i \neq j\}$ [8].

Generalized databases and query evaluation

A *generalized database* [7] is a first-order interpretation (finitely) represented as constraints on the tuples of relations. Generalized 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. In this work we focus on equality constraints. Let us present basic definitions from [7] in the context of the situation calculus language $\mathcal{L}$ we specified.

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

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

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 formula corresponding to this relation. Then, R is associated with the k-ary relation $\{\boldsymbol{c} \mid \boldsymbol{c} \in \mathcal{C}^k,\ \mathcal{E} \models \phi_R(\boldsymbol{c})\}$. It is easy to see that any finite relation can be represented as a generalized relation, while infinite relations exist that are not captured by generalized ones.

The notion of generalized relation extends naturally to databases: a *generalized database* is a finite set of generalized relations. Differently from standard settings in databases, since generalized databases represent in general infinite relations, answers to queries are in general infinite and cannot be represented by means of finite relations. Nonetheless, it turns out that query answers over generalized databases can be represented as generalized relations with constraints, thus providing a closed representation system.

First 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 $\varphi(\boldsymbol{x})$ be a first-order query over the relation symbols $R_1, \ldots, R_n$ and D a generalized database over the same relations. Let $\varphi[R_1/\phi_{R_1}, \ldots, R_n/\phi_{R_n}](\boldsymbol{x})$ 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 $\varphi'(\boldsymbol{x})$, is then a finite representation of the answer to query φ over D.

The second trick is to observe that $\varphi'(\boldsymbol{x})$ can be represented as a finite set of generalized tuples that characterize the isomorphism types of regular tuples in the answer of the query. Kanellakis *et al.* [7] specify a procedure that first builds all the (finitely many) generalized tuples ψ over $\boldsymbol{x}$ using only the constants mentioned in $\varphi'(\boldsymbol{x})$, and then checks which of these are consistent with $\varphi'(\boldsymbol{x})$. The set of the ones that are consistent is a generalized relation that (finitely) represents the answer to $\varphi(\boldsymbol{x})$ over D.

Kanellakis *et al.* [7] 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, this constitutes a notable case of infinite databases for which an effective procedure exists to answer queries.

BATs with generalized fluent databases (GFDBs)

Reiter [13] investigates the case where the initial knowledge base (KB) is a *definitional theory* with respect to the fluent atoms in S_0, i.e., with S_0 characterized as follows: $\bigwedge_{F_i \in \mathcal{F}} \forall \boldsymbol{x_i}. F_i(\boldsymbol{x}_i, S_0) \equiv \phi_i(\boldsymbol{x}_i)$, where $\phi_i(\boldsymbol{x}_i)$, called the *definition* for F_i, is an unrestricted first-order formula mentioning no situations. When the underlying language $\mathcal{L}$ includes only fluent predicates, as it is the case in this paper, a KB in such form is called a *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 $\mathit{In}(x_1, x_2, S_0)$, namely $\mathit{In}(\mathit{box}, \mathit{it}_1, S_0)$ and $\mathit{In}(\mathit{box}, \mathit{it}_2, S_0)$:

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

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:

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

Note that this definition can be rewritten as a formula that corresponds to a generalized relation by distributing over the disjunction. Also, more complicated definitions that include quantification do not actually add to the expressiveness, as first-order theories of equality admit quantifier elimination [5].

We identify BATs over generalized databases as follows.

Definition 2. *A set $\mathcal{D}_0$ of first-order sentences uniform in S_0 is a* generalized fluent database (GFDB) *iff it has the form $\bigwedge_{F_i \in \mathcal{F}} \forall \boldsymbol{x_i}. F(\boldsymbol{x}_i, S_0) \equiv \phi_i(\boldsymbol{x}_i)$, where $\phi_i(\boldsymbol{x}_i)$ is a formula that corresponds to a generalized relation over $\boldsymbol{x}$, 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.*

Theorem 1. *Let ϕ be a definitional KB. There exists a GFDB ϕ' such that $\mathcal{E} \models \phi \equiv \phi'$.*

As discussed in the previous section for such KBs there is also a decidable LOGSPACE data complexity procedure for evaluating queries. Note also that equivalence of GFDBs can also be decided, formed as an appropriate query. With these tools available in the next sections we will proceed to show how solutions for progression and projection can be obtained for BAT-GFDBs. We close this section with a simple example of a GFDB-BAT.

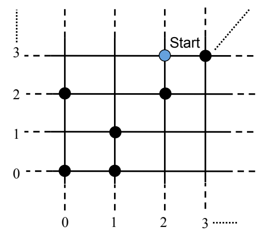

Fig. 1. Map of the *grid* domain.

Example 1. Figure 1 shows a space where an agent can move only along specified lines. The agent starts in $(2, 3)$ and can initially move only vertically, i.e., to any position s.t. $x = 2$. After moving, the agent can change direction (at the next move) only if it has stopped at a crossing point (marked with solid circles). For instance, if the agent moves from $(2, 3)$ to $(2, 1)$, it cannot move next to, e.g., $(0, 1)$. Instead, if the agent stops in $(2, 2)$, it can then move along the x-axis to, e.g., $(10, 2)$. The crossing points are placed along the diagonal, i.e., points s.t. $x = y$, and at $(0, 2)$ and $(1, 0)$. A BAT describing this domain is as follows:

- Action types: $\mathcal{A} = \{moveTo(x, y)\}$

– Fluents: $\mathcal{F} = \{At(x,y,s), Dest(x,y,s), Cross(x,y,s)\}$

– $\mathcal{D}_0$: $At(x,y,S_0) \equiv x = 2 \wedge y = 3, \quad Dest(x,y,S_0) \equiv x = 2$
$Cross(x,y,S_0) \equiv (x = y) \vee (x = 0 \wedge y = 2) \vee (x = 1 \wedge y = 0)$

– $\mathcal{D}_{ap}$: $Poss(moveTo(x,y),s) \equiv Dest(x,y,s)$

– $\mathcal{D}_{ss}$: $Cross(x,y,do(moveTo(x',y'),s)) \equiv Cross(x,y,s)$
$At(x,y,do(moveTo(x',y'),s)) \equiv (x = x' \wedge y = y')$

$$\begin{aligned}
&Dest(x,y,do(moveTo(x',y'),s)) \equiv (Cross(x',y',s) \wedge (x = x' \vee y = y')) \vee \\
&\quad \exists x'',y''.At(x'',y'',s) \wedge [(x' = x'' \wedge y' \neq y'' \wedge x = x') \vee \\
&\quad (y' = y'' \wedge x' \neq x'' \wedge y = y') \vee (y' = y'' \wedge x' = x'' \wedge Dest(x,y,s))]
\end{aligned}$$

Observe that the extension of fluent $Dest$ is initially infinite. Indeed, according to its definition, the fluent contains all possible tuples s.t. $x = 2$. Such tuples represent the infinitely many possible destinations available to the agent in S_0.

Progression of BAT-GFDBs

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 $do(\alpha, S_0)$.

Lin and Reiter [9] 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 [9, 15]. Nonetheless, 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 2 ([9]). *Let $\mathcal{D}_0$ be $\bigwedge_{F_i \in \mathcal{F}} \forall \boldsymbol{x_i}.F_i(\boldsymbol{x}_i, S_0) \equiv \phi_i(\boldsymbol{x}_i)$, and for all $F_i \in \mathcal{F}$, let $\mathcal{D}_{ss}$ include an SSA of the form $F_i(\boldsymbol{x_i}, do(a,s)) \equiv \Phi_i(\boldsymbol{x_i}, \alpha, S_0)$. For each $F_i \in \mathcal{F}$, let $\Phi'_i(\boldsymbol{x_i}, \alpha, S_0)$ be the sentence obtained by replacing every occurrence of atoms $F_j(\boldsymbol{o}, S_0)$ in $\Phi_i(\boldsymbol{x_i}, \alpha, S_0)$ by $\phi_j(\boldsymbol{o})$, and $\mathcal{D}_\alpha$ be $\bigwedge_{F_i \in \mathcal{F}} \forall \boldsymbol{x_i}.F_i(\boldsymbol{x_i}, do(a,s)) \equiv \Phi'_i(\boldsymbol{x_i}, \alpha, S_0)$. Then, $\mathcal{D}_\alpha$ is a progression of $\mathcal{D}_0$ wrt 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 [7], 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 $\Phi_i(\boldsymbol{x_i}, \alpha, S_0)$ 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 2 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. Theorem 1 then provides a way to transform, by means of quantifier elimination, the resulting $\mathcal{D}_\alpha$ of Theorem 2 into the form of a generalized fluent database.

Theorem 3. *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.

Example 2. The following theory $\mathcal{D}_\alpha$ is the progression of the initial GFDB $\mathcal{D}_0$ of Example 1, wrt action $\alpha = moveTo(2,2)$ (and theory $\mathcal{D}$):

$$At(x, y, do(\alpha, S_0)) \equiv x = 2 \wedge y = 2, \quad Dest(x, y, do(\alpha, S_0)) \equiv x = 2 \vee y = 2$$
$$Cross(x, y, do(\alpha, S_0)) \equiv (x = y) \vee (x = 0 \wedge y = 2) \vee (x = 1 \wedge y = 0)$$

Notice that, similarly to the initial situation, after executing α in S_0, the agent still has an infinite set of destinations available: all those s.t. $x = 2$ or $y = 2$.

Finally, since every definitional KB can be expressed as a GFDB, this analysis also illustrates a subtle detail about the way we understand progression. Both a progression $\mathcal{D}_\alpha$ according to Theorem 2 and a progression $\mathcal{D}'_\alpha$ according to Theorem 3 qualify as logically correct progressions of $\mathcal{D}_0$ and are logically equivalent (under the assumption of $\mathcal{E}$). Nonetheless, $\mathcal{D}_\alpha$ 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 into a practical *normal form*.

Another way to look at it is that the progression procedure of Theorem 2 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. Theorem proving is then needed in order to reason over the specification, even for a simple look-up query for a given atom. Even though this logical specification may in fact be beneficial in some cases, in practice we expect that materializing the update into a GFDB normal form (that explicitly lists the generalized tuples for each fluent) would offer similar advantages as updates do in regular databases.

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 [13]. The following is a straightforward result.

Theorem 4. *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 determining 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 by iteratively progressing $\mathcal{D}_0$ wrt $\alpha_1, \ldots, \alpha_n$ according to Th. 3 and then evaluating the query over the resulting GFDB following the method of [7]. 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. A *generalized* version of the projection problem is when ϕ may refer to any number or combination of future situations. For instance, referring to Ex. 1, the formula $\forall s. do(moveTo(2,2), S_0) \sqsubseteq s \supset \exists xy. Dest(x, y, s)$ states that after executing action $moveTo(2,2)$ in the initial situation, the agent has an available destination in any future situation.

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: $\phi := x = c \mid x = y \mid F(\boldsymbol{x}, s) \mid F(\boldsymbol{x}, \sigma) \mid \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi$, for F a fluent symbol, c a constant, and σ a ground situation term. $\mathcal{L}_p$ formulas 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, whose free variables (if any) are only of sort situation.

We also consider a class of BAT-GFDBs which we call C-bounded. To define it, let T_V be the (finite) set of all generalized tuples ψ that use equality constraints with (only variable) symbols from the finite set of variables V, and s.t. ψ does not contain multiple occurrences of some equality constraint. Notice that since generalized tuples are conjunctions (thus multiple occurrences of a conjunct do not change their semantics), T_V essentially contains all the possible generalized tuples one can build using symbols from V.

Definition 3. *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 $\mathcal{D}$ *(or simply C-bounded) iff for every fluent $F(\boldsymbol{x}, s) \in \mathcal{F}$, it is the case that $\mathcal{D} \models \bigvee_{\Psi \in 2^{T_V}} \exists \boldsymbol{y}. F(\boldsymbol{x}, \sigma) \equiv \bigvee_{\psi \in \Psi} \psi$, where: V is partitioned into X and Y, with X the set of variables occurring in $\boldsymbol{x}$, Y any set of variable symbols such that $|Y| = B$, and $\boldsymbol{y}$ are the free variables of ψ coming from Y. $\mathcal{D}$ 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.*

Notice that Ψ above is a set of generalized tuples, thus the formula $\bigvee_{\psi \in \Psi} \psi$ is a generalized relation. Intuitively, Definition 3 requires that the definition of each fluent in σ is a generalized relation mentioning at most B distinct constants. An example of C-bounded BAT-GFDB is provided by the action theory of Ex. 1.

For this class of theories, we have the following result.

Theorem 5. *Given a BAT-GFDB $\mathcal{D}$ that is C-bounded by some B, and a generalized projection query sentence φ in $\mathcal{L}_p$, it is decidable to check whether $\mathcal{D} \models \varphi$.*

The rest of this section details the proof of this theorem.

Definition 4. *A* (labelled) transition system over GFDBs *(for a GFDB-BAT $\mathcal{D}$),* GFDB-TS *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.

In Def. 4, the label $L(q)$ of a generic node q is a GFDB, thus uniform in S_0 as required by the corresponding definition. Such GFDBs 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 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 an alternative semantics over GFDB-TSs.

Definition 5. *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 $T, q_0 \models \varphi$.*

Every BAT-GFDB $\mathcal{D}$ induces an infinite GFDB-TS, as defined below.

Definition 6. *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;*
- *$q_0 = S_0$;*
- *$q \xrightarrow{A(\mathbf{c})} q'$ iff $q' = do(A(\mathbf{c}), q)$;*

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

– *$L(q)$ 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. $q' \xrightarrow{A(\boldsymbol{c})} q$, then $L(q')$ is the progression of $L(q)$ wrt $A(\boldsymbol{c})$ and $\mathcal{D}$, where $do(A(\boldsymbol{c}), q)$ is replaced by S_0.*

Our first result 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 $\mathcal{D} \models \varphi$ iff $T_\mathcal{D} \models \varphi$.*

Proof. By induction on the structure of φ.

Thus, one can check whether $\mathcal{D} \models \varphi$ using $T_\mathcal{D}$. Obviously, this does not imply decidability, as both the situation terms and the state space of $T_\mathcal{D}$ are in general infinite. We show next how to circumvent this problem when $\mathcal{D}$ is a BAT-GFDB C-bounded by a bound B.

Starting from $\mathcal{D}$ and φ, we construct a *finite* GFDB-TS $\hat{T}_{\mathcal{D},\varphi}$ that is indistinguishable from $T_\mathcal{D}$, by φ. To this end, fix a finite set of constants $H \subseteq \mathcal{C}$, s.t. $\mathcal{C}_\mathcal{D} \cup \mathcal{C}_\varphi \subseteq H$ 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, where $Progr(\mathcal{D}_0, A(\boldsymbol{c}))$ denotes the result of progressing an initial theory $\mathcal{D}_0$ w.r.t. a ground action $A(\boldsymbol{c})$, which we assume to be a GFDB (see Th. 3). The symbol $\equiv_\mathcal{E}$ represents logical equivalence between theories, under $\mathcal{E}$.

The procedure inductively builds a GD-TS for $\mathcal{D}$, 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 $A(\boldsymbol{h})$ consists in progressing (line 10) the labeling DB of the current node q (initially q_0) w.r.t. $A(\boldsymbol{h})$, provided it is executable according to the labeling $L(q)$ (line 9), then replacing, in the obtained progression, the situation term $do(A(\boldsymbol{h}), S_0)$ 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 11–15); if instead some node q' exists with $L(q')$ logically equivalent to P, then q' is simply retrieved from Q (line 17), and no new node is added. In either case, a transition from q to q' under the executed action is added to $\rightarrow$ (line 15). Every time a fresh node is added to Q, it is stored in the set *Front*, containing the nodes of Q to be expanded. Initially, $Front = \{q_0\}$. 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. Follows from the facts: H is finite; checking $\mathcal{E}, L(q) \models Poss(A(\boldsymbol{h}), S_0)$ is decidable as $L(q)$ is a GFDB; checking $P \equiv_\mathcal{E} L(q)$ 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.

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

```
 1: procedure BUILD$\hat{T}(\mathcal{D},\varphi)$
 2:     $Q := \{q_0\}$;
 3:     $\rightarrow := \emptyset$;
 4:     $L(q_0) := \mathcal{D}_0$;
 5:     $Front := \{q_0\}$;
 6:     while $Front \neq \emptyset$ do
 7:         for all $q \in Front$ do
 8:             $Front := Front \setminus \{q\}$;
 9:             for all $A(\boldsymbol{h})$ s.t. $A \in \mathcal{A}$, $\boldsymbol{h} \in \boldsymbol{H}$ and $\mathcal{E}, L(q) \models Poss(A(\boldsymbol{h}), S_0)$ do
10:                 $P := Progr(L(q), A(\boldsymbol{h}))[do(S_0, A(\boldsymbol{h}))/S_0]$;
11:                 if $\neg\exists q' \in Q$ s.t. $P \equiv_{\mathcal{E}} L(q')$ then
12:                     let $q'$ a fresh node;
13:                     $Q := Q \cup q'$;
14:                     $L(q') := P$;
15:                     $Front := Front \cup \{q'\}$;
16:                 else
17:                     let $q' \in Q$ be s.t. $L(q') \equiv_{\mathcal{E}} P$;
18:                 end if
19:                 $\rightarrow := \rightarrow \cup (q, A(\boldsymbol{h}), q')$;
20:             end for
21:         end for
22:     end while
23:     return $(Q, q_0, \rightarrow, L)$;
24: end procedure
```

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

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

Proof. (Sketch) Given two GFDBs $\mathcal{D}_0$ and $\mathcal{D}'_0$, and a set $C \subseteq \mathcal{C}$ of constants, write $\mathcal{D}_0 \approx_C \mathcal{D}'_0$, if there exists a bijection $\gamma : \mathcal{C}_{\mathcal{D}_0} \cup C \rightarrow \mathcal{C}_{\mathcal{D}'_0} \cup C$ 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 $\mathcal{D}''_0 \equiv_{\mathcal{E}} \mathcal{D}'_0$. (This intuitively means that $\mathcal{D}_0$ and $\mathcal{D}'_0$ are logically equivalent up to renaming of the constants not mentioned in C.) Then, let $T_{\mathcal{D}} = (Q, q_0, \rightarrow, L)$, $\hat{T}_{\mathcal{D},\varphi} = (\hat{Q}, \hat{q}_0, \hat{\rightarrow}, \hat{L})$, and $C_{\mathcal{D},\varphi} = \mathcal{C}_{\mathcal{D}} \cup \mathcal{C}_{\varphi}$. The proof is based on proving 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 $L(q_0) \approx_{C_{\mathcal{D},\varphi}} \hat{L}(\hat{q}_0)$ (see Algorithm 1), 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 (*), omitted for space reasons, is by induction on the structure of φ.

To complete the proof of Theorem 5, it remains to show that checking whether $\hat{T}_{\mathcal{D},\varphi} \models \varphi$ is decidable.

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. (Sketch) To perform the check, we use the following recursive procedure (the cases of boolean connectives $\neg$, $\wedge$ and $\vee$ are as standard):

```
procedure CHECK$\hat{T}(q, \varphi)$
    if $\varphi = \phi \in \mathcal{L}_n$ and $\phi$ is uniform in $s$ then
        return $\hat{T}_{\mathcal{D},\varphi}, q \models \varphi$;
    end if
    if $\varphi = \phi \in \mathcal{L}_n$ and $\phi$ is uniform in $\sigma$ then
        if $q_\sigma$ does not exist in $Q$ then
            return $false$;
        else
            return CHECK$\hat{T}(q_\sigma, \varphi[\sigma/s])$;
        end if
    end if
    if $\varphi = \exists s.do([\alpha_1, \ldots, \alpha_n], S_0) \sqsubseteq s \wedge \phi$ then
        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
            if CHECK$\hat{T}(q_m, \varphi[s/\sigma']) == true$, for $\sigma' = do([\alpha_1, \ldots, \alpha_{m-1}], S_0)$ then
                return $true$;
            end if
        end for
        return $false$;
    end if
end procedure
```

(Termination and correctness proofs omitted for brevity.)

Lemmas 1, 2, 3 and 4 prove, together, Th. 5. By exploiting Th. 5 we can prove the following notable result.

Theorem 6. *Given a BAT-GFDB $\mathcal{D}$ and a natural number B, checking whether $\mathcal{D}$ is C-bounded by B is decidable.*

Proof. (Sketch) From $\mathcal{D}$, a theory $\mathcal{D}'$, C-bounded by B by construction, can be derived that matches $\mathcal{D}$ up to the situations (if any) that violate C-boundedness, and s.t. the situations preceding a violation are marked with distinguished facts. This can be done because the formula $\varphi(s) = \bigvee_{\Psi \in 2^{T_V}} \exists \boldsymbol{y}.F(\boldsymbol{x}, s) \equiv \bigvee_{\psi \in \Psi} \psi$, which, for appropriate V, expresses that s is C-bounded by B, is regressable. We can then prove that $\mathcal{D} \models \forall s.\varphi(s)$ iff $\mathcal{D}' \models \forall s.\varphi'(s)$, with $\varphi'(s)$ expressing that situation s is not marked with any of the distinguished facts discussed above. Since $\mathcal{D}'$ is C-bounded, by Th. 5, $\mathcal{D}' \models \varphi'$ is decidable. Thus, so is $\mathcal{D} \models \varphi$.

Related work

Our work relates definitional KBs and BATs over them to the work in databases and, in particular, constraint query languages (CQL). The representation of infinitely many tuples we use here shares a lot of similarities with the work in database theory about finitary representations of infinite query answers [1], and the more general approach of CQL of [7] that is our main inspiration.

Proper KBs [8] also generalize regular databases by allowing possibly infinite sets of positive or negative ground facts to be expressed, as well as tuples to be undefined (incomplete information). This provides enough expressive power to make theorem proving, in general, undecidable with proper KBs, even for queries about S_0. This is overcome in [8] by an approximate reasoning method which is always logically sound, but also complete only under specific conditions. The case of a KB as a generalized database with equality constraints, instead, is less expressive, as it captures only complete information, but effective, and logically correct methods exist for query answering, projection, and progression. In particular, wrt (possibly generalized) projection queries, our approach can deal with full first-order queries over S_0 and any projected ground future situation, as well as generalized projection queries of a particular form. In contrast, the approach of [8] can guarantee completeness only under some constraints on the first-order queries [11], that limit their expressivity.

The case of bounded action theories of [3] is the only one in the literature that investigates conditions under which generalized projection queries can be decided over BATs. They require a finite upperbound on the number of positive atomic facts for all models and situations, and look into queries that can be expressed over BATs using a first-order variant of the μ-calculus [4]. Our work extends this work in the case where fluents may have *infinite* extensions, concisely represented by means of equality constraints. We are able then to prove similar results for a wide class of general projection queries. Finally, the two-variable variant of situation calculus language in [6] allows richer forms of incomplete information in the initial KB, but is bound by the limitation of using only two variables, e.g., not being able to express reachability relations.

Conclusions and future work

In this paper we looked into situation calculus action theories over generalized fluent databases with equality constraints (GFDB), connecting the situation calculus with constraint query languages. We showed that GFDBs characterize the class of definitional KBs and that for action theories over such KBs (BAT-GFDBs), the KBs are *closed under progression.* We proved that simple projection queries over BAT-GFDBs are decidable in general. Also, extending the notion of *boundedness* proposed in [3], we introduced the notion of *C-boundedness* and showed that, under this, a wide class of generalized projection queries that include quantification over situations is decidable. Finally, we proved decidability of checking C-boundedness of a BAT-GFDB for some bound.

For future work we want to consider other constraints, in particular extending GFDBs to include linear orderings. We believe that this work can provide the ground for specifying action theories that capture topological properties and reason effectively over rich temporal aspects relating to projection and progression. We also intend to look into controlled ways to express incomplete information similar to the extensions of proper KBs in [10] and [2]. We believe that the latter can be used to include a practical form of incomplete information.

References

1. Chomicki, J., Imieliński, T.: Finite representation of infinite query answers. ACM Trans. Database Syst. 18(2), 181–223 (1993)
2. De Giacomo, G., Lespérance, Y., Levesque, H.J.: Efficient Reasoning in Proper Knowledge Bases with Unknown Individuals. In: Proc. of IJCAI'11. pp. 827–832 (2011)
3. De Giacomo, G., Lespérance, Y., Patrizi, F.: Bounded Situation Calculus Action Theories and Decidable Verification. In: Proc of KR'12 (2012)
4. Emerson, E.A.: Model Checking and the Mu-calculus. In: Descriptive Complexity and Finite Models. pp. 185–214 (1996)
5. Enderton, H., Enderton, H.B.: A Mathematical Introduction to Logic, Second Edition. Academic Press (2001)
6. Gu, Y., Soutchanski, M.: Decidable Reasoning in a Modified Situation Calculus. In: Proc. of IJCAI '07. pp. 1891–1897 (2007)
7. Kanellakis, P.C., Kuper, G.M., Revesz, P.Z.: Constraint Query Languages. Journal of Computer and System Sciences 51(1), 26–52 (1995)
8. Levesque, H.J.: A Completeness Result for Reasoning with Incomplete First-Order Knowledge Bases. In: Proc. of KR'98 (1998)
9. Lin, F., Reiter, R.: How to Progress a Database. Artificial Intelligence 92(1-2), 131–167 (1997)
10. Liu, Y., Lakemeyer, G., Levesque, H.J.: A Logic of Limited Belief for Reasoning with Disjunctive Information. In: Proc. of KR'04. pp. 587–597 (2004)
11. Liu, Y., Lakemeyer, G.: On the Expressiveness of Levesque's Normal Form. J. Artif. Int. Res. 31(1), 259–272 (2008)
12. Liu, Y., Levesque, H.J.: Tractable Reasoning with Incomplete First-Order Knowledge in Dynamic Systems with Context-Dependent Actions. In: Proc.of IJCAI'05 (2005)
13. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
14. Reiter, R.: The Projection Problem in the Situation Calculus: A Soundness and Completeness Result, with an Application to Database Updates. In: Proc. of AIPS'92. pp. 198–203 (1992)
15. Vassos, S., Levesque, H.J.: How to progress a database III. Artificial Intelligence 195, 203–221 (2013)
16. Vassos, S., Patrizi, F.: A Classification of First-Order Progressable Action Theories in Situation Calculus. In: Proc. of IJCAI'13 (2013)