ASP Solving for Expanding Universes ⋆

M. Gebser[1,3], T. Janhunen[1], H. Jost[3], R. Kaminski[3], and T. Schaub[2,3⋆⋆]

[1]Aalto University Helsinki [2]INRIA Rennes [3]University of Potsdam

Abstract. Over the last years, Answer Set Programming has significantly extended its range of applicability, and moved beyond solving static problems to dynamic ones, even in online environments. However, its nonmonotonic nature as well as its upstream instantiation process impede a seamless integration of new objects into its reasoning process, which is crucial in dynamic domains such as logistics or robotics. We address this problem and introduce a simple approach to successively incorporating new information into ASP systems. Our approach rests upon a translation of logic programs and thus refrains from any dedicated algorithms. We prove its modularity as regards the addition of new information and show its soundness and completeness. We apply our methodology to two domains of the Fifth ASP Competition and evaluate traditional one-shot and incremental multi-shot solving approaches.

1 Introduction

Answer Set Programming (ASP; [1]) is deeply rooted in the paradigm of nonmonotonic reasoning. That is, conclusions can be invalidated upon the arrival of new information. Unfortunately, this carries over to computationally relevant characterizations, involving completion and loop formulas, and thus extends to the data structures capturing "nonmonotonicity" in modern ASP solvers. Hence, when solving in dynamic domains like logistics or robotics, the emergence of new properties or even new objects cannot be accounted for in a modular way, since the existing structures become invalidated. This is different from monotonic (instantiation-based) approaches, like the original DPLL procedure [2], where new objects can be modularly incorporated by adding the instantiations involving them. In fact, incremental satisfiability solving has been successfully applied in domains like finite model finding [3], model checking [4], and planning [5], yet relying on application-specific instantiators rather than general-purpose grounding.

Incremental instantiation was so far neglected in ASP since traditional systems were designed for one-shot solving and thus needed to be relaunched whenever the problem specification changed. This is clearly disadvantageous in highly dynamic domains like logistics or robotics. Although new generation ASP systems, like *clingo* 4 [6], allow for multi-shot solving and thus abolish the need for relaunching, there is yet no principled way of modularly extending a problem specification upon the arrival of new objects.

⋆ This work was funded by DFG (SCHA 550/9), the Finnish Centre of Excellence in Computational Inference Research (COIN) supported by the Academy of Finland (AoF) under grant 251170, as well as DAAD and AoF under joint project 57071677/279121. A draft version with proofs is available at `http://www.cs.uni-potsdam.de/wv/publications/`.

⋆⋆ Affiliated with Simon Fraser University, Canada, and IIIS Griffith University, Australia.

In what follows, we address a rather general variant of this problem by allowing new information to successively expand the (Herbrand) universe. Our approach rests upon a simple translation of logic programs and thus refrains from dedicated algorithms (though it is only meaningful in the context of multi-shot ASP solving). We prove the modularity of our approach as regards the addition of new information and show its soundness and completeness. Finally, we illustrate our methodology, evaluate the resulting performance of solving approaches, and discuss related work.

2 Background

A signature $(\mathcal{P}, \mathcal{C}, \mathcal{V})$ consists of a set $\mathcal{P}$ of *predicate symbols*, a set $\mathcal{C}$ of *constant symbols*, also called Herbrand universe, and a set $\mathcal{V}$ of *variable symbols*; we usually omit the designation "symbol" for simplicity. The members of $\mathcal{C} \cup \mathcal{V}$ are *terms*. Given a predicate $p \in \mathcal{P}$ of arity n, also denoted as p/n, along with terms $t_1, \ldots, t_n$, $p(t_1, \ldots, t_n)$ is an *atom* over p/n. An atom a and $\sim a$ are (positive or negative, respectively) *literals*, where '$\sim$' stands for default negation; we sometimes (ab)use the same terminology for classical literals a and $\neg a$. Given a set $B = \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\}$ of literals, $B^+ = \{a_1, \ldots, a_m\}$ and $B^- = \{a_{m+1}, \ldots, a_n\}$ denote the atoms occurring positively or negatively in B. A *logic program* R over $(\mathcal{P}, \mathcal{C}, \mathcal{V})$ is a set of *rules* $r = a \leftarrow B$, where a is an atom and B is a set of literals; if $B = \emptyset$, r is also called a *fact*. By $head(r) = a$ and $body(r) = B$, we refer to the *head* or *body* of r, respectively. We extend this notation to R by letting $head(R) = \{head(r) \mid r \in R\}$.

We denote the set of variables occurring in an atom a by $var(a)$. The variables in a rule r are $var(r) = var(head(r)) \cup \bigcup_{a \in body(r)^+ \cup body(r)^-} var(a)$. An atom, rule, or program is *non-ground* if it includes some variable, and *ground* otherwise. By $atom(\mathcal{P}, \mathcal{C}) = \{p(c_1, \ldots, c_n) \mid p/n \in \mathcal{P}, c_1 \in \mathcal{C}, \ldots, c_n \in \mathcal{C}\}$, we refer to the collection of ground atoms over predicates in $\mathcal{P}$. A *ground substitution* for a set V of variables is a mapping $\sigma : V \to \mathcal{C}$, and $\Sigma(V, \mathcal{C})$ denotes the set of all ground substitutions for V. The *instance* $a\sigma$ (or $r\sigma$) of an atom a (or a rule r) is obtained by substituting occurrences of variables in V according to σ. The *ground instantiation* of a program R is the set $grd(R, \mathcal{C}) = \{r\sigma \mid r \in R, \sigma \in \Sigma(var(r), \mathcal{C})\}$ of ground rules.

An *interpretation* $I \subseteq atom(\mathcal{P}, \mathcal{C})$ is a *supported model* [7] of a program R if $I = \{head(r) \mid r \in grd(R, \mathcal{C}), body(r)^+ \subseteq I, body(r)^- \cap I = \emptyset\}$. Moreover, I is a *stable model* [8] of R if I is a $\subseteq$-minimal (supported) model of the reduct $\{head(r) \leftarrow body(r)^+ \mid r \in grd(R, \mathcal{C}), body(r)^- \cap I = \emptyset\}$. Any stable model of R is a supported model of R as well, while the converse does not hold in general [9].

Supported and stable models can also be characterized in terms of classical models. To this end, given a set B of literals, let $bf(B) = (\bigwedge_{a \in B^+} a) \wedge (\bigwedge_{a \in B^-} \neg a)$ denote the *body formula* for B. Moreover, let $rf(r) = bf(body(r)) \rightarrow head(r)$ be the *rule formula* for a rule r. Then, we associate a ground logic program R with the set $RF(R) = \{rf(r) \mid r \in R\}$ of rule formulas. Given some ground atom a, the *completion formula* for a relative to R is $cf(R, a) = a \rightarrow \bigvee_{r \in R, head(r) = a} bf(body(r))$. For a set $\mathcal{A}$ of ground atoms, $CF(R, \mathcal{A}) = \{cf(R, a) \mid a \in \mathcal{A}\}$ denotes the corresponding set of completion formulas. The theory $RF(R) \cup CF(R, atom(\mathcal{P}, \mathcal{C}))$ is also known as *Clark's completion* [10], and its classical models coincide with the supported models of R. In

order to extend the correspondence to stable models, for a set L of ground atoms, let $supp(R, L) = \{body(r) \mid r \in R, head(r) \in L, body(r)^+ \cap L = \emptyset\}$ denote the *external supports* for L [11]. Then, $lf(R, a, L) = a \rightarrow \bigvee_{B \in supp(R,L)} bf(B)$ is the *loop formula* for $a \in L$ relative to R [12]. Further distinguishing two sets $\mathcal{A}$ and $\mathcal{B}$ of ground atoms, we let $LF(R, \mathcal{A}, \mathcal{B}) = \{lf(R, a, L) \mid L \subseteq \mathcal{A} \cup \mathcal{B}, a \in L \cap \mathcal{A}\}$ be the corresponding set of loop formulas. Note that $LF(R, atom(\mathcal{P}, \mathcal{C}), \emptyset) \models CF(R, atom(\mathcal{P}, \mathcal{C}))$, and as shown in [11, 12], the classical models of $RF(R) \cup LF(R, atom(\mathcal{P}, \mathcal{C}), \emptyset)$ match the stable models of R. Thus, when $\mathcal{A} = atom(\mathcal{P}, \mathcal{C})$ and $\mathcal{B} = \emptyset$, $LF(R, \mathcal{A}, \mathcal{B})$ yields the same as corresponding concepts from the literature, but we use $\mathcal{A}$ and $\mathcal{B}$ below to control the set $\mathcal{A}$ of atoms whose derivability is expressed by particular loop formulas.

3 Expanding Logic Programs

As in Datalog [13], we consider signatures $(\mathcal{P}_E \cup \mathcal{P}_I, \mathcal{C}, \mathcal{V})$ such that $\mathcal{P}_E \cap \mathcal{P}_I = \emptyset$. The part $\mathcal{P}_E$ includes *extensional* predicates provided by facts, while the *intensional* predicates in $\mathcal{P}_I$ are defined by rules. We thus deal with programs $F \cup R$ composed of (ground) facts F over $(\mathcal{P}_E, \mathcal{C}, \emptyset)$ and (non-ground) rules R over $(\mathcal{P}_E \cup \mathcal{P}_I, \emptyset, \mathcal{V})$ such that $\{p/n \mid p(X_1, \ldots, X_n) \in head(R)\} \subseteq \mathcal{P}_I$.

Example 1. For $\mathcal{P}_E = \{cs/1, st/1, in/2\}$ (for courses and students) and $\mathcal{P}_I = \{ok/1, ko/1\}$, the following non-ground rules R define the intensional predicates in $\mathcal{P}_I$:

$$ok(C) \leftarrow cs(C), st(S), in(S, C) \tag{1}$$
$$ko(C) \leftarrow cs(C), {\sim}ok(C) \tag{2}$$

Moreover, consider facts F over the extensional predicates in $\mathcal{P}_E$ and $\mathcal{C} = \{c_1, c_2, s_1, s_2\}$ as follows:

$$\begin{array}{lll} cs(c_1), & st(s_1), & in(s_1, c_1), \\ cs(c_2), & st(s_2), & in(s_2, c_1). \end{array}$$

The atoms over intensional predicates in $\mathcal{P}_I$ in the (unique) stable model of $F \cup R$ are $ok(c_1)$ and $ko(c_2)$. □

To characterize supported and stable models in terms of classical models relative to facts F over $(\mathcal{P}_E, \mathcal{C}, \emptyset)$, let $E(F, \mathcal{P}_E, \mathcal{C}) = F \cup \{\neg a \mid a \in atom(\mathcal{P}_E, \mathcal{C}) \setminus F\}$ denote the set of literals fixing atoms over extensional predicates. Then, supported models of $F \cup R$ match classical models of $RF(R') \cup CF(R', atom(\mathcal{P}_I, \mathcal{C})) \cup E(F, \mathcal{P}_E, \mathcal{C})$, where $R' = grd(R, \mathcal{C})$. Similarly, the latter theory augmented with $LF(R', atom(\mathcal{P}_I, \mathcal{C}), \emptyset)$ captures stable models of $F \cup R$.

For expressing the gradual expansion of an (infinite) Herbrand universe $\mathcal{C}$, we consider sequences over constants in $\mathcal{C}$.

Definition 1. *A constant stream over $\mathcal{C}$ is a sequence $(c_i)_{i \geq 1}$ such that $c_{i+1} \in \mathcal{C} \setminus \mathcal{C}_i$ for $i \geq 0$ and $\mathcal{C}_i = \{c_j \mid 1 \leq j \leq i\}$.*

Note that $\mathcal{C}_i \setminus \mathcal{C}_{i-1} = \{c_i\}$ for $i \geq 1$ and a constant $c_i \in \mathcal{C}$. Furthermore, given a set R of (non-ground) rules, each $\mathcal{C}_i$ yields a finite ground instantiation $grd(R, \mathcal{C}_i)$. While ground rules can simply be accumulated when the set of constants grows, completion (and loop) formulas cannot.

Example 2. Reconsider the rules R and facts F from Example 1. Relative to the constant stream $(c_1, s_1, s_2, c_2, \dots)$, the ground instantiations of R for $\mathcal{C}_1 = \{c_1\}$ and $\mathcal{C}_2 = \{c_1, s_1\}$, $R_1 = grd(R, \mathcal{C}_1)$ and $R_2 = grd(R, \mathcal{C}_2)$, are:

$$R_1 = \left\{ ok(c_1) \leftarrow cs(c_1), st(c_1), in(c_1, c_1) \qquad ko(c_1) \leftarrow cs(c_1), {\sim}ok(c_1) \right\}$$

$$R_2 = \left\{ \begin{array}{ll} ok(c_1) \leftarrow cs(c_1), st(c_1), in(c_1, c_1) & ko(c_1) \leftarrow cs(c_1), {\sim}ok(c_1) \\ ok(c_1) \leftarrow cs(c_1), st(s_1), in(s_1, c_1) & ko(s_1) \leftarrow cs(s_1), {\sim}ok(s_1) \\ ok(s_1) \leftarrow cs(s_1), st(c_1), in(c_1, s_1) & \\ ok(s_1) \leftarrow cs(s_1), st(s_1), in(s_1, s_1) & \end{array} \right\}$$

Relative to R_1, we obtain

$$cf(R_1, ok(c_1)) = ok(c_1) \rightarrow (cs(c_1) \wedge st(c_1) \wedge in(c_1, c_1)).$$

Along with

$$E_1 = E(\{cs(c_1)\}, \mathcal{P}_E, \mathcal{C}_1) = \{cs(c_1), \neg st(c_1), \neg in(c_1, c_1)\}$$

and in view of $((cs(c_1) \wedge \neg ok(c_1)) \rightarrow ko(c_1)) \in RF(R_1)$, $RF(R_1) \cup CF(R_1, \{ok(c_1), ko(c_1)\}) \cup E_1$ entails $\neg ok(c_1)$ and $ko(c_1)$. Turning to $R_2 \supseteq R_1$, we have that $RF(R_1) \subseteq RF(R_2)$. However, the rules defining $ok(c_1)$ yield

$$\begin{aligned} cf(R_2, ok(c_1)) = ok(c_1) \rightarrow ((cs(c_1) \wedge st(c_1) \wedge in(c_1, c_1)) \\ \vee (cs(c_1) \wedge st(s_1) \wedge in(s_1, c_1))), \end{aligned}$$

so that $cf(R_2, ok(c_1)) \neq cf(R_1, ok(c_1))$. Moreover,

$$\begin{aligned} E_2 &= E(\{cs(c_1), st(s_1), in(s_1, c_1)\}, \mathcal{P}_E, \mathcal{C}_2) \\ &= E_1 \cup \{st(s_1), in(s_1, c_1), \neg cs(s_1), \neg in(c_1, s_1), \neg in(s_1, s_1)\} \end{aligned}$$

and $((cs(c_1) \wedge st(s_1) \wedge in(s_1, c_1)) \rightarrow ok(c_1)) \in RF(R_2)$ entail $ok(c_1)$. Thus, $RF(R_2) \cup CF(R_1, \{ok(c_1), ko(c_1)\}) \cup E_2$ is unsatisfiable, and $cf(R_1, ok(c_1))$ must be replaced by $cf(R_2, ok(c_1))$ to obtain a (unique) model of $RF(R_2) \cup CF(R_2, \{ok(c_1), ko(c_1), ok(s_1), ko(s_1)\}) \cup E_2$, providing $ok(c_1)$, $\neg ko(c_1)$, $\neg ok(s_1)$, and $\neg ko(s_1)$ as conclusions. □

In incremental CDCL-based Boolean constraint solvers (cf. [14]), a replacement as above amounts to the withdrawal of all conflict information relying on invalidated completion (and loop) formulas, essentially restricting the "memory" of an incremental solver to direct consequences of rules or rule formulas, respectively. In order to resolve this problem, we in the following provide a translation approach on the first-order level, leading to ground instantiations such that corresponding completion and loop formulas can be accumulated, even when expanding the underlying Herbrand universe.

Our translation approach successively extends the signature of (non-ground) rules. To this end, given a set $\mathcal{P}_I$ of intensional predicates, we let $\mathcal{P}_I^k = \{p^k/n \mid p/n \in \mathcal{P}_I\}$ be a corresponding set of new predicates labeled with k. For an atom $p(X_1, \dots, X_n)$, we denote its labeled counterpart by $p(X_1, \dots, X_n)^k = p^k(X_1, \dots, X_n)$. Modifying the head of a rule r in this way yields $r^k = head(r)^k \leftarrow body(r)$.

The label k (or $k{+}1$) of a predicate serves as place holder for integers. Given $i \geq 0$, let $p^k[i] = p^i$ (or $p^{k+1}[i] = p^{i+1}$) if $p^k \in \mathcal{P}_I^k$ (or $p^{k+1} \in \mathcal{P}_I^{k+1}$) is labeled, while $p[i] = p$ for unlabeled predicates $p \in \mathcal{P}_E \cup \mathcal{P}_I$. We extend this notation to sets $\mathcal{P}$ of predicates and to atoms $p(X_1, \ldots, X_n)$ by letting $\mathcal{P}[i] = \{p[i] \mid p \in \mathcal{P}\}$ and $p(X_1, \ldots, X_n)[i] = p[i](X_1, \ldots, X_n)$. For a set R of rules, $R[i] = \{r[i] \mid r \in R\}$, where $r[i] = \mathit{head}(r)[i] \leftarrow \{a[i] \mid a \in \mathit{body}(r)^+\} \cup \{{\sim}a \mid a \in \mathit{body}(r)^-\}$.

Definition 2. *For a set R of rules over $(\mathcal{P}_E \cup \mathcal{P}_I, \emptyset, \mathcal{V})$, we define the sets $\Phi(R)$, $\Pi(\mathcal{P}_I)$, and $\Delta(\mathcal{P}_I)$ of rules as follows:*

$$\begin{aligned}
\Phi(R) &= \{r^k \mid r \in R\},\\
\Pi(\mathcal{P}_I) &= \{p(X_1, \ldots, X_n) \leftarrow p^k(X_1, \ldots, X_n) \mid p/n \in \mathcal{P}_I\},\\
\Delta(\mathcal{P}_I) &= \{p^k(X_1, \ldots, X_n) \leftarrow p^{k+1}(X_1, \ldots, X_n) \mid p/n \in \mathcal{P}_I\}.
\end{aligned}$$

Example 3. Labeling the heads of the rules in (1) and (2) leads to the following rules (without negative literals over labeled predicates) in $\Phi(R)$ for R from Example 1:

$$\begin{aligned}
\mathit{ok}^k(C) &\leftarrow \mathit{cs}(C), \mathit{st}(S), \mathit{in}(S, C)\\
\mathit{ko}^k(C) &\leftarrow \mathit{cs}(C), {\sim}\mathit{ok}(C)
\end{aligned}$$

In view of $\mathcal{P}_I = \{\mathit{ok}/1, \mathit{ko}/1\}$, $\Pi(\mathcal{P}_I)$ consists of the rules:

$$\mathit{ok}(C) \leftarrow \mathit{ok}^k(C) \qquad\qquad \mathit{ko}(C) \leftarrow \mathit{ko}^k(C)$$

Moreover, the rules in $\Delta(\mathcal{P}_I)$ prepare definition expansions:

$$\mathit{ok}^k(C) \leftarrow \mathit{ok}^{k+1}(C) \qquad\qquad \mathit{ko}^k(C) \leftarrow \mathit{ko}^{k+1}(C) \qquad\qquad \square$$

Given a constant stream $(c_i)_{i \geq 1}$, we aim at successive ground instantiations of $\Phi(R)$, $\Pi(\mathcal{P}_I)$, and $\Delta(\mathcal{P}_I)$ capturing the supported as well as the stable models of $F \cup R$ relative to each universe $\mathcal{C}_i$ and arbitrary facts F over $(\mathcal{P}_E, \mathcal{C}_i, \emptyset)$. To this end, we denote the ground substitutions and atoms that are *particular* to some $i \geq 0$ by $\Sigma(V, \mathcal{C}_i, i) = \{\sigma \in \Sigma(V, \mathcal{C}_i) \mid \max\{j \mid (X \mapsto c_j) \in \sigma\} = i\}$ and $\mathit{atom}(\mathcal{P}, \mathcal{C}_i, i) = \{p(X_1, \ldots, X_n)\sigma \mid p/n \in \mathcal{P}, \sigma \in \Sigma(\{X_1, \ldots, X_n\}, \mathcal{C}_i, i)\}$.[1] The resulting partition of substitutions (and atoms) forms the base for a selective instantiation of $\Phi(R)$, $\Pi(\mathcal{P}_I)$, and $\Delta(\mathcal{P}_I)$ relative to $(c_i)_{i \geq 1}$.

Definition 3. *For a set R of rules over $(\mathcal{P}_E \cup \mathcal{P}_I, \emptyset, \mathcal{V})$ and a constant stream $(c_i)_{i \geq 1}$ over $\mathcal{C}$, we define the expansible instantiation of R for $j \geq 0$ as $\mathit{exp}(R, j) = \bigcup_{i=0}^{j} R^i$, where*

$$\begin{aligned}
R^i = &\{(r[i])\sigma \mid r \in \Phi(R) \cup \Pi(\mathcal{P}_I), \sigma \in \Sigma(\mathit{var}(r), \mathcal{C}_i, i)\}\\
&\cup \{(r[i])\sigma \mid r \in \Delta(\mathcal{P}_I), \sigma \in \Sigma(\mathit{var}(r), \mathcal{C}_i)\}.
\end{aligned}$$

[1] Letting $\max \emptyset = 0$, since $\mathcal{C}_0 = \emptyset$, we get $\Sigma(\emptyset, \mathcal{C}_0, 0) = \Sigma(\emptyset, \emptyset) = \{\emptyset\}$, and $\mathit{atom}(\mathcal{P}, \emptyset, 0) = \{p \mid p/0 \in \mathcal{P}\}$ consists of atomic propositions.

Example 4. Starting with $\mathcal{C}_0 = \emptyset$, the rules $\Phi(R)$, $\Pi(\mathcal{P}_I)$, and $\Delta(\mathcal{P}_I)$ from Example 3 yield $exp(R, 0) = R^0 = \emptyset$ because each of them contains some variable. For $\mathcal{C}_1 = \{c_1\}$, however, we obtain the following set R^1 of ground rules:

$$R^1 = \left\{ \begin{array}{ll} ok^1(c_1) \leftarrow cs(c_1), st(c_1), in(c_1, c_1) & \\ ko^1(c_1) \leftarrow cs(c_1), {\sim}ok(c_1) & \\ ok(c_1) \leftarrow ok^1(c_1) & ko(c_1) \leftarrow ko^1(c_1) \\ ok^1(c_1) \leftarrow ok^2(c_1) & ko^1(c_1) \leftarrow ko^2(c_1) \end{array} \right\}$$

Observe that, beyond substituting variables with c_1, the label k (or $k{+}1$) is replaced by 1 (or 2). Also note that the atom $ok(c_1)$ over the original predicate $ok/1$ is derivable from $ok^1(c_1)$, an atom over the new predicate $ok^1/1$. Unlike the completion formula $cf(R_1, ok(c_1))$ from Example 2,

$$cf(R^1, ok^1(c_1)) = ok^1(c_1) \rightarrow ((cs(c_1) \wedge st(c_1) \wedge in(c_1, c_1)) \vee ok^2(c_1))$$

includes the yet undefined atom $ok^2(c_1)$ to represent derivations becoming available when another constant is added. In fact, such an additional derivation is contained in R^2, consisting of new ground rules relative to $\mathcal{C}_2 = \{c_1, s_1\}$:

$$R^2 = \left\{ \begin{array}{ll} ok^2(c_1) \leftarrow cs(c_1), st(s_1), in(s_1, c_1) & \\ ok^2(s_1) \leftarrow cs(s_1), st(c_1), in(c_1, s_1) & \\ ok^2(s_1) \leftarrow cs(s_1), st(s_1), in(s_1, s_1) & \\ ko^2(s_1) \leftarrow cs(s_1), {\sim}ok(s_1) & \\ ok(s_1) \leftarrow ok^2(s_1) & ko(s_1) \leftarrow ko^2(s_1) \\ ok^2(c_1) \leftarrow ok^3(c_1) & ko^2(c_1) \leftarrow ko^3(c_1) \\ ok^2(s_1) \leftarrow ok^3(s_1) & ko^2(s_1) \leftarrow ko^3(s_1) \end{array} \right\}$$

While the first six ground rules in R^2, stemming from $\Phi(R)$ and $\Pi(\mathcal{P}_I)$, include the second constant, viz. s_1, two of the four instances of rules in $\Delta(\mathcal{P}_I)$ mention c_1 only.□

Intuitively, the substitutions applied to $\Phi(R)$ (and $\Pi(\mathcal{P}_I)$) aim at new rule instances mentioning the constant c_i at stream position i, and the replacement of labels k by i makes sure that the defined predicates are new. Via instances of rules in $\Pi(\mathcal{P}_I)$, the new predicates are mapped back to the original ones in $\mathcal{P}_I$, at position i concentrating on ground atoms including c_i. The purpose of $\Delta(\mathcal{P}_I)$, on the other hand, is to keep the definitions of atoms in $atom(\mathcal{P}_I, \mathcal{C}_i)$ expansible, and the rules with yet undefined body atoms provide an interface for connecting additional derivations.

For each $i \geq 0$ and R^i as in Definition 3, we have that $head(R^i) = atom(\mathcal{P}_I, \mathcal{C}_i, i) \cup atom(\mathcal{P}_I^k[i], \mathcal{C}_i)$. In view of distinct constants in arguments or different predicate names, respectively, $head(R^i) \cap head(R^j) = \emptyset$ for $i > j \geq 0$. Hence, letting $head(exp(R, -1)) = \emptyset$, it also holds that

$$\begin{aligned} RF(R^i) \cap RF(R^j) &= \emptyset, \\ CF(R^i, head(R^i)) \cap CF(R^j, head(R^j)) &= \emptyset, \\ LF(exp(R, i), head(R^i), head(exp(R, i{-}1))) & \\ \cap\, LF(exp(R, j), head(R^j), head(exp(R, j{-}1))) &= \emptyset. \end{aligned}$$

As a consequence, the theories

$$RF^i(R) = \bigcup_{j=0}^{i} RF(R^j),$$
$$CF^i(R) = \bigcup_{j=0}^{i} CF(R^j, head(R^j)),$$
$$LF^i(R) = \bigcup_{j=0}^{i} LF(exp(R,j), head(R^j), head(exp(R,j-1)))$$

constitute disjoint unions. As shown next, they reproduce corresponding concepts for $exp(R,i)$ in a modular fashion.

Proposition 1. *Given a set R of rules over $(\mathcal{P}_E \cup \mathcal{P}_I, \emptyset, \mathcal{V})$ and a constant stream $(c_i)_{i\geq 1}$ over $\mathcal{C}$, we have for $j \geq 0$:*

$$RF^j(R) = RF(exp(R,j)),$$
$$CF^j(R) = CF(exp(R,j), head(exp(R,j))),$$
$$LF^j(R) \equiv LF(exp(R,j), head(exp(R,j)), \emptyset).$$

We now turn to the correspondence between supported as well as stable models of $F \cup R$ and $F \cup exp(R,i)$ for $i \geq 0$ and arbitrary facts F over $(\mathcal{P}_E, \mathcal{C}_i, \emptyset)$. To this end, we denote the *expansion atoms* over new predicates in $exp(R,i)$ by $expatom(\mathcal{P}_I, i) = \bigcup_{j=0}^{i} atom(\mathcal{P}_I^k[j], \mathcal{C}_j)$. For some $a \in atom(\mathcal{P}, \mathcal{C})$, by $\|a\| = \min\{j \geq 0 \mid a \in atom(\mathcal{P}, \mathcal{C}_j)\}$, we refer to the (unique) least j such that $a \in atom(\mathcal{P}, \mathcal{C}_j, j)$. Similarly, $\|r\| = \max\{\|a\| \mid a \in \{head(r)\} \cup body(r)^+ \cup body(r)^-\}$ denotes the smallest j such that all atoms in a ground rule r are contained in $atom(\mathcal{P}, \mathcal{C}_j)$. Moreover, we map any interpretation $I \subseteq atom(\mathcal{P}, \mathcal{C}_i)$ to an extended interpretation $I^* \subseteq atom(\mathcal{P}, \mathcal{C}_i) \cup expatom(\mathcal{P}_I, i)$ as follows:

$$I^* = I \cup \{head(r)^k[j] \mid r \in grd(R, \mathcal{C}_i), I \models bf(body(r)), \|head(r)\| \leq j \leq \|r\|\}.$$

That is, I^* augments a given I with expansion atoms for the heads of rules r whose bodies hold wrt I, where the label k is replaced by the integers from $\|head(r)\|$ to $\|r\|$. Note that the expansion atoms in $atom(\mathcal{P}_I^{k+1}[i], \mathcal{C}_i)$, which have no derivations in $exp(R,i)$, are fixed to false in I^* and any other interpretation $I' \subseteq atom(\mathcal{P}, \mathcal{C}_i) \cup expatom(\mathcal{P}_I, i)$.

The following result shows that our translation approach yields the intended semantics, viz. supported or stable models of a program $F \cup R$, relative to each universe $\mathcal{C}_i$ for $i \geq 0$.

Theorem 1. *Given a set R of rules over $(\mathcal{P}_E \cup \mathcal{P}_I, \emptyset, \mathcal{V})$ and a constant stream $(c_i)_{i\geq 1}$ over $\mathcal{C}$, we have for $j \geq 0$:*

1. *If $I \subseteq atom(\mathcal{P}, \mathcal{C}_j)$ is a supported (or stable) model of $(I \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup R$, then I^* is a supported (or stable) model of $(I \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup exp(R,j)$.*
2. *If $I' \subseteq atom(\mathcal{P}, \mathcal{C}_j) \cup expatom(\mathcal{P}_I, j)$ is a supported (or stable) model of $(I' \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup exp(R,j)$, then $I' = I^*$ for the supported (or stable) model $I = I' \cap atom(\mathcal{P}, \mathcal{C}_j)$ of $(I' \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup R$.*

Example 5. The ground rules R^1 and R^2 from Example 4 yield completion formulas $C_1 = CF(R^1, head(R^1))$ and $C_2 = CF(R^2, head(R^2))$ as follows:

$$C_1 = \left\{ \begin{array}{ll} ok^1(c_1) \rightarrow ((cs(c_1) \wedge st(c_1) \wedge in(c_1, c_1)) \vee ok^2(c_1)) & \\ ko^1(c_1) \rightarrow ((cs(c_1) \wedge \neg ok(c_1)) \vee ko^2(c_1)) & \\ ok(c_1) \rightarrow ok^1(c_1) & ko(c_1) \rightarrow ko^1(c_1) \end{array} \right\}$$

$$C_2 = \left\{ \begin{array}{ll} ok^2(c_1) \rightarrow ((cs(c_1) \wedge st(s_1) \wedge in(s_1, c_1)) \vee ok^3(c_1)) & \\ ok^2(s_1) \rightarrow ((cs(s_1) \wedge st(c_1) \wedge in(c_1, s_1)) & \\ \qquad \vee\ (cs(s_1) \wedge st(s_1) \wedge in(s_1, s_1)) \vee ok^3(s_1)) & \\ ko^2(c_1) \rightarrow ko^3(c_1) & \\ ko^2(s_1) \rightarrow ((cs(s_1) \wedge \neg ok(s_1)) \vee ko^3(s_1)) & \\ ok(s_1) \rightarrow ok^2(s_1) & ko(s_1) \rightarrow ko^2(s_1) \end{array} \right\}$$

Along with literals E_1 and E_2 as in Example 2, fixing atoms over extensional predicates, we obtain (supported) models $I_1^* = \{cs(c_1), ko(c_1), ko^1(c_1)\}$ and $I_2^* = \{cs(c_1), st(s_1), in(s_1, c_1), ok(c_1), ok^1(c_1), ok^2(c_1)\}$ of $RF(R^1) \cup C_1 \cup E_1$ or $RF(R^1) \cup RF(R^2) \cup C_1 \cup C_2 \cup E_2$, respectively. In the transition from I_1^* to I_2^*, $ko(c_1)$ is withdrawn and exchanged with $ok(c_1)$, as with R_1 and R_2 from Example 2. In contrast to the latter, however, the completion formulas C_2 do not invalidate C_1, but rather their (disjoint) union can be used. □

The benefit of expansible instantiation, $exp(R, i)$, in comparison to plain instantiation, $grd(R, \mathcal{C}_i)$, is that completion (and loop) formulas remain intact and can, like ground rules, be accumulated during the successive evolvement of a Herbrand universe. On the other hand, the downside is that, beyond the $\mathcal{O}(|grd(R, \mathcal{C}_i)|)$ ground rules stemming from $\Phi(R)$ and $\Pi(\mathcal{P}_I)$, additional $\mathcal{O}(i \times |atom(\mathcal{P}_I, \mathcal{C}_i)|)$ instances of rules in $\Delta(\mathcal{P}_I)$ are introduced for propagating derivations via expansion atoms. However, for an intensional predicate $p/n \in \mathcal{P}_I$ such that $var(r) = var(head(r))$ for all $r \in R$ with $head(r) = p(X_1, \ldots, X_n)$, definitions of atoms over p/n stay local because rule instances relying on a new constant c_i only provide derivations for atoms including c_i. In view of this, the introduction of a respective labeled predicate and corresponding rules in $\Phi(R)$, $\Pi(\mathcal{P}_I)$, and $\Delta(\mathcal{P}_I)$ is unnecessary, and the original rule(s), such as (2) for $ko/1$ in Example 1, can be instantiated (like members of $\Phi(R) \cup \Pi(\mathcal{P}_I)$) instead.

Example 6. Consider the following non-ground rules R over $\mathcal{P}_E = \{arc/2, vtx/1, init/1\}$ and $\mathcal{P}_I = \{cycle/2, other/2, reach/1\}$, aiming at directed Hamiltonian cycles:

$$R = \left\{ \begin{array}{l} cycle(X, Y) \leftarrow arc(X, Y), {\sim}other(X, Y) \\ other(X, Y) \leftarrow arc(X, Y), cycle(X, Z), Y \neq Z \\ \quad reach(Y) \leftarrow cycle(X, Y), init(X) \\ \quad reach(Y) \leftarrow cycle(X, Y), reach(X) \\ \quad reach(Y) \leftarrow vtx(Y), {\sim}reach(Y) \end{array} \right\}$$

Since the variables X and Y occur in the head $cycle(X, Y)$ of the first rule, expansion atoms for $cycle/2$ and respective rules can be omitted. Keeping the original rule, a simplified set R^1 of ground rules is obtained relative to $\mathcal{C}_1 = \{v_1\}$:[2]

[2] The condition $Y \neq Z$ filters admissible ground substitutions.

$$R^1 \cong \left\{ \begin{array}{c} cycle(v_1,v_1) \leftarrow arc(v_1,v_1), {\sim}other(v_1,v_1) \\ reach^1(v_1) \leftarrow cycle(v_1,v_1), init(v_1) \\ reach^1(v_1) \leftarrow cycle(v_1,v_1), reach(v_1) \\ reach^1(v_1) \leftarrow vtx(v_1), {\sim}reach(v_1) \\ other(v_1,v_1) \leftarrow other^1(v_1,v_1) \\ reach(v_1) \leftarrow reach^1(v_1) \\ other^1(v_1,v_1) \leftarrow other^2(v_1,v_1) \\ reach^1(v_1) \leftarrow reach^2(v_1) \end{array} \right\}$$

Given $F_1 = \{arc(v_1,v_1), vtx(v_1)\}$, $F_1 \cup R^1$ has $I_1^* = F_1 \cup \{cycle(v_1,v_1), reach(v_1), reach^1(v_1)\}$ as supported model that is not stable because, for $L_1 = \{reach(v_1), reach^1(v_1)\}$,

$$\begin{array}{c} lf(R^1, reach(v_1), L_1) = reach(v_1) \rightarrow (reach^2(v_1) \vee \\ (cycle(v_1,v_1) \wedge init(v_1)) \vee (vtx(v_1) \wedge \neg reach(v_1))) \end{array}$$

belongs to $LF(R^1, head(R^1), \emptyset)$. While $I_1^* \models RF(R^1) \cup CF(R^1, head(R^1))$, there is no model I' of $RF(R^1) \cup LF(R^1, head(R^1), \emptyset)$ such that $F_1 \subseteq I' \subseteq F_1 \cup head(R^1)$.

Letting $R_1^2 = R^1 \cup R^2$, where R^2 is the set of new ground rules for $\mathcal{C}_2 = \{v_1, v_2\}$, along with $F_2 = F_1 \cup \{arc(v_1,v_2), arc(v_2,v_1), vtx(v_2)\}$, $F_2 \cup R_1^2$ yields the supported model

$$\begin{array}{c} I_2^* = F_2 \cup \{cycle(v_1,v_2), cycle(v_2,v_1), other(v_1,v_1), other^1(v_1,v_1), other^2(v_1,v_1), \\ reach(v_1), reach(v_2), reach^1(v_1), reach^2(v_1), reach^2(v_2)\}, \end{array}$$

which is not stable either. For $L_2 = \{reach(v_1), reach(v_2), reach^1(v_1), reach^2(v_1), reach^2(v_2)\}$, since the loop formula

$$\begin{array}{c} lf(R_1^2, reach(v_2), L_2) = reach(v_2) \rightarrow ((\bigvee_{i=1}^2 reach^3(v_i)) \vee \\ (cycle(v_1,v_1) \wedge init(v_1)) \vee (cycle(v_2,v_1) \wedge init(v_2)) \vee \\ (cycle(v_1,v_2) \wedge init(v_1)) \vee (cycle(v_2,v_2) \wedge init(v_2)) \vee \\ (vtx(v_1) \wedge \neg reach(v_1)) \vee (vtx(v_2) \wedge \neg reach(v_2))) \end{array}$$

is contained in $LF(R_1^2, head(R^2), head(R^1))$, $I_2^* \not\models LF(R_1^2, head(R^2), head(R^1))$. However, when considering loop formulas for atoms defined by R^1 and R^2 in isolation, one can check that $I_2^* \models LF(R^1, head(R^1), \emptyset) \cup LF(R^2, head(R^2), \emptyset)$. In fact, positive dependencies between the atoms in L_2 involve rules from both R^1 and R^2. That is, R^1 and R^2 are not mutually independent in the sense of [15, 16]. □

4 Solving Expansible Programs

For an empirical evaluation of our translation approach, we modeled two benchmark domains, *Graph Coloring* and *Partner Units*, of the Fifth ASP Competition [17] by expansible programs in the language of *clingo* 4 [6]. Starting from an empty graph and no colors, the expansible program for *Graph Coloring* allows for a successive incorporation of vertices, arcs, and colors. While the addition of vertices and arcs constrains

admissible colorings, colors serve as resources that must be increased whenever a coloring task turns out to be unsatisfiable. In *Partner Units*, pairwisely connected zones and sensors need to be mapped to units, where two units are partners if the zone of a connected pair is mapped to one of them and the corresponding sensor to the other. Moreover, at most two zones and two sensors may share a unit, and the number of partners per unit must not exceed a given threshold, varying between two and four in problem instances of the Fifth ASP Competition. That is, the demand for units increases whenever the capacities are exceeded upon the successive addition of zones and sensors.

For both benchmark domains, the idea is to gradually expand and solve instances over arbitrarily many objects by introducing the objects, along with respective data, one after the other. Similarly, resources such as colors or units are increased on demand, rather than fixing and thus limiting them a priori. For instance, the following sequence of facts induces four successive *Graph Coloring* tasks: $F_1 = \{vtx(v_1, 1)\}$, $F_2 = \{col(n_1, 2)\}$, $F_3 = \{vtx(v_2, 3), arc(v_1, v_2, 3)\}$, and $F_4 = \{col(n_2, 4)\}$. While introducing the first vertex v_1 in F_1 yields an unsatisfiable task, a coloring is obtained after supplying color n_1 in F_2. However, one color is no longer sufficient when adding the second vertex v_2 and an arc to v_1 in F_3. Thus, F_4 provides another color n_2, leading to colorings mapping v_1 to n_1 (or n_2) and v_2 to n_2 (or n_1). Note that each of the above facts includes as last argument the maximum position of mentioned vertices or colors in the constant stream $(v_1, n_1, v_2, n_2, \dots)$. For one, this enables a reuse of constants for referring to vertices and colors, and w.l.o.g. we may assume that colors are denoted by consecutive integers starting with 1, viz. $n_1 = 1$, $n_2 = 2$, and so on. For another, the arguments indicating stream positions can be explored to distinguish corresponding rules in a parametrized *clingo* 4 program as follows:

$$col(C) \leftarrow col(C, k) \tag{3}$$
$$vtx(X) \leftarrow vtx(X, k) \tag{4}$$
$$arc(X, Y) \leftarrow arc(X, Y, k) \tag{5}$$
$$new(X, C, k) \leftarrow vtx(X, k), col(C) \tag{6}$$
$$new(X, C, k) \leftarrow vtx(X), col(C, k) \tag{7}$$
$$\{map(X, C)\} \leftarrow new(X, C, k) \tag{8}$$
$$\leftarrow map(X, C), map(Y, C), arc(X, Y, k), col(C) \tag{9}$$
$$\leftarrow map(X, C), map(Y, C), arc(X, Y), col(C, k) \tag{10}$$
$$has(X, C) \leftarrow new(X, C, k), map(X, C) \tag{11}$$
$$has(X, C) \leftarrow new(X, C, k), has(X, C{+}1) \tag{12}$$
$$\leftarrow new(X, C, k), map(X, C{-}1), has(X, C) \tag{13}$$
$$\leftarrow vtx(X, k), {\sim}has(X, 1) \tag{14}$$

Assuming that the program parameter k is successively replaced by the stream positions of objects in supplied facts, the rules in (3)–(5) provide projections dropping the positions from respective colors, vertices, or arcs. The auxiliary predicate $new/3$, defined in (6) and (7), indicates pairs of vertices and colors such that either of them is introduced at a stream position substituted for parameter k. Given this, applicable instances of the choice rule (cf. [18]) in (8) have distinct heads at different positions, so that expansion

atoms can be omitted for $map/2$. The integrity constraints (i.e., rules with false heads) in (9) and (10) deny choices of the same color for adjacent vertices, where the body atoms $arc(X, Y, k)$ or $col(C, k)$ confine applicable instances to new arcs or colors, respectively. The purpose of atoms of the form $has(v, n)$ is to indicate that a vertex v is mapped to some color $m \geq n$. When either v or n is introduced at a stream position, the rule in (11) captures the case that v is mapped to n, while colors added later on are addressed by the rule in (12). Making use of the convention that colors are denoted by consecutive integers, the latter includes expansion atoms of the form $has(v, n{+}1)$, rather than $has^{k+1}(v, n)$ or a corresponding *clingo* 4 representation $has(v, n, k{+}1)$, respectively. Note that, by saving an argument for the predicate label, the number of introduced expansion atoms remains linear, thus avoiding a quadratic blow-up as discussed below Example 5. The integrity constraints in (13) and (14) further investigate atoms over $has/2$ to make sure that each vertex is mapped to exactly one color. Finally, to keep *clingo* 4 off discarding body atoms that are not necessarily defined when instances of (12) and (14) are introduced, the program in (3)–(14) has to be accompanied by *#external* $has(X, C{+}1) : new(X, C, k)$ and *#external* $has(X, 1) : vtx(X, k)$. Without going into details, we note that *Partner Units* can be modeled in a similar way, where zones and sensors amount to vertices and units to colors.

Table (a) and (b) provide experimental results of running *clingo* 4 (version 4.4.0) on the *Graph Coloring* and *Partner Units* instances of the Fifth ASP Competition. In particular, we used the Python interface of *clingo* 4 to successively introduce objects, viz. vertices or zones and sensors, and instantiate respective rules in (parametrized) expansible programs.[3] Whenever this leads to an unsatisfiable task, another color or unit is added in turn, thus obtaining sequences of satisfiable as well as unsatisfiable tasks of gradually increasing size. With sequential runtimes restricted to 10 minutes per instance on a Linux PC equipped with 2.4GHz processors, columns headed by #S and ∅S report the number of solved satisfiable tasks along with the corresponding average runtime in seconds, and columns headed by #U and ∅U provide the same information for unsatisfiable tasks. We compare the performance of *clingo* 4 in two operation modes: one-shot solving, in which each task is processed independently from scratch, and multi-shot solving, where rule instances are successively accumulated and conflict information can be carried over between tasks. Experimental results with one-shot solving are given in the middle parts of Table (a) and (b), followed by multi-shot solving on the right. Notably, as instantiation times are negligible, the comparison primarily contrasts the search performance in solving successive tasks either independently or incrementally.

Given the combinatorial nature of the benchmark domains, one- and multi-shot solving scale up to tasks of similar size, leading to the same number of solved tasks. The gap between both solving approaches turns out to be small on the instances of *Graph Coloring*, where instance 32 yields an exceptionally hard unsatisfiable task that can still be solved in time. On the *Partner Units* instances in Table (b), however, we observe that multi-shot solving consistently reduces the runtime for rather easy satisfiable tasks. Except for two instances (091 and 127), it also yields shorter runtimes for unsatisfiable tasks, even by an order of magnitude in several cases (026, 100, 175, and 188). Comparing the numbers of conflicts revealed that, in these cases, multi-shot solving indeed

[3] `http://svn.code.sf.net/p/potassco/code/trunk/gringo/examples/`

Instance	#S	∅S	#U	∅U	#S	∅S	#U	∅U
04	125	0.28	5	0.10	125	0.15	5	0.04
05	125	0.13	5	0.12	125	0.03	5	0.08
07	125	0.13	5	0.16	125	0.02	5	0.13
08	125	0.14	5	0.20	125	0.04	5	0.16
13	130	0.13	5	0.11	130	0.02	5	0.04
21	121	0.26	5	0.10	121	0.27	5	0.03
22	121	0.29	5	0.09	121	0.80	5	0.02
23	135	1.17	5	0.11	135	2.07	5	0.05
25	125	0.21	5	0.11	125	0.03	5	0.04
32	140	0.22	6	68.59	140	0.07	6	95.33
36	128	0.52	5	0.18	128	0.55	5	0.14
39	124	0.15	5	0.15	124	0.07	5	0.11
40	121	0.59	5	0.13	121	2.14	5	0.07
46	124	0.69	5	0.15	124	0.50	5	0.10
47	132	0.68	5	0.12	132	0.63	5	0.04
48	128	1.81	5	0.10	128	1.76	5	0.02
50	130	0.88	5	0.12	130	0.49	5	0.04
56	139	2.17	5	0.17	139	2.43	5	0.11
59	128	0.35	5	0.21	128	0.09	5	0.15
60	118	0.16	5	0.11	118	0.04	5	0.02

(a) One- vs. multi-shot: *Graph Coloring*

Instance	#S	∅S	#U	∅U	#S	∅S	#U	∅U
026	40	0.10	10	34.69	40	0.04	10	3.00
052	40	0.10	10	15.29	40	0.03	10	5.58
058	40	0.10	10	22.97	40	0.04	10	5.67
069	40	0.11	10	12.87	40	0.04	10	5.98
091	40	0.10	10	3.71	40	0.04	10	8.42
099	40	0.10	10	10.49	40	0.04	10	5.41
100	40	0.09	10	57.05	40	0.04	10	2.13
102	40	0.10	10	13.29	40	0.04	10	8.45
114	40	0.10	10	16.95	40	0.04	10	7.13
115	40	0.10	10	19.02	40	0.04	10	3.98
119	40	0.10	10	26.01	40	0.04	10	5.65
127	40	0.10	10	4.99	40	0.04	10	9.38
153	40	0.11	10	32.08	40	0.04	10	6.04
154	40	0.10	10	22.75	40	0.04	10	10.23
156	40	0.10	10	11.63	40	0.04	10	9.41
161	40	0.11	10	24.56	40	0.04	10	5.10
175	40	0.12	10	48.44	40	0.04	10	4.86
180	36	0.08	9	2.02	36	0.03	9	1.84
188	40	0.11	10	54.67	40	0.03	10	2.69
196	40	0.07	10	26.80	40	0.03	10	13.97

(b) One- vs. multi-shot: *Partner Units*

encounters an order of magnitude fewer conflicts, viz. about 200,000 vs. 2,000,000 on average with one-shot solving. The other way round, the difference amounts to a maximum factor of 2 (roughly 250,000 vs. 500,000 conflicts) between single- and multi-shot solving on instance 091. This indicates an increased robustness due to the reuse of conflict information in multi-shot solving.

5 Discussion

We introduced a simple approach to successively incorporating new objects into (multi-shot) ASP solving. Our approach rests upon a translation and refrains from dedicated algorithms. Also, it is modular and thus allows for adding new information without altering the existing ground program or underlying constraints, respectively. Hence, our approach enables incremental finite model finding [19], even under nonmonotonicity faced with supported or stable instead of classical models. Technically, it employs a less restrictive notion of modularity than [15, 16]: Proposition 1 applies to successive ground programs of an expansible instantiation (according to Definition 3) for the rules in Example 6, although the ground programs are not mutually independent. In view of a lacking theoretical elaboration, incremental ASP systems like *clingo* 4 do not yet provide automatic support for expanding the definitions of atoms or handling mutual positive dependencies between successive ground programs. Our work thus outlines potential future system refinements in these regards.

A related approach is *dlvhex* [20] using external sources for value invention. This is accomplished by dedicated grounding algorithms incorporating external objects. Once grounding is completed, no further objects are taken into account. Unlike this, ASP systems relying on lazy grounding, like *asperix* [21], *gasp* [22], and *omiga* [23], aim at instantiating variables on demand. However, all of them rely on a fixed Herbrand universe and use dedicated grounding and solving algorithms, whose performance does not match that of modern ASP systems. Although the lazy grounding approach in [24] is tailored for query answering, interestingly, it introduces so-called Tseitin variables resembling ground expansion atoms in partial instantiations.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University (2003)
2. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7** (1960) 201–215
3. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In Proceedings of MODEL'03. (2003)
4. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electronic Notes in Theoretical Computer Science **89**(4) (2003)
5. Rintanen, J., Heljanko, K., Niemelä, I.: Planning as satisfiability: Parallel plans and algorithms for plan search. Artificial Intelligence **170**(12-13) (2006) 1031–1080
6. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: *Clingo* = ASP + control: Preliminary report. [25] Available at `http://arxiv.org/abs/1405.3694v1`
7. Apt, K., Blair, H., Walker, A.: Towards a theory of declarative knowledge. In Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann (1987) 89–148
8. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Proceedings of ICLP'88. MIT (1988) 1070–1080
9. Fages, F.: Consistency of Clark's completion and the existence of stable models. Journal of Methods of Logic in Computer Science **1** (1994) 51–60
10. Clark, K.: Negation as failure. In Logic and Data Bases. Plenum (1978) 293–322
11. Lee, J.: A model-theoretic counterpart of loop formulas. In Proceedings of IJCAI'05. Professional Book Center (2005) 503–508
12. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157**(1-2) (2004) 115–137
13. Ullman, J.: Principles of Database and Knowledge-Base Systems. Computer Science (1988)
14. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In Handbook of Satisfiability. IOS (2009) 131–153
15. Oikarinen, E., Janhunen, T.: Modular equivalence for normal logic programs. In Proceedings of ECAI'06. IOS (2006) 412–416
16. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity aspects of disjunctive stable models. Journal of Artificial Intelligence Research **35** (2009) 813–857
17. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: The design of the fifth answer set programming competition. [25] Available at `http://arxiv.org/abs/1405.3710v4`
18. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
19. Gebser, M., Sabuncu, O., Schaub, T.: An incremental answer set programming based system for finite model computation. AI Communications **24**(2) (2011) 195–212
20. Eiter, T., Fink, M., Krennwallner, T., Redl, C.: Grounding HEX-programs with expanding domains. In Proceedings of GTTV'13. (2013) 3–15
21. Lefèvre, C., Nicolas, P.: The first version of a new ASP solver: ASPeRiX. In Proceedings of LPNMR'09. Springer (2009) 522–527
22. Dal Palù, A., Dovier, A., Pontelli, E., Rossi, G.: Answer set programming with constraints using lazy grounding. In Proceedings of ICLP'09. Springer (2009) 115–129
23. Dao-Tran, M., Eiter, T., Fink, M., Weidinger, G., Weinzierl, A.: OMiGA: An open minded grounding on-the-fly answer set solver. In Proceedings of JELIA'12. Springer (2012) 480–483
24. De Cat, B., Denecker, M., Stuckey, P.: Lazy model expansion by incremental grounding. In Technical Communications of ICLP'12. LIPIcs (2012) 201–211
25. Technical Communications of ICLP'14. Theory and Practice of Logic Programming **14**(4-5) (2014) Online supplement

A Proofs

Proposition 1. *Given a set R of rules over $(\mathcal{P}_E \cup \mathcal{P}_I, \emptyset, \mathcal{V})$ and a constant stream $(c_i)_{i\geq 1}$ over $\mathcal{C}$, we have for $j \geq 0$:*

$$RF^j(R) = RF(exp(R,j)),$$
$$CF^j(R) = CF(exp(R,j), head(exp(R,j))),$$
$$LF^j(R) \equiv LF(exp(R,j), head(exp(R,j)), \emptyset).$$

Proof (Sketch). $RF^j(R) = RF(exp(R,j))$ and $CF^j(R) = CF(exp(R,j), head(exp(R,j)))$ follow from $exp(R,j) = \bigcup_{i=0}^{j} R^i$ and $head(R^l) \cap head(R^i) = \emptyset$ for $j \geq l > i \geq 0$. For $J = LF(exp(R,j), head(exp(R,j)), \emptyset)$, $LF^j(R) \subseteq J$ yields $J \models LF^j(R)$. On the other hand, any $(a_L \rightarrow \bigvee_{B\in supp(exp(R,j),L)} bf(B)) \in J \setminus LF^j(R)$ is such that $L \subseteq head(exp(R,j))$, $a_L \in head(R^i)$ for $j > i \geq 0$, and $L \cap \bigcup_{l=i+1}^{j} head(R^l) \neq \emptyset$. Consider some interpretation $I \subseteq atom(\mathcal{P}, \mathcal{C}_j) \cup \bigcup_{i=0}^{j} atom(\mathcal{P}_I^k[i], \mathcal{C}_i)$ such that $I \not\models (a_L \rightarrow \bigvee_{B\in supp(exp(R,j),L)} bf(B))$. Then, since $B^+ \cap (L \setminus I) \neq \emptyset$ and $I \not\models bf(B)$ hold for every $B \in supp(exp(R,j), L \cap I) \setminus supp(exp(R,j), L)$, we have that $I \not\models \bigvee_{B\in supp(exp(R,j),L\cap I)} bf(B)$. Given that $a_L \in L \cap I$, $head(R^i) \cap (L \cap I) \neq \emptyset$ holds for $i = \max\{l \geq 0 \mid head(R^l) \cap (L \cap I) \neq \emptyset\}$. Hence, there is some $a \in head(R^i) \cap (L \cap I)$ such that $I \not\models (a \rightarrow \bigvee_{B\in supp(exp(R,i),L\cap I)} bf(B))$. As $LF(exp(R,i), head(R^i), head(exp(R,i-1)))$ includes $a \rightarrow \bigvee_{B\in supp(exp(R,i),L\cap I)} bf(B)$, we conclude that $I \not\models LF^j(R)$, which in turn yields $LF^j(R) \models J$. □

Theorem 1. *Given a set R of rules over $(\mathcal{P}_E \cup \mathcal{P}_I, \emptyset, \mathcal{V})$ and a constant stream $(c_i)_{i\geq 1}$ over $\mathcal{C}$, we have for $j \geq 0$:*

1. *If $I \subseteq atom(\mathcal{P}, \mathcal{C}_j)$ is a supported (or stable) model of $(I \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup R$, then I^* is a supported (or stable) model of $(I \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup exp(R,j)$.*
2. *If $I' \subseteq atom(\mathcal{P}, \mathcal{C}_j) \cup expatom(\mathcal{P}_I, j)$ is a supported (or stable) model of $(I' \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup exp(R,j)$, then $I' = I^*$ for the supported (or stable) model $I = I' \cap atom(\mathcal{P}, \mathcal{C}_j)$ of $(I' \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup R$.*

Proof (Sketch). Given some $I \subseteq atom(\mathcal{P}, \mathcal{C}_j)$, by construction, we have that $I^* \models RF(\{r \in exp(R,j) \mid head(r) \in expatom(\mathcal{P}_I, j)\}) \cup CF(exp(R,j), expatom(\mathcal{P}_I, j))$, while $I' \not\models RF(\{r \in exp(R,j) \mid head(r) \in expatom(\mathcal{P}_I, j)\}) \cup CF(exp(R,j), expatom(\mathcal{P}_I, j))$ for any $I' \neq I^*$ such that $I \subseteq I' \subseteq I \cup expatom(\mathcal{P}_I, j)$. In particular, for every $a \in atom(\mathcal{P}_I, \mathcal{C}_j)$, it holds that $a^k[\|a\|] \in I^*$ iff $a = head(r)$ for some $r \in grd(R, \mathcal{C}_j)$ such that $I \models bf(r)$. In view of $\{r \in exp(R,j) \mid head(r) \in atom(\mathcal{P}_I, \mathcal{C}_j)\} = \{a \leftarrow a^k[\|a\|] \mid a \in atom(\mathcal{P}_I, \mathcal{C}_j)\}$, we conclude that $I \models RF(grd(R, \mathcal{C}_j)) \cup CF(grd(R, \mathcal{C}_j), atom(\mathcal{P}_I, \mathcal{C}_j))$ iff $I^* \models RF(exp(R,j)) \cup CF(exp(R,j), atom(\mathcal{P}_I, \mathcal{C}_j) \cup expatom(\mathcal{P}_I, j))$. Hence, I is a supported model of $(I \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup R$ iff I^* is a supported model of $(I \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup exp(R,j)$, which does not admit supported models $I' \neq I^*$ such that $I \subseteq I' \subseteq I \cup expatom(\mathcal{P}_I, j)$.

For $L \subseteq atom(\mathcal{P}_I, \mathcal{C}_j)$ and any $a \in L$, it holds that $I \models lf(grd(R, \mathcal{C}_j), a, L)$ iff $I^* \models lf(exp(R, j), a, L \cup \{a_L^k[i] \mid a_L \in L, \|a_L\| \leq i \leq j\})$. On the other hand, for $L' \subseteq atom(\mathcal{P}_I, \mathcal{C}_j) \cup expatom(\mathcal{P}_I, j)$ and any $a \in L'$, if $I^* \not\models lf(exp(R, j), a, L')$, then we have that $L = L' \cap I \neq \emptyset$ and $I \not\models lf(grd(R, \mathcal{C}_j), a_L, L)$ for each $a_L \in L$. We conclude that $I \models LF(grd(R, \mathcal{C}_j), atom(\mathcal{P}_I, \mathcal{C}_j), \emptyset)$ iff $I^* \models LF(exp(R, j), atom(\mathcal{P}_I, \mathcal{C}_j) \cup expatom(\mathcal{P}_I, j), \emptyset)$, so that I is a stable model of $(I \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup R$ iff I^* is a stable model of $(I \cap atom(\mathcal{P}_E, \mathcal{C}_j)) \cup exp(R, j)$. $\square$