Online appendix for the paper

A model building framework for Answer Set Programming with external computations

published in Theory and Practice of Logic Programming

Thomas Eiter, Michael Fink

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
(*e-mail:* {eiter,fink}@kr.tuwien.ac.at)

Giovambattista Ianni

Dipartimento di Matematica, Cubo 30B, Università della Calabria
87036 Rende (CS), Italy
(*e-mail:* ianni@mat.unical.it)

Thomas Krennwallner, Christoph Redl

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
(*e-mail:* {tkren,redl}@kr.tuwien.ac.at)

Peter Schüller

Computer Engineering Department, Faculty of Engineering, Marmara University
Goztepe Kampusu, Kadikoy 34722, Istanbul, Turkey
(*e-mail:* peter.schuller@marmara.edu.tr)

submitted 27 January 2015; revised 24 June 2015; accepted 28 June 2015

Appendix A Proofs

Proof of Theorem 1 (Splitting Theorem)
Given a set of ground atoms M and a set of rules R, we denote by $M|_R = M \cap gh(R)$ the projection of M to ground heads of rules in R.

($\Rightarrow$) Let $M \in \mathcal{AS}(P)$. We show that (1) $M|_R \in \mathcal{AS}(R)$ and that (2) $M \in \mathcal{AS}(P \setminus R \cup facts(M|_R))$.

As for (1), we first show that $M|_R$ satisfies the reduct $fR^{M|_R}$, and then that $M|_R$ is indeed a minimal model of $fR^{M|_R}$. M satisfies fP^M and $R \subseteq P$. Observe that, by definition of FLP reduct, $fR^M \subseteq fP^M$. By definition of rule splitting set, satisfiability of rules in R does not depend on heads of rules in $P \setminus R$ (due to the restriction of external atoms to extensional semantics, this is in particular true for external atoms in R). Therefore $fR^{M|_R} = fR^M$, M satisfies $fR^{M|_R}$, and $M|_R$ satisfies $fR^{M|_R}$. For showing $M|_R \in \mathcal{AS}(R)$, it remains to show that $M|_R$ is a minimal model of $fR^{M|_R}$.

Assume towards a contradiction that some $S \subset M|_R$ is a model of $fR^{M|_R}$. Then there is a nonempty set $A = M|_R \setminus S$ of atoms with $A \subseteq gh(R)$. Let $M^\star = M \setminus A$. We next show that $M^\star$ is a model of fP^M, which implies that $M \notin \mathcal{AS}(P)$. Assume on the

contrary that $M^\star$ is not a model of fP^M. Hence there exists some rule $r \in fP^M$ such that $H(r) \cap M^\star = \emptyset$, $B^+(r) \subseteq M^\star$, $B^-(r) \cap M^\star = \emptyset$ and external atoms in $B^+(r)$ (resp., $B^-(r)$) evaluate to true (resp., false) wrt. $M^\star$. S agrees with $M^\star$ on atoms from $gh(R)$, and S satisfies $fR^{M|_R}$. The truth values of external atoms in bodies of rules in R depends only on atoms from $gh(R)$, therefore external atoms in R evaluate to the same truth value wrt. S and $M^\star$. Therefore $r \notin fR^{M|_R}$ and $r \in f(P \setminus R)^M$. Since $r \in P \setminus R$, $H(r) \subseteq gh(P \setminus R)$, and because M and $M^\star$ agree on atoms from $gh(P \setminus R)$, $H(r) \cap M^\star = \emptyset$ from above implies that $H(r) \cap M = \emptyset$. Because $r \in fP^M$, its body is satisfied in M, and since its head has no intersection with M, we get that fP^M is not satisfied by M, which is a contradiction. Therefore $M^\star$ is a model of fP^M. As $M^\star \subset M$, this contradicts our assumption that $M \in \mathcal{AS}(P)$. Therefore $S = M|_R = X$ is a minimal model of fR^M.

We next show that M satisfies the reduct $f(P \setminus R \cup facts(M|_R))^M$, and then that it is indeed a minimal model of the reduct. By the definition of reduct, $f(P \setminus R \cup facts(M|_R))^M = f(P \setminus R)^M \cup facts(M|_R)$. M satisfies $facts(M|_R)$ because $M|_R \subseteq M$. Furthermore $f(P \setminus R)^M \subseteq fP^M$, hence M satisfies $f(P \setminus R)^M$. Therefore M satisfies $f(P \setminus R \cup facts(M|_R))^M$.

To show that M is a minimal model of $f(P \setminus R \cup facts(M|_R))^M$, assume towards a contradiction that some $S \subset M$ is a model of $f(P \setminus R \cup facts(M|_R))^M$. Since $facts(M|_R)$ is part of the reduct, $M|_R \subseteq S$, therefore $S|_{gh(R)} = M|_R$. By definition of rule splitting set, satisfiability of rules in R does not depend on heads of rules in $P \setminus R$, hence S satisfies fR^M. Because S satisfies $f(P \setminus R \cup facts(M|_R))^M = f(P \setminus R)^M \cup facts(M|_R)$, it also satisfies $f(P \setminus R)^M$. Since S satisfies both fR^M, S satisfies $fP^M = f(P \setminus R)^M \cup fR^M$. This is a contradiction to $M \in \mathcal{AS}(P)$. Therefore $S = M$ is a minimal model of $f(P \setminus R \cup facts(M|_R))^M$.

($\Leftarrow$) Let $M \in \mathcal{AS}(P \setminus R \cup facts(X))$ and let $X \in \mathcal{AS}(R)$. We first show that M satisfies fP^M, and then that it is a minimal model of fP^M.

As facts X are part of the program $P \setminus R \cup facts(X)$, and by definition of rule splitting set, $P \setminus R$ contains no rule heads unifying with $gh(R)$, hence we have $X = M|_R$. Furthermore $f(P \setminus R \cup facts(X))^M \setminus facts(X) \cup fR^M = fP^M$, and as M satisfies the left side, it satisfies the right side. To show that M is a minimal model of fP^M, assume $S \subset M$ is a smaller model of fP^M. By definition of reduct, S also satisfies $f(P \setminus R)^M$ and fR^M. Since R is a splitting set, satisfiability of rules in R does not depend on heads of rules in $P \setminus R$, therefore $fR^M = fR^{M|_R} = fR^X$ and $S|_{gh(R)}$ satisfies fR^X. Since $S \subset M$, we have $S|_{gh(R)} \subseteq X$. Because X is a minimal model of fR^X, $S|_{gh(R)} \subset X$ is impossible and $S|_{gh(R)} = X$. Therefore $S|_{gh(P \setminus R)} \subset M|_{gh(P \setminus R)}$. Because S satisfies $f(P \setminus R)^M$ and $S|_{gh(R)} = X$, S also satisfies $f(P \setminus R \cup facts(X))^M$. Since $S \subset M$, this contradicts the fact that M is a minimal model of $P \setminus R \cup facts(X)$. Therefore $S = M$ is a minimal model of fP^M. $\square$

Proof of Theorem 2 (Generalized Splitting Theorem)
By definition of generalized bottom, the set $C = B \setminus R$ contains only constraints, therefore $gh(B) = gh(R)$ and $M|_{gh(B)} = M|_{gh(R)}$. As $R \subseteq B$ and $B \setminus R$ contains only constraints, $\mathcal{AS}(B) \subseteq \mathcal{AS}(R)$. The only difference between Theorem 1 and Theorem 2 is, that for obtaining X, the latter takes additional constraints into account.

($\Rightarrow$) It is sufficient to show that $M|_{gh(B)}$ does not satisfy the body of any constraint in

$C \subseteq P$ if M does not satisfy the body of any constraint in P. Since B is a generalized bottom, no negative dependencies of constraints C to rules in $P \setminus B$ exist; therefore if the body of a constraint $c \in C$ is not satisfied by M, the body of c is not satisfied by $M|_{gh(B)}$. As M satisfies P, it does not satisfy any constraint body in P, hence the projection $M|_{gh(B)}$ does not satisfy any constraint body in $B \setminus R$.

($\Leftarrow$) It is sufficient to show that an answer set of R that satisfies a constraint body in C also satisfies that constraint body in P, which raises a contradiction. As constraints in C have no negative dependencies to rules in $P \setminus B$, a constraint with a satisfied body in $M|_{gh(R)}$ also has a satisfied body in M, therefore the result follows. □

Proof of Proposition 1
Assume towards a contradiction that there exist a non-constraint $r \in P$, a rule $s \in P$ with $r \to_{m,n} s$, and $u' \in U|_r$, $v' \in U|_s$ such that $(u', v') \notin E$. Due to Definition 9, $r \to_{m,n} s$ implies that s has $H(s) \neq \emptyset$ and therefore that s is a non-constraint. Definition 14 (b) then implies that $U|_r = \{u'\}$ and $U|_s = \{v'\}$ (non-constraints are present in exactly one unit).

Case (i): for $r \to_n s$, Definition 14 (c) specifies that for all $u \in U|_r$ and $v \in U|_s$ there exists an edge $(u, v) \in E$, therefore also $(u', v') \in E$, which is a contradiction.

Case (ii): for $r \to_m s$, Definition 14 (d) specifies that some $u \in U|_r$ exists such that for every $v \in U|_s$ there exists an edge $(u, v) \in E$; since $U|_r = \{u'\}$ and $U|_s = \{v'\}$, it must hold that $(u', v') \in E$, which is a contradiction. □

Proof of Proposition 2
Given two distinct units $u_1, u_2 \in U$, assume towards a contradiction that some $\gamma \in gh(u_1) \cap gh(u_2)$ exists. Then there exists some $r \in u_1$ with $\alpha \in H(r)$ and $\alpha \sim \gamma$, and there exists some $s \in u_2$ with $\beta \in H(s)$ and $\beta \sim \gamma$. As $\alpha \sim \gamma$ and $\beta \sim \gamma$ and γ is ground, we obtain $\alpha \sim \beta$; hence, by Definition 9 (iii) we have $r \to_m s$ and $s \to_m r$. As r and s have nonempty heads, they are non-constraints. Thus by Proposition 1, there exist edges $(u_1, u_2), (u_2, u_1) \in E$. As an evaluation graph is acyclic, it follows $u_1 = u_2$; this is a contradiction. □

Proof of Proposition 3
For an lde-safe program P, the graph $\mathcal{E} = (\{P\}, \emptyset)$ is a valid evaluation graph. □

Proof of Theorem 3
For any set of rules, let $constr(S) = \{r \in S \mid H(r) = \emptyset\}$ denote the set of constraints in S. We say that the *dependencies of $r \in Q$ are covered at unit $u \in U$*, if for every rule $s \in Q$ such that $r \to_{m,n} s$ and $s \notin u$, it holds that $(u, u') \in E$ for all $u' \in U|_s$, i.e., u has an edge to all units containing s.

To prove that $B = u^<$ is a generalized bottom of $P = u^\leq$ wrt. the rule splitting set $R = u^< \setminus constr(u^<)$ as by Definition 12, we prove that (a) $R \subseteq B \subseteq P$, (b) $B \setminus R$ contains only constraints, (c) no constraint in $B \setminus R$ has nonmonotonic dependencies to rules in $P \setminus B$, and (d) R is a rule splitting set of P.

Statement (a) corresponds to $u^< \setminus constr(u^<) \subseteq u^< \subseteq u^\leq$ and $u^\leq$ is defined as $u^\leq = u^< \cup u$, therefore the relations all hold. For (b), $B \setminus R = u^< \setminus (u^< \setminus constr(u^<))$, and as $A \setminus (A \setminus B) = A \cap B$, it is easy to see that $B \setminus R = u^< \cap constr(u^<)$ and thus

$B \setminus R$ only contains constraints. For (c), we show a stronger property, namely that no rule (constraint or non-constraint) in B has nonmonotonic dependencies to rules in $P \setminus B$. $B = u^<$ is the union of evaluation units $V = \{v \in U \mid v < u\}$. By Definition 14 (c) all nonmonotonic dependencies $r \rightarrow_n s$ are covered at every unit w such that $w \in U_r$. Hence if $r \in w$ and $w \in V$, then either $s \in w$ or $s \in w^<$ holds, and hence $s \in w^\leq \subseteq u^<$. As $P \setminus B = u^\leq \setminus u^<$, no nonmonotonic dependencies from $B = u^<$ to $P \setminus B$ exist and (c) holds. For (d) we know that $R = u^< \setminus \mathit{constr}(u^<)$ contains no constraints, and by Proposition 1 all dependencies of non-constraints in R are covered by $\mathcal{E}$. Therefore $r \in R$, $r \rightarrow_{m,n} s$, and $s \in P$ implies that $s \in R$. Consequently, (d) holds which proves the theorem. □

Proof of Theorem 4
Similar to the proof of Theorem 3, we show this in four steps; given $P = u^<$, $R = u'^\leq \setminus \mathit{constr}(u'^\leq)$, and $B = u'^\leq = u' \cup u'^<$, we show that (a) $R \subseteq B \subseteq P$, (b) $B \setminus R$ contains only constraints, (c) no constraint in $B \setminus R$ has nonmonotonic dependencies to rules in $P \setminus B$, and (d) R is a rule splitting set of P. Let $\mathit{preds}_\mathcal{E}(u) = \{u_1, \ldots, u_k\}$ and Let $V = \{v \in U \mid v < u'\}$ be the set of units on which u' transitively depends. (Note that $V \subset \mathit{preds}_\mathcal{E}(u)$ and $u \notin V$.) As $u'^<$ contains all units u' transitively depends on, we have $B = u' \cup \bigcup_{w \in V} w$.

For (a), $R \subseteq B$ holds trivially, and $B \subseteq P$ holds by definition of $u^<$ and $u'^\leq$ and because $u' \in \mathit{preds}_\mathcal{E}(u)$. Statement (b) holds, because $B \setminus R$ removes R from B, i.e., it removes everything that is not a constraint in B from B, therefore only constraints remain. For (c) we show that no rule in B has a nonmonotonic dependency to rules in $P \setminus B$. By Definition 14 (c), all nonmonotonic dependencies are covered at all units. Therefore a rule $r \in w$, $w \in \{u'\} \cup V$ with $r \rightarrow_n s$, $s \in U$ implies that either $s \in w$, or that s is contained in a predecessor unit of w and therefore in u' or in V. Hence there are no nonmonotonic dependencies from rules in B to any rules not in B, and hence also not to rules in $P \setminus B$ and (c) holds. For (d) we know that R contains no constraints and by Proposition 1 all dependencies of non-constraints in R are covered by $\mathcal{E}$. Therefore $r \in R$, $r \rightarrow_{m,n} s$, $s \in P$ implies that $s \in R$ and the theorem holds. □

Proof of Proposition 4
(⇒) The added vertex m' is assigned to one unit and gets assigned a type. Furthermore, the graph stays acyclic as only outgoing edges from m' are added. I-connectedness is satisfied, as it is satisfied in $\mathcal{I}$ and we add no o-interpretation. O-connectedness is satisfied, as m' gets appropriate edges to o-interpretations at its predecessor units, and for other i-interpretations it is already satisfied in $\mathcal{I}$.

For FAI intersection, observe that if we add an edge (m', m_i) to $\mathcal{I}$ and it holds that $m_i \in \mathit{o\text{-}ints}_\mathcal{I}(u_i)$, then m' reaches in $\mathcal{I}$ only one o-interpretation at u_i, and due to O-connectedness that o-interpretation is connected to exactly one i-interpretation at u_i, which is part of the original graph $\mathcal{I}$ and therefore satisfies FAI intersection. Therefore it remains to show that the union of subgraphs of $\mathcal{I}$ reachable in $\mathcal{I}$ from m_1,…,m_k, contains one o-interpretation at each unit in the subgraph of $\mathcal{E}$ reachable from u_1,…,u_k. We make a case distinction.

Case (I): two o-interpretations $m_i \in \mathit{o\text{-}ints}_\mathcal{I}(u_i)$, $m_j \in \mathit{o\text{-}ints}_\mathcal{I}(u_j)$ in the join, with $1 \leq i < j \leq k$, have no common unit that is reachable in $\mathcal{E}$ from u_i and from u_j: then

the condition is trivially satisfied, as the subgraphs of $\mathcal{I}$ reachable in $\mathcal{I}$ from m_i and m_j, respectively, do not intersect at any unit.

Case (II): two o-interpretations $m_i \in o\text{-}ints_{\mathcal{I}}(u_i)$, $m_j \in o\text{-}ints_{\mathcal{I}}(u_j)$ in the join, with $1 \leq i < j \leq k$, have at least one common unit that is reachable from u_i and from u_j in $\mathcal{E}$. Let u^f be a unit reachable in $\mathcal{E}$ from both u_i and u_j on two paths that do not intersect before reaching u^f. From u_i to u^f, and from u_j to u^f, exactly one o-interpretation is reachable in $\mathcal{I}$ from m_i and m_j, respectively, as these paths do not intersect. u^f is a FAI of u, and as the join is defined, we reach in $\mathcal{E}$ exactly one o-interpretation at unit u^f from m_i and m_j. Due to O-connectedness, we also reach in $\mathcal{I}$ exactly one i-interpretation m'' at u^f from m_i and m_j. Now m'' is common to subgraphs of $\mathcal{I}$ that are reachable in $\mathcal{I}$ from m_i and m_j, and m'' satisfies FAI intersection in $\mathcal{I}$.

Consequently, FAI intersection is satisfied in $\mathcal{I}'$ for all pairs of predecessors of m' and therefore in all cases. As no vertex m with $\{(m, m_1), \ldots, (m, m_k)\} \subseteq F$ exists and and as $\mathcal{I}$ satisfies Uniqueness, also $\mathcal{I}'$ satisfies Uniqueness.

($\Leftarrow$) Assume towards a contradiction that $\mathcal{I}'$ is an i-graph but that the join is not defined. Then there exists some FAI $u' \in fai(u)$ such that either no or more than one o-interpretation from $o\text{-}ints_{\mathcal{I}}(u)$ is reachable in $\mathcal{I}$ from some m_i, $1 \leq i \leq k$. As $\mathcal{I}$ is an i-graph, due to I-connectedness and O-connectedness, if a unit u' is a FAI and therefore u' is reachable in $\mathcal{E}$ from u_i, then at least one i-interpretation and one o-interpretation at u' is reachable in $\mathcal{I}$ from m_i. If more than one o-interpretation is reachable in $\mathcal{I}$ from some m_i, $1 \leq i \leq k$, this means that more than one o-interpretation at u' is reachable in $\mathcal{I}'$ from the newly added i-interpretation m. However, this violates FAI intersection in $\mathcal{I}'$, which is a contradiction. Hence the result follows. $\square$

Proof of Proposition 5

($\Rightarrow$) Whenever the join is defined, $\mathcal{A}'$ is an i-graph by Proposition 4. It remains to show that $int(m')^+ \in \mathcal{AS}(u^<)$, and that $\mathcal{A}'$ fulfills items (a) and (c) of an answer set graph. By Theorem 4 we know that for each u_i, $u_i^\leq$ is a generalized bottom of $u^<$ wrt. the set $R_i = \{r \in u_i^\leq \mid B(r) \neq \emptyset\}$. For each u_i, therefore $Y \in \mathcal{AS}(u^<)$ iff $Y \in \mathcal{AS}(u^< \setminus R_i \cup facts(X))$ for some $X \in \mathcal{AS}(u_i^\leq)$. As $\mathcal{A}$ is an answer set graph, for each m_i we know that $int(m_i)^+ \in \mathcal{AS}(u_i^\leq)$; hence $Y \in \mathcal{AS}(u^<)$ if $Y \in \mathcal{AS}(u^< \setminus R_i \cup int(m_i)^+)$. Now from the evaluation graph properties we know that $u^< = u_1^\leq \cup \cdots \cup u_k^\leq$, and from the construction of $int(m')$ and its dependencies in $\mathcal{A}'$ we obtain that $int(m')^+ = int(m_1)^+ \cup \cdots \cup int(m_k)^+$. It follows that $int(m')^+ \in \mathcal{AS}(u^<)$, which satisfies condition (a). Due to the definition of join, condition (c) is also satisfied and $\mathcal{A}'$ is indeed an answer set graph.

($\Leftarrow$) As $\mathcal{A}'$ is an answer set graph, it is an i-graph, and hence by Proposition 4 $m = m_1 \bowtie \cdots \bowtie m_k$ is defined. $\square$

Proof of Theorem 5

We prove this theorem using Proposition 6. We construct $\mathcal{E}'' = (U'', E'')$ with $U'' = U \cup \{u_{final}\}$, $u_{final} = \emptyset$, and $E'' = E \cup \{(u_{final}, u) \mid u \in U\}$. As u_{final} contains no rules and as $\mathcal{E}''$ is acyclic, no evaluation graph property of gets violated and $\mathcal{E}''$ is also an evaluation graph. As $\mathcal{A}$ contains no interpretations at u_{final} and dependencies from units in U are the same in $\mathcal{E}$ and $\mathcal{E}''$, $\mathcal{A}$ is in fact an answer set graph for $\mathcal{E}''$. We now modify $\mathcal{A}$ to obtain $\mathcal{A}''$ as follows.

We add the set $M_{new} = \{m \mid m = m_1 \bowtie \cdots \bowtie m_n \text{ is defined at } u_{final} \text{ (wrt. } \mathcal{A})\}$ as i-interpretations of u_{final} and dependencies from each $m \in M_{new}$ to the respective o-interpretations m_i, $1 \leq i \leq n$. By Proposition 5, $\mathcal{A}''$ is an answer set graph for $\mathcal{E}''$, and moreover $\mathcal{A}''$ gets input-complete for u_{final} by construction. As $\mathcal{A}''$ is input-complete for $U \cup \{u_{final}\}$ and output-complete for U, by Proposition 6 we have that $\mathcal{AS}(P) = i\text{-}ints_{\mathcal{A}}(u_{final}) = M_{new}$. As for every join $m = m_1 \bowtie \cdots \bowtie m_n$, we have $int(m) = int(m_1) \cup \cdots \cup int(m_n)$, to complete the proof of the theorem, it remains to show that the join m between $m_1,\ldots,m_n$ is defined at u_{final} iff the subgraph $\mathcal{A}'$ of $\mathcal{A}$ reachable from the o-interpretations m_i in F fulfills $|o\text{-}ints_{\mathcal{A}}(u_i)| = 1$, for each $u_i \in U$. As the join involves all units in U, and since $\mathcal{A}''$ is an answer set graph and thus an i-graph, it follows from the conditions for an i-graph that at each $u_i \in U$ exactly one o-interpretation is reachable from m, and thus also from each m_i; thus the condition for $\mathcal{A}'$ holds. Conversely, if the subgraph $\mathcal{A}'$ fulfills $|o\text{-}ints_{\mathcal{A}}(u_i)| = 1$ for each $u_i \in U$, then clearly the FAI condition for the join m being defined is fulfilled. □

Proof of Proposition 6
As u_{final} depends on all units in $U \setminus \{u_{final}\}$, due to O-connectedness every i-interpretation $m \in i\text{-}ints_{\mathcal{A}}(u_{final})$ depends on one o-interpretation at every unit in $U \setminus \{u_{final}\}$. Let $U \setminus \{u_{final}\} = \{u_1, \ldots, u_k\}$ and let $M_M = \{m_1, \ldots, m_k\}$ be the set of o-interpretations such that $(m, m_i) \in F$ and $m_i \in o\text{-}ints_{\mathcal{A}}(u_i)$, $1 \leq i \leq k$. Then, due to FAI intersection, M_m contains each o-interpretation that is reachable from m in $\mathcal{A}$, and M_m contains only interpretations with this property. Hence $int(m)^+ = int(m_1) \cup \cdots \cup int(m_k)$, and due to condition (c) in Definition 19, we have $int(m) = int(m)^+$. By the dependencies of u_{final}, we have $u^{<}_{final} = P$, and as u_{final} is input-complete, we have that $\mathcal{AS}(P) = \mathcal{AS}(u^{<}_{final}) = \{int(m)^+ \mid m \in i\text{-}ints_{\mathcal{A}}(u_{final})\}$. As $int(m) = int(m)^+$ for every i-interpretation m at u_{final}, we obtain the result. □

Proof of Proposition 7
The proposition follows from Property 1, which asserts that the grounding P' has the same answer sets as P, and from the soundness and completeness of the evaluation algorithm for ground HEX-programs as asserted by Property 2. □

Proof of Theorem 6
We show by induction on its construction that $\mathcal{I} = (M, F, unit, type, int)$ is an answer set graph for $\mathcal{E}$, and that at the beginning of the while-loop $\mathcal{I}$ is input- and output-complete for $V \setminus U$.

(Base) Initially, $\mathcal{I}$ is initially and $V = U$, hence the base case trivially holds.

(Step) Suppose that $\mathcal{I}$ is an answer set graph for $\mathcal{E}$ at the beginning of the while-loop, and that it is input- and output-complete for $V \setminus U$. As the chosen u only depends on units in $V \setminus U$, it depends only on output-complete units. For a leaf unit u, (b) creates an empty i-interpretation and therefore makes u input-complete. For a non-leaf unit u, the first for-loop (c) builds all possible joins of interpretations at predecessors of u and adds them as i-interpretations to $\mathcal{I}$. As all predecessors of u are output-complete by the hypothesis, this makes u input-complete. Now suppose that Condition (d) is false, i.e., $u \neq u_{final}$. Then the second for-loop (e) evaluates u wrt. every i-interpretation at u and adds the result to

u as an o-interpretation. Due to Proposition 7, EVALUATELDESAFE$(u, int(m'))$ returns all interpretations o such that $o \in \{X \setminus int(m') \mid X \in \mathcal{AS}(u \cup facts(int(m'))\}$. As u depends on all units on which its rules depend, and as i-interpretations contain all atoms from o-interpretations of predecessor units (due to condition (c) of Definition 19), we have EVALUATELDESAFE$(u, int(m'))$ = EVALUATELDESAFE$(u, int(m')^+)$. By Theorem 3, $u^<$ is a generalized bottom of $u^\leq$, and by the induction hypothesis $int(m')^+ \in \mathcal{AS}(u^<)$; hence by Theorem 2, we have that $int(m')^+ \cup o \in \mathcal{AS}(u^\leq)$. Consequently, adding a new o-interpretation m with interpretation $int(m) = o$ and dependency to m' to the graph $\mathcal{I}$ results in $int(m)^+ \in \mathcal{AS}(u^\leq)$, and adding all of them makes $\mathcal{I}$ output-complete for u. Finally, in (f) u is removed from U; hence at the end of the while-loop $\mathcal{I}$ is an answer set graph and again input- and output-complete for $V \setminus U$.

It remains to consider the case where Condition (d) is true. Then u_{final} was made input-complete, which means that all predecessors of u_{final} are output-complete. As u_{final} depends on all other units, we have $U = \{u_{final}\}$ and the algorithm returns $i\text{-}ints_{\mathcal{A}}(u)$; by Proposition 6, it thus returns $\mathcal{AS}(P)$, which will happen in the $|V|$-th iteration of the while loop. □

Appendix B Example Run of Algorithm 2

We provide here an example run of Algorithm 2 for our running example.

Example 1 (*ctd.*)
Consider an evaluation graph $\mathcal{E}_2'$ which is $\mathcal{E}_2$ plus $u_{final} = \emptyset$, which depends on all other units. Following Algorithm 2 we first choose $u = u_1$, and as u_1 has no predecessor units, step (b) creates the i-interpretation m_1 with $int(m_1) = \emptyset$. As $u_1 \neq u_{final}$, we continue and in loop (e) obtain $O = \mathcal{AS}(u_1) = \big\{\{swim(ind)\}, \{swim(outd)\}\big\}$. We add both answer sets as o-interpretations m_2 and m_3 and then finish the outer loop with $U = \{u_2, u_3, u_4, u_{final}\}$. In the next iteration, we could choose $u = u_2$ or $u = u_3$; assume we choose u_2. Then $preds_{\mathcal{E}}(u_2) = \{u_1\}$ and $k = 1$, and we enter the loop (c) and build all joins that are possible with o-interpretations at u_1 (all joins are trivial and all are possible), i.e., we copy the interpretations and store them at u_2 as new i-interpretations m_4 and m_5. In the loop (e), we obtain O = EVALUATELDESAFE$(u_2, \{swim(ind)\}) = \emptyset$, as indoor swimming requires money which is excluded by $c_8 \in u_2$. Therefore i-interpretation $\{swim(ind)\}$ yields no o-interpretation, indicated by $\lightning$. However, we obtain O = EVALUATELDESAFE$(u_2, \{swim(outd)\}) = \{\emptyset\}$: as outdoor swimming neither requires money nor anything else, i-interpretation $\{swim(outd)\}$ derives no additional atoms and yields the empty answer set, which we store as o-interpretation m_6 at u_2; the iteration ends with $U = \{u_3, u_4, u_{final}\}$. In the next iteration we choose $u = u_3$, we add in loop (c) i-interpretations m_7 and m_8 to u_3, and in loop (e) o-interpretations $m_9, \ldots, m_{12}$ to u_3; the iteration ends with $U = \{u_4, u_{final}\}$. In the next iteration we choose $u = u_4$; this time we have multiple predecessors, and in loop (c) we check join candidates $m_6 \bowtie m_9$ and $m_6 \bowtie m_{10}$, which are both not defined. The other join candidates are $m_6 \bowtie m_{11}$ and $m_6 \bowtie m_{12}$, which are both defined; we thus add their results as i-interpretations m_{13} and m_{14}, respectively, to u_4. The loop (e) computes then one o-interpretation m_{15} for i-interpretation m_{13} and no o-interpretation for m_{14}. The iteration ends with $U = \{u_{final}\}$. In the next iteration, we

Algorithm 1: ANSWERSETSONDEMAND

Input: evaluation graph $\mathcal{E}$ for program P, with final unit $u_{final} = \emptyset$
Output: the answer sets of P

initialize global storage $\mathcal{S}$
repeat
  m_{out} := GETNEXTOUTPUTMODEL (u_{final})
  if $m_{out} \neq$ UNDEF **then** output m_{out}
until $m_{out} =$ UNDEF

have $preds_{\mathcal{E}}(u_{final}) = \{u_1, u_2, u_3, u_4\}$ and the loop (c) checks all combinations of one o-interpretation at each unit in $preds_{\mathcal{E}}(u_{final})$. Only one such join candidate is defined, namely $m = m_3 \bowtie m_6 \bowtie m_{11} \bowtie m_{15}$, whose result is stored as a new i-interpretation at u_{final}. The check (d) now succeeds, and we return all i-interpretations at u_{final}; i.e., we return $\{m\} = \big\{\{swim(outd), goto(altD), ngoto(gansD), go, need(loc, yogamat)\}\big\}$. This is indeed the set of answer sets of P_{swim}.

Appendix C On Demand Model Streaming Algorithm

Algorithm 2 fully evaluates all other units before computing results at the final evaluation unit u_{final}, and it keeps the intermediate results in memory. If we are only interested in one or a few answer sets, many unused results may be calculated.

Using the same evaluation graph, we can compute the answer sets with a different, more involved algorithm ANSWERSETSONDEMAND (shown in Algorithm 1) that operates demand-driven from units, starting with u_{final}, rather than data-driven from completed units. It uses in turn several building blocks that are shown in Algorithms 2–4

ANSWERSETSONDEMAND calls Algorithm GETNEXTOUTPUTMODEL for u_{final} and outputs its output models, i.e., the answer sets of the input program P given by the evaluation graph $\mathcal{E}$, one by one until it gets back UNDEF. Like Algorithm 2, GETNEXTOUTPUTMODEL builds in combination with the other algorithms an answer set graph $\mathcal{A}$ for $\mathcal{E}$ that is input-complete at all units, if all statements marked with '$(+)$' are included; omitting them, it builds $\mathcal{A}$ virtually and has at any time at most one input and one output model of each unit in memory.

Roughly speaking, the models at units are determined in the same order in which a right-to-left depth-first-traversal of the evaluation graph $\mathcal{E}$ would backtrack from edges. This is because first all models of the subgraph reachable from a unit u are determined, then models at the unit u, and then the algorithm backtracks. The models of the subgraph are retrieved with GETNEXTINPUTMODEL one by one, and using $NextAnswerSet$ the output models are generated and returned. The latter function is assumed to return, given a HEX-program P and the i-th element in an arbitrary but fixed enumeration $I_1, I_2, \ldots, I_m$ of the answer sets of P (without duplicates), the next answer set I_{i+1}, where by convention $I_0 =$ UNDEF and the return value for I_m is UNDEF. This is easy to provide on top of current solvers, and the incremental usage of $NextAnswerSet$ allows for an efficient stateful realization (e.g. answer set computation is suspended).

The trickiest part of this approach is GETNEXTINPUTMODEL, which has to create locally

Algorithm 2: GETNEXTOUTPUTMODEL(u)

Input: u: unit
Output: m_{out}: next omodel at u or UNDEF

if $refs\text{O}(u) > 0$ **then return** UNDEF
if $cur\text{I}(u) =$ UNDEF **then** $cur\text{I}(u) :=$ GETNEXTINPUTMODEL(u)
while $cur\text{I}(u) \neq$ UNDEF **do**
  $cur\text{O}(u) := NextAnswerSet(u \cup facts(cur\text{I}(u)), cur\text{O}(u))$
  if $cur\text{O}(u) \neq$ UNDEF **then**
(+) add omodel $cur\text{O}(u)$ to $\mathcal{A}$ with dependency to $cur\text{I}(u)$
    return $cur\text{O}(u)$
  $cur\text{I}(u) :=$ GETNEXTINPUTMODEL(u)
return UNDEF

Algorithm 3: ENSUREMODELINCREMENT(u, at)

Input: u: unit with $\{u_1, \ldots, u_k\} = preds_{\mathcal{E}}(u)$, at: index $1 \leq at \leq k$
Output: at': index $at \leq at' \leq k$ or UNDEF

repeat
  $refs\text{O}(u_{at}) := refs\text{O}(u_{at}) - 1$
  $m :=$ GETNEXTOUTPUTMODEL(u_{at})
  if $m =$ UNDEF **then** $at := at + 1$
  else
    $refs\text{O}(u_{at}) := refs\text{O}(u_{at}) + 1$
    return at
until $at = k + 1$
return UNDEF

and in an incremental fashion all joins that are globally defined, i.e., all combinations of incrementally available output models of predecessors which share a common predecessor model at all FAIs. To generate all combinations of output models in the right order, it uses the algorithm ENSUREMODELINCREMENT.

The algorithms operate on a global data structure $\mathcal{S} = (\mathcal{E}, \mathcal{A}, cur\text{I}, cur\text{O}, refs\text{O})$ called *storage*, where

- $\mathcal{E} = (U, E)$ is the evaluation graph containing $u_{final} \in U$,
- $\mathcal{A} = (M, F, unit, type, int)$ is the (virtually built) answer set graph,
- $cur\text{I} : U \to M \cup \{\text{UNDEF}\}$ and $cur\text{O} : U \to M \cup \{\text{UNDEF}\}$, are functions that informally associate with a unit u the current input respectively output model considered, and
- $refs\text{O} : U \to \mathbb{N} \cup \{0\}$ is a function that keeps track of how many current input models point to the current output model of u; this is used to ensure correct joins, by checking in GETNEXTOUTPUTMODEL that the condition (IG-F) for sharing models in the interpretation graph is not violated (for details see Section 5.1.2 and Definition 17).

Algorithm 4: GETNEXTINPUTMODEL(u)

Input: u: unit
Output: m_{out}: imodel at u or UNDEF

(a) **if** $preds_{\mathcal{E}}(u) = \emptyset$ **then**
  if $cur\text{I}(u) =$ UNDEF **then**
(+) add imodel $\emptyset$ at u to $\mathcal{A}$
    return $\emptyset$
  else return UNDEF
let $\{u_1, \ldots, u_k\} = preds_{\mathcal{E}}(u)$ /* assume this order is fixed for each unit u */
if $cur\text{I}(u) \neq$ UNDEF **then**
  $at :=$ ENSUREMODELINCREMENT$(u, 1)$
  if $at =$ UNDEF **then return** UNDEF
  $at := at - 1$
else $at := k$
(b) **while** $at \neq 0$ **do**
  if $cur\text{O}(u_{at}) \neq$ UNDEF **then**
    $refs\text{O}(u_{at}) := refs\text{O}(u_{at}) + 1$
    $at := at - 1$
  else
    $m :=$ GETNEXTOUTPUTMODEL(u_{at})
    if $m =$ UNDEF **then**
      if $at = k$ **then return** UNDEF
      $at :=$ ENSUREMODELINCREMENT$(u, at + 1)$
      if $at =$ UNDEF **then return** UNDEF
    else
      $refs\text{O}(u_{at}) := refs\text{O}(u_{at}) + 1$
      $at := at - 1$
let $m = cur\text{O}(u_1) \bowtie \cdots \bowtie cur\text{O}(u_k)$
(+) add imodel m to $\mathcal{A}$ with dependencies to $cur\text{O}(u_1), \ldots, cur\text{O}(u_k)$
return m

Initially, the storage $\mathcal{S}$ is empty, i.e., it contains the input evaluation graph $\mathcal{E}$, an empty answer set graph $\mathcal{A}$, and the functions are set to $cur\text{I}(u) =$ UNDEF, $cur\text{O}(u) =$ UNDEF, and $refs\text{O}(u) = 0$ for all $u \in U$. The call of GETNEXTOUTPUTMODEL for u_{final} triggers the right-to-left depth-first traversal of the evaluation graph.

We omit tracing Algorithm ANSWERSETSONDEMAND on our running example, as this would take quite some space; however, one can check that given the evaluation graph $\mathcal{E}_2$, it correctly outputs the single answer set

$$I = \{swim(outd), goto(altD), ngoto(gansD), go, need(loc, yogamat)\}.$$

Formally, it can be shown that given an evaluation graph $\mathcal{E} = (U, E)$ of a program P such that $\mathcal{E}$ contains a final unit $u_{final} = \emptyset$, Algorithm ANSWERSETSONDEMAND outputs one

by one all answer sets of P, without duplicates, and that in the version without $(+)$-lines, it stores at most one input and one output model per unit (hence the size of the used storage is linear in the size of the ground program $grnd(P)$).

Appendix D Overview of Liberal Domain-Expansion Safety

Strong domain-expansion safety is overly restrictive, as it also excludes programs that clearly *are* finitely restrictable. In this section we give an overview about the notion and refer to (Eiter et al. 2014) for details.

Example 2
Consider the following program:

$$P = \left\{ \begin{array}{ll} r_1 \colon p(a). & r_3 \colon s(Y) \leftarrow p(X), \&concat[X, a](Y). \\ r_2 \colon q(aa). & r_4 \colon p(X) \leftarrow s(X), q(X). \end{array} \right\}$$

It is not strongly safe because Y in the cyclic external atom $\&concat[X, a](Y)$ in r_3 does not occur in an ordinary body atom that does not depend on $\&concat[X, a](Y)$. However, P is finitely restrictable as the cycle is "broken" by $dom(X)$ in r_4.

To overcome unnecessary restrictions of strong safety in (Eiter et al. 2006), *liberal domain-expansion safety* (lde-safety) has been introduced (Eiter et al. 2014), which incorporates both syntactic and semantic properties of a program. The details of the notion are not necessary for this paper, except that all lde-safe programs have finite groundings with the same answer sets; we give here a brief overview.

Unlike strong safety, liberal de-safety is not a property of entire atoms but of *attributes*, i.e., pairs of predicates and argument positions. Intuitively, an attribute is lde-safe, if the number of different terms in an answer-set preserving grounding (i.e. a grounding which has the same answer sets if restricted to the positive atoms as the original program) is finite. A program is lde-safe, if all its attributes are lde-safe.

The notion of lde-safety is designed in an extensible fashion, i.e., such that several safety criteria can be easily integrated. For this we parametrize our definition of lde-safety by a *term bounding function (TBF)*, which identifies variables in a rule that are ensured to have only finitely many instantiations in the answer set preserving grounding. Finiteness of the overall grounding follows then from the properties of TBFs.

For an ordinary predicate $p \in \mathcal{P}$, let $p{\restriction}i$ be the *i-th attribute of p* for all $1 \leq i \leq ar(p)$. For an external predicate $\&g \in \mathcal{X}$ with input list $\mathbf{X}$ in rule r, let $\&g[\mathbf{X}]_r{\restriction}_T i$ with $T \in \{\text{I}, \text{O}\}$ be the *i-th input resp. output attribute of $\&g[\mathbf{X}]$ in r* for all $1 \leq i \leq ar_T(\&g)$. For a ground program P, the *range* of an attribute is, intuitively, the set of ground terms which occur in the position of the attribute. Formally, for an attribute $p{\restriction}i$ we have $range(p{\restriction}i, P) = \{t_i \mid p(t_1, \ldots, t_{ar(p)}) \in A(P)\}$; for an attribute $\&g[\mathbf{X}]_r{\restriction}_T i$ we have $range(\&g[\mathbf{X}]_r{\restriction}_T i, P) = \{x_i^T \mid \&g[\mathbf{x}^{\text{I}}](\mathbf{x}^{\text{O}}) \in EA(P)\}$, where $\mathbf{x}^s = x_1^s, \ldots, x_{ar_s(\&g)}^s$.

We use the following monotone operator to compute by fixpoint iteration a finite subset of $grnd(P)$ for a program P:

$$G_P(P') = \bigcup_{r \in P} \{r\theta \mid \exists I \subseteq \mathcal{A}(P'), I \not\models \bot, I \models B^+(r\theta)\},$$

where $\mathcal{A}(P') = \{\mathbf{T}a, \mathbf{F}a \mid a \in A(P')\} \setminus \{\mathbf{F}a \mid a \leftarrow . \in P\}$ and $r\theta$ is the ground instance of r under variable substitution $\theta\colon \mathcal{V} \to \mathcal{C}$. Note that in this definition, I might be partial, but by convention we assume that all atoms which are not explicitly assigned to true are false. That is, G_P takes a ground program P' as input and returns all rules from $grnd(P)$ whose positive body is satisfied under some assignment over the atoms of Π'. Intuitively, the operator iteratively extends the grounding by new rules if they are possibly relevant for the evaluation, where relevance is in terms of satisfaction of the positive rule body under some assignment constructable over the atoms which are possibly derivable so far. Obviously, the least fixpoint $G_P^\infty(\emptyset)$ of this operator is a subset of $grnd(P)$; we will show that it is finite if P is lde-safe according to our new notion. Moreover, we will show that this grounding preserves all answer sets as all omitted rule instances have unsatisfied bodies anyway.

Example 3
Consider the following program P:

$$r_1\colon\ s(a). \quad r_2\colon\ dom(ax). \quad r_3\colon\ dom(axx).$$
$$r_4\colon\ s(Y) \leftarrow s(X), \&concat[X, x](Y), dom(Y).$$

The least fixpoint of G_P is the following ground program:

$$r'_1\colon\ \ s(a). \quad r'_2\colon\ dom(ax). \quad r'_3\colon\ dom(axx).$$
$$r'_4\colon\ \ s(ax) \leftarrow s(a), \&concat[a, x](ax), dom(ax).$$
$$r'_5\colon\ s(axx) \leftarrow s(ax), \&concat[ax, x](axx), dom(axx).$$

Rule r'_4 is added in the first iteration and rule r'_5 in the second.

Towards a definition of lde-safety, we say that a term in a rule is *bounded*, if the number of substitutions in $G_P^\infty(\emptyset)$ for this term is finite. This is abstractly formalized using *term bounding functions*.

Definition 1 (*Term Bounding Function (TBF)*)
A *term bounding function*, denoted $b(P, r, S, B)$, maps a program P, a rule $r \in P$, a set S of (already safe) attributes, and a set B of (already bounded) terms in r to an enlarged set of (bounded) terms $b(P, r, S, B) \supseteq B$, such that every $t \in b(P, r, S, B)$ has finitely many substitutions in $G_P^\infty(\emptyset)$ if (i) the attributes S have a finite range in $G_P^\infty(\emptyset)$ and (ii) each term in $terms(r) \cap B$ has finitely many substitutions in $G_P^\infty(\emptyset)$.

Intuitively, a TBF receives a set of already bounded terms and a set of attributes that are already known to be lde-safe. Taking the program into account, the TBF then identifies and returns further terms which are also bounded.

The concept yields lde-safety of attributes and programs from the boundedness of variables according to a TBF. We provide a mutually inductive definition that takes the empty set of lde-safe attributes $S_0(P)$ as its basis. Then, each iteration step $n \geq 1$ defines first the set of bounded terms $B_n(r, P, b)$ for all rules r, and then an enlarged set of lde-safe attributes $S_n(P)$. The set of lde-safe attributes in step $n+1$ thus depends on the TBF, which in turn depends on the domain-expansion safe attributes from step n.

Definition 2 (*Liberal Domain-Expansion Safety*)

Let b be a term bounding function. The set $B_n(r, P, b)$ of *bounded terms* in a rule $r \in P$ in step $n \geq 1$ is $B_n(r, P, b) = \bigcup_{j \geq 0} B_{n,j}(r, P, b)$ where $B_{n,0}(r, P, b) = \emptyset$ and for all $j \geq 0$, $B_{n,j+1}(r, P, b) = b(P, r, S_{n-1}(P), B_{n,j})$.

The set of *domain-expansion safe attributes* $S_\infty(P) = \bigcup_{i \geq 0} S_i(P)$ of a program P is iteratively constructed with $S_0(P) = \emptyset$ and for $n \geq 0$:

- $p\restriction i \in S_{n+1}(P)$ if for each $r \in P$ and atom $p(t_1, \ldots, t_{ar(p)}) \in H(r)$, we have that term $t_i \in B_{n+1}(r, P, b)$, i.e., t_i is *bounded*;
- $\&g[\mathbf{X}]_r\restriction_{\text{I}} i \in S_{n+1}(P)$ if each $\mathbf{X}_i$ is a *bounded* variable, or $\mathbf{X}_i$ is a predicate input parameter p and $p\restriction 1, \ldots, p\restriction ar(p) \in S_n(P)$;
- $\&g[\mathbf{X}]_r\restriction_{\text{O}} i \in S_{n+1}(P)$ if and only if r contains an external atom $\&g[\mathbf{X}](\mathbf{Y})$ such that $\mathbf{Y}_i$ is bounded, or $\&g[\mathbf{X}]_r\restriction_{\text{I}} 1, \ldots, \&g[\mathbf{X}]_r\restriction_{\text{I}} ar_{\text{I}}(\&g) \in S_n(P)$.

A program P is *liberally domain-expansion (lde) safe*, if it is safe and all its attributes are domain-expansion safe.

A detailed description of liberal safety is beyond the scope of this paper. However, it is crucial that each liberally domain-expansion safe HEX-program P is finitely restrictable, i.e., there is a finite subset P_g of $grnd_C(P)$ s.t. $\mathcal{AS}(P_g) = \mathcal{AS}(grnd_C(P))$. A concrete grounding algorithm GROUNDHEX is given in (Eiter et al. 2014); we use GROUNDHEX(P) in this article to refer to a finite grounding of P that has the same answer sets.

References

EITER, T., FINK, M., KRENNWALLNER, T., AND REDL, C. 2014. Domain expansion for ASP-programs with external sources. Tech. Rep. INFSYS RR-1843-14-02, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria.

EITER, T., IANNI, G., SCHINDLAUER, R., AND TOMPITS, H. 2006. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *European Semantic Web Conference (ESWC)*. Springer, 273–287.