A Additional Details on Experiments

In this section, we provide additional information on our experiments and plots. Source code, data, and results are available online at https://tinyurl.com/nagg-supplementary-material.

As noted in the main matter, we facilitate hybrid grounding in our benchmarks, where we split up benchmark encodings into a *hard* part Π_t that is grounded by `NaGG` and the remainder Π_c that is grounded by `gringo` or `idlv`. The implementation `NaGG` thereby first applies a selected rewriting procedure. The concrete ASP encodings for scenarios S1–S4 are available at https://tinyurl.com/nagg-supplementary-material; these encodings also highlight the corresponding suprogram Π_t subject to rewriting. Afterwards, we ground Π_t with `NaGG` and Π_c with `gringo` or `idlv`. When comparing `NaGG` to the other grounders `gringo` and `idlv`, these grounders take the instance *without* aggregate rewritings, since the rewritings slow down those grounders.

A.1 Additional Plots

In Figures 5 we show the combined solving (grounding) profile of S3-T4, S3-T8, and S4-T6. Circles mark instances solved (grounded) within 1800s and 32GB RAM. Note the significantly reduced grounding size, the additional number of solved (grounded) instances, and the – compared to `gringo` or `idlv`– weaker dependence of `NaGG` on instance density, instance size and rule density.

In Figures 6 we show in the left column the combined grounding profile of S4-T4 for `NaGG`-`gringo`. In the right column we show the results of S1. For S1 notice the reduced grounding and solving times of `NaGG-idlv` and `NaGG-gringo`, when compared to `idlv` and `gringo`. Additionally, note the jump in grounding size and grounding time at instance number 27. This is due to an increase in the instance size. More specifically, all instances below instance number 27 have 75 individuals of two types of people, whereas starting with instance number 27, this increases to 100. As the number of lines per input instance is quadratic in the number of individuals, this raises the number of input lines from 11250 (below 27) to 20000 (27 and after). More specifically, let p be the number of individuals, then $2 \cdot p^2$ is the number of input lines. This increase leads to a more than doubling of grounding time and solving time.

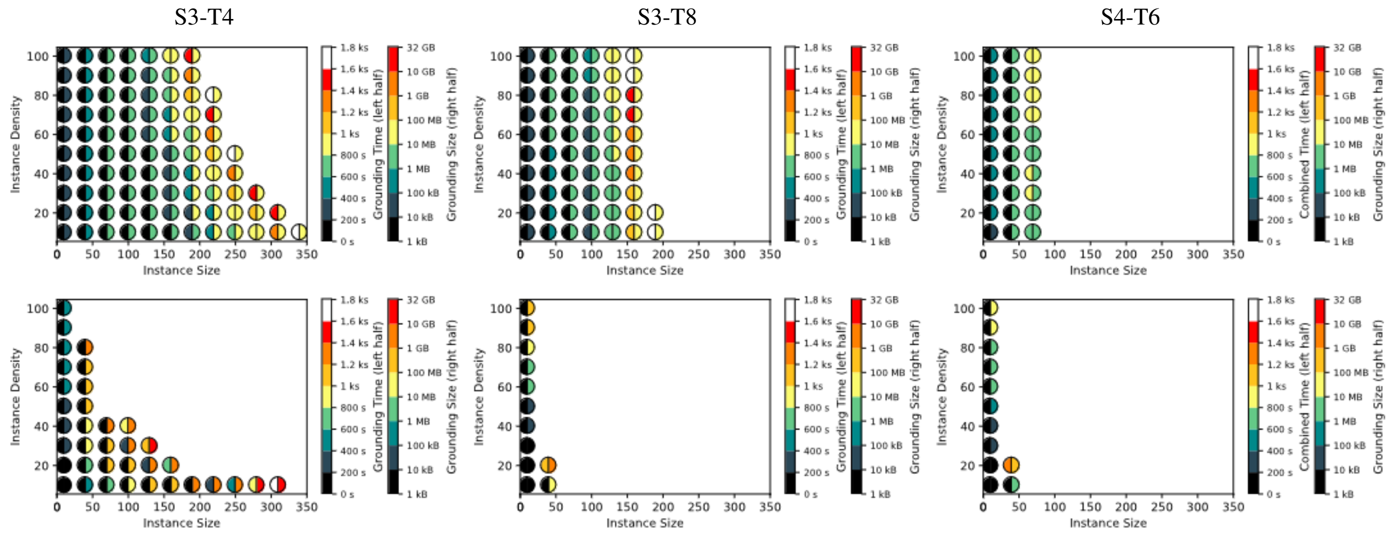

Figure 5: Combined solving profile of `NaGG-gringo`/`NaGG-idlv` (top row) and `gringo`/`idlv` (bottom row). S3-T4 (first column; top: `NaGG-gringo`, bottom: `gringo`), S3-T8 (second column; top: `NaGG-gringo`, bottom: `gringo`), S4-T6 (third column; top: `NaGG-idlv`, bottom: `idlv`). Circles mark instances solved (grounded) within 1800s and 32GB RAM.

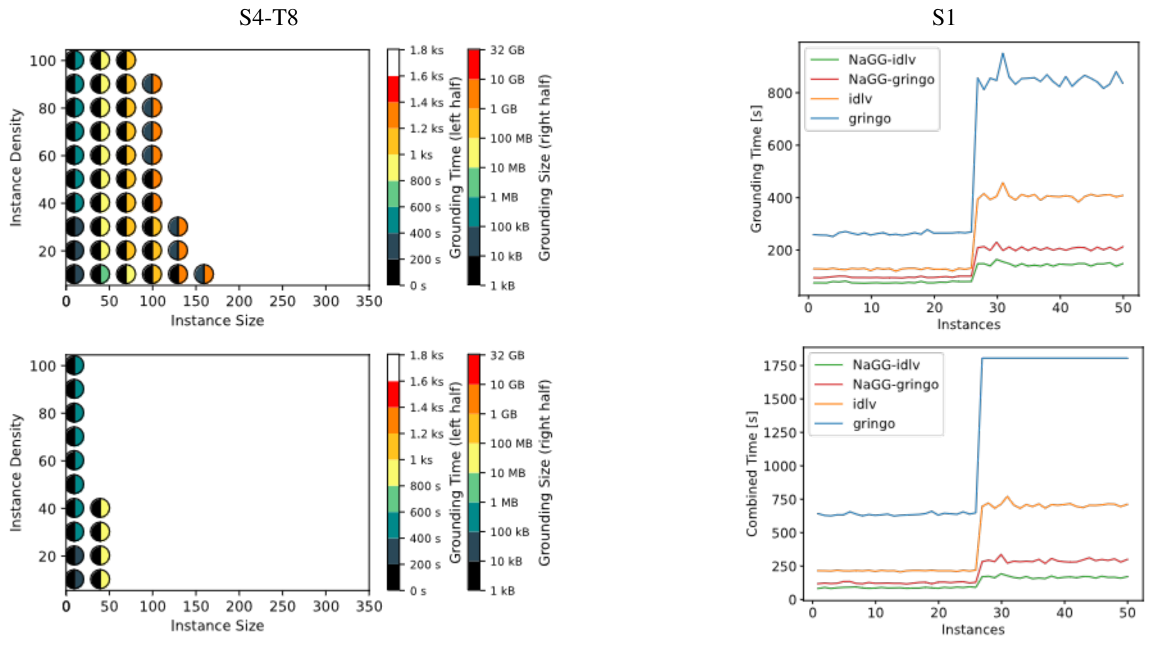

Figure 6: Combined solving profile of S4-T8 (left column), of `NaGG-gringo` (top row) and `gringo` (bottom row). Circles mark grounded instances within 1800s and 32GB RAM. Grounding and combined (grounding and solving) time of S1 (right column). Time limit 1800s.

Replacing Rules (3) for (Shared) SCCs by:

$$\mathcal{G}(\{r'\}) \qquad \text{for every } r' \in \Pi_c, \text{ with } \forall S \in \text{scc}(\Pi): (|E(S_{\Pi_t})| = 0 \text{ or } |E(S_{\{r'\}})| = 0) \tag{15}$$

$$a \leftarrow a_{l+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n, (a_{l+1} \prec a), \ldots, (a_m \prec a) \qquad \text{for every } r' \in \Pi_c, \text{ with } \exists S \in \text{scc}(\Pi): (|E(S_{\Pi_t})| \geq 1 \text{ and } |E(S_{\{r'\}})| \geq 1), r \in \mathcal{G}(\{r'\}), H_r = \{a_1, \ldots, a_l\}, a \in H_r, B_r^+ = \{a_{l+1}, \ldots, a_m\}, a \cup B_r^- = \{a, a_{m+1}, \ldots, a_n\} \tag{16}$$

$$\leftarrow a_{l+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n, \neg a \qquad \text{for every } r' \in \Pi_c, \text{ with } \exists S \in \text{scc}(\Pi): (|E(S_{\Pi_t})| \geq 1 \text{ and } |E(S_{\{r'\}})| \geq 1), r \in \mathcal{G}(\{r'\}), H_r = \{a_1, \ldots, a_l\}, a \in H_r, B_r^+ = \{a_{l+1}, \ldots, a_m\}, a \cup B_r^- = \{a, a_{m+1}, \ldots, a_n\} \tag{17}$$

Additional Rules for Foundedness of (Shared) SCCs:

$$[p_1(\mathbf{D}_1) \prec p_2(\mathbf{D}_2)] \vee [p_2(\mathbf{D}_2) \prec p_1(\mathbf{D}_1)] \leftarrow \qquad \text{for every SCC } S \in \text{scc}(\Pi) \text{ with } |E(S_{\Pi_t})| \geq 1, p_1(\mathbf{X}_1), p_2(\mathbf{X}_2) \in S, \mathbf{D}_1 \in \text{dom}(\mathbf{X}_1), \mathbf{D}_2 \in \text{dom}(\mathbf{X}_2), p_1(\mathbf{D}_1) \neq p_2(\mathbf{D}_2) \tag{18}$$

$$\leftarrow [p_1(\mathbf{D}_1) \prec p_2(\mathbf{D}_2)], [p_2(\mathbf{D}_2) \prec p_3(\mathbf{D}_3)], [p_3(\mathbf{D}_3) \prec p_1(\mathbf{D}_1)] \qquad \text{for every SCC } S \in \text{scc}(\Pi) \text{ with } |E(S_{\Pi_t})| \geq 1, p_1(\mathbf{X}_1), p_2(\mathbf{X}_2), p_3(\mathbf{X}_3) \in S, \mathbf{D}_1 \in \text{dom}(\mathbf{X}_1), \mathbf{D}_2 \in \text{dom}(\mathbf{X}_2), \mathbf{D}_3 \in \text{dom}(\mathbf{X}_3), p_1(\mathbf{D}_1) \neq p_2(\mathbf{D}_2), p_2(\mathbf{D}_2) \neq p_3(\mathbf{D}_3), p_1(\mathbf{D}_1) \neq p_3(\mathbf{D}_3) \tag{19}$$

$$\text{uf}_r(\mathbf{D}_{\mathbf{X}}) \leftarrow \text{uf}_{y_1}(\mathbf{D}_{\langle \mathbf{X}, y_1 \rangle}), \ldots, \text{uf}_{y_\ell}(\mathbf{D}_{\langle \mathbf{X}, y_\ell \rangle}), \neg[p(\mathbf{D}_Y) \prec h_r(D_{\mathbf{X}})] \qquad \text{for every SCC } S \in \text{scc}(\Pi) \text{ with } |E(S_{\Pi_t})| \geq 1, h(\mathbf{X}) \in S, p(\mathbf{Y}) \in B_r^+, \mathbf{D} \in \text{dom}(\langle \mathbf{X}, \mathbf{Y} \rangle), \mathbf{Y} = \langle y_1, \ldots, y_\ell \rangle, p(\mathbf{D}_{\mathbf{Y}}) \notin \mathcal{F} \tag{20}$$

$$\text{uf}_{r_r}(\mathbf{D}_{\mathbf{X}}) \leftarrow \neg[h_r(\mathbf{D}_Y) \prec h(D_{\mathbf{X}})] \qquad \text{for every SCC } S \in \text{scc}(\Pi) \text{ with } |E(S_{\Pi_t})| \geq 1, h(\mathbf{X}) \in S, p(\mathbf{Y}) \in B_r^+, \mathbf{D} \in \text{dom}(\langle \mathbf{X}, \mathbf{Y} \rangle), \mathbf{Y} = \langle y_1, \ldots, y_\ell \rangle, p(\mathbf{D}_{\mathbf{Y}}) \notin \mathcal{F} \tag{21}$$

Replace Rules (2) with:

$$h_r(\mathbf{D}) \vee \overline{h_r(\mathbf{D})} \leftarrow \qquad h(\mathbf{D}) \leftarrow h_r(\mathbf{D}) \qquad \text{for every } r \in \Pi_t, h(\mathbf{X}) \in \text{hpred}(r), \mathbf{D} \in \text{dom}(\mathbf{X}) \tag{22}$$

Replace Rules (7) with:

$$\text{sat}_r \leftarrow \text{sat}_{x_1}(\mathbf{D}_{\langle x_1 \rangle}), \ldots, \text{sat}_{x_\ell}(\mathbf{D}_{\langle x_\ell \rangle}), h(\mathbf{D}) \qquad \text{for every } r \in \Pi_t, h(\mathbf{X}) \in H_r, \mathbf{D} \in \text{dom}(\mathbf{X}), \mathbf{X} = \langle x_1, \ldots, x_\ell \rangle \tag{23}$$

Replace Rules (12) with:

$$\leftarrow \text{uf}_r(\mathbf{D}), h_r(\mathbf{D}) \qquad \leftarrow \text{uf}_{r_r}(\mathbf{D}), h_r(\mathbf{D}) \qquad \text{for every } h(\mathbf{X}) \in \text{hpred}(\Pi_t), \mathbf{D} \in \text{dom}(\mathbf{X}), \{r_1, \ldots, r_m\} = \{r \in \Pi_t \mid h(\mathbf{X}) \in H_r\} \tag{24}$$

Figure 7: We extend the hybrid grounding procedure $\mathcal{H}$ of Figure 1 to the hybrid grounding procedure for shared cycles $\mathcal{H}_{lv}$, which, although similar, adds level mappings to deal with shared cycles between Π_c and Π_t. More precisely, Rules (15), (16) and (17) replace Rules (3), where non-ground rules contained in Π_c, i.e., those that shall be grounded via traditional grounding, are adapted to consider the level mapping. Further, by adding Rules (18), (19), (20) and (21), we ensure total level mappings, transitivity, and foundedness, respectively. Additionally, Rules (22), and (24) (and thereby replacing Rules (2), and (12)) we ensure *local foundedness* for each rule in Π_t s.t. it is known which h_r founds which h, which is necessary for level mappings. Lastly, we replace Rule (7) with Rule (23), so a rule is satisfied when h is in the answer set. This is important as it ensures that a rule is not "forced to found" some h_r, when it can not do that due to ordering constraints.

B Additional Proofs of Section 3

First, we state the grounding time of $\mathcal{H}$.

Proposition 2 (Runtime)**.** *Let Π be any tight non-ground program, where predicate arities are bounded by a. Then, $\mathcal{H}(\emptyset, \Pi)$ runs in time $\mathcal{O}((|\Pi| + |\mathrm{at}(\Pi)|) \cdot |\mathrm{dom}(\Pi)|^{2 \cdot a})$.*

Proof (Sketch). (i) Observe that by $\mathcal{H}$ each rule $r \in \Pi$ has to be grounded by Rules (2)–(12), so $|\Pi| + |\mathrm{at}(\Pi)|$ many.

(ii) Further, for each such rule $r \in \Pi \cup \mathrm{at}(\Pi)$ the grounding time is only dependent on the respective grounding times of Rules (2)–(12).

(iii) Note that for each rule of the Rules (2)–(8), we instantiate the variables of exactly one predicate. In the worst case, this predicate has arity a and therefore, these rules can be grounded in $\mathcal{O}(|\mathrm{dom}(\Pi)|^{a}))$ time (and size). In contrast to this, the Rules (9)–(12) instantiate at maximum two predicates. Therefore, their worst-case grounding time (and size) is $\mathcal{O}(|\mathrm{dom}(\Pi)|^{2 \cdot a}))$.

By combining (i), (ii), and (iii), we get $\mathcal{O}((|\Pi| + |\mathrm{at}(\Pi)|) \cdot |\mathrm{dom}(\Pi)|^{2 \cdot a})$. (8) □

As a side remark, we note that one can significantly reduce this worst-case scenario for some important scenarios. One such example is depicted in Example 3, where parts of the evaluation of the body, are independent of the evaluation in the head. In Example 3 this is the case, as the evaluation of $e(A, B), e(A, C), e(B, C)$ is independent of the evaluation of $a(X)$.

Proposition 1 (Runtime)**.** *Let Π be any tight non-ground program, where predicate arities are bounded by a. Then, $\mathcal{H}_{lv}(\emptyset, \Pi)$ runs in time $\mathcal{O}((|\Pi| + |\mathrm{at}(\Pi)|) \cdot |\mathrm{dom}(\Pi)|^{3 \cdot a})$.*

Proof (Sketch). (i) Observe that by $\mathcal{H}_{lv}$ each rule $r \in \Pi$ has to be grounded by Rules (2)–(12), and Rules (15)–(24), and these are $|\Pi| + |\mathrm{at}(\Pi)|$ many.

(ii) Further, for each such rule $r \in \Pi \cup \mathrm{at}(\Pi)$ the grounding time is only dependent on the respective grounding times of Rules (2)–(12), and Rules (15)–(24).

(iii) Remind yourself that by Observation 2, Rules (2)–(12) are grounded in $\mathcal{O}((|\Pi| + |\mathrm{at}(\Pi)|) \cdot |\mathrm{dom}(\Pi)|^{2 \cdot a})$.

(iv) As Rules (15)–(17) are grounded by traditional means, but $\Pi_c = \emptyset$, the respective runtime of these rules are constant. However, for Rules (18)–(21) we instantiate a maximum of 3 predicates (in Rule (19)), and therefore this part runs in $\mathcal{O}(|\mathrm{dom}(\Pi)|^{3 \cdot a}))$. Lastly, Rules (22)–(24) instantiate one predicate, which runs in $\mathcal{O}(|\mathrm{dom}(\Pi)|^{a}))$.

By combining (i), (ii), (iii), and (iv), we get $\mathcal{O}((|\Pi| + |\mathrm{at}(\Pi)|) \cdot |\mathrm{dom}(\Pi)|^{3 \cdot a})$. □

Hybrid Grounding with Level Mappings. Figure 7 depicts the extended hybrid grounding procedure $\mathcal{H}_{lv}$, which is similar to $\mathcal{H}$ of Figure 1, but adds level mappings to deal with shared cycles between Π_c and Π_t. More precisely, Rules (15), (16) and (17) replace Rules (3), where non-ground rules contained in Π_c, i.e., those that shall be grounded via traditional grounding, are adapted to consider the level mapping. Further, by adding Rules (18), (19), (20) and (21), we ensure total level mappings, transitivity, and foundedness, respectively. Additionally, Rules (22), and (24). (and thereby replacing Rules (2), and (12)) we ensure *local foundedness* for each rule in Π_t s.t. it is known which h_r founds which h, which is necessary for level mappings. Lastly, we replace Rule (7) with Rule (23), so a rule is satisfied when h is in the answer set. This is important as it ensures that a rule is not "forced to found" some h_r, when it can not do that due to ordering constraints.

Theorem 1 (Grounding-Splitting Theorem Normal)**.** *Given a partition of any non-ground HCF program Π into a HCF program Π_c and a tight program $\Pi_t = \Pi \setminus \Pi_c$. The answer sets of $\mathcal{H}_{lv}(\Pi_c, \Pi_t)$ restricted to $\mathrm{at}(\mathcal{G}(\Pi))$ match those of $\mathcal{G}(\Pi)$.*

Proof (Sketch). We briefly sketch both directions. In general we want to prove two directions. For the first direction ("$\Longrightarrow$"), assume an answer set M' of $\mathcal{H}_{lv}(\Pi_c, \Pi_t)$; we check that M' restricted to $\mathrm{at}(\mathcal{G}(\Pi))$ is an answer set of $\mathcal{G}(\Pi)$. For the second direction ("$\Longleftarrow$") we assume an answer set M of $\mathcal{G}(\Pi)$, and then check that there exists an answer set M' that equals M with restriction to $\mathrm{at}(\mathcal{G}(\Pi))$.

"$\Longrightarrow$": Let M' be an answer set of $\mathcal{H}_{lv}(\Pi_c, \Pi_t)$. From this, we define an interpretation $M = M' \cap \mathrm{at}(\Pi)$. It remains to show that M is indeed an answer set of $\mathcal{G}(\Pi)$. To ensure that $\mathcal{G}(\Pi)$ is indeed an answer set, we need to check satisfiability and foundedness. First we check satisfiability and then foundedness.

Satisfiability: Outline of the proof: We assume that M does not satisfy Π, from where we derive that M' also cannot be an answer set, which contradicts our assumption.

So, towards a contradiction, we assume that M does not satisfy Π. As M does not satisfy Π, there has to be a rule r', s.t. $B^{+}_{r'} \subseteq M$, $B^{-}_{r'} \cap M = \emptyset$, and $H_r \cap M = \emptyset$. Here we distinct three cases:

1. r' is in no SCC and $r' \in \Pi_c$: Then there is an exact correspondence between r' in Π and by the rule r'' from Equation (15) in $\mathcal{H}_{lv}(\Pi_c, \Pi_t)$. Therefore r'' is not satisfied, which contradicts our assumption.
2. r' is in an SCC and $r' \in \Pi_c$: Then r' is rewritten into two rules, by Equations (16) and (17). Crucially Equation (17) is a constraint, which holds exactly when $B^{+}_{r'} \subseteq M$, $B^{-}_{r'} \cap M = \emptyset$, and $H_r \cap M = \emptyset$. Therefore M' cannot be an answer set, which contradicts our assumption.
3. $r' \in \Pi_t$: Then r' is grounded purely by our techniques. The Equations (5), (6) and (23) ensure that $sat_{r'}$ is then not derived. Therefore by Equation (4) sat is not derived and the constraint in Equation (8) prevents M' from being an answer set, which contradicts our assumption.

As all cases contradict our assumption, we derive that M has to be satisfied.

Foundedness: We continue with the foundedness of M, where it needs to be that each predicate $p \in M$ needs to be founded. This check can be split into two general cases, the first one checking that for each $p \in M$ there exists at least one rule that is suitable for justifying p (1), and the second one being the ordering constraint (2). I.e., there must exist an ordering ψ, $\psi : I \to \{0, \ldots, |I| - 1\}$, s.t. for each $p \in M$ there exists a rule r, s.t. $\forall b \in B^{+}_{r} : \psi(b) < \psi(p)$.

Suitable for Justifying: This can be proved by a proof by contradiction, similar to the proof in the satisfiability part. Due to the fact that we present a proof sketch and not a full proof, we just outline the general procedure at this point. By contradiction, we assume that there exists at least one $p \in M$ that is unfounded due to the reason that there does not exist a rule $r \in \Pi$ s.t. r is suitable for justifying p. We then create a candidate set Ψ of rules that could justify p, i.e., the heads of the rules Ψ intersecting with p. Then we distinguish for each $r \in \Psi$ three exact same cases as in the satisfiability proof and derive for each case that if r is not suitable for justifying p in $\mathcal{G}(\Pi)$, the derived rule(s) in $\mathcal{H}_{lv}(\Pi_c, \Pi_t)$ are also not able to justify p. But from this it follows that no rule in $\mathcal{H}_{lv}(\Pi_c, \Pi_t)$ is suitable to justify p, and therefore M' is not an answer set, which contradicts our assumption.

In more detail, cases 1 and 2 (where $r \in \Pi_c$) are similar, s.t. it needs to be argued that for each such r, there exists one corresponding rule in $\mathcal{H}_{lv}(\Pi_c, \Pi_t)$ and this corresponding rule (with interpretation M') cannot found p due to the same reasons as r is not founded by M (Equations (17)–(15)).

Case 3 is an interesting case, as we have $r \in \Pi_t$. Remind yourself that we assumed $p \in M$, but p is unfounded. Then due to cases 1 and 2, and Equations (22), we follow $p_r \in M'$. And further, the constraints from Equations (24) do not fire, as otherwise M' would not be an answer set. From this we conclude $uf_r \notin M'$, and $uf_{r_r} \notin M'$. Then by Equations (10)–(11), $\forall b_+ \in B_r^+, b_+ \in M'$ and $\forall b_- \in B_r^-, b_- \notin M'$. But this is a contradiction, as then by construction, there has to exist a rule that founds p.

Ordering: The overall goal is to construct an ordering ψ, $\psi : I \to \{0, \dots, |I| - 1\}$, s.t. for each $p \in M$ there exists a rule r, s.t. $\forall b \in B_r^+ : \psi(b) < \psi(p)$.

For this first observe the following: By our Equations (18) and (19) we derive an ordering for each $S \in SCC$. Further, by Equations (16), (21) and (20), it is ensured that a predicate may only be derived if its positive body predicates are derived beforehand, for each $S \in SCC$. Therefore, we have "local" ordering functions defined, one per $S \in SCC$, which we denote ψ_S. The "local" validity of ψ_S is ensured by construction.

The next step is to combine them s.t. they form an overall coherent ordering ψ. Without going into the details, one can do that by analyzing the dependency graph D_Π and observing the incoming and outgoing edges of each strongly connected component. To be more specific, if one replaces each $S \in SCC$ in the dependency graph D_Π with a single vertex, but leaves the incoming and outgoing edges in place, one derives (multiple) directed acyclic graphs (DAG). Let DAG be the set of such DAGs, then we create one ordering per DAG ($d \in DAG$), denoted as ψ_d. Such a ψ_d is created by traversing the DAG d by starting at the source nodes and filling up ψ_d according to the number of nodes already visited before the current node. Therefore, every such ψ_d is valid by construction.

Lastly, we combine all these ψ_d together into one ordering function ψ, where the order can be arbitrary. This ensures the validity of ψ_d, which would need to be argued in more detail.

Combining the pieces: As we argued that satisfiability and foundedness (by showing that an ordering ψ exists and that for every $p \in M$ there exists a rule $r \in \Pi$ s.t. r is suitable for justifying p) of M is ensured, we can conclude that M is an answer set of $\mathcal{G}(\Pi)$.

"$\Longleftarrow$": Let M be an answer set of $\mathcal{G}(\Pi)$. Overall for showing "$\Longleftarrow$" we take a two step approach. The first step is to create an M' and the second is to argue why M' is an answer set.

Creating M': Then, since M is an answer set of $\Pi' = \mathcal{G}(\Pi)$ and Π is HCF, there has to exist a level mapping ψ : $\text{at}(\Pi') \to \{0, |M| - 1\}$ such that (i) M is a model of Π' and (ii) for every $a \in M$, we have that there is $r \in \Pi'$ that founds r by ψ. In more details, (ii) implies that $a \in H_r$, $B_r^+ \subseteq M$, $M \cap B_r^- = \emptyset$, $M \cap (H_r \setminus \{a\}) = \emptyset$, and $\psi(b) < \psi(a)$ for every $b \in B_r^+$. From this, we therefore define an $A = \{a_r \mid (a, r', r) \in F\}$, where F will indicate the atom-rule-groundrule pairs such that the atom is founded by the rule, defined as $F = \{(a, r', r) \mid r' \in \Pi_t, r \in \mathcal{G}(\{r'\}), h(\mathbf{D}) \in H_r, a = h(\mathbf{D}), B_r^+ \subseteq M, (M \setminus \{a\}) \cap (B_r^- \cup H_r) = \emptyset, B_r^+ = \{b | b \in B_r^+, \psi(b) < \psi(a)\}\}$. Then, we define $U = \{uf_x(\mathbf{D}, d') \mid (h(\mathbf{D}), r', r) \in F, p(\mathbf{D}') \in B_r^+ \cup B_r^- \cup H_r, 1 \leq i \leq |\mathbf{D}|, d' = (\mathbf{D}')_i, x$ is the $i-$th variable of predicate $p, x \notin H_{r'}\}$, which collects atoms regarding the grounding required to show foundedness of $h(\mathbf{D})$. Further, $R_{sat} = \{sat_r | r \in \Pi_t\}$ and $R_{uf} = \{uf_r(\mathbf{D}) | r' \in \Pi_t, r \in \mathcal{G}(\{r'\}), h(\mathbf{D}) \in H_r, \nexists (h(\mathbf{D}), r', r) \in F\}$. We define atoms over all potential variable substitutions by $V = \{sat_x(d) \mid d \in \text{dom}(x), x \in \text{var}(r), r \in \Pi_t\}$ Second to last, we capture the level mapping ψ by $L = \{(a \prec b) \mid S \in SCC(\Pi), |E(S_{\Pi_t})| \geq 1, a, b \in S, \psi(a) < \psi(b)\}$. Finally, we capture the additional helper-rule level mapping by $L_{help} = \{(h_r \prec h) \mid r \in \Pi_t, S \in SCC(\Pi), |E(S_{\Pi_t})| \geq 1, h \in H_r, h \in S\}$. From this, we define interpretation $M' = M \cup A \cup U \cup L \cup L_{help} \cup R_{sat} \cup R_{uf} \cup \{sat\}$. It remains to argue, why M' is indeed an answer set of $\mathcal{H}_{lv}(\Pi_c, \Pi_t)$.

Argument why M' is an answer set: Let $\Pi'' = \mathcal{H}_{lv}(\Pi_c, \Pi_t)$. First, observe that indeed M' is a model of Rules (2)–(12) as well as of (15)–(24). It remains to show that there does not exist an interpretation $M'' \subsetneq M'$ such that M'' is a model of $(\Pi'')^{M'}$. Suppose towards a contradiction that such an interpretation M'' exists. However, from this, we can either conclude one of the following consequences.

1. There is a rule $r \in \Pi'$ such that M is not a model of $\{r\}$. By construction of M', this contradicts that M is an answer set of Π.
2. There is an atom $a \in M$ that is unfounded by every rule $r \in \Pi'$ and despite any ψ. In this case, also by construction of M', M can not be an answer set of Π.
3. There is an atom $a \in M$ that is founded by a rule $r \in \Pi'$ and ψ; but there is a (shared) SCC $S \in \text{scc}(\Pi')$, i.e., $|E(S_{\Pi_t})| \geq 1$, such that there is an atom in S that is not founded, but required for foundedness of a. However, by construction, i.e., in particular Rules (16), (21) and (20) can not be founded due to an unsupported cycle.

It remains to argue, why despite Rules (22), every answer set M of $\mathcal{H}(\Pi_{t_1}, \Pi_{t_2})$ has a unique answer set $M \cap \text{at}(\Pi)$ of Π. This works analogously to the proof sketch of Lemma 1 in the main matter. □

C Further Details on Rewriting Techniques

This section extends Section 4 by generalizing $\mathcal{RS}^{\text{count}}$ to multiple elements. Before we do that, we provide a quick overview of our rewriting techniques, regarding grounding times obtained by hybrid grounding, which is followed by a general observation and Lemma 3 on applying rewriting procedures f.

Arity and Grounding Time Summary

We give a short summary of our results on the maximum arity, number of introduced rules and on the grounding times of our rewriting techniques presented in the paper. A compact overview is depicted in Table 2.

We provide an overview of our notation used in the proofs below: As stated in the theorems and lemmas, let Π be an HCF program and f be a rewriting technique. Further, let r be a rule with an aggregate $aggr\{E\} \prec u$ (and possibly some assumptions), where the aggregate has α many elements, ϕ_r is the maximum arity of a rule r *before rewriting*, $|h|$ is the length of the maximum head length of an element, $|d|$ is the length of the introduced variable dependencies for all tuple-predicates, and $|h'|$ and $|d'|$ are the respective lengths of the maximum combined ($|h'|+|d'|$) length of a tuple predicate for an element. A rule r or program Π is rewritten via procedure f, thereby obtaining program $f(r)$ or $f(\Pi)$, respectively. In general, we denote by ϕ_j an upper bound on the arity of an entity j. Throughout our proofs, we use ϕ_r, $\phi_{f(r)}$, ϕ_{Π_t} which denote an upper bound of an arity on a rule r, on a rewriting procedure applied to a rule $f(r)$, and on a tight program Π_t respectively. As a final note, we define $\mathcal{G}$ in Equation (34), which is superexponential.

Given $f(\Pi)$, we then apply hybrid grounding on a tight subset $\Pi_t \subseteq f(\Pi)$, s.t. $\mathcal{H}(\emptyset, \Pi_t)$, where an upper bound on the grounding time is known, which is polynomial for fixed arity. Indeed, Proposition 1 states the grounding time of hybrid grounding for a tight program Π_t in accordance to its maximum arity ϕ_{Π_t}, domain size $|\text{dom}(\Pi)|$ and cardinality ($|\Pi_t| + |\text{at}(\Pi_t)|$), with $\mathcal{O}((|\Pi_t| + |\text{at}(\Pi_t)|) \cdot |\text{dom}(\Pi)|^{2\cdot\phi_\Pi})$.

The $\mathcal{RA}$ procedure is applicable for all aggregates, all relations $\prec$ and all terms u. It produces as a result an upper bound of arity of $max\{|h'|+|d'|, \phi_{\Pi_t}\}$ by introducing α many rules, but note that we require one rule of $\mathcal{RA}$ to be grounded by traditional means. Our first fully stated rewriting technique $\mathcal{RS}^{\text{count}}$ has a maximum arity of $max\{|h|+|d|, 2\cdot|h|, \phi_{\Pi_t}\}$ by introducing α many rules of quadratic size. A different technique $\mathcal{RS}_{+}^{\text{count}}$ has a maximum arity of $max\{|d|, 2\cdot|h|, \phi_{\Pi_t}\}$, but at the cost of more than exponential number of rules ($\mathcal{G}$) of quadratic size. We use the $\mathcal{RM}^{\text{count}}$ procedure when we assume element singletons and monotone aggregates, thereby getting a maximum arity of $max\{2\cdot|h|, \phi_{\Pi_t}\}$ by introducing a constant number of rules of quadratic size.

For $\mathcal{RS}^{\text{max}}$ and $\mathcal{RS}^{\text{min}}$ we derive the same result of a maximum arity of $max\{|h| + |d|, \phi_{\Pi_t}\}$, by introducing α many rules. In the single element case ($\mathcal{RM}^{\text{max}}$) we derive a maximum arity of ϕ_r with a constant number of rules. Additionally, note that $\mathcal{RM}^{\text{max}}$ can be edited to work for monotone min aggregates as well.

For the sum aggregate with multiple elements and fixed u, we get a maximum arity of $max\{|h| + |d|, 2\cdot|h|, u, \phi_{\Pi_t}\}$ by introducing $\alpha + u$ many rules of quadratic size.

C.1 Workhorse Lemmas

We start our discussion about the additional details on rewriting techniques with an observation and a lemma, which are used throughout the proofs of this section. Provided one of our procedures, these yield an upper bound on the arity of the programs obtained by one of our procedures, which is then used to show upper bounds on grounding times, when facilitating hybrid grounding.

Observation 1 (Maximum arity of $f(\Pi)$)**.** *Let* Π *be a program with maximum arity* ϕ_Π*, where there exists a rule* $r \in \Pi$*, s.t.* r *contains a single aggregate.*

Further, let f *be an arbitrary rewriting procedure that rewrites* r *into a set of rules* $f(r)$*, where the maximum arity* $\phi_{f(r)}$ *of* $f(r)$ *is known.*

By replacing r *with* $f(r)$ *we generate the program* Π'*. Then the maximum arity of* Π' *is at most* $\phi_{\Pi'} := max\{\phi_{f(r)}, \phi_\Pi\}$*.*

Proof (Sketch). By assumption the maximum arity of Π is ϕ_Π and for $f(r)$ is $\phi_{f(r)}$.

We know that by applying $f(\Pi)$, the part $\Pi \setminus r$ stays unchanged, where therefore ϕ_Π is an upper bound on the arity. The rest of the program comprises of $f(r)$ with an upper bound on the arity of ϕ_r.

By combining the two program parts we get that an upper bound on the arity of Π' is the maximum of one of either part, but as we cannot be certain whether $\phi_{f(r)}$ or ϕ_Π is larger, we know that the maximum arity of Π' has to be at most $\phi_{f(r)}$ or ϕ_Π, respectively. This directly leads to $\phi_{\Pi'} := max\{\phi_{f(r)}, \phi_\Pi\}$. □

We now state the grounding time for a rewritten program.

Lemma 3 (Grounding time $f(\Pi)$)**.** *Let* Π *be an HCF program with maximum arity* ϕ_Π*, where there exists a rule* $r \in \Pi$*, s.t.* r *contains a single aggregate.*

Further, let f *be a rewriting procedure and* $\Pi_t \subseteq f(\Pi)$*. Then by applying hybrid grounding,* Π_t *can be grounded by* $\mathcal{H}(\emptyset, \Pi_t)$ *in* $O((|\Pi_t| + |at(\Pi_t)| + |f(r)|) \cdot |\text{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$*.*

Proof (Sketch). We want to apply the observation of the maximum arity of a rewriting procedure (Observation 1) on the proposition of the grounding time of hybrid grounding (Proposition 1).

Applying f to Π yields Π' ($\Pi' := f(\Pi)$) and by definition, Π_t is a tight subset of $f(\Pi)$ ($\Pi_t \subseteq f(\Pi)$), which is therefore applicable for hybrid grounding. By Observation 1 the arity of Π' is at most $\phi_{\Pi'} := max\{\phi_{f(r)}, \phi_\Pi\}$. As we do not need to consider predicate arities that are outside of Π_t, we can restrict ourselves to ϕ_{Π_t}, where $max\{\phi_{f(r)}, \phi_{\Pi_t}\}$ is a useful upper bound on the arity to compare rewriting procedures. Additionally, f produces $|f(r)|$ many rules. Then $|\Pi| + |at(\Pi)| + |f(r)| = |\Pi'| + |at(\Pi')|$ is the program size of Π' and therefore $|\Pi_t| + |at(\Pi_t)| + |f(r)|$ is an upper bound on the program size of Π_t. Further, let $|\text{dom}(\Pi)|$ be an upper bound on the domain *size* of program Π_t, as

Technique	#rules $f(r)$	Upper Bound on the Arity $\phi_{f(r)}$	Hybrid Grounding Time of $\mathcal{H}(\emptyset, \Pi_t)$	Lemmas, Corollaries and Theorems
$\mathcal{RS}^{\text{count}}$	α	$max\{\lvert h\rvert + \lvert d\rvert, 2 \cdot \lvert h\rvert, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert + \alpha) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 4, and 5
$\mathcal{RS}^{\text{max}}$	α	$max\{\lvert h\rvert + \lvert d\rvert, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert + \alpha) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 6, and 7
$\mathcal{RS}^{\text{min}}$	α	$max\{\lvert h\rvert + \lvert d\rvert, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert + \alpha) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 8, and 9
$\mathcal{RS}^{\text{sum}}$	$\alpha + u$	$max\{\lvert h\rvert + \lvert d\rvert, 2 \cdot \lvert h\rvert, u, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert + \alpha + u) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 10, and 11
$\mathcal{RS}_{*}^{\text{count}}$	$\mathcal{G} + \alpha$	$max\{\lvert d\rvert, \lvert h'\rvert + \lvert d'\rvert, 2 \cdot \lvert h\rvert, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert + \mathcal{G} + \alpha) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Cor. 1, and 2
$\mathcal{RS}_{+}^{\text{count}}$	$\mathcal{G}$	$max\{\lvert d\rvert, 2 \cdot \lvert h\rvert, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert + \mathcal{G}) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 12, and 13
$\mathcal{RS}_{+}^{\text{max}}$	α	$max\{\lvert d\rvert, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert + \alpha) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 14, and 15
$\mathcal{RM}^{\text{count}}$	1	$max\{2 \cdot \lvert h\rvert, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 2, and 16
$\mathcal{RM}^{\text{max}}$	1	ϕ_r	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 17, and 18
$\mathcal{RA}$	α	$max\{\lvert h'\rvert + \lvert d'\rvert, \phi_r\}$	$\mathcal{O}((\lvert\Pi_t\rvert + \lvert\text{at}(\Pi_t)\rvert + \alpha) \cdot \lvert\text{dom}(\Pi)\rvert^{2\cdot\phi_{\Pi_t}})$	Lem. 19, and 20

Table 2: The table depicts a summary of the number of rules introduced, upper bounds on the arities and hybrid grounding times of the rewriting techniques. For any rewriting procedure f, when applied to a rule r with an aggregate $aggr\{E\} \prec u$, where we assume u to be a term, that the aggregate has α many elements, and where r has maximum arity ϕ_r, we generate *#rules* many rules, with maximum arity $\phi_{f(r)}$. $\mathcal{G}$ is defined in Equation (34). $|h|$ is the length of the maximum head length of an element, $|d|$ is the length of the introduced variables dependencies for all tuple-predicates, and $|h'|$ and $|d'|$ are the respective lengths of the maximum combined ($|h'| + |d'|$) length of a tuple predicate for an element (for $\mathcal{RS}_{*}^{\text{count}}$ and $\mathcal{RA}$ techniques). Given a rewritten program $f(\Pi)$, we apply hybrid grounding on a tight subset $\Pi_t \subseteq f(\Pi)$, so $\mathcal{H}(\emptyset, \Pi_t)$, where an upper bound on the grounding time is known, which is a polynomial for a fixed arity (Proposition 1). Note that ϕ_{Π_t} is the maximum arity of Π_t, which (for practical settings) includes parts of $\phi_{f(r)}$, which is why $\phi_{f(r)}$ is highly important on practical runtime.

$|\text{dom}(\Pi_t)| \leq |\text{dom}(\Pi)|$. We ground Π_t by hybrid grounding, $\mathcal{H}(\emptyset, \Pi_t)$.

Therefore, we can now apply Proposition 1 and get: $O((|\Pi_t| + |at(\Pi_t)| + |f(r)|) \cdot |\text{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$, note the following useful upper bound on the grounding time for comparing rewriting techniques: $O((|\Pi_t| + |at(\Pi_t)| + |f(r)|) \cdot |\text{dom}(\Pi)|^{2\cdot max\{\phi_{f(r)}, \phi_{\Pi_t}\}})$. □

C.2 $\mathcal{RS}$ Revisited

In this section we generalize $\mathcal{RS}$ to (i) multiple elements, (ii) variable dependencies, (iii) variable bounds, (iv) different comparison operators, and (v) different aggregate types.

Details for the single element case

Before doing that, we revisit the single-element case, as we omitted some crucial details for the sake of readability. To keep the discussion coherent, we stick to the $count_{\prec}$ predicate, as introduced in Rules (14). Let r be a non-ground rule containing a count aggregate with one element and no global variable dependencies, as depicted below.

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), .., \neg p_n(\boldsymbol{X}_n), \text{count}\{\boldsymbol{T} : l_1(\boldsymbol{Y}_1), \ldots, l_k(\boldsymbol{Y}_k)\} \geq u \quad (25)$$

We rewrite Rule (25), where we replace the aggregate by a fresh atom $count_{\prec}$, with the intended meaning that $count_{\prec}$ holds whenever the aggregate holds, see Rule (26a). We encode element tuples by variable vectors $\boldsymbol{T}$ and a fresh *tp* predicate. Rule (26c) therefore derives $tp(\boldsymbol{T})$ whenever the element body $l_1(\boldsymbol{Y}_1), \ldots, l_k(\boldsymbol{Y}_k)$ holds. Then, for a count-aggregate to hold, at least u *element tuples* of the aggregate element have to exist. Therefore, Rule (26b) contains u many *tp* predicate occurrences, where each tuple is indexed $\boldsymbol{T}_1$ to $\boldsymbol{T}_u$. These tuples need to differ, using some *alldiff* predicate.

Note that *alldiff* is not provided by ASP, which we therefore encode in-place as follows. Intuitively, one has to compare $\frac{1}{2} \cdot u \cdot (u-1)$ many tuple combinations and every single one has to differ. This could be encoded by comparing vectors via $\neq$, which is not supported. Instead, we use the built-in *exclusive-or* ($\oplus$) and *or* ($\vee$) functions. We then combine both, to encode a single comparison ($\boldsymbol{T}_1 \neq \boldsymbol{T}_2$) by specifying that each index of the variable vector (denoted by the superscript) has to differ: $0 \neq (\boldsymbol{T}_1^1 \oplus \boldsymbol{T}_2^1) \vee ... \vee (\boldsymbol{T}_1^o \oplus \boldsymbol{T}_2^o)$. But note, that for the sake of readability, we keep in Rule (26b) the *alldiff* formulation, although the details differ.

Rewriting Procedure $\mathcal{RS}_{\text{se,nv}}^{\text{count}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), count_{\prec}. \quad (26a)$$

$$count_{\prec} \leftarrow tp(\boldsymbol{T}_1), ..., tp(\boldsymbol{T}_u), alldiff(\boldsymbol{T}_1, ..., \boldsymbol{T}_u). \quad (26b)$$

$$tp(\boldsymbol{T}) \leftarrow l_1(\boldsymbol{Y}_1), ..., l_k(\boldsymbol{Y}_k). \quad (26c)$$

Example 5.

Consider the following rule r:

$\leftarrow a(X,Y), count\{A, B : a(A,B)\} \geq 3.$

The result of $\mathcal{RS}_{se,nv}^{count}(r)$ *is:* $\leftarrow a(X,Y), count_{\prec}.$

$tp(A,B) \leftarrow a(A,B).count_{\prec} \leftarrow tp(A_1,B_1), tp(A_2,B_2), tp(A_3,B_3), alldiff((A_1,B_1),(A_2,B_2),(A_3,B_3))\,.$

Generalization (ii), variable dependencies

In order for our approach to consider variable dependencies, we need to ensure that whenever a global variable occurs inside an aggregate element, its (local) variable value coincides with the global value. We achieve this by handing down dependent variables **V** through the predicates of our rewriting. In the following, we assume known variable dependencies **V**, which can be inferred from a preprocessor.

We add these variable dependencies $\boldsymbol{V}$ to $count_{\prec}$ and *tp*, resulting in predicates $count_{\prec}(\mathbf{V})$ and $tp(\mathbf{T}, \mathbf{V})$, respectively. This results in an adapted rewriting procedure $\mathcal{RS}_{\text{se}}^{\text{count}}$, where we assume a single-element aggregate. Notably, for Rule (26c), we need to add the positive body of the rule, in order to ensure variable safeness, resulting in Rule (27c). Variable safeness for Rule (26b) is already ensured, therefore $\boldsymbol{V}$ can simply be added, resulting in Rule (27b). And finally Rule (26a) is updated to Rule (27a).

Rewriting Procedure $\mathcal{RS}_{\text{se}}^{\text{count}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), count_{\prec}(\boldsymbol{V}). \tag{27a}$$

$$count_{\prec}(\boldsymbol{V}) \leftarrow tp(\boldsymbol{T}_1, \boldsymbol{V}), ..., tp(\boldsymbol{T}_u, \boldsymbol{V}), alldiff(\boldsymbol{T}_1, ..., \boldsymbol{T}_u). \tag{27b}$$

$$tp(\boldsymbol{T}, \boldsymbol{V}) \leftarrow l_1(\boldsymbol{Y}_1), ..., l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m). \tag{27c}$$

Example 6.

Consider the following rule r:

$$\leftarrow a(X, Y), count\{A, B : a(A, B), b(A, X)\} \geq 3.$$

The result of $\mathcal{RS}_{se}^{count}(r)$ *is:* $\leftarrow a(X, Y), count_{\prec}(X).$
$tp(A, B, X) \leftarrow a(A, B), b(A, X), a(X, Y).$
$count_{\prec}(X) \leftarrow tp(A_1, B_1, X), tp(A_2, B_2, X), tp(A_3, B_3, X), alldiff((A_1, B_1), (A_2, B_2), (A_3, B_3)).$

Generalizations (i) and (ii)

Note that the extension discussed here is the same as the generalization (i) of Subsection 4.2, just with added variable dependencies. In contrast to Subsection 4.2 we provide the reader here with more details, especially regarding the construction of *dummy constants* (which we call here *skolem-constants*).

In the next step, we extend our construction of $\mathcal{RS}_{\text{se}}^{\text{count}}$ to enable α many elements, i.e., $e_1, \ldots, e_\alpha$. Thereby the main difference is that (i) tuples may have potentially different lengths and (ii) these tuples can be generated by multiple elements. These length differences need to be considered, as our approach uses *tp* predicates of fixed arity, i.e., we can only compare same-length tuples via *alldiff*. To accommodate Issue (i) in our rewriting techniques, we use a preprocessing procedure *h*, as depicted in Equation (28). For this procedure, we assume that the maximum lengths of all element heads ($max_{|\boldsymbol{T}|}$) are known and that we introduced up to $max_{|\boldsymbol{T}|}$ constants $skolems := \{c_1, ..., c_{max_{|\boldsymbol{T}|}}\}$ accordingly. The *h* procedure is then called for each element *e* of head length $|\boldsymbol{T}|$=1, which fills the gap between $|\boldsymbol{T}|$ and $max_{|\boldsymbol{T}|}$ with the corresponding constants.

$$\boldsymbol{T}' := h(\boldsymbol{T}, skolems) \tag{28}$$

Example 7. *Let for example* $max_{|\boldsymbol{T}|}$=3*, i.e., skolems* $= \{c_1, c_2, c_3\}$*, and assume that* $|\boldsymbol{T}| = 1$*, where* $\boldsymbol{T} = \langle X \rangle$*. Then,* $\boldsymbol{T}' = h(\boldsymbol{T}, skolems) = \langle X, c_2, c_3 \rangle$*.*

Then, due to (ii), variable dependencies might differ between elements, which raises a similar issue as above, since *tp* predicates have fixed arity. To tackle this, we include all variable dependencies for all elements in each *tp* predicate. Thereby, we compute dependencies by the preprocessing procedure *d*, as given in Equation (29). This procedure takes all elements and global variables as input and it returns a variable vector of variable dependencies for all predicates.

$$\boldsymbol{V}' := d(e_1, ..., e_\alpha, glob\_vars) \tag{29}$$

Since it could be that a global variable does not occur inside an element, we have to include the positive body $p_2(\boldsymbol{X_2}), ..., p_m(\boldsymbol{X}_m)$ of the rewritten rule into *tp*, see Rules (30c).

The resulting rewriting procedure $\mathcal{RS}^{\text{count}}$ with variable dependencies is depicted in Rules (30). Note that we do not include extra indices for differing elements in Rules (30b) and (30c).

Rewriting Procedure $\mathcal{RS}^{\text{count}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), count_{\prec}(\boldsymbol{V}'). \tag{30a}$$

$$count_{\prec}(\boldsymbol{V}') \leftarrow tp(\boldsymbol{T}'_1, \boldsymbol{V}'), ..., tp(\boldsymbol{T}'_u, \boldsymbol{V}'), alldiff(\boldsymbol{T}'_1, ..., \boldsymbol{T}'_u). \tag{30b}$$

For every element:

$$tp(\boldsymbol{T}', \boldsymbol{V}') \leftarrow l_1(\boldsymbol{Y}_1), \ldots, l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m). \tag{30c}$$

Utilizing Hybrid Grounding. In order to efficiently apply hybrid grounding in practice for $\mathcal{RS}^{\text{count}}$, Π_t above comprises Rules (30c) and (30b), such that Π_c contains Rules (30a) and potential remaining rules of Π.

On the theoretical side, our goal is to apply Theorem 1 on a tight program Π_t and the remainder $\Pi_c = \Pi \setminus \Pi_t$. Since non-ground answer set existence is NEXPTIME-complete, we cannot make polynomial grounding time guarantees on Π_c. However, by using Proposition 1, we obtain in the following a runtime result on Π_t.

Results. Before we discuss grounding times of $\mathcal{RS}^{\text{count}}$, we require upper bounds on obtained predicate arities. To this end, we use the following notation. Let ϕ_r denote the maximum arity of any predicate in *r*, $|h| = |\boldsymbol{T}'| = max_{|\boldsymbol{T}|}$ be the maximum element head length, and $|d| = |\boldsymbol{V}'|$ be the length of the variable vector of the output of the *d*-preprocessor.

Lemma 4 (Maximum arity of $\mathcal{RS}^{\text{count}}$)**.** *Let r be a rule containing a single count aggregate with u being an integer constant and* $\prec$ *being* $\geq$*. Then, the maximum arity of* $\mathcal{RS}^{count}(r)$ *is bounded by* $max\{|h| + |d|, 2 \cdot |h|, \phi_r\}$*.*

Proof (Sketch). As the *tp* predicate combines *h* and *d*, the arity of *tp* is $|h| + |d|$. Consequently, the maximum arity of Rules (30c) are $max\{|h| + |d|, \phi_r\}$. Then, *alldiff* comprises an exclusive or ($\oplus$) and or ($\vee$), e.g., $0 \neq (\boldsymbol{T}_1^1 \oplus \boldsymbol{T}_2^1) \vee ... \vee (\boldsymbol{T}_1^m \oplus \boldsymbol{T}_2^m)$. These comparisons have arity $2 \cdot |h|$. Since in Rule (30b) only *alldiff* and the *tp*-predicates occur, an upper bound on its arity is $max\{|h| + |d|, 2 \cdot |h|\}$. The arity of $count_{\prec}$ is $|d|$. Therefore, an upper bound on the arity of Rules (30a) is $max\{\phi_r, |d|\}$. Combining these results (note $|d| \leq |h| + |d|$), the arity of Π' is bounded by $max\{|h| + |d|, 2 \cdot |h|, \phi_r\}$. □

By application of Observation 1 the result immediately carries over from a single rule to programs. Consequently, we obtain the following lemma.

Lemma 5 (Grounding time of $\mathcal{RS}^{\text{count}}$)**.** *Let Π be an HCF program with a rule using a count aggregate over α elements with integer u, and $\prec=\leq$. Further, let $\Pi_t \subseteq \mathcal{RS}^{count}(\Pi)$ and ϕ_{Π_t} be the largest predicate arity of Π_t. Then, the runtime of $\mathcal{H}(\emptyset, \Pi_t)$ is in $\mathcal{O}((|\Pi_t|+|\operatorname{at}(\Pi_t)|+\alpha) \cdot |\operatorname{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$.*

Proof (Sketch). For the proof, it is enough to combine the results obtained so far. In principle, we apply the rewriting procedure $\mathcal{RS}^{\text{count}}$ on our results about the grounding time of a program where a rewriting procedure was applied (Lemma 3).

In more detail, as we use hybrid grounding for $\mathcal{H}(\emptyset, \Pi_t)$, we want to apply the results of the arity of $\mathcal{RS}^{\text{count}}$ (Lemma 4) to the lemma about the grounding time of rewriting procedures for hybrid grounding (Lemma 3), which uses the results of grounding time for hybrid grounding (Proposition 1). We know that $\mathcal{RS}^{\text{count}}$ produces $O(\alpha)$ many rules, and by the result of Lemma 4 that the maximum arity of $\mathcal{RS}^{\text{count}}(\Pi)$ is bounded by $max\{|h| + |d|, 2 \cdot |h|, \phi_\Pi\}$. With this stated, we can now apply Lemma 3, from where we derive $\mathcal{O}((|\Pi_t|+|\operatorname{at}(\Pi_t)|+\alpha) \cdot |\operatorname{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$. □

The following example shows the use of $\mathcal{RS}^{\text{count}}$ together with calculating the maximum resulting arity of the rule r.

Example 8. *The following example is an extension of the previous examples, where we show the application of $\mathcal{RS}^{count}$ to the rule r, resulting in a set R' of rules $R' = \mathcal{RS}^{count}(r)$. Note that the maximum arity of r is $\phi_r = 2$.*

$$\leftarrow a(X,Y), count\{A,B : a(A,B), b(A,X); A : p(A), a(A,Y); A : q(A)\} \geq 3.$$

When rewriting r to $\mathcal{RS}^{count}(r)$, the rule is rewritten to the set of rules R' shown below. Note the introduced skolem constant c_2 for the second arity position the tp predicates. Further note that the maximum arity of the set of rules R' is $\phi_{R'} := max\{h| + |d|, 2 \cdot |h|, \phi_r\}$ and as $|h| = 2$, $|d| = 2$, $\phi_r = 2$, $|h| + |d| = 4$, $2 \cdot |h| = 4$, therefore $\phi_{R'} = 4$.

$\leftarrow a(X,Y), count_{\prec}(X,Y).\quad tp(A,B,X,Y) \leftarrow a(A,B), b(A,X), a(X,Y).\ tp(A,c_2,X,Y) \leftarrow p(A), a(A,X), a(X,Y).$
$tp(A,c_2,X,Y) \leftarrow q(A), a(X,Y). count_{\prec}(X,Y) \leftarrow tp(A_1,B_1,X,Y), tp(A_2,B_2,X,Y), tp(A_3,B_3,X,Y),$
$alldiff((A_1,B_1),(A_2,B_2),(A_3,B_3)).$

$\mathcal{RS}^{\textbf{max}}$ In this step we take a first look at generalizations in the domain of aggregates (v), namely the generalization of $\mathcal{RS}^{\text{count}}$ to *max* aggregates. Note that we proceed with analogous assumptions as for $\mathcal{RS}^{\text{count}}$, regarding $\geq u$. We use notation as above and use the $max_{\prec}(\boldsymbol{V}')$ predicate instead of $count_{\prec}(\boldsymbol{V}')$, which requires adapting the Rule (30b). We define $tp(\boldsymbol{T}', \boldsymbol{V}')$ and state that whenever the first variable of $\boldsymbol{T}'$ (denoted as $\boldsymbol{T}'^1$) has a value greater or equal than u, then $max_{\prec}(\boldsymbol{V}')$ holds. Rules (31) show the adapted part of the resulting rewriting procedure $\mathcal{RS}^{\text{max}}$.

Rewriting Procedure $\mathcal{RS}^{\text{max}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), max_{\prec}(\boldsymbol{V}'). \tag{31a}$$

$$max_{\prec}(\boldsymbol{V}') \leftarrow tp(\boldsymbol{T}', \boldsymbol{V}'), \boldsymbol{T}'^1 \geq u. \tag{31b}$$

For every element: (31c)
$$tp(\boldsymbol{T}', \boldsymbol{V}') \leftarrow l_1(\boldsymbol{Y}_1), \ldots, l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m).$$

Results. We first state the maximum arity of $\mathcal{RS}^{\text{max}}$ and then state the (improved) hybrid grounding time.

Lemma 6 (Maximum arity of $\mathcal{RS}^{\text{max}}$)**.** *Let r be a rule containing a single max aggregate with u being an integer constant and $\prec$ being $\geq$. Then, the maximum arity of $\mathcal{RS}^{max}(r)$ is bounded by $max\{|h| + |d|, \phi_r\}$.*

Proof (Sketch). The results for predicate arity *tp* (bound by $|h| + |d|$) carry over from Lemma 4, together with the upper bound on the arity of the Rules (30a) (bound by $max\{|d|, \phi_r\}$) and (30c) (bound by $max\{|h| + |d|, \phi_r\}$). Rule (31b) comprises *tp*, one $max_{\prec}(\boldsymbol{V}')$ (bound by $|d|$) predicate and a comparison with one variable. The combined upper bound on the arity is, therefore, $|h| + |d|$. Combining these results, an upper bound to the arity of $\mathcal{RS}^{\text{max}}(r)$ as $max\{|h| + |d|, \phi_r\}$. □

By Observation 1 an upper bound on the maximum arity of $\mathcal{RS}^{\text{max}}(\Pi)$ directly follows. Note the difference in $\phi_{\Pi'}$ between $\mathcal{RS}^{\text{count}}$ and $\mathcal{RS}^{\text{max}}$, where for $\mathcal{RS}^{\text{count}}$ the maximum arity is $\phi_{\Pi', \mathcal{RS}^{\text{count}}} := max\{|h| + |d|, 2 \cdot |h|, \phi_\Pi\}$ and for $\mathcal{RS}^{\text{max}}$ is $\phi_{\Pi', \mathcal{RS}^{\text{max}}} := max\{|h| + |d|, \phi_\Pi\}$. Compared to Lemma 5, with this optimized rewriting for max aggregates, we can improve the hybrid grounding time.

Lemma 7 (Grounding time of $\mathcal{RS}^{\text{max}}$)**.** *We take the same assumptions as for the $\mathcal{RS}^{count}$ case (Lemma 5), with the sole difference being that the respective aggregate is a max aggregate. Then the respective hybrid grounding time is in $O((|\Pi_t| + |at(\Pi_t)| + \alpha) \cdot |\operatorname{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$.*

Proof (Sketch). As in the proof of Lemma 5, we combine previously stated results. Meaning we want to insert the result of the upper bound on the arity of $\mathcal{RS}^{\text{max}}$ (Lemma 6) into the result about the grounding time by applying hybrid grounding, as stated in Lemma 3.

As $\mathcal{RS}^{\text{max}}$ produces $O(\alpha)$ many rules, we derive our result of $O((|\Pi_t| + |at(\Pi_t)| + \alpha) \cdot |\operatorname{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$. □

Example 9. *We show the max aggregate by the same example as for the count aggregate, where we just switch out the count to the max aggregate. Therefore, consider the rule r below. We assume $r \in \Pi$, s.t. Π has maximum arity $\phi_\Pi = 2$.*

$$\leftarrow a(X,Y), max\{A,B : a(A,B), b(A,X); A : p(A), a(A,Y); A : q(A)\} \geq 3.$$

When applying the rewriting procedure $\mathcal{RS}^{max}$ as depicted in Rules (31), we receive the rewritten program below. Note the skolem constant c and that the maximum arity of the program is $\phi_{\Pi'} := max\{\phi_\Pi, |h| + |d|\}$ and as $|h| + |d| = 4$, therefore $\phi_{\Pi'} = 4$.

$\leftarrow a(X,Y), max\_left(X,Y).\quad max\_left(X,Y) \leftarrow tp(A,B,X,Y), A >= 3. tp(A,B,X,Y) \leftarrow a(A,B), b(A,X), a(X,Y). tp(A,c,X,Y) \leftarrow q(A), a(X,Y).$
$tp(A,c,X,Y) \leftarrow p(A), a(A,X), a(X,Y).$

$\mathcal{RS}^{\mathbf{min}}$ The next step towards generalizing (v) is shown below for the *min* aggregate, which is the $\mathcal{RS}^{\text{min}}$ procedure as shown in the Rules (32). It is very similar to $\mathcal{RS}^{\text{max}}$. Crucially, we still have the assumption of $\geq u$. Therefore, *min* behaves as an anti-monotonic aggregate.

Intuitively for $\mathcal{RS}^{\text{min}}$, we want to derive, that if we are able to derive any value $< u$, we know that the aggregate does not hold. Therefore, we define $\mathcal{RS}^{\text{min}}$ similarly to $\mathcal{RS}^{\text{max}}$, and the the definitions of $\boldsymbol{V'}$, $\boldsymbol{T'}$, carry over. However, the comparison operator in Rule (32a) is reversed, when compared to $\mathcal{RS}^{\text{max}}$. So the semantics of $min_{\prec}$ is now intuitively: Are we able to derive a value that is strictly smaller than u. Due to our assumption that the aggregate checks whether $\geq u$, whenever we cannot derive $min_{\prec}$, we know that the aggregate holds. This implies that a default negation is required in Rule (32a), i.e., $\neg min_{\prec}$.

Rewriting Procedure $\mathcal{RS}^{\text{min}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), \neg min_{\prec}(\boldsymbol{V'}). \tag{32a}$$

$$min_{\prec}(\boldsymbol{V'}) \leftarrow tp(\boldsymbol{T'}, \boldsymbol{V'}), \boldsymbol{T'}^1 < u. \tag{32b}$$

For every element: (32c)

$$tp(\boldsymbol{T'}, \boldsymbol{V'}) \leftarrow l_1(\boldsymbol{Y}_1), \ldots, l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m).$$

Lemma 8 (Maximum arity of $\mathcal{RS}^{\text{min}}$)**.** *Let r be a rule containing a single min aggregate with u being an integer constant and $\prec$ being $\geq$. Then, the maximum arity of $\mathcal{RS}^{min}(r)$ is bounded by $max\{|h| + |d|, \phi_r\}$.*

Proof (Sketch). As all rules produced by Rules (32c) of the $\mathcal{RS}^{\text{min}}$ and of the respective rule of the $\mathcal{RS}^{\text{max}}$ rewriting are the same, we know by Lemma 6 that the maximum arity of these rules is $max\{|h| + |d|, \phi_r\}$.

The result of the Rule (32b) has three predicates, $min_{\prec}$ with arity $|d|$, *tp* with arity $|h| + |d|$ and $\boldsymbol{T'}^1 < u$ with arity 1. Therefore $|h| + |d|$ is an upper bound on the arity.

The outcome produced by Rule (32a) can be split into two parts, where the first part contains all the rules from rule r, where ϕ_r is an upper bound. The second part is $\neg min_{\prec}(\boldsymbol{V'})$, where $|d|$ is an upper bound.

Combining all of the above we get an upper bound for $\mathcal{RS}^{\text{min}}(r)$ of $max\{|h| + |d|, \phi_r\}$. ☐

Lemma 9 (Grounding time of $\mathcal{RS}^{\text{min}}$)**.** *We take the same assumptions as for the $\mathcal{RS}^{max}$ case (Lemma 7), with the sole exception of the aggregate being the min aggregate. By applying our results about hybrid grounding we get an upper bound of $O((|\Pi_t| + |at(\Pi_t)| + \alpha) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$.*

Proof (Sketch). We combine our results from above. We know that $\mathcal{RS}^{\text{min}}$ produces $O(\alpha)$ many rules. Therefore we apply Lemma 3 and receive our upper bound of $O((|\Pi_t| + |at(\Pi_t)| + \alpha + u) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$. ☐

$\mathcal{RS}^{\mathbf{sum}}$ **for Multiple Elements.** In the following, we generalize the count $\mathcal{RS}^{\text{count}}$ technique to accommodate the sum aggregate. For the sum aggregate, every tuple has a *weight*, which corresponds to the first element of the tuple, and which is 1 for every tuple in the count aggregate. Therefore, it could be that a single tuple exists with weight $\geq u$, or two tuples with a combined weight, etc.

We accommodate this by including all combinations of possible tuples, which are exactly u many, thereby starting at one tuple up to u many tuples (where u many corresponds to the count case):

Therefore our encoding produces α many rules in Rules (33c) and u many rules in Rules (33b).

Rewriting Procedure $\mathcal{RS}^{\text{sum}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), sum_{\prec}(\boldsymbol{V'}). \tag{33a}$$

For each integer $1 \leq i \leq u$:

$$sum_{\prec}(\boldsymbol{V'}) \leftarrow tp(\boldsymbol{T'_1}, \boldsymbol{V'}), ..., tp(\boldsymbol{T'_i}, \boldsymbol{V'}), alldiff(\boldsymbol{T'_1}, ..., \boldsymbol{T'_i}), u \leq \boldsymbol{T}^1_1 + ... + \boldsymbol{T}^1_i. \tag{33b}$$

For each element : (33c)

$$tp(\boldsymbol{T'}, \boldsymbol{V'}) \leftarrow l_1(\boldsymbol{Y}_1), ..., l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m).$$

Lemma 10 (Maximum arity of $\mathcal{RS}^{\text{sum}}$)**.** *Let r be a rule containing a single sum aggregate with u being an integer constant and $\prec$ being $\geq$. Then, the maximum arity of $\mathcal{RS}^{sum}(r)$ is bounded by $max\{|h| + |d|, 2 \cdot |h|, |u|, \phi_r\}$.*

Proof (Sketch). By Lemma 4 we know that the arity of $\mathcal{RS}^{\text{count}}(r)$ is bounded by $max\{|h| + |d|, 2 \cdot |h|, \phi_r\}$. All rules generated by the $\mathcal{RS}^{\text{sum}}$ in the Rules (33a) and (33c) correspond to the Rules (30a) and (30c) of the $\mathcal{RS}^{\text{count}}$ and have therefore the same upper bound on the arity.

Let us assume that $i = u$ in Rules (33b): Then this corresponds to a single rule, where it follows that this rule corresponds largely to the rule produced by Rule (30b). The only difference is the comparison $u \leq \boldsymbol{T}^1_1 + ... + \boldsymbol{T}^1_u$. This comparison has an arity of $|u|$ as $|u|$ many predicates occur. Therefore the combined arity of the rule generated by Rules (33b), when $i = u$ is $max\{|h| + |d|, 2 \cdot |h|, |u|\}$.

For every other $i < u$ it holds that the arity of *tp* and *alldiff* does not change, but the arity of the aforementioned comparison changes. In more detail, it is $|i|$ and therefore $< |u|$, so $|u|$ is an upper bound for all i.

Combining these observations we get that $max\{|h|+|d|, 2 \cdot |h|, |u|\}$ is an upper bound on all rules produced by Rules (33b).

Finally, when we combine all of the above observations we get an upper bound on the arity over all rules by $max\{|h| + |d|, 2 \cdot |h|, |u|, \phi_r\}$. ☐

Lemma 11 (Grounding time of $\mathcal{RS}^{\text{sum}}$)**.** *We take the same assumptions as for the $\mathcal{RS}^{count}$ case (Lemma 5), with the sole assumption that we assume a sum aggregate. Then by facilitating hybrid grounding we derive $O((|\Pi_t| + |at(\Pi_t)| + \alpha + u) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$.*

Proof (Sketch). We combine our results from above. We know that $\mathcal{RS}^{\text{sum}}$ produces $O(\alpha + u)$ many rules. Therefore we apply Lemma 3 and derive $O((|\Pi_t| + |at(\Pi_t)| + \alpha + u) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$. ☐

C.3 Rewriting by Combination - Differing Element Head Lengths

In contrast to the so far discussed methods, which circumvent the multiple elements problem by introducing a general element $\mathcal{E}$, the methods presented here generate all possible combinations of elements.

$\mathcal{RS}_{+}^{\textbf{count}}$ and $\mathcal{RS}_{*}^{\textbf{count}}$ for Multiple Elements. The count rewriting version presented here ($\mathcal{RS}_{+}^{\text{count}}$ and its for readability modified version $\mathcal{RS}_{*}^{\text{count}}$) is especially useful for aggregates with few elements having differing element head lengths.

For the sake of explanation, we start with the (monotonic, $\geq u$) count aggregate. This technique generates different *tp* predicates for each element, meaning that the *tp* predicates are indexed accordingly and have varying arities.

As for the $\mathcal{RS}^{\text{count}}$ technique, we want to have rule(s) similar to Rule (30b), where we check whether the $count_{\prec}$ predicate holds.Now observe that to check whether the $count_{\prec}$ predicate holds, we can do this by introducing u many tp-predicates. But crucially, as in this approach presented here, the tppredicates indicate a single element and do not represent all elements we do not know which *tp* to add. Although it is not known in general which combination of *tp* might be beneficial for asserting $count_{\prec}$, we know that whenever the aggregate holds, at least one combination of u many *tp*, from possibly differing elements, asserts it.

From these observations, we can infer that we can generate all possible combinations of *tp* predicates and assert that whenever at least one of them holds, $count_{\prec}$ also holds. Note that the number of combinations can be thought of as a *selection of a multiset* combinatorial operation, as we have u many *positions* for a *tp*, and we have α (number of elements in the aggregate) many different *tp* predicates.

Therefore we present a solution where one checks each conceivable combination c_i of *tp* predicates and assumes that each combination c_i is aware of the *tp* predicates that are used in its combination. The number of possible combinations depends on u and the number of elements α in a combinatorial way.

$$\mathcal{G} = \binom{\alpha + u - 1}{u} \tag{34}$$

A crucial observation for practice is that in a combination, we can skip checking combinations of *tp* predicates of differing lengths, as they have to differ per ASP semantics.

We depict the rewriting procedure $\mathcal{RS}_{*}^{\text{count}}$ in Rule (35). Rule(s) (35b) found the $count_{\prec}$ predicate. Note that variable dependencies are also handled in these rule(s), by using the same notion of $\boldsymbol{V}'$ as in $\mathcal{RS}^{\text{count}}$, and ensuring variable safeness by adding $B_r^+ = \{p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m)\}$ to the rule(s).

The $count_{\prec}$ aggregate can then be used as in the $\mathcal{RS}^{\text{count}}$ case in Rule (35a). And *tp* predicates are indexed in Rules (35c) according to their element.

Rewriting Procedure $\mathcal{RS}_{*}^{\text{count}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), count_{\prec}(\boldsymbol{V}'). \tag{35a}$$

For each conceivable combination c_i generate:

$$count_{\prec}(\boldsymbol{V}') \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), tp_{c_i^1}(\boldsymbol{X}_1, \boldsymbol{V}_1), ..., tp_{c_i^u}(\boldsymbol{X}_u, \boldsymbol{V}_u), \mathit{alldiff}(\boldsymbol{X}_1, ..., \boldsymbol{X}_u) \tag{35b}$$

For each element e_j :

$$tp_j(\boldsymbol{T}, \boldsymbol{V}) \leftarrow l_1(\boldsymbol{Y}_1), ..., l_k(\boldsymbol{Y}_k). \tag{35c}$$

Note that due to readability concerns, we did not pull the body of the *tp* predicates into Rules (35b). Therefore, the depicted rewriting procedure is $\mathcal{RS}_{*}^{\text{count}}$, otherwise, it would be $\mathcal{RS}_{+}^{\text{count}}$. So, if we insert Rules (35c) into Rules (35b), we obtain version $\mathcal{RS}_{+}^{\text{count}}$, which is depicted below in the Rules (36a). As above c_i is the i^{th} combination, where c_i^j denotes the j^{th} position (which corresponds to a specific *tp* predicate). By this, and the additional index about the position inside the element, we index the body predicates l.

For each conceivable combination c_i generate:

$$count_{\prec}(\boldsymbol{V}') \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), l_{c_i^1,1}(\boldsymbol{Y}_{c_i^1,1}), ..., l_{c_i^1,k}(\boldsymbol{Y}_{c_i^1,k}), ..., l_{c_i^u,1}(\boldsymbol{Y}_{c_i^u,1}), ..., l_{c_i^u,k}(\boldsymbol{Y}_{c_i^u,k}), \mathit{alldiff}(\boldsymbol{X}_1, ..., \boldsymbol{X}_u). \tag{36a}$$

Note that for the proofs we discuss $\mathcal{RS}_{+}^{\text{count}}$, which has a better bound on the maximum arity than $\mathcal{RS}_{*}^{\text{count}}$, where we just sketch corollaries.

Lemma 12 (Maximum arity of $\mathcal{RS}_{+}^{\text{count}}$)**.** *Let r be a rule containing a single count aggregate with u being an integer constant and $\prec$ being $\geq$. Then, the maximum arity of $\mathcal{RS}_{+}^{\text{count}}(r)$ is bounded by $max\{|d|, 2 \cdot |h|, \phi_r\}$.*

Proof (Sketch). Let $|d''|$ denote the maximum arity of a $count_i$ predicate. Note that $|d''| \leq |d|$, therefore $|d|$ is an upper bound of $|d''|$.

Then the rules generated by Rules (35b), can be split into three parts. The first part $count_i$ has maximum arity of $|d|$. The second part, the respective bodies of the elements, have maximum arity of ϕ_r. The third part $\mathit{alldiff}(\boldsymbol{X}_1, ..., \boldsymbol{X}_u)$ has a maximum arity of $2 \cdot |h|$. Therefore, the combined maximum arity of these rules are $max\{2 \cdot |h|, |d|, \phi_r\}$. We know that the maximum arity of Rule (35a) has a maximum arity of $max\{|d|, \phi_r\}$.

Therefore, the total combined arity of $\mathcal{RS}_{+}^{\text{count}}(r)$ is bounded by $max\{|d|, 2 \cdot |h|, \phi_r\}$. □

By Observation 1 we derive the maximum arity of a program $\mathcal{RS}_{+}^{\text{count}}(\Pi)$.

Lemma 13 (Grounding time of $\mathcal{RS}_{+}^{\text{count}}$)**.** *We take the same assumptions as for the $\mathcal{RS}^{count}$ case (Lemma 5). Then by facilitating hybrid grounding we derive $O((|\Pi_t| + |at(\Pi_t)| + \mathcal{G}) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$.*

Proof (Sketch). We combine the results from above. Thereby, $\mathcal{RS}_{+}^{\text{count}}$ produces $O(\mathcal{G})$ many rules. By facilitating Lemma 3 we get our result of $O((|\Pi_t| + |at(\Pi_t)| + \mathcal{G}) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$. □

Next, we sketch corollaries about the maximum arity of $\mathcal{RS}_{*}^{\text{count}}$ and its grounding time.

Corollary 1 (Maximum arity of $\mathcal{RS}_{*}^{\text{count}}$)**.** *Let r be a rule containing a single count aggregate with u being an integer constant and $\prec$ being $\geq$. Then, the maximum arity of $\mathcal{RS}_{*}^{count}(r)$ is bounded by $max\{|d|, |h'| + |d'|, 2 \cdot |h|, \phi_r\}$.*

Proof (Idea). Most results from Lemma 12 carry over to the $\mathcal{RS}_{*}^{\text{count}}$ case, but through the separation of Rules (35b) and (35c) we get that the maximum arity of a *tp* could have an impact, where we denote $|h'| + |d'|$ as the maximum arity of any *tp*, where $|h'|$ and $|d'|$ are the respective tuple and variable-dependencies parts. As it is not possible to make a good estimation of $|h'| + |d'|$ w.r.t. to the other parts, we have to give it into our upper bound for the maximum arity, which is therefore $max\{|d|, |h'| + |d'|, 2 \cdot |h|, \phi_r\}$. $\square$

Corollary 2 (Grounding time of $\mathcal{RS}_{*}^{\text{count}}$)**.** *We take the same assumptions as for the $\mathcal{RS}_{+}^{count}$ case (Lemma 13). Then by facilitating hybrid grounding we derive $O((|\Pi_t| + |at(\Pi_t)| + \mathcal{G} + \alpha) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$.*

Proof (Idea). Most results from Lemma 13 carry over to the $\mathcal{RS}_{*}^{\text{count}}$ case. By the above-mentioned separation of Rules (35b) and (35c) we, on the one hand, received an other maximum arity of the rewriting procedure (Corollary 1), and on the introduced α many new rules. Combining the results, we receive the following grounding time for $\mathcal{RS}_{*}^{\text{count}}$: $O((|\Pi_t| + |at(\Pi_t)| + \mathcal{G} + \alpha) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$. $\square$

Note that we acknowledge $\mathcal{O}(\mathcal{G} + \alpha)$ to be in $\mathcal{O}(\mathcal{G})$, but due to reasons of readability (highlighting the practical differences of our rewriting procedures), we include α into the program size.

$\mathcal{RS}_{+}^{\text{max}}$ for Multiple Elements. The rewriting procedure $\mathcal{RS}_{+}^{\text{max}}$ for the max aggregate, which is depicted in Rules (37), is excellently suited for aggregates with differing element head length.

For $\mathcal{RS}_{+}^{\text{max}}$ we generate for each element a *max* predicate, which, if it holds, guarantees that an element tuple exists where the first item is greater or equal then u. Note that we need to ensure variable dependencies and variable safeness, therefore, we include the positive body of rule r into the body of *max* and further use the d function as stated for $\mathcal{RS}^{\text{count}}$. This is depicted in Rules (37b).

Note that this is similar to $\mathcal{RS}^{\text{max}}$, but we pull the *tp* predicates into the Rules (37b). An alternative (not shown) encoding would be similar to $\mathcal{RS}_{+}^{\text{count}}$ with respect to the double negation, where we would generate different *max* predicates.

Rewriting Procedure $\mathcal{RS}_{+}^{\text{max}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), max_{\prec}(\boldsymbol{V}'). \tag{37a}$$

For each element :

$$max(\boldsymbol{V}') \leftarrow l_1(\boldsymbol{Y}_1), ..., l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \boldsymbol{T}^1 \geq u. \tag{37b}$$

Lemma 14 (Maximum arity of $\mathcal{RS}_{+}^{\text{max}}$)**.** *Let r be a rule containing a single max aggregate with u being an integer constant and $\prec$ being $\geq$. Then, the maximum arity of $\mathcal{RS}_{+}^{max} r$ is bounded by $max\{|d|, \phi_r\}$.*

Proof (Sketch). The rules generated by Rules (37b) can be split up into two parts. The first part is the $max(\boldsymbol{V}')$ part, which has arity $|d|$. Part two being the body of the elements and the rule r with an upper bound on the arity of the predicates of ϕ_Π. The third part is then the $\boldsymbol{T}^1 \geq u$ comparison, with arity 1. Therefore these rules have an upper bound on the arity is $max\{|d|, \phi_\Pi\}$. The rule generated by Rule (37a) has the same upper bound on the arity.

Therefore the combined upper bound on the arity is $max\{|d|, \phi_\Pi\}$. $\square$

Lemma 15 (Grounding time of $\mathcal{RS}_{+}^{\text{max}}$)**.** *We take the same assumptions as for the $\mathcal{RS}^{max}$ case (Lemma 7). Then by facilitating our results of hybrid grounding we derive an upper bound of $O((|\Pi_t| + |at(\Pi_t)| + \alpha) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$.*

Proof (Sketch). We combine our results from above. We know that $\mathcal{RS}_{+}^{\text{max}}$ produces $O(\alpha)$ many rules. Therefore by applying Lemma 3 we get $O((|\Pi_t| + |at(\Pi_t)| + \alpha) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$. $\square$

C.4 Singleton Aggregates Revisited

An important special case of aggregates is where the set of aggregate elements is a singleton.

$\mathcal{RM}^{\text{count}}$ for Singletons. For the $\mathcal{RM}^{\text{count}}$ method, as discussed in the main part of the paper, we derive the following results.

Lemma 2 (Maximum arity of $\mathcal{RM}^{\text{count}}$)**.** *Let r be a rule containing a single count aggregate, with a singleton element set, with u being an integer constant and $\prec$ being $\geq$. Then, the maximum arity of $\mathcal{RM}^{count}(r)$ is bounded by $max\{2 \cdot |h|, \phi_r\}$.*

Proof (Sketch). $\mathcal{RM}^{\text{count}}(r)$ generates a rule which can be split into two parts. The first part contains all predicates of the rule (except the aggregate) and all predicates of the element, whereas the second part contains the *alldiff* predicate.

Observe that the arity of the first part is bounded by ϕ_r. Further, note that the arity of the second part is $2 \cdot |h|$ as shown in 4.

Therefore, by combining both results, we obtain an upper bound on the arity of $max\{2 \cdot |h|, \phi_r\}$. $\square$

We derive by Observation 1 the arity of the complete program $\mathcal{RM}^{\text{count}}(\Pi)$.

Lemma 16 (Grounding time of $\mathcal{RM}^{\text{count}}$)**.** *We take the same assumptions as for the $\mathcal{RS}^{count}$ case (Lemma 5), with the sole difference being that the respective aggregate has only one element. Then by facilitating hybrid grounding, we derive an upper bound of the grounding time of $O((|\Pi_t| + |at(\Pi_t)|) \cdot |\text{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$.*

Proof (Sketch). We combine different results from above. First we know that $\mathcal{RM}^{\text{count}}$ produces $O(1)$ many rules with a known upper bound on the arity of $\phi_{\Pi_t} \leq max\{2{\cdot}|h|, \phi_{\Pi_t}\}$ (Lemma 2). Therefore by applying Lemma 3, we derive $O((|\Pi_t| + |at(\Pi_t)|) \cdot |\text{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$. □

$\mathcal{RM}^{\text{max}}$ for Singletons. When considering the max aggregate, we start our thinking by the rewriting procedure $\mathcal{RS}^{\text{max}}$ and shorten it by the assumptions of a singleton element set and a monotonic aggregate. For this case, we do not need to explicitly state the *body* predicates and additionally do not explicitly need to state the $max_{\prec}$ predicate. The result is the rewriting procedure $\mathcal{RM}^{\text{max}}$ which is depicted in Rules (38).

Rewriting Procedure $\mathcal{RM}^{\text{max}}$

$$p_1(\boldsymbol{X}_1) \leftarrow\ p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), l_1(\boldsymbol{Y}_1), ..., l_k(\boldsymbol{Y}_k), \boldsymbol{T}^1 \geq u. \tag{38a}$$

Lemma 17 (Maximum arity of $\mathcal{RM}^{\text{max}}$). *Let r be a rule containing a single max aggregate, with a singleton element set, with u being an integer constant and $\prec$ being $\geq$. Then, the maximum arity of $\mathcal{RM}^{max}(r)$ is bounded by ϕ_r.*

Proof (Sketch). $\mathcal{RM}^{\text{max}}(r)$ generates a rule which can be split into two parts. The first part contains all predicates of the rule (except the aggregate) and all predicates of the element, whereas the second part contains the $\boldsymbol{T}^1 \geq u$ comparison.

Observe that the arity of the first part is bounded by ϕ_r. Further, note that the arity of the second part is 1, as only 1 variable exists.

Therefore, by combining both results, we obtain an upper bound on the arity of ϕ_r. □

Lemma 18 (Grounding time of $\mathcal{RM}^{\text{max}}$). *We take the same assumptions as for the $\mathcal{RS}^{count}$ case (Lemma 5), with the differences being that the respective aggregate is a max aggregate and has only one element. Then by facilitating the results from hybrid grounding we derive $O((|\Pi_t| + |at(\Pi_t)|) \cdot |\text{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$.*

Proof (Sketch). By combining previous results, we know that $\mathcal{RM}^{\text{max}}$ produces $O(1)$ many rules and has an upper bound on the respective arity of ϕ_{Π_t} (Lemma 17). By this information we can derive through Lemma 3 our upper bound of $O((|\Pi_t| + |at(\Pi_t)|) \cdot |\text{dom}(\Pi)|^{2\cdot\phi_{\Pi_t}})$. □

C.5 Extension of the Rewriting Procedures to arbitrary Comparison Operators

In this section, we generalize issue (iv) of our aggregate rewriting techniques, namely arbitrary comparison operators. Note that in the following, we show the rewriting on the lines of the $\mathcal{RS}$ methods.

We assume a rule $r \in \Pi$ with an aggregate $agg\{E\} \prec u$, for *u* being integer and $\prec \in \{<, \leq, >, \geq, =, \neq\}$. Further, we assume the aggregate to occur in B_r^+, for the sake of explanation. Note that if *u* is a variable (generalization (iii)) one needs to combine the findings of Section C.6 with the findings of this section.

When one thinks about the different rewriting techniques for the different aggregate types for $\mathcal{RS}$, there are multiple recurring ideas. First, the creation of the tuple predicates tp is repeated. Then the creation of an *aggregate predicate*, which differs for each aggregate type, is repeated in an abstract way. And lastly, the original rule is rewritten, s.t. the *aggregate predicate* is added.

To show the details of the extension, first, we need to define the term *standard form* of an aggregate. Intuitively an aggregate is in *standard form*, iff it is monotonic, with $\geq$ or $\leq$, and with a fixed integer bound u. The *standard forms* of all aggregate types are: The count aggregate's standard form is $count\{E\} \geq u$, for the max aggregate it is $max\{E\} \geq u$, for the (positive) sum aggregate (requiring items $\geq$ 0) it is $sum_+\{E\} \geq u$, and lastly for the min aggregate it is $min\{E\} \leq u$.

With the preliminaries behind us, we can present a unifying description of the rewritings, where we denote the *aggregate predicate* as $agg_{\prec}$. The meaning of $agg_{\prec}$ is that it holds iff an aggregate in *standard form* holds. We now transform the aggregate to standard form according to Table 3, for the operators $\prec \in \{<, \leq, >, \geq\}$. Note that we sometimes have to change the bound u by an increment or decrement. We denote this as u'. Further, by adding back our $agg_{\prec}$ predicate we either have to add it to B_r^+, or B_r^-, depending on $\prec$.

Now, we treat the operators $\prec \in \{=, \neq\}$ as two special cases. For both respective translations, we require two *aggregate predicates*. Take, for example, the aggregate $count\{E\} = u$. We transform this aggregate into two standard form aggregates, where the first one is $count\{E\} \geq u$ and the second one is $count\{E\} \geq u + 1$. By this reasoning, we know that whenever the first aggregate holds, and the second one does not, the original aggregate has to hold.

Put generally, for the $count$, max and sum_+ aggregates we generate two aggregate predicates, $agg_{\prec,1}$ (which encodes $count\{E\} \geq u$) and one $agg_{\prec,2}$ (which encodes $count\{E\} \geq u + 1$). For the *min* aggregate we generate we generate two aggregate predicates, $agg_{\prec,1}$ (which encodes $count\{E\} \leq u$) and one $agg_{\prec,2}$ (which encodes $count\{E\} \leq u - 1$). Due to the similarity of the approaches, we can unify them when we take the last step. Therefore, we add $agg_{\prec,1}$ in the positive body B_r^+ and $agg_{\prec,2}$ in the negative body B_r^- of the original rule.

For the $\neq$ operator we take a very similar approach, where we first encode $=$ and then encode the negation of $=$. Therefore, the encoding for $\neq$ is nearly the same as above, i.e., we create two aggregates $agg_{\prec,1}$ and $agg_{\prec,2}$ with the same meaning as above. But, in contrast to $=$, we do not add them to the original rule, but create a *helper-rule*, with the head $agg_{\prec,3}$ and add $agg_{\prec,1}$ to the positive B_r^+ and $agg_{\prec,2}$ to the negative B_r^- body, which encodes $=$. The last step is to add $agg_{\prec,3}$ to the negative body B_r^- of the original rule.

C.6 Extension of the Rewriting Procedures to arbitrary Variable Bounds

In this section we show the rewriting to arbitrary variable bounds (iii). As a representative of all other rewriting methods, we take the $\mathcal{RS}^{\text{count}}$ method to show how this can be done.

	count, *max*, sum_+				*min*			
	$\geq$	$>$	$\leq$	$<$	$\geq$	$>$	$\leq$	$<$
standard form u'	*u*	*u+1*	*u+1*	*u*	*u-1*	*u*	*u*	*u-1*
signum $agg_{\prec}$	$+$	$+$	*not*	*not*	*not*	*not*	$+$	$+$

Table 3: Overview table for converting aggregate expressions $agg\{E\} \prec u$ into the respective aggregate *standard form*, for the comparison operators $\{<, \leq, >, \geq\}$. The *standard form* u' expresses how the u of the aggregate needs to be changed such that it is equivalent to the standard form. The *signum* denotes how the aggregate predicate has to be added to the original rule, either in the positive body B_r^+ (denoted as $+$) or in the negative body B_r^- (denoted as *not*).
The standard forms are: $count\{E\} \geq u'$, $max\{E\} \geq u'$, $sum_+\{E\} \geq u'$, and $min\{E\} \leq u'$.

Assume an aggregate like $count\{E\} \geq Y$ (monotonic), where Y is a bound variable. The principal idea is to catch all possible values for Y, which we can easily do as Y is bound, and therefore, we are able to directly derive its domain, which is denoted as $dom(Y)$. With this information, we are able to generate one (count) aggregate predicate $count_{\prec}$ per value $u \in dom(Y)$ in the domain.

But this comes with a caveat, as at this point $count_{\prec}$ encodes then effectively any $u \in dom(Y)$. To fix this we add the value u as the last term in $count_{\prec}$, i.e., $count_{\prec}(\mathbf{V}', u)$. We depict the full procedure in Equations (39).

Rewriting Procedure $\mathcal{RS}^{\text{count}}$ for variable bounds

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), count_{\prec}(\boldsymbol{V}', Y). \tag{39a}$$

For every integer: $u \in dom(Y)$:

$$count_{\prec}(\boldsymbol{V}', u) \leftarrow tp(\boldsymbol{T}_1', \boldsymbol{V}'), ..., tp(\boldsymbol{T}_u', \boldsymbol{V}'), alldiff(\boldsymbol{T}_1', ..., \boldsymbol{T}_u'). \tag{39b}$$

For every element:

$$tp(\boldsymbol{T}', \boldsymbol{V}') \leftarrow l_1(\boldsymbol{Y}_1), \ldots, l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m). \tag{39c}$$

The procedure above carries over to all aggregate types and comparison operators, as long as Y is bound. But what to do when variable Y is not bound and the comparison operator is the equality operator, therefore the aggregate corresponds to a variable assignment?

Then, a theoretically feasible way to do it is to calculate a feasible upper bound of a possible count value, denoted as Ψ_{cnt}, and then check which count value in the range *0* to Ψ_{cnt} holds. For this calculation let $e \in E$ be an element, let $\mathbf{T}_e$ be the element head vector, $|\mathbf{T}_e|$ the length of the head vector, $dom(\mathbf{T}_e)$ the union over all individual domains of the head vector (total domain), and $|dom(\mathbf{T}_e)|$ be its corresponding number of domain elements. Then Ψ_{cnt} is an upper bound on the result of count:

$$\Psi_{cnt} = \Sigma_{e \in E} |dom(\mathbf{T}_e)|^{|\mathbf{T}_e|} \tag{40}$$

We now develop a full rewriting procedure for this case. To this end, we use our results from Section C.5, where we defined the behavior of an aggregate of the form $agg_{\prec} = u$, by introducing two $agg_{\prec}$, representing the aggregate in standard form. By using this insight, we create for each possible value $u, 0 \leq u \leq \Psi_{cnt}$ a rule that expresses the *count* aggregate in standard form. This gives rise to our full rewriting procedure in Rules (41). Note that in Rule (41) we add a variable Y to the tp.

Rewriting Procedure $\mathcal{RS}^{\text{count}}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), count_{\prec}(\boldsymbol{V}', Y), not\ count_{\prec}(\boldsymbol{V}', Y+1) \tag{41a}$$

For every integer: $u, 0 \leq u \leq \Psi_{cnt}$:

$$count_{\prec}(\boldsymbol{V}', u) \leftarrow tp(\boldsymbol{T}_1', \boldsymbol{V}'), ..., tp(\boldsymbol{T}_u', \boldsymbol{V}'), alldiff(\boldsymbol{T}_1', ..., \boldsymbol{T}_u') \tag{41b}$$

For every element:

$$tp(\boldsymbol{T}', \boldsymbol{V}') \leftarrow l_1(\boldsymbol{Y}_1), \ldots, l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m). \tag{41c}$$

The above methods for the count aggregate can be adopted for *min/max* and *sum* aggregates. For the *sum* aggregate we replace the Ψ_{cnt} by Ψ_{sum}.

$$\Psi_{sum} = \Sigma_{e \in E} \left(max(dom(\mathbf{T}_e)) \cdot |dom(\mathbf{T}_e)|^{|\mathbf{T}_e|} \right) \tag{42}$$

Additionally, one has to replace Equation (41b) with the appropriate formulation for the sum.

For the *min* and *max* aggregates the essential insight is that the domain is not *expanded* by the aggregate, meaning that the overall domain remains unaffected. Therefore, it is not necessary to define an upper bound Ψ. The easier way is to directly define the possible domain Dom_{max}, or Dom_{min} respectively.

In the following, we show how to do it for the *max* aggregate but note that it can be done for the *min* aggregate analogously. Let $\mathbf{T}^1$ be the first term of the element head, then $dom(\mathbf{T}^1)$ is the domain of the first term of the element. With this defined we show the creation of the domain in the following.

$$Dom_{max} = \bigcup_{e \in E} dom(T_e^1) \tag{43}$$

With all of the above defined we are able to replace Equations (41b) by Equations (44) for the *max*-aggregate:

$$\text{For every integer: } u \in Dom_{max} \\ count_{\prec}(\boldsymbol{V}', u) \leftarrow tp(\boldsymbol{T}_1', \boldsymbol{V}'), \boldsymbol{T}_1'^1 \geq u. \tag{44}$$

Due to the nature of the *max* and *min* aggregates there is another highly efficient way to encode such a aggregate.

This approach is inspired by the mathematical definition of the maximum, which states that a value is the maximum if there is no other element that is larger. Through the predicate $not\_largest$ we encode that a tuple tp is not the largest tuple in an aggregate. Therefore, if for a specific tuple tp we can not derive the predicate $not\_largest$, we know that it is the largest. Note that multiple such tuples may exist.

Another take on a rewriting procedure for the max aggregate

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), max_{\prec}(\boldsymbol{V}', Y). \tag{45a}$$

$$max_{\prec}(\boldsymbol{V}', \boldsymbol{T}_1^{'1}) \leftarrow tp(\boldsymbol{T}_1', \boldsymbol{V}'), not\ not\_largest(tp(\boldsymbol{T}_1', \boldsymbol{V}')). \tag{45b}$$

$$not\_largest(tp(\boldsymbol{T}_1', \boldsymbol{V}')) \leftarrow tp(\boldsymbol{T}_1', \boldsymbol{V}), tp(\boldsymbol{T}_2', \boldsymbol{V}), \boldsymbol{T}_1^{'1} < \boldsymbol{T}_2^{'1}. \tag{45c}$$

For every element:

$$tp(\boldsymbol{T}', \boldsymbol{V}') \leftarrow l_1(\boldsymbol{Y}_1), \ldots, l_k(\boldsymbol{Y}_k), p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m). \tag{45d}$$

C.7 $\mathcal{RA}$- Symphony of Hybrid Grounding and Aggregates

So far, we focused on translating aggregates by completely removing the aggregates. In this approach, we use an intermediate form, where we ground single dense elements with hybrid grounding, while leaving the rest of the aggregate untouched.

We refer to the approach presented here as $\mathcal{RA}$, and we treat directly any function $aggr$ and relation $\prec$. As above, we generate auxiliary predicates, but additionally, we directly replace element bodies $l_1(\boldsymbol{Y}_1), ..., l_k(\boldsymbol{Y}_k)$ in the aggregate expression with the respective predicate $tp_j(\boldsymbol{T}_j, \boldsymbol{V}_j)$. This is depicted in Rule (46a). In order to derive $tp_j(\boldsymbol{T}_j, \boldsymbol{V}_j)$ for each element e_j by Rules (46b), we assume precomputed element head lengths $\boldsymbol{T}_j$ and variable dependencies $\boldsymbol{V}_j$.

Rewriting Procedure $\mathcal{RA}$

$$p_1(\boldsymbol{X}_1) \leftarrow p_2(\boldsymbol{X}_2), ..., p_m(\boldsymbol{X}_m), \neg p_{m+1}(\boldsymbol{X}_{m+1}), ..., \neg p_n(\boldsymbol{X}_n), aggr\{\boldsymbol{T}_1 : tp_1(\boldsymbol{T}_1, \boldsymbol{V}_1); \ldots; \boldsymbol{T}_\alpha : tp_\alpha(\boldsymbol{T}_\alpha, \boldsymbol{V}_\alpha)\} \prec u. \tag{46a}$$

For some element(s) e_j:

$$tp_j(\boldsymbol{T}_j, \boldsymbol{V}_j) \leftarrow l_1(\boldsymbol{Y}_1), ..., l_k(\boldsymbol{Y}_k). \tag{46b}$$

This rewriting still enables the utilization of hybrid grounding and Theorem 1. Thereby, Rules (46b) comprise Π_t, which allows us to deal with dense aggregate bodies and keep the actual aggregation, namely Rule (46a) in Π_c. When comparing Rule (46a) with Rule (26a), observe that $\mathcal{RA}$ shifts effort from hybrid grounding to traditional grounding.

Example 10. *Consider the following rule r:*

$$t(Z) \leftarrow a(X,Y), sum\{A, B : a(A,B), b(A,X); A : p(A), a(A,Y); A : q(A)\} = Z.$$

The result of $\mathcal{RA}(r)$ (for all elements) is:

$$t(Z) \leftarrow a(X,Y), sum\{A, B : tp_1(A,B,X); A : tp_2(A,Y); A : tp_3(A)\} = Z.$$

$$tp_1(A,B,X) \leftarrow a(A,B), b(A,X).tp_2(A,Y) \leftarrow p(A), a(A,Y).$$

$$tp_3(A) \leftarrow q(A).$$

In the following, we derive the grounding time results for the $\mathcal{RA}$ rewriting technique when grounded with the hybrid grounding technique. First, we state the maximum arity and then the grounding time.

Lemma 19 (Maximum arity of the $\mathcal{RA}(r)$ rewriting technique)**.** *Let r be a rule containing a single aggregate with α elements. Then, the maximum arity of $\mathcal{RA}(r)$ is bounded by $max\{|h'| + |d'|, \phi_r\}$.*

Proof (Sketch). The tp_j predicates have an arity that depends on their respective tuple variable vector $\boldsymbol{T}$ and on their variable dependencies vector $\boldsymbol{V}$. The maximum arity of all *tp* predicates is denoted as $|max_{tp_j}| = |h'| + |d'|$. In Rules (46b) the maximum arity is bounded by $max\{|h'| + |d'|, \phi_r\}$, as it could happen that the predicate with ϕ_r occurs there, and further, that the *tp* with arity $|h'| + |d'|$ exists there. For Rule (46a) it holds that it could happen that the predicate with arity ϕ_r exists there, and it is certain that the *tp* predicate with arity $|h'| + |d'|$ resides there. Therefore $max\{|h'| + |d'|, \phi_r\}$ is an upper bound on the arity of Rule (46a).

By combining the results for the Rules (46b) and (46a), we get an upper bound on the arity of $max\{|h'| + |d'|, \phi_r\}$. □

Finally, we come to the grounding time result for $\mathcal{RA}$ when grounded by hybrid grounding.

Lemma 20 (Grounding time of $\mathcal{RA}$)**.** *Let Π be an HCF program with a rule using an aggregate over α elements. Further, let $\Pi_t \subseteq \mathcal{RA}(\Pi)$ and ϕ_{Π_t} be the largest predicate arity of Π_t. Then, the runtime of $\mathcal{H}(\emptyset, \Pi_t)$ is in $\mathcal{O}((|\Pi_t| + |at(\Pi_t)| + \alpha) \cdot |\mathrm{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$.*

Proof (Sketch). As for the other grounding time proofs, we again use our results thus far. We use hybrid grounding for $\mathcal{H}(\emptyset, \Pi_t)$, and we want to apply $\mathcal{RA}$ to Lemma 3. We note that rule 46a cannot be in Π_t, by how we apply the hybrid grounding technique as specified in Section 4.

We know that $\mathcal{RA}$ produces $O(\alpha)$ many rules. So we apply Lemma 3 and receive our upper bound of $\mathcal{H}(\emptyset, \Pi_t)$ is in $\mathcal{O}((|\Pi_t| + |at(\Pi_t)| + \alpha) \cdot |\mathrm{dom}(\Pi)|^{2 \cdot \phi_{\Pi_t}})$. □