In: Proc. 8th International Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001), pp. 407–421. LNCS 2250, © 2001 Springer.

Reasoning about Evolving Nonmonotonic Knowledge Bases[*]

Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,michael,giuliana,tompits}@kr.tuwien.ac.at

Abstract. Recently, several approaches to updating knowledge bases modeled as extended logic programs (ELPs) have been introduced, ranging from basic methods to incorporate (sequences of) sets of rules into a logic program, to more elaborate methods which use an update policy for specifying how updates must be incorporated. In this paper, we introduce a framework for reasoning about evolving knowledge bases, which are represented as ELPs and maintained by an update policy. We describe a formal model which captures various update approaches, and define a logical language for expressing properties of evolving knowledge bases. We further investigate the semantical properties of knowledge states with respect to reasoning. In particular, we describe finitary characterizations of the evolution, and derive complexity results for our framework.

1 Introduction

Updating knowledge bases is an important issue in the area of data and knowledge representation. While this issue has been studied extensively in the context of classical knowledge bases [18, 11], attention to it in the area of nonmonotonic knowledge bases, in particular in logic programming, is more recent. Various approaches to evaluating logic programs in the light of new information have been presented, cf. [1]. The proposals range from basic methods to incorporate an update U, given by a set of rules, or a sequence $U_1, \dots, U_n$ of such updates into a (nonmonotonic) logic program P [1, 21, 13, 6], to more general methods which use an *update policy* to specify, by means of update actions, how the updates $U_1, \dots, U_n$ should be incorporated into the current state of knowledge [17, 2, 8]. Using these approaches, queries to the knowledge base, like "is a fact f true in P after updates $U_1, \dots, U_n$?", can then be evaluated.

Notably, the formulation of such queries is treated on an *ad-hoc basis*, and more involved queries such as "is a fact f true in P after updates $U_1, \dots, U_n$ and possibly further updates?" are not considered. More generally, reasoning about an evolving knowledge base KB, maintained using an update policy, is not formally addressed. However, it is desirable to know about properties of the contents of the evolving knowledge base, which also can be made part of a specification for an update policy. For example, it may be important to know that a fact a is always true in KB, or that a fact b is never true in KB. Analogous issues, called *maintenance* and *avoidance*, have been

[*] This work was supported by the Austrian Science Fund (FWF) under grants P13871-INF and N Z29-INF.

recently studied in the agent community [20]. Other properties may involve temporal relationships such as if $message\_to(tom)$ is true in KB at some point, meaning that a message should be sent to Tom, then $sent\_message\_to(tom)$ will become true in the evolving KB at some point, representing that a message to Tom was sent.

In this paper, we aim at a framework for expressing reasoning problems over evolving knowledge bases, which are modeled as extended logic programs [12] and possibly maintained by an update policy as described above. In particular, we are interested in a logical language for expressing properties of the evolving knowledge base, whose sentences can be evaluated using a clear-cut formal semantics. The framework should, on the one hand, be general enough to capture different approaches to incorporating updates $U_1, \ldots, U_n$ into a logic program P and, on the other hand, pay attention to the specific nature of the problem. Furthermore, it should be possible to evaluate a formula, which specifies a desired evolution behavior, across different realizations of update policies based on different grounds.

The main contributions of this paper are summarized as follows.

(1) We introduce a formal model in which various approaches for updating extended logic programs can be expressed (Section 3). In particular, we introduce the concept of an *evolution frame*, which is a structure $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ whose components serve to describe the evolution of knowledge states. Informally, a *knowledge state* $s = \langle KB; E_1, \ldots, E_n \rangle$ consists of an initial knowledge base KB, given by an extended logic program over an alphabet $\mathcal{A}$, and a sequence $E_1, \ldots, E_n$ of *events*, which are sets of rules E_i, drawn from a class of possible events $\mathcal{EC}$, that are communicated to an agent maintaining the knowledge base. The agent reacts on an event by adapting its belief set through the update policy Π, which singles out update actions $A \subseteq \mathcal{AC}$ from a set of possible update actions $\mathcal{AC}$ for application. These update actions are executed, at a physical level, by compilation, using a function ρ into a single logic program P, or, more generally, into a sequence $(P_1, \ldots, P_n)$ of logic programs, denoted $comp_{EF}(s)$. The semantics of the knowledge state s, its *belief set*, $Bel(s)$, is given by the belief set of the compiled knowledge state, and is obtained by applying a belief operator $Bel(\cdot)$ for (sequences of) logic programs to $comp_{EF}(s)$. Suitable choices of EF allow one to model different settings of logic program updates, such as [1, 17, 13, 6].

(2) We define the syntax and, based on evolution frames, the semantics of a logical language for reasoning about evolving knowledge bases (Section 4), which employs linear and branching-time operators familiar from Computational Tree Logic (CTL) [9]. Using this language, properties of an evolving knowledge base can be formally stated and evaluated in a systematic fashion, rather than ad hoc. For example, the above maintenance and avoidance problems can be expressed by formulas $\mathsf{AG}\, a$ and $\mathsf{AG}\neg b$, respectively.

(3) We investigate semantical properties of knowledge states for reasoning (Section 5). In particular, since in principle a knowledge base may evolve forever, we are concerned with finitary characterizations of evolution. To this end, we introduce various notions of equivalence between knowledge states, and show several filtration results.

(4) We derive complexity results for reasoning (Section 6). Namely, given an evolution frame EF, a knowledge state s, and a formula φ, does $EF, s \models \varphi$ hold? While this problem is undecidable in general, we single out meaningful conditions under which

the problem has 2-EXPSPACE, EXPSPACE, and PSPACE complexity, respectively, and apply this to the EPI framework under the answer set semantics [8], showing that its propositional fragment has PSPACE-complexity. We also consider the complexity of sequences of extended logic programs (ELPs). We show that deciding whether two sequences $\boldsymbol{P} = (P_1, \ldots, P_n)$ and $\boldsymbol{Q} = (Q_1, \ldots, Q_m)$ of propositional ELPs are strongly equivalent under update answer set semantics, i.e., for every sequence $\boldsymbol{R} = (R_1, \ldots, R_k)$, $k \geq 0$, the concatenated sequences $\boldsymbol{P} + \boldsymbol{R}$ and $\boldsymbol{Q} + \boldsymbol{R}$ have the same belief sets, is coNP-complete. This is not immediate, since potentially infinitely many $\boldsymbol{P} + \boldsymbol{R}$ and $\boldsymbol{Q} + \boldsymbol{R}$ need to be checked.

By expressing various approaches in our framework, we obtain a formal semantics for reasoning problems in them. Furthermore, results about properties of these approaches (e.g., complexity results) may be concluded from the formalism by this embedding, as we illustrate for the EPI framework.

2 Preliminaries

We consider knowledge bases represented as *extended logic programs* (ELPs) [12], which are finite sets of rules built over a first-order alphabet $\mathcal{A}$ using default negation not and strong negation $\neg$. A rule has the form

$$r: \quad L_0 \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n, \tag{1}$$

where each L_i is a literal of form A or $\neg A$, where A is an atom over $\mathcal{A}$. The set of all rules is denoted by $\mathcal{L}_{\mathcal{A}}$. We call L_0 the *head* of r (denoted by $H(r)$), and the set $\{L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n\}$ the *body* of r (denoted by $B(r)$). We allow the case where L_0 is absent from r; such a rule r is called a *constraint*. If $B(r) = \emptyset$, then r is called *fact*. We often write L_0 for a fact $r = L_0 \leftarrow$. Further extensions, e.g., not in the rule head [1], might be added to fit other frameworks.

An *update program*, $\boldsymbol{P}$, is a sequence $(P_1, \ldots, P_n)$ of ELPs ($n \geq 1$), representing the evolution of program P_1 in the light of new rules $P_2, \ldots, P_n$. The semantics of update programs can abstractly be described as a mapping $Bel(\cdot)$, which associates with every sequence $\boldsymbol{P}$ a set $Bel(\boldsymbol{P}) \subseteq \mathcal{L}_{\mathcal{A}}$ of rules, intuitively viewed as the consequences of $\boldsymbol{P}$. $Bel(\cdot)$ may be instantiated in terms of various proposals for update semantics, like, e.g., the approaches described in [1, 21, 13, 6, 17].

For a concrete example, we consider the answer set semantics for propositional update programs introduced in [6, 7], which defines answer sets of $\boldsymbol{P} = (P_1, \ldots, P_n)$ in terms of answers sets of a single ELP P as follows. An *interpretation*, S, is a set of classical literals containing no opposite literals A and $\neg A$. The *rejection set*, $Rej(S, \boldsymbol{P})$, of $\boldsymbol{P}$ with respect to an interpretation S is $Rej(S, \boldsymbol{P}) = \bigcup_{i=1}^{n} Rej_i(S, \boldsymbol{P})$, where $Rej_n(S, \boldsymbol{P}) = \emptyset$, and, for $n > i \geq 1$, $Rej_i(S, \boldsymbol{P})$ contains every rule $r \in P_i$ such that $H(r') = \neg H(r)$ and $S \models B(r) \cup B(r')$, for some $r' \in P_j \setminus Rej_j(S, \boldsymbol{P})$ with $j > i$. That is, $Rej(S, \boldsymbol{P})$ contains the rules in $\boldsymbol{P}$ which are rejected by unrejected rules from later updates. Then, an interpretation S is an *answer set* of $\boldsymbol{P} = (P_1, \ldots, P_n)$ iff S is a consistent answer set [12] of the program $P = \bigcup_i P_i \setminus Rej(S, \boldsymbol{P})$. The set of all answer sets of $\boldsymbol{P}$ is denoted by $\mathcal{AS}(\boldsymbol{P})$. This definition properly generalizes consistent answer sets from single ELPs to sequences of ELPs. Update answer sets for arbitrary (non-ground) update programs $\boldsymbol{P}$ are defined in terms of their ground instances similar to the case of answer sets for ELPs [12].

Example 1. Let $P_0 = \{b \leftarrow \mathit{not}\, a,\ a \leftarrow\}$, $P_1 = \{\neg a \leftarrow,\ c \leftarrow\}$, and $P_2 = \{\neg c \leftarrow\}$. Then, P_0 has the single answer set $S_0 = \{a\}$ with $\mathit{Rej}(S_0, P_0) = \emptyset$; (P_0, P_1) has as answer set $S_1 = \{\neg a, c, b\}$ with $\mathit{Rej}(S_1, (P_0, P_1)) = \{a \leftarrow\}$; and $(P_0,\ P_1,\ P_2)$ has the unique answer set $S_2 = \{\neg a, \neg c, b\}$ with $\mathit{Rej}(S_2, (P_0, P_1, P_2)) = \{c \leftarrow, a \leftarrow\}$.

The belief operator $\mathit{Bel}_E(\cdot)$ in the framework of [6] is given by $\mathit{Bel}_E(\boldsymbol{P}) = \{r \in \mathcal{L}_\mathcal{A} \mid S \models r$ for all $S \in \mathcal{AS}(\boldsymbol{P})\}$, where $S \models r$ means that for each ground instance r' of r, either $H(r') \in S$, or $L \notin S$ for some $L \in B(r')$, or $L \in S$ for some $\mathit{not}\, L \in B(r')$.

3 Knowledge-Base Evolution

We start with the basic formal notions of an *event* and of the *knowledge state* of an agent maintaining a knowledge base.

Definition 1. *Let $\mathcal{A}$ be some alphabet. An* event class over $\mathcal{A}$ (*or simply* event class, *if no ambiguity arises*) *is a collection $\mathcal{EC} \subseteq 2^{\mathcal{L}_\mathcal{A}}$ of finite sets of rules. The members $E \in \mathcal{EC}$ are called* events.

Informally, $\mathcal{EC}$ describes the possible events (i.e., sets of communicated rules) an agent may experience. In the most general case, an event is an arbitrary ELP; in a simpler setting, an event may just be a set facts. In a deductive database setting, the latter case corresponds to an extensional database undergoing change while the intensional part of the database remains fixed.

Definition 2. *Let $\mathcal{EC}$ be an event class over some alphabet $\mathcal{A}$. A* knowledge state over $\mathcal{EC}$ (*simply, a* knowledge state) *is a tuple $s = \langle KB; E_1, \ldots, E_n\rangle$, where $KB \subseteq \mathcal{L}_\mathcal{A}$ is an ELP* (*called* initial knowledge base) *and each E_i $(1 \leq i \leq n)$ is an event from $\mathcal{EC}$. The* length of s, *denoted $|s|$, is n.*

Intuitively, $s = \langle KB; E_1, \ldots, E_n\rangle$ captures the agent's knowledge, starting from its initial knowledge base. When a new event E_{n+1} occurs, the current knowledge state s changes to $s' = \langle KB; E_1, \ldots, E_n, E_{n+1}\rangle$, and the agent is required to adapt its belief set in accordance with the new event by obeying its given update policy.

The "universe" in which the evolution of an agent's knowledge base takes place is given by the following concept:

Definition 3. *An* evolution frame *is a tuple $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, \mathit{Bel}\rangle$, where*

– *$\mathcal{A}$ is a finite (first-order) alphabet;*
– *$\mathcal{EC}$ is an event class over $\mathcal{A}$;*
– *$\mathcal{AC}$ is a set of* update commands (*or* actions)*;*
– *Π is an* update policy, *which is a function mapping every knowledge state s over $\mathcal{EC}$ and an event $E \in \mathcal{EC}$ into a set $\Pi(s, E) \subseteq \mathcal{AC}$ of update commands;*

– *ρ is a mapping, called* realization assignment, *which assigns to each knowledge state s over $\mathcal{EC}$ and each set $A \subseteq \mathcal{AC}$ of update commands a sequence $\rho(s, A) = (P_0, \ldots, P_n)$ of ELPs $P_i \subseteq \mathcal{L}_{\mathcal{A}}$ $(1 \leq i \leq n)$; and*
– *Bel is a belief operator for sequences of ELPs.*

The set of all knowledge states determined by EF is denoted by $\mathcal{S}_{EF}$.

The components of an evolution frame allow us to model various update approaches, as we discuss later on.

We already mentioned above that different event classes $\mathcal{EC}$ might be conceived. Simple, elementary update commands are $insert(r)$ and $delete(r)$, which add and remove a rule to a logic program, respectively, without a sophisticated semantics handling potential inconsistencies (which may be delegated to the underlying update semantics). More involved update commands have been proposed in the literature (cf., e.g., [2, 8]). However, several update frameworks can be modeled using these simple commands.

Update policies Π allow for specifying sensible and flexible ways to react upon incoming events. A very simple policy is $\Pi_{ins}(s, E) = \{insert(r) \mid r \in E\}$; it models an agent which incorporates the new information unconditionally. More sophisticated policies may define exceptions for the incorporation of rules from events, or the insertion of rules may be conditioned on the belief in other rules.

While Π determines *what* to do, the realization assignment ρ states *how* this should be done. Informally, $\rho(s, A)$ "executes" actions A on the knowledge state s by producing a logic program P or, more generally, a sequence $\boldsymbol{P}$ of logic programs. We can use ρ to "compile" a knowledge state s into a (sequence of) logic programs, by determining the set of actions A from the last event in s. We introduce the following notation.

For any knowledge state $s = \langle KB; E_1, \ldots, E_n \rangle$ over $\mathcal{EC}$, denote by $\pi_i(s) = \langle KB; E_1, \ldots, E_i \rangle$ its projection to the first i events, for $0 \leq i \leq n$. We call $\pi_i(s)$ a *previous knowledge state* (or simply an *ancestor*) of s if $i < n$. Dually, each knowledge state s' over $\mathcal{EC}$ is *a future knowledge state* (or simply a *descendant*) of s if s is previous to s'. Furthermore, $\pi_{n-1}(s)$ is the *predecessor* of s, and s' is a *successor* of s if s is predecessor of s'. Finally, for events $E'_1, \ldots, E'_m$, we write $s + E'_1, \ldots, E'_m$ to denote the concatenated knowledge state $\langle KB; E_1, \ldots, E_n, E'_1, \ldots, E'_m \rangle$ (a similar notation applies to the concatenation of sequences of logic programs).

Definition 4. *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ be an evolution frame. For any knowledge state $s = \langle KB; E_1, \ldots, E_n \rangle$ over $\mathcal{EC}$, the* compilation associated with s *is*

$$comp_{EF}(s) = \begin{cases} \rho(s, \emptyset) & \text{if } |s| = 0, \text{ i.e., } s = \langle KB \rangle; \\ \rho(\pi_{n-1}(s), \Pi(\pi_{n-1}(s), E_n)) & \text{otherwise.} \end{cases}$$

This definition of compilation is fairly general. It first computes the actions for the latest event E_n, and then requires that these actions are executed on the predecessor state. Observe that, in view of $comp_{EF}(s)$, we could equally well model update policies as unary functions $\hat{\Pi}(\cdot)$ such that $\hat{\Pi}(s) = \Pi(\pi_{n-1}(s), E_n)$. However, we chose binary update policies to stress the importance of the last event in s.

An important class of compilations are those in which $comp(s')$ for a future knowledge state s' results by appending some further elements to the sequence $comp(s)$ of logic programs for the current knowledge state s. This motivates the following notion:

Definition 5. *Given an evolution frame $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel\rangle$, $comp_{EF}(\cdot)$ is* incremental *iff, for each $s = \langle KB; E_1, \ldots, E_n\rangle$, $comp_{EF}(s) = (P_0, \ldots, P_n)$ such that $\rho(\langle KB\rangle, \emptyset) = P_0$ and $\rho(\pi_{i-1}(s), \Pi(\pi_{i-1}(s), E_i)) = (P_0, \ldots, P_i)$ for $1 \leq i \leq n$.*

This amounts to the expected meaning:

Proposition 1. *The mapping $comp_{EF}(\cdot)$ is incremental iff, for each knowledge state s, $comp_{EF}(s) = Q$ if $|s| = 0$, and $comp_{EF}(s) = comp_{EF}(\pi_{|s|-1}(s)) + Q$ otherwise, where Q is a logic program and "+" is the concatenation of sequences.*

A simple, incremental compilation results for $\mathcal{AC}_{ins} = \{insert(r) \mid r \in \mathcal{L}_\mathcal{A}\}$, $\Pi = \Pi_{ins}$ as defined above, and ρ_{ins} such that $comp_{EF}(\langle KB\rangle) = KB$ and $comp_{EF}(s) = comp_{EF}(\pi_{|s|-1}(s)) + (\{r \mid insert(r) \in A\})$. Note that $comp_{EF}(\langle KB; E_1, \ldots, E_n\rangle)$ is in this setting just the sequence $(KB, E_1, \ldots, E_n)$.

While incremental compilations are natural, we stress that others are of course also relevant. In particular, the compilation might perform optimizations (cf. Section 5.2), or output only an ordinary logic program.

Finally, the belief set emerging from a knowledge state is as follows:

Definition 6. *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel\rangle$ be an evolution frame and s a knowledge state. The belief set of s, denoted $Bel(s)$, is given by $Bel(comp_{EF}(s))$.*

Remarks. Our definition of an update policy and of a realization assignment, which effectively lead to the notion of a compilation, is quite general. We may stipulate additional postulates upon them, like the incrementability property or an iterativity property (which me omit here), and similar on $Bel(\cdot)$.

Our definition does not capture nondeterministic update policies, where $\Pi(s, E)$ may return one out of several possible sets of update actions. Accordingly, the notion of a knowledge state can be extended by taking previous actions into account, i.e., a knowledge state s is then of the form $\langle KB, (E_1, A_1), \ldots, (E_n, A_n)\rangle$, where each E_i is an event, and A_i is the set of update commands executed at step i. In practice, we may assume a suitable *selection function* σ, which chooses one of the possible outcomes of $\Pi(s, E)$, and we are back to a deterministic update policy Π_σ. If the selection function σ is unknown, we may consider all evolution frames EF_σ arising for each σ.

Example 2. Consider a rather simple mailing agent, which has the following initial knowledge base KB, whose rules are instantiated over suitable variable domains:

$$\begin{array}{lrl}
r_1: & type(M, private) & \leftarrow from(M, tom); \\
r_2: & type(M, business) & \leftarrow subject(M, project); \\
r_3: & type(M, other) & \leftarrow not\ type(M, private), not\ type(M, business), msg(M); \\
r_4: & trash(M) & \leftarrow remove(M), not\ save(M); \\
r_5: & remove(M) & \leftarrow date(M, T), today(T'), not\ save(M), T' > (T + 30); \\
r_6: & found(M) & \leftarrow search(T), type(M, T), not\ trash(M); \\
r_7: & success & \leftarrow found(M); \\
r_8: & failure & \leftarrow search(T), not\ success.
\end{array}$$

The knowledge base contains rules about classifying message types (r_1–r_3), trash and removal of mails (r_4, r_5), and further rules (r_6–r_8) to determine success or failure of a search for messages of a particular type. An event E might consist in this setting of one or more of the following items:

– at most one fact $today(d)$, for some date d;
– a fact $empty\_trash$, which causes messages in the trash to be eliminated;
– facts $save(m)$ or $remove(m)$, for mail identifiers m;
– at most one fact $search(t)$, for some mail type $t \in \{other, business, private\}$;
– zero or more sets of facts $from(m,n)$, $subject(m,s)$, or $date(m,d)$ for mail identifier m, name n, subject s, and date d.

The update policy Π may be as follows:

$$\begin{aligned}
\Pi(s,E) = {} & \{insert(R) \mid R \in E\} \cup \{insert(msg(M)) \mid from(M,N) \in E\} \\
& \cup \{delete(today(D)) \mid today(D') \in E, today(D) \in Bel(s)\} \\
& \cup \{delete(\alpha) \mid \alpha \in \{trash(M), msg(M), type(M,T)\}, \\
& \qquad\qquad empty\_trash \in E, trash(M) \in Bel(s)\} \\
& \cup \{delete(\alpha) \mid \alpha \in \{from(M,N), subject(M,S), date(M,D)\}, \\
& \qquad\qquad save(M) \notin Bel(s), msg(M) \in Bel(s), remove(M) \in E\} \\
& \cup \{delete(\alpha) \mid \alpha \in Bel(s) \cap \{search(T), found(T), success, \\
& \qquad\qquad\qquad failure, empty\_trash\}\ \}
\end{aligned}$$

This update policy (which does not respect possible conflicts of $save$ and $remove$), intuitively adds all incoming information, plus a fact $msg(M)$ for each incoming mail to the knowledge base. The current date is maintained by deleting the old date. As well, all old information from a previous event, relative to a search or to the trash, is removed. If an event contains $empty\_trash$, then all messages in the trash are eliminated.

Capturing frameworks for knowledge evolution. Finally, we briefly discuss how existing frameworks for updating nonmonotonic knowledge bases can be captured in terms of evolution frames. This is possible at two different levels:

(1) At an "immediate update" level, frameworks for updating logic programs can be considered, where each event is an *update program*, and the update policy is the (implicit) way in which update programs and the current knowledge are combined, depending on the semantics of updates of each approach. For example, the formalisms of update programs [6, 7], dynamic logic programming [1], revision programming [16, 17], abductive theory updates [13], and updates through prioritized logic programs (PLPs) [21] fall into this category.

(2) At a higher level, frameworks can be considered which allow for specifying an explicit *update policy* in some specification language, and which offer a greater flexibility in the handling of updates. Examples of such frameworks are EPI [8], LUPS [2], and, while not directly given in these terms, $\mathcal{PDL}$ [14].

For illustration, we consider update programs [6] and the EPI framework for update policies. Update programs are captured by the following evolution frame:

$$EF_{\lhd} = \langle \mathcal{A}, \mathcal{EC}_{\mathcal{A}}, \mathcal{AC}_{ins}, \Pi_{ins}, \rho_{ins}, Bel_E \rangle,$$

where $\mathcal{EC}_{\mathcal{A}}$ is the collection of all ELPs over $\mathcal{A}$, and Bel_E is the belief operator defined in Section 2. The EPI framework corresponds to the evolution frame

$$EF_{\mathsf{EPI}} = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}_{\mathsf{EPI}}, \Pi_{\mathsf{EPI}}, \rho_{\mathsf{EPI}}, Bel_E \rangle,$$

where

– $\mathcal{AC}_{\mathsf{EPI}} = \{\mathbf{assert}(r), \mathbf{retract}(r), \mathbf{always}(r), \mathbf{cancel}(r),$
$\mathbf{assert\_event}(r), \mathbf{retract\_event}(r), \mathbf{always\_event}(r) \mid r \in \mathcal{L}_{\mathcal{A}}\}$
and the commands have the meaning as in [8];
– Π_{EPI} is defined by any set of update statements in the language EPI, which are evaluated through a logic program as defined in [8];
– ρ_{EPI} realizes the translation $tr(KB; U_1, \ldots, U_n)$ from [8], which compiles the initial knowledge base KB and the sets of update commands $U_1, \ldots, U_n$, in response to the events $E_1, \ldots, E_n$ in $s = \langle KB, E_1, \ldots, E_n \rangle$, into a sequence $(P_0, \ldots, P_n)$ of ELPs. The resulting compilation $comp_{\mathsf{EPI}}$ is incremental.

Furthermore, the following formalisms can be expressed in a similar fashion: dynamic logic programming [1] (by allowing not in rule heads), LUPS [2], abductive theory updates [13], and program updates by means of PLPs [21]. Thus, several well-known approaches to updating logic programs can be modeled by evolution frames.

4 Reasoning About Knowledge-Base Evolution

We now introduce our logical language for expressing properties of evolving knowledge bases. The primitive logical operators of the language are: (i) the Boolean connectives $\wedge$ ("and") and $\neg$ ("not"); (ii) the evolution quantifiers A ("for all futures") and E ("for some future"); and (iii) the linear temporal operators X ("next time") and U ("until").

Atomic formulas are identified with rules in $\mathcal{L}_{\mathcal{A}}$; composite formulas are either *state formulas* or *evolution formulas*, defined as follows:

1. Each atomic formula is a state formula.
2. If φ, ψ are state formulas, then $\varphi \wedge \psi$ and $\neg\varphi$ are state formulas.
3. If φ is an evolution formula, then $\mathsf{E}\varphi$ and $\mathsf{A}\varphi$ are state formulas.
4. If φ, ψ are state formulas, then $\mathsf{X}\varphi$ and $\varphi\mathsf{U}\psi$ are evolution formulas.

Further Boolean connectives $\vee$ ("or"), $\supset$ ("implies"), and $\equiv$ ("equivalence") are defined in the usual manner. As well, we use $\mathsf{F}\varphi = \top\mathsf{U}\varphi$ ("finally ϕ"), where $\top$ stands for any tautology, $\mathsf{AG}\varphi = \neg\mathsf{EF}\neg\phi$, and $\mathsf{EG}\phi = \neg\mathsf{AF}\phi$ ("globally ϕ").

Next, we define the semantics of such formulas with respect to a given evolution frame $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$. To this end, we introduce the following notation:

A sequence $p = (s_i)_{i \geq 0}$ of knowledge states over $\mathcal{EC}$ is called a *path* iff each s_i $(i > 0)$ is a successor of s_{i-1}. We denote by p_i the state at position i in p, i.e., $p_i = s_i$.

Definition 7. *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ be an evolution frame, s a knowledge state over $\mathcal{EC}$, and p a path. The relation $\models$ is recursively defined as follows:*

1. *$EF, s \models r$ iff $r \in Bel(s)$, for any atomic formula r;*
2. *$EF, s \models \varphi_1 \wedge \varphi_2$ iff $EF, s \models \varphi_1$ and $EF, s \models \varphi_2$;*
3. *$EF, s \models \neg\varphi$ iff $EF, s \not\models \varphi$;*
4. *$EF, s \models \mathsf{E}\varphi$ iff $EF, p' \models \varphi$, for some path p' starting at s;*
5. *$EF, s \models \mathsf{A}\varphi$ iff $EF, p' \models \varphi$, for each path p' starting at s;*
6. *$EF, p \models \mathsf{X}\varphi$ iff $EF, p_1 \models \varphi$;*
7. *$EF, p \models \varphi_1\mathsf{U}\varphi_2$ iff $EF, p_i \models \varphi_2$ for some $i \geq 0$ and $EF, p_j \models \varphi_1$ for all $j < i$.*

If $EF, s \models \varphi$ holds, then knowledge state s is said to *satisfy* formula φ *in the evolution frame* EF (or φ is a *consequence of* s *in the evolution frame* EF).

Notice that any evolution frame EF induces an infinite transition graph which amounts to a standard Kripke structure $K_{EF} = \langle S, R, L \rangle$, where $S = \mathcal{S}_{EF}$ is the set of knowledge states, R is the successor relation between knowledge states, and L labels each state s with $Bel(S)$, such that s satisfies φ in EF iff $K_{EF}, s \models \varphi$ (where $\models$ is defined in the usual way).

Example 3. In order to see whether the mailing agent in Example 2 works properly, we may consider the following properties. For convenience, we allow in formulas non-ground rules as atoms, which stand for the conjunction of all ground instances which is assumed to be finite. Recall that we identify facts with literals.

1. There can never be two current dates:
$$\mathsf{AG}((\mathit{today}(D) \wedge \mathit{today}(D')) \supset D = D'). \tag{2}$$
2. The type of a message cannot change:
$$\mathsf{AG}(\mathit{type}(M,T) \supset \neg\mathsf{EF}(\mathit{type}(M,T') \ \wedge \ T \neq T')). \tag{3}$$
3. A message is not trashed until it is either deleted or saved:
$$\mathsf{AG}\big(\mathit{msg}(m) \supset \mathsf{AG}(\neg \mathit{trash}(m) \mathsf{U} (\mathit{delete}(m) \vee \mathit{save}(m)))\big). \tag{4}$$

While the initial KB satisfies formulas (2) and (4) in the respective EPI evolution frame EF_{EPI}, it is easily seen that it does not satisfy formula (3).

5 Knowledge-State Equivalence

While syntactically different, it may happen that knowledge states s and s' are semantically equivalent in an evolution frame, i.e., s and s' may have the same set of consequences for the current and all future events. We now consider how such equivalences can be exploited to filtrate a given evolution frame EF such that, under suitable conditions, we can decide $EF, s \models \varphi$ in a finite structure extracted from the associated Kripke structure K_{EF}. We start with the following notions of equivalence.

Definition 8. *Let* $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ *be an evolution frame and* $k \geq 0$ *some integer. Furthermore, let* s, s' *be knowledge states over* $\mathcal{EC}$*. Then,*

1. s *and* s' *are* k-equivalent *in* EF*, denoted* $s \equiv^k_{EF} s'$*, if* $Bel(s + E_1, \ldots, E_{k'}) = Bel(s' + E_1, \ldots, E_{k'})$*, for all events* $E_1, \ldots, E_{k'}$ *from* $\mathcal{EC}$ *and all* $k' \leq k$*;*
2. s *and* s' *are* strongly equivalent *in* EF*, denoted* $s \equiv_{EF} s'$*, iff* $s \equiv^k_{EF} s'$ *for every* $k \geq 0$*.*

We call 0-equivalent states also *weakly equivalent*. The following result is obvious.

Theorem 1. *Let* $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ *be an evolution frame and* s, s' *knowledge states over* $\mathcal{EC}$*. Then,*

1. $s \equiv_{EF} s'$ *implies that* $EF, s \models \varphi$ *is equivalent to* $EF, s' \models \varphi$*, for any formula* φ*;*
2. $s \equiv_{EF}^{k} s'$ *implies that* $EF, s \models \varphi$ *is equivalent to* $EF, s' \models \varphi$*, for any formula* φ *in which* U *does not occur and the nesting depth w.r.t.* E *and* A *is at most* k*.*

Due to Part 1 of Theorem 1, strong equivalence can be used to filtrate an evolution frame EF in the following way. For an equivalence relation E over some set X, and any $x \in X$, let $[x]_E = \{y \mid \langle x, y\rangle \in E\}$ be the equivalence class of x and let $X/E = \{[x]_E \mid x \in X\}$ be the set of all equivalence classes. Furthermore, E is said to have a *finite index* (*with respect to* X) iff X/E is finite. Then, any equivalence relation E over some set $S \subseteq \mathcal{S}_{EF}$ of knowledge states of EF compatible with $\equiv_{EF}$ (i.e., such that $s\,E\,s'$ implies $s \equiv_{EF} s'$, for all $s, s' \in S$) induces a Kripke structure $K_{EF}^{E,S} = \langle S/E, R_E, L_E\rangle$, where $[s]_E\, R_E\, [s']_E$ iff $s\,R\,s'$ and $L_E([s]_E) = L(s)$, which is bisimilar to the Kripke structure K_{EF} restricted to the knowledge states in S. Thus, for every knowledge state s and formula φ, it holds that $EF, s \models \phi$ iff $K_{EF}^{E,S}, [s]_E \models \phi$, for any $S \subseteq \mathcal{S}_{EF}$ such that S contains all descendants of s.

In the following, we consider two cases in which S/E has finite index.

5.1 Local belief operators

In the first case, we consider $\equiv_{EF}$ itself as a relation compatible with strong equivalence. We obtain a finite index if, intuitively, the belief set $Bel(s)$ associated with s evolves differently only in a bounded context. We have the following result.

Theorem 2. *Let* $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel\rangle$ *be an evolution frame such that* $\mathcal{EC}$ *is finite, and let* $S \subseteq \mathcal{S}_{EF}$ *be some set of knowledge states over* $\mathcal{EC}$*. Then, the following two conditions are equivalent:*

(a) $\equiv_{EF}$ *has a finite index with respect to* S*.*
(b) $\equiv_{EF}^{0}$ *has a finite index with respect to* S *and there is some* $k \geq 0$ *such that* $s \equiv_{EF}^{k} s'$ *implies* $s \equiv_{EF} s'$*, for all* $s, s' \in S$*.*

Moreover, in case (*a*)*, there is some* $k \geq 0$ *such that* $|S/\equiv_{EF}| < d^{|\mathcal{EC}|^k}$*, where* $d = |S/\equiv_{EF}^{0}|$*.*

The condition that $\equiv_{EF}^{0}$ has a finite index, i.e., such that only finitely many knowledge states s have different belief sets, is, e.g., satisfied by common belief operators if every s is compiled to a sequence $comp_{EF}(s)$ of ELPs over a finite set of function-free atoms (in particular, if $\mathcal{A}$ is a finite propositional alphabet).

By taking natural properties of $Bel(\cdot)$ and $comp_{EF}(\cdot)$ into account, we can derive an alternative version of Theorem 2. To this end, we introduce the following notation.

Given a belief operator $Bel(\cdot)$, we call update programs $\boldsymbol{P}$ and $\boldsymbol{P}'$ *k-equivalent*, if $Bel(\boldsymbol{P} + (Q_1, \ldots, Q_k)) = Bel(\boldsymbol{P}' + (Q_1, \ldots, Q_k))$, for every ELPs $Q_1, \ldots, Q_i$ $(0 \leq i \leq k)$. Likewise, $\boldsymbol{P}$ and $\boldsymbol{P}'$ are *strongly equivalent*, if they are k-equivalent for all $k \geq 0$. We say that $Bel(\cdot)$ is *k-local*, if k-equivalence of $\boldsymbol{P}$ and $\boldsymbol{P}'$ implies strong equivalence of $\boldsymbol{P}$ and $\boldsymbol{P}'$, for any update programs $\boldsymbol{P}$ and $\boldsymbol{P}'$. Furthermore, $Bel(\cdot)$ is *local*, if $Bel(\cdot)$ is k-local for some $k \geq 0$. We obtain the following result:

Theorem 3. *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ be an evolution frame such that $\mathcal{EC}$ is finite and $\equiv^0_{EF}$ has a finite index with respect to $S \subseteq \mathcal{S}_{EF}$. If $Bel(\cdot)$ is local and $comp_{EF}(\cdot)$ is incremental, then $\equiv_{EF}$ has a finite index with respect to S.*

As an application of this result, we show that certain EPI evolution frames have a finite index. Recall that $Bel_E(\cdot)$ is the belief operator of the answer set semantics of update programs [7], as described in Section 2. We can show the following result:

Theorem 4. *Bel_E is local. In particular, 1-equivalence of update programs **P** and **P**$'$ implies k-equivalence of **P** and **P**$'$, for all $k \geq 1$.*

The proof is by induction and appeals to the rejection mechanism of the semantics. Furthermore, in any EPI evolution frame $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}_{\mathsf{EPI}}, \Pi_{\mathsf{EPI}}, \rho_{\mathsf{EPI}}, Bel_E \rangle$, the update policy Π_{EPI} is, informally, given by a logic program such that Π_{EPI} returns a set of update actions from a finite set A_0 of update actions, which are compiled to rules from a finite set R_0 of rules, provided $\mathcal{EC}$ is finite. Consequently, $\equiv^0_{EF}$ has finite index with respect to any set S of knowledge states s which coincide on $\pi_0(s)$, i.e. the initial knowledge base KB. Furthermore, $comp_{\mathsf{EPI}}(\cdot)$ is incremental. Thus, we obtain:

Corollary 1. *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}_{\mathsf{EPI}}, \Pi_{\mathsf{EPI}}, \rho_{\mathsf{EPI}}, Bel_E \rangle$ be an* EPI *evolution frame such that $\mathcal{EC}$ is finite, and let $S \subseteq \mathcal{S}_{EF}$ be a set of knowledge states such that $\{\pi_0(s) \mid s \in S\}$ is finite. Then, $\equiv_{EF}$ has a finite index with respect to S. Moreover, $|S/\equiv_{EF}| \leq d^{|\mathcal{EC}|}$, where $d = |S/\equiv^0_{EF}|$.*

5.2 Contracting belief operators

Next, we discuss a refinement of strong equivalence, called *canonical equivalence*, which also yields a finite index, providing the evolution frame possesses, in some sense, only a "bounded history". In contradistinction to the previous case, canonical equivalence uses semantical properties which allow for a syntactic simplification of update programs. We need the following notions.

Definition 9. *Let $Bel(\cdot)$ be a belief operator. Then, $Bel(\cdot)$ is called* contracting *iff the following conditions hold:* (*i*) *$Bel(\boldsymbol{P}+\emptyset+\boldsymbol{P}') = Bel(\boldsymbol{P}+\boldsymbol{P}')$, for all update programs **P** and **P**$'$; and* (*ii*) *$Bel(\boldsymbol{P}) = Bel(P_0, \ldots, P_{i-1}, P_i \setminus \{r\}, P_{i+1}, \ldots, P_n)$, for any sequence $\boldsymbol{P} = (P_0, \ldots, P_n)$ and any rule $r \in P_i \cap P_j$ such that $i < j$. An evolution frame $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ is contracting iff $Bel(\cdot)$ is contracting.*

Examples of contracting belief operators are $Bel_E(\cdot)$ and the analogous operator from [1]. By repeatedly removing duplicate rules r and empty programs P_i from any sequence $\boldsymbol{P} = (P_0, \ldots, P_n)$ of ELPs, we eventually obtain a non-reducible sequence $\boldsymbol{P}^* = (P^*_1, \ldots, P^*_m)$, which is called the *canonical form* of **P**. Observe that $m \leq n$ always holds, and that $\boldsymbol{P}^*$ is uniquely determined, i.e., the reduction process is Church-Rosser. We get the following property:

Theorem 5. *For any contracting belief operator $Bel(\cdot)$ and any update sequence **P**, we have that **P** and $\boldsymbol{P}^*$ are strongly equivalent.*

Let us call knowledge states s and s' in an evolution frame EF *canonically equivalent*, denoted $s \equiv_{EF}^{can} s'$, iff they are strongly equivalent in the canonized evolution frame EF^*, which results from EF by replacing $comp_{EF}(s)$ with its canonical form $comp_{EF}(s)^*$ (i.e., $comp_{EF^*}(s) = comp_{EF}(s)^*$). We note the following property.

Theorem 6. *Let EF be a contracting evolution frame. Then, $EF, s \models \varphi$ iff $EF^*, s \models \varphi$, for any knowledge state s and any formula φ. Furthermore, $\equiv_{EF}^{can}$ is compatible with $\equiv_{EF}$ for any $S \subseteq \mathcal{S}_{EF}$, i.e., $s \equiv_{EF}^{can} s'$ implies $s \equiv_{EF} s'$, for every $s, s' \in S$.*

As a result, we may use $\equiv_{EF}^{can}$ for filtration of EF, based on the following concept.

Definition 10. *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ be an evolution frame and $c \geq 0$ an integer. We say that EF is* c-bounded *iff there are functions α, f, and g such that*

1. *α is a function mapping knowledge states into sets of events such that, for each $s = \langle KB; E_1, \ldots, E_n \rangle$, $\alpha(s) = \langle E_{n-c'+1}, \ldots, E_n \rangle$, where $c' = \min(n, c)$; and*
2. *$\Pi(s, E) = f(Bel(s), \alpha(s), E)$ and $\rho(s, A) = g(Bel(s), \alpha(s), A)$, for each knowledge state $s \in \mathcal{S}_{EF}$, each event $E \in \mathcal{EC}$, and each $A \subseteq \mathcal{AC}$.*

This means that, in a c-bounded evolution frame, the compilation $comp_{EF}(s)$ depends only on the belief set of the predecessor s' of s and the latest c events in s.

Theorem 7. *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ be an evolution frame where $\mathcal{EC}$ is finite, and let $S \subseteq \mathcal{S}_{EF}$. If (i) EF is contracting, (ii) there is some finite set $R_0 \subseteq \mathcal{L}_{\mathcal{A}}$ such that $comp_{EF}(s) \subseteq R_0$, for any $s \in S$, and (iii) EF is c-bounded, for some $c \geq 0$, then $\equiv_{EF}^{can}$ has a finite index with respect to S.*

6 Complexity

In this section, we study the computational complexity of the following reasoning task:

TEMPEVO: Given an evolution frame $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$, a knowledge state s over $\mathcal{EC}$, and some formula φ, does $EF, s \models \varphi$ hold?

In order to obtain decidability results, we assume that the constituents of the evolution frame EF in TEMPEVO are all computable. More specifically, we assume that (i) $\mathcal{EC}$, $\mathcal{AC}$, and Bel are given as computable functions deciding $E \in \mathcal{EC}$, $a \in \mathcal{AC}$, and $r \in Bel(\boldsymbol{P})$, and (ii) Π and ρ are given as computable functions. Nonetheless, even under these stipulations, it is easy to see that TEMPEVO is undecidable.

The results of Section 5 provide a basis for characterizing some decidable cases. We consider here the following class of propositional evolution frames $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ (i.e., $\mathcal{A}$ is propositional). Call EF *regular*, if the following applies:

1. the membership tests $E \in \mathcal{EC}$ and $r \in Bel(\boldsymbol{P})$, as well as Π and ρ are computable in polynomial space (the latter with polynomial size output); e.g., the functions may be computable in the polynomial hierarchy;
2. rules in compilations $comp_{EF}(s)$ and events E have size polynomial in the representation size of EF, denoted by $\|EF\|$ (i.e., repetition of the same literal in a rule is bounded), and events have size at most polynomial in $\|EF\|$;

3. $Bel(\cdot)$ is model based, i.e., $Bel(\boldsymbol{P})$ is determined by a set of k-valued models, where k is small (typically, $k \leq 3$ as for $Bel_E(\cdot)$).

The conditions 1 and 3 apply to the approaches in [1, 6, 8, 16, 17, 13, 21], and condition 2 is reasonable to impose; note that none of these semantics is sensible to repetitions of literals in rule bodies.

Theorem 8. *Deciding $EF, s \models \varphi$, given a regular propositional evolution frame $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$, a knowledge state s, and a formula φ is*

1. 2-EXPSPACE*-complete, if $Bel(\cdot)$ is k-local for some k which is polynomial in $\|EF\|$, and $comp_{EF}(\cdot)$ is incremental;*
2. EXPSPACE*-complete, if EF is contracting and c-bounded, where c is polynomial in $\|EF\|$; and*
3. PSPACE*-complete, if EF is as in 2 and, moreover, all rules in the compilations $comp_{EF}(s')$ of successors s' of s are from a set R_0 of size polynomial in $\|EF\|$.*

For the upper bounds of these results, we note that in the case where φ has form $\mathsf{E}\psi$, only finite paths of length at most $|S/ \equiv_{EF}|$ must be considered for satisfying ψ, where S is the set of all future knowledge states of s. Part 1 of the theorem can then be shown by Theorem 3 using the estimation given in Theorem 2. Concerning Part 2, there are $\mathcal{O}(2^{l^{|\mathcal{A}|}+\|EF\|^m}) = \mathcal{O}(2^{2^{m'\|EF\|}})$ many knowledge states s that are not strongly equivalent, for some constants l, m and m'; each $Bel(s)$ can be represented, using canonical update programs, together with the last c events, in single exponential space. Furthermore, the representation of every successor state is computable in polynomial space in the input size. Hence, overall exponential space is sufficient. Finally, the additional condition in Part 3 of the theorem guarantees PSPACE complexity. The lower bounds can be shown by encoding Turing machine computations into particular evolution frames.

Part 3 of Theorem 8 implies that the propositional EPI framework has also PSPACE complexity. While here, in general, $Bel(s)$ depends on all events in s, it is possible to restrict $\mathcal{AC}_{\mathsf{EPI}}$ to the commands $\mathbf{assert}$ and $\mathbf{retract}$, by efficient coding techniques which store relevant history information in $Bel(s)$, such that the compilation in $comp_{\mathsf{EPI}}(s)$ depends only on $Bel(\pi_{n-1}(s))$ and the last event E_n in s. Furthermore, the policy Π_{EPI} is sensible only to polynomially many rules in events, and $comp_{\mathsf{EPI}}(s)$ contains only rules from a fixed set R_0 of rules, whose size is polynomial in the representation size of EF. Thus, we get the following corollary.

Corollary 2. *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}_{\mathsf{EPI}}, \Pi_{\mathsf{EPI}}, \rho_{\mathsf{EPI}}, Bel_E \rangle$ be a propositional* EPI *evolution frame, let s be a knowledge state, and let φ be a formula. Then, deciding $EF, s \models \varphi$ is in* PSPACE.

On the other hand, computations of a PSPACE Turing machine can be easily encoded in a propositional EPI evolution frame using a single event which models the clock. Thus, Corollary 2 has a matching lower bound.

We conclude our complexity analysis with results concerning weak, strong, and k-equivalence of two propositional update programs, respectively.

Theorem 9. *Deciding whether two given propositional update programs $\boldsymbol{P}$ and $\boldsymbol{Q}$ are weakly equivalent, i.e., satisfying $Bel_E(\boldsymbol{P}) = Bel_E(\boldsymbol{Q})$, is* coNP*-complete.*

Intuitively, the upper bound follows from the property that, for any propositional update programs $\boldsymbol{P}$ and $\boldsymbol{Q}$, $Bel(\boldsymbol{P}) = Bel(\boldsymbol{Q})$ is equivalent to $\mathcal{AS}(\boldsymbol{P}) = \mathcal{AS}(\boldsymbol{Q})$. The matching lower bound follows easily from the coNP-completeness of deciding whether an ELP has no answer set (cf. [5]).

For deciding 1-equivalence, the following lemma is useful:

Lemma 1. *Let $\boldsymbol{P}$ and $\boldsymbol{Q}$ be propositional update programs. Then, $\boldsymbol{P}$ and $\boldsymbol{Q}$ are not 1-equivalent under Bel_E iff there is an ELP P and a set S such that (i) $S \in \mathcal{AS}(\boldsymbol{P} + P)$ but $S \notin \mathcal{AS}(\boldsymbol{Q} + P)$, or vice versa, (ii) $|S|$ is at most the number of atoms in $\boldsymbol{P} + \boldsymbol{Q}$ plus 1, and (iii) $|P| \leq |S| + 1$. Furthermore, P has polynomial size in the size of $\boldsymbol{P}$ and $\boldsymbol{Q}$.*

Intuitively, this holds since any answer set S of $\boldsymbol{P} + P$ can be generated by at most $|S|$ many rules. Furthermore, if S is not an answer set of $\boldsymbol{Q} + P$, by unfolding rules in P we may disregard for an S all but at most one atom which does not occur in $\boldsymbol{P}$ or $\boldsymbol{Q}$. To generate a violation of S in $\boldsymbol{Q} + P$, an extra rule might be needed; this means that a P with $|P| \leq |S| + 1$ is sufficient.

Theorem 10. *Deciding strong equivalence (or k-equivalence, for any fixed $k \geq 0$) of two given propositional update programs $\boldsymbol{P}$ and $\boldsymbol{Q}$ is* coNP*-complete.*

Proof. (Sketch) For $k = 0$, the result is given by Theorem 9. For $k \geq 1$, the membership part follows from Lemma 1, in virtue of Theorem 4. Hardness can be shown by constructing, given a propositional DNF ϕ, suitable programs P and Q in polynomial time such that ϕ is valid in classical logic iff $\boldsymbol{P} = (P)$ and $\boldsymbol{Q} = (Q)$ are 1-equivalent. □

Note that Theorems 9, 10 and Lemma 1 make no finiteness assumption on the alphabet $\mathcal{A}$. They also hold for ground update programs $\boldsymbol{P}$ and $\boldsymbol{Q}$ in a first-order alphabet, where P in Lemma 1 is ground.

7 Discussion and Conclusion

We presented a general framework for reasoning about evolving logic programs, which can be applied to several approaches for updating logic programs in the literature. Since the semantics of evolution frames can be captured by Kripke structures, it is suggestive to transform reasoning problems on them into model checking problems [4]. However, in current model checking systems, state transitions must be stated in a polynomial-time language, and descriptions of these Kripke structures would require exponential space also for evolution frames with PSPACE complexity (e.g., EPI evolution frames). Thus, extensions of model checking systems would be needed for fruitful usability.

Lobo et al. introduced the $\mathcal{PDL}$ [14] language for policies, which contain event-condition-action rules and serve for modeling reactive behavior on observations from an environment. While similar in spirit, their model is different, and [14] focuses on detecting action conflicts (which, in our framework, is not an issue). In [15], reasoning tasks are considered which center around actions. Further related research is on planning, where certain reachability problems are PSPACE-complete (cf. [3]). Similar results were obtained in [20] for related agent design problems. However, in all these works, the problems considered are ad hoc, and no reasoning language is considered.

Fagin et al.'s [10] important work on knowledge in multi-agent systems addresses evolving knowledge, but mainly at an axiomatic level. Wooldridge's [19] logic for reasoning about multi-agent systems embeds CTL* and has belief, desire and intention modalities. The underlying model is very broad, and aims at agent communication and cooperation. It remains to see how our particular framework fits into these approaches.

Our ongoing work addresses these and further issues. Further meaningful properties of evolution frames would be interesting; e.g., iterativity of the compilation $comp_{EF}$, i.e., the events are incorporated one at a time, or properties of the belief operator Bel. Other issues are algorithms and fragments of lower (especially, polynomial) complexity.

References

1. J. Alferes, J. Leite, L. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic Updates of Non-Monotonic Knowledge Bases. *J. Logic Programming*, 45(1–3):43–70, 2000.
2. J. Alferes, L. Pereira, H. Przymusinska, and T. Przymusinski. LUPS - A Language for Updating Logic Programs. In *Proc. LPNMR'99*, LNAI 1730, pp. 162–176. Springer, 1999.
3. C. Baral, V. Kreinovich, and R. Trejo. Computational Complexity of Planning and Approximate Planning in the Presence of Incompleteness. *AIJ*, 122(1–2):241–267, 2000.
4. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
5. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. In *Proc. 12th IEEE International Conference on Computational Complexity* (*CCC '97*), pp. 82–101, 1997. Full paper *ACM Computing Surveys*, to appear.
6. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Considerations on Updates of Logic Programs. In *Proc. JELIA 2000*, LNAI 1919, pp. 2–20. Springer, 2000.
7. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On Properties of Update Sequences Based on Causal Rejection. *Theory and Practice of Logic Programming*, to appear.
8. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. A Framework for Declarative Update Specifications in Logic Programs. In *Proc. IJCAI'01*, pp. 649–654.
9. E. Emerson. Temporal and Modal Logics, Vol. B. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier, 1990.
10. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT, 1995.
11. D. Gabbay and P. Smets, editors. *Handbook on Defeasible Reasoning and Uncertainty Management Systems, Vol. III*. Kluwer Academic, 1998.
12. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
13. K. Inoue and C. Sakama. Updating Extended Logic Programs through Abduction. In *Proc. LPNMR'99*, LNAI 1730, pp. 147–161. Springer, 1999.
14. J. Lobo, R. Bhatia, and S. Naqvi. A Policy Description Language. In *Proc. AAAI/IAAI'99*, pp. 291–298. AAAI Press / MIT Press, 1999.
15. J. Lobo and T. Son. Reasoning about Policies Using Logic Programs. In *Proc. AAAI 2001 Spring Symposium on Answer Set Programming*, pp. 210–216, 2001.
16. V. Marek and M. Truszczyński. Revision Specifications by Means of Programs. In *Proc. JELIA'94*, LNAI 838, pp. 122–136. Springer, 1994.
17. V. Marek and M. Truszczyński. Revision Programming. *TCS*, 190(2):241–277, 1998.
18. M. Winslett. *Updating Logical Databases*. Cambridge University Press, 1990.
19. M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.
20. M. Wooldridge. The Computational Complexity of Agent Design Problem. In *Proc. International Conference on Multi-Agent Systems* (*ICMAS*) *2000*. IEEE Press, 2000.
21. Y. Zhang and N. Foo. Updating Logic Programs. In *Proc. ECAI'98*, pp. 403–407. 1998.