moviola: Interpreting Dynamic Logic Programs via Multi-shot Answer Set Programming

Orkunt Sabuncu[1,2(✉)] and João Leite[2]

[1] TED University, Ankara, Turkey
orkunt.sabuncu@tedu.edu.tr
[2] NOVA LINCS, Universidade Nova de Lisboa, Caparica, Portugal
jleite@fct.unl.pt

Abstract. The causal rejection-based update semantics assign meanings to a Dynamic Logic Program (DLP), which is a sequence of logic programs each one updating the preceding ones. Although there are translations of DLPs under these update semantics to logic programs of Answer Set Programming (ASP), they have not led to efficient and easy to use implementations. This is mainly because such translations aim offline solving in a sense that the resulting logic program is given to an answer set solver to compute models of the current DLP and for any future updates the whole process has to be repeated from scratch. We aim to remedy this situation by utilizing multi-shot ASP, composed of iterative answer set computations of a changing program without restarting from scratch at every step. To this end, we developed a system called *moviola*, utilizing the multi-shot answer set solver *clingo*. Using the system, a user can interactively write a DLP, update it, compute its models according to various semantics on the fly.

1 Introduction

Dynamic knowledge bases incorporate new information that may not only augment the knowledge base, but also contradict with previous information. A DLP represents such knowledge base by a sequence of logic programs, each updates the preceding ones.

There are various causal rejection-based update semantics assign meanings to a DLP. They have been extensively studied and there are transformations of DLPs under these semantics to logic programs of ASP [1]. However, these transformations [5,9,10] have not led to efficient and easy to use implementations. The underlying reason for this is that these transformations foresee an offline solving process, i.e., the process ends after finding models of the input DLP. Hence, whenever a DLP is updated, the whole process of transforming and solving has to be repeated from scratch for the updated DLP. We aim to remedy this situation by utilizing multi-shot ASP, composed of iterative answer set computations of a changing program without killing the solver and restarting from scratch at every step. To this end, we developed a system called *moviola*[1],

[1] https://github.com/owizo/moviola.

M. Balduccini and T. Janhunen (Eds.): LPNMR 2017, LNAI 10377, pp. 336–342, 2017.
DOI: 10.1007/978-3-319-61660-5_30

utilizing the multi-shot answer set solver *clingo*. Using the system, a user can interactively write a DLP, update it, compute its models according to various semantics on the fly.

In multi-shot ASP, it is not allowed to join two programs both having definitions of an atom (i.e., each program has a rule that has the same atom in the head). This is a problem since the solver fixes necessary conditions for an atom to be true considering the rules defining it in one program and these conditions cannot be altered when we want to join the other program. This issue becomes an obstacle when encoding update semantics of DLPs in multi-shot ASP. We overcome this obstacle by using a technique called *chaining*, which involves binding redefinitions of an atom by rule chains. Similar techniques have been used in other contexts [2,3].

The lack of modern implementations of update semantics of DLPs builds up a barrier to utilize DLP in various application domains. We hope *moviola* helps to bridge the gap between theory and practice of update semantics of DLPs.

2 Preliminaries

A DLP is a finite sequence of ground non-disjunctive logic programs (denoted by $\langle \mathcal{P}_i \rangle_{i<n}$), each one updating the preceding ones. Unlike normal logic programs, programs in a DLP may include rules having default negation in the heads. Although default negation in the head can be compiled away to form a normal logic program and it does not increase the expressive power of the program [4], it plays an important role in DLPs by facilitating updates with contradictory knowledge.

The causal rejection-based update semantics utilize the principle that a rule should be rejected when a more recent contradictory rule appears for assigning meanings to a DLP. Among this class of semantics are justified update (JU; [5]), update answer set (AS; [6]), dynamic stable models (DS; [7]), and refined dynamic stable models (RD; [8]) semantics. In this section we cover JU and RD semantics.

Two rules r and k are *in conflict*, denoted by $r \bowtie k$, when they have complementary head literals.[2] The set $all(\boldsymbol{P})$ is composed of all rules belonging to the programs in $\boldsymbol{P}$.

Definition 1 (JU-model). *Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP over a set $\mathcal{A}$ of propositional atoms and $J \subseteq \mathcal{A}$ be an ASP interpretation. The set of rejected rules is defined as*

$$\mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}, J) = \{r \in \mathcal{P}_i \mid \exists j \exists k : i < j < n \ \wedge \ k \in \mathcal{P}_j \ \wedge \ r \bowtie k \ \wedge \ J \models B_k\},$$

where B_k is the body of k. J is a JU-model of $\boldsymbol{P}$ iff J is an answer set of the program $all(\boldsymbol{P}) \setminus \mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}, J)$.

[2] In this work, we assume that strong negation of atoms do not appear in logic programs of a DLP (refer to [9] for expansion of a DLP to make rule conflicts uniform).

Definition 2 (RD-model). *Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP over a set $\mathcal{A}$ of propositional atoms and $J \subseteq \mathcal{A}$ be an ASP interpretation. The set of rejected rules is defined as*

$$\mathsf{rej}_{\mathsf{RD}}(\boldsymbol{P}, J) = \{r \in \mathcal{P}_i \mid \exists j \exists k : i \leq j < n \ \wedge \ k \in \mathcal{P}_j \ \wedge \ r \bowtie k \ \wedge \ J \models B_k\},$$

where B_k is the body of k, and the set of default assumptions is defined as

$$\mathsf{def}(\boldsymbol{P}, J) = \{\sim l \mid l \in \mathcal{A} \ \wedge \ \neg \exists r \in all(\boldsymbol{P}) : (h = l \ \wedge \ J \models B_r)\},$$

where r is of the form $h \leftarrow B_r$. J is a RD-model of $\boldsymbol{P}$ iff $J' = least([all(\boldsymbol{P}) \setminus \mathsf{rej}_{\mathsf{RD}}(\boldsymbol{P}, J)] \cup \mathsf{def}(\boldsymbol{P}, J))$ where $least(X)$ denotes the least model of program X with all literals treated as positive atoms and $J' = J \cup \sim(\mathcal{A} \setminus J)$.

Example 1. Let $\boldsymbol{P} = \langle \{p.\}, \{\sim p \leftarrow \sim p.\} \rangle$ be a DLP. Observe that $M_1 = \{p\}$ is both a JU-model and a RD-model of $\boldsymbol{P}$. Considering $M_2 = \{\}$, the only rule of the first program in $\boldsymbol{P}$ is in $\mathsf{rej}_{\mathsf{RD}}(\boldsymbol{P}, M_2)$ and the default assumption $\sim p$ is not included in $\mathsf{def}(\boldsymbol{P}, M_2)$. Thus, M_2 is not a RD-model given that the least model of the corresponding program does not satisfy the condition in RD semantics. However, M_2 is a JU-model of $\boldsymbol{P}$ although it is unintended (the second program $\boldsymbol{P}$ is a tautology and it is not expected to change models of the DLP before the update).

3 Encoding Dynamic Logic Programs via Answer Set Programming

We developed a translation that encodes RD semantics of DLPs via traditional (i.e., single-shot) ASP.[3] This translation establishes the foundation for the multi-shot ASP encoding used by *moviola*. It is in principle similar to the translation defined in [10], but it is developed in anticipation of its extension to a multi-shot encoding. Later in this section, we present the core of this multi-shot encoding.

Single-shot ASP encoding. Let $\mathcal{A}$ be a set of propositional atoms. We define $\mathcal{A}^n$ and $\mathcal{A}^c$ as the sets $\{a^n | a \in \mathcal{A}\}$ and $\{c_a | a \in \mathcal{A}\}$ of new propositional atoms, respectively. For a literal l, the transformed literal l^n is equal to p^n if $l = \sim p$ and to p if $l = p$ where $p \in \mathcal{A}$ and $p^n \in \mathcal{A}^n$. The transformation extends to sets of literals, rules, and programs, i.e., $B^n = \{l^n | l \in B\}$, r^n is $h^n \leftarrow B^n$. given a rule r of the form $h \leftarrow B.$, and $P^n = \{r^n | r \in P\}$ given a program P. Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP over a set $\mathcal{A}$ of propositional atoms. For each rule $r \in all(\boldsymbol{P})$, d_r is a new propositional atom (considering r as an id of the rule) and $\mathcal{R} = \{d_r | r \in all(\boldsymbol{P})\}$. For a rule $r \in all(\boldsymbol{P})$ of the form $h \leftarrow B.$, the transformed rule r^d is $h \leftarrow B, \sim d_r.$ where $d_r \in \mathcal{R}$. Given a program P, $P^d = \{r^d | r \in P\}$. Additionally, we define $\mathcal{A}^- = \{a | \sim a \text{ occurs in } \boldsymbol{P}\}$.

The role of the transformation $(.)^n$ is to represent negative literals by a new positive atom. This is needed considering that an atom in $\mathcal{A}$ may have no

[3] Here, we explain the encoding for only RD semantics due to space constraints.

default assumption due to a rule with a satisfied body and the nature of *least* handling negative literals as positive ones. This technique is also applied in a similar translation [10] of RD semantics via ASP. Additionally, $c_p \in \mathcal{A}^c$ atom intuitively encodes conditions of when generation of default assumption for p must be avoided.

Next, we will define some program parts that will be utilized to assemble the transformed logic program encoding RD update semantics of DLPs.

Definition 3. *The base logic program* $\mathsf{B}(\boldsymbol{P})$*, rejection logic program* $\mathsf{R}_{\mathsf{RD}}(\boldsymbol{P})$*, and defaults logic program* $\mathsf{D}(\boldsymbol{P})$ *are defined as:* $\mathsf{B}(\boldsymbol{P}) = \{({\mathcal{P}_i}^n)^d | i < n\}$,

$$\begin{aligned}\mathsf{R}_{\mathsf{RD}}(\boldsymbol{P}) =& \{d_r \leftarrow {B_k}^n. \mid r \in \mathcal{P}_i, d_r \in \mathcal{R}, \exists j \exists k : i \leq j < n \ \wedge \ k \in \mathcal{P}_j \ \wedge \ r \bowtie k \wedge \\ & \qquad B_k \, is \, the \, body \, of \, k\}, \\ \mathsf{D}(\boldsymbol{P}) =& \{p^n \leftarrow {\sim}p, {\sim}c_p. \mid p \in \mathcal{A}^-\} \cup \\ & \{\leftarrow p, p^n. \mid p \in \mathcal{A}^-\} \ \cup \ \{\leftarrow {\sim}p, {\sim}p^n. \mid p \in \mathcal{A}^-\} \cup \\ & \{c_p \leftarrow B^n. \mid r \in all(\boldsymbol{P}) \, is \, a \, rule \, of \, the \, form \, p \leftarrow B. \, and \, p \in \mathcal{A}\}.\end{aligned}$$

Lemma 1. *Let* $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ *be a DLP and* $R = \mathsf{B}(\boldsymbol{P}) \cup \mathsf{R}_{\mathsf{RD}}(\boldsymbol{P}) \cup \mathsf{D}(\boldsymbol{P})$ *be the transformed program.* J *is a* RD*-model of* $\boldsymbol{P}$ *iff* J' *is an answer set of* R *s.t.* $J = J' \cap \mathcal{A}$.

Towards a multi-shot encoding. We explain the multi-shot ASP encoding used by *moviola* with an example DLP that is updated iteratively. Let $\mathcal{P}_1 = \{p \leftarrow {\sim}q.\}$ be the first logic program of a DLP $\langle \mathcal{P}_1 \rangle$ over a set $\mathcal{A}$ of propositional atoms. The following logic program is formed using the transformation defined in Definition 3 and captures the RD semantics of $\langle \mathcal{P}_1 \rangle$. Recall that $q^n \in \mathcal{A}^n$, c_p, $c_q \in \mathcal{A}^c$, $q \in \mathcal{A}^-$, and $d_1 \in \mathcal{R}$ (i.e., the identifier of the only rule of $\mathcal{P}_1$ is 1).

$$p \leftarrow q^n, {\sim}d_1. \ \text{(a)} \qquad c_p \leftarrow q^n. \ \text{(b)} \qquad q^n \leftarrow {\sim}q, {\sim}c_q. \ \text{(c)} \qquad (1)$$

$$\leftarrow q, q^n. \qquad \leftarrow {\sim}q, {\sim}q^n. \qquad (2)$$

Its only answer set $\{p, q^n, c_p\}$ corresponds to the only RD-model $\{p\}$ of the DLP.

Considering $\langle \mathcal{P}_1 \rangle$, *moviola* does not use rules (1–2) naturally, but it generates a multi-shot program that is based on these rules. It uses a technique called *chaining*, which is an effective remedy for the problem of redefinitions in multi-shot ASP. During the transformation process of rules defining an atom, we need an additional auxiliary *chain rule* and a *chain atom* that anticipate a future update program having a rule that has the same atom in the head. The chain formed by the chain rule is at first *open*, since it will be *closed* later by a rule defining the corresponding chain atom in a transformed future update program. To this end, we define *tagged* versions of atoms used in the traditional ASP program. Regarding the atom p in rule (1.a), for instance, the tagged atom ${}^0p$ is used in rule (3.a) to encode the only rule in $\mathcal{P}_1$ and ${}^1p$ is used as a chain atom to encode the corresponding chain rule (3.b) for p.

$${}^0p \leftarrow q^n, {\sim}d_1. \ \text{(a)} \qquad {}^0p \leftarrow {}^1p. \ \text{(b)} \qquad p \leftarrow {}^0p. \ \text{(c)} \qquad (3)$$

Formally, given a DLP $\langle \mathcal{P}_i \rangle_{i<n}$ and an atom a which is first defined in $\mathcal{P}_i$, the transformation uses ${}^{i-1}a^n$ for the rules defining a in $\mathcal{P}_i$ and ${}^{i}a^n$ for a's chain. Note that *moviola* uses non-tagged atoms in bodies of transformed rules corresponding to the rules of input DLP (for instance, q^n in rule (3.a)). Thus, whenever an atom is first defined in a DLP, a rule (for instance (3.c)) is added to define the non-tagged version of the atom.

Furthermore, in addition to atoms in $\mathcal{A}$ and $\mathcal{A}^n$, atoms in $\mathcal{A}^c$ and $\mathcal{R}$ also need chaining since future update programs may cause redefinitions of these atoms. Considering the example DLP $\langle \mathcal{P}_1 \rangle$, *moviola* generates the following rules in addition to rules (3).

$$ {}^0q^n \leftarrow \sim q, \sim c_q. \qquad {}^0q^n \leftarrow {}^1q^n. \qquad q^n \leftarrow {}^0q^n. \tag{4} $$

$$ {}^0c_p \leftarrow q^n. \qquad {}^0c_p \leftarrow {}^1c_p. \qquad c_p \leftarrow {}^0c_p. \tag{5} $$

$$ \leftarrow q, q^n. \ \text{(a)} \qquad \leftarrow \sim q, \sim q^n. \ \text{(b)} \qquad d_1 \leftarrow {}^1d_1. \ \text{(c)} \tag{6} $$

The only answer set $S_1 = \{p, {}^0p, q^n, {}^0q^n, c_p, {}^0c_p\}$ of the multi-shot ASP program composed of rules (3–6) corresponds to the only RD-model of $\langle \mathcal{P}_1 \rangle$.

Next, let us update $\langle \mathcal{P}_1 \rangle$ with the program $\mathcal{P}_2 = \{p \leftarrow x.\}$. Considering the updated DLP $\langle \mathcal{P}_1, \mathcal{P}_2 \rangle$, *moviola* generates a program composed of the following rules and joins it with the previous program using *clingo*. Observe that redefinition of p via the only rule of $\mathcal{P}_2$ closes the open chain through atom ${}^1p$ in rule (7.a). Moreover, the new chain rule (7.b) encodes provision for a new chain that may be closed via future updates including a rule having p in the head. Similar chaining for c_p is achieved by rules (8).

$$ {}^1p \leftarrow x, \sim d_2. \ \text{(a)} \qquad {}^1p \leftarrow {}^2p. \ \text{(b)} \qquad d_2 \leftarrow {}^2d_2. \ \text{(c)} \tag{7} $$

$$ {}^1c_p \leftarrow x. \qquad {}^1c_p \leftarrow {}^2c_p. \tag{8} $$

For the joined program, the multi-shot ASP solver of *moviola* computes S_1 again as the only answer set that corresponds to the only RD-model of $\langle \mathcal{P}_1, \mathcal{P}_2 \rangle$.

Recall that atoms in $\mathcal{R}$ may have redefinitions via future updates and for our running example, rules (6.c) and (7.c) already form new open chains. To illustrate encoding of rejecting rules in a multi-shot way, consider the update program $\mathcal{P}_3 = \{\sim p \leftarrow \sim p.\}$ and the resulting DLP $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \rangle$. The only rule of $\mathcal{P}_3$ is in conflict with rules of $\mathcal{P}_1$ and $\mathcal{P}_2$. Hence, there must be rules that define d_1 and d_2, and may cause rules of $\mathcal{P}_1$ and $\mathcal{P}_2$ to be rejected. Considering the open chains via atoms ${}^1d_1$ and ${}^2d_2$, *moviola* generates the following rules.

$$ {}^1d_1 \leftarrow p^n. \ \text{(a)} \quad {}^2d_2 \leftarrow p^n. \ \text{(b)} \quad {}^1d_1 \leftarrow {}^3d_1. \ \text{(c)} \quad {}^2d_2 \leftarrow {}^3d_2. \ \text{(d)} \tag{9} $$

Since there may be future updates having rules that are in conflict with rules of $\mathcal{P}_1$ and $\mathcal{P}_2$, rules (9.c) and (9.d) generate new open chains.

In addition to rules (9), the following rules are generated when update $\mathcal{P}_3$ arrives.

$$ {}^2p^n \leftarrow p^n, \sim d_3. \qquad {}^2p^n \leftarrow {}^3p^n. \qquad {}^2p^n \leftarrow \sim p, \sim c_p. \qquad d_3 \leftarrow {}^3d_3. \tag{10} $$

$$ p^n \leftarrow {}^2p^n. \qquad \leftarrow p, p^n. \qquad \leftarrow \sim p, \sim p^n. \tag{11} $$

clingo computes the answer set S_1 again as the only answer set that corresponds to the only RD-model of $\langle \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3 \rangle$.

4 Implementation

We implemented an interactive system called *moviola*, utilizing the multi-shot answer set solver *clingo* (version 5.1) and its Python scripting support. *moviola* is composed of a controller part that forms the interaction with the user and is written in Python, and an ASP meta-encoding capturing various update semantics of DLPs. The system is available online at a Github repository. (see footnote 3)

When a user enters an update logic program, *moviola* converts it to set of ASP facts. Later, these facts are fed into *clingo* with the meta-encoding [11] that practically implements the multi-shot ASP based transformation. The meta-encoding represents not just one (RD) semantics but all the causal rejection-based semantics. This is achieved by adding switch atoms that control which one of the semantics is active. This leads to a useful feature of *moviola* that the user may change the semantics anytime and investigate their differences. The reader may refer to its repository for demonstration of this feature and the full meta-encoding. (see footnote 3)

One aspect of the multi-shot ASP encoding is that definitions of chain atoms may come in a program joined later on. Hence, the grounder of *clingo* simplifies these chain atoms and their respective rules. To prohibit this, chain atoms are declared as external atoms [12]. Another aspect in multi-shot ASP is that the underlying module theory does not allow positive loops spanning over multiple programs that are joined [13]. For some DLPs, this situation may occur in the transformed encoding. To avoid unsound answers due to these undetected loops, we utilize the acyclicity theory feature of *clingo* [14].

5 Conclusion

We developed *moviola*, an interactive system that encodes various update semantics of DLPs via multi-shot ASP. It interprets DLPs using the multi-shot ASP solver *clingo*. Unlike previous implementations, *moviola* does not do redundant work by restarting computation from scratch at every step of update.

One important future work is to present the formalization of the transformation used by *moviola* and to provide its correctness. Although we have tested *moviola* with various small DLPs having theoretical importance, we have not conducted experiments involving large DLPs. Consequently, performing extensive empirical analysis is another important line of future research.

Acknowledgments. This work was partially supported by FCT under strategic project UID/CEC/04516/2013.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, New York (2003)
2. De Cat, B., Denecker, M., Stuckey, P., Bruynooghe, M.: Lazy model expansion: interleaving grounding with search. J. Artif. Intell. Res. **52**, 235–286 (2015)
3. Gebser, M., Janhunen, T., Jost, H., Kaminski, R., Schaub, T.: ASP solving for expanding universes. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) LPNMR 2015. LNCS (LNAI), vol. 9345, pp. 354–367. Springer, Cham (2015). doi:10.1007/978-3-319-23264-5_30
4. Janhunen, T.: On the effect of default negation on the expressiveness of disjunctive rules. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, pp. 93–106. Springer, Heidelberg (2001). doi:10.1007/3-540-45402-0_7
5. Leite, J.A., Pereira, L.M.: Generalizing updates: from models to programs. In: Dix, J., Pereira, L.M., Przymusinski, T.C. (eds.) LPKR 1997. LNCS, vol. 1471, pp. 224–246. Springer, Heidelberg (1998). doi:10.1007/BFb0054796
6. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: On properties of update sequences based on causal rejection. Theor. Pract. Logic Program. **2**(6), 711–767 (2002)
7. Alferes, J.J., Leite, J., Pereira, L., Przymusinska, H., Przymusinski, T.: Dynamic updates of non-monotonic knowledge bases. J. Logic Program. **45**(1–3), 43–70 (2000)
8. Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: The refined extension principle for semantics of dynamic logic programming. Stud. Logica **79**(1), 7–32 (2005)
9. Slota, M., Leite, J.: On condensing a sequence of updates in answer-set programming. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (2013)
10. Banti, F., Alferes, J.J., Brogi, A.: Operational semantics for DyLPs. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS, vol. 3808, pp. 43–54. Springer, Heidelberg (2005). doi:10.1007/11595014_5
11. Gebser, M., Kaminski, R., Schaub, T.: Complex optimization in answer set programming. Theor. Pract. Logic Program. **11**(4–5), 821–839 (2011)
12. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: *Clingo* = ASP + control: preliminary report. In: Technical Communication of the 30th International Conference on Logic Programming (2014)
13. Oikarinen, E.: Modular answer set programming. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 462–463. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74610-2_46
14. Bomanson, J., Gebser, M., Janhunen, T., Kaufmann, B., Schaub, T.: Answer set programming modulo acyclicity. Fund. Inform. **147**(1), 63–91 (2016)