On Condensing a Sequence of Updates in Answer-Set Programming[*]

Martin Slota and **João Leite**
CENTRIA & Departamento de Informática
Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Abstract

Update semantics for Answer-Set Programming assign models to sequences of answer-set programs which result from the iterative process of updating programs by programs. Each program in the sequence represents an update of the preceding ones.

One of the enduring problems in this context is *state condensing*, or the problem of determining a single logic program that faithfully represents the sequence of programs. Such logic program should 1) be written in the same alphabet, 2) have the same stable models, and 3) be equivalent to the sequence of programs when subject to further updates.

It has been known for more than a decade that update semantics easily lead to non-minimal stable models, so an update sequence cannot be represented by a single non-disjunctive program. On the other hand, more expressive classes of programs were never considered, mainly because it was not clear how they could be updated further.

In this paper we solve the state condensing problem for two foundational rule update semantics, using nested logic programs. Furthermore, we also show that disjunctive programs with default negation in the head can be used for the same purpose.

1 Introduction

The growing use of Answer-Set Programming [Gelfond and Lifschitz, 1988; 1991] in dynamic domains calls more and more urgently for plausible and efficient methods for incorporating new, possibly conflicting rules into an answer-set program. Such methods are particularly important when dealing with *updates*, i.e., when the newly introduced rules represent a change that has occurred in the domain modelled by the program.

Naturally, an update of an answer-set program should result in a new answer-set program that *replaces* the old one and continues to be used in its place. In other words, the update should be specified as a binary operator on some class of programs over the same alphabet, guaranteeing that updates can be iterated if the need arises. However, despite the large body of work on answer-set program updates, this fundamental requirement has been largely neglected in the literature.

For instance, the largest group of program update semantics, based on the *causal rejection principle* [Leite and Pereira, 1998; Alferes *et al.*, 2000; Eiter *et al.*, 2002; Alferes *et al.*, 2005], define stable models for sequences of non-disjunctive programs where each program represents an update of the preceding ones. However, since they sometimes admit non-minimal models, which no non-disjunctive program can capture, they must resort to the introduction of additional meta-level atoms in order to construct a single program whose stable models correspond to the models assigned to the sequence. This leads to difficulties with iterating the update process.

A different approach in [Sakama and Inoue, 2003] deals with program updates by borrowing ideas from literature on *belief revision* and utilising an abductive framework to accomplish such updates. In this case, multiple alternative programs can be the result of an update and no mechanism is provided to choose among them.

A somewhat similar situation occurs with the approach of [Zhang, 2006] where intricate syntactic transformations are combined with a semantics for prioritised logic programs that ultimately leads to a set of logic programs. Since all of these programs together represent the result of the update, it is once again unclear how to construct a single program that combines all of them.

The rule update semantics suggested in [Delgrande *et al.*, 2007] are also based on syntactic transformations into a logic program with preferences among rules, but in contrast with [Zhang, 2006], the semantics of such programs is defined by directly specifying their preferred stable models and not by translation into an ordinary program (or a set thereof). Thus, although an actual syntactic object is constructed that represents the update, it needs to be interpreted in a richer formalism to take into account preferences among rules.

Finally, frameworks that specify program updates by manipulating dependencies on default assumptions induced by rules [Šefránek, 2006; 2011; Krümpelmann, 2012] are mainly concerned with identifying the effects of irrelevant updates and other theoretical properties of the stable models assigned to a pair or sequence of programs. They do not consider rep-

[*]Supported by Fundação para a Ciência e a Tecnologia under project “ERRO – Efficient Reasoning with Rules and Ontologies” (PTDC/EIA-CCO/121823/2010).

resenting the result of an update by a single program.

In this paper, we unravel the true potential of specifying updates as binary operators on some class of programs. Despite the fact that existing program update semantics do not seem compatible with this point of view, we show that at least some of them *can* be viewed in this manner.

In particular, we return to the foundational approaches to rule updates based on causal rejection, the *justified update semantics* [Leite and Pereira, 1998], (or *JU-semantics* for short) as well as the closely related *update answer set semantics* [Eiter *et al.*, 2002] (or *UA-semantics* for short). For these, we define binary update operators and show that applying the operator to any sequence of programs produces a program whose stable models coincide with the stable models assigned to the original sequence under JU- and UA-semantics, respectively. In this way, the new operator works as a way to *condense* any sequence of non-disjunctive programs into a single program that includes all relevant information about the sequence, not only to identify its stable models, but also for the purpose of performing further updates. Thereby, we solve the long-standing problem known as *state condensing* from the literature on causal rejection semantics for program updates.

To achieve this, our operators must deal with a more general class of programs than non-disjunctive ones. First we define simple and elegant operators that produce *nested logic programs* [Turner, 2003] with the required property. Subsequently, we show that the full expressivity of nested programs is not necessary for this purpose by defining an additional pair of operators that produce *disjunctive logic programs*, with default negation in heads of rules to allow for non-minimal stable models, and still maintain the same properties w.r.t. JU- and UA-semantics. Nevertheless, the former operators usually produce more readable programs, sometimes exponentially more concise than the programs returned by the latter ones.

Furthermore, since the update semantics defined in [Alferes *et al.*, 2000; 2005] coincide with JU- and UA-semantics on the class of acyclic programs [Homola, 2004], our results partially extend to those semantics as well.

The remainder of this paper is structured as follows: In Sect. 2 we provide the necessary technical background for our investigation. Section 3 introduces operators for condensing an update sequence into a single nested program while in Sect. 4 we refine the defined operators to work with the more restricted class of disjunctive logic programs. Finally, in Sect. 5 we discuss our findings in a broader context and conclude.

2 Preliminaries

Throughout this paper we consider a fixed finite set of propositional atoms $\mathcal{A}$. We adopt the syntax and stable model semantics of logic programs with nested expressions [Turner, 2003], introduced next. Subsequently, we define the semantics for program updates for which we develop condensing operators in Sects. 3 and 4.

Programs with Nested Expressions

The *set of objective literals* $\mathcal{L}$ consists of all atoms $p \in \mathcal{A}$ and their (strong) negations $\neg p$. The *opposite literal* to an objective literal l is denoted by $-l$ and defined as follows: for any atom p, $-p = \neg p$ and $-\neg p = p$. *Elementary formulas* are objective literals and the 0-place connectives $\bot$ ("false") and $\top$ ("true"). Formulas are built from elementary formulas using the unary connective *not* (default negation) and the binary connectives $\land$ and $\lor$.

A *(nested) rule* is an expression π of the form

$$\mathsf{h}(\pi) \leftarrow \mathsf{b}(\pi)$$

where $\mathsf{h}(\pi)$ and $\mathsf{b}(\pi)$ are formulas, called the *head* and *body* of π. A rule of the form $\mathsf{h}(\pi) \leftarrow \top$ is usually identified with the formula $\mathsf{h}(\pi)$. A *(nested) program* is a finite set of rules.

An *interpretation* is a consistent set of objective literals. Satisfaction of a formula ϕ in an interpretation J, denoted by $J \models \phi$, is defined recursively as follows:

- For elementary ϕ, $J \models \phi$ if and only if $\phi \in J$ or $\phi = \top$;
- $J \models \phi_1 \land \phi_2$ if and only if $J \models \phi_1$ and $J \models \phi_2$;
- $J \models \phi_1 \lor \phi_2$ if and only if $J \models \phi_1$ or $J \models \phi_2$;
- $J \models \mathit{not}\, \phi$ if and only if $J \not\models \phi$.

Furthermore, J satisfies a rule π, denoted by $J \models \pi$, if $J \models \mathsf{b}(\pi)$ implies $J \models \mathsf{h}(\pi)$, and J satisfies a program $\mathcal{P}$, denoted by $J \models \mathcal{P}$, if $J \models \pi$ for all $\pi \in \mathcal{P}$.

The *reduct* of a formula ϕ relative to J, denoted by ϕ^J, is obtained by replacing, in ϕ, every maximal occurrence of a formula of the form $\mathit{not}\, \psi$ with $\bot$ if $J \models \psi$ and with $\top$ otherwise. The reducts of a rule π and of a program $\mathcal{P}$ are, respectively,

$$\pi^J = \left(\mathsf{h}(\pi)^J \leftarrow \mathsf{b}(\pi)^J\right) \quad \text{and} \quad \mathcal{P}^J = \left\{\, \pi^J \;\middle|\; \pi \in \mathcal{P} \,\right\} \quad .$$

Finally, J is a *stable model* of a program $\mathcal{P}$ if it is subset-minimal among the interpretations that satisfy $\mathcal{P}^J$.

Semantics for Program Updates

Causal rejection semantics for program updates assign stable models to sequences of non-disjunctive programs. Their formal introduction requires a few additional concepts first.

A *default literal* is a formula of the form $\mathit{not}\, l$ where $l \in \mathcal{L}$. The set of all *literals* $\mathcal{L}^*$ consists of all objective and default literals. The *complementary literal* to a literal L is denoted by $\overline{L}$ and defined as follows: for any objective literal l, $\overline{l} = \mathit{not}\, l$ and $\overline{\mathit{not}\, l} = l$.

We say that a rule is *disjunctive* if its body is either $\top$ or a conjunction of literals and its head is a disjunction of literals; *non-disjunctive* if its body is either $\top$ or a conjunction of literals and its head is a literal. A program is *disjunctive* if all its rules are disjunctive; *non-disjunctive* if all its rules are non-disjunctive.

Now we can proceed with defining the JU-semantics [Leite and Pereira, 1998] and UA-semantics [Eiter *et al.*, 2002] for program updates. Both semantics can be seen as formal embodiments of the *causal rejection principle* which states that every rule must remain in effect as long as it is not contradicted by a newer rule. We define them in their generalised forms, allowing for default negation in heads of rules [Leite, 2003].[1]

A *dynamic logic program* (DLP) is a finite sequence of non-disjunctive programs. Given a DLP $\boldsymbol{P}$, we denote by $\mathsf{all}(\boldsymbol{P})$ the set of all rules belonging to the programs in $\boldsymbol{P}$.[2]

A conflict between rules occurs when the head literal of one rule is the default or strong negation of the head literal of the other rule. Similarly as in [Leite, 2003], we consider the conflicts between an objective literal and its default negation as *primary* while conflicts between objective literals are translated into a primary conflict by expanding the DLP accordingly. This expansion employs the *coherence principle*: when an objective literal l is derived, its complement $-l$ cannot be concurrently true and thus $not\ -l$ must be true. Formally, for a DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$, the *expanded version of* $\boldsymbol{P}$ is the DLP $\boldsymbol{P}^* = \langle \mathcal{P}_i^* \rangle_{i<n}$ where for every $i < n$,

$$\mathcal{P}_i^* = \mathcal{P}_i \cup \{\, not\ -\mathsf{h}(\pi) \leftarrow \mathsf{b}(\pi) \mid \pi \in \mathcal{P}_i \wedge \mathsf{h}(\pi) \in \mathcal{L} \,\} \ .$$

Furthermore, we say that rules π, σ *are in conflict*, denoted by $\pi \bowtie \sigma$, if $\mathsf{h}(\pi) = \overline{\mathsf{h}(\sigma)}$.

The *JU-semantics* [Leite and Pereira, 1998] defines a set of *rejected rules*, which depends on a stable model candidate, and then verifies that the candidate is indeed a stable model of the remaining rules.

Definition 1 (JU-Semantics [Leite and Pereira, 1998])**.** Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP and J an interpretation. The set of rejected rules $\mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}, J)$ contains all rules $\pi \in \mathcal{P}_i$ such that for some rule $\sigma \in \mathcal{P}_j$ with $j > i$,

$$\pi \bowtie \sigma \text{ and } J \models \mathsf{b}(\sigma) \ .$$

The set $[\![\boldsymbol{P}]\!]_{\mathsf{JU}}$ of *JU-models of a DLP* $\boldsymbol{P}$ consists of all interpretations J such that J is a stable model of the program

$$\mathsf{all}(\boldsymbol{P}^*) \setminus \mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}^*, J) \ .$$

Under the JU-semantics, a rule π is rejected if a more recent rule σ is in conflict with π and the body of σ is satisfied in the stable model candidate J. The only difference in the UA-semantics [Eiter *et al.*, 2002] is that rejected rules are prevented from rejecting other rules:

Definition 2 (UA-Semantics [Eiter *et al.*, 2002])**.** Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP and J an interpretation. The set of rejected rules $\mathsf{rej}_{\mathsf{UA}}(\boldsymbol{P}, J)$ contains all rules $\pi \in \mathcal{P}_i$ such that for some rule $\sigma \in \mathcal{P}_j$ with $j > i$,

$$\sigma \notin \mathsf{rej}_{\mathsf{UA}}(\boldsymbol{P}, J) \text{ and } \pi \bowtie \sigma \text{ and } J \models \mathsf{b}(\sigma) \ .[3]$$

[1]Default negation in heads of rules enhances the expressivity of the formalism as it allows us to express that an objective literal should cease being true, distinguishing this case from the one where the opposite literal must become true.

[2]In order to avoid issues with rules that are repeated in multiple components of a DLP, we assume throughout this paper that every rule is uniquely identified in all set-theoretic operations. This can be formalised by assigning a unique name to each rule and performing operations on names instead of the rules themselves. However, for the sake of simplicity, we leave the technical realisation to the reader.

[3]Note that although this definition is recursive, the defined set is unique. This is because we assume that every rule is uniquely identified and to determine whether a rule from $\mathcal{P}_i$ is rejected, the recursion only refers to rejected rules from programs $\mathcal{P}_j$ with j strictly greater than i. One can thus first find the rejected rules in $\mathcal{P}_{n-1}$ (always $\emptyset$ by the definition), then those in $\mathcal{P}_{n-2}$ and so on until $\mathcal{P}_0$.

The set $[\![\boldsymbol{P}]\!]_{\mathsf{UA}}$ of *UA-models of a DLP* $\boldsymbol{P}$ consists of all interpretations J such that J is a stable model of the program

$$\mathsf{all}(\boldsymbol{P}^*) \setminus \mathsf{rej}_{\mathsf{UA}}(\boldsymbol{P}^*, J) \ .$$

The main difference between these semantics is that the latter is more sensitive to irrelevant updates, such as an update by a tautological rule that cannot indicate any change in the modelled world because it is always satisfied. One example that distinguishes the JU- and UA-semantics is the DLP

$$\boldsymbol{P}_1 = \langle \{\, p \,\}, \{\, \neg p \,\}, \{\, p \leftarrow p \,\} \rangle \qquad (1)$$

for which the only JU-model is $\{\, \neg p \,\}$, while, due to the presence of the last rule, the UA-semantics admits the additional undesired model $\{\, p \,\}$.

3 Condensing into a Nested Program

Now that all technical preliminaries are covered, we can proceed with the definition of a state condensing operator for the JU-semantics, and subsequently also for the UA-semantics. More specifically, we will define binary operators $\oplus_{\mathsf{JU}}$ and $\oplus_{\mathsf{UA}}$ that take an original program and its update as arguments and return the updated program. We naturally generalise any such operator $\oplus$ to a sequence of programs $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ inductively as follows:

$$\bigoplus \langle\rangle = \emptyset \ ,$$

$$\bigoplus \langle \mathcal{P}_i \rangle_{i<n+1} = \left(\bigoplus \langle \mathcal{P}_i \rangle_{i<n} \right) \oplus \mathcal{P}_n$$

The property that $\oplus_{\mathsf{JU}}$ and $\oplus_{\mathsf{UA}}$ must fulfill is that for any DLP $\boldsymbol{P}$, the stable models of the program $\bigoplus_{\mathsf{JU}} \boldsymbol{P}$ are exactly the JU-models of $\boldsymbol{P}$ and, similarly, the stable models of the program $\bigoplus_{\mathsf{UA}} \boldsymbol{P}$ are exactly the UA-models of $\boldsymbol{P}$.

Both $\oplus_{\mathsf{JU}}$ and $\oplus_{\mathsf{UA}}$ are defined by utilising the concept of an *activation formula*, which captures the condition under which literal L is derived by some rule in a program $\mathcal{U}$. Formally, the *activation formula for* L *in* $\mathcal{U}$ is defined as follows:

$$\alpha_{\mathcal{U}}(L) = \bigvee \{\, \mathsf{b}(\pi) \mid \pi \in \mathcal{U} \wedge \mathsf{h}(\pi) = L \,\} \ .[4]$$

The operator $\oplus_{\mathsf{JU}}$ is based on the following simple idea: When updating a program $\mathcal{P}$ by a program $\mathcal{U}$, each rule π from $\mathcal{P}^*$ with literal L in its head must be disabled when some rule from $\mathcal{U}^*$ for the literal complementary to L is activated. This can be achieved by augmenting the body of π with the additional condition $not\ \alpha_{\mathcal{U}^*}(\overline{L})$. Formally:

Definition 3 (Condensing operator $\oplus_{\mathsf{JU}}$)**.** A *JU-rule* is a rule with a single literal in its head and a *JU-program* is any set of JU-rules.

The binary operator $\oplus_{\mathsf{JU}}$ on the set of all JU-programs is defined as follows: Given two JU-programs $\mathcal{P}$ and $\mathcal{U}$, the JU-program $\mathcal{P} \oplus_{\mathsf{JU}} \mathcal{U}$ consists of the following rules:

1. for all $\pi \in \mathcal{P}^*$, the rule

$$\mathsf{h}(\pi) \leftarrow \mathsf{b}(\pi) \wedge not\ \alpha_{\mathcal{U}^*}\!\left(\overline{\mathsf{h}(\pi)}\right) \ ;$$

2. all rules in $\mathcal{U}^*$.

$$\begin{array}{lll}
p \leftarrow not((not\,q \wedge not\,r) \vee s) & & not\,\neg p \\
q \leftarrow p & & not\,\neg q \leftarrow p \\
r \leftarrow not\,\top & & not\,\neg r \\
not\,p \leftarrow not\,q \wedge not\,r \wedge not\,s & & not\,p \leftarrow s \wedge not\,s \\
not\,r \leftarrow not\,r & & \\
p \leftarrow s & not\,\neg p \leftarrow s & s \\
r \leftarrow r & not\,\neg r \leftarrow r & not\,\neg s
\end{array}$$

Figure 1: The program $\bigoplus_{\mathsf{JU}} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$

The following example illustrates the relationship between the JU-semantics and the condensing operator $\oplus_{\mathsf{JU}}$, while the subsequent theorem pinpoints the fact that it actually holds in general.

Example 4. Suppose that programs $\mathcal{P}, \mathcal{U}$ are as follows:

$$\begin{array}{llll}
\mathcal{P}: & p & \mathcal{U}: & not\,p \leftarrow not\,q \wedge not\,r \\
& q \leftarrow p & & not\,p \leftarrow s \\
& r & & not\,r
\end{array}$$

In addition to the rules from $\mathcal{U}^*$ (which in this case coincides with $\mathcal{U}$), the program $\bigoplus_{\mathsf{JU}} \langle \mathcal{P}, \mathcal{U} \rangle$ contains the following six rules:[5]

$$\begin{array}{ll}
p \leftarrow not((not\,q \wedge not\,r) \vee s) & not\,\neg p \\
q \leftarrow p & not\,\neg q \leftarrow p \\
r \leftarrow not\,\top & not\,\neg r
\end{array} \qquad (2)$$

Also, the two stable models of $\bigoplus_{\mathsf{JU}} \langle \mathcal{P}, \mathcal{U} \rangle$ are $\emptyset$ and $\{ p, q \}$, and they coincide with the JU-models of the DLP $\langle \mathcal{P}, \mathcal{U} \rangle$.

Now consider that $\mathcal{V}$ contains the following three rules:

$$\mathcal{V}: \quad p \leftarrow s \qquad r \leftarrow r \qquad s$$

The program $\bigoplus_{\mathsf{JU}} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$ contains the following rules:

- the six rules listed in (2);
- the following rules, which originate in $\mathcal{U}$:

$$not\,p \leftarrow not\,q \wedge not\,r \wedge not\,s \enspace ,$$
$$not\,p \leftarrow s \wedge not\,s \enspace ,$$
$$not\,r \leftarrow not\,r \enspace ;$$

- all rules in $\mathcal{V}^*$.

The full program is also shown in Fig. 1. Its unique stable model is $\{ p, q, s \}$, which coincides with the unique JU-model of $\langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$.

Theorem 5 (State condensing using $\oplus_{\mathsf{JU}}$)**.** *Let $\boldsymbol{P}$ be a DLP. An interpretation J is a JU-model of $\boldsymbol{P}$ if and only if it is a stable model of $\bigoplus_{\mathsf{JU}} \boldsymbol{P}$.*

Thus, the single nested program $\mathcal{P} = \bigoplus_{\mathsf{JU}} \boldsymbol{P}$ faithfully represents $\boldsymbol{P}$ in the sense that it captures all information from $\boldsymbol{P}$ that is relevant w.r.t. the JU-semantics: not only does it have the required stable models, but any further updates of $\boldsymbol{P}$ can be performed directly on $\mathcal{P}$ by applying the operator $\oplus_{\mathsf{JU}}$ and the obtained stable models will be the same.

[4]Note that $\bigvee \emptyset$ is simply $\bot$.

[5]For the sake of readability, we omit the conjuncts $\top$ and $not\,\bot$ in rule bodies.

$$\begin{array}{lll}
p \vee not\,q \leftarrow not\,s & & not\,\neg p \\
p \vee not\,r \leftarrow not\,s & & not\,\neg q \leftarrow p \\
q \leftarrow p & & not\,\neg r \\
not\,p \leftarrow not\,q \wedge not\,r \wedge not\,s & & not\,p \leftarrow s \wedge not\,s \\
not\,r \leftarrow not\,r & & \\
p \leftarrow s & not\,\neg p \leftarrow s & s \\
r \leftarrow r & not\,\neg r \leftarrow r & not\,\neg s
\end{array}$$

Figure 2: The program $\bigoplus_{\mathsf{JU}}^{\vee} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$

In case of the UA-semantics, a very similar operator can be used to obtain the same result. The only difference is that for a DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$, in addition to the rules in $\bigoplus_{\mathsf{JU}} \boldsymbol{P}$, the program $\bigoplus_{\mathsf{UA}} \boldsymbol{P}$ will also contain the choice rule

$$\mathsf{h}(\pi) \vee \overline{\mathsf{h}(\pi)} \leftarrow \mathsf{b}(\pi)$$

for all $i < n$ and every rule $\pi \in \mathcal{P}_i$ with $\mathsf{h}(\pi) \in \mathcal{L}$. Intuitively, these additional rules account for the differences in the definition of $\mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}, J)$ and $\mathsf{rej}_{\mathsf{UA}}(\boldsymbol{P}, J)$. They make sure that no rule is ever completely eliminated due to extra conditions added to its body, but stays partially in effect by generating alternative worlds for the objective literal in its head: one where it is satisfied and one where it is not. Essentially, this means that whenever the original body of the rule is satisfied, its head cannot be assumed false by default.

Definition 6 (Condensing operator $\oplus_{\mathsf{UA}}$)**.** A *UA-rule* is a rule with either a single literal or a disjunction of two literals L and $\overline{L}$ in its head, and a *UA-program* is any set of UA-rules.

The binary operator $\oplus_{\mathsf{UA}}$ on the set of all UA-programs is defined as follows: Given two UA-programs $\mathcal{P}$ and $\mathcal{U}$, the UA-program $\mathcal{P} \oplus_{\mathsf{UA}} \mathcal{U}$ consists of the following rules:

1. for all $\pi \in \mathcal{P}^*$ with $\mathsf{h}(\pi) \in \mathcal{L}^*$, the rule
$$\mathsf{h}(\pi) \leftarrow \mathsf{b}(\pi) \wedge not\,\alpha_{\mathcal{U}^*}\Big(\overline{\mathsf{h}(\pi)}\Big) \enspace ;$$
2. all $\pi \in \mathcal{P}$ such that $\mathsf{h}(\pi)$ is of the form $L \vee \overline{L}$;
3. for all $\pi \in \mathcal{U}$ with $\mathsf{h}(\pi) \in \mathcal{L}$, the rule
$$\mathsf{h}(\pi) \vee \overline{\mathsf{h}(\pi)} \leftarrow \mathsf{b}(\pi) \enspace ;$$
4. all rules in $\mathcal{U}^*$.

Similarly as with the operator $\oplus_{\mathsf{JU}}$, we first illustrate the inner workings of operator $\oplus_{\mathsf{UA}}$ on an example, highlighting its difference from $\oplus_{\mathsf{JU}}$ as well as its relationship with the UA-semantics, and then formally establish this relationship in the general case.

Example 7. Consider again the programs $\mathcal{P}, \mathcal{U}$ and $\mathcal{V}$ from Example 4. Except for the rules in $\bigoplus_{\mathsf{JU}} \langle \mathcal{P}, \mathcal{U} \rangle$, the program $\bigoplus_{\mathsf{UA}} \langle \mathcal{P}, \mathcal{U} \rangle$ also contains the following three rules:

$$p \vee not\,p \qquad q \vee not\,q \leftarrow p \qquad r \vee not\,r \qquad (3)$$

Nevertheless, its stable models, $\emptyset$ and $\{ p, q \}$, are the same as those of $\bigoplus_{\mathsf{JU}} \langle \mathcal{P}, \mathcal{U} \rangle$, and they also coincide with the UA-models of the DLP $\langle \mathcal{P}, \mathcal{U} \rangle$.

$p \leftarrow not((not\,q \wedge not\,r) \vee s)$		$not\,\neg p$
$q \leftarrow p$		$not\,\neg q \leftarrow p$
$r \leftarrow not\,\top$		$not\,\neg r$
$p \vee not\,p$	$q \vee not\,q \leftarrow p$	$r \vee not\,r$
$not\,p \leftarrow not\,q \wedge not\,r \wedge not\,s$		$not\,p \leftarrow s \wedge not\,s$
$not\,r \leftarrow not\,r$		
$p \leftarrow s$	$not\,\neg p \leftarrow s$	s
$r \leftarrow r$	$not\,\neg r \leftarrow r$	$not\,\neg s$
$p \vee not\,p \leftarrow s$	$r \vee not\,r \leftarrow r$	$s \vee not\,s$

Figure 3: The program $\bigoplus_{\mathsf{UA}}\langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$

$p \leftarrow q \wedge not\,s$		$not\,\neg p$
$p \leftarrow r \wedge not\,s$		$not\,\neg q \leftarrow p$
$q \leftarrow p$		$not\,\neg r$
$p \vee not\,p$	$q \vee not\,q \leftarrow p$	$r \vee not\,r$
$not\,p \leftarrow not\,q \wedge not\,r \wedge not\,s$		$not\,p \leftarrow s \wedge not\,s$
$not\,r \leftarrow not\,r$		
$p \leftarrow s$	$not\,\neg p \leftarrow s$	s
$r \leftarrow r$	$not\,\neg r \leftarrow r$	$not\,\neg s$
$p \vee not\,p \leftarrow s$	$r \vee not\,r \leftarrow r$	$s \vee not\,s$

Figure 4: The program $\bigoplus^{\vee}_{\mathsf{UA}}\langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$

The situation is more interesting after $\mathcal{V}$ is added to the sequence. The resulting program $\bigoplus_{\mathsf{UA}}\langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$ contains all the rules in $\bigoplus_{\mathsf{JU}}\langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$, the three rules listed in (3) and, additionally, the following three rules:

$$p \vee not\,p \leftarrow s \qquad r \vee not\,r \leftarrow r \qquad s \vee not\,s$$

For reference, the entire program is also listed in Fig. 3. Due to the last rule in (3), it has the extra stable model $\{ p, q, r, s \}$ in addition to the unique stable model $\{ p, q, s \}$ of $\bigoplus_{\mathsf{JU}}\langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$. These stable models also coincide with the UA-models of the DLP $\langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$.

Theorem 8 (State condensing using $\oplus_{\mathsf{UA}}$)**.** *Let $\boldsymbol{P}$ be a DLP. An interpretation J is a UA-model of $\boldsymbol{P}$ if and only if it is a stable model of $\bigoplus_{\mathsf{UA}} \boldsymbol{P}$.*

As with the JU-semantics, the significance of this theorem is in that the operator $\oplus_{\mathsf{UA}}$ provides a full characterisation of the UA-semantics: It can condense any given DLP $\boldsymbol{P}$ into a single nested program $\mathcal{P}$ such that the stable models of $\mathcal{P}$ coincide with the UA-models of $\boldsymbol{P}$ and any further updates on top of $\boldsymbol{P}$ can be equivalently performed on $\mathcal{P}$ using $\oplus_{\mathsf{UA}}$.

4 Condensing into a Disjunctive Program

The condensing operators defined in the previous section can be further modified in order to produce a program that meets certain additional requirements. In the present section we show that nested expressions can be completely eliminated from the resulting program while still preserving the same tight relationship with the original program update semantics. Thus, we introduce a new pair of operators, $\oplus^{\vee}_{\mathsf{JU}}$ and $\oplus^{\vee}_{\mathsf{UA}}$, that operate on disjunctive programs with default negation in heads of rules. Note that due to the non-minimality of JU- and UA-models for certain DLPs, disjunctive programs *without* default negation in heads of rules would already be insufficient for this purpose.

The ideas underlying the new operators are fairly straightforward. Essentially, nested expressions are introduced into the resulting programs only by the negations of activation formulas in their bodies, so these are the parts of rules that need to be translated into conjunctions in bodies and disjunctions in heads of rules. In particular, by utilising De Morgan's law and distributivity of conjunction over disjunction, we can obtain a new formula, a disjunction of conjunctions of default literals and double-negated objective literals, that is strongly equivalent to the original formula. For instance, in case of the first rule in (2), we can equivalently write the condition $not((not\,q \wedge not\,r) \vee s)$ as

$$(not\,not\,q \wedge not\,s) \vee (not\,not\,r \wedge not\,s) \enspace .$$

Then it suffices to break up the resulting rule into multiple rules, each with one of the disjuncts of this formula in the body, and remove one of the negations from each double-negated literals and "move" it into the head of the newly constructed rule. In case of the first rule in (2), the result would be the following two disjunctive rules:

$$p \vee not\,q \leftarrow not\,s \qquad p \vee not\,r \leftarrow not\,s \tag{4}$$

Formally, we call the set of literals, without double negation, within each of the disjuncts described above a *blocking set*. Given a non-disjunctive program $\mathcal{U}$ and a literal L, if the formula $\alpha_{\mathcal{U}}(L)$ contains $\top$ as one of its disjuncts, then there is no blocking set for L in $\mathcal{U}$. Otherwise, suppose that $\alpha_{\mathcal{U}}(L)$ is the formula

$$(L^1_1 \wedge \dots \wedge L^1_{k_1}) \vee \dots \vee (L^n_1 \wedge \dots \wedge L^n_{k_n}) \enspace .$$

A *blocking set for L in $\mathcal{U}$* is any set of literals

$$\left\{ \overline{L^1_{i_1}}, \dots, \overline{L^n_{i_n}} \right\}$$

where $1 \leq i_j \leq k_j$ for every j with $1 \leq j \leq n$. We denote the set of all blocking sets for L in $\mathcal{U}$ by $\beta_{\mathcal{U}}(L)$. Also, for any set of literals S,

$$\begin{aligned} S^+ &= \{ l \in \mathcal{L} \mid l \in S \} \enspace , \\ S^- &= \{ l \in \mathcal{L} \mid not\,l \in S \} \enspace , \\ not\,S &= \{ not\,l \mid l \in S \} \enspace . \end{aligned}$$

Each nested rule

$$\mathsf{h}(\pi) \leftarrow \mathsf{b}(\pi) \wedge not\,\alpha_{\mathcal{U}^*}(\overline{\mathsf{h}(\pi)}) \enspace ,$$

can thus be replaced by a set of disjunctive rules

$$\mathsf{h}(\pi) \vee \bigvee not\,S^+ \leftarrow \mathsf{b}(\pi) \wedge \bigwedge not\,S^-$$

where $S \in \beta_{\mathcal{U}^*}\left(\overline{\mathsf{h}(\pi)}\right)$. Furthermore, when $\mathsf{h}(\pi)$ is a default literal, it is more convenient to move the new default literals from the head into the body since this operation preserves stable models [Inoue and Sakama, 1998] and makes it easier to pinpoint the original head literal in the rule. This leads us to the following definition of $\oplus^{\vee}_{\mathsf{JU}}$:

Definition 9 (Condensing operator $\oplus_{\mathsf{JU}}^{\vee}$)**.** The binary operator $\oplus_{\mathsf{JU}}^{\vee}$ on the set of all disjunctive programs is defined as follows: Given two disjunctive programs $\mathcal{P}$ and $\mathcal{U}$, the disjunctive program $\mathcal{P} \oplus_{\mathsf{JU}}^{\vee} \mathcal{U}$ consists of the following rules:

1. for all $\pi \in \mathcal{P}^*$ such that for some $l \in \mathcal{L}$, either $\mathsf{h}(\pi) = l$ or l is the unique non-negated disjunct of $\mathsf{h}(\pi)$, and all $S \in \beta_{\mathcal{U}^*}(\mathit{not}\, l)$, the rule
$$\mathsf{h}(\pi) \vee \bigvee \mathit{not}\, S^+ \leftarrow \mathsf{b}(\pi) \wedge \bigwedge \mathit{not}\, S^- \quad ;$$
2. for all $\pi \in \mathcal{P}^*$ such that for some $l \in \mathcal{L}$, $\mathsf{h}(\pi) = \mathit{not}\, l$, and all $S \in \beta_{\mathcal{U}^*}(l)$, the rule
$$\mathsf{h}(\pi) \leftarrow \mathsf{b}(\pi) \wedge \bigwedge S^+ \wedge \bigwedge \mathit{not}\, S^- \quad ;$$
3. all rules in $\mathcal{U}^*$.

If we consider the programs $\mathcal{P}, \mathcal{U}$ and $\mathcal{V}$ from Example 4, the disjunctive program $\bigoplus_{\mathsf{JU}}^{\vee} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$ differs from the nested one in two points: 1) its only nested rule is turned into the two disjunctive rules in (4), and 2) the rule $r \leftarrow \mathit{not}\, \top$ is completely eliminated because its body is never satisfied and, formally, there is no blocking set for $\mathit{not}\, r$ in $\mathcal{U}$. For comparison, Fig. 2 includes a listing of the rules in $\bigoplus_{\mathsf{JU}}^{\vee} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$.

The operator $\oplus_{\mathsf{JU}}^{\vee}$ preserves the main property of $\oplus_{\mathsf{JU}}$.

Theorem 10 (State condensing using $\oplus_{\mathsf{JU}}^{\vee}$)**.** *Let $\boldsymbol{P}$ be a DLP. An interpretation J is a JU-model of $\boldsymbol{P}$ if and only if it is a stable model of $\bigoplus_{\mathsf{JU}}^{\vee} \boldsymbol{P}$.*

As for the UA-semantics, analogical modifications can be applied in the definition of $\oplus_{\mathsf{UA}}$ to obtain an operator that produces a disjunctive program. Furthermore, due to the additional choice rules included in the result, the rules can be further simplified, when compared to the rules produced by $\oplus_{\mathsf{JU}}^{\vee}$. In particular, the first group of rules can be treated the same way as the second, leading to the following definition of $\oplus_{\mathsf{UA}}^{\vee}$:

Definition 11 (Condensing operator $\oplus_{\mathsf{UA}}^{\vee}$)**.** The binary operator $\oplus_{\mathsf{UA}}^{\vee}$ on the set of all disjunctive programs is defined as follows: Given two disjunctive programs $\mathcal{P}$ and $\mathcal{U}$, the disjunctive program $\mathcal{P} \oplus_{\mathsf{UA}}^{\vee} \mathcal{U}$ consists of the following rules:

1. for all $\pi \in \mathcal{P}^*$ with $\mathsf{h}(\pi) \in \mathcal{L}^*$ and all $S \in \beta_{\mathcal{U}^*}\left(\overline{\mathsf{h}(\pi)}\right)$, the rule
$$\mathsf{h}(\pi) \leftarrow \mathsf{b}(\pi) \wedge \bigwedge S^+ \wedge \bigwedge \mathit{not}\, S^- \quad ;$$
2. all $\pi \in \mathcal{P}$ such that $\mathsf{h}(\pi)$ is of the form $L \vee \overline{L}$;
3. for all $\pi \in \mathcal{U}$ with $\mathsf{h}(\pi) \in \mathcal{L}$, the rule
$$\mathsf{h}(\pi) \vee \overline{\mathsf{h}(\pi)} \leftarrow \mathsf{b}(\pi) \quad ;$$
4. all rules in $\mathcal{U}^*$.

Returning to the programs $\mathcal{P}$, $\mathcal{U}$ and $\mathcal{V}$ from Example 4, the differences between programs $\bigoplus_{\mathsf{UA}}^{\vee} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$ and $\bigoplus_{\mathsf{UA}} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$ are similar to those between $\bigoplus_{\mathsf{JU}}^{\vee} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$ and $\bigoplus_{\mathsf{JU}} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$ and the rules of $\bigoplus_{\mathsf{UA}}^{\vee} \langle \mathcal{P}, \mathcal{U}, \mathcal{V} \rangle$ are listed in Fig. 4. Also, $\oplus_{\mathsf{UA}}^{\vee}$ preserves the main property of $\oplus_{\mathsf{UA}}$.

Theorem 12 (State condensing using $\oplus_{\mathsf{UA}}^{\vee}$)**.** *Let $\boldsymbol{P}$ be a DLP. An interpretation J is a UA-model of $\boldsymbol{P}$ if and only if it is a stable model of $\bigoplus_{\mathsf{UA}}^{\vee} \boldsymbol{P}$.*

Although operators $\oplus_{\mathsf{JU}}^{\vee}$ and $\oplus_{\mathsf{UA}}^{\vee}$ eliminate the necessity for using nested rules to condense a DLP into a single program, this comes at a cost. Namely, the size of the nested program resulting from applying operators $\oplus_{\mathsf{JU}}$ and $\oplus_{\mathsf{UA}}$ is always linear in size of the argument programs, while in case of $\oplus_{\mathsf{JU}}^{\vee}$ and $\oplus_{\mathsf{UA}}^{\vee}$, the resulting program can be exponentially larger. Furthermore, Figs. 1 to 4 suggest that the representations produced by $\oplus_{\mathsf{JU}}^{\vee}$ and $\oplus_{\mathsf{UA}}^{\vee}$ will be less faithful to the form of the rules in the original programs, and thus less readable. This indicates that the nested program is more suitable as a way to store the condensed program, both in terms of space and readability. Additionally, in order to find its stable models, a more efficient translation can be used that utilises additional atoms to prevent the exponential explosion. However, such a translation will no longer be equivalent to the original program sequence w.r.t. performing further updates.

5 Discussion

Defining update operators for answer-set programs that actually produce an answer-set program written in the same alphabet has been a long enduring problem, also known as *state condensing*, i.e., condensing a sequence of answer-set programs – interpreted as updates – into a single answer set program. Existing semantics typically proceed by characterising the models of the update and, at most, either describe a set of answer-set programs that could represent the update, instead of only one, or produce an answer-set program written in a language extended with a considerable amount of new atoms, making it difficult to understand and to further update.

Part of the problems emerge because some semantics employ complex mechanisms such as preferences or other minimality criteria that make it impossible to encode the result in a single answer-set program, while others have model-theoretic characterisations that assign non-minimal models to certain update sequences, and it is well known that stable models of non-disjunctive answer-set programs are minimal.

In this paper we addressed and solved the problem of state condensing for two of the most relevant semantics for updates, by resorting to more expressive classes of answer-set programs, namely nested and disjunctive. In all four cases, two for each semantics using both classes of answer-set programs, the resulting program is written with the same alphabet and is ready to be further updated. We have illustrated with some examples that the resulting programs written using nested answer-set programming are perhaps more readable than those written using disjunctive answer-set programs, in the sense that they more closely match the intuitions underlying the semantics for updates that we consider.

A similar approach to answer-set program updates has previously been considered by Osorio and Cuevas [2007] but the possibility of performing iterated updates has not been explored. This work is also related to our work on semantic rule updates [Slota and Leite, 2010; 2012a] where state condensing is taken as one of the basic characteristics of rule updates, making it possible to relate them more directly with classical belief update principles and semantics [Slota and Leite, 2012b]. In the future, it would also be interesting to look for condensing operators for other rule update semantics.

References

[Alferes *et al.*, 2000] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming*, 45(1-3):43–70, September/October 2000.

[Alferes *et al.*, 2005] José Júlio Alferes, Federico Banti, Antonio Brogi, and João Alexandre Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1):7–32, 2005.

[Delgrande *et al.*, 2007] James P. Delgrande, Torsten Schaub, and Hans Tompits. A preference-based framework for updating logic programs. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, volume 4483 of *Lecture Notes in Computer Science*, pages 71–83, Tempe, AZ, USA, May 15-17 2007. Springer.

[Eiter *et al.*, 2002] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming (TPLP)*, 2(6):721–777, 2002.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988)*, pages 1070–1080, Seattle, Washington, August 15-19 1988. MIT Press.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–385, 1991.

[Homola, 2004] Martin Homola. Dynamic logic programming: Various semantics are equal on acyclic programs. In João Alexandre Leite and Paolo Torroni, editors, *Proceedings of the 5th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA V)*, volume 3487 of *Lecture Notes in Computer Science*, pages 78–95, Lisbon, Portugal, September 29-30 2004. Springer.

[Inoue and Sakama, 1998] Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.

[Krümpelmann, 2012] Patrick Krümpelmann. Dependency semantics for sequences of extended logic programs. *Logic Journal of the IGPL*, 20(5):943–966, 2012.

[Leite and Pereira, 1998] João Alexandre Leite and Luís Moniz Pereira. Generalizing updates: From models to programs. In Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusinski, editors, *Proceedings of the 3rd International Workshop on Logic Programming and Knowledge Representation (LPKR '97), October 17, 1997, Port Jefferson, New York, USA*, volume 1471 of *Lecture Notes in Computer Science*, pages 224–246. Springer, October 1998.

[Leite, 2003] João Alexandre Leite. *Evolving Knowledge Bases*, volume 81 of *Frontiers of Artificial Intelligence and Applications, xviii + 307 p. Hardcover*. IOS Press, 2003.

[Osorio and Cuevas, 2007] Mauricio Osorio and Víctor Cuevas. Updates in answer set programming: An approach based on basic structural properties. *Theory and Practice of Logic Programming*, 7(4):451–479, 2007.

[Sakama and Inoue, 2003] Chiaki Sakama and Katsumi Inoue. An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming (TPLP)*, 3(6):671–713, 2003.

[Šefránek, 2006] Ján Šefránek. Irrelevant updates and nonmonotonic assumptions. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA 2006)*, volume 4160 of *Lecture Notes in Computer Science*, pages 426–438, Liverpool, UK, September 13-15 2006. Springer.

[Šefránek, 2011] Ján Šefránek. Static and dynamic semantics: Preliminary report. *Mexican International Conference on Artificial Intelligence*, pages 36–42, 2011.

[Slota and Leite, 2010] Martin Slota and João Leite. On semantic update operators for answer-set programs. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 957–962, Lisbon, Portugal, August 16-20 2010. IOS Press.

[Slota and Leite, 2012a] Martin Slota and João Leite. Robust equivalence models for semantic updates of answer-set programs. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 158–168, Rome, Italy, June 10-14 2012. AAAI Press.

[Slota and Leite, 2012b] Martin Slota and João Leite. A unifying perspective on knowledge updates. In Luis Fariñas del Cerro, Andreas Herzig, and Jérôme Mengin, editors, *Proceedings of the 13th European Conference on Logics in Artificial Intelligence (JELIA 2012)*, volume 7519 of *Logics in Artificial Intelligence (LNAI)*, pages 372–384, Toulouse, France, September 26-28 2012. Springer.

[Turner, 2003] Hudson Turner. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming (TPLP)*, 3(4-5):609–622, 2003.

[Zhang, 2006] Yan Zhang. Logic program-based updates. *ACM Transactions on Computational Logic*, 7(3):421–472, 2006.