# Early Recovery in Logic Program Updates*

Martin Slota[1], Martin Baláž[2], and João Leite[1]

[1] CENTRIA & Departamento de Informática, Universidade Nova de Lisboa
[2] Faculty of Mathematics, Physics and Informatics, Comenius University

**Abstract.** We pinpoint the limitations of existing approaches to the treatment of *strong* and *default negation* in answer-set program updates and formulate the *early recovery principle* that plausibly constrains their interaction.

**Keywords:** answer-set programming, updates, strong negation, default negation.

## 1 Introduction

The increasingly common use of rule-based knowledge representation languages in highly dynamic and information-rich contexts, such as the Semantic Web [1], requires standardised support for updates of knowledge represented by rules. Answer-set programming [2, 3] forms the natural basis for investigation of rule updates, and various approaches to answer-set program updates have been explored [4–16].

The most straightforward kind of conflict arising between an original rule and its update occurs when the original conclusion logically contradicts the newer one. Though the technical realisation and final result may differ significantly, depending on the particular rule update semantics, this kind of conflict is resolved by letting the newer rule prevail over the older one. Actually, under most semantics, this is also the *only* type of conflict that is subject to automatic resolution [4–6, 9, 10, 13, 14].

From this perspective, allowing for both *strong* and *default negation* to appear in heads of rules is essential for an expressive and universal rule update framework [7]. While strong negation is the natural candidate here, used to express that an atom *becomes explicitly false*, default negation allows for more fine-grained control: the atom only *ceases to be true*, but its truth value may not be known after the update. The latter also makes it possible to move between any pair of epistemic states by means of updates, as illustrated in the following example:

*Example 1 (Railway crossing [7]).* Suppose that we use the following logic program to choose an action at a railway crossing:

$$\mathsf{cross} \leftarrow \neg\mathsf{train}. \qquad \mathsf{wait} \leftarrow \mathsf{train}. \qquad \mathsf{listen} \leftarrow {\sim}\mathsf{train}, {\sim}\neg\mathsf{train}.$$

* M. Slota and J. Leite were supported by Fundação para a Ciência e a Tecnologia under project "ERRO – Efficient Reasoning with Rules and Ontologies" (PTDC/EIA-CCO/121823/2010). The collaboration between the co-authors resulted from the Slovak–Portuguese bilateral project "ReDIK – Reasoning with Dynamic Inconsistent Knowledge", supported by the APVV agency under SK-PT0-0028-10 and by Fundação para a Ciência e a Tecnologia (FCT/2487/3/6/2011/S).

P. Cabalar and T.C. Son (Eds.): LPNMR 2013, LNAI 8148, pp. 512–517, 2013.

The intuitive meaning of these rules is as follows: one should cross if there is evidence that no train is approaching; wait if there is evidence that a train is approaching; listen if there is no such evidence.

Consider a situation where a train is approaching, represented by the fact $(\mathsf{train}.)$. After this train has passed by, we want to update our knowledge to an epistemic state where we lack evidence with regard to the approach of a train. If this was accomplished by updating with the fact $(\neg\mathsf{train}.)$, we would cross the tracks at the subsequent state, risking being killed by another train that was approaching. Therefore, we need to express an update stating that all past evidence for an atom is to be removed, which can be accomplished by allowing default negation in heads of rules. In this scenario, the intended update can be expressed by the fact $(\sim\mathsf{train}.)$.

With regard to the support of negation in rule heads, existing rule update semantics fall into two categories: those that only allow for strong negation, and those that primarily consider default negation. As illustrated above, the former are unsatisfactory as they render many belief states unreachable by updates. As for the latter, they optionally provide support for strong negation by means of a syntactic transformation.

Two such transformations are known from the literature, both of them based on the principle of coherence: if an atom $p$ is true, its strong negation $\neg p$ cannot be true simultaneously, so $\sim\neg p$ must be true, and also vice versa, if $\neg p$ is true, then so is $\sim p$. The first transformation, introduced in [17], encodes this principle directly by adding, to both the original program and its update, the rules $(\sim\neg p \leftarrow p.)$ and $(\sim p \leftarrow \neg p.)$ for every atom $p$. This way, every conflict between an atom $p$ and its strong negation $\neg p$ directly translates into two conflicts between the objective literals $p$, $\neg p$ and their default negations. However, the added rules lead to undesired side effects that stand in direct opposition with basic principles underlying updates. Specifically, despite the fact that the empty program does not encode any change in the modelled world, the stable models assigned to a program may change after an update by the empty program.

This undesired behaviour is addressed in an alternative transformation from [7] that encodes the coherence principle more carefully. Nevertheless, this transformation also leads to undesired consequences, as demonstrated in the following example:

*Example 2 (Faulty sensor).* Suppose that we collect data from sensors and multiple sensors are used to supply information about the critical fluent $p$. In case of a malfunction of one of the sensors, we may end up with the inconsistent logic program $\{p., \neg p.\}$. At this point, no stable model of the program exists and action needs to be taken to find out what is wrong. If a problem is found in the sensor that supplied the first fact $(p.)$, after the sensor is repaired, this information needs to be reset by updating the program with the fact $(\sim p.)$. Following the universal pattern in rule updates, where recovery from conflicting states is always possible, we would expect that some stable model be assigned to the updated program. However, the transformational semantics for strong negation defined in [7] still does not provide any stable model – we remain without a valid epistemic state despite the fact that all conflicts have been solved.

In this short paper we discuss the issues with combining strong and default negation in the context of rule updates. Namely, after presenting the necessary preliminaries, we formulate a generic desirable principle that is violated by the existing approaches.

## 2 Preliminaries

We assume that a countable set of propositional atoms $\mathcal{A}$ is given and fixed. An *objective literal* is an atom $p \in \mathcal{A}$ or its strong negation $\neg p$. We denote the set of all objective literals by $\mathcal{L}$. A *default literal* is an objective literal preceded by $\sim$ denoting default negation. A *literal* is either an objective or a default literal. We denote the set of all literals by $\mathcal{L}^*$. As a convention, double negation is absorbed, so that $\neg\neg p$ denotes the atom $p$ and $\sim\sim l$ denotes the objective literal $l$. Given a set of literals $S$, we introduce the following notation: $S^+ = \{\, l \in \mathcal{L} \mid l \in S \,\}$, $S^- = \{\, l \in \mathcal{L} \mid \sim l \in S \,\}$, $\sim S = \{\, \sim L \mid L \in S \,\}$.

A *rule* is a pair $\pi = (\mathsf{H}_\pi, \mathsf{B}_\pi)$ where $\mathsf{H}_\pi$ is a literal, referred to as the *head of* $\pi$, and $\mathsf{B}_\pi$ is a finite set of literals, referred to as the *body of* $\pi$. Usually we write $\pi$ as $(\mathsf{H}_\pi \leftarrow \mathsf{B}_\pi^+, \sim\mathsf{B}_\pi^-.)$. A *fact* is a rule whose body is empty. A *program* is a set of rules.

An *interpretation* is a consistent subset of the set of objective literals, i.e., a subset of $\mathcal{L}$ does not contain both $p$ an $\neg p$ for any atom $p$. The satisfaction of an objective literal $l$, default literal $\sim l$, set of literals $S$, rule $\pi$ and program $P$ in an interpretation $J$ is defined in the usual way: $J \models l$ iff $l \in J$; $J \models \sim l$ iff $l \notin J$; $J \models S$ iff $J \models L$ for all $L \in S$; $J \models \pi$ iff $J \models \mathsf{B}_\pi$ implies $J \models \mathsf{H}_\pi$; $J \models P$ iff $J \models \pi$ for all $\pi \in P$. Also, $J$ is a *model of* $P$ if $J \models P$, and $P$ is *consistent* if it has a model. Furthermore, the set $[\![P]\!]_{\mathsf{SM}}$ of *stable models of* $P$ consists of all interpretations $J$ such that $J^* = \mathsf{least}(P \cup \mathsf{def}(J))$[1] where $\mathsf{def}(J) = \{\, \sim l. \mid l \in \mathcal{L} \setminus J \,\}$, $J^* = J \cup \sim(\mathcal{L} \setminus J)$ and $\mathsf{least}(\cdot)$ denotes the least model of the argument program in which all literals are treated as propositional atoms.

Turning our attention to rule updates, a *rule update semantics* assigns stable models to a sequence of programs where each component represents an update of the preceding ones. Formally, a *dynamic logic program* (DLP) is a finite sequence of programs and by $\mathsf{all}(\boldsymbol{P})$ we denote the multiset of all rules in the components of $\boldsymbol{P}$. A rule update semantics $\mathsf{S}$ assigns a *set of S-models*, denoted by $[\![\boldsymbol{P}]\!]_{\mathsf{S}}$, to $\boldsymbol{P}$. We concentrate on semantics based on the causal rejection principle [4–7, 9, 10, 13] which states that a rule is *rejected* if it is in a direct conflict with a more recent rule. The fundamental type of conflict between rules $\pi$ and $\sigma$ occurs when they have complementary heads, i.e. $\mathsf{H}_\pi = \sim\mathsf{H}_\sigma$. We define the most mature of these semantics, the *refined dynamic stable models* [9, 10]. Let $\boldsymbol{P} = \langle P_i \rangle_{i<n}$ be a DLP without strong negation. Given an interpretation $J$, the multisets of *rejected rules* $\mathsf{rej}(\boldsymbol{P}, J)$ and of *default assumptions* $\mathsf{def}(\boldsymbol{P}, J)$ are defined by:

$$\mathsf{rej}(\boldsymbol{P}, J) = \left\{\, \pi \in P_i \;\middle|\; i < n \wedge \exists j \geq i\, \exists \sigma \in P_j : \mathsf{H}_\pi = \sim\mathsf{H}_\sigma \wedge J \models \mathsf{B}_\sigma \,\right\} \quad ,$$
$$\mathsf{def}(\boldsymbol{P}, J) = \{\, \sim l \mid l \in \mathcal{L} \wedge \neg(\exists \pi \in \mathsf{all}(\boldsymbol{P}) : \mathsf{H}_\pi = l \wedge J \models \mathsf{B}_\pi) \,\} \quad .$$

The set $[\![\boldsymbol{P}]\!]_{\mathsf{RD}}$ of *RD-models of* $\boldsymbol{P}$ consists of all interpretations $J$ such that

$$J^* = \mathsf{least}\left([\mathsf{all}(\boldsymbol{P}) \setminus \mathsf{rej}(\boldsymbol{P}, J)] \cup \mathsf{def}(\boldsymbol{P}, J)\right) \quad .$$

Support for strong negation can be added to this semantics by performing a syntactic transformation that translates conflicts between opposite objective literals $l$ and $\neg l$ into conflicts between objective literals and their default negations. Two such transformations have been suggested based on the principle of coherence [7, 17]. For any program $P$ and DLP $\boldsymbol{P} = \langle P_i \rangle_{i<n}$ they are defined as follows: $\boldsymbol{P}^\dagger = \langle P_i^\dagger \rangle_{i<n}$, $\boldsymbol{P}^\ddagger = \langle P_i^\ddagger \rangle_{i<n}$,

$$P^\dagger = P \cup \{\, \sim\neg l \leftarrow l. \mid l \in \mathcal{L} \,\} \quad , \quad P^\ddagger = P \cup \{\, \sim\neg\mathsf{H}_\pi \leftarrow \mathsf{B}_\pi. \mid \pi \in P \wedge \mathsf{H}_\pi \in \mathcal{L} \,\} \quad .$$

[1] The original definition based on reducts [2, 3, 18] is equivalent to the one we use here [7].

## 3 Early Recovery Principle

The problem with existing semantics for strong negation in rule updates is that semantics based on the first transformation ($\boldsymbol{P}^{\dagger}$) assign too many models to some DLPs, while semantics based on the second transformation ($\boldsymbol{P}^{\ddagger}$) sometimes do not assign any model to a DLP that should have one. The former is illustrated in the following example:

*Example 3.* Consider the DLP $\boldsymbol{P}_1 = \langle P, U \rangle$ where $P = \{ p., \neg p. \}$ and $U = \emptyset$. Since $P$ has no stable model and $U$ does not encode any change in the represented domain, it should follow that $\boldsymbol{P}_1$ has no stable model either. However, $[\![ \boldsymbol{P}_1^{\dagger} ]\!]_{\mathsf{RD}} = \{ \{ p \}, \{ \neg p \} \}$, i.e. two models are assigned to $\boldsymbol{P}_1$ when using the first transformation to add support for strong negation. To verify this, observe that $\boldsymbol{P}_1^{\dagger} = \langle P^{\dagger}, U^{\dagger} \rangle$ where

$$P^{\dagger}: \quad p. \qquad \neg p. \qquad\qquad U^{\dagger}: \quad \sim p \leftarrow \neg p. \qquad \sim\neg p \leftarrow p.$$
$$\sim p \leftarrow \neg p. \qquad \sim\neg p \leftarrow p.$$

Consider the interpretation $J_1 = \{ p \}$. It is not difficult to verify that $\mathsf{rej}(\boldsymbol{P}_1^{\dagger}, J_1) = \{ \neg p., \sim\neg p \leftarrow p. \}$ and $\mathsf{def}(\boldsymbol{P}_1^{\dagger}, J_1) = \emptyset$, so it follows that

$$\mathsf{least}\left( \left[ \mathsf{all}(\boldsymbol{P}_1^{\dagger}) \setminus \mathsf{rej}(\boldsymbol{P}_1^{\dagger}, J_1) \right] \cup \mathsf{def}(\boldsymbol{P}_1^{\dagger}, J_1) \right) = \{ p, \sim\neg p \} = J_1^{*} \ .$$

In other words, $J_1 \in [\![ \boldsymbol{P}_1^{\dagger} ]\!]_{\mathsf{RD}}$ and similarly it can be verified that $\{ \neg p \} \in [\![ \boldsymbol{P}_1^{\dagger} ]\!]_{\mathsf{RD}}$.

Thus, the problem with the first transformation is that an update by an empty program, which does not express any change in the represented domain, may affect the original semantics. This behaviour goes against basic and intuitive principles underlying updates, grounded already in the classical belief update postulates [19, 20] and satisfied by virtually all belief update operations [21] as well as by the vast majority of existing rule update semantics, including the original RD-semantics.

This undesired behaviour can be corrected by using the second transformation instead. The more technical reason is that it does not add any rules to a program in the sequence unless that program already contains some original rules. However, its use leads to another problem: sometimes *no model* is assigned when in fact one is expected.

*Example 4.* Consider again Example 2, formalised as the DLP $\boldsymbol{P}_2 = \langle P, V \rangle$ where $P = \{ p., \neg p. \}$ and $V = \{ \sim p. \}$. It is reasonable to expect that since $V$ resolves the conflict present in $P$, a stable model should be assigned to $\boldsymbol{P}_2$. However, $[\![ \boldsymbol{P}_2^{\ddagger} ]\!]_{\mathsf{RD}} = \emptyset$. To verify this, observe that $\boldsymbol{P}_2^{\ddagger} = \langle P^{\ddagger}, V^{\ddagger} \rangle$ where $P^{\ddagger} = \{ p., \neg p., \sim p., \sim\neg p. \}$ and $V^{\ddagger} = \{ \sim p. \}$. Given an interpretation $J$, we conclude that $\mathsf{rej}(\boldsymbol{P}_2^{\ddagger}, J) = P^{\ddagger}$ and $\mathsf{def}(\boldsymbol{P}_2^{\ddagger}, J) = \emptyset$, so $J$ cannot belong to $[\![ \boldsymbol{P}_2^{\ddagger} ]\!]_{\mathsf{RD}}$ since

$$\mathsf{least}\left( \left[ \mathsf{all}(\boldsymbol{P}_2^{\ddagger}) \setminus \mathsf{rej}(\boldsymbol{P}_2^{\ddagger}, J) \right] \cup \mathsf{def}(\boldsymbol{P}_2^{\ddagger}, J) \right) = \{ \sim p \} \neq J^{*} \ .$$

Based on this example, in the following we formulate a generic *early recovery principle* that formally identifies conditions under which *some* stable model should be assigned to a DLP. For the sake of simplicity, we concentrate on DLPs of length 2 which are composed of facts.

We begin by defining, for every objective literal $l$, the sets of literals $\overline{l}$ and $\overline{\sim l}$ as follows: $\overline{l} = \{\sim l, \neg l\}$ and $\overline{\sim l} = \{l\}$. Intuitively, for every literal $L$, $\overline{L}$ denotes the set of literals that are in conflict with $L$. Furthermore, given two sets of facts $P$ and $U$, we say that $U$ *solves all conflicts in* $P$ if for each pair of rules $\pi, \sigma \in P$ such that $\mathsf{H}_\sigma \in \overline{\mathsf{H}_\pi}$ there is a fact $\rho \in U$ such that either $\mathsf{H}_\rho \in \overline{\mathsf{H}_\pi}$ or $\mathsf{H}_\rho \in \overline{\mathsf{H}_\sigma}$.

Considering a rule update semantics $\mathsf{S}$, the new principle simply requires that when $U$ solves all conflicts in $P$, $\mathsf{S}$ will assign *some model* to $\langle P, U\rangle$. Formally:

**Early recovery principle:** If $P$ is a set of facts and $U$ is a consistent set of facts that solves all conflicts in $P$, then $[\![\langle P, U\rangle]\!]_{\mathsf{S}} \neq \emptyset$.

We conjecture that rule update semantics should generally satisfy the above principle. In contrast with the usual behaviour of belief update operators, the nature of existing rule update semantics ensures that recovery from conflict is always possible, and this principle simply formalises the sufficient conditions for such recovery.

The introduced principle can guide the future addition of full support for both kinds of negations in other approaches to rule updates, such as those proposed in [8, 11, 14, 16]. Stronger versions of the principle that apply to DLPs of arbitrary length and with programs other than just sets of facts are also conceivable.

Furthermore, these considerations are also interesting in the context of updates of hybrid knowledge bases [22, 23] and for the development of well-founded rule and hybrid update semantics [24, 25]. An interesting path for future work is also the study of updates of expressive extensions of logic programs [26] in which different negations besides the classical one can be considered.

## References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American 284(5), 28–37 (2001)
2. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Proceedings of the 5th International Conference and Symposium on Logic Programming, ICLP/SLP 1988, pp. 1070–1080. MIT Press (1988)
3. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9(3-4), 365–385 (1991)
4. Leite, J.A., Pereira, L.M.: Generalizing updates: From models to programs. In: Dix, J., Moniz Pereira, L., Przymusinski, T.C. (eds.) LPKR 1997. LNCS (LNAI), vol. 1471, pp. 224–246. Springer, Heidelberg (1998)
5. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic updates of non-monotonic knowledge bases. The Journal of Logic Programming 45(1-3), 43–70 (2000)
6. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: On properties of update sequences based on causal rejection. Theory and Practice of Logic Programming (TPLP) 2(6), 721–777 (2002)
7. Leite, J.A.: Evolving Knowledge Bases. Frontiers of Artificial Intelligence and Applications, vol. 81, xviii + 307 p. Hardcover. IOS Press (2003)
8. Sakama, C., Inoue, K.: An abductive framework for computing knowledge base updates. Theory and Practice of Logic Programming (TPLP) 3(6), 671–713 (2003)
9. Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: The refined extension principle for semantics of dynamic logic programming. Studia Logica 79(1), 7–32 (2005)

10. Banti, F., Alferes, J.J., Brogi, A., Hitzler, P.: The well supported semantics for multidimensional dynamic logic programs. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 356–368. Springer, Heidelberg (2005)
11. Zhang, Y.: Logic program-based updates. ACM Transactions on Computational Logic 7(3), 421–472 (2006)
12. Šefránek, J.: Irrelevant updates and nonmonotonic assumptions. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 426–438. Springer, Heidelberg (2006)
13. Osorio, M., Cuevas, V.: Updates in answer set programming: An approach based on basic structural properties. Theory and Practice of Logic Programming 7(4), 451–479 (2007)
14. Delgrande, J.P., Schaub, T., Tompits, H.: A preference-based framework for updating logic programs. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 71–83. Springer, Heidelberg (2007)
15. Šefránek, J.: Static and dynamic semantics: Preliminary report. In: Mexican International Conference on Artificial Intelligence, pp. 36–42 (2011)
16. Krümpelmann, P.: Dependency semantics for sequences of extended logic programs. Logic Journal of the IGPL 20(5), 943–966 (2012)
17. Alferes, J.J., Pereira, L.M.: Update-programs can update programs. In: Dix, J., Przymusinski, T.C., Moniz Pereira, L. (eds.) NMELP 1996. LNCS, vol. 1216, pp. 110–131. Springer, Heidelberg (1997)
18. Inoue, K., Sakama, C.: Negation as failure in the head. Journal of Logic Programming 35(1), 39–78 (1998)
19. Keller, A.M., Winslett, M.: On the use of an extended relational model to handle changing incomplete information. IEEE Transactions on Software Engineering 11(7), 620–633 (1985)
20. Katsuno, H., Mendelzon, A.O.: On the difference between updating a knowledge base and revising it. In: Allen, J.F., Fikes, R., Sandewall, E. (eds.) Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, KR 1991, pp. 387–394. Morgan Kaufmann Publishers (1991)
21. Herzig, A., Rifi, O.: Propositional belief base update and minimal change. Artificial Intelligence 115(1), 107–138 (1999)
22. Slota, M., Leite, J.: Towards Closed World Reasoning in Dynamic Open Worlds. Theory and Practice of Logic Programming 10(4-6), 547–564 (2010)
23. Slota, M., Leite, J., Swift, T.: Splitting and updating hybrid knowledge bases. Theory and Practice of Logic Programming 11(4-5), 801–819 (2011)
24. Banti, F., Alferes, J.J., Brogi, A.: Well founded semantics for logic program updates. In: Lemaître, C., Reyes, C.A., González, J.A. (eds.) IBERAMIA 2004. LNCS (LNAI), vol. 3315, pp. 397–407. Springer, Heidelberg (2004)
25. Knorr, M., Alferes, J.J., Hitzler, P.: Local closed world reasoning with description logics under the well-founded semantics. Artificial Intelligence 175(9-10), 1528–1554 (2011)
26. Gonçalves, R., Alferes, J.J.: Parametrized logic programming. In: Janhunen, T., Niemelä, I. (eds.) JELIA 2010. LNCS, vol. 6341, pp. 182–194. Springer, Heidelberg (2010)