I N F S Y S
R E S E A R C H
R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

REASONING ABOUT EVOLVING NONMONOTONIC KNOWLEDGE BASES

Thomas EITER
Michael FINK
Giuliana SABBATINI
Hans TOMPITS

INFSYS RESEARCH REPORT 1843-02-11

SEPTEMBER 2002

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT
INFSYS RESEARCH REPORT 1843-02-11, SEPTEMBER 2002

REASONING ABOUT EVOLVING NONMONOTONIC KNOWLEDGE BASES

Thomas Eiter, Michael Fink Giuliana Sabbatini, and Hans Tompits [1]

Abstract. Recently, several approaches to updating knowledge bases modeled as extended logic programs 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 extended logic programs and maintained by an update policy. We first describe a formal model which captures various update approaches, and we define a logical language for expressing properties of evolving knowledge bases. We then investigate semantical and computational properties of our framework, where we focus on properties of knowledge states with respect to the canonical reasoning task of whether a given formula holds on a given evolving knowledge base. In particular, we present finitary characterizations of the evolution for certain classes of framework instances, which can be exploited for obtaining decidability results. In more detail, we characterize the complexity of reasoning for some meaningful classes of evolving knowledge bases, ranging from polynomial to double exponential space complexity.

[1]Institut für Informationssysteme, Abteilung Wissensbasierte Systeme 184/3, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria. E-mail: [eiter, michael, giuliana, tompits]@kr.tuwien.ac.at

Acknowledgements: This work was supported by the Austrian Science Fund (FWF) projects P13871-INF and N Z29-INF.

Copyright © 2002 by the authors

Contents

1 Introduction **1**

2 Preliminaries **3**
2.1 Update answer sets 4
2.2 Dynamic answer sets 4

3 Knowledge-Base Evolution **5**
3.1 Events and knowledge states 6
3.2 Evolution frame 6
3.3 Compilation and belief set 8
3.4 Examples 9

4 Capturing Frameworks for Knowledge Evolution **11**
4.1 Update Programs and EPI 12
4.2 Dynamic Logic Programs, LUPS, and LUPS* 13
4.3 Revision Programs 14
4.4 Abductive Theory Updates 14
4.5 Program Updates by means of PLPs 15
4.6 Further approaches 16

5 Reasoning About Knowledge-Base Evolution **17**

6 Knowledge-State Equivalence **19**
6.1 Local belief operators 21
6.2 Contracting belief operators 24
6.3 Canonical evolution frames 25

7 Complexity **27**
7.1 Complexity of state equivalence 36

8 Related Work and Conclusion **39**
8.1 Further work 40

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 (cf., e.g., [32, 19]), 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. The proposals range from basic methods to incorporate an update U, given by a set of rules, or a sequence of such updates $U_1, \ldots, U_n$, into a (nonmonotonic) logic program P [2, 18, 21, 12], to more general methods which use an *update policy* to specify, by means of update actions, how the updates $U_1, \ldots, U_n$ should be incorporated into the current state of knowledge [27, 3, 14]. Using these approaches, queries to the knowledge base, like "is a fact f true in P after updates $U_1, \ldots, U_n$?", 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 $U_1, \ldots, U_n$ 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 recently been studied in the agent community [34]. Other properties may involve more complex temporal relationships, which relate the truth of facts in the knowledge base over time. A simple example of this sort is the property that whenever the fact $message_to(tom)$, which intuitively means that a message should be sent to Tom, is true in KB at some point, then the fact $sent_message_to(tom)$, representing that a message has been sent to Tom, will be true in the evolving knowledge base at some point in the future.

Main problems addressed. In this paper, we aim at a framework for expressing reasoning problems over evolving knowledge bases, which are modeled as extended logic programs [20] and may be maintained by an update policy as mentioned 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 definitions.

Main results. The main contributions and results 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. 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 of an agent associated with the knowledge base. This structure comprises

- a logic programming semantics, Bel, for extended logic programs P, respectively for sequences $(P_1, P_2, \ldots, P_m)$ of extended logic programs P_i, over an alphabet $\mathcal{A}$;
- a nonempty class of *events*, which are sets of rules communicated to the agent; and
- an update frame $\langle \mathcal{AC}, \Pi, \rho \rangle$, consisting of a set of update actions $\mathcal{AC}$, an update policy Π, and a realization assignment ρ, which together specify how to incorporate *events*, which are sets of rules drawn from a class of possible events $\mathcal{EC}$ and communicated to the agent, into the knowledge base.

In our framework, a *knowledge state* $s = \langle KB; E_1, \ldots, E_n \rangle$ of the agent consists of an initial knowledge base KB, given by an extended logic program over the alphabet $\mathcal{A}$, and a sequence of events $E_1, \ldots, E_n$. Associated with the knowledge state s is the *belief set* $Bel(s)$ of the agent, which comprises all formulas which the agent beliefs to hold given its state of knowledge.

The agent reacts on an event by adapting its belief state 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 the realization assignment ρ, into a single logic program P, resp. a sequence of logic programs $(P_0, \ldots, P_n)$, denoted $comp_{EF}(s)$. The belief set $Bel(s)$ is then given by the belief set of the compiled knowledge state, and is obtained by applying the belief operator $Bel(\cdot)$ for (sequences of) logics programs to $comp_{EF}(s)$. Suitable choices of EF allow one to model different settings of logic program updates, such as [2, 27, 21, 12].

(2) We define the syntax and, based on evolution frames, the semantics of a logical language for reasoning about evolving knowledge bases, which employs linear and branching-time operators familiar from Computational Tree Logic (CTL) [16]. 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 maintenance problem from above can be expressed by a formula $\mathsf{AG}\, a$, and the avoidance problem by a formula $\mathsf{AG}{\sim}b$; the property about Tom's messages is expressed by

$$\mathsf{AG}(message_to(tom) \rightarrow \mathsf{AF} sent_message_to(tom)).$$

(3) We investigate semantical properties of knowledge states for reasoning. Since in principle a knowledge base may evolve forever, we are in particular concerned with obtaining finitary characterizations of evolution. To this end, we introduce various notions of equivalence between knowledge states, and show several filtration results: under certain properties of the components of EF, evolution of a knowledge state s in an evolution frame EF can be described by a finite transition graph $G^\star(s, EF)$, which is a subgraph bisimilar to the whole natural transition graph $G(s, EF)$ over knowledge states that includes an arc from $s_1 = \langle KB, E_1, \ldots, E_n \rangle$ to every immediate successor state $s_2 = \langle KB, E_1, \ldots, E_n, E_{n+1} \rangle$. In some cases, $G^\star(s, EF)$ is constructible by exploiting locality properties of the belief operator $Bel(\cdot)$ and increasing compilations $comp_{EF}(\cdot)$, while in others it results by canonization of the knowledge states.

In a concrete case study, we establish this for evolution frames which model policies in the EPI framework for logic program updates using the answer set semantics [14], as well as for the LUPS [3, 4] and LUPS* policies [22] under the dynamic stable model semantics [1, 2]. Similar results apply to updates under other semantics in the literature.

(4) We derive complexity results for reasoning. Namely, we analyze the problem of deciding, given an evolution frame EF, a knowledge state s, and a formula φ, whether $EF, s \models \varphi$ holds. While this problem is undecidable in general, we single out several cases in which the problem is decidable, adopting some general assumptions about the underlying evolution frame. In this way, we identify meaningful conditions under which the problem ranges from PSPACE up to 2-EXPSPACE complexity. We then apply this to the EPI framework under the answer set semantics [14, 15], and show that its propositional fragment has PSPACE-complexity. Similar results might be derived for the LUPS and LUPS* frameworks. We also consider the complexity of sequences of extended logic programs (ELPs) and generalized logic programs (GLPs), respectively. We show that deciding whether two sequences $\boldsymbol{P} = (P_0, \ldots, P_n)$ and $\boldsymbol{Q} = (Q_0, \ldots, Q_m)$ of

propositional ELPs are strongly equivalent under the update answer set semantics, i.e., whether for every sequence $\boldsymbol{R} = (R_0, \ldots, R_k)$, $k \geq 0$, the concatenated sequences $\boldsymbol{P} + \boldsymbol{R}$ and $\boldsymbol{Q} + \boldsymbol{R}$ have the same belief sets, is coNP-complete. This result is not immediate, since potentially infinitely many pairs $\boldsymbol{P} + \boldsymbol{R}$ and $\boldsymbol{Q} + \boldsymbol{R}$ need to be checked. Thus, testing strong equivalence between sequences of ELPs is not more expensive than standard inference of a literal from all answer sets of an ELP (cf. [11]). Analogous results hold for sequences of GLPs.

To the best of our knowledge, no similar effort to formally express reasoning about evolving nonmonotonic knowledge bases at a level as considered here has been put forth so far. 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. Note that J. Leite, in his recent Ph.D. thesis [23], considers properties of evolving logic programs in a language inspired by our EPI language [14, 15], and derives some properties for dynamic logic programs similar to properties for update programs derived in Section 7.

The rest of this paper is structured as follows. In the next section, we give some basic definitions and fix notation. In Section 3, we introduce our notion of an evolution frame, which is the basic setting for describing update formalisms, and in Section 4, we show how different approaches to updating logic programs can be captured by it. In Section 5, we then define the syntax and semantics of our logical language for reasoning about evolving knowledge bases. Section 6 is devoted to the study of equivalence relations over knowledge states, which are useful for filtration of the infinite transition graph that arises from an evolving knowledge base. In particular, conditions are investigated under which a restriction to a finite subgraph is feasible. After that, we address in Section 7 the complexity of reasoning. Related work is discussed in Section 8, where we also draw some conclusions and outline issues for further research.

2 Preliminaries

We consider knowledge bases represented as *extended logic programs* (ELPs) [20], 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 $\mathcal{A}$. For a literal L, the *complementary literal*, $\neg L$, is $\neg A$ if $L = A$, and A if $L = \neg A$, for some atom A. For a set S of literals, we define $\neg S = \{\neg L \mid L \in S\}$. We also denote by $Lit_{\mathcal{A}}$ the set $\mathcal{A} \cup \neg\mathcal{A}$ of all literals 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 define $B^+(r) = \{L_1, \ldots, L_m\}$ and $B^-(r) = \{L_{m+1}, \ldots, L_n\}$. We allow the case where L_0 is absent from r, providing $B(r) \neq \emptyset$; such a rule r is called a *constraint*. If $H(r) = \{L_0\}$ and $B(r) = \emptyset$, 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 [2], might be added to fit other frameworks.

An *update program*, $\boldsymbol{P}$, is a sequence $(P_0, \ldots, P_n)$ of ELPs ($n \geq 0$), representing the evolution of program P_0 in the light of new rules $P_1, \ldots, P_n$. We sometimes use $\cup\boldsymbol{P}$ to denote the set of all rules occurring in $\boldsymbol{P}$, i.e., $\cup\boldsymbol{P} = \bigcup_{i=1}^{n} P_i$. The semantics of update programs can abstractly be described in terms of a belief operator $Bel(\cdot)$, 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 [2, 18, 21, 12, 27].

2.1 Update answer sets

For concrete examples, we consider the answer set semantics for propositional update programs as introduced in [12, 13], as well as the semantics for dynamic logic programs as defined in [2, 23]. The former semantics defines answer sets of a sequence of ELPs, $\boldsymbol{P} = (P_0, \ldots, P_n)$, 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 $Rej(S, \boldsymbol{P}) = \bigcup_{i=0}^{n} Rej_i(S, \boldsymbol{P})$, where $Rej_n(S, \boldsymbol{P}) = \emptyset$, and, for $n > i \geq 0$, $Rej_i(S, \boldsymbol{P})$ 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 $\boldsymbol{P} = (P_0, \ldots, P_n)$ iff S is a consistent answer set [20] of the program $P = \bigcup_i P_i \setminus Rej(S, \boldsymbol{P})$, i.e., S is a minimal consistent set of literals closed under the rules of the *reduct* $P^S = \{H(r) \leftarrow B^+(r) \mid r \in P \text{ and } B^-(r) \cap S = \emptyset\}$. The set of all answer sets of $\boldsymbol{P}$ is denoted by $\mathcal{U}(\boldsymbol{P})$. This definition properly generalizes consistent answer sets from single ELPs to sequences of ELPs. Nevertheless, we use $\mathcal{AS}(P)$ to denote the set of all answer sets of a single ELP P. Moreover, an ELP P is called *inconsistent* if it has no consistent answer set, i.e., $\mathcal{AS}(P) = \emptyset$. Update answer sets for arbitrary (non-ground) update programs $\boldsymbol{P}$ are defined in terms of its ground instance similar as answer sets for ELPs [20].

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

The belief operator $Bel_E(\cdot)$ in the framework of [12] is given by $Bel_E(\boldsymbol{P}) = \{r \in \mathcal{L}_\mathcal{A} \mid S \models r \text{ for all } S \in \mathcal{U}(\boldsymbol{P})\}$, 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 $not\, L \in B(r')$.

2.2 Dynamic answer sets

By the term *dynamic answer sets* we refer to the extension of dynamic stable models, defined for sequences of *generalized logic programs* (*GLPs*) by Alferes *et al.* [2], to the three-valued case. In GLPs, default negation may appear in the head of rules, but strong negation is excluded. The definition of dynamic stable models uses a slightly non-standard concept of stable models, where weakly negated literals $not\, A$ (A some atom) are treated like ordinary propositional atoms, and rules $A_0 \leftarrow A_1, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n$ are viewed as *Horn clauses*. Accordingly, an interpretation I is in this context understood as a set of atoms and weakly negated atoms such that $A \in I$ iff $not\, A \notin I$ holds for each atom A. To distinguish it from a usual interpretation, we call it a *generalized interpretation*. As usual, a set B, comprising atoms and weakly negated atoms, is true in a generalized interpretation I, symbolically $I \models B$, iff $B \subseteq I$. Towards defining stable models, the following notation is required:

Let, for a set of atoms $\mathcal{A}$, $not\, \mathcal{A}$ stand for the set $\{not\, A \mid A \in \mathcal{A}\}$. Furthermore, for $M \subseteq \mathcal{A} \cup not\, \mathcal{A}$, we set $M^- = \{not\, A \mid not\, A \in M\}$, and, for $Z \in \mathcal{A} \cup not\, \mathcal{A}$, we define $not\, Z = not\, A$ if $Z = A$ and $not\, Z = A$ if $Z = not\, A$. For a program P over $\mathcal{A}$, the deductive closure, $Cn_\mathcal{A}(P)$, is given by the set

$\{L \mid L \in \mathcal{A} \cup not\, \mathcal{A}$ and $P \vdash L\}$, where P is interpreted as a propositional Horn theory and "$\vdash$" denotes classical derivability. A generalized interpretation S is a stable model of a program P iff $S = Cn_{\mathcal{A}}(P \cup S^{-})$.

Let $\boldsymbol{P} = (P_0, \ldots, P_n)$ be a sequence of GLPs over $\mathcal{A}$, and let I be a generalized interpretation. Alferes *et al.* [2] introduce the following concepts:

$$Rejected(I, \boldsymbol{P}) = \bigcup_{i=0}^{n} \{r \in P_i \mid \exists r' \in P_j, \text{ for some } j \in \{i+1, \ldots, n\}, \text{ such that } H(r') = not\, H(r) \text{ and } I \models B(r) \cup B(r')\};$$

$$Defaults(I, \boldsymbol{P}) = \{not\, A \mid \nexists r \text{ in } \boldsymbol{P} \text{ such that } H(r) = A \text{ and } I \models B(r)\}.$$

A set $S \subseteq \mathcal{A} \cup not\, \mathcal{A}$ is a *dynamic stable model* of $\boldsymbol{P}$ iff

$$S = Cn((\cup \boldsymbol{P} \setminus Rejected(S, \boldsymbol{P})) \cup Defaults(S, \boldsymbol{P})).$$

We remark that Alferes *et al.* defined dynamic stable models of $\boldsymbol{P}$ as projections $S = S' \cap (\mathcal{A} \cup not\, \mathcal{A})$ of the stable models of a single GLP, $\boldsymbol{P}_{\oplus}$, resulting from a transformation (for a detailed definition cf. [2]), and then proved the above characterization as a result.

Alferes *et al.* [2] defined also an extension of their semantics to the three-valued case: Let $\boldsymbol{P} = (P_0, \ldots, P_n)$ be a sequence of ELPs over $\mathcal{A}$. Then, a consistent set $S \subseteq Lit_{\mathcal{A}}$ is a *dynamic answer set* of $\boldsymbol{P}$ iff $S \cup \{not\, L \mid L \in Lit_{\mathcal{A}} \setminus S\}$ is a dynamic stable model of the sequence $\boldsymbol{P} = (P_0, \ldots, P_n \cup \{not\, A \leftarrow \neg A,\ not\, \neg A \leftarrow A \mid A \in \mathcal{A}\})$ of GLPs. Here, the rules in $\{not\, A \leftarrow \neg A,\ not\, \neg A \leftarrow A \mid A \in \mathcal{A}\}$ serve for emulating classical negation through weak negation.

Example 2 Let, as in the previous example, $P_0 = \{b \leftarrow not\, a,\ a \leftarrow\}$, $P_1 = \{\neg a \leftarrow,\ c \leftarrow\}$, and $P_2 = \{\neg c \leftarrow\}$ over $\mathcal{A} = \{a, b, c\}$. Then, P_0 has the single dynamic answer set $S_0 = \{a, not\, b, not\, c\}$, where $Rejected(S_0, P_0) = \emptyset$ and $Defaults(S_0, P_0) = \{not\, b, not\, c\}$; the sequence (P_0, P_1) has the dynamic answer set $S_1 = \{\neg a, c, b\}$, where $Rejected(S_1, (P_0, P_1)) = \{a \leftarrow\}$ and $Defaults(S_1, (P_0, P_1)) = \{not\, \neg b,\ not\, \neg c\}$; and (P_0, P_1, P_2) has the single dynamic answer set $S_2 = \{\neg a, \neg c, b\}$, where $Rejected(S_2, (P_0, P_1, P_2)) = \{a \leftarrow,\ c \leftarrow\}$ and $Defaults(S_2, (P_0, P_1, P_2)) = \{not\, \neg b\}$. Note that in these simple examples, update answer sets and dynamic answer sets coincide, which is not the case in general [13]. □

Similarly to the belief operator $Bel_E(\cdot)$, we can define a belief operator $Bel_{\oplus}(\cdot)$ for dynamic stable models as $Bel_{\oplus}(\boldsymbol{P}) = \{r \in \mathcal{L}_{\mathcal{A}} \mid S \models r \text{ for all } S \in \mathcal{D}(\boldsymbol{P})\}$, where $\mathcal{D}(\boldsymbol{P})$ denotes the set of all dynamic stable models of $\boldsymbol{P}$.

Finally, we remark that while we defined answer sets and belief sets for sequences of finite programs, they can be defined for sequences of possibly infinite programs in an analogous way.

3 Knowledge-Base Evolution

We assume that the agent has an initial knowledge base, KB, in form of an extended logic program, and a background *update policy*, Π, describing the update behavior of the agent, i.e., how it has to react when it receives new information from the environment. Information arrives to the agent in form of a sequence of *events*, each event being a finite set of rules from a given *event class*. The update policy specifies what rules or facts have to be incorporated into or retracted from the knowledge base, depending on the content of the event and on the belief set of the agent. The evolution of the agent's *knowledge state* is thus completely described when KB and a sequence of events $E_1, \ldots, E_n$ are given, provided an update policy Π is specified.

3.1 Events and knowledge states

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; a plain case is that an event just consists of a set of facts, which are formed over a subset of the alphabet. In a deductive database setting, the latter case corresponds to an extensional database that is undergoing change while the intensional part of the database remains fixed.

Definition 2 *Let $\mathcal{EC}$ 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. The set of all knowledge states over $\mathcal{A}$ given $\mathcal{EC}$ is denoted by $s(\mathcal{EC})$.*

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 the new knowledge state $s' = \langle KB; E_1, \ldots, E_n, E_{n+1} \rangle$, and the agent should adapt its belief set in accordance with the new event obeying its given update policy.

3.2 Evolution frame

The "universe" in which the evolution of an agent's knowledge base takes place is given by the concept of an *evolution frame*, which comprises different components that parameterize the update mechanism and the semantics used on the evolving knowledge base. This structure comprises, together with an alphabet $\mathcal{A}$,

- a semantics, $Bel(\cdot)$, for ELPs, resp. sequences of ELPs, over $\mathcal{A}$;
- a nonempty event class $\mathcal{EC}$ over $\mathcal{A}$; and,
- an *update frame* $\langle \mathcal{AC}, \Pi, \rho \rangle$, consisting of a set of *update commands* $\mathcal{AC}$, an *update policy* Π, and a *realization assignment* ρ, which together specify how to incorporate events into the knowledge base.

In more detail, the components of an update frame are as follows.

Update commands. The update commands (or *actions*) in $\mathcal{AC}$ are names for commands which are supposed to be executed on the knowledge base. 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., [3, 14]). However, several update frameworks can be modeled using these simple commands. The semantics (i.e., effects) of update actions are given by the realization assignment, ρ, which is described below.

Update policy. The update policy Π, which is a function mapping every pair (s, E) of a knowledge state s over $\mathcal{EC}$ (i.e., $s \in s(\mathcal{EC})$) and an event $E \in \mathcal{EC}$ into a set $\Pi(s, E) \subseteq \mathcal{AC}$ of update commands, determines *which* actions should be executed. Update policies 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.

Realization assignment. The realization assignment ρ assigns to each pair (s, A) of a knowledge state s over $\mathcal{EC}$ and a set $A \subseteq \mathcal{AC}$ of update commands a sequence $\rho(s, A) = (P_0, \ldots, P_n)$ of ELPs P_i over $\mathcal{A}$ $(0 \leq i \leq n)$. It associates in this way a meaning with the set of actions A which must be executed on the knowledge state s, in terms of an ELP, resp. a sequence of ELPs, and "realizes" the update in this way. The agent's beliefs from the updated knowledge base may then be given by the operator $Bel(\cdot)$ applied to the result of $\rho(s, A)$ as defined in Section 3.3 below.

Different possibilities for concrete realization assignments ρ may be used. A simple realization assignment, $\rho_{\pm}(s, A)$, which works for sets A of actions of form $insert(r)$ and $delete(r)$, and assumes that each knowledge state s is assigned with an ordinary ELP $P(s)$, is given by

$$\rho_{\pm}(s, A) = (P(s) \cup \{r \mid insert(r) \in A\}) \setminus \{r \mid delete(r) \in A\},$$

i.e., the insertion and deletion commands in A are "physically" implemented, with no further enforcement that consistency is preserved, or, as for deletion, that r is actually logically deleted from the knowledge base. Its restriction to insertion commands is the realization assignment $\rho_{ins} = (s, A) = P(s) \cup \{r \mid insert(r) \in A\}$, which may be used in contexts where data are not physically removed, for whatever reasons.

More sophisticated realization assignments might block, at the logical level, the applicability of rules in the knowledge base, by using a sequence $(P_0, \ldots, P_n)$ of ELPs as a representation, and aim at enforcing consistency of the knowledge base. For instance, in the dynamic logic programming semantics of sequences of ELPs in [2, 14], more recent rules occur later in a sequence and override rules from programs which occur earlier in the sequence; this mechanism is also used in the EPI framework for incorporating changes to the knowledge base at the logical level [14, 15].

Summarizing, we formally define an evolution frame as follows. Let, for any alphabet $\mathcal{A}$, denote $ELP^*(\mathcal{A})$ the set of all sequences $\boldsymbol{P} = (P_0, \ldots, P_n)$, $n \geq 0$, of ELPs P_i over $\mathcal{A}$.

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

- $\mathcal{A}$ *is a finite (first-order) alphabet;*
- $\mathcal{EC}$ *is a nonempty event class over* $\mathcal{A}$*;*
- $\mathcal{AC}$ *is a set of update commands (or actions);*
- $\Pi : s(\mathcal{EC}) \times \mathcal{EC} \to 2^{\mathcal{AC}}$ *is an update policy;*
- $\rho : s(\mathcal{EC}) \times 2^{\mathcal{AC}} \to ELP^*(\mathcal{A})$ *is a realization assignment; and*
- $Bel : ELP^*(\mathcal{A}) \to 2^{\mathcal{L}_{\mathcal{A}}}$ *is a belief operator for sequences of ELPs.*

The set of all knowledge states in EF, denoted by $\mathcal{S}_{EF}$*, is given by* $s(\mathcal{EC})$*.*

The concept of an evolution frame allows us to model various update approaches, as we discuss below in Section 4.

3.3 Compilation and belief set

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, in general, a sequence of logic programs $\boldsymbol{P}$. 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 $0 \leq i \leq n$. In particular, $\pi_0(s)$ is the initial knowledge base KB. We call $\pi_i(s)$ a *previous knowledge state* (or simply an *ancestor*) of s if $i < n$. Dually, a 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 is used for 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*

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

Note that $comp_{EF}(\cdot)$ is a function which is fully determined by EF; we often write $comp(\cdot)$ instead of $comp_{EF}(\cdot)$ if EF is understood.

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. Furthermore, Π may be restricted in the compilation process, e.g., such that only the belief set $Bel(\pi_{n-1}(s))$ of the predecessor state is considered rather than the whole state itself; this will be considered in Section 6.2.

Incremental Compilation. An important class of compilations are those in which, for a future knowledge state s', $comp(s')$ 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 definition amounts to the expected behavior:

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, Q' are logic programs and "+" is the concatenation of sequences.*

Proof. The proof proceeds by straightforward induction on $|s|$. □

Example 3 A simple incremental compilation results for $\mathcal{AC}_{ins} = \{insert(r) \mid r \in \mathcal{L}_{\mathcal{A}}\}$, $\Pi = \Pi_{ins}$ as defined in Subsection 3.2, and ρ_{ins} such that $comp_{EF}(\langle KB \rangle) = KB$ and $comp_{EF}(s) = comp_{EF}(\pi_{|s|-1}(s)) + (\{r \mid insert(r) \in A\})$, where $A = \Pi_{ins}(\pi_{|s|-1}(s), E_n)$, given that $s = \langle KB; E_1, \ldots, E_n \rangle$. Note that $comp_{EF}(s)$ is in this setting just the sequence $(KB, E_1, \ldots, E_n)$. □

While incremental compilations are natural, we stress that other compilations are of course also highly relevant. In particular, the compilation might perform optimizations (cf. Section 6.2), or output only an ordinary logic program.

We also point out that our notion of incremental compilation should not be confused with an *iterative compilation*; such a compilation would, similar in spirit, consider the events E_i in a knowledge state $s = \langle KB, E_1, \ldots, E_n \rangle$ in their chronological order one by one and instantaneously incorporate updates $A_i = \Pi(\pi_{i-1}(s), E_i)$ into the result $comp_{EF}(\pi_{i-1}(s))$ for the previous knowledge state and return a single, ordinary logic program as the result.

The compilation of a knowledge state into a (sequence of) ELPs is used, via the semantics $Bel(\cdot)$ for sequences of ELPs, to ascribe a set of beliefs to the agent in the respective knowledge state. More formally, 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 $Bel(comp_{EF}(s))$.*

This completes the exposition of evolution frames and their semantics. Before we consider some examples, let us close this subsection with some remarks.

Remarks. (1) As mentioned earlier, our definition of an update policy, and similarly of a realization assignment, which effectively lead to the notion of a compilation, is very general. We may stipulate additional postulates upon them, like the incrementability property or an iterativity property (which we omit here). Likewise, the concept of a semantics $Bel(\boldsymbol{P})$ for sequences $\boldsymbol{P}$ of ELPs is very abstract, and further axioms and conditions could be imposed on it. An example of this is the requirement that $Bel(\boldsymbol{P})$ is characterized by rules of bounded length, and in particular by rules without repeated literals; this will be the case in Section 7.

(2) Our definition does not capture nondeterministic update policies, where $\Pi(s, E)$ may return one out of several possible sets of update actions. In order to model this, 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.

3.4 Examples

Let us illustrate our framework on two examples, which serve as running examples throughout the remainder of the paper.

Example 4 (Shopping Agent) Consider a shopping agent selecting Web shops in search for some specific

merchandise. Suppose its knowledge base, KB, contains the rules

$$\begin{aligned} r_1: &\quad query(S) \leftarrow sale(S), up(S), not\ \neg query(S); \\ r_2: &\quad site_queried \leftarrow query(S); \\ r_3: &\quad notify \leftarrow not\ site_queried; \end{aligned}$$

and a fact $r_0 : date(0)$ as an initial time stamp. Here, r_1 expresses that a shop S, which has a sale and whose Web site is up, is queried by default, and r_2, r_3 serve to detect that no site is queried, which causes '$notify$' to be true.

Assume that an event, E, might consist of one or more of the following items:

- at most one fact $date(t)$, for some date t;
- facts $up(s)$ or $\neg up(s)$, stating that a shop s is up or down, respectively;
- ground rules of form $sale(s) \leftarrow date(t)$, stating that shop s has a sale on date t.

An update policy $\mathcal{U}$ may be defined as follows:

$$\begin{aligned} \Pi(s,E) = &\{insert(\alpha) \mid \alpha \in \{up(S), \neg up(S), date(T)\}, \alpha \in E\} \cup \\ &\{insert(sale(S) \leftarrow date(T)),\ insert(track(S,T)) \mid \\ &\quad sale(S) \leftarrow date(T) \in E, date(T') \in Bel(s), T \geq T'\} \cup \\ &\{delete(track(S,T)),\ delete(sale(S) \leftarrow date(T)) \mid \\ &\quad date(T') \in E, track(S,T) \in Bel(s), date(T) \in Bel(s), T' \neq T\} \cup \\ &\{delete(date(T)) \mid date(T') \in E, date(T) \in Bel(s), T' \neq T\}\}. \end{aligned}$$

Informally, this update policy incorporates information about future sales, only. The information of the sale is removed, when the sale ends (assuming the time stamps increase). To this end, facts $track(S,T)$ are used to keep track of inserted sale information. Similarly, the current time stamp $date(t)$ is maintained by deleting the old values. The realization assignment ρ might be $\rho_{\pm}$ from Subsection 3.2, which always returns a single ELP, and for Bel we may take any function which coincides on sequences $\boldsymbol{P} = P_0$ of length one with the standard answer set semantics for ELPs. Or, we might choose a realization assignment which maps s and a set of $insert(r)$ and $delete(r)$ commands to a sequence $(P_0, \ldots, P_n)$ of ELPs, using as Bel the answer set semantics for sequences of ELPs as discussed in Subsection 3.2. □

Example 5 (Mail Agent) Consider a more complex mail agent, which has the following initial knowledge base KB, whose rules are instantiated over suitable variable domains:

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

The knowledge base allows to express several attributes of a message and determine the *type* of a message based on these attributes (rules r_1 and r_2). By means of r_3, a default type is assigned to all messages which are neither $private$ nor $business$. Rule r_4 implicitly states that a $save$ operation is stronger than a $remove$ one. Note that in this way, once a message has been saved, it can never be removed. By means of r_5, all

those messages are removed which have not been saved and are older than thirty days. Rules r_6, r_7 and r_8 are used to look for all messages of a given type, which have not been sent to the trash yet, and to signal if at least one such message has been found ($success$) or not ($failure$).

Suppose that an event E may consist in this scenario 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:

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

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. Like in the previous example, the realization assignment ρ may be given by $\rho_{\pm}$ from Subsection 3.2, or could map s and A incrementally to a sequence of ELPs using as Bel simply the answer set semantics for sequences of ELPs. □

4 Capturing Frameworks for Knowledge Evolution

To emphasize the generality of our framework, we now discuss how existing frameworks for updating non-monotonic 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 [12, 13], dynamic logic programs [2], revision programs [26, 27], abductive theory updates [21], and updates through prioritized logic programs (PLPs) [18] fall into this class.

(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 [14], LUPS and LUPS* [3, 4, 22], KABUL [23], and, while not directly given in these terms, $\mathcal{PDL}$ [25].

In what follows, we show how some of the above mentioned frameworks can be expressed in evolution frames, which shows the generality of the approach. We start capturing the formalisms at the update level introduced in Section 2, i.e., the answer set semantics for update programs, represented by $Bel_E(\cdot)$, and the dynamic stable model semantics for generalized logic programs, represented by $Bel_\oplus(\cdot)$. For both semantics, we also show how they are combined with convenient specification languages to form higher level frameworks: $Bel_\oplus(\cdot)$ is combined with the language LUPS [3, 4], which allows for more flexibility of the update process, permitting to dynamically specify the contents of a sequence of updates by means of update commands; and the semantics $Bel_E(\cdot)$ is employed together with the language EPI [14], which is more expressive than LUPS. It allows for update statements to depend on other update statements in the same EPI policy, and more complex conditions on both the current belief set and the actual event can be specified. Further frameworks and semantics are also discussed here, albeit more briefly and stressing only the main characterizations. We repeatedly use the particular set $\mathcal{AC}_{ins}$ of insert commands, the insert policy Π_{ins}, and the insert realizations ρ_{ins} and $\rho_\pm$ from Subsection 3.2.

4.1 Update Programs and EPI

Update programs [12, 13] 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 $Bel_E(\cdot)$ is the belief operator defined in Section 2. The EPI framework [14, 15] 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

- $\mathcal{AC}_{\mathsf{EPI}} = \{\mathbf{assert}(r), \mathbf{retract}(r), \mathbf{always}(r), \mathbf{cancel}(r), \mathbf{ignore}(r),$
  $\mathbf{assert_event}(r), \mathbf{retract_event}(r), \mathbf{always_event}(r) \mid r \in \mathcal{L}_\mathcal{A}\},$
  and the commands have the meaning as in [14];
- Π_{EPI} is defined by any set of update statements in the language EPI, which are evaluated through a logic program as defined in [14];
- ρ_{EPI} realizes the translation $tr(KB; U_1, \ldots, U_n)$ from [14], 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 $comp_{\mathsf{EPI}}(\cdot)$ is incremental.

Observe that, while $tr(\cdot)$ as in [14] is involved and has to keep track of persistent update commands $\mathbf{always}[_\mathbf{event}](r)$ from the past, as shown in [15], it is possible, by encoding persistent commands in polynomial time in the belief set, to restrict actions, without loss of expressiveness, to the commands $\mathbf{assert}$ and $\mathbf{retract}$ (whose meaning is the intuitive one) and making ρ actually depend only the belief set $Bel(\pi_{n-1}(s))$ of the predecessor and the event E_n.

4.2 Dynamic Logic Programs, LUPS, and LUPS*

Dynamic logic programming [1, 2] can be captured by the following evolution frame:

$$EF_{\oplus} = \langle \mathcal{A}, \mathcal{EC}_{gp}, \mathcal{AC}_{ins}, \Pi_{ins}, \rho_{ins}, Bel_{\oplus} \rangle,$$

where $\mathcal{EC}_{gp}$ is the collection of all finite sets of generalized logic program rules, i.e., no strong negation is available and weak negation can occur in the head of rules, and $Bel_{\oplus}(\cdot)$ is the semantics of dynamic logic programs as given in Section 2.

The LUPS framework [3] for update specifications corresponds to the following evolution frame:

$$EF_{L} = \langle \mathcal{A}, \mathcal{EC}_{L}, \mathcal{AC}_{L}, \Pi_{L}, \rho_{L}, Bel_{\oplus} \rangle,$$

where

- $\mathcal{EC}_{L}$ is the collection of all finite sets of LUPS statements (cf. [3]);
- $\mathcal{AC}_{L} = \{\mathbf{assert}(r), \mathbf{retract}(r), \mathbf{always}(r), \mathbf{cancel}(r),$
  $\mathbf{assert_event}(r), \mathbf{retract_event}(r), \mathbf{always_event}(r) \mid r \in \mathcal{L}_{\mathcal{A}}\},$
  where the commands have the meaning explained in [3];
- Π_{L} is defined by $\Pi_{L}(s, E) = \{cmd(r) \in \mathcal{AC}_{L} \mid E$ contains $cmd(r)$ *when* $cond$ and $cond \in Bel_{\oplus}(s)\}$;
- ρ_{L} is as described in [3]; that is, $\rho_{L}(s, A)$ adds for $s = \langle KB, E_1, \ldots, E_n \rangle$ and A a program P_{n+1} to the sequence of programs $(P_0, \ldots, P_n)$ associated with s, returning $(P_0, \ldots, P_{n+1})$, where P_{n+1} is computed from the persistent commands PC_n valid at state s, $Bel(s)$, and the LUPS commands in A.

In [22], the semantics of LUPS has been slightly modified and extended by a permanent retraction command. The resulting framework, LUPS*, can be captured by the following evolution frame:

$$EF_{L^*} = \langle \mathcal{A}, \mathcal{EC}_{L^*}, \mathcal{AC}_{L^*}, \Pi_{L^*}, \rho_{L^*}, Bel_{\oplus} \rangle,$$

where

- $\mathcal{EC}_{L^*}$ is the collection of all finite sets of LUPS* statements (cf. [22]);
- $\mathcal{AC}_{L^*} = \{\mathbf{assert}(r), \mathbf{retract}(r), \mathbf{assert_event}(r), \mathbf{retract_event}(r),$
  $\mathbf{always\ assert}(r), \mathbf{always\ retract}(r), \mathbf{always\ assert_event}(r),$
  $\mathbf{always\ retract_event}(r), \mathbf{cancel\ assert}(r), \mathbf{cancel\ retract}(r) \mid r \in \mathcal{L}_{\mathcal{A}}\},$
  where the commands have the meaning as described in [22];
- Π_{L^*} is defined by $\Pi_{L^*}(s, E) = \{cmd(r) \in \mathcal{AC}_{L^*} \mid E$ contains $cmd(r)$ *when* $cond$ and $cond \in Bel_{\oplus}(s)\}$;
- ρ_{L^*} is given as in [22]; like before, $\rho_{L^*}(s, A)$ adds a program P_{n+1} to the sequence of programs $(P_0, \ldots, P_n)$ associated with s, where P_{n+1} is computed from persistent commands PC_n^* valid at state s, $Bel(s)$, and the LUPS* commands in A.

Like in the case of EPI, the compilation functions $comp_{L}(\cdot)$ and $comp_{L^*}(\cdot)$ are incremental, and also persistent commands ($\mathbf{always}(r)$, $\mathbf{always_event}(r)$, $\mathbf{always\ assert}(r)$, $\mathbf{always\ assert_event}(r)$, $\mathbf{always\ retract}(r)$, and $\mathbf{always\ retract_event}(r)$, respectively) can be eliminated through coding into the knowledge base.

4.3 Revision Programs

In [26, 27], a language for revision specification of knowledge bases is presented, which is based on logic programming under the stable model semantics. A knowledge base is in this context a set of atomic facts, i.e., a plain relational database. Revision rules describe which elements are to be present (so-called *in*-rules) or absent (*out*-rules) from the knowledge base, possibly under some conditions. A fixed-point operator, which satisfies some minimality conditions, is introduced to compute the result of a revision program. As for stable models, there may be several knowledge bases or no knowledge base satisfying a given revision program.

The framework of revision programs can be captured by the following evolution frame:

$$EF_{Rev} = \langle \mathcal{A}, \mathcal{EC}_{Rev}, \mathcal{AC}_{Rev}, \Pi_{Rev}, \rho_{Rev}, Bel_{Rev} \rangle,$$

where

- $\mathcal{EC}_{Rev}$ is the collection of finite sets of *revision* rules, i.e., negation-free rules whose constituents are of the form $in(B)$ or $out(B)$, where B is an atom from $\mathcal{A}$;
- $\mathcal{AC}_{Rev} = \{insert(B), delete(B) \mid B \in \mathcal{A}\}$;
- Π_{Rev} is defined by $\Pi_{Rev}(s, E) = \{insert(B) \mid B \in I\} \cup \{delete(B) \mid B \in O\}$, where (I, O) is the *necessary change* (cf. [26]) for $comp(s)$ with respect to E;
- ρ_{Rev} is defined by $\rho_{Rev}(s, \emptyset) = KB$ if $s = \langle KB \rangle$, and

  $$\rho_{Rev}(s, A) = (comp(s) \cup \{B \mid insert(B) \in A\}) \setminus \{B \mid delete(B) \in A\}$$

  otherwise, i.e., $\rho_{Rev}(s, A)$ corresponds to $\rho_{\pm}(s, A)$ where $P(s) = comp(s)$. Notice that $\rho_{Rev}(s, A)$, and in particular $comp(s)$, where $s = \langle KB; E_1, \ldots, E_n \rangle$, is thus a set of facts.
- Bel_{Rev} is such that, for each ELP P, it returns the collection of facts in P.

4.4 Abductive Theory Updates

Inoue and Sakama [21] developed an approach to theory update which focuses on nonmonotonic theories. They introduced an extended form of abduction and a framework for modeling and characterizing nonmonotonic theory change through abduction. Intuitively, this is achieved by extending an ordinary abductive framework by introducing the notions of negative explanation and *anti-explanation* (which makes an observation invalid by adding hypotheses), and then defining autoepistemic updates by means of this framework.

The framework of extended abduction is then used in [21] to model updates of nonmonotonic theories which are represented by ELPs. For theory updates, the whole knowledge base is subject to change. New information in form of an update program has to be added to the knowledge base and, if conflicts arise, higher priority is given to the new knowledge. The updated knowledge base is defined as the union $Q \cup U$ of the new information U and a maximal subset $Q \subseteq P$ of the original program that is consistent with the new information (which is always assumed to be consistent). The abductive framework is in this context used for specifying priorities between current and new knowledge, by choosing as abducibles the difference

between the initial and the new logic program. The framework for updates by means of abduction can be captured by the following evolution frame:

$$EF_{Abd} = \langle \mathcal{A}, \mathcal{EC}_{\mathcal{A}}, \mathcal{AC}_{Abd}, \Pi_{Abd}, \rho_{Abd}, Bel_{Abd} \rangle,$$

where

- $\mathcal{AC}_{Abd} = \mathcal{AC}_{ins} \cup \mathcal{AC}_{del}$, where $\mathcal{AC}_{del} = \{delete(r) \mid r \in \mathcal{L}_{\mathcal{A}}\}$;
- Π_{Abd} is defined by

  $$\Pi_{Abd}(s, E) = \{insert(r) \mid r \in E\} \cup \{delete(r) \mid r \in F \subseteq comp(s) \setminus E\},$$

  where F is, as defined in [21], a maximal set of rules to be removed from the current knowledge base $comp(s)$, which is a single logic program. Note that, in general, F may not be unique. Hence, for a deterministic update policy, we assume a suitable *selection function* σ which chooses one of the possible outcomes for F.
- ρ_{Abd} is defined by $\rho_{Abd}(s, \emptyset) = KB$ if $s = \langle KB \rangle$, and

  $$\rho_{Abd}(s, A) = (comp(s) \setminus \{r \mid delete(r) \in A\}) \cup \{r \mid insert(r) \in A\}$$

  otherwise, i.e., ρ_{Abd} amounts to $\rho_{\pm}(s, A)$ for $P(s) = comp(s)$, provided that A does not contain conflicting commands $delete(r)$ and $insert(r)$ for any rule r.
- Bel_{Abd} is such that, for any ELP P, it is the ordinary answer set semantics of ELPs.

4.5 Program Updates by means of PLPs

In [18], the update of a knowledge base of ground literals by means of a *prioritized logic program* (PLP) is addressed. The idea in updating the initial program, P, with respect to the new one, Q, is to first eliminate contradictory rules from P with respect to Q, and then to solve conflicts between the remaining rules by means of a suitable PLP. The semantics of the update is thus given by the semantics of the corresponding PLP, for which Zhang and Foo use the one they have proposed earlier in [35], which extends the answer set semantics. The method is to reduce PLPs to ELPs by progressively deleting rules that, due to the defined priority relation, are to be ignored. The answer sets of the resulting ELP are the intended answer sets of the initial PLP. Formulated initially for static priorities, the method was extended to dynamic priorities, which are handled by a transformation into a corresponding (static) PLP.

The framework for updates by means of PLPs is defined only for single step updates, and a generalization to multiple steps is not immediate. We can model a single-step update by the following evolution frame:

$$EF_{PLP} = \langle \mathcal{A}, \mathcal{EC}_{\mathcal{A}}, \mathcal{AC}_{ins} \cup \mathcal{AC}_{del}, \Pi_{PLP}, \rho_{PLP}, Bel_{PLP} \rangle,$$

where

- Π_{PLP} is defined by $\Pi_{PLP}(s, E) = \{insert(r) \mid r \in E\} \cup \{delete(r) \mid r \in R(s, E)\}$ where $R(s, E)$ is computed, along the procedure in [18], as a set of rules to be retracted from the current knowledge base;

- ρ_{PLP} is defined by $\rho_{PLP}(s, \emptyset) = KB$ if $s = \langle KB \rangle$, and $\rho_{PLP}(s, A) = (P_1, P_2)$ otherwise, where $P_1 = comp(s) \setminus \{r \mid delete(r) \in A\}$ and $P_2 = \{r \mid insert(r) \in A\}$;

- Bel_{PLP} is the semantics for prioritized logic programs [35], viewing (P_1, P_2) as a program where the rules of P_2 have higher priority than the ones in P_1.

Thus, several well-known approaches to updating logic programs can be modeled by evolution frames.

4.6 Further approaches

We remark that further approaches, though not concerned with logic programs, might be similarly captured. For example, to some extent, Brewka's declarative revision strategies [8] can be captured. Brewka introduced a nonmonotonic framework for belief revision which allows reasoning about the reliability of information, based on meta-knowledge expressed in the object language itself. In this language, revision strategies can be declaratively specified as well. The idea is to revise nonmonotonic theories by adding new information to the current theory, and to use an appropriate nonmonotonic inference relation to compute the accepted conclusions of the new theory.

The desired result is achieved in two steps. The first step consists in an extension of default systems in order to express preference information in the object language, together with an appropriate new definition of theory extensions. In a second step, a notion of prioritized inference is introduced, formalized as the least fixed-point of a monotone operator, thus identifying epistemic states with preferential default theories under this ad hoc semantics. The approach can be captured by a suitable evolution frame

$$EF_{\mathcal{T}} = \langle \mathcal{A}, \mathcal{EC}_{\mathcal{T}}, \mathcal{AC}_{\mathcal{T}}, \Pi_{\mathcal{T}}, \rho_{\mathcal{T}}, Bel_{\mathcal{T}} \rangle,$$

which naturally models the insertion of formulas into a preference default theory, i.e.,

- $\mathcal{EC}_{\mathcal{T}}$ is the set of all propositional formulas of the language;

- $\mathcal{AC}_{\mathcal{T}} = \{insert(f) \mid f \in \mathcal{EC}_{\mathcal{T}}\}$;

- $\Pi_{\mathcal{T}}$ is implicitly encoded in the current knowledge state (i.e., the current preference default theory, cf. [8]), and is such that $\Pi_{\mathcal{T}}(s, E) = \Pi_{ins}(s, E) = \{insert(f) \mid f \in E\}$;

- $\rho_{\mathcal{T}}$ produces the new preference default theory by simply executing the insertion of the new formula(s) into it, i.e., $\rho_{\mathcal{T}}(s, A) = \rho_{ins}(s, A) = \mathcal{T}(s) \cup \{f \mid insert(f) \in A\}$, where $\mathcal{T}(s) = comp(s)$ and $\rho_{\mathcal{T}}(s, \emptyset) = KB$ if $s = \langle KB \rangle$, as usual; and

- $Bel_{\mathcal{T}}$ is the function assigning to each preference default theory its set of accepted conclusions, as defined in [8].

However, there also exist frameworks which take a different point of view and cannot be captured by our definition of an evolution frame. This is the case if the state of an agent and the environment, and thus the action taken by an agent, are dependent on the whole history of events and actions taken (which is also known as a *run*), not only on the current state. The approach of Wooldridge [34] is an example of such a framework.

5 Reasoning About Knowledge-Base Evolution

We now introduce our logical language for expressing properties of evolving knowledge bases, EKBL (Evolving Knowledge Base Logic), which we define as a branching-time temporal logic akin to CTL [16], which has become popular for expressing temporal behavior of concurrent processes and modules in finite state systems.

Syntax. The primitive logical operators of the language EKBL are:

- the Boolean connectives $\wedge$ ("and") and $\sim$ ("not");
- the evolution quantifiers A ("for all futures") and E ("for some future"); and
- the linear temporal operators X ("next time") and U ("until").

Atomic formulas of EKBL are identified with the rules in the language $\mathcal{L}_{\mathcal{A}}$, given an alphabet $\mathcal{A}$; composite formulas are *state formulas*, as defined—by means of atomic formulas and *evolution formulas*—below. Note that we use the symbol $\sim$ for negation in composite formulas, in order to distinguish it from the negation symbols used in atomic formulas occurring in rules.

1. Each atomic formula is a state formula.
2. If φ and ψ are state formulas, then $\varphi \wedge \psi$ and $\sim\varphi$ 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.

Intuitively, evolution formulas describe properties of the evolving knowledge base, since they use the linear-time operators "next time" and "until," which apply to a given infinite evolution path consisting of knowledge states which are reached by successive events. The operator X refers to the next state of the path and states that the formula φ is true, while the operator U refers to a (possibly empty) initial segment of the path, and asserts that φ holds true in each state of this segment and that immediately after it ψ holds true.

We may extend our language by defining further Boolean connectives $\vee$ ("or"), $\supset$ ("implies"), and $\equiv$ ("equivalence") in terms of other connectives, as well as important linear-time operators such as $\mathsf{F}\varphi$ ("finally φ") and $\mathsf{G}\varphi$ ("globally φ"), which intuitively evaluate to true in path p if φ is true at some resp. every stage p_i.

The following examples illustrate the use of the logical language EKBL for expressing certain properties a of given evolution frame.

Example 6 Even for our rather simple shopping agent of Example 4 some interesting properties can be formulated. For convenience, we allow in formulas non-ground rules as atoms, which is a shorthand for the conjunction of all ground instances which is assumed to be finite. Recall that we identify facts with literals.

- There can never be two current dates:

$$\varphi_1 \;=\; \mathsf{AG}((date(T) \wedge date(T')) \supset T = T'). \tag{2}$$

- If there is a shop on sale which is up, then a query is always performed:

$$\varphi_2 \;=\; \mathsf{AG}((up(S) \;\wedge\; (sale(S)) \supset site_queried). \tag{3}$$

□

Example 7 In order to see whether the mail agent in Example 5 works properly, the first property of the previous example (formula (2)), stating that there can never be two different current dates, applies with slight syntactic modifications, i.e.,

$$\varphi_3 \;=\; \mathsf{AG}((today(D) \wedge today(D')) \supset D = D'). \tag{4}$$

In addition, we may consider the following properties.

- The type of a message cannot change:

$$\varphi_4 \;=\; \mathsf{AG}(type(M,T) \supset {\sim}\mathsf{EF}(type(M,T') \;\wedge\; T \neq T')). \tag{5}$$

- If a message is removed or saved (at least once), then the message is never trashed until it is either deleted or saved:

$$\begin{aligned}\varphi_5 \;=\; \mathsf{AG}\big(\quad & (msg(m) \wedge \mathsf{AF}(remove(m) \vee save(m))) \supset \\ & \mathsf{A}({\sim}trash(m)\mathsf{U}(remove(m) \vee save(m))\;\big).\end{aligned} \tag{6}$$

□

Semantics. We now define formally the semantics of formulas in our language with respect to a given evolution frame. To this end, we introduce the following notation.

Definition 7 *Given an event class $\mathcal{EC}$, a* path *is an (infinite) sequence $p = (s_i)_{i\geq 0}$ of knowledge states $s_i \in s(\mathcal{EC})$ such that s_i is a successor of s_{i-1}, for every $i > 0$. By p_i we denote the knowledge state at stage i in p, i.e., $p_i = s_i$, for every $i > 0$.*

Definition 8 *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel\rangle$ be an evolution frame, let s be a knowledge state over $\mathcal{EC}$, and let p be a path. The satisfaction relation $EF, s \models \varphi$, resp. $EF, p \models \varphi$, where φ is an* EKBL *formula, is recursively defined as follows:*

1. *$EF, s \models r$ iff $r \in Bel(s)$, for any atomic* EKBL *formula r;*
2. *$EF, s \models \varphi_1 \wedge \varphi_2$ iff $EF, s \models \varphi_1$ and $EF, s \models \varphi_2$;*
3. *$EF, s \models {\sim}\varphi$ 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 $EF, s \models \varphi$ (resp., $EF, p \models \varphi$) holds, then knowledge state s (resp., path p) is said to *satisfy* formula φ *in* the evolution frame EF, or φ is *true* at state s (resp., path p) *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).

As easily seen, the operators F and G are expressed by $\mathsf{F}\varphi = \top \mathsf{U} \varphi$ and $\mathsf{G}\varphi = \sim(\top \mathsf{U} \sim\varphi)$, respectively, where $\top$ is any tautology; thus, $\mathsf{AG}\varphi = \sim\mathsf{E}(\top \mathsf{U} \sim\varphi)$ and $\mathsf{EG}\varphi = \sim\mathsf{A}(\top \mathsf{U} \sim\varphi)$. Other common linear-time operators can be similarly expressed, e.g., $\varphi \mathsf{B} \psi = \sim((\sim\varphi)\mathsf{U}\psi)$ ("φ before ψ"), or $\varphi \mathsf{V} \psi = \sim(\sim\varphi \mathsf{U}(\sim\psi))$ ("φ releases ψ").

Let us reconsider our running examples.

Example 8 It is easily verified that the initial knowledge base KB of the shopping agent satisfies both formulas (2) and (3) in the respective EPI evolution frame EF_{EPI}, i.e., $EF_{\mathsf{EPI}}, KB \models \varphi_1$ and $EF_{\mathsf{EPI}}, KB \models \varphi_2$.

As for KB as in Example 5 for the mail agent, this set satisfies formulas (4) and (6) in the respective EPI evolution frame EF_{EPI}, i.e., $EF_{\mathsf{EPI}}, KB \models \varphi_3$ and $EF_{\mathsf{EPI}}, KB \models \varphi_4$, while it is easily seen that it does not satisfy formula (5), i.e., $EF_{\mathsf{EPI}}, KB \not\models \varphi_5$. □

In what follows, we are mainly interested in relations of the form $EF, KB \models \varphi$, i.e., whether some formula φ is satisfied by some initial knowledge base KB with respect to some given evolution frame EF. In particular, we analyze in Section 7 the computational complexity of this problem.

6 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 9 *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* $\mathcal{EC}$*, where* $k' \in \{0, \ldots, k\}$*;*

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 obtained easily.

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. *$s \equiv^k_{EF} s'$ implies that $EF, s \models \varphi$ is equivalent to $EF, s' \models \varphi$, for any state formula φ in which U does not occur and the nesting depth with respect to E and A is at most k.*

Proof. We prove Part 1 of the theorem by induction on the formula structure of the state formula φ.

INDUCTION BASE. Let φ be an atomic formula and assume $s \equiv_{EF} s'$, for knowledge states s, s'. Obviously, it holds that $Bel(s) = Bel(s')$. Thus, it follows that $EF, s \models \varphi$ iff $Bel(s') \models \varphi$ and, hence, $EF, s \models \varphi$ iff $EF, s' \models \varphi$.

INDUCTION STEP. Assume that Part 1 of Theorem 1 holds for formulas ψ of depth at most $n-1$, i.e., $s \equiv_{EF} s'$ implies $EF, s \models \psi$ iff $EF, s' \models \psi$. Let φ be a formula of depth n, and consider the following cases.

- $\varphi = \psi_1 \wedge \psi_2$ or $\varphi = {\sim}\psi_1$. Then, ψ_1 and ψ_2 are of depth $n-1$ and, by the induction hypothesis, it holds that $EF, s \models \varphi$ iff $EF, s' \models \psi_1$ and $EF, s' \models \psi_2$, respectively $EF, s \models \varphi$ iff $EF, s' \not\models \psi_1$. Thus, again $EF, s \models \varphi$ iff $EF, s' \models \varphi$ follows.
- $\varphi = \mathsf{E}\psi$ or $\varphi = \mathsf{A}\psi$. Then, ψ is an evolution formula of depth $n-1$ of the form $\mathsf{X}\psi_1$ or $\psi_1 \mathsf{U} \psi_2$, where ψ_1 and ψ_2 have depth $n-2$. Since $s \equiv_{EF} s'$, we have that $EF, p \models \psi$ for a path $p = (s_i)_{i \geq 0}$ such that $s_0 = s$ iff $EF, p' \models \psi$, where $p' = (s'_i)_{i \geq 0}$ results from p by replacing s_i with any s'_i such that $s_i \equiv_{EF} s'_i$, for all $i \geq 0$. Since $s_0 \equiv_{EF} s'_0$, for each $i \geq 0$, such an s'_i exists. Thus, the induction hypothesis implies that $EF, p \models \psi$ iff $EF, p' \models \psi$.

  Hence, $EF, s \models \mathsf{E}\psi$ iff $EF, s' \models \mathsf{E}\psi$ follows. By symmetry of $\equiv_{EF}$, we similarly conclude that $EF, s \models \mathsf{A}\psi$ iff $EF, s' \models \mathsf{A}\psi$.

Thus, for every state formula of depth n, the statement in Part 1 of Theorem 1 holds. This concludes the induction and proves Part 1 of our result.

Concerning Part 2 of the theorem, observe that in order to prove a formula φ in which U does not occur and the evolution quantifier depth is at most $k \geq 0$, initial path segments of length at most $k+1$ need to be considered. This follows from the fact that evolution subformulas of φ can only be of form $\mathsf{X}\psi$. Moreover, since every evolution formula must be preceded by a quantifier E or A, at most k nested evolution formulas can occur in φ and every evolution formula of the above form, i.e., $\mathsf{X}\psi$, can be verified by considering the truth value of ψ in successor states of the current state. Hence, initial path segments of length at most $k+1$ do suffice. Since for two knowledge states s and s' such that $s \equiv^k_{EF} s'$, all knowledge states reachable in k steps are equivalent, $EF, s \models \varphi$ iff $EF, s' \models \varphi$ holds by the same inductive argument as in the proof of Part 1 above. Thus, Part 2 of Theorem 1 follows. □

By 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 $[x]_E = \{y \mid x\, E\, y\}$ 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), if 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 $K^{E,S}_{EF} = \langle S/E, R_E, L_E \rangle$, 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 $EF, s \models \varphi$ iff $K^{E,S}_{EF}, [s]_E \models \varphi$, 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. Prior to this, we introduce some convenient terminology and notation.

For any state $s \in \mathcal{S}_{EF}$, we denote by $dsc(s)$ the set of knowledge states containing s and all its descendants (with respect to $\mathcal{EC}$ in EF, which will be clear from the context), and by $\mathcal{T}(s)$ the ordered tree with root s where the children of each node s' are its successor states according to EF, and s' is labeled with $Bel(s')$. Furthermore, for any $S \subseteq \mathcal{S}_{EF}$, we define $dsc(S) = \bigcup_{s \in S} dsc(s)$, and call S *successor closed*, if $S = dsc(S)$, i.e., each successor of a knowledge state in S belongs to S. Note that for any $s \in \mathcal{S}_{EF}$, $\mathcal{T}(s)$ has node set $dsc(s)$, which is successor closed.

6.1 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. This is made precise in 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 a successor-closed set of knowledge states over $\mathcal{EC}$. Then, the following two conditions are equivalent:*

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

Proof.
($2 \Rightarrow 1$). Consider, for any $s \in S$, the tree $\mathcal{T}(s)$. At depth $i \geq 0$, there are $|\mathcal{EC}|^i$ different nodes, and thus up to depth k in total $\sum_{i=0}^{k} |\mathcal{EC}|^i = \frac{|\mathcal{EC}|^{k+1}-1}{|\mathcal{EC}|-1} < 2|\mathcal{EC}|^k$ many different nodes if $|\mathcal{EC}| > 1$, and $k+1$ many if $|\mathcal{EC}| = 1$. Thus, if $d = |S/ \equiv^0_{EF}|$ is the number of different equivalence classes of the relation $\equiv^0_{EF}$ with respect to S, then there are less than $c = d^{\max(\,2|\mathcal{EC}|^k, k+1)}$ many trees $\mathcal{T}(s)$, where $s \in S$, which are different up to depth k. Thus, there are at most c knowledge states $s_1, \ldots, s_c$, $s_i \in S$, $1 \leq i \leq c$, which are pairwise not strongly equivalent. Consequently, $\equiv_{EF}$ has at most c different equivalence classes with respect to S, and thus $\equiv_{EF}$ has a finite index with respect to S.

($1 \Rightarrow 2$). Suppose the relation $\equiv_{EF}$ has at most n different equivalence classes w.r.t. S. Then, there are at most n knowledge states $s_1, \ldots, s_n \in S$ which are pairwise not strongly equivalent, i.e., $s_i \not\equiv_{EF} s_j$ for all $1 \leq i < j \leq n$. Since strongly equivalent states are also weakly equivalent, n is thus also a finite upper bound for the equivalence classes of the relation $\equiv^0_{EF}$ w.r.t. S.

Now, for $i, j \in \{1, \ldots, n\}$ such that $i \neq j$, let $l = l_{i,j}$ be the smallest integer l for s_i and s_j, $1 \leq i < j \leq n$, such that $Bel(s_i + E_1, \ldots, E_l) \neq Bel(s_j + E_1, \ldots, E_l)$, but $Bel(s_i + E'_1, \ldots, E'_m) = Bel(s_j + E'_1, \ldots, E'_m)$, for all sequences of events $E'_1, \ldots, E'_m$, $0 \leq m < l$. Furthermore, let $k = \max_{i,j}(l_{i,j})$ be the largest such l over all s_i and s_j. Note that k is well defined and finite because of the finite index of $\equiv_{EF}$ w.r.t. S. It follows that if any two knowledge states $s, s' \in S$ are k-equivalent, then they are also strongly equivalent. Indeed, suppose the contrary, i.e., suppose $s \equiv^k_{EF} s'$, but $s \not\equiv_{EF} s'$. Then there exists a sequence of l events, $l > k$, such that $Bel(s + E_1, \ldots, E_l) \neq Bel(s' + E_1, \ldots, E_l)$, but $Bel(s + E'_1, \ldots, E'_m) = Bel(s' + E'_1, \ldots, E'_m)$ holds for all sequences of events $E'_1, \ldots, E'_m$, $0 \leq m \leq k < l$. From the assumption that $s \not\equiv_{EF} s'$, it follows that $s \equiv_{EF} s_i$ and $s' \equiv_{EF} s_j$, for some $i, j \in \{1, \ldots, n\}$ such that $i \neq j$. This implies that $l_{i,j} > k$, which contradicts the maximality of k. Thus, $s \equiv^k_{EF} s'$ implies $s \equiv_{EF} s'$, for all $s, s' \in S$. □

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

We remark that, as can be seen from the proof of Theorem 2, Condition 1 implies Condition 2 also for arbitrary S, while the converse does not hold in general for an S which is not successor closed.

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 concepts.

Definition 10 *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 all programs $Q_1, \ldots, Q_i$ $(0 \leq i \leq k)$. 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 some successor closed $S \subseteq \mathcal{S}_{EF}$. If $Bel(\cdot)$ is local and $comp_{EF}(\cdot)$ is incremental, then $\equiv_{EF}$ has a finite index with respect to S.*

Proof. Similar to the proof of Theorem 2, consider, for any knowledge state $s \in S$, the tree $\mathcal{T}(s)$. Each node in s' has label $Bel(s') = Bel(\boldsymbol{P} + (Q_1, \ldots, Q_n))$, where $\boldsymbol{P} = comp_{EF}(s)$ and Q_i, $i \geq 1$, are the increments of $comp_{EF}(\cdot)$ corresponding to the successive events E_i in $s' = s + E_1, \ldots, E_n$. Note that incrementality of $comp_{EF}(\cdot)$ guarantees that the length of $comp_{EF}(s')$ is at most n plus the length of $comp_{EF}(s)$. Since $Bel(\cdot)$ is k-local, up to depth k, there are at most $c = d^{\max(2|\mathcal{EC}|^k, k+1)}$ many different trees, where $d = |S/\equiv^0_{EF}|$. Thus, there are at most c update programs $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_c$, and, hence, knowledge states $s_1, \ldots, s_c$, which are pairwise not strongly equivalent. Consequently, $\equiv_{EF}$ has at most c different equivalence classes, from which the result follows. □

As an application of this result, we show that certain EPI evolution frames have a finite index. To this end, we use the following lemmata.

We say that a semantics $Bel(\cdot)$ for sequences of propositional ELPs satisfies *strong noninterference*, if it satisfies, for every propositional update sequence $\boldsymbol{P}$, the following condition: If, for every ELP P_1, P_2, and Q such that $Q \subseteq P_2$ and no pair of rules r, r' exists with $H(r) = \neg H(r')$, where $r \in Q$ and $r' \in (P_2 \setminus Q) \cup P_1$, then $Bel(\boldsymbol{P}, P_1, P_2) = Bel(\boldsymbol{P}, P_1 \cup Q, P_2 \setminus Q)$, i.e., the rules from Q can be moved from the last component to the penultimate one.

Recall that $Bel_E(\cdot)$ is the belief operator of the answer set semantics of update programs [13], as described in Section 2.

Lemma 1 $Bel_E(\cdot)$ *satisfies* strong noninterference*.*

Proof. The proof appeals to the rejection mechanism of the semantics. Let P_1, P_2, and Q be ELPs, such that the following condition $(*)$ holds: $Q \subseteq P_2$ and no pair of rules r, r' exists with $H(r) = \neg H(r')$, where $r \in Q$ and $r' \in (P_2 \setminus Q) \cup P_1$.

If $Q = \emptyset$, the lemma holds trivially. So let $r \in Q$, but no rule $r' \in Q$ exists such that $H(r) = \neg H(r')$. Then, there must not be such a rule r' in P_1 or P_2, otherwise condition $(*)$ is not fulfilled. Hence, no rule of

P_1 is rejected by r. Moreover, adding r to P_1 can neither cause an inconsistency of P_1, nor can r be rejected by a rule from $P_2 \setminus Q$. Thus, $Bel_E(\boldsymbol{P} + (P_1, P_2)) = Bel_E(\boldsymbol{P} + (P_1 \cup Q, P_2 \setminus Q))$ holds in this case.

Now let Q also contain some rule r' such that $H(r) = \neg H(r')$ (P_1 cannot contain such rules without violating the condition $(*)$). Then, Q must contain all rules with heads $H(r)$ and $\neg H(r)$ of P_2, and no such rule may exist in P_1, in order to fulfill $(*)$. Again, no rule of P_1 can be rejected by any rule of Q, and no rule of Q can be rejected by any rule from $P_2 \setminus Q$. Additionally, adding Q to P_1 makes P_1 inconsistent iff P_2 is inconsistent. As a consequence, also in this case, $Bel_E(\boldsymbol{P} + (P_1, P_2)) = Bel_E(\boldsymbol{P} + (P_1 \cup Q, P_2 \setminus Q))$. Since there are no other possibilities left, the lemma is shown. □

In our next result, we require Part 1 of the following lemma, which in turn will be relevant in Section 7.1.

Lemma 2 *Let **P** and **Q** be sequences of ELPs. Then,*

1. $Bel_E(\boldsymbol{P}) = Bel_E(\boldsymbol{Q})$ *if* $\mathcal{U}(\boldsymbol{P}) = \mathcal{U}(\boldsymbol{Q})$*, and*
2. *given that **P** and **Q** are propositional sequences over possibly infinite alphabets,* $\mathcal{U}(\boldsymbol{P}) = \mathcal{U}(\boldsymbol{Q})$ *if* $Bel_E(\boldsymbol{P}) = Bel_E(\boldsymbol{Q})$.

Proof. As for Part 1, if $\mathcal{U}(\boldsymbol{P}) = \mathcal{U}(\boldsymbol{Q})$ then $Bel_E(\boldsymbol{P}) = Bel_E(\boldsymbol{Q})$ is immediate from the definition of $Bel_E(\cdot)$.

To show Part 2, it suffices to prove that, given ELPs P_1 and P_2 over a set $\mathcal{A}$ of atoms,

$$\mathcal{AS}(P_1) = \mathcal{AS}(P_2) \text{ if } Bel_E(P_1) = Bel_E(P_2).$$

Suppose $\mathcal{AS}(P_1) \neq \mathcal{AS}(P_2)$, and assume first that $\mathcal{A}$ is finite. Without loss of generality, suppose that $S = \{L_1, \ldots, L_k\} \in \mathcal{AS}(P_1)$ but $S \notin \mathcal{AS}(P_2)$. This means that the constraint

$$c: \quad \leftarrow L_1, \ldots, L_k, not\, L_{k+1}, \ldots, not\, L_m\,,$$

where $L_{k+1}, \ldots, L_m$ are all the atoms from $\mathcal{A}$ missing in S, is in $Bel_E(P_2)$ but not in $Bel_E(P_1)$. However, this contradicts the hypothesis that $Bel_E(P_1) = Bel_E(P_2)$.

This proves the result for finite $\mathcal{A}$. For infinite $\mathcal{A}$, it is possible to focus on the finite set of atoms occurring in $\boldsymbol{P} \cup \boldsymbol{Q}$, since, as well-known for the answer set semantics, $A, \neg A \notin S$ for each $A \in \mathcal{A} \setminus \mathcal{A}'$ and $S \in \mathcal{AS}(P)$ if P is an ELP on $\mathcal{A}' \subseteq \mathcal{A}$. □

Now we can show the following result.

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

Proof. We show the result for propositional update sequences **P** and **P'** by induction on $k \geq 1$. Since the evaluation of $Bel_E(\cdot)$ for non-ground update sequences amounts to the evaluation of propositional sequences, the result for the non-ground case follows easily.

INDUCTION BASE. The base case $k = 1$ is trivial.

INDUCTION STEP. Assume that 1-equivalence of **P** and **P'** implies that they are $(k-1)$-equivalent, $k > 1$. Suppose further, that **P** and **P'** are 1-equivalent, but not k-equivalent. Then, there exist programs $Q_1, \ldots, Q_{k'}$, where $k' \in \{2, \ldots, k\}$, such that $Bel_E(\boldsymbol{P} + Q_1, \ldots, Q_{k'}) \neq Bel_E(\boldsymbol{P'} + Q_1, \ldots, Q_{k'})$, i.e.,

according to Part 1 of Lemma 2, there exists a (consistent) answer set $S \in \mathcal{U}(\boldsymbol{P} + Q_1, \ldots, Q_{k'})$, such that $S \notin \mathcal{U}(\boldsymbol{P'} + Q_1, \ldots, Q_{k'})$. We can remove every rule r from $Q_{k'-1}$ and $Q_{k'}$ such that either $S \not\models B(r)$, or r is a member of

$$Rej_{k'-1}(S, \boldsymbol{P} + Q_1, \ldots, Q_{k'}) \cup Rej_{k'}(S, \boldsymbol{P} + Q_1, \ldots, Q_{k'}) = Rej_{k'-1}(S, \boldsymbol{P} + Q_1, \ldots, Q_{k'}) = Rej_{k'-1}(S, \boldsymbol{P'} + Q_1, \ldots, Q_{k'}).$$

Let the resulting programs be denoted by $Q'_{k'-1}$ and $Q'_{k'}$, respectively. Note that still $S \in \mathcal{U}(\boldsymbol{P} + Q_1, \ldots, Q_{k'-2}, Q'_{k'-1}, Q'_{k'})$ and $S \notin \mathcal{U}(\boldsymbol{P'} + Q_1, \ldots, Q_{k'-2}, Q'_{k'-1}, Q'_{k'})$ must hold, since these rules can neither be generating for S, i.e., fire w.r.t. S, nor reject other rules. Observe also that $Q'_{k'-1} \cup Q'_{k'}$ cannot contain a pair of rules with conflicting heads. Otherwise, contradicting our assumption, S would be inconsistent since both rules were generating for S.

Now we construct the program $Q^*_{k'-1} = Q'_{k'-1} \cup Q'_{k'}$. From the strong noninterference property (Lemma 1), it follows that $Bel_E(\boldsymbol{P} + Q_1, \ldots, Q_{k'-2}, Q^*_{k'-1}, \emptyset) \neq Bel_E(\boldsymbol{P'} + Q_1, \ldots, Q_{k'-2}, Q^*_{k'-1}, \emptyset)$. Since, for every update sequence $\boldsymbol{Q}$, $Bel_E(\boldsymbol{Q} + \emptyset) = Bel_E(\boldsymbol{Q})$, it follows $Bel_E(\boldsymbol{P} + Q_1, \ldots, Q_{k'-2}, Q^*_{k'-1}) \neq Bel_E(\boldsymbol{P'} + Q_1, \ldots, Q_{k'-2}, Q^*_{k'-1})$. This means that $\boldsymbol{P}$ and $\boldsymbol{P'}$ are not $k'-1$-equivalent; however, this contradicts the induction hypothesis that $\boldsymbol{P}$ and $\boldsymbol{P'}$ are $k-1$-equivalent. Hence, $\boldsymbol{P}$ and $\boldsymbol{P'}$ are k-equivalent. □

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, from the proof of Theorem 3, 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 successor-closed 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, $|S/\equiv_{EF}| \leq d^{2|\mathcal{EC}|}$, where $d = |S/\equiv^0_{EF}|$.*

In [23], an analogous result has been shown for $Bel_{\oplus}(\cdot)$, i.e., 1-equivalence of dynamic update programs $\boldsymbol{P}$ and $\boldsymbol{P'}$ implies their strong equivalence, and, thus, $Bel_{\oplus}(\cdot)$ is local. Since for update policies over the LUPS or LUPS* language and their respective compilations, the same as for their EPI counterparts holds, we also get the following result:

Corollary 2 *Let EF be a LUPS evolution frame $\langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}_L, \Pi_L, \rho_L, Bel_{\oplus} \rangle$ or a LUPS* evolution frame $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}_{L^*}, \Pi_{L^*}, \rho_{L^*}, Bel_{\oplus} \rangle$ such that $\mathcal{EC}$ is finite, and let $S \subseteq \mathcal{S}_{EF}$ be a successor-closed 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, $|S/\equiv_{EF}| \leq d^{2|\mathcal{EC}|}$, where $d = |S/\equiv^0_{EF}|$.*

6.2 Contracting belief operators

Next, we discuss a refinement of strong equivalence, called *canonical equivalence*, which also yields a finite index, provided that 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 11 *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 $\boldsymbol{P}$ and $\boldsymbol{P}'$; 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 operator $Bel_{\oplus}(\cdot)$ (see Section 2).

By repeatedly removing duplicate rules and empty programs from any sequence $\boldsymbol{P} = (P_0, \ldots, P_n)$ of ELPs, we eventually obtain a non-reducible sequence $\boldsymbol{P}^* = (P_0^*, \ldots, P_m^*)$, which is called the *canonical form* of $\boldsymbol{P}$. 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 $\boldsymbol{P}$, we have that $\boldsymbol{P}$ and $\boldsymbol{P}^*$ are strongly equivalent.*

Proof. We must show that $\boldsymbol{P}$ and $\boldsymbol{P}^*$ are k-equivalent, for every $k \geq 0$. The proof is by induction on $k \geq 0$.

INDUCTION BASE. We show that $\boldsymbol{P}$ and $\boldsymbol{P}^*$ are 0-equivalent. The proof is by induction on the reduction process, i.e., on the number of required removals of rules or empty programs from $\boldsymbol{P}$ in order to obtain $\boldsymbol{P}^*$. For the induction base, suppose $\boldsymbol{P} = \boldsymbol{P}^*$. Then, $\boldsymbol{P}$ and $\boldsymbol{P}^*$ are trivially 0-equivalent. For the induction step, assume that $Bel(\boldsymbol{Q}) = Bel(\boldsymbol{Q}^*)$, for all sequences of programs $\boldsymbol{Q}$ such that the canonical form $\boldsymbol{Q}^*$ can be constructed using $n - 1$ removals of rules and empty programs. Let $\boldsymbol{P}$ be a sequence of programs such that the construction of $\boldsymbol{P}^*$ requires n removing steps, and let $\boldsymbol{P}'$ denote any sequence of programs obtained from $\boldsymbol{P}$ after $n - 1$ removals. Then, $Bel(\boldsymbol{P}) = Bel(\boldsymbol{P}')$, by induction hypothesis. Furthermore, $Bel(\boldsymbol{P}') = Bel(\boldsymbol{P}^*)$ follows trivially from Bel being contracting. Thus, $Bel(\boldsymbol{P}) = Bel(\boldsymbol{P}^*)$. We have shown that for any sequence $\boldsymbol{P}$ of programs, if $Bel(\cdot)$ is contracting, then $\boldsymbol{P}$ and $\boldsymbol{P}^*$ are 0-equivalent.

INDUCTION STEP. Suppose $k > 0$, and let $\boldsymbol{Q} = (\boldsymbol{P} + Q_1, \ldots, Q_k)$ and $\boldsymbol{R} = (\boldsymbol{P}^* + Q_1, \ldots, Q_k)$. Furthermore, let $\boldsymbol{Q}^*$ and $\boldsymbol{R}^*$ denote the canonical forms of $\boldsymbol{Q}$ and $\boldsymbol{R}$, respectively. We show that $\boldsymbol{Q}^* = \boldsymbol{R}^*$.

Suppose $\boldsymbol{P}^*$ is obtained from $\boldsymbol{P}$ using n reduction steps and $\boldsymbol{R}^*$ is obtained reducing $\boldsymbol{R}$ in m steps. We construct $\boldsymbol{Q}^*$ as follows. We first perform n reduction steps on the subsequence $\boldsymbol{P}$ of $\boldsymbol{Q}$, resulting in the sequence $\boldsymbol{R}$. Then we apply m reduction steps on $\boldsymbol{R}$. Since the reduction process is Church-Rosser, no further reductions can be applied, which proves $\boldsymbol{Q}^* = \boldsymbol{R}^*$. From the induction base, it follows that $\boldsymbol{Q}$ and $\boldsymbol{Q}^*$ are weakly equivalent, which proves k-equivalence of $\boldsymbol{P}$ and $\boldsymbol{P}^*$. □

6.3 Canonical evolution frames

In this section, we study the relationship between an evolution frame and its *canonized form*:

Definition 12 *Given an evolution frame EF, we call knowledge states $s, s' \in \mathcal{S}_{EF}$* 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., $comp_{EF^*}(s) = comp_{EF}(s)^*$).*

Immediately, we note the following properties.

Theorem 6 *Let EF be a contracting evolution frame. Then,*

1. *$EF, s \models \varphi$ iff $EF^*, s \models \varphi$, for any knowledge state s and any formula φ.*

2. *$\equiv_{EF}^{can}$ 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 $s, s' \in S$.*

Proof. In order to show Part 1, we consider the Kripke structures K_{EF} and K_{EF^*}, corresponding to a contracting evolution frame EF and its canonized evolution frame, respectively. Since, for every $s \in S$, it holds that $Bel(comp_{EF}(s))$ = $Bel(comp_{EF}(s)^*)$ = $Bel(comp_{EF^*}(s))$, equal states have equal labels in K_{EF} and K_{EF^*}. Hence, K_{EF} and K_{EF^*} coincide. As a consequence, $K_{EF}, s \models \varphi$ iff $K_{EF^*}, s \models \varphi$, for every $s \in S$, and hence $EF, s \models \varphi$ iff $EF^*, s \models \varphi$, for every $s \in \mathcal{S}_{EF}$.

As for the proof of Part 2, assume $s \equiv_{EF}^{can} s'$, for $s, s' \in S$ and some $S \subseteq \mathcal{S}_{EF}$. Then, it holds that $Bel(comp_{EF^*}(s)) = Bel(comp_{EF^*}(s'))$. Moreover, $Bel(comp_{EF}(s)) = Bel(comp_{EF^*}(s))$ holds, as well as $Bel(comp_{EF}(s')) = Bel(comp_{EF^*}(s'))$, which implies $Bel(comp_{EF}(s)) = Bel(comp_{EF}(s'))$, and the same is true for all corresponding successor states of s and s' due to the fact that they are canonically equivalent. Thus, $s \equiv_{EF}^{can} s'$ 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 13 *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 *if 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 $s = \langle KB; E_1, \ldots, E_n \rangle$, $\alpha(s) = \{E_{n-c'+1}, \ldots, E_n\}$, 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 $comp_{EF}(s)$ only depends on the belief set of the predecessor s' of s and the last $c + 1$ events in s (including the latest event). In particular, $c = 0$ means that only the latest event needs to be considered.

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 $S \subseteq \mathcal{S}_{EF}$ be successor closed and such that $\{\pi_0(s) \mid s \in S\}$ is finite. If (i) EF is contracting, (ii) there is some finite set $R_0 \subseteq \mathcal{L}_{\mathcal{A}}$ such that $comp_{EF}(s)$ contains only rules from R_0, for every $s \in S$, and (iii) EF is c-bounded, for some $c \geq 0$, then $\equiv_{EF}^{can}$ has a finite index with respect to S.*

Proof. We prove that $\equiv_{EF}^{can}$ has finite index w.r.t. S by means of Theorem 2. That is, we must show that $\equiv_{EF^*}^{0}$ has finite index w.r.t. S and that there exists a $k \geq 0$, such that for any two knowledge states $s, s' \in S$, $s \equiv_{EF^*}^{k} s'$ implies $s \equiv_{EF}^{can} s'$.

We first show that the relation $\equiv_{EF^*}^{0}$, i.e., weak canonical equivalence, has finite index w.r.t. S.

For any knowledge state $s \in S$, $comp_{EF^*}(s)$ yields an update sequence $\boldsymbol{P}$ of at most $|R_0|$ programs, i.e., $\boldsymbol{P} = (P_0, \ldots, P_n)$ and $n \leq |R_0|$ holds. To see this, suppose otherwise that $n > |R_0|$. Since $comp_{EF^*}(s)$ is canonical (and thus *contracting under empty updates*), none of the programs P_i, $0 \leq i \leq n$, is empty. Furthermore, since $comp_{EF}(s)$ only contains rules from R_0, this also holds for $comp_{EF^*}(s)$. Hence, there must be at least one rule $r \in R_0$, which occurs in at least two programs P_i, P_j, $0 \leq i, j \leq n$, and $i \neq j$. This, however, contradicts the fact that $comp_{EF^*}(s)$ is canonical (and thus *contracting under rule repetition*). Hence, our assumption does not hold, which proves $n \leq |R_0|$. Moreover, $|\bigcup_{i=0}^{n} P_i| \leq |R_0|$ holds for the canonical compilation $\boldsymbol{P}$ by the same argument: If $|\bigcup_{i=0}^{n} P_i| > |R_0|$, then there must be at least one rule $r \in R_0$, such that $r \in P_i \cap P_j$ for at least two programs P_i, P_j, $0 \leq i < j \leq n$; this contradicts that $comp_{EF^*}(s)$ is contracting.

As a consequence, we can roughly estimate the number of different canonical compilations $comp_{EF^*}(s)$ by

$$d = 2^{|R_0|-1}(|R_0|+1)! = \mathcal{O}(2^{|R_0|(\log|R_0|+1)})$$

(note that $(|R_0|+1)! \leq 2^{1+(|R_0|-1)\log(|R_0|+1)} \leq 2^{1+|R_0|\log|R_0|}$ for $|R_0| > 0$). This upper bound can be explained as follows. A canonical compilation need not contain all rules of R_0, hence we add a special fact for signaling that, given an ordered sequence of rules from R_0 and the special fact, a canonical compilation consists of all rules in the sequence up to the special fact. There are $(|R_0|+1)!$ permutations of such sequences, which is an over-estimate of the number of canonical compilations consisting of different ordered sequences of rules. For each such sequence, there are $2^{|R_0|-1}$ possibilities for the rules to be grouped into sequences of at most $|R_0|+1$ programs, respecting their order. To see this, observe that if we fixed a grouping into a sequence of programs for all but the last rule, then for the last rule there are two possibilities: It can either be added to the last program of the sequence, or we add a new program, consisting of the last rule only, to the sequence. Applying this argument recursively and observing that for the first rule there is only one possibility—it has to go into the first program of the sequence—the given bound follows. Hence, at most d different canonical compilations $comp_{EF^*}(s)$ can be built for all $s \in S$. Thus, at most d different belief sets $Bel(s)$ exist among all $s \in S$, proving that $\equiv^0_{EF^*}$ has finite index w.r.t. S.

Secondly, we show by induction on $k \geq c$ that for any two knowledge states $s, s' \in S$ canonical c-equivalence $s \equiv^c_{EF^*} s'$ implies strong canonical equivalence $s \equiv^{can}_{EF} s'$, which proves our result in virtue of Theorem 2. More precisely, we show for all $k \geq c$, that in the canonized evolution frame EF^*, c-equivalence of knowledge states $s, s' \in S$ implies their k-equivalence in EF^*.

INDUCTION BASE ($k = c$). Canonical c-equivalence of knowledge states $s, s' \in S$ trivially implies $s \equiv^c_{EF^*} s'$.

INDUCTION STEP ($k > c$). Assume that, for any two knowledge states $s, s' \in S$ and some $k > c$, $s \equiv^c_{EF^*} s'$ implies $s \equiv^{k-1}_{EF^*} s'$. We show that under this assumption $s \equiv^k_{EF^*} s'$ follows.

Let $s_k = s + E_1, \ldots, E_k$, $s_{k-1} = s + E_1, \ldots, E_{k-1}$, $s'_k = s' + E_1, \ldots, E_k$, and $s'_{k-1} = s' + E_1, \ldots, E_{k-1}$. Since $k > c$, the sets $\alpha(s_{k-1})$ and $\alpha(s'_{k-1})$ are equal and $Bel(s_{k-1}) = Bel(s'_{k-1})$ holds by induction hypothesis. Hence, the following equations hold:

$$\begin{aligned} A = f(Bel(s_{k-1}), \alpha(s_{k-1}), E_k) &= f(Bel(s'_{k-1}), \alpha(s'_{k-1}), E_k), \quad \text{and} \\ g(Bel(s_{k-1}), \alpha(s_{k-1}), A) &= g(Bel(s'_{k-1}), \alpha(s'_{k-1}), A). \end{aligned}$$

Consequently, $comp_{EF}(s_k) = comp_{EF}(s'_k)$ holds, which implies that $comp_{EF^*}(s_k) = comp_{EF^*}(s'_k)$, and thus $Bel(s_k) = Bel(s'_k)$. This proves canonical k-equivalence.

This proves that $s \equiv^c_{EF^*} s'$ implies $s \equiv^{can}_{EF} s'$, for any two knowledge states $s, s' \in S$. Since we have also shown that $\equiv^0_{EF^*}$ has finite index w.r.t. S, it follows from Theorem 2 that $\equiv^{can}_{EF}$ has finite index w.r.t. S. □

We remark that the existence of R_0 is trivial if we have a function-free (finite) alphabet, and, as common in many logic programming semantics, repetition of literals in rule bodies has no effect, and thus the set of nonequivalent rules is finite. A similar remark applies to the initial knowledge bases $\pi_0(s)$.

7 Complexity

In this section, we investigate the computational complexity of our evolution framework. To this end, in what follows we assume that the alphabet $\mathcal{A}$ of the evolution frames under consideration is finite and

propositional. Thus, we only deal with finite propositional (sequences of) programs which are the result of the state compilation $comp(s)$.

First, 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. Indeed, the compilation function may efficiently simulate Turing machine computations, such that the classical Halting Problem can be encoded easily in the above reasoning problem.

The results of Section 6 provide a basis for characterizing some decidable cases. We consider here the following class of propositional evolution frames.

Definition 14 *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ be a propositional evolution frame (i.e., $\mathcal{A}$ is propositional). Then, EF is called* regular *if the following three conditions hold:*

1. *The membership tests $E \in \mathcal{EC}$ and $r \in Bel(\boldsymbol{P})$ are feasible in* PSPACE *(e.g., located in the polynomial hierarchy), and the functions Π and ρ are computable in polynomial space (the latter with polynomial size output).*

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 fully characterized by rules of length polynomial in $\|EF\|$, i.e., there is some constant c such that $r \in Bel(\boldsymbol{P})$ iff $r \in Bel(\boldsymbol{P}')$ for all rules r of length $\leq \|EF\|^c$ implies $Bel(\boldsymbol{P}) = Bel(\boldsymbol{P}')$, for all update sequences $\boldsymbol{P}$ and $\boldsymbol{P}'$.*

Conditions 1 and 3 apply to the approaches in [2, 12, 14, 26, 27, 21, 18], and Condition 2 is reasonable to impose; note that none of these semantics is sensible to repetitions of literals in rule bodies. However, we could imagine semantics where, similar as in linear logic, literals are "consumed" in the inference process, and that repetition of literals alludes to available resources.

Before we state our first complexity result, let us briefly recall some well-known complexity results for the above-mentioned approaches. Deciding whether a literal $L \in Bel(P)$, for a literal L and a finite, propositional ELP P is coNP-complete. The complexity does not increase for the update approaches in [2, 12, 14, 26, 27, 18], i.e., deciding $L \in Bel_S(\boldsymbol{P})$ is coNP-complete for $S \in \{E, \oplus, Rev, PLP\}$, where $\boldsymbol{P}$ is a finite, propositional sequence of (at most two in case of PLP) ELPs. However, the complexity for abductive theory updates [21], when considering all possible selection functions, increases one level in the polynomial hierarchy: Deciding $L \in Bel_{Abd}(\boldsymbol{P})$ is Π_2^P-complete.

The following lemma will be used several times in the sequel.

Lemma 3 *Given a regular evolution frame $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$, a knowledge state $s \in \mathcal{S}_{EF}$, and a formula φ, suppose that $\equiv_{EF}$ has finite index, c, w.r.t. $S = dsc(s)$. Then, there exists a deterministic Turing machine M which checks $EF, s \models \varphi$ in space polynomial in $(q+1) \cdot (m_s + \log c + \|EF\| + \|\varphi\|)$, where q is the nesting depth of evolution quantifiers in φ, m_s is the maximum space required to store $s' \in S$ representing a class of $S/\equiv_{EF}$, and $\|\varphi\|$ denotes the size of formula φ.*

Proof. We first show that for evaluating evolution quantifiers $\mathsf{E}\psi$ or $\mathsf{A}\psi$, we may consider finite paths of length c in EF. Note that every path of length greater than c in EF must contain at least one pair of strongly equivalent knowledge states. While this is trivial if ψ is of form $\mathsf{X}\psi_1$, consider the case where ψ is of form $\psi_1 \mathsf{U} \psi_2$.

If a path p starting at s of arbitrary length exists such that $EF, p \models \psi_1 \mathsf{U} \psi_2$, then there exists also a path p' starting at s and of length at most c, such that $EF, p' \models \psi_1 \mathsf{U} \psi_2$. To see this, note that $EF, p \models \psi_1 \mathsf{U} \psi_2$ implies that $\psi_1 \mathsf{U} \psi_2$ is satisfied in a finite path p'' which is an initial segment of p. We can repeatedly shorten p'' to obtain p' as follows. For any pair of strongly equivalent knowledge states $s = p''_i$ and $s' = p''_j$ such that $i < j$, consider the sequence $p''_i, \ldots, p''_j$ of knowledge states between them. If ψ_2 is satisfied by none of them, we can cut $p''_{i+1}, \ldots, p''_j$ and replace each state $p''_{i+1}, p''_{i+2}, \ldots$ by an equivalent successor of p''_j, p''_{j+1} such that we obtain a (finite) path in EF. Otherwise, i.e., if $EF, p''_l \models \psi_2$ for some $l \in \{i, \ldots, j\}$, we can cut p'' immediately after the first such p''_l. It is easily verified that the resulting path p' has length at most c and still satisfies $\psi_1 \mathsf{U} \psi_2$.

Now consider the case $\mathsf{A}(\psi_1 \mathsf{U} \psi_2)$. Obviously, if $\psi_1 \mathsf{U} \psi_2$ is satisfied by all finite paths of length c starting at s, it will also be satisfied by all paths of arbitrary length. To see the converse direction, assume there is an infinite path p starting at s such that $EF, p \not\models \psi_1 \mathsf{U} \psi_2$. We show that then a finite path p' of length at most c starting at s exists such that $EF, p' \not\models \psi_1 \mathsf{U} \psi_2$. Observe that either (i) ψ_2 is false in every p_i, $i \geq 0$, or (ii) there exists some $i \geq 0$ such that $EF, p_i \models \sim\psi_1 \wedge \sim\psi_2$, and $EF, p_j \models \psi_1 \wedge \sim\psi_2$, for every $j \in \{0, \ldots, i-1\}$. In case (i), we can, as above, transform the initial segment $p'' = p_0, p_1, \ldots, p_{c-1}$ of p by repeatedly removing sequences between pairs of strongly equivalent knowledge states and eventually obtain a path p' as claimed. In case (ii), we start with $p'' = p_0, p_1, \ldots, p_i$ and again repeatedly remove sequences between pairs of strongly equivalent knowledge states to obtain a path p' of length at most c starting at s such that $EF, p' \not\models \psi_1 \mathsf{U} \psi_2$. Hence, if all infinite paths p starting at s satisfy $\psi_1 \mathsf{U} \psi_2$, then so do all paths of length c starting at s.

This proves that if there are at most c strongly inequivalent descendants of s, it suffices to consider paths of length c to prove whether $EF, s \models \varphi$.

Now an algorithm for deciding $EF, S \models \varphi$ is as follows. Starting at s, it recursively checks the satisfiability of φ by checking the satisfiability of its subformulas and evaluating Boolean connectives. For any subformula φ' of form $\mathsf{E}\psi$ (resp., $\mathsf{A}\psi$), guess nondeterministically, step by step, a path p in EF starting at s in order to witness $EF, p \models \psi$ (resp., refute $EF, p \models \psi$ and exploit $\mathsf{A}\psi \equiv \sim\mathsf{E}\sim\psi$) and check this by iterating through p for (at most) c steps, using a counter. The counter occupies space $\log c$ in a standard binary coding. Per nesting level, the algorithm requires space for one counter and for one descendant of s, which is bounded by m_s. Furthermore, due to the fact that EF is regular, $\varphi \in Bel(s)$ can be checked, for all $s \in \mathcal{S}_{EF}$ and atomic φ, in space polynomial in s, EF, and $\|\varphi\|$. Hence at each level, the algorithm runs in space Δ which is polynomial in $m_s + \log c + \|EF\| + \|\varphi\|$.

By applying Savitch's Theorem in the formulation for Turing machines with oracle access (cf. Theorem 2.27 in [5]), we can show by induction on the evolution quantifier depth $q \geq 0$ of a formula φ, that deciding $EF, s \models \varphi$ is feasible on a deterministic Turing machine M using space at most $(q+1)\Delta^2$. Savitch's Theorem states that if language A can be decided by a nondeterministic Turing machine with oracle set B

in space $f(n)$, then it can be decided by a deterministic Turing machine with oracle set B in space $f(n)^2$, providing $f(n) \geq \log n$. Furthermore, $f(n)$ must be space constructible (which is the case in our application of the lemma).

INDUCTION BASE ($q = 0$). Since the above algorithm operates deterministically in space $\Delta \leq \Delta^2$, the existence of M is obvious.

INDUCTION STEP ($q > 0$). Assume that formulas of evolution quantifier depth $\leq q - 1$ can be decided in deterministic space $q \cdot \Delta^2$ on some Turing machine M', and let φ have evolution quantifier depth q. If φ is of form $\mathsf{E}\psi$ (resp., $\mathsf{A}\psi$), then the above algorithm amounts to a nondeterministic oracle Turing machine M' using work space bounded by Δ and calling an oracle for deciding subformulas of form $\mathsf{E}\psi'$ (resp., $\mathsf{A}\psi'$). By Savitch's Theorem, there is a deterministic Turing machine M'' using work space at most Δ^2 which is equivalent to M' and uses the same oracle set. By the induction hypothesis, the oracle queries can be deterministically decided in space $q \cdot \Delta^2$. Hence, from M'' we can construct a deterministic Turing machine M deciding $EF, s \models \varphi$ which operates in work space $\Delta^2 + q \cdot \Delta^2 = (q+1)\Delta^2$. This M is easily extended to decide all φ of evolution quantifier depth q within the same space bound. This concludes the induction and the proof of the lemma. □

We then obtain the following complexity results.

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 c-bounded, where c is polynomial in $\|EF\|$, contracting, and functions Π and ρ are computable in space polynomial in the size of $comp_{EF}(\cdot)$.*

3. PSPACE*-complete, if EF is as in 2 and, moreover, all rules in the compilations $comp_{EF}(s')$ of descendants s' of s are from a set R_0 of size polynomial in $\|EF\|$.*

Proof. We first prove the upper bounds of these results. Recall that we assume a finite propositional alphabet $\mathcal{A}$. Hence, by Condition 3 of a regular evolution frame EF, there are only finitely many different belief sets $Bel(s)$. Indeed, the number of rules of length L is bounded by $(4|\mathcal{A}|)^L$, and hence there are $\mathcal{O}(2^{\|EF\|^{l_1}})$ (single exponential in EF) many rules, where l_1 is some constant, which are relevant for characterizing belief sets, and there are $\mathcal{O}(2^{2^{\|EF\|^{l_1}}})$, i.e., double exponentially many, different belief sets $Bel(\boldsymbol{P})$. This implies that $|S/\equiv^0_{EF}| \leq d$ where $d = \mathcal{O}(2^{2^{\|EF\|^{l_1}}})$ for any set $S \subseteq \mathcal{S}_{EF}$. Observe also that $\mathcal{EC}$ is finite and $|\mathcal{EC}| = \mathcal{O}(2^{\|EF\|^{l_2}})$, for some constant l_2. This follows from the finiteness of $\mathcal{A}$ and the fact that rules in events, as well as events themselves, have size at most polynomial in $\|EF\|$. In particular, there exist $\mathcal{O}(2^{\|EF\|^{l_{2,1}}})$ many different rules in events, for some constant $l_{2,1}$, and thus there are $\mathcal{O}(2^{\|EF\|^{l_{2,1}}})^{\|EF\|^{l_{2,2}}} = \mathcal{O}(2^{\|EF\|^{l_2}})$ many different events in $\mathcal{EC}$, for some constants $l_{2,2}$ and l_2. In the following, let $S = dsc(s)$.

Membership, Part 1. In order to prove an upper bound for Part 1 of the theorem, since $Bel(\cdot)$ is k-local, $comp_{EF}(\cdot)$ is incremental, and $\equiv^0_{EF}$ has finite index w.r.t. S, which is successor closed, Theorem 3 can be applied, establishing that $\equiv_{EF}$ has finite index w.r.t. S. Recall from the proof of Theorem 3 that an upper

bound for $|S/\equiv_{EF}|$ is given by $d^{2^{|\mathcal{EC}|^k}}$, where k is polynomial in $\|EF\|$, i.e., $k = \|EF\|^{l_3}$ for some constant l_3. Furthermore, $|\mathcal{EC}| = \mathcal{O}(2^{\|EF\|^{l_2}})$, for constant l_2. Hence, we obtain that there are at most

$$d^{(2^{\mathcal{O}(2^{\|EF\|^{l_2}})^{\|EF\|^{l_3}}})} = d^{\mathcal{O}(2^{\|EF\|^{l_2+l_3}})} = \mathcal{O}(2^{2^{\|EF\|^{l_1}}})^{\mathcal{O}(2^{\|EF\|^{l_2+l_3}})} = \mathcal{O}(2^{2^{\|EF\|^{l}}}),$$

i.e., double exponentially many knowledge states $s' \in S$ which are pairwise not strongly equivalent, for some constant l; in other words, $|S/\equiv_{EF}| = \mathcal{O}(2^{2^{\|EF\|^{l}}})$. Furthermore, we can store a representative, s', of every class in $S/\equiv_{EF}$ by storing KB and at most double exponentially many events. Since every event can be stored in polynomial space, overall double exponential space is sufficient to store s'. By application of Lemma 3, $EF, s \models \varphi$ can be verified in space polynomial in $(q+1)\cdot(m_s + \log c + \|EF\| + \|\varphi\|)$. We have shown above that m_s satisfies $m_s = \mathcal{O}(2^{2^{\|EF\|^{l_0}}})$, for some constant l_0. Furthermore, we have shown that the index of $\equiv_{EF}$ w.r.t. S, c, satisfies $c = \mathcal{O}(2^{2^{\|EF\|^{l}}})$. Hence, $\log c = \mathcal{O}(2^{\|EF\|^{l'}})$, for a constant l'. Consequently, $EF, s \models \varphi$ can be verified in 2-EXPSPACE.

Membership, Part 2. An upper bound for Part 2 of the theorem can be obtained as follows. The fact that $\equiv^0_{EF}$ has finite index w.r.t. S implies that $\equiv^0_{EF^*}$ has finite index w.r.t. S, too. And, as we have shown in the proof of Theorem 7, canonical c-equivalence implies strong canonical equivalence in a c-bounded, contracting evolution frame. Thus, $\equiv^{can}_{EF}$ has finite index w.r.t. S by Theorem 2. Furthermore, according to Theorem 6, $\equiv^{can}_{EF}$ is compatible with $\equiv_{EF}$. Hence, we may represent $Bel(s')$, $s' \in S$, by the canonical compilation $comp_{EF^*}(s')$, together with the last c events in s', where c is polynomial in $\|EF\|$. The polynomial size bound for rules in $comp_{EF}(s')$ also holds for $comp_{EF^*}(s')$ and thus, since EF is contracting, $\|comp_{EF^*}(s')\|$ is bounded by the number of different rules, which is $\mathcal{O}(2^{\|EF\|^{l}})$, for some constant l. Recalling the bound of $\mathcal{O}(2^{|R_0|(\log|R_0|+1)})$ for the number of different canonical compilations from the proof of Theorem 7, we obtain that there are

$$\mathcal{O}(2^{2^{\|EF\|^{l'}}\cdot(\log 2^{\|EF\|^{l'}}+1)}) = \mathcal{O}(2^{2^{\|EF\|^{l''}}}),$$

i.e., double exponential many different canonical compilations, where l' and l'' are suitable constants. Multiplied with the number of possibilities for the last c events, $|\mathcal{EC}|^c$, which is single exponential, as

$$|\mathcal{EC}|^c = \mathcal{O}(2^{\|EF\|^{l_2}})^c = \mathcal{O}(2^{\|EF\|^{h}}),$$

for some constant h, we obtain again that there are at most double exponentially many knowledge states $s' \in S$ which are pairwise not strongly equivalent. However, since $\|comp_{EF^*}(s')\| \leq |\mathcal{A}|^{\|EF\|^{l_1}} = \mathcal{O}(2^{\|EF\|^{h'}})$, for some constant h', we can represent every strongly inequivalent descendant $s' \in S$ of s, using $comp_{EF^*}(s')$ together with the last c events in single exponential space. Thus, EXPSPACE membership follows from Lemma 3.

Membership, Part 3. Next, we prove PSPACE membership for Part 3 of the theorem. The additional condition on $comp_{EF}(s')$ that all rules are from a set R_0 of size polynomial in $\|EF\|$ guarantees that $\|comp_{EF^*}(s')\|$ is polynomial in the size of EF, for any $s' \in S$. Using the same estimate as above, we thus obtain at most single exponentially many different canonical compilations, for states s'. Multiplied with the exponential number of possibilities for the last c events, we now obtain at most single exponentially many strongly inequivalent descendants of s. For storing them, we use again $comp_{EF^*}(s')$ together with the last c events, requiring the space of $\|comp_{EF^*}(s')\|$ plus c times the space of an event. Since $\|comp_{EF^*}(s')\|$,

c, and the size of an event are all polynomial in the size of EF, overall polynomial space is needed for representation, establishing PSPACE membership in virtue of Lemma 3.

Hardness. We show the lower bounds by encoding suitable Turing machine computations, using padding techniques, into particular evolution frames. In order to obtain a lower bound for Part 1, consider a regular evolution frame EF, where $\mathcal{A} = \{A_i \mid 1 \leq i \leq n\} \cup \{accept\}$, hence, $|\mathcal{A}| = n + 1$, and $Bel(\boldsymbol{P})$, defined below, is semantically given by a set of classical interpretations, where not is classical negation and repetition of literals in rule bodies is immaterial. Then, there exist 2^{n+1} classical interpretations yielding $2^{2^{n+1}}$ different belief sets $B_0, \ldots, B_{2^{2^{n+1}}-1}$. We assume an enumeration of interpretations $I_0, \ldots, I_{2^{n+1}-1}$, such that I_0 does not contain $accept$. Moreover, we consider a single event $E = \emptyset$. Let $l = 2^{2^n}$. The number of events, i, encountered for reaching a successor state s' of s_0 in $i < l$ steps serves as an index of its belief set, i.e., $Bel(s') = B_i$. For $i < l - 1$, B_i is obtained using interpretations I_0 and I_j as models, such that the j-th bit, $1 \leq j \leq \log l = 2^n$, of index i is 1. Thus, the belief sets $B_0, \ldots, B_{l-2}$ are pairwise distinct and under classical model-based semantics, $accept \notin B_i$ holds for $0 \leq i \leq l - 2$.

In state s_{l-1}, we simulate in polynomial time the behavior of an 2-EXPSPACE Turing machine M on some fixed input I. To this end, we use an action a and an update policy Π, such that $\Pi(s, E) = a$ for $|s| = m \cdot l + l - 1$, $m \geq 0$, if M accepts I, and $\Pi(s, E) = \emptyset$ otherwise. For all other knowledge states, i.e., if $|s| \bmod l \neq l - 1$, $\Pi(s, E) = \emptyset$. The realization assignment $\rho(s, A)$ is incremental and adds an empty program, $\emptyset$, if $A = \emptyset$, and the program $\{accept \leftarrow\}$ in case of $A = \{a\}$. The semantics $Bel(P_1, \ldots, P_k)$ is as follows. If $k \bmod l \neq l - 1$, then $Bel(P_1, \ldots, P_k) = B_j$, where $j = k \bmod l$. Otherwise, if $P_k = \{accept \leftarrow\}$, then $Bel(P_1, \ldots, P_k) = B$, where B is a fixed belief set containing $accept$, and, if $P_k \neq \{accept \leftarrow\}$, then $Bel(P_1, \ldots, P_k) = B_{l-1}$, where B_{l-1} is defined as B_i for $i < l - 1$, i.e., $accept \notin B_{l-1}$. As easy to see, there are at most $l + 1$ states $s_0, \ldots, s_l$ which are not 0-equivalent, and 1-equivalence of two states s and s' implies strong equivalence of s and s'. To see the latter, observe that $s \equiv^1_{EF} s'$ iff $|s| \bmod (l + 1) = |s'| \bmod (l + 1)$. Thus, $Bel(\cdot)$ is local. Furthermore, it is easily verified that the functions Π and ρ can be computed in polynomial time. The same is true for deciding $r \in Bel(\boldsymbol{P})$, $\boldsymbol{P} = (P_1, \ldots, P_k)$, where we proceed as follows. We first compute $j = k \bmod l$. If $j \neq l - 1$, then we scan the bits $b_1, b_2, \ldots, b_{\log j}$ of j, and for every bit $b_{j'}$ such that $b_{j'} = 1$, we compute its index, j', in binary (which occupies at most n bits), and extend its representation to length n by adding leading zeros if necessary. The resulting binary string is regarded as representation of the interpretation $I_{j'}$, where the bits encode the truth values of the atoms $A_1, \ldots, A_n$ and $accept$ is false. Hence, each model J of $\boldsymbol{P}$ can be computed in polynomial time; checking whether $J \models r$ is easy. Thus, deciding $r \in Bel(\boldsymbol{P})$ is polynomial if $j \neq l - 1$. Otherwise, i.e., if $j = l - 1$, depending on P_k, $r \in B$ (resp., $r \in B_{l-1}$) can be similarly decided in polynomial time.

Summarizing, B_{l-1} contains $accept$ iff M accepts I iff $EF, s_0 \models \mathsf{EF}\, accept$. Note that the dual formula $\varphi = \mathsf{AG}\, accept$ can be used if every $B_0, \ldots, B_{l-2}$ contains $accept$ (and B_{l-1} contains $accept$ iff M accepts I). Furthermore, the membership tests $E \in \mathcal{EC}$ and $r \in Bel(\boldsymbol{P})$, as well as the functions Π and ρ, are computable in PSPACE (in fact, even in polynomial time), thus deciding TEMPEVO in EF is 2-EXPSPACE-hard.

Let us now prove a lower bound for Part 2 of the theorem. Again, we consider a regular evolution frame EF over a finite alphabet $\mathcal{A} = \{A_i \mid 1 \leq i \leq n\} \cup \{accept\}$. Moreover, we consider the single event $E = \emptyset$. Let $comp_{EF}(\cdot)$ be an incremental compilation function that compiles a knowledge state s, $|s| < 2^n$, into a sequence of programs, $\boldsymbol{P} = (\{r_0\}, , \ldots, \{r_{|s|-1}\})$, consisting of $|s|$ programs each consisting of a single positive, non-tautological rule, such that all rules are pairwise distinct and do not contain $accept$. Furthermore, let the semantics $Bel(\cdot)$ be given by $Bel(P_0, , \ldots, P_n)$ containing all rules which are true in

the classical models of P_n. Note that under these assumptions, all states of length less than 2^n have mutually different belief sets, and $comp_{EF}(\cdot) = comp^{can}_{EF}(\cdot)$.

In state s, $|s| = 2^n$, we simulate in polynomial time the behavior of an EXPSPACE Turing machine M on input I. To this end, $\Pi(\pi_{2^n-1}(s), E)$ returns $A = \{a\}$, where a is an action which causes $accept$ to be included in the belief set B_{2^n} iff M accepts I, otherwise $\Pi(\pi_{2^n-1}(s), E) = \emptyset$. For all knowledge states s', such that $|s'| \bmod 2^n \neq 0$, $\Pi(\pi_{|s'|-1}(s'), E) = \emptyset$. Thus, M accepts I iff $EF, s_0 \models \mathsf{EF}\, accept$. Since EF is contracting and 0-bounded, and since the membership tests $E \in \mathcal{EC}$ and $r \in Bel(\cdot)$, as well as the functions Π and ρ are computable in PSPACE, it follows that deciding TEMPEVO in EF is EXPSPACE-hard.

Finally, we give a proof for the lower bound of Part 3 of the theorem, by encoding the problem of evaluating a quantified Boolean formula (QBF), which is well known to be PSPACE-hard, in the EPI framework: Let $\psi = Q_1 x_1 \ldots Q_n x_n \alpha$ be a QBF and let $\varphi = PQ_1 x_1, \ldots PQ_n x_n \alpha$, where $PQ_i = \mathsf{A}$ if $Q_i = \forall$ and $PQ_i = \mathsf{E}$ if $Q_i = \exists$, $1 \leq i \leq n$, be its corresponding state formula. Consider the following evolution frame EF_{EPI}, where $\mathcal{A} = \{x_i, c_i \mid 1 \leq i \leq n\} \cup \{0, 1\}$, the initial knowledge base $KB = \{x_i \mid 1 \leq i \leq n\} \cup \{c_1\}$, $\mathcal{EC} = \{\{0\}, \{1\}\}$, and the update policy Π_{EPI} is given by the following actions:

$$\begin{aligned}\Pi_{\mathsf{EPI}}(s, E) \quad = \{\, &\mathbf{assert}(c_{i+1}) \mid c_i \in Bel(s), 1 \leq i \leq n-1\} \cup \\ \{\, &\mathbf{retract}(c_i) \mid c_i \in Bel(s), 1 \leq i \leq n\} \cup \\ \{\, &\mathbf{assert}(\neg x_i) \mid c_i \in Bel(s), 0 \in E, 1 \leq i \leq n\} \cup \\ \{\, &\mathbf{retract}(x_i) \mid c_i \in Bel(s), 0 \in E, 1 \leq i \leq n\}.\end{aligned}$$

Intuitively, a counter for events is implemented using atoms c_i, $1 \leq i \leq n$, and each event, which may be 0 or 1, assigns a truth value to the variable encoded by literals over atoms x_i, $1 \leq i \leq n$. Hence, Π_{EPI} creates a truth assignment in n steps. Thus, it is easily verified that $EF_{\mathsf{EPI}}, KB \models \varphi$ iff ψ is true. Note that after n steps, i.e., for all knowledge states $|s| \geq n$, $\Pi_{\mathsf{EPI}}(s, E)$ is always empty. This implies that EF_{EPI} is n-bounded. Moreover, Π is factual, i.e., it consists only of facts (of update commands), yielding a contracting compilation function $comp_{\mathsf{EPI}}(\cdot)$ which uses only facts over $\mathcal{A}$. Thus, and since the membership tests $E \in \mathcal{EC}$ and $r \in Bel_E(\cdot)$, as well as the functions Π and ρ are computable in PSPACE, it follows that TEMPEVO in Part 3 is PSPACE-hard. □

While, for the propositional EPI framework, $Bel(s)$ depends in general on all events in s, it is possible to restrict $\mathcal{AC}_{\mathsf{EPI}}$ to the commands **assert** and **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, as shown in [15]. 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, by Part 3 of Theorem 8, we get the following result.

Corollary 3 *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* PSPACE*-complete.*

The encoding of the QBF evaluation problem in the proof of Part 3 of Theorem 8 has further interesting properties. The initial knowledge base used, KB, is stratified and the resulting update policy is also *stratified* and *factual* as defined in [14, 15]. This means that $r \in Bel_E(\boldsymbol{P})$ can be decided in polynomial time for the given evolution frame. Since, moreover, the membership test $E \in \mathcal{EC}$, as well as the functions Π and ρ are computable in polynomial time, we get another corollary. To this end, we introduce the following notion.

Definition 15 *Let $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$ be a propositional evolution frame. EF is called* strongly regular *if the membership tests $E \in \mathcal{EC}$ and $r \in Bel(\boldsymbol{P})$ are feasible in polynomial time, as well as Π and ρ are computable in polynomial time.*

Now we can state the following result.

Corollary 4 *Deciding $EF, s \models \varphi$, given a strongly 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* PSPACE*-complete, if EF is c-bounded, where c is polynomial in $\|EF\|$, contracting, and all rules in the compilations $comp_{EF}(s')$ of descendants s' of s are from a set R_0 of size polynomial in $\|EF\|$.*

Thus, concerning evolution frames according to Part 3 of Theorem 8, we stay within the same complexity class if we suppose strong regularity. For strongly regular evolution frames according to Parts 1 and 2 of the theorem, we can establish the following result.

Theorem 9 *Given a strongly regular propositional evolution frame, $EF = \langle \mathcal{A}, \mathcal{EC}, \mathcal{AC}, \Pi, \rho, Bel \rangle$, a knowledge state s, and a formula φ, deciding $EF, s \models \varphi$ is*

1. 2-EXPTIME*-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 c-bounded, where c is polynomial in $\|EF\|$, and contracting.*

Proof. We first prove 2-EXPTIME membership for Part 1 of the theorem.

We do so by constructing a Kripke structure $K' = \langle S', R', L' \rangle$ in double exponential time in $\|EF\|$, such that $K', s \models \varphi$ iff $EF, s \models \varphi$, and S', R' are of size at most double exponential in the size of EF. This proves 2-EXPTIME membership by a well known result from model checking [10], stating that there is an algorithm for determining whether φ is true in state s of $K' = \langle S', R', L' \rangle$, running in time $\mathcal{O}(|\varphi| \cdot (|S'| + |R'|))$, where $|\varphi|$ denotes the evolution quantifier nesting depth of φ.

The Kripke structure $K' = \langle S', R', L' \rangle$ results from the Kripke structure $K_{EF}^{E,S} = \langle S, R, L \rangle$, where $E = S/\equiv_{EF}$, by restricting the labeling L to atomic subformulas of φ. Let $\mathcal{A}_\varphi$ denote the set of all atomic subformulas in φ. Then, $S' = S$, $R' = R$, and L' is the labeling function assigning to every $s \in S'$ a label $L'(s) = Bel(s) \cap \mathcal{A}_\varphi$. It is well known that $K', s \models \varphi$ iff $K_{EF}^{E,S}, s \models \varphi$, which in turn holds iff $EF, s \models \varphi$. Recall from the proof of Lemma 3 that in order to prove $EF, s \models \varphi$, paths need to be considered only up to length c, where $c = |E|$ is the maximum number of strongly inequivalent descendants of s. Moreover, we can use one knowledge state as a representative for every equivalence class in E, thus c strongly inequivalent knowledge states are sufficient. Recall also from the proof of Part 1 of Theorem 8 that for the given evolution frame EF, c is double exponential in $\|EF\|$, and that there are at most single exponentially many different events, i.e., $|\mathcal{EC}| = \mathcal{O}(2^{\|EF\|^l})$. We construct K' using a branch and bound algorithm that proceeds as follows.

The algorithm maintains a set O of open knowledge states, as well as the sets S', R', and L' of K'. Initially, $O = \{s\}$, $S' = \{s\}$, $R' = \emptyset$, and $L'(s) = Bel(s) \cap \mathcal{A}_\varphi$. For every knowledge state $s \in O$, the algorithm removes s from O and generates all possible (immediate) successor states s' of s. For every such s', it is checked whether it is strongly inequivalent to every $s \in S$. If so, s' is added to O and S, the tuple $\langle s, s' \rangle$ is added to R, and $L'(s') = Bel(s') \cap \mathcal{A}_\varphi$ is computed. Otherwise, if s' is strongly equivalent to a knowledge sate $s'' \in S'$, then the tuple $\langle s', s'' \rangle$ is added to R'. The algorithm proceeds until O is empty.

Since there are at most c strongly inequivalent descendants of s, the algorithm puts into O at most c, i.e., double exponentially many knowledge states, each of which has size at most double exponential in $\|EF\|$. Furthermore, since there exist at most single exponentially many different events, in every expansion of a node in O, at most exponentially many successors are generated, each in polynomial time. Since $Bel(\cdot)$ is polynomial and k-local, we can detect $s \equiv_{EF} s'$ in single exponential time by comparing the trees $\mathcal{T}(s)$ and

$\mathcal{T}(s')$ up to depth k, respectively. On levels 0, 1, ..., k, $\mathcal{T}(s)$ and $\mathcal{T}(s')$ contain $|\mathcal{EC}|^{2k} = \mathcal{O}(2^{\|EF\|^l})^{2k} = \mathcal{O}(2^{\|EF\|^{l'}})$ many nodes each, where l' is some constant. For each pair s_1 and s'_1 of corresponding nodes in $\mathcal{T}(s)$ and $\mathcal{T}(s')$, we must check whether $Bel(s_1) = Bel(s'_1)$ holds. Condition 3 of a regular evolution frame EF implies that single exponentially many tests $r \in Bel(comp(s_1))$ iff $r \in Bel(comp(s'_1))$ (for all rules r of length polynomial in $\|EF\|$) are sufficient. Strong regularity implies that deciding $r \in Bel(s_1) = Bel(comp(s_1))$ and $r \in Bel(s'_1) = Bel(comp(s'_1))$ are polynomial. Hence, deciding $Bel(s_1) = Bel(s'_1)$ is feasible in single exponential time in $\|EF\|$.

Summing up, testing for (at most) double exponentially many knowledge states s times single exponentially many successor states s' whether $s \equiv_{EF} s'$ can be done in

$$\mathcal{O}(2^{2^{\|EF\|^{l_1}}} \cdot 2^{\|EF\|^{l_2}} \cdot 2^{\|EF\|^{l_3}}) = \mathcal{O}(2^{2^{\|EF\|^{l}}})$$

time, for constants l_1, l_2, l_3 and l. Thus, the overall algorithm proceeds in double exponential time, i.e., K' can be computed in in double exponential time. This proves 2-EXPTIME membership of $EF, s \models \varphi$.

Hardness follows from a suitable encoding of 2-EXPTIME Turing machines M. To this end, a similar construction as in the hardness proof of Part 1 of Theorem 8 can be used, where the update policy $\Pi(s, E)$ simulates a 2-EXPTIME Turing machine rather than a 2-EXPSPACE Turing machine; note that the components of EF there have polynomial time complexity.

We prove Part 2 of the theorem by showing that the lower bound does not decrease when demanding strong regularity. To this end, we encode the computations of an EXPSPACE Turing machine, M, into a strongly regular evolution frame EF, such that EF is c-bounded, where c is polynomial in $\|EF\|$, and contracting.

Assume that M has binary tape alphabet $\{0, 1\}$ and runs in space 2^l, where l is polynomial in $\|EF\|$. Let us consider the following strongly regular evolution frame EF, where $\mathcal{A} = \{A_i \mid 1 \leq i \leq l\} \cup \{P_i \mid 1 \leq i \leq l\} \cup \{Q_i \mid 1 \leq i \leq m\} \cup \{Q_{accept}\}$, hence, $|\mathcal{A}| = 2l + m + 1 = k$. Then, there exist 2^k classical models, which we use to represent the configuration of M as follows. Atoms Q_i, $1 \leq i \leq m$, and Q_{accept} encode the state of M. Literals over atoms A_i and P_i, $1 \leq i \leq l$, are used to represent an index of M's tape in binary format. We use a "disjunctive" semantics as follows. Observe that we could use conjunctions of literals over atoms Q_i and P_i to encode the current state of M and the position of M's head, and conjunctions of literals over atoms A_i to encode the fact that the tape cell at the encoded index contains 1. By building the disjunction of a set of such conjunctions, we get a formula in disjunctive normal form (DNF) describing by its models the current configuration of M. We can make use of this observation by stipulating that we use rules to describe anti-models, i.e., interpretations which are not models of the current knowledge base KB. By defining $Bel(\cdot)$, taking the negation of the conjunction of all rules, we get a DNF, describing the models of KB, as intended, and $r \in Bel(KB)$ can be computed in polynomial time (by checking whether r is entailed by every disjunct), as required. Furthermore, the semantics for sequences of programs $Bel(P_1, \ldots, P_n)$ is defined by the semantics of their union $Bel(P_1 \cup \cdots \cup P_n)$.

We simulate in polynomial time the behavior of the EXPSPACE Turing machine M on input I as follows. Without loss of generality, we assume that the leftmost cell of M's tape (the cell at index 0) is always 1, that M initially is in state Q_0 and its head is in Position 0, and that M uses the first steps to write I to the tape (without accepting). Hence, the initial configuration can be represented by the single disjunct:

$$Q_1 \wedge \neg Q_2 \wedge \ldots \wedge \neg Q_m \wedge \neg Q_{accept} \wedge \neg P_1 \wedge \ldots \wedge \neg P_l \wedge \neg A_1 \wedge \ldots \wedge \neg A_l.$$

Thus, the initial knowledge base KB consists of the single constraint:

$$\leftarrow Q_1, \neg Q_2, \ldots, \neg Q_m, \neg Q_{accept}, \neg P_1, \ldots, \neg P_l, \neg A_1, \ldots, \neg A_l.$$

Table 1: Complexity results for regular and strongly regular evolution frames.

evolution frame EF	regular	strongly regular
(1): k-local & incremental	2-EXPSPACE-complete	2-EXPTIME-complete
(2): c-bounded & contracting	EXPSPACE-complete	EXPSPACE-complete
(3): (2) & $\|R_0\|$ polynomial	PSPACE-complete	PSPACE-complete

We use a single event $E = \emptyset$ as the tick of the clock and let $\Pi(s, E)$ implement M's transition function. That is, $\Pi(s, E)$ is a set A of actions $insert(r)$ and $delete(r')$, where r and $r'$are constraints over $\mathcal{A}$. Furthermore, we use $\rho_{\pm}$ (see Section 3.2) for adding (resp., removing) the constraints to (resp., from) the knowledge base KB, which amounts to the addition (resp., removal) of corresponding disjuncts to (resp., from) the DNF representing the current configuration of M. Since for every transition of M at most $|KB| + 1$ rules need to be inserted and at most $|KB|$ rules need to be removed, Π and $\rho_{\pm}$ are polynomial in the representation size of the belief set. Moreover, M accepts I iff $EF, KB \models \mathsf{EF}\, Q_{accept}$. Since EF is contracting and 0-bounded, and since the membership tests $E \in \mathcal{EC}$ and $r \in Bel(\cdot)$, as well as the functions Π and ρ are computable in time polynomial in the size of EF, deciding TEMPEVO in EF is EXPSPACE-hard. □

The complexity results obtained so far are summarized in Table 1. Further results can be derived by imposing additional meaningful constraints on the problem instances. We remark that if we restrict the semantics for $Bel(\cdot)$ to be defined in terms of a unique model (e.g., the extended well-founded semantics for ELPs [7, 30]), then in case of a c-bounded and contracting regular evolution frame EF, the complexity of deciding TEMPEVO drops from EXPSPACE to PSPACE. This can be argued by the observation that, in case of a unique model semantics, we have only single exponentially many different belief sets, and a knowledge state s can be represented by storing the (unique) model of $comp(s)$ and the last c events, which is possible in polynomial space. On the other hand, already for 0-bounded, contracting, strongly regular evolution frames with polynomial-size rule set R_0, the problem TEMPEVO is PSPACE-hard, as can be shown by adapting the construction in the proof of Part 3 in Theorem 3 to, e.g., evolution frames based on stratified or well-founded semantics for ELPs [7, 30].

7.1 Complexity of state equivalence

We conclude our complexity analysis with results concerning weak, strong, and k-equivalence of two finite propositional update programs under $Bel_E(\cdot)$ and $Bel_{\oplus}(\cdot)$, respectively.

We can state our first result, concerning the complexity of deciding weak equivalence under $Bel_E(\cdot)$, as a consequence of Lemma 2 (cf. Section 6.1).

Theorem 10 *Deciding whether two given finite propositional update programs* $\boldsymbol{P}$ *and* $\boldsymbol{Q}$ *are weakly equivalent under* Bel_E*, i.e., satisfying* $Bel_E(\boldsymbol{P}) = Bel_E(\boldsymbol{Q})$*, is* coNP*-complete.*

Proof. Membership follows from Lemma 2: The problem of checking $Bel_E(\boldsymbol{P}) = Bel_E(\boldsymbol{Q})$ for finite propositional update programs $\boldsymbol{P}$ and $\boldsymbol{Q}$ is equivalent to the task of checking whether they yield the same answer sets, i.e., whether $\mathcal{U}(\boldsymbol{P}) = \mathcal{U}(\boldsymbol{Q})$, which is in coNP.

For the lower bound, suppose that, without loss of generality, $\boldsymbol{P}$ has no answer set. Then checking whether $Bel_E(\boldsymbol{P}) = Bel_E(\boldsymbol{Q})$, for an update sequence $\boldsymbol{Q}$, amounts to the task of testing whether $\boldsymbol{Q}$ has no answer set, which is coNP-complete (cf. [11]). □

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

Lemma 4 *Let $\boldsymbol{P}$ and $\boldsymbol{Q}$ be finite propositional update programs over possibly infinite alphabets. Then, $\boldsymbol{P}$ and $\boldsymbol{Q}$ are not 1-equivalent under $Bel_E(\cdot)$ iff there is an ELP P and a set S such that (i) $S \in \mathcal{U}(\boldsymbol{P}+P)$ but $S \notin \mathcal{U}(\boldsymbol{Q}+P)$, or vice versa, (ii) $|S|$ is at most the number of different literals in $\boldsymbol{P}+\boldsymbol{Q}$ plus 1, and (iii) $|P| \leq |S|+1$. (Note that P has polynomial size in the size of $\boldsymbol{P}$ and $\boldsymbol{Q}$.)*

Proof. 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 literal 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.

If part. Let $\boldsymbol{P}$ and $\boldsymbol{Q}$ be finite propositional update programs and S a set such that Conditions (i), (ii), and (iii) hold. Then, $\boldsymbol{P}$ and $\boldsymbol{Q}$ are not 1-equivalent, since, by Lemma 2, $Bel_E(\boldsymbol{P}+P) \neq Bel_E(\boldsymbol{Q}+P)$ follows from (i).

Only-if part. Let $\boldsymbol{P}$ and $\boldsymbol{Q}$ be finite propositional update programs which are not 1-equivalent, i.e., there exists an ELP P such that $Bel_E(\boldsymbol{P}+P) \neq Bel_E(\boldsymbol{Q}+P)$. Moreover, again by application of Lemma 2, there exists a set S such that, without loss of generality, $S \in \mathcal{U}(\boldsymbol{P}+P)$ but $S \notin \mathcal{U}(\boldsymbol{Q}+P)$, i.e., Condition (i) holds.

By means of P and S, we construct a program P' and a set S' such that Conditions (i), (ii), and (iii) hold for P' and S': Consider the program $\Pi_1 = ((\boldsymbol{P} \cup P) \setminus Rej(S, \boldsymbol{P}+P))^S$. Then, according to the update answer set semantics, S can be generated from the rules in Π_1 by means of constructing its least fixed-point. Moreover, this still holds for the following simplification P_0^S of P^S. First, all rules which are not applied when constructing S can be removed. Second, among the remaining rules, we delete all rules with equal heads, except one of them, namely the rule which is applied first in the least fixed-point construction of S. (If several rules with equal head are applied at this level of the fixed-point construction, then we keep an arbitrary of them.) Thus, P_0^S consists of k positive rules, $r_1, \ldots, r_k$, with k different heads, $L_1, , \ldots, L_k$, which are exactly those literals derived by P_0^S. Hence, $|P_0^S| \leq |S|$.

We will create the program P' from P_0^S by employing unfolding. This means that some of the literals L_i, $1 \leq i \leq k$, will be eliminated by replacing every rule $r \in P_0^S$ such that $L_i \in B(r)$ by a rule r' such that $H(r') = H(r)$ and $B(r') = (B(r) \setminus \{L_i\}) \cup B(r_i)$, where r_i is the (single) rule having $H(r_i) = L_i$. Consider the program $\Pi_2 = (\boldsymbol{Q} \cup P \setminus Rej(S, \boldsymbol{Q}+P))^S$. Since S is not in $\mathcal{U}(\boldsymbol{Q}+P)$, it has a least fixed-point different from S. There are two scenarios:

1. Some literal $L_i \in S$ cannot be derived in Π_2. Let $S' = \{L \in S \mid L \text{ occurs in } \boldsymbol{P}+\boldsymbol{Q}\} \cup \{L_i\}$, and construct the program P' from P_0^S by unfolding, eliminating all literals $L \notin S'$.

2. All literals $L_i \in S$ can be derived, as well as some literal $L_{k+1} \notin S$ is derived by a rule $r \in P^S \setminus P_0^S$. Let $S' = \{L \in S \mid L \text{ occurs in } \boldsymbol{P}+\boldsymbol{Q}\} \cup \{L_{k+1}\}$, and build P' from P_0^S by adding the rule r_{k+1} : $L_{k+1} \leftarrow B(r)$ and eliminating all literals $L \notin S'$ from the resulting program by unfolding.

Then, in both cases, S' is not a least fixed-point of $((\boldsymbol{Q} \cup P') \setminus Rej(S', \boldsymbol{Q}+P'))^{S'}$, while it is a least fixed-point of $((\boldsymbol{P} \cup P') \setminus Rej(S', \boldsymbol{P}+P'))^{S'}$. This proves Condition (i). Furthermore, $|S'|$ is at most the

number of different literals in $\boldsymbol{P}+\boldsymbol{Q}$ plus 1, and $|P_0^S| \leq |S|$ implies $|P'| \leq |S'|+1$. Hence, Conditions (ii) and (iii) hold. □

Theorem 11 *Deciding strong equivalence (or k-equivalence, for a given $k \geq 0$) of two given finite propositional update programs $\boldsymbol{P}$ and $\boldsymbol{Q}$ over possibly infinite alphabets, is* coNP*-complete under $Bel_E(\cdot)$.*

Proof. For $k = 0$, the result is given by Theorem 10. Since, according to Theorem 4, 1-equivalence implies k-equivalence for all $k \geq 1$ under $Bel_E(\cdot)$, it remains to show coNP-completeness for $k = 1$.

Membership follows from Lemma 4: For deciding whether $\boldsymbol{P}$ and $\boldsymbol{Q}$ are not 1-equivalent, we guess a set S and a program P according to Conditions (ii) and (iii) of Lemma 4. Then we check in time polynomial in the size of $\boldsymbol{P}+\boldsymbol{Q}$ whether $\boldsymbol{P}$ and $\boldsymbol{Q}$ are not 1-equivalent. Hence, this problem is in NP. Consequently, checking whether $\boldsymbol{P}$ and $\boldsymbol{Q}$ are 1-equivalent is in coNP.

For showing coNP-hardness, for $k = 1$, we give a reduction from the problem of tautology checking. Consider a formula $F = \bigvee_{i=1}^{m}(L_{i1} \wedge L_{i1} \wedge L_{i3})$ over atoms $A_1, \ldots, A_n$, and two programs P and Q over an alphabet $\mathcal{A} \supseteq \{A_1, \ldots, A_n, T\}$ as follows:

$$\begin{aligned} P &= \{\neg A_i \leftarrow not\, A_i,\ A_i \leftarrow not\, \neg A_i \mid i = 1, \ldots, n\} \cup \\ &\quad \{T \leftarrow L_{j1}, L_{j2}, L_{j3} \mid j = 1, \ldots, m\}; \\ Q &= \{\neg A_i \leftarrow not\, A_i,\ A_i \leftarrow not\, \neg A_i \mid i = 1, \ldots, n\} \cup \\ &\quad \{T \leftarrow\}. \end{aligned}$$

Clearly, P and Q can be constructed in polynomial time. We show that F is a tautology if and only if P and Q are 1-step equivalent.

If part. Suppose F is not a tautology. Then, there is a truth assignment σ to $A_1, \ldots, A_n$ such that F is false, i.e., $L_{i1} \wedge L_{i1} \wedge L_{i3}$ is false for $1 \leq i \leq m$. Let R be the program consisting of facts $A_i \leftarrow$, for every atom A_i, $1 \leq i \leq n$, which is true in σ. It is easily verified that the set $S = \{A_i \mid A_i \leftarrow \in R\} \cup \{\neg A_j \mid A_j \leftarrow \notin R\}$ is the only update answer set of $P + R$, while $S' = S \cup \{T\}$ is the only update answer set of $Q + R$. Thus, P and Q are not 1-equivalent.

Only-if part. Suppose F is a tautology. Towards a contradiction, assume that P and Q are not 1-equivalent. Then, by virtue of Lemma 2, there is a (consistent) program R and some set S such that either $S \in \mathcal{U}(P+R)$ and $S \notin \mathcal{U}(Q+R)$, or $S \notin \mathcal{U}(P+R)$ and $S \in \mathcal{U}(Q+R)$ holds. Observe that, for any set S and every program R, the sets $Rej(S, P+R)$ and $Rej(S, Q+R)$ do not differ w.r.t. rules in $P \cap Q$. Furthermore, P^S and Q^S do not differ w.r.t. to rules in $(P \cap Q)^S$.

We first show that $|S \cap \{A_i, \neg A_i\}| = 1$ holds, for $1 \leq i \leq n$. Indeed, since S is consistent, $\{A_i, \neg A_i\} \subseteq S$ cannot hold for any $1 \leq i \leq n$. On the other hand, suppose that neither $A_i \in S$, nor $\neg A_i \in S$ holds for some $1 \leq i \leq n$. Then, S entails the rules $\neg A_i \leftarrow not\, A_i$ and $A_i \leftarrow not\, \neg A_i$ of $P \cap Q$, which also cannot be rejected (since neither $A_i \in S$ nor $\neg A_i \in S$). Thus, both $\neg A_i \leftarrow$ and $A_i \leftarrow$ are in $((P \cup R) \setminus Rej(S, P+R))^S$ as well as in $((Q \cup R) \setminus Rej(S, Q+R))^S$. However, this contradicts the assumption that S is a consistent answer set of either $P + R$ or $Q + R$. This proves $|S \cap \{A_i, \neg A_i\}| = 1$, for $1 \leq i \leq n$.

Assume first that $\neg T \in S$. Then, every rule r of P such that $H(r) = T$ and $S \models B(r)$ are in $Rej(S, P+R)$, and $T \leftarrow$ is in $Rej(S, Q+R)$. Since S is an answer set of either $P + R$ or $Q + R$, it is either the least set of literals closed under the rules of $P_1 = ((P \cup R) \setminus Rej(S, P+R))^S$ or under the rules of $Q_1 = ((Q \cup R) \setminus Rej(S, Q+R))^S$. Since P and Q coincide on all rules with head different from T, it

follows that S must be the least set of literals closed under the rules of P_1 as well as of Q_1. Thus, S is an answer set of both $P + R$ and $Q + R$, which is a contradiction. Hence, $\neg T \notin S$ holds.

It is now easy to show that $T \in S$ must hold. Indeed, if $S \in \mathcal{U}(P + R)$, then, since F is a tautology, $S \models B(r)$ for some rule $r \in P$ such that $H(r) = T$. Moreover, $r \notin Rej(S, P + R)$ since $\neg T \notin S$, which in turn means $T \in S$. If, on the other hand, S is an answer set of $Q + R$, then $T \leftarrow \notin Rej(S, Q + R)$ holds, and thus $T \in S$ must hold.

Now suppose that $S \in \mathcal{U}(P + R)$. Since F is a tautology, $S \models B(r)$ for some rule $r \in P$ such that $H(r) = T$. Since $\neg T \notin S$, it follows that $T \in S$. Since $T \leftarrow$ is in Q, and P and Q coincide on all rules except those with head T, it follows that S is the least set of literals closed under the rules of $((Q \cup R) \setminus Rej(S, Q + R))^S)$. Thus, $S \in \mathcal{U}(Q + R)$, which is a contradiction. On the other hand, suppose $S \notin \mathcal{U}(Q+R)$ first. Since $\neg T \notin S$, we have $T \in S$, and thus clearly S must be the least set of literals closed under the rules of $((P \cup R) \setminus Rej(S, P + R))^S)$. Hence, $S \in \mathcal{U}(P + R)$, which is again a contradiction.

Hence, a program R and a set S as hypothesized cannot exist. This shows that P and Q are 1-equivalent.

We have shown that for every $k \geq 0$, deciding k-equivalence of finite propositional update sequences $\boldsymbol{P}$ and $\boldsymbol{Q}$ is coNP-complete under the $Bel_E(\cdot)$ semantics, which proves our result. □

In [23] it is shown that two dynamic logic programs, $\boldsymbol{P}$ and $\boldsymbol{Q}$, are not k-equivalent, for $k > 0$, iff there exists a GLP P such that $\mathcal{D}(\boldsymbol{P} + P) \neq \mathcal{D}(\boldsymbol{Q} + P)$. This result, together with a corresponding version of Lemma 2 and complexity results for dynamic logic programming from [23], can be used to obtain the following analogous result.

Proposition 2 *Deciding weak, strong, or k-equivalence, for a given $k \geq 0$, of two given finite propositional dynamic logic programs $\boldsymbol{P}$ and $\boldsymbol{Q}$, is* coNP*-complete under $Bel_{\oplus}(\cdot)$.*

8 Related Work and Conclusion

Our work on evolving nonmonotonic knowledge bases is related to several works in the literature on different issues.

Clearly, our formalization of reasoning from evolution frames is closely related to model checking of CTL formulas [10], and so are our complexity results. The major difference is, however, that in Kripke structures the models are given implicitly by its labels. Nevertheless, 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. However, in current model checking systems (e.g., the *Symbolic Model Verifier* (SMV) [28], or its new version NuSMV [9]), state transitions must be specified in a polynomial-time language, but descriptions of these Kripke structures would require exponential space even for evolution frames with PSPACE complexity (e.g., EPI evolution frames). Thus, extensions of model checking systems would be needed for fruitful usability.

Our filtration results for identifying finitary characterizations, which are based on various notions of equivalence between knowledge states, are somewhat related to results in [29, 24], obtained independently of our work and of each other. While we were concerned with the equivalence of sequences of ELPs (including the case of single ELPs), one can define two logic programs, P_1 and P_2, to be equivalent (i.e., weakly equivalent in our terminology), if they yield the same answer sets. They are called strongly equivalent, similar in spirit to 1-equivalence in our terminology, iff, for any logic program P, programs $P_1 \cup P$ and $P_2 \cup P$ have the same answer sets. Pearce *et al.* [29] investigated efficient (i.e., linear-time computable) encodings of *nested logic programs*, a proper generalization of disjunctive logic programs, into QBFs. In

accordance with our results, they found that deciding whether two propositional nested logic programs are strongly equivalent is coNP-complete. In [24], the same result has been shown, but merely for disjunctive logic programs.

Lobo *et al.* introduced the $\mathcal{PDL}$ [25] language for policies, which contain *event-condition-action rules*, serving for modeling reactive behavior on observations from an environment. While similar in spirit, their model is different, and [25] focuses on detecting action conflicts (which, in our framework, is not an issue). In [31], reasoning tasks are considered which center around actions. Further related research is on planning, where certain reachability problems are PSPACE-complete (cf. [6]). Similar results were obtained in [34] 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 [17] important work on knowledge in multi-agent systems addresses evolving knowledge, but mainly at an axiomatic level. Wooldridge's [33] 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.

Leite [23] introduces in his Ph.D. thesis a language, KABUL, which is inspired by our EPI language, but goes beyond it, since this language foresees also possible updates to the update policy. That is, the function Π may change over time, depending on external events. This is not modeled by our evolution frames, in which Π is the same at every instance of time. However, a generalization towards a time-dependent update policy—and possibly other time-dependent components of an evolution frame—seems not difficult to accomplish. Furthermore, Leite's work does not include a formal language for expressing properties of evolving knowledge bases like ours, and also does not address complexity issues of the framework.

8.1 Further work

In this paper, we have presented a general framework for modeling knowledge bases built over (extended) logic programs, for which we have then defined a formal language, EKBL, for stating and evaluating properties of a nonmonotonic knowledge base which evolves over time. As we have shown, this framework, which results from an abstraction of previous work on update languages for nonmonotonic logic programs [14, 15], can be used to abstractly model several approaches for updating logic programs in the literature. Knowledge about properties of the framework may thus be helpful to infer properties of these and other update approaches, and in particular about their computational properties. In this line, we have studied semantic properties of the framework, and we have identified several classes of evolution frames for which reasoning about evolving knowledge bases in the language EKBL is decidable. In the course of this, we have established that reasoning about propositional evolving knowledge bases maintained by EPI update policies under the answer set semantics [14, 15] is PSPACE-complete.

While we have tackled several issues in this paper, other issues remain for further work. One issue is to identify further meaningful semantic constraints on evolution frames or their components, and investigate the semantic and computational properties of the resultant evolution frames. For example, iterativity of the compilation $comp_{EF}$, i.e., where the events are incorporated one at a time, or properties of the belief operator Bel, would be interesting to explore.

Another interesting topic, and actually related to this, is finding fragments of lower comlexity and, in particular, of polynomial-time complexity. Furthermore, the investigation of special event classes, e.g., *event patterns*, which exhibit regularities in sequences of events, is an interesting issue.

References

[1] J. Alferes, J. Leite, L. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic Logic Programming. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning* (*KR 1998*), pages 98–111. Morgan Kaufmann, 1998.

[2] J. Alferes, J. Leite, L. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic Updates of Non-Monotonic Knowledge Bases. *Journal of Logic Programming*, 45(1–3):43–70, 2000.

[3] J. Alferes, L. Pereira, H. Przymusinska, and T. Przymusinski. LUPS - A Language for Updating Logic Programs. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 1999*), volume 1730 of *LNAI*, pages 162–176. Springer, 1999.

[4] J. Alferes, L. Pereira, H. Przymusinska, and T. Przymusinski. LUPS - A language for updating logic programs. *Artificial Intelligence*, 138(1–2):87–116, 2002.

[5] J. Balcázar, J. Diaz, and J. Gabarró. *Structural Complexity I*. Springer, 1988. 2nd edition.

[6] C. Baral, V. Kreinovich, and R. Trejo. Computational Complexity of Planning and Approximate Planning in the Presence of Incompleteness. *Artificial Intelligence*, 122(1–2):241–267, 2000.

[7] G. Brewka. Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *Journal of Artificial Intelligence Research*, 4:19–36, 1996.

[8] G. Brewka. Declarative Representation of Revision Strategies. In W. Horn, editor, *Proc. 14th European Conf. on Artificial Intelligence* (*ECAI 2000*). IOS Press, 2000.

[9] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A New Symbolic Model Checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[10] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[11] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001. Abstract in *Proceedings of the 12th IEEE International Conference on Computational Complexity* (*CCC 1997*), pp. 82–101, 1997.

[12] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Considerations on Updates of Logic Programs. In M. Aciego, I. de Guzmán, G. Brewka, and L. Pereira, editors, *Proceedings of the 7th European Workshop on Logic in Artificial Intelligence* (*JELIA 2000*), volume 1919 of *LNAI*, pages 2–20. Springer, 2000.

[13] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On Updates of Logic Programs: Semantics and Properties. Technical Report INFSYS RR-1843-00-08, Technische Universität Wien, Inst. für Informationssysteme, Abt. Wissensbasierte Systeme, 2000.

[14] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. A framework for declarative update specifications in logic programs. In B. Nebel, editor, *Proceedings of the 17th Joint International Conference on Artificial Intelligence* (*IJCAI 2001*), pages 649–654. Morgan Kaufmann, 2001.

[15] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Declarative Update Policies for Nonmonotonic Knowledge Bases. Technical Report INFSYS RR-1843-02-07, Technische Universität Wien, Inst. für Informationssysteme, Abt. Wissensbasierte Systeme, 2002.

[16] E. Emerson. Temporal and Modal Logics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16. Elsevier, 1990.

[17] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[18] N. Foo and Y. Zhang. Updating Logic Programs. In H. Prade, editor, *Proceedings 13th European Conference on Artificial Intelligence* (*ECAI 1998*), pages 403–407. Wiley, 1998.

[19] D. Gabbay and P. Smets, editors. *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, volume III: Belief Change. Kluwer Academic, 1998.

[20] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3–4):365–386, 1991.

[21] K. Inoue and C. Sakama. Updating Extended Logic Programs through Abduction. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 1999*), volume 1730 of *LNAI*, pages 147–161. Springer, 1999.

[22] J. Leite. A Modified Semantics for LUPS. In *Proceedings of the 10th Portuguese Conference on Artificial Intelligence* (*EPIA 2001*), volume 2258 of *LNCS*, pages 261–275. Springer, 2001.

[23] J. Leite. *Evolving Knowledge Bases – Specification and Semantics*. PhD thesis, Departamento de Informática, Universidade Nova de Lisboa, 2829-516 Caparica, Lisboa, Portugal, 2002.

[24] F. Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning* (*KR 2002*), pages 170–176. Morgan Kaufmann, 2002.

[25] J. Lobo, R. Bhatia, and S. Naqvi. A Policy Description Language. In *Proceedings of the 16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence* (*AAAI/IAAI 1999*), pages 291–298. AAAI Press / MIT Press, 1999.

[26] V. Marek and M. Truszczyński. Revision Specifications by Means of Programs. In C. MacNish, D. Pearce, and L. Pereira, editors, *Proceedings European Workshop on Logics in Artificial Intelligence* (*JELIA 1994*), volume 838 of *LNAI*, pages 122–136. Springer, 1994.

[27] V. Marek and M. Truszczyński. Revision Programming. *Theoretical Computer Science*, 190(2):241–277, 1998.

[28] K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluver Academic, 1993.

[29] D. Pearce, H. Tompits, and S. Woltran. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In *Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA'01)*, volume 2258 of *LNCS*, pages 306–320. Springer, 2001.

[30] L. Pereira and J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 1992), Vienna, Austria*, pages 102–106. John Wiley and Sons, 1992.

[31] T. Son and J. Lobo. Reasoning about Policies using Logic Programs. In A. Provetti and T. Son, editors, *Proceedings AAAI 2001 Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning, Stanford, CA (Workshop Technical Report SS-01-01)*, pages 210–216. AAAI Press, 2001.

[32] M. Winslett. *Updating Logical Databases*. Cambridge University Press, 1990.

[33] M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[34] M. Wooldridge. The Computational Complexity of Agent Design Problems. In E. Durfee, editor, *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS 2000)*, pages 341–348. IEEE Press, 2000.

[35] Y. Zhang and N. Foo. Answer sets for prioritized logic programs. In *International Logic Programming Symposium*, pages 69–83. MIT Press, 1997.